Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»

Виконав: студент II курсу
групи 3ПІ-22м спеціальності
121 – Інженерія програмного забезпечення
Богомазов Данило Вікторович

Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
«15» грудня 2023 р.

Опонент: к.т.н., доцент каф. ОТ Богомолов С.В.
«15» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«15» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« 19 » вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Богомазову Данилу Вікторовичу

1. Тема роботи – Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія».

Керівник роботи: Катєльніков Денис Іванович, д.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від
« 18 » вересня 2023 р. № 247.

2. Строк подання студентом роботи
5 грудня 2023 р.

3. Вихідні дані до роботи: офіційні правила гри «Монополія»; кількість гравців від двох до шести; розмір графічного вікна 1920 x 1080 пікселів; ігровий рушій Unity; середовище розробки Microsoft Visual Studio 2019; мова програмування C#.

4. Зміст текстової частини: розглядання розвитку гри «Монополія»; формулювання проблеми і способу вирішення; розглядання розвитку штучного інтелекту; обґрунтування вибору інтерфейсу; опис інтерфейсу ігрового додатку; розробка діаграм та блок-схем; розробка штучного інтелекту різних рівнів складності; обґрунтування вибору мови програмування, середовища розробки, ігрового рушія; створення детальної інструкції користувача; проведення тестування ігрового додатку.

5. Перелік ілюстративного матеріалу: мета, об'єкт та предмет дослідження; наукова новизна; постановка задач розробки; інтсрументи для розробки ігрового додатку; структура ігрового додатку; головне меню додатку; меню вибору гравців;

основна сцена гри; розробка штучного інтелекту; графічне предста
штучного інтелекту; тестування ігрового додатку; публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д.І., д.т.н., доцент кафедри ПЗ	19.09.2023	05.12.2023
5	Причепа І. В. к.е.н., доц. каф. ЕПВМ	17.11.2023	25.11.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	При
1	Аналіз вибору методу розробки та постановка задачі дослідження	20.09. 2023 - 01.10.2023	
2	Розробка архітектури та алгоритмів програмного продукту	02.10.2023 - 16.10.2023	
3	Розробка функціоналу та модулів програми	17.10.2023 - 11.11.2023	
4	Тестування програми	12.11.2023 - 20.11.2023	
5.	Економічна частина	21.11.2023 - 8.12.2023	

Студент _____ **Богомаз**
(підпис) (прізвище та

Керівник магістерської кваліфікаційної роботи _____ **Катєльн**
(підпис) (прізвище та

АНОТАЦІЯ

УДК 004.932

Богомазов Д.В. Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія». Магістерська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. - 98 с.

На укр. мові. Бібліогр.: 31 назва; рис.: 18; табл.: 14.

Метою даної магістерської роботи є розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія», який здатен конкурувати з гравцями та надавати їм високий рівень виклику.

У роботі проведено детальний аналіз аналогів з використанням модуля штучного інтелекту для гри «Монополія». Здійснено подальший розвиток вказаного модуля, базуючись на застосуванні високоефективних стратегій гри, вдосконаленні алгоритмів прийняття рішень та оптимізації роботи гри. Продемонстровано, що підвищення рівня інтелектуальної дії гри можливе через інтеграцію розширених методів машинного навчання та адаптації до різноманітних гравецьких стратегій, забезпечуючи високий рівень геймплейної складності та задоволення для користувачів.

Для створення програмного додатку був використаний інструмент для розробки відеоігор Unity та середовище розробки Microsoft Visual Studio 2019. Додаток написаний на мові програмування C#.

Ключові слова: Unity, Visual Studio 2019, C#, «Монополія», ігровий додаток, блок-схема, UML-діаграма, юніт-тести, графічний інтерфейс.

ABSTRACT

Bohomazov D.V. Development of a Method and Software for Artificial Intelligence Module for the Game "Monopoly". Master's Qualification Work in the field of 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2023. - 98 p.

In Ukrainian. Bibliography: 31 titles; figures: 18; tables: 14.

The aim of this master's thesis is to develop a method and software for the artificial intelligence module for the game "Monopoly" capable of competing with players and providing them with a high level of challenge. The paper provides a detailed analysis of analogs using the artificial intelligence module for the game "Monopoly". Further development of the specified module has been carried out, based on the application of highly effective game strategies, improvement of decision-making algorithms, and optimization of game performance. It is demonstrated that an increase in the level of game intelligence is possible through the integration of advanced machine learning methods and adaptation to various player strategies, ensuring a high level of gameplay complexity and satisfaction for users.

To create the software application, the Unity game development tool and Microsoft Visual Studio 2019 development environment were used. The application is written in the C# programming language.

Keywords: Unity, Visual Studio 2019, C#, "Monopoly," game application, flowchart, UML diagram, unit tests, graphical interface.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний світ характеризується стрімким розвитком інформаційних технологій та штучного інтелекту, які змінюють спосіб функціонування різних сфер нашого життя, включаючи ігрову індустрію. Гри, зокрема, стають все складнішими та більш реалістичними завдяки впровадженню штучного інтелекту (ШІ). Однак досі прориви в ШІ були, в основному, в сфері генеративного ШІ, який породжував текст без аналізу самої ігрової ситуації. В той же час питання розробки методів побудови ШІ, який буде при виборі рішень в значній мірі покладатись саме на аналіз ігрової ситуації, залишається відкритим. Це і обумовлює актуальність розробки методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія», яка представляє собою одну з найвідоміших настольних ігор і, одночасно, прекрасний полігон для випробування розроблених користувачем модулів ШІ.

Вибір даної теми обумовлений кількома важливими чинниками. По-перше, гра «Монополія» є однією з найпопулярніших настільних ігор у світі та має складний набір правил та стратегій, що вимагає інтелектуальних навичок для успішної гри. По-друге, розробка модуля штучного інтелекту для цієї гри відкриває можливості для дослідження та вдосконалення алгоритмів прийняття рішень, використаних у ШІ, що може бути корисним для інших сфер, таких як робототехніка, фінанси тощо. По-третє, розробка ШІ для гри «Монополія» сприяє подальшому вдосконаленню ігрової індустрії, дозволяючи створити більш інтелектуальних та конкурентоздатних віртуальних противників.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою даної магістерської роботи є підвищення інтерактивності грального процесу ігрового програмного додатку «Монополія» за рахунок реалізації штучного інтелекту -гравця, який здатен конкурувати з гравцями та надавати їм високий рівень виклику.

Основними задачами дослідження є:

- провести аналіз правил та стратегій гри «Монополія»;
- розробити алгоритми прийняття рішень для ШІ на основі аналізу існуючих стратегій та методів;
- розробити алгоритми прийняття рішень в ключових ситуаціях на основі дерева ухвалення рішень та правил ЯКЩО-ТОДІ;
- реалізувати програмний модуль штучного інтелекту для гри «Монополія»;
- розробити користувацький інтерфейс;
- розробити програмний продукт;
- провести тестування.

Об’єкт дослідження: процес розробки покрокових стратегічних економічних ігрових додатків «Монополія»;

Предмет дослідження: методи та засоби розробки модулів штучного інтелекту для покрокових стратегічних економічних ігрових додатків «Монополія»;

Методи дослідження. У процесі досліджень використовувались: теорія рішень, методи оптимізації, теорія графів для розробки моделі ШІ, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод побудови модуля штучного інтелекту на основі дерева ухвалення рішень, у якому, на відміну від існуючих методів, у вузлах використовуються не жорсткий набір умов, а розширений набір продуктивних правил ЯКЩО-ТОДІ, що дозволило не тільки зробити процес прийняття рішень більш прозорим, але й надало можливість додавати нові правила по мірі накопичення досвіду.

2. Уперше запропоновано систему використання методу штучного інтелекту для ідентифікації та адаптації до індивідуальних стилів гри гравців у «Монополії». Розроблений підхід використовує аналіз дій гравців та автоматизоване визначення їхніх стратегій для покращення взаємодії штучного інтелекту з гравцями.

Практична цінність отриманих результатів.

Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби модуля штучного інтелекту для гри в покрокову стратегічну економічну гру «Монополія».

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: розробка ігрового застосунку з елементами штучного інтелекту з використанням технології Unity та мови C# [1]; розробка модуля мережевого обміну для ігрового застосунку з елементами штучного інтелекту з використанням технології Unity та мови C# [2]; розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»[3]; розробка модуля штучного інтелекту для гри "Монополія"[4].

Апробація матеріалів дисертації. Основні положення роботи доповідалися та обговорювалися на Міжнародних і Всеукраїнських конференціях: «LI Науково-технічна конференція підрозділів Вінницького національного технічного університету» (Вінниця, 2022); Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації - 2022» (Одеса, 2022); Всеукраїнська науково-технічна конференція молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації - 2023» (Одеса, 2023); Збірник матеріалів Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Суми/Вінниця, 2023).

Публікації. Основні результати досліджень опубліковано в 4 наукових працях: 4 – у матеріалах конференцій.

1 АНАЛІЗ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Розвиток настільної гри «Монополія»

Найперша версія гри Monopoly була створена ще на початку 20 сторіччя в 1903 році, тоді ця гра мала назву «The Landlord's Game». Цю гру запатентувала американська антимонополістка Ліззі Мегі з метою пояснити теорію єдиного податку Генрі Джорджа. Вона знала, що деяким людям буде важко зрозуміти логіку цієї теорії, тому вирішила подати її у вигляді гри. Зображення найпершої версії гри можна побачити на рисунку 1.1 [5].

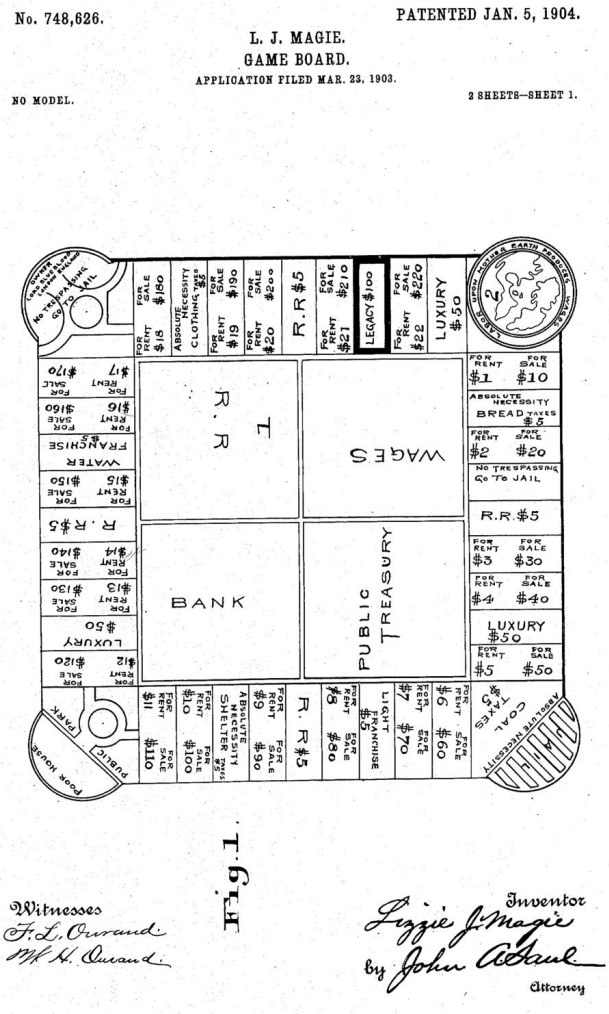


Рисунок 1.1 – Зображення найпершої версії гри

У 1923 році Ліззі Мегі переїхала до Вашингтона, округ Колумбія, зі своїм чоловіком і повторно запатентувала переглянуту версію «The Landlord's Game» в 1924 році (під своїм заміжнім іменем Елізабет Мегі Філліпс). Ця версія, на відміну від її першого патентного малюнка, містила назви вулиць. Нова версія гри зображена на рисунку 1.2.

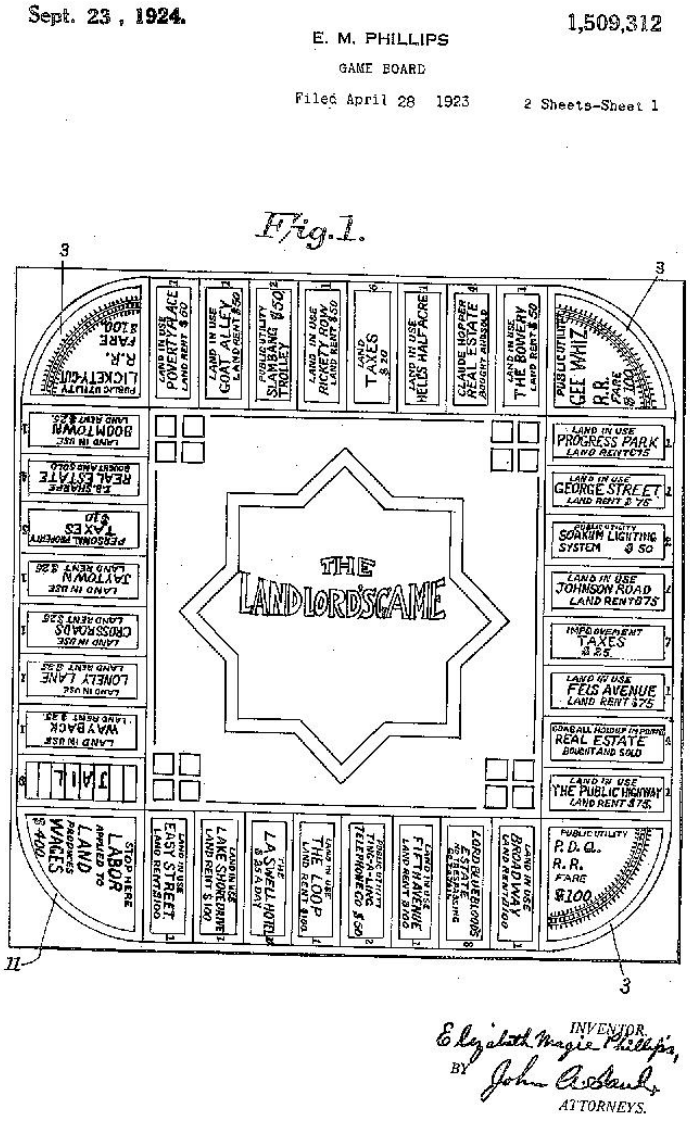


Рисунок 1.2 – Зображення другої версії гри

У 1932 році Чарльз Дерроу вдохновився грою «The Landlord's Game» і вирішив розробити власну під назвою «Monopoly». Саме ним був розроблений дизайн гри, який став широковідомим по всьому світі.

У 1935 році компанія з виробництва настільних ігор Parker Brothers викупили патент на гру «Monopoly». Залишивши дизайн, який розробив Чарльз Дарроу та підкорегувавши деякі правила гра вперше з’явилася у широкому продажі. На рисунку 1.3 зображений дизайн, який розробив Чарльз Дарроу.

З того часу гра стала поширюватися світом з неабиякою швидкістю. Відповідно до Hasbro з моменту випуску гри у 1935 році в неї зіграло понад 750 мільйонів людей, що зробило її найпопулярнішою комерційною гру у світі.

Гра навіть потрапила у Книгу Рекордів Гінеса в 1999 році, оскільки на той момент в неї зіграло вже понад 500 мільйонів людей [6].

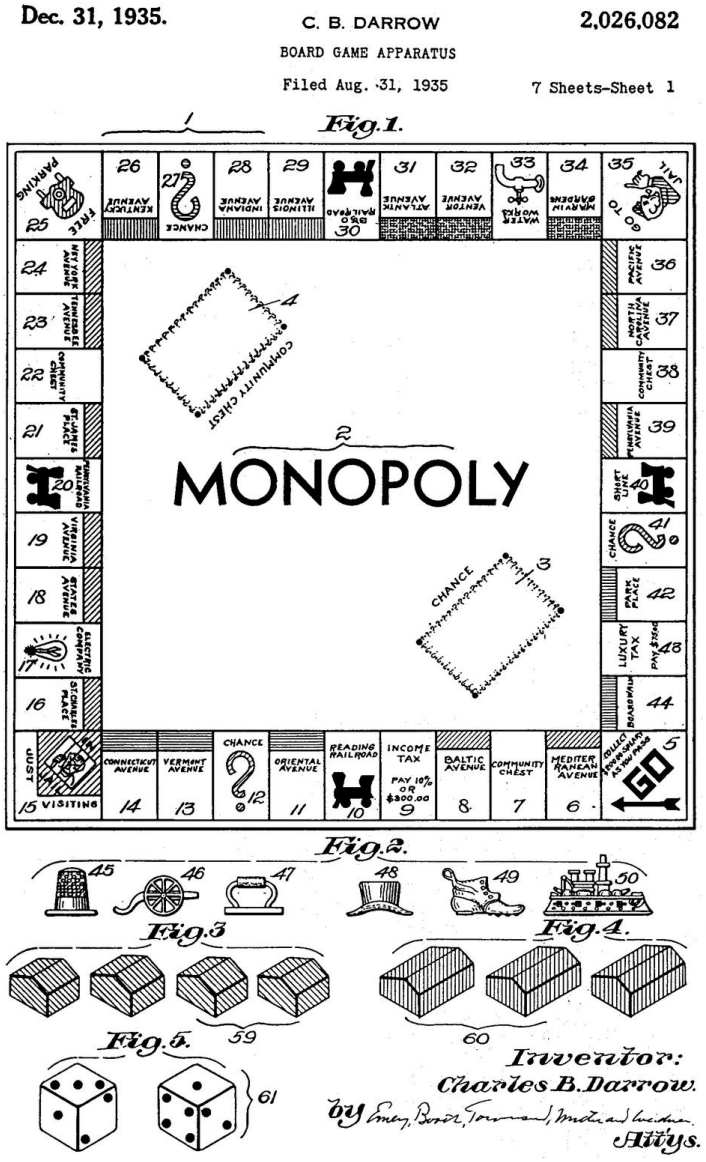


Рисунок 1.3 – Зображення версії гри, яка з’явилася в широкому продажі

1.2 Аналіз стану проблеми

Проблема класичної гри «Monopoly» є в тому, що вона дуже випадкова і від дій гравця, по суті, мало що залежить. Це все виникає через те, що у гравців дуже не велика кількість дій, які вони можуть зробити. І справді, гравець може лише: купити, поторгувати. Це дуже не великий спектр дій враховуючи те, що гравець не може обирати куди саме він потрапить на цьому ходу.

Ще однією проблемою, на мій погляд, є не дуже якісний дизайн поля, адже деякі поля (наприклад в'язниця або паркова) не роблять взагалі нічого і якщо брати на увазі минулий мінус гри, гравець може втратити хід майже ні на що.

І ще один мінус, який вже відноситься до цифрових аналогів, адже в настільній версії змінити це майже неможливо, це необхідна мінімальна кількість людей для того, щоб грати.

У розроблюваному додатку будуть змінені класичні правила гри на більш гнучкі, окрім того буде додано багато різноманітних варіантів дій, які гравець може зробити під час свого ходу. Крім того буде змінений дизайн, щоб майже всі поля мали сенс. І наостанок буде доданий штучний інтелект у вигляді ботів, з якими гравець навіть не маючи людей, з якими можна було б зараз пограти, зможе почати гру.

Штучний інтелект дозволяє автоматизувати циклічні процеси вивчення та пошуку найліпших шляхів вирішення проблеми. Також ця технологія робить будь-який продукт «інтелектуальним». В цілому у сучасних реаліях можна зустріти штучний інтелект будь-де, наприклад: автопілот в автомобілях «Тесла», роботи-кур'єри з доставкою їжі, камери на дорогах, які фіксують правопорушення та відправляють звіт в той же момент в державні структури, розпізнавання лиць в будь-якому сучасному смартфоні та, не менш важливо, в ігровій індустрії.

Штучний інтелект зародився в 1960 році, але великої популярності не здобув, адже реалізацій було мало і вони були слабі, через такі проблеми, як: надмірна вартість машинного часу, вельми скромні обчислювальні ресурси, обмеженість мов програмування, громіздкість елементної бази тощо [7].

З часом технології поступово вдосконалювалися, і можливість

впровадження штучного інтелекту ставала все доступнішою щороку. Сьогодні неможливо уявити собі життя без штучного інтелекту, оскільки він практично впроваджений в усі аспекти нашого життя.

Ігрова індустрія постійно розвивалася. Для забезпечення захоплюючого геймплею гравцю потрібна можливість взаємодії. Перші відеоігри, такі як Spacewar!, Pong і Gotcha, були призначені для гри гравця проти гравця, але іноді було важко знайти співгравця. Тому був розроблений ігровий штучний інтелект, який дозволяє гравцю грати проти комп'ютера або разом із ним.

Сучасні ігри майже завжди використовують штучний інтелект, і він впливає на цікавість гри. Штучний інтелект може бути занадто важким для перемоги або, навпаки, занадто простим. Створення відповідного штучного інтелекту - складна задача.

Далі розглянемо основні методи знаходження нових знань на основі даних. Інтелектуальні інформаційні технології спрямовані на виявлення нових знань для покращення результатів діяльності користувача. Один з результатів моделювання - це виявлення відношень у даних.

Існують шість методів виявлення й аналізу знань:

- Класифікація;
- Регресія;
- Прогнозування тимчасових послідовностей (рядів);
- Кластеризація;
- Послідовність.

«Послідовність» застосовується в програмному додатку [8].

1.3 Порівняльний аналіз аналогів

Оскільки гра «Monopoly» є однією з найвідоміших настільних ігор у світі, не дивно що було розроблено чимало ігор, які були надихнуті нею. Окрім настільних ігор, з поширенням комп’ютерів почали з’являтися і цифрові аналоги гри. Деякі з них повністю переносять гру зі всіма правилами у цифровий вигляд, а деякі намагаються зробити щось дуже схоже, щоб не втрачати суть гри, але

додаючи свої особливості.

Monopoly Silvergames – браузерна гра, яка повністю перенесла весь дизайн і правила гри з настільного варіанту у цифровий. Оскільки гра є браузерною і безкоштовною, то це є великим плюсом, адже в неї зможуть зіграти всі, хто має інтернет. До мінусів даного аналога можна додати необхідність мати доступ до інтернету та повна адаптація настільної версії гри без додавання якихось цікавих змін у правилах або дизайну [9].

Зображення інтерфейсу додатку «Monopoly Silvergames» зображено на рисунку 1.4.

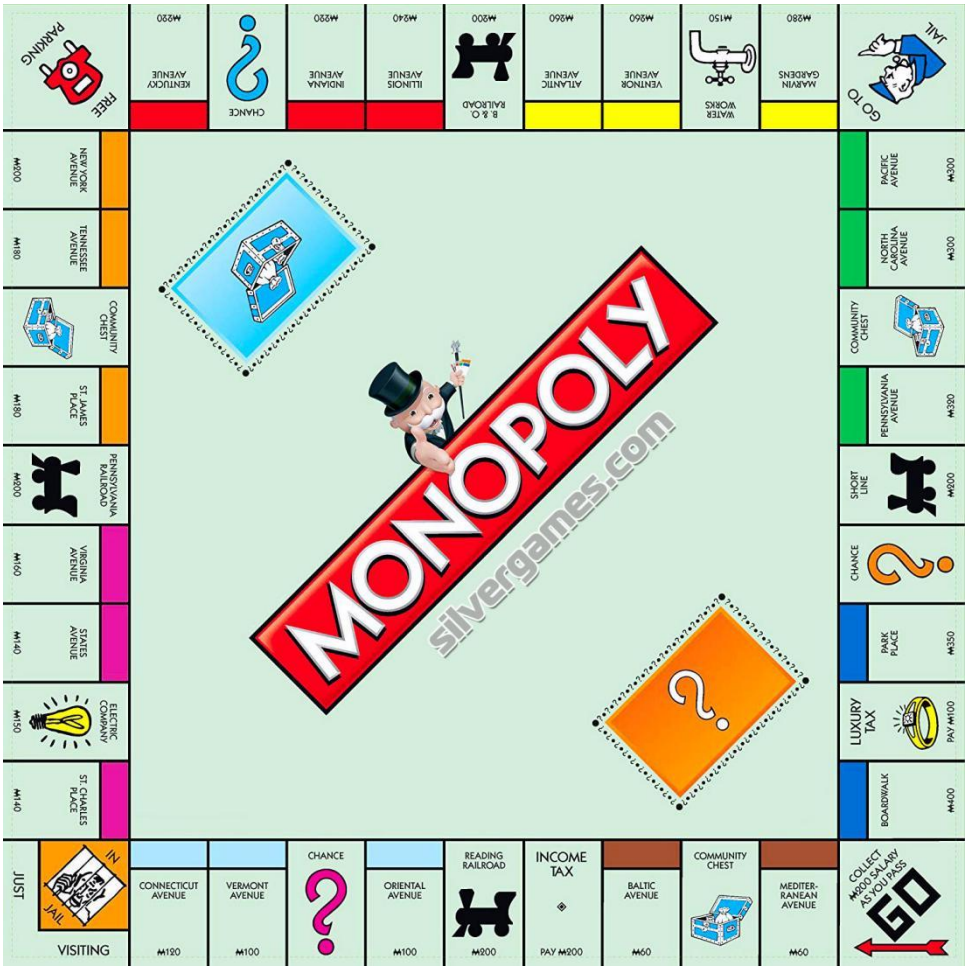


Рисунок 1.4 – Зображення головного екрану додатку «Monopoly Silvergames»

Capitalista – ще одна браузерна настільна гра, але це вже не просто перенесення настільної версії у цифровий вигляд, а додавання чогось нового. В цій грі був змінений дизайн, додані 3D моделі. Також є можливість спілкуватися з людьми у вбудованому чаті. З мінусів можна виділити не дуже вдалий дизайн, адже для настільних ігор вид з боку має досить не звичний вигляд і деякі 3д моделі, хоч і мають прозорість, але все ж таки закривають частини ігрового поля, що погано впливає на ігровий процес. Також можна відмітити, що було змінено лише дизайн оригінальної гри, а правила залишились без змін [10].

Інтерфейс гри «Capitalista» зображений на рисунку 1.5.

Рисунок 1.5 – Зображення інтерфейсу головного екрану додатку «Capitalista»

Monopoly star. На відміну від попередніх аналогів monopoly star змінила не лише дизайн, а й правила гри. Це вже відрізняється від класичної монополії, але все ж таки, випадковість все ще зависока і від гравця залежить не так багато, як хотілося б. Тут додали нову для гри механіку, а саме взяти кредит, в розроблюваному ігровому додатку також буде така можливість, але, на мою думку цих змін замало. Окрім цього, у цій грі немає підтримки штучного

інтелекту, а значить вона також потребує більше одного «живого» гравця, що є мінусом [11].

Інтерфейс гри «Monopoly star» зображений на рисунку 1.6.

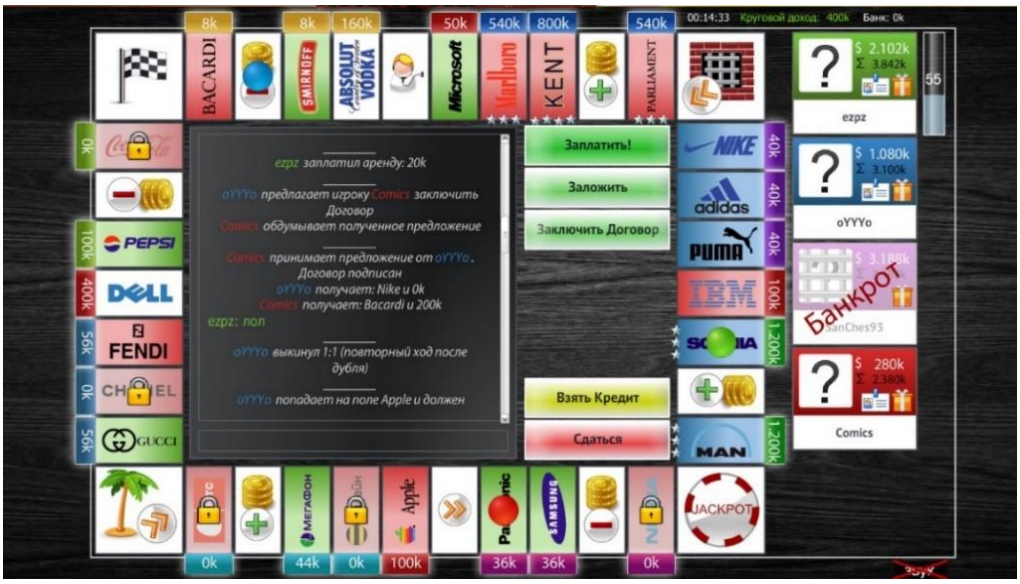


Рисунок 1.6 – Зображення інтерфейсу додатку «Monopoly star»

Отже, розглянувши аналоги було визначено їх плюси та мінуси і порівняно з розробленим додатком «Монополія».

1.4 Розвиток штучного інтелекту

Розвиток штучного інтелекту може бути умовно розділений на три етапи.

Етап перший, який припадає на кінець 50-х років XX століття, відзначався початком перших досліджень у галузі штучного інтелекту. У 1956 році з'явилася перша програма зі штучним інтелектом під назвою «Логік-Теоретик», яка була розроблена для доведення теорем в численних висловах. Також у 1957 році була створена програма для гри в шахи, що в подальшому привела до концепції універсального вирішення завдань. На цей час існували два основних методи розв'язання задач: евристичний метод, який використовував припущення та перевірку, і алгоритмічний метод, який передбачав механічне виконання послідовності кроків для отримання правильної відповіді. У 1965 році Дж. Робінсон розробив метод резолюцій, який базувався на доведенні теорем у логіці

предикатів шляхом створення суперечностей.

На першому етапі розвитку методи штучного інтелекту випробовувалися в іграх, головоломках та математичних задачах через їхню простоту та зрозумілість. Такий вибір обумовлювався можливістю легкого адаптування методів до цих завдань. Спад популярності цих досліджень припав на кінець 60-х років XX століття, коли вже робилися спроби використовувати розроблені методи для реальних задач, а не лише в штучних середовищах.

Другий етап, який припадає на початок 70-тих років XX століття, відзначився кардинальним розширенням досліджень у цій галузі. Вчені зрозуміли, що раніше створеним програмам не вистачає глибоких знань в конкретних природних областях, і для цього потрібно використовувати методи логічного мислення та накопичені знання в символьній формі. Однак виникла проблема, як передати ці знання програмі, якщо їхній розробник не володіє ними. Виникла необхідність забезпечити системи штучного інтелекту можливостями, яких не було в звичних мовах програмування, щоб програма сама могла виділяти знання з експертної інформації. До 1970 року було розроблено безліч програм на основі цих ідей, такі як програма DENDRAL, призначена для створення структурних хімічних формул на основі даних, отриманих від мас-спектрометра.

Третій етап розпочався в середині 70-х років XX століття і відзначався зсувом акценту від створення автономних систем до створення людино-машинних систем, які поєднували інтелект людини з обчислювальною здатністю машини для розв'язання загальних завдань [12].

Можна виділити три основні підходи в моделюванні штучного інтелекту.

Перший підхід. У цьому напрямку дослідженням підлягає структура та механізми функціонування людського мозку, а основною метою є розкриття таємниць людського мислення. На цьому шляху важливими кроками є побудова моделей на основі психофізіологічних даних, проведення експериментів з цими моделями, формулювання нових гіпотез щодо механізмів інтелектуальної діяльності та постійне вдосконалення цих моделей.

Другий підхід. В цьому напрямку дослідженням піддається штучний

інтелект. Цей підхід передбачає моделювання інтелектуальної діяльності за допомогою обчислювальних машин. Основною метою робіт в цьому напрямку є створення алгоритмічного та програмного забезпечення для комп'ютерів, яке дозволить їм розв'язувати інтелектуальні завдання на рівні або навіть краще за людей.

Третій підхід. Цей напрямок орієнтований на створення інтерактивних інтелектуальних систем, які поєднують можливості природного і штучного інтелекту. Головним завданням в цьому напрямку є оптимальний розподіл функцій між природним і штучним інтелектом і організація діалогу між людиною і машиною [13].

У розроблюваному ігровому додатку вибрано другий підхід, оскільки завдання полягає у розробці бота, який буде здатен точно імітувати дії людини та ефективно вирішувати ігрові завдання, не поступаючись гравцям-людям. Для досягнення цієї мети було обрано метод розв'язання, що базується на алгоритмічних підходах. Боту буде передано конкретну послідовність кроків, яку він виконає під час свого ходу.

1.5 Постановка задачі розробки

Розглядаючи завдання розробки програмного додатку «Монополія» з включенням елементів штучного інтелекту, визначено наступні завдання розробки:

- Розробка графічного інтерфейсу: Розробка імпресивного та зручного для користувачів інтерфейсу гри «Монополія», який включає в себе графіку, анімацію та інші візуальні компоненти.
- Логіка гри: Реалізація правил гри «Монополія», включаючи класичні правила гри, такі як купівля, продаж, обмін, окрім того будуть додані додаткові правила гри, які будуть робити гру більш цікавою та менш залежної від випадковостей.
- Штучний інтелект: Розробка алгоритмів та стратегій для гри в «Монополію» для комп'ютерних супротивників. Боти повинні бути здатні робити

інформовані рішення, враховуючи стан гри та дії гравців.

- Оптимізація та тестування: Вдосконалення та оптимізація програмного коду для забезпечення ефективної роботи додатку. Проведення тестувань для виявлення та виправлення помилок та недоліків.
- Документація: Підготовка документації для користувачів та розробників, яка описує правила гри, функціональність додатку та інші важливі аспекти.
- Забезпечення безпеки: Захист гри від можливих атак та недозволених втручань, а також забезпечення конфіденційності даних гравців.
- Підтримка користувачів: Надання технічної підтримки та виправлення можливих проблем, які виникають у користувачів гри.

Ці завдання є ключовими етапами розробки програмного додатку «Монополія» з елементами штучного інтелекту, які необхідно вирішити для успішного втілення проекту.

1.6 Висновки

У першому розділі було розглянуто розвиток настільної гри Монополія. Продемонстрована еволюція і розвиток гри. Були виділені основні проблеми класичної Монополії і способи їх вирішення.

Розглянуто декілька існуючих аналогів, проаналізовано їх плюси і мінуси відносно розроблюваного додатку.

Окрім цього, був описаний розвиток штучного інтелекту і виділено основіні методи і підходи розробки додатків з використовуванням штучного інтелекту.

В кінці розділу було сформовано постановку задачі розробки.

2 РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Розробка діаграм

Уніфікована мова моделювання, або UML, є мовою, яка використовується для моделювання та візуалізації системних структур та процесів. Однією з ключових особливостей UML є можливість використання різних діаграм для подання різних аспектів системи [14].

UML діаграми дозволяють інженерам та розробникам отримати глибоке розуміння системи та її компонентів. Вони стають важливим інструментом при розробці програмного забезпечення, допомагаючи командам взаємодіяти, розуміти та узгоджувати концепції та рішення.

Однією з основних переваг UML є її здатність до представлення як структурних елементів системи (таких як класи, об'єкти, компоненти), так і динамічних взаємодій між цими елементами (за допомогою діаграм взаємодій та станів).

UML діаграми використовуються на різних етапах розробки програмного забезпечення, від концептуального проектування до реалізації та тестування. Це дозволяє розробникам створювати чіткі та узгоджені моделі, сприяючи виконанню завдань ефективно та враховуючи всі необхідні аспекти системи.

Однією з ключових переваг UML є те, що вона служить як спільна мова комунікації між усіма учасниками проекту. Розробники, архітектори, тестувальники та замовники можуть використовувати UML для ефективного обміну ідеями, розуміння вимог та вирішення різноманітних завдань.

UML діаграми стали стандартом у світі розробки програмного забезпечення. Вони допомагають створювати чіткі та консистентні моделі, полегшуючи роботу команд та забезпечуючи високий рівень розуміння складних систем. Завдяки своїй універсальності та ефективності, UML продовжує залишатися ключовим інструментом у сфері розробки програмного забезпечення.

Різновиди UML діаграм зображені на рисунку 2.1.

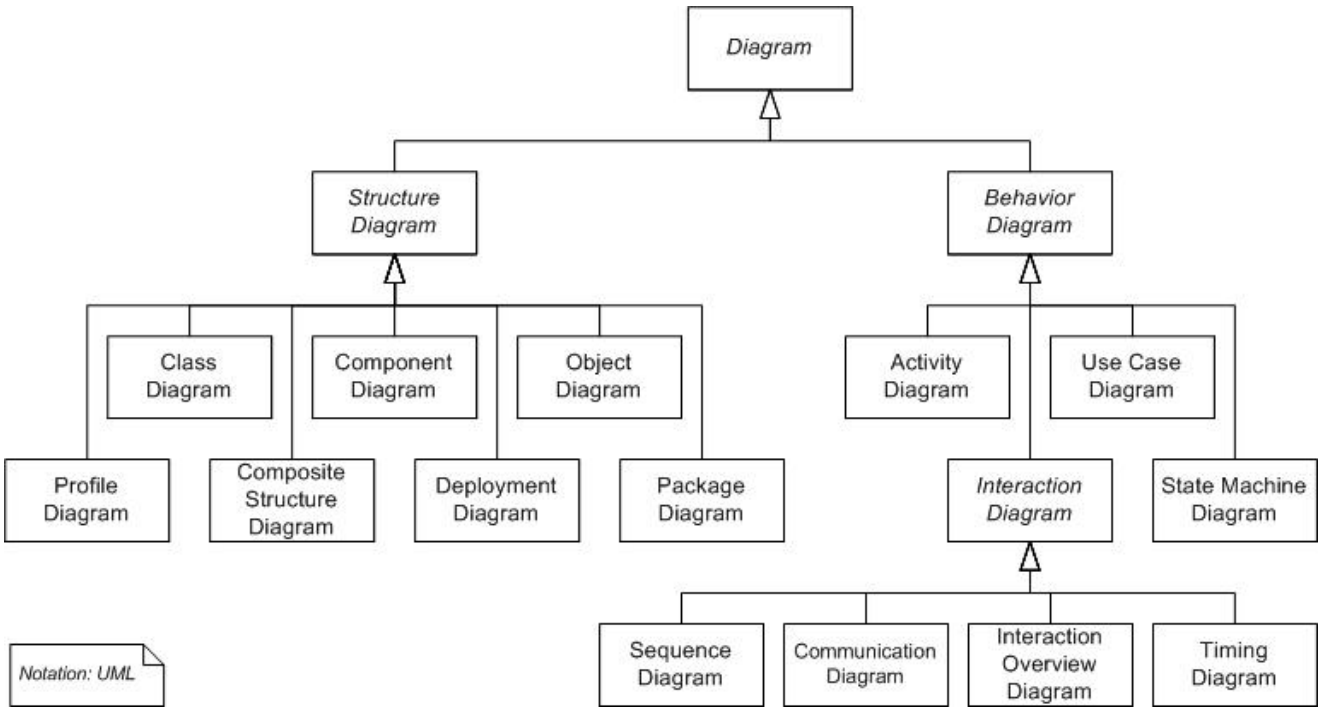


Рисунок 2.1 – Класифікація UML діаграм

Для ігрового додатку буде використана діаграма прецедентів для опису можливостей користувача та діаграма послідовності для опису процесу ходу гри.

Діаграма прецедентів зображена на рисунку 2.2. На діаграмі можна побачити, що гравець може перш за все обрати параметри гри, тоді він отримає можливість обирати такі речі, як кількість початкових грошей, складність гри (штучного інтелекту) та кількість і параметри гравців. Окрім цього гравець може почати гру, тоді він отримує можливість взаємодіяти з секторами, іншими гравцями, ботами та центральним меню. Всі дії які виконує гравець потрібно обробити. Обробкою даних займається система.

Діаграма послідовностей зображена на рисунку 2.3. За допомогою цієї діаграми можна зрозуміти, як здійснюється хід гравця. Спочатку він кидає кубики за допомогою правої панелі, після цього інформація про нове місце розташування надходить до центральної панелі, яка в свою чергу змінює місце гравця. Після чого гравець може взаємодіяти з меню сектора та центральним меню. В кінці гравець завершає хід за допомогою правої панелі, після чого змінюється профілі гравців на лівій панелі.

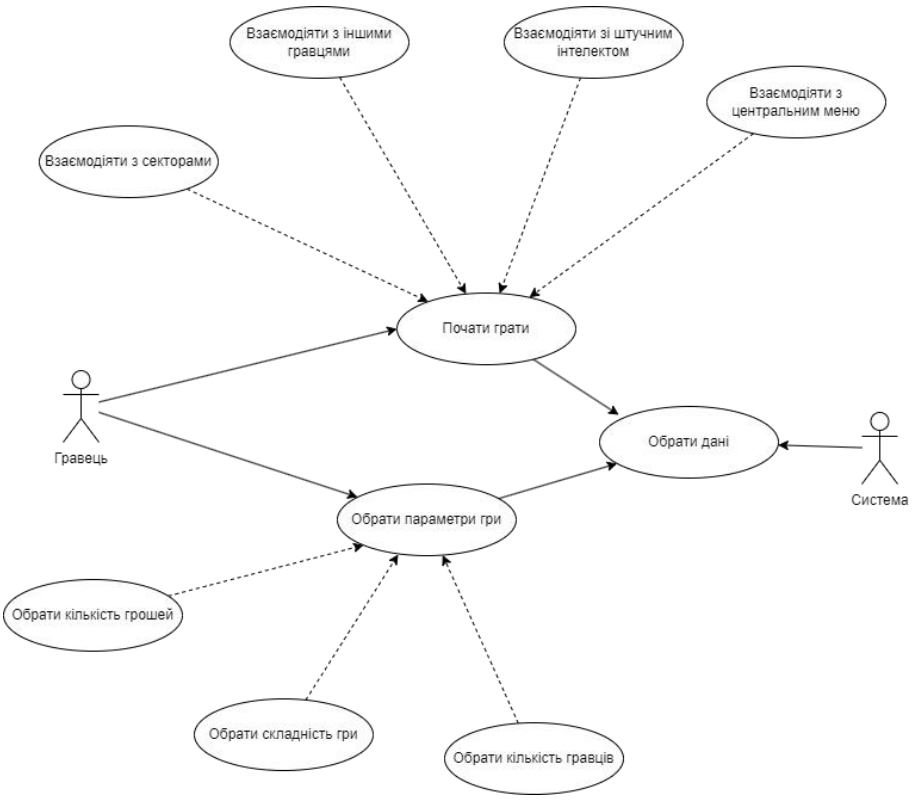


Рисунок 2.2 – Діаграма прецедентів

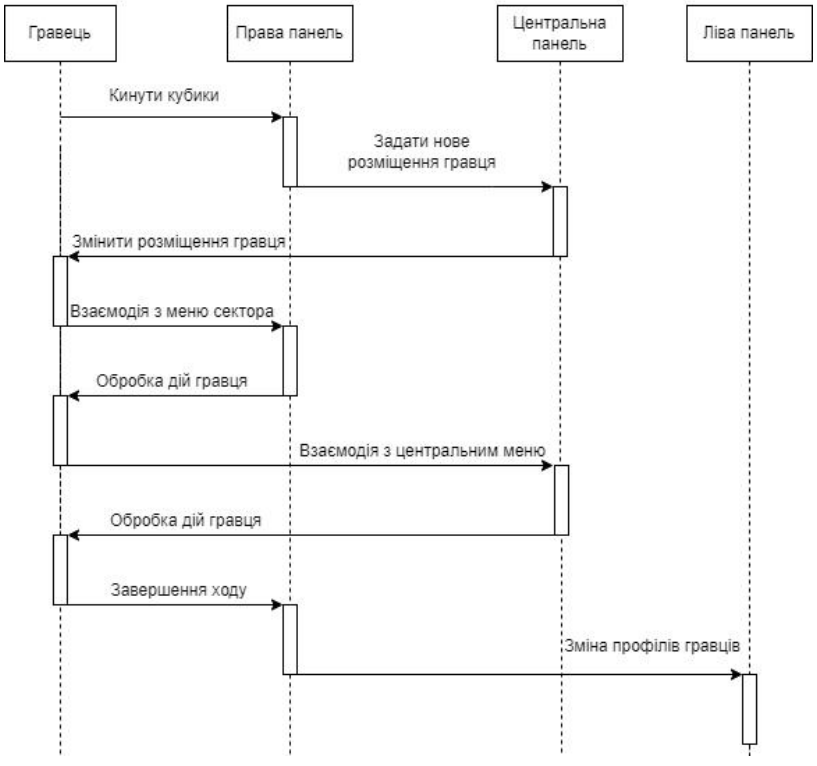


Рисунок 2.3 – Діаграма послідовностей

2.2 Розробка блок-схем алгоритмів роботи програми

Для візуалізації алгоритмів програми було обрано блок-схеми, які є поширеним засобом представлення алгоритмів та процесів. Блок-схема використовує блоки різних форм, пов'язані лініями, що вказують на послідовність кроків.

Програмний додаток гри «Монополія» розподіляється на три сцени. Для кожної з цих сцен буде розроблена власна блок-схема, що відобразить послідовність операцій та логіку роботи програми в кожному етапі гри.

Цей підхід дозволяє структурувати та узгоджувати роботу програми на рівні алгоритмів перед переходом до фази написання коду. Окрім того, блок-схеми є ефективним інструментом для комунікації та спільної роботи з розробницьким колективом, оскільки вони надають зрозуміле та лаконічне представлення процесів та взаємодії компонентів програми.

Перша сцена – це «Головне меню». В головному меню у користувача є декілька варіантів дій. Початок гри відкриває нову сцену «Меню вибору гравців». Налаштування, завантаження та про розробника відкривають меню відображення зі своїми параметрами. Вихід з ігри закриває додаток. Блок-схема головного меню зображена на малюнку 2.4.

Друга сцена – це «Меню вибору гравців». Знаходячись в цьому меню користувач може обрати кількість гравців (мінімальна кількість 2). Для кожного гравця потрібно обрати індивідуальні параметри, окрім цього потрібно обрати чи є гравець ботом, чи людиною. Після того, як потрібна кількість гравців була обрана і кожному з них були обрані параметри потрібно обрати початкову кількість грошей (за замовчуванням 2000). Якщо серед гравців був обраний хоч один гравець-бот, тоді потрібно обрати складність штучного інтелекту. Після всіх виборів можна розпочинати гру. Алгоритм роботи «Меню вибору гравців» зображено на рисунку 2.5.

Третя сцена – це безпосередньо «Гра». На цій сцені відбувається основна логіка програми та реалізовано штучний інтелект у вигляді ботів, які імітують дії людини. Основний алгоритм зображений на рисунку 2.6 та 2.7.

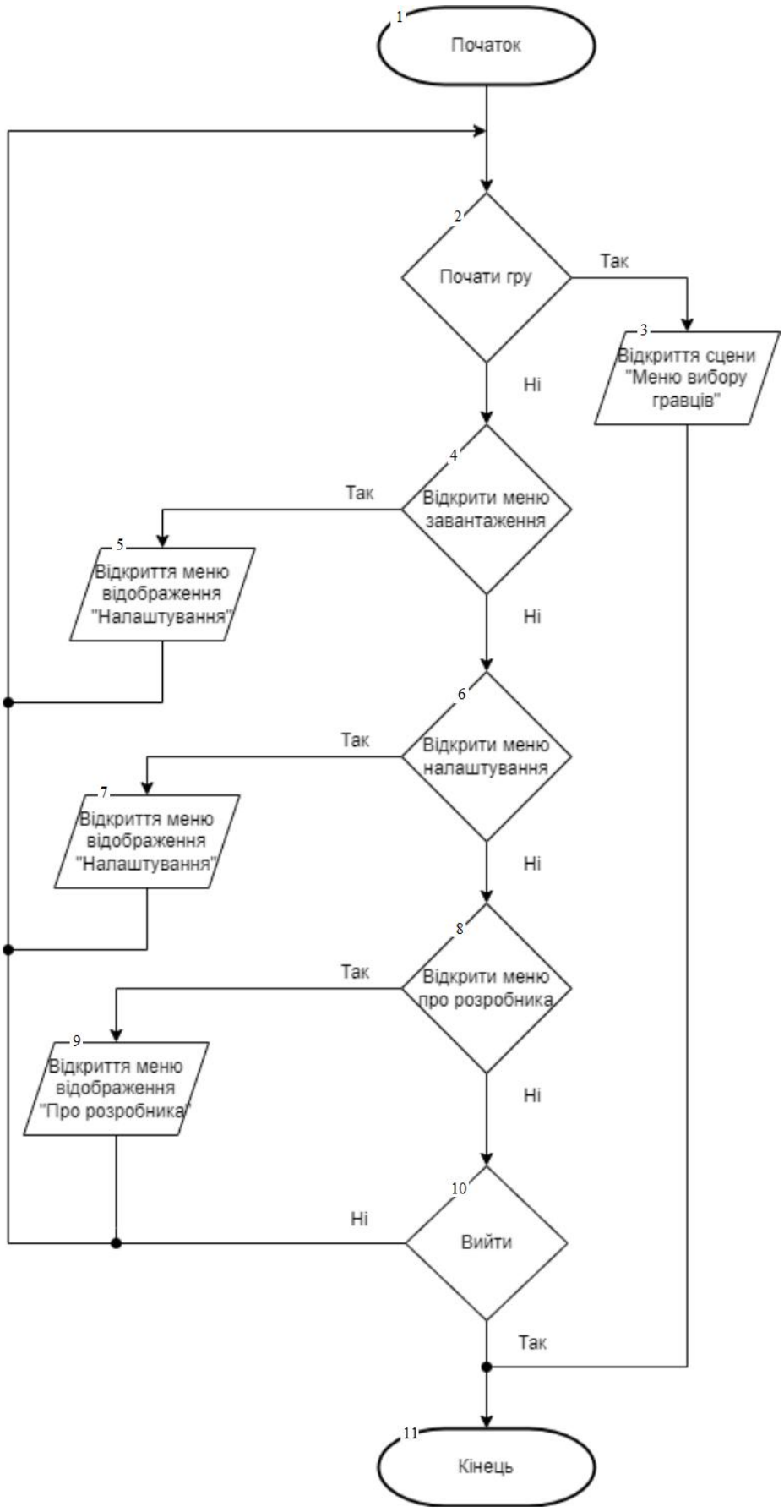


Рисунок 2.4 – Алгоритм головного меню

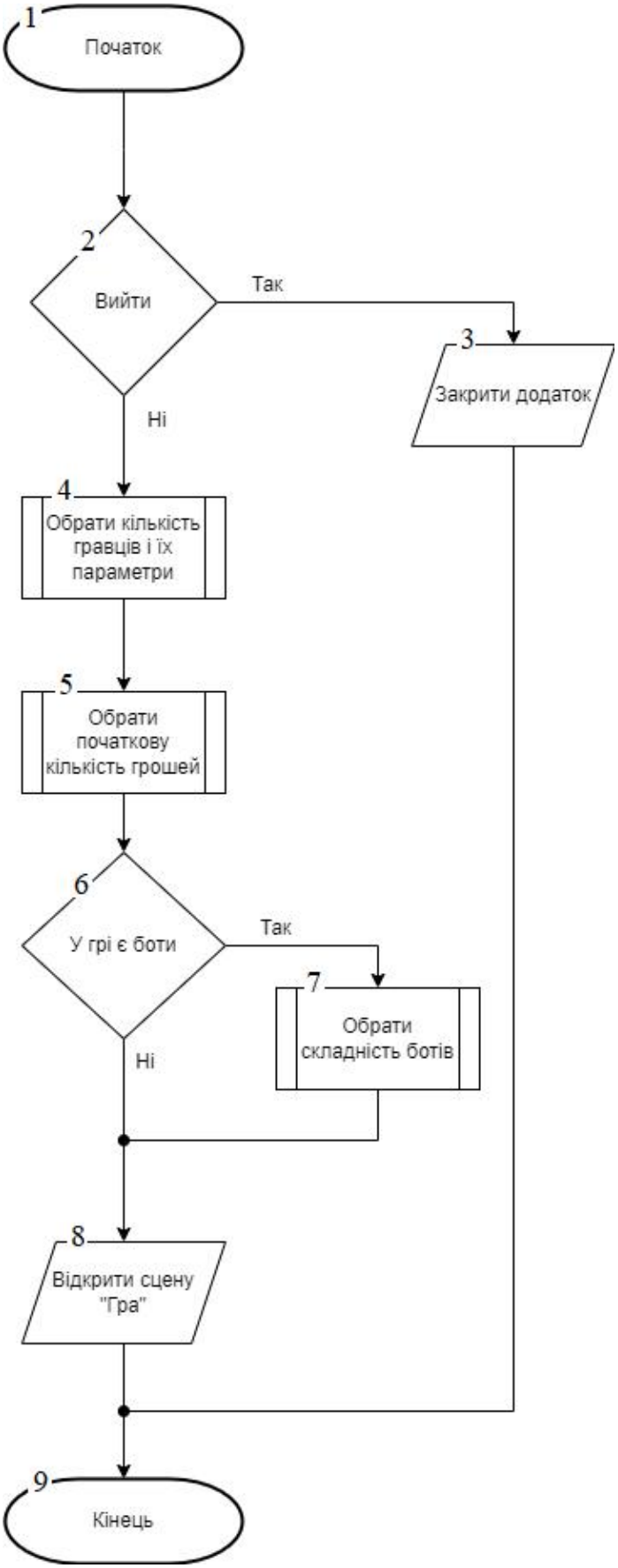


Рисунок 2.5 – Алгоритм меню вибору гравців

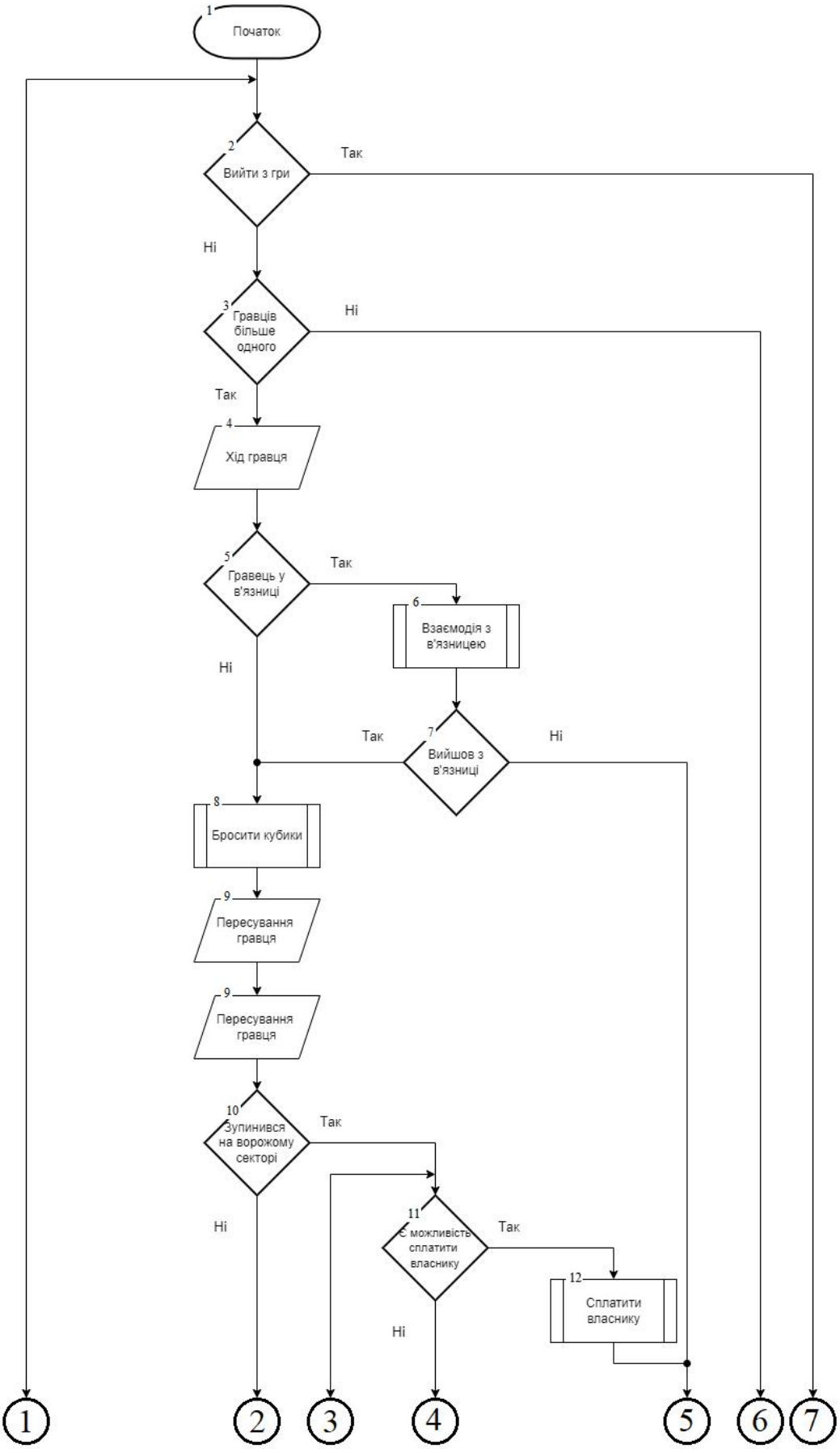


Рисунок 2.6 – Основний алгоритм роботи сцени «Гра»

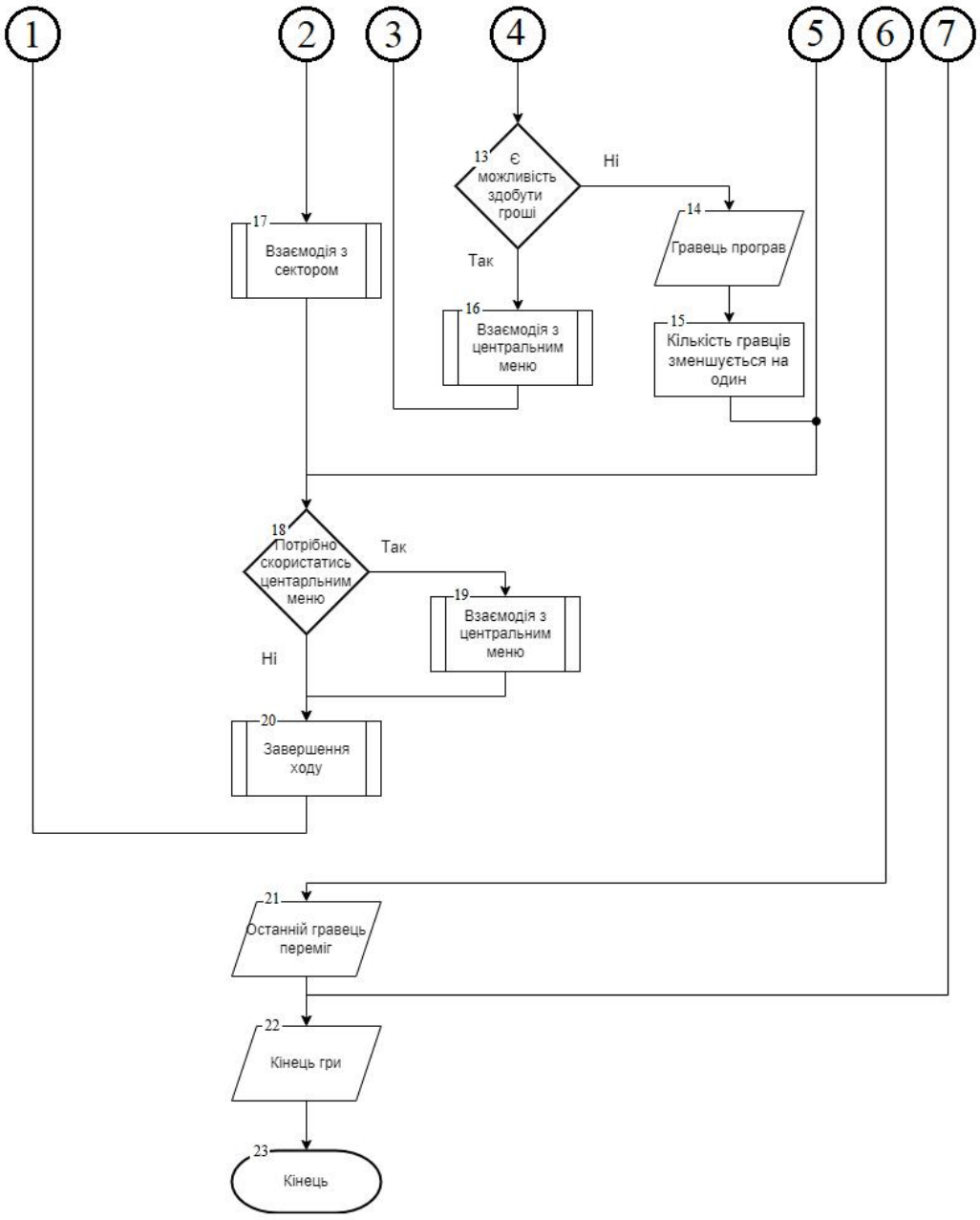


Рисунок 2.7 – Основний алгоритм роботи сцени «Гра»

2.3 Розробка методів штучного інтелекту

У грі реалізовано дві складності гри. Складність гри залежить від того наскільки «розумним» буде штучний інтелект.

Якщо обрати легку складність боти будуть виконувати дії за певним алгоритмом, звісно вони не є прямо зовсім «дурними» вони все ще можуть показати себе і завдати гравцям певних хлопот і навіть перемогти при деяких умовах.

Якщо є обрати складну складність, то боти вже будуть не просто мати якусь можливість перемогти, а будуть що не на є головними претендетами на перемогу.

2.3.1 Розробка методу побудови модуля штучного інтелекту на основі дерева ухвалення рішень

Легкі боти реалізовані на примітивному рівні і виконують конкретні дії, які прописані у них в логіці. Алгоритм дій бота зображений на рисунку 2.8.

Під час свого ходу бот має наступну поведінку.

1. Якщо він знаходиться у в’язниці – він може взаємодіяти лише з в’язницею. В нього є дві можливі дії:

1.1. В бота достатньо коштів для виплати залогу. Він виплачує і виходить з в’язниці.

1.2. Коштів не достатньо. Бот кидає кубики в надії, що випаде 3 дубля поспіль.

2. Якщо бот не у в’язниці, або він вийшов з в’язниці – він кидає кубики і взаємодіє з сектором, на який потрапив, від сектору залежить алгоритм дій:

2.1. Сектор «Поле»/«Бізнес» – не куплене.

2.1.1. В бота вже є сектор з цього району, або в нього менше 6 куплених секторів і в нього достатньо грошей – він купляє сектор.

2.1.2. В іншому випадку він не взаємодіє з меню сектора.

2.2. Сектор «Поле»/«Бізнес» – куплене.

2.2.1. В бота вистачає коштів, щоб заплатити власнику сектора – він виплачує.

2.2.2. В іншому випадку він намагається знайти кошти наступними шляхами:

2.2.2.1. Взяти кредит. Якщо боту не вистачає менше ніж він може отримати за прохід через сектор «Старт» – він бере кредит.

2.2.2.2. В іншому випадку – продає будинки з секторів і самі сектори.

2.2.2.3. Якщо і після цього, йому не вистачає кидає кубик надії.

2.3. Сектор «Зупинка».

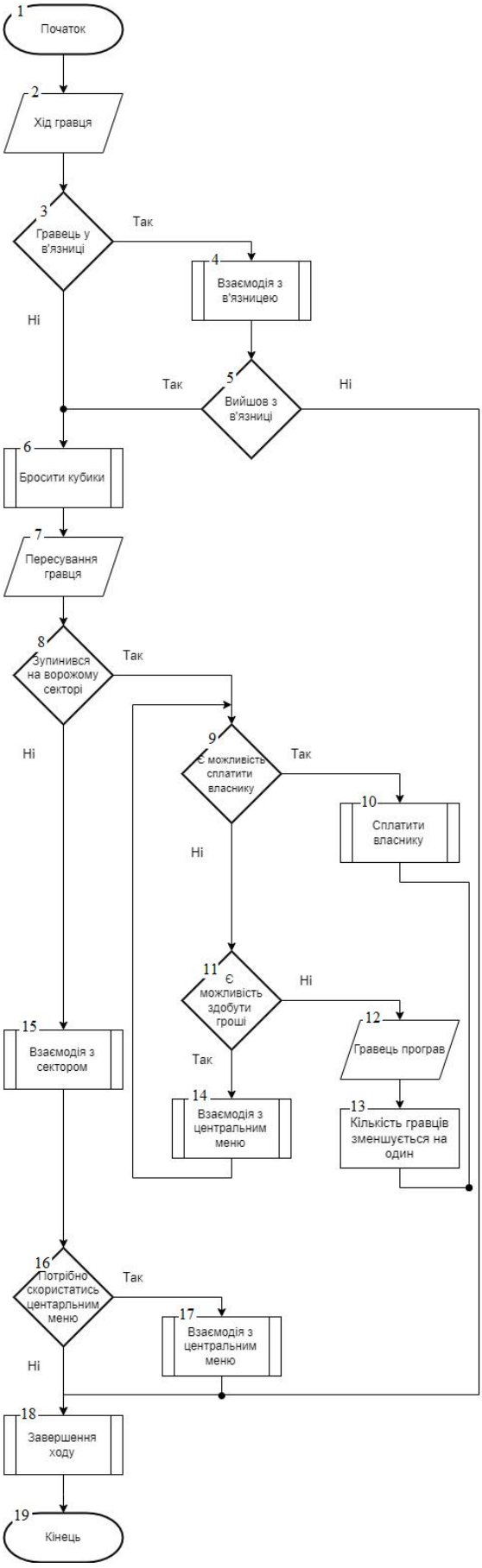


Рисунок 2.8 – Алгоритм роботи бота

2.3.1. Якщо на шляху до наступної зупинки знаходиться багато, а саме 3 або більше, ворожих секторів бот користується зупинкою.

2.3.2. В іншому випадку ніяк не взаємодіє з меню сектора.

2.4. Сектор «Казино». Є вірогідність в 40%, що бот захоче грати.

2.4.1. Якщо він хоче грати – ставка вибирається випадково: шанс на ставку в 5% від всіх коштів – 50%, в 10% – 35%, в 20% – 10% і в 50% – 5%.

2.4.2. В іншому випадку ніяк не взаємодіє з сектором.

2.5. Сектор «Таксі».

2.5.1. Якщо боту не вистачає одного сектору для того, щоб в нього з’явився район – він обирає потрібний сектор і, якщо в нього достатньо коштів їде до того сектору.

2.5.2. Якщо попереду багато ворожих секторів, 3 або більше, переходить до сектору «Бонус», який знаходиться перед сектором «Старт».

2.5.3. В іншому випадку ніяк не взаємодіє з сектором.

3. Після взаємодії з секторами, бот може взаємодіяти з центральним меню:

3.1. Меню «Обмін». Якщо боту не вистачає 1 чи двох секторів, і вони є у інших гравців, він починає обмін. На обміні він обирає потрібного гравця і потрібний сектор, після чого виставляє суму грошей, формула по якій визначається максимальна кількість коштів, яку може заплатити бот наступна: price + price *(150 + 50*n)%, де price – це ринкова вартість сектору, а n – кількість районів у бота. Якщо гравця і бота все влаштовує відбувається обмін.

3.2. Меню «Бізнес». Якщо боту не вистачає лише бізнесу для отримання району – він купляє цей бізнес через меню.

3.3. Меню «Дома». Якщо в бота коштів більше за 20% від проходу круга, і цього вистачає на будівництво дома – він будує дім.

3.4. Меню «Банк», «Кубик надії», «Продаж» і «Дома» для продажу домів, використовується, якщо потрібно оплатити потрапляння на ворожий сектор.

4. Після взаємодії з центральним меню бот закінчує хід.

Взаємодія гравця з ботом. Під час гри гравець може взаємодіяти зі штучним

інтелектом через центральне меню «Обмін». Для цього потрібно орати в випадаючому меню бота обрати вимогу обміну і натиснути кнопку готовності, після чого бот вирішує, влаштовує його цей обмін, чи ні. Логіка ботам під час такого обміну наступна:

1. Якщо гравець хоче купити сектор і в бота лише 1 сектор з цього району:

1.1. Якщо у гравця 2 сектора з одного району і він хоче купити третій – тоді формула найменшої вартості наступна: price + price * (200 + 50*n)%.

1.2. Якщо у гравця 1 сектор з одного району і він хоче купити другий – тоді формула найменшої вартості наступна: price + price * (100 + 50*n)%.

2. Продаж здійснюється за схемою, яка була описана в минулому списку пункт 3.1.

2.3.2 Розробка методу штучного інтелекту для ідентифікації та адаптації до індивідуальних стилів гри гравців у «Монополії»

Для того, щоб зробити ботів більш «розумними» було прийнято рішення додати штучному інтелекту можливість розвиватися. Це було реалізовано наступним чином:

У грі реалізована не така велика кількість механік, але все ж таки і не настільки мала. Для того, щоб боти могли розвиватися було проведено певна кількість партій між ботами. В стандартний алгоритм ботів (легка складність) була внесена певна кількість змін.

Під час свого ходу вони виконували випадкові дії, які незалежали не від чого. Кожна така дія записувалась в базу даних разом з коефіцієнтом «правильності» даної дії. Після того, як один з ботів перемагав коефіцієнт всіх його дій мав 1, а програвших 0.

Після того, як була проведена певна кількість партій і так звана симуляція закінчилась починалась нова, де боти вже вирішували, що їм робити не випадково, а бравши інформацію з бази даних, де обирали дію з коефіцієнтом 1 (насправді вони беруть дію з найбільшим рівнем коефіцієнта). Якщо подібної дії не має в базі, то на поміч знову приходить випадковість.

За таким алгоритмом пройшло чимало симуляцій і була згенерована непогана база даних з доволі великою кількістю дій.

2.4 Обґрунтування вибору інтерфейсу

2.4.1 Огляд різновидів інтерфейсів

Для досягнення успішних результатів у нашому проекті необхідно визначити найбільш підходящий тип інтерфейсу. Розглянемо кілька різновидів інтерфейсів, які можуть бути релевантні для нашого завдання:

1. Текстовий інтерфейс: Текстові інтерфейси зазвичай використовуються в командному режимі та вимагають введення тексту або команд користувачем. Вони можуть бути ефективними для оптимізації продуктивності, але можуть бути менш інтуїтивними для незвичайних користувачів. Приклад текстового інтерфейсу зображено на рисунку 2.9 [15].

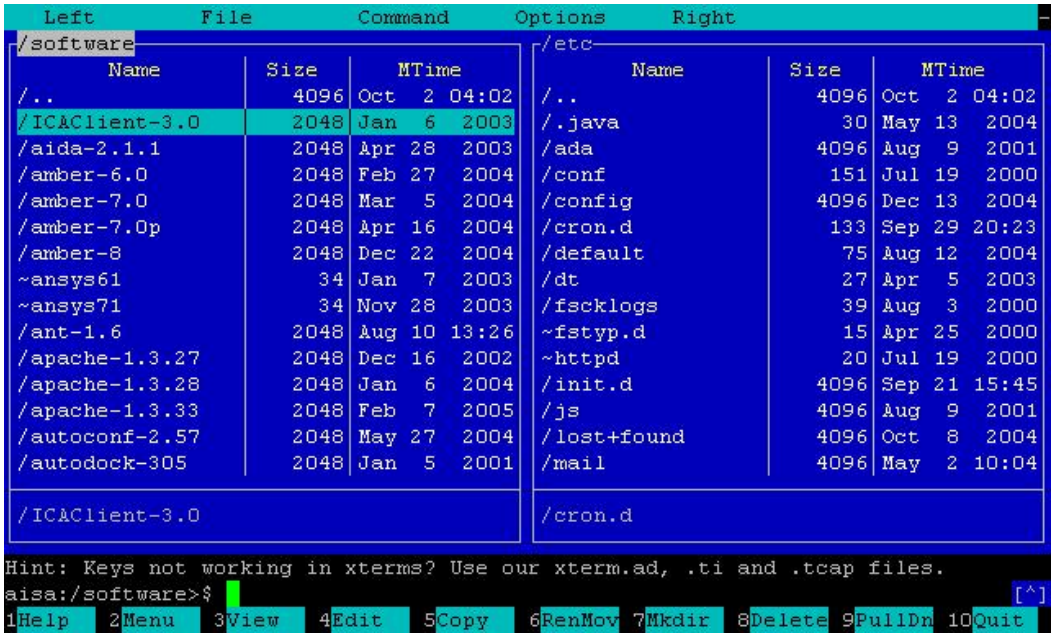


Рисунок 2.9 – Приклад текстового інтерфейсу

2. Графічний інтерфейс (GUI): GUI надає графічні об'єкти, такі як кнопки, меню, та інші елементи, які користувач може взаємодіяти з допомогою миші або клавіш клавіатури. GUI зазвичай вважається більш інтуїтивним і доступним для більшості користувачів. Приклад графічного інтерфейсу

зображений на рисунку 2.10 [16].

Рисунок 2.10 – Приклад графічного інтерфейсу

3. Веб-інтерфейс: Веб-інтерфейси базуються на веб-технологіях і доступні через веб-браузер. Вони дозволяють забезпечити доступ до програми або сервісу через Інтернет, що робить їх особливо зручними для віддалених користувачів. Приклад веб-інтерфейсу зображений на рисунку 2.11 [17].

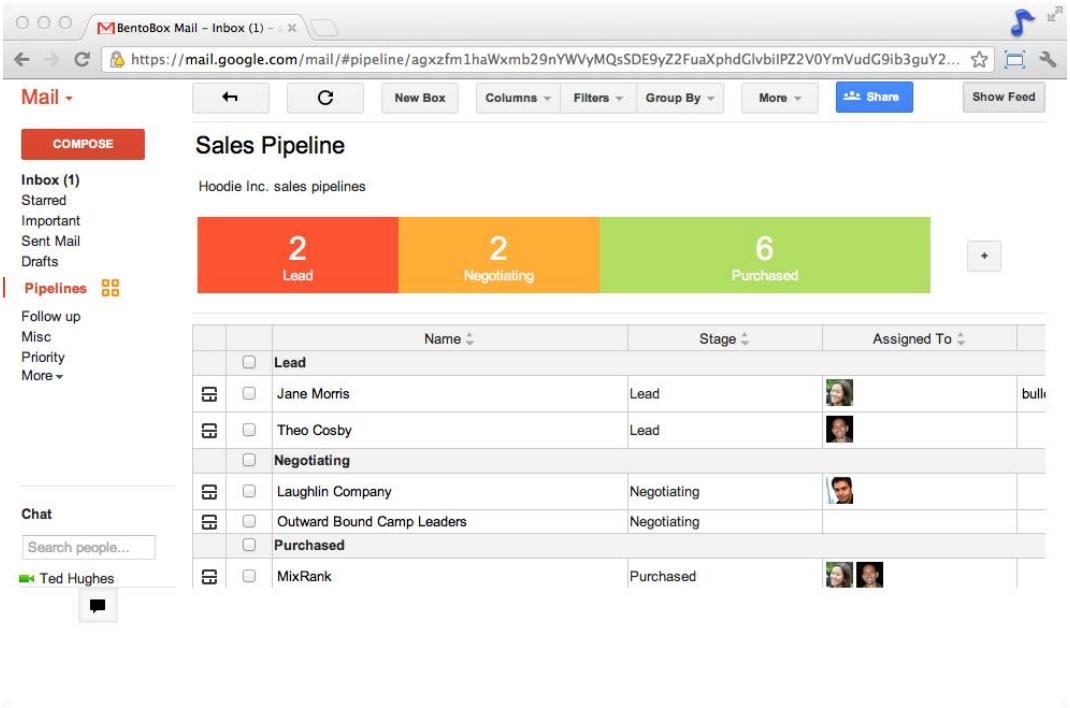


Рисунок 2.11 – Приклад веб-інтерфейсу

2.4.2 Переваги та недоліки

Порівнюючи різні типи інтерфейсів, ми можемо виділити певні переваги та недоліки:

Текстовий інтерфейс:

- Переваги:
  - Ефективний для введення команд та операцій.
  - Може вимагати менше системних ресурсів.
- Недоліки:
  - Менш інтуїтивний для користувачів, які не знайомі з командами.
  - Обмежена можливість візуальної інформації.

Графічний інтерфейс (GUI):

- Переваги:
  - Інтуїтивний для більшості користувачів.
  - Забезпечує великий обсяг візуальної інформації.
- Недоліки:
  - Вимагає більше часу та ресурсів для розробки.
  - Може бути важким у використанні на пристроях з обмеженим розширенням дисплею.

Веб-інтерфейс:

- Переваги:
  - Доступний через Інтернет, що робить його глобально доступним.
  - Може бути легко оновлений для всіх користувачів.
- Недоліки:
  - Вимагає підтримки серверної інфраструктури.
  - Залежить від доступу до Інтернету.

2.4.3 Вибір графічного інтерфейсу

Для розроблюваного ігрового додатку було обрано графічний інтерфейс (GUI). Цей вибір обґрунтовується його інтуїтивністю для більшості користувачів,

що є важливим аспектом у використанні нашого проекту. GUI також забезпечує можливість візуальної інтеракції з користувачем та підтримує багатий набір функцій. Цей вибір відповідає меті нашого проекту та забезпечує оптимальну якість користувацького досвіду.

2.5 Опис інтерфейсу

Інтерфейс програмного додатку складається з трьох основних вікон.

На рисунку 2.12 зображена графічна схема інтерфейсу «Головного меню».В головному вікні знаходить поле в якому відображаються меню різноманітних кнопок. Інтерфейс відображення кнопки «Завантажити гру» зображений на рисунку 2.13, кнопки «Налаштування» зображений на рисунку 2.14, кнопки «Про розробника» на рисунку 2.15.

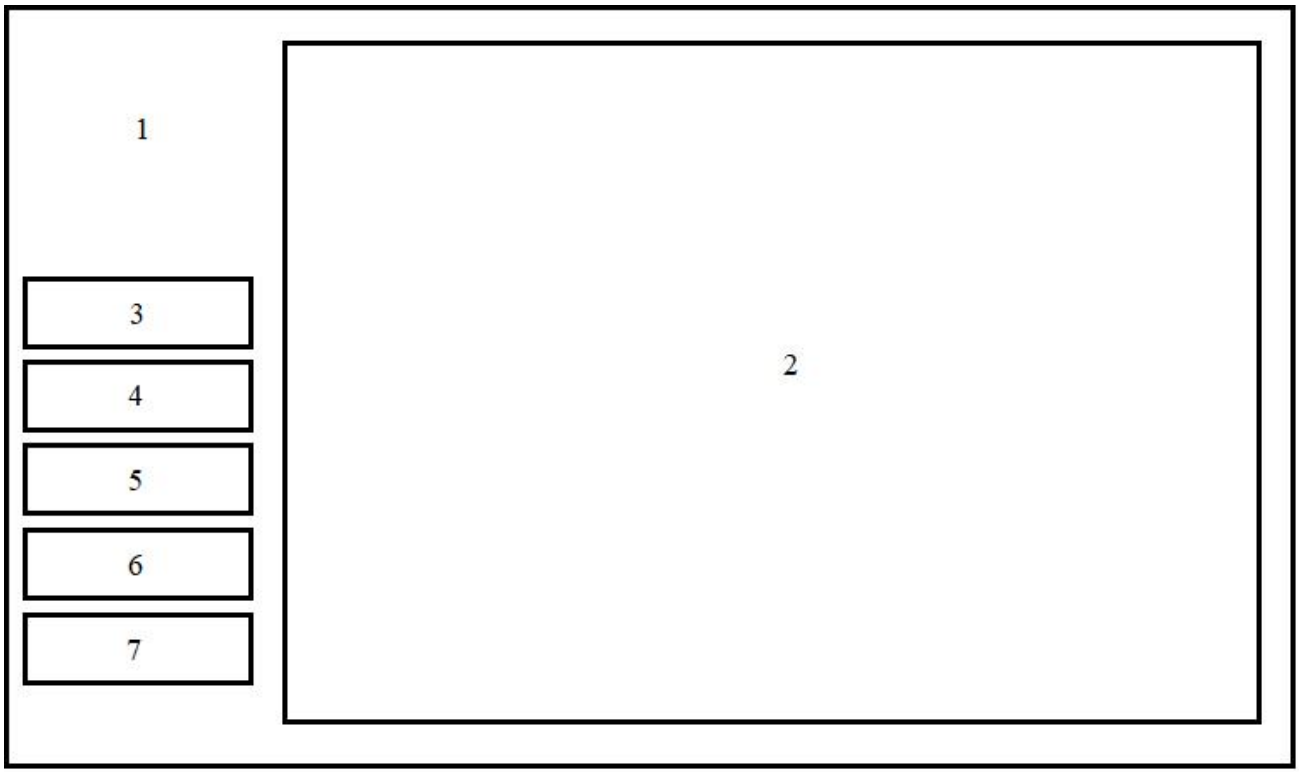


Рисунок 2.12 – Графічний інтерфейс головного меню

Елементи головного меню :

1. Фонове зображення.

2. Вікно відображення.
3. Кнопка «Почати гру».
4. Кнопка «Завантажити гру».
5. Кнопка «Налаштування».
6. Кнопка «Про розробника».
7. Кнопка «Вихід».

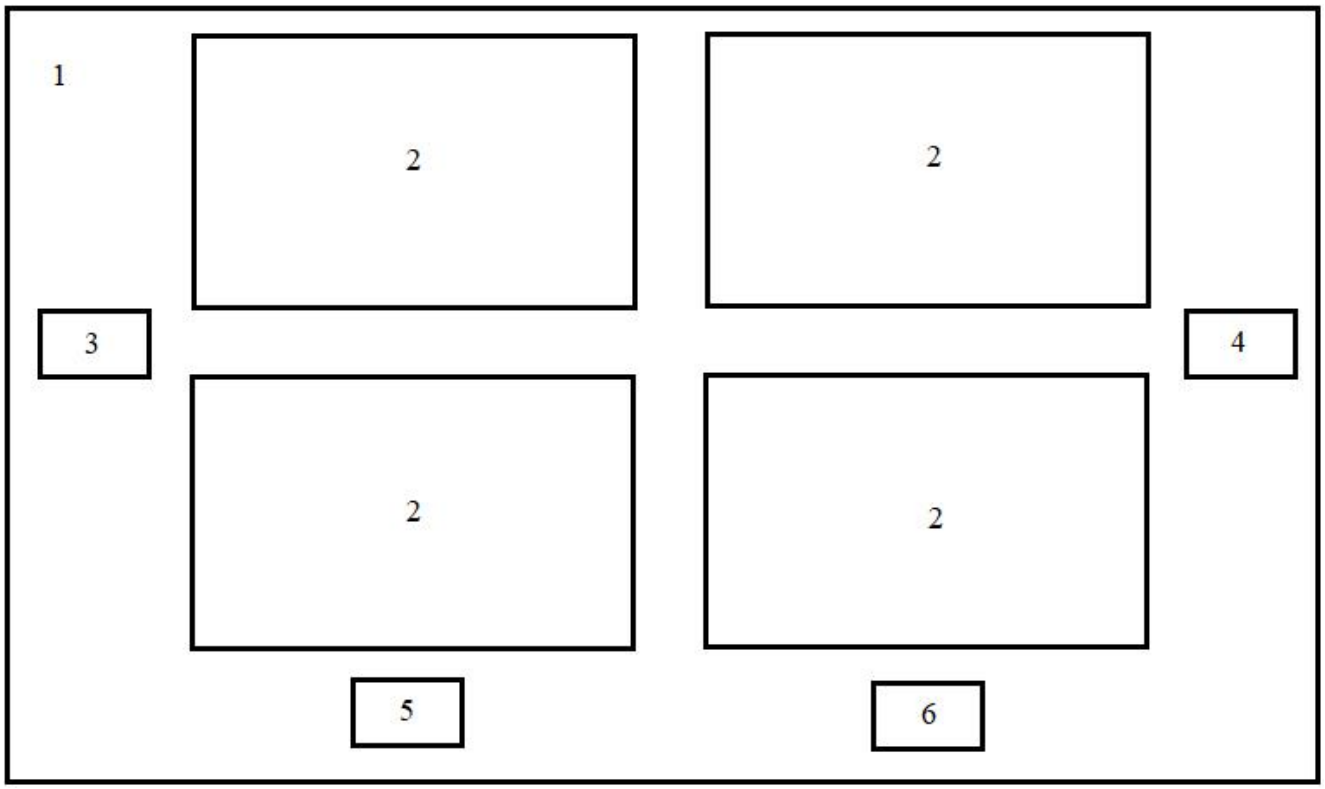


Рисунок 2.13 – Відображення меню завантаження

Елементи меню завантаження :

1. Фонове зображення.
2. Зображення файлів збереження.
3. Кнопка «Перейти вліво».
4. Кнопка «Перейти вправо».
5. Кнопка «Видалити збереження».
6. Кнопка «Завантажити збереження».

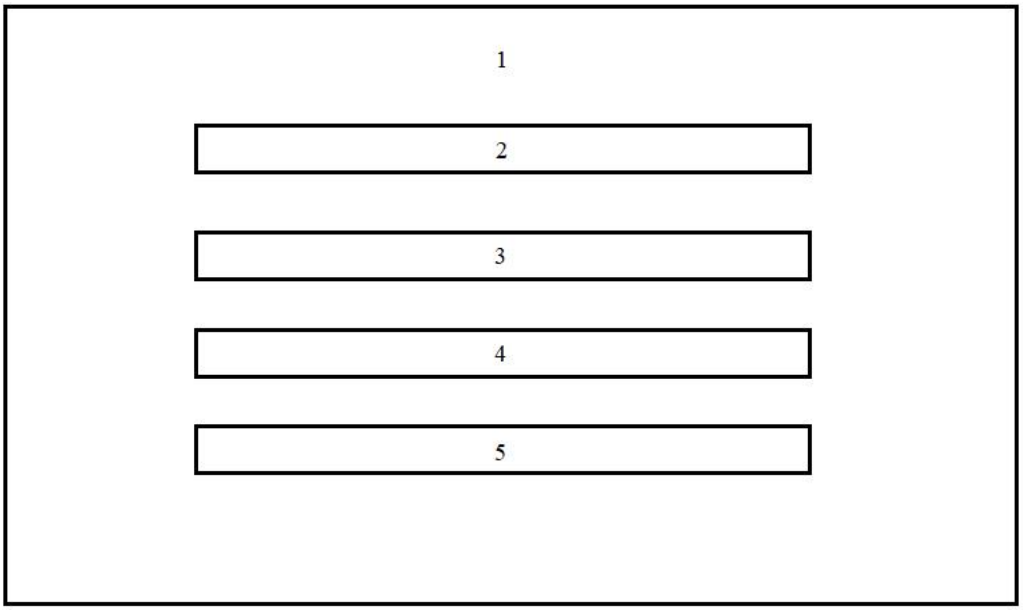


Рисунок 2.14 – Відображення меню налаштування

Елементи меню налаштування :

1. Фонове зображення.
2. Зміна гучності.
3. Зміна розширення.
4. Зміна формату екрана.
5. Зміна мови.

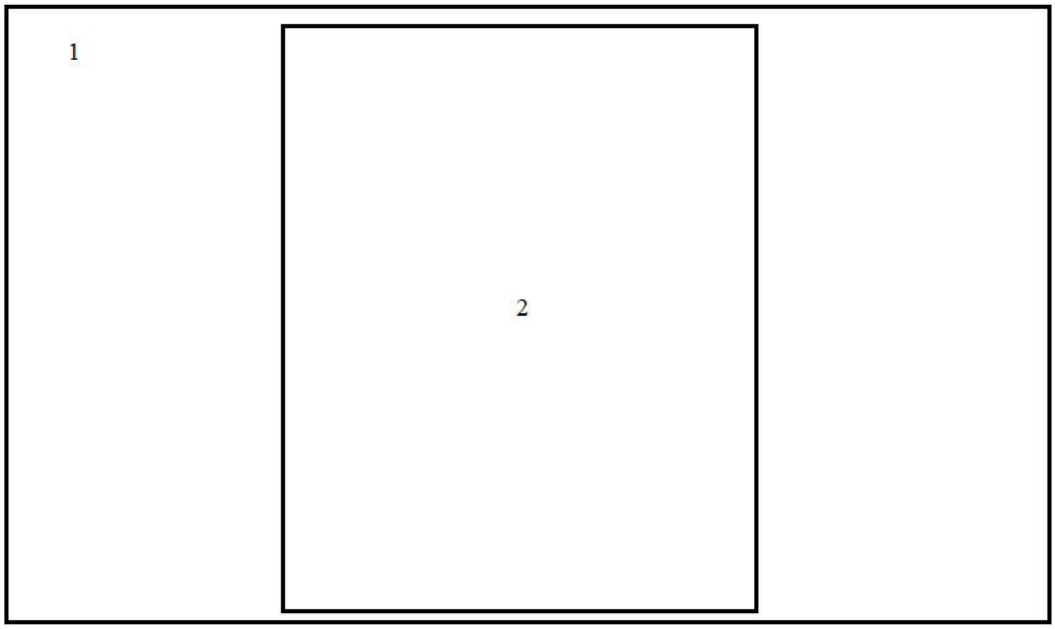


Рисунок 2.15 – Відображення меню інформації про розробника

Елементи меню інформації про розробника :

1. Фонове зображення.
2. Інформація про розробника.

На рисунку 2.16 зображена графічна схема інтерфейсу «Меню вибору гравців».

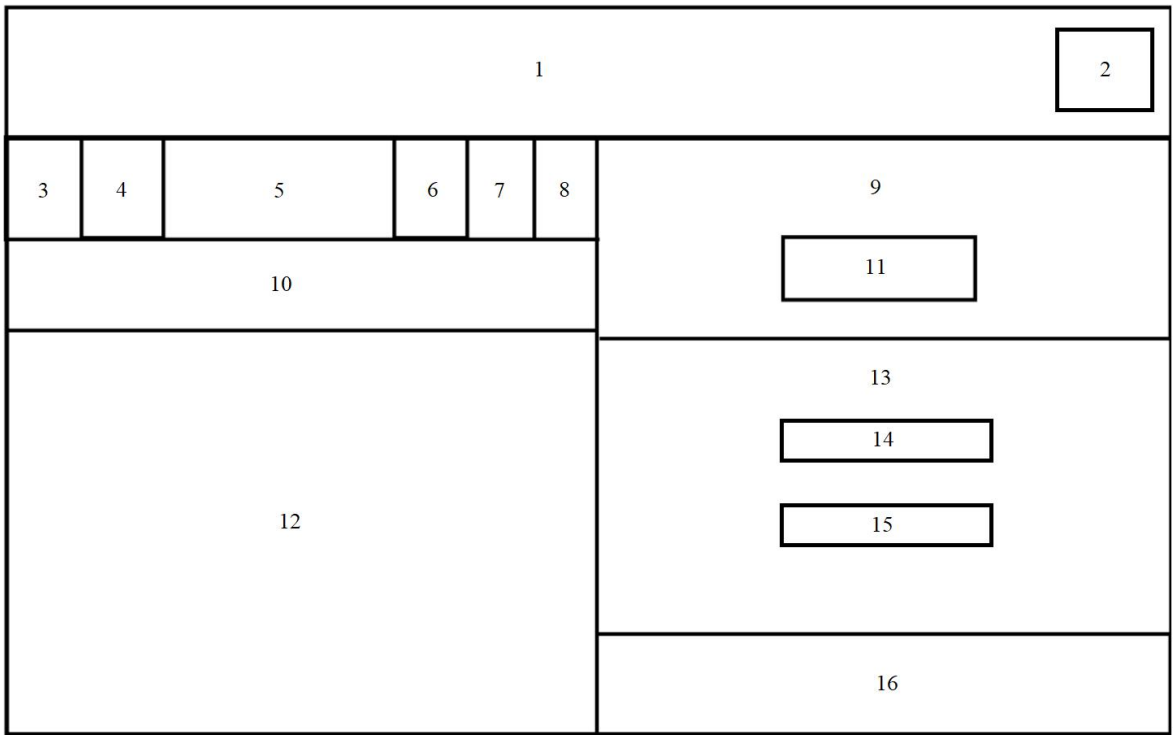


Рисунок 2.16 – Графічний інтерфейс меню вибору гравців

Елементи меню вибору гравців :

1. Шапка меню.
2. Кнопка «Повернутись у головне меню»
3. Стовпець «№».
4. Стовпець «Портрет».
5. Стовпець«Ім’я».
6. Стовпець «Колір».
7. Стовпець «Людина/Бот».
8. Стовпець «Видалити».
9. Поле з вибором грошей.

10. Кнопка «Створити».
11. Поле для вводу початкових грошей.
12. Поле для майбутніх гравців.
13. Поле з вибором складності.
14. Вибір легкої складності.
15. Вибір складної складності.
16. Кнопка «Почати гру».

На рисунку 2.17 зображена графічна схема інтерфейсу самої гри «Гра».

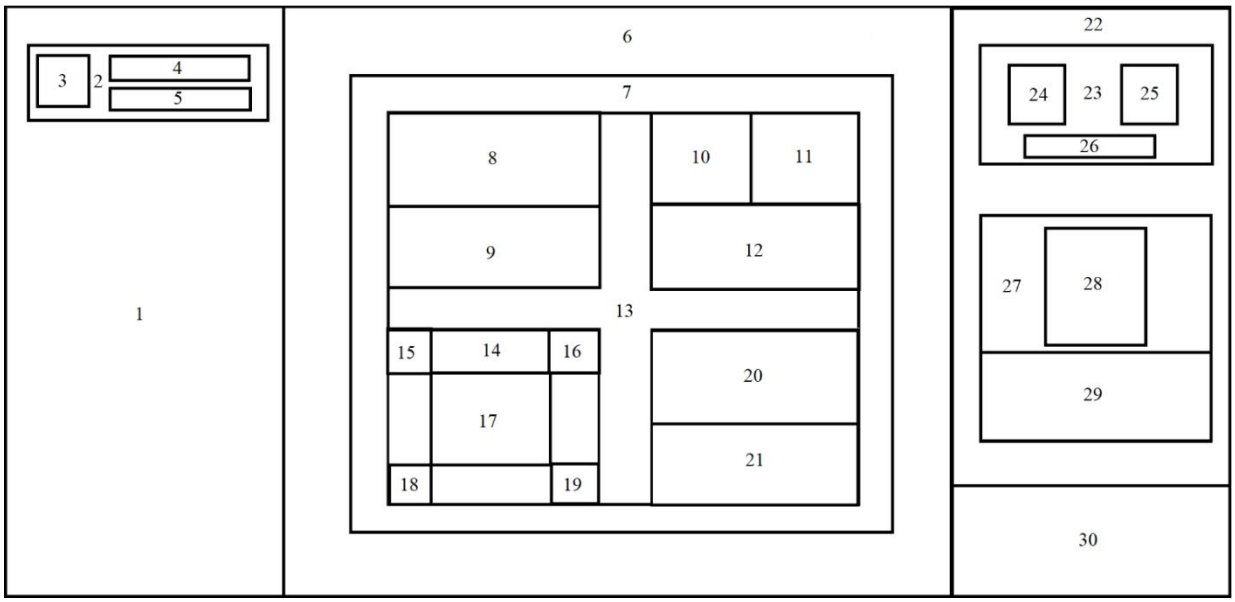


Рисунок 2.17 – Графічний інтерфейс гри

Елементи гри :

1. Ліва панель з профілями гравців.
2. Профіль гравця.
3. Портрет гравця.
4. Ім’я гравця.
5. Кількість грошей гравця.
6. Панель для секторів.
7. Дорога.
8. Кнопка «Банк».

9. Кнопка «Обмін».
10. Кнопка «Здатись».
11. Кнопка «Кубик надії».
12. Кнопка «Продаж».
13. Центральне меню.
14. Панель погоди.
15. Зображення погоди 1.
16. Зображення погоди 2.
17. Зображення діючої погоди.
18. Зображення погоди 3.
19. Зображення погоди 4.
20. Кнопка «Будинки».
21. Кнопка «Бізнес».
22. Права панель з інформацією про діючий хід.
23. Панель з кубиками.
24. Лівий кубик.
25. Правий кубик.
26. Кнопка «Бросити кубики».
27. Панель активної карти.
28. Зображення активної карти.
29. Меню активної карти.
30. Придбані активним гравцем сектори.
31. Кнопка «Завершити хід».

Інтерфейс гри складається з трьох частин:

- Ліва панель – на цій панелі розташовуються профілі гравців. В профілях гравців знаходиться їх портрет, ім’я, кількість коштів, колір та індикатор людина/бот.
- Центральна панель – на цій панелі знаходяться 3 важливі елементи: карти, або сектора, це об’єкти з якими може взаємодіяти гравець, коли потрапляє на них під час свого ходу; дорога – по ній відбувається візуальне пересування

гравців по полю; центральне меню – в центральному меню є багато різних об’єктів з якими може взаємодіяти гравець, а також індикатор погоди, в центра розташована діюча погода, а в кутах можлива.

- Права панель – на правій панелі зображаться статус діючого ходу. В ній гравець може кинути кубики, тим самим почавши свій хід, на панелі відображаються властивості сектору на який потрапляє гравець після кидка кубиків, а саме: зображення карти на її особисте меню, меню змінюється в залежності від типу сектору. Також на цій панелі зображені сектора, які куплені у діючого гравця і є кнопка «Завершити хід», за допомогою якою гравець може завершити хід і передати хід наступному гравцю.

2.6 Висновки

У другому розділі були проаналізовані декілька типів інтерфейсу серед яких було обрано графічний, через його зручність для користувача.

Було створено і детально описано інтерфейс розроблюваного додатку. Всі вікна програмного додатку були розписані.

Для розробки ігрового додатку були створені UML діаграми прецедентів на послідновності. Окрім того, для основних трьох сцен було розроблено блок-схеми алгоритмів.

В кінці розділу було проаналізовано розробку штучного інтелекту, описана робота примітивного бота для легкої складності та описана модель створення ботів для складної складності.

3 РОЗРОБКА ФУНКЦІОНАЛУ ТА МОДУЛІВ ПРОГРАМИ

3.1 Обґрунтування вибору програмних засобів для розробки

Перед початком безпосередньої розробки додатку – необхідно визначитись із засобами розробки, а саме:

- обрати мову програмування;
- інтсрумент для розробки відеоігор;
- обрати середовище написання коду.

3.1.1 Вибір мови програмування

У світі розробки ігор вибір мови програмування визначає не лише технічні аспекти, а й впливає на продуктивність, швидкість розробки та підтримку. Однією з найпопулярніших мов для розробки ігор є C#, особливо в контексті використання ігрового двигуна Unity. У цьому розділі буде проведений докладний аналіз вибору мови програмування, порівняно C# з трьома іншими популярними мовами: Python, Java та C++.

1. C# (C-Sharp) – це мова програмування, розроблена компанією Microsoft, призначена для розробки різноманітних застосунків, включаючи веб-додатки та ігри. Основною характеристикою C# є його об'єктно-орієнтований підхід, що спрощує структуру та розробку програм [18].

Плюси:

- Інтеграція з Unity. C# є основною мовою програмування для Unity, що робить його ідеальним вибором для розробки ігор на цьому платформенному ігровому двигуні.
- Простота та ефективність. C# відомий своєю простотою та ефективністю, що сприяє швидкому робочому процесу та полегшує вивчення для новачків.
- Безпека типів. Система строго визначених типів дозволяє виявляти помилки під час компіляції, що робить код більш надійним.

Мінуси:

- Використання тільки з Unity. Хоча C# широко використовується у розробці ігор, його застосування обмежено контекстом використання Unity.

2. Python – це високорівнева, інтерпретована мова програмування, яка була створена Гвідо ван Россумом та вперше випущена у 1991 році. Вона відзначається простотою синтаксису, що дозволяє розробникам писати код менше та швидше, а також використовується в різних областях, включаючи веб-розробку, науку даних, штучний інтелект та геймдев [19].

Плюси:

- Простота та читабельність. Python славиться своєю простотою та читабельністю, що полегшує розробку та сприяє швидкому прототипуванню.
- Активна спільнота. Python має велику та активну спільноту розробників, що забезпечує широкий вибір бібліотек та ресурсів.

Мінуси:

- Виконання в інтерпретованому режимі. Хоча це дозволяє швидше розробляти, воно може впливати на швидкодію додатка, особливо у великих іграх.
- Обмежені можливості для графіки. Python може бути менш ефективним для великих графічних проєктів порівняно з іншими мовами.

3. Java – це високорівнева, об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (зараз власністю Oracle) у 1995 році. Вона відома своєю переносимістю, об'єктно-орієнтованим підходом та широким застосуванням у веб-розробці, мобільних додатках та інших областях програмування [20].

Плюси:

- Переносимість. Java славиться своєю переносимістю, що дозволяє запускати програми на різних платформах без модифікацій.
- Об'єктно-орієнтована мова. Java є об'єктно-орієнтованою мовою, що полегшує організацію та розробку складних систем.

Мінуси:

- Менший вплив у геймдеві. Хоча Java використовується в ігровій

розробці, вона менше поширена порівняно з іншими мовами, такими як C# чи C++.

- Вартість може залежати від виручки гри. Вартість використання Java може бути пов'язана з виручкою гри, що може впливати на фінансовий аспект розробки.

4. C++ – це високорівнева, об'єктно-орієнтована мова програмування, яка є розширенням мови програмування C. Вона була розроблена Бьярном Страуструпом у 1979 році та стала однією з найпопулярніших мов програмування у світі. C++ зазвичай використовується для розробки ефективних та продуктивних програм, включаючи ігри, операційні системи та програмне забезпечення високої продуктивності [21].

Плюси:

- Продуктивність. C++ відомий своєю високою продуктивністю, що робить його ідеальним для великих та ресурсомістких ігор.
- Близькість до апарату. Мова надає розробникам великий контроль над апаратною частиною системи, що корисно для оптимізації та роботи із специфічним обладнанням.

Мінуси:

- Складність та об'єм коду. C++ може бути складним для вивчення та використання, особливо для новачків. Об'єм коду також може бути більшим порівняно із більш високорівневими мовами.
- Більше відповідальності для розробників. Розробники повинні бути відповідальні за керування пам'яттю та уникати помилок, що може бути вимагаючим завданням.

Вибір мови програмування для розробки ігор – це компроміс між ефективністю, простотою та конкретними потребами проєкту. У порівнянні з іншими мовами, C# видається оптимальним вибором для розробки в Unity, забезпечуючи простоту, продуктивність та безпеку типів, необхідні для успішної ігрової розробки. Однак, вибір повинен бути зроблений з урахуванням конкретних вимог і особливостей проєкту.

3.1.2 Вибір інструменту розробки

1. Unity – це кросплатформенний ігровий рушій і інтегроване середовище розробки (IDE), що використовується для створення ігор та інтерактивних додатків. Відомий своєю дружбою до новачків та потужним функціоналом для досвідчених розробників. Unity підтримує розробку як 2D, так і 3D ігор, а також може бути використаний для створення додатків для різних платформ, включаючи PC, мобільні пристрої, консолі та віртуальну реальність. Інтерфейс ігрового рушія Unity зображений на рисунку 3.1 [22].

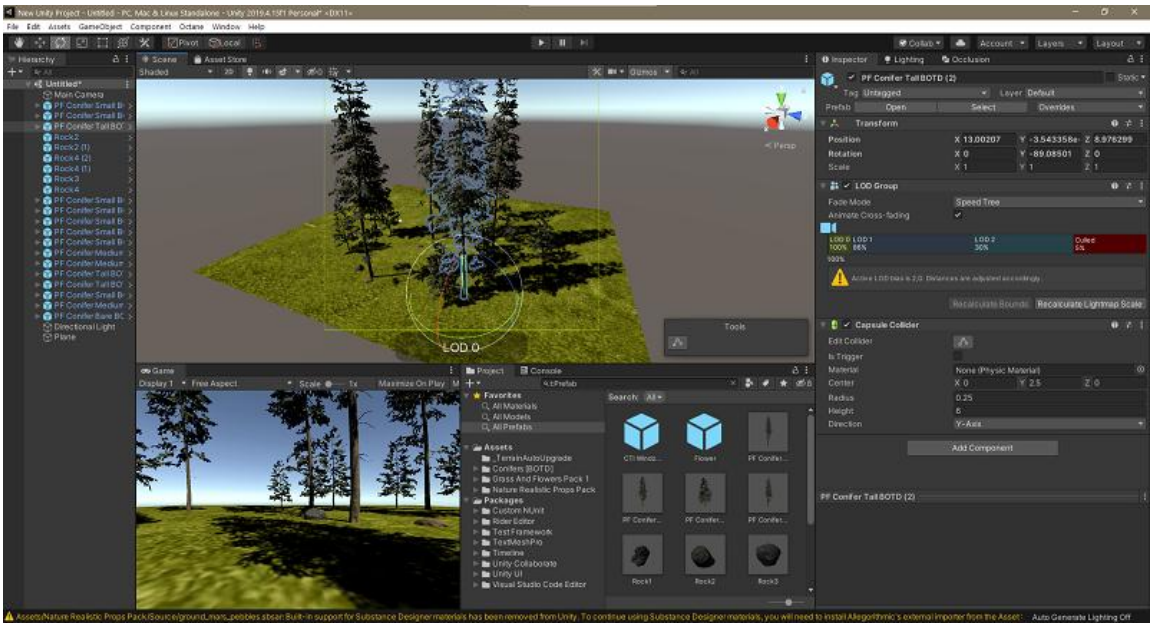


Рисунок 3.1 – Інтерфейс ігрового рушія Unity

- Мова програмування. Unity використовує мову програмування C#, яка вважається дружелюбною для новачків та має потужний функціонал для досвідчених розробників.
- Графічний движок. Unity має власний графічний движок, який забезпечує хороші можливості для розробки як 2D, так і 3D проектів. Зручний інтерфейс редактора сприяє швидкій ітерації.
- Вартість. Є безкоштовна версія Unity з рядом обмежень. Однак платні плани, такі як Unity Pro, надають доступ до додаткового функціоналу та інструментів.

- Спільнота та документація. Unity має велику та активну спільноту розробників. Є обширна документація, туторіали та форуми, що полегшують процес вивчення та розвитку.
- Кросплатформеність. Unity дозволяє розробляти ігри для різних платформ, включаючи PC, мобільні пристрої, консолі, віртуальну реальність та інші.

2. Unreal Engine – це потужний ігровий рушій, розроблений компанією Epic Games. Відомий своєю високоякісною графікою та реалістичною обробкою світла, Unreal Engine використовується для розробки великих ігор AAA-класу, а також інтерактивних віртуальних середовищ, архітектурних візуалізацій та інших проектів. Інтерфейс ігрового рушія Unreal Engine зображений на рисунку 3.2 [23].

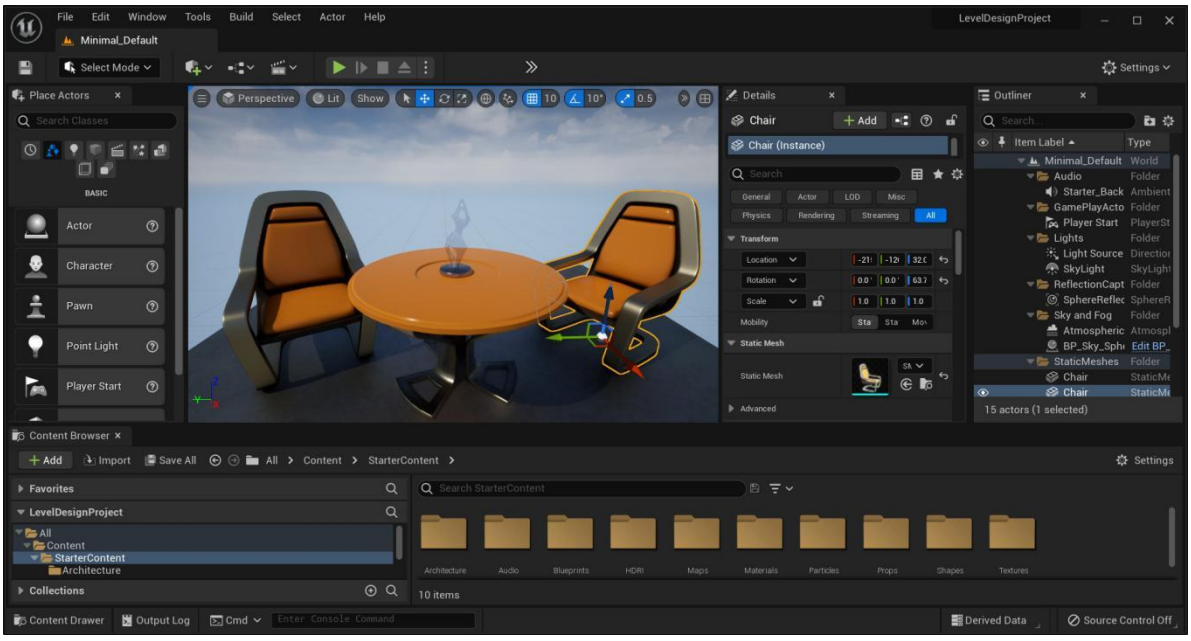


Рисунок 3.2 – Інтерфейс ігрового рушія Unreal Engine

- Мова програмування. Використовує C++, що може бути більш важливим для розробників з досвідом у цій мові.
- Графічний движок. Unreal Engine славиться своєю потужністю в графіці та реалістичною обробкою світла. Він часто використовується для створення великих ігор AAA-класу.
- Вартість. Безкоштовний для проектів, якщо ваш прибуток менше 1

мільйона доларів. Вимагається відрахування від прибутку для комерційних проектів.

- Спільнота та документація. Unreal також має значну спільноту і обширну документацію. Однак, через більшу специфіку та складність, іноді виникають певні виклики для новачків.
- Кросплатформеність. Підтримує широкий спектр платформ, включаючи консолі та VR.

3. Godot Engine – це відкритий ігровий рушій та розробницьке середовище з відкритим вихідним кодом. Розроблений спільнотою розробників, Godot відзначається своєю безкоштовністю, легкістю вивчення та використанням. Він підтримує як 2D, так і 3D розробку ігор, а також інших інтерактивних додатків. Godot використовує мову програмування GDScript, яка є подібною до Python, а також підтримує C# та VisualScript. Інтерфейс ігрового рушія Godot Engine зображений на рисунку 3.3 [24].

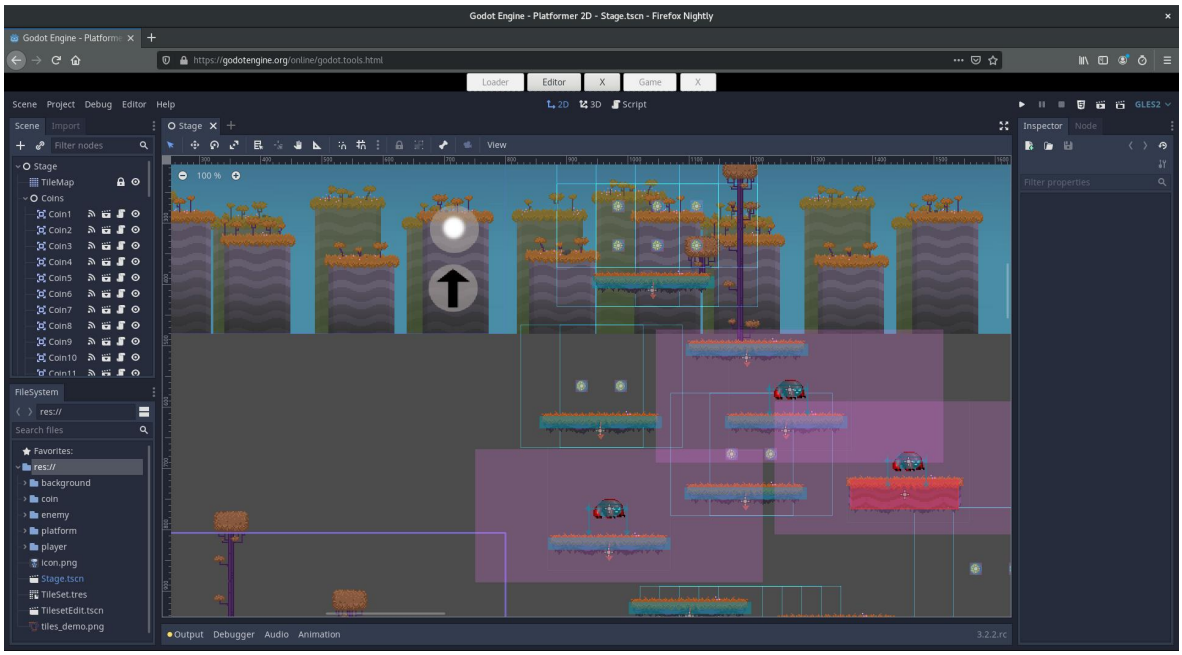


Рисунок 3.3 – Інтерфейс ігрового рушія Godot Engine

- Мова програмування. Підтримує GDScript, C#, і VisualScript. GDScript подібний до Python, що робить його легким для вивчення.

- Графічний движок. Має вбудований графічний движок з підтримкою 2D та 3D графіки. Хоча менш потужний, ніж у конкурентів, проте ефективний для більшості проектів.
- Вартість. Повністю безкоштовний та з відкритим вихідним кодом.
- Спільнота та документація. Іноді менша спільнота порівняно з Unity та Unreal, але все ще активна. Документація також вдосталь.
- Кросплатформеність. Підтримує різні платформи, але може вимагати деякої додаткової роботи для налаштування.

Отже, Unity – це потужний та гнучкий ігровий двигун з великими можливостями та активною спільнотою. Зокрема для початківців та розробників, які орієнтуються на кросплатформеність, Unity може бути відмінним вибором, надаючи ефективний та швидкий процес розробки.

3.1.3 Вибір середовище написання коду

1. Visual Studio – це інтегроване середовище розробки (IDE), розроблене компанією Microsoft для підтримки різних мов програмування, створення програмного забезпечення, веб-додатків, мобільних додатків та ігор. Visual Studio надає розробникам широкий набір інструментів, включаючи редактор коду, відладчик, дизайнер інтерфейсу, систему контролю версій, та інші корисні функції [25].

- Мови програмування. Підтримує широкий спектр мов, зокрема C#, C++, Python, Java і багато інших.
- Інтеграція. Інтегровано з широким спектром інструментів і технологій від Microsoft, таких як Azure, .NET, і багато інших. Має вбудовану підтримку систем контролю версій.
- Продуктивність. Має високу продуктивність та широкий функціонал, включаючи розумне завершення коду, налагодження, профілювання і багато іншого.
- Розширення. Підтримує розширення, що дозволяє розробникам розширювати функціональність IDE через додаткові плагіни.

Eclipse – це інтегроване середовище розробки (IDE), що використовується для програмування в різних мовах, таких як Java, C++, Python, PHP та інші. Eclipse було розроблено фондом Eclipse, неприбутковою організацією, і стало відкритим програмним забезпеченням. Основними рисами середовища розробки Eclipse є модульність, розширюваність та можливість роботи з різними типами проектів [26]:

- Мови програмування. Підтримує різні мови, включаючи Java, C++, Python, PHP, та інші.
- Інтеграція. Є гнучким інструментом, але потребує ручного налаштування для підключення різних інструментів.
- Продуктивність. Може вимагати додаткових налаштувань для досягнення високої продуктивності, іноді менш інтуїтивний.
- Розширення. Має систему розширень, але іноді менше багатофункціональний порівняно з Visual Studio.

IntelliJ IDEA – IntelliJ IDEA - це інтегроване середовище розробки (IDE), яке використовується для програмування на різних мовах, зокрема на Java, Kotlin, Groovy, Scala та інших. IntelliJ IDEA розроблено компанією JetBrains і визнано своєю потужною та ефективною платформою для створення програмного забезпечення [27]:

- Мови програмування. Переважно спрямований на Java, але також підтримує Kotlin, Groovy, Scala, та інші.
- Інтеграція. Відмінно інтегрується з різними технологіями, має вбудовану підтримку інструментів розробки.
- Продуктивність. Відомий своєю високою продуктивністю та інтелектуальним завершенням коду.
- Розширення. Має розширювану архітектуру і дозволяє використовувати плагіни для розширення функціоналу.

Отже, Visual Studio видається привабливим вибором завдяки своїй високій продуктивності, масштабності, інтеграції з технологіями Microsoft і багатомовності.

3.1.4 Висновок

Отже, було розглянуто декілька мов програмування, ігрових рушіїв та середовищ розробки. Серед яких було обрано мову програмування C#, ігровий рушій Uniity та середовище розробки Visual Studio.

3.2 Інструкція користувача

Гра монополія складається з трьох сцен: «Головне меню», «Меню вибору гравців», «Гра». Розглянемо інструкцію для кожної сцени.

Інструкція для сцени «Головне меню» (рис 3.4).

1. Почати гру (рисунок 3.4 – пункт 1) – користувач переходить зі сцени «Головне меню» до сцени «Меню вибору гравців».
2. Відкрити меню завантаження (рисунок 3.4 – пункт 2) – відкривається меню завантаження.
3. Відкрити меню налаштувань (рисунок 3.4 – пункт 3) – відкривається меню налаштування.
4. Відкрити меню про розробника (рисунок 3.4 – пункт 4) – відкривається меню завантаження.
5. Вийти з гри (рисунок 3.4 – пункт 5) – користувач виходить з гри.
6. Меню відображень – в цьому вікні відображаються різноманітні меню, які відкривають кнопки «Завантаження», «Налаштування», «Про розробника».

В меню завантаження користувач може завантажити гру, яка була збережена в минулому. Меню завантаження зображене на рисунку 3.5.

У вікні можна побачити чотири панелі в яких будуть зображення зі збереженою грою (рисунок 3.5 – пункт 1). Якщо збережень буде більше чотирьох можна перегорнути на наступну сторінку за допомогою кнопки вправо (рисунок 3.5 – пункт 3), щоб повернутись назад потрібно натиснути кнопку вліво (рисунок 3.5 – пункт 2).

При натисканні на збереження воно стає активно і можна обрати дві дії: видалити збереження за допомогою кнопки «Delete» (рисунок 3.5 – пункт 4), після чого збереження буде видалено; завантажити збереження за допомогою кнопки

«Load» (рисунок 3.5 – пункт 5), після чого користувач перейде зі сцени «Головне меню» до сцени «Гра» зі збереженими даними.

Рисунок 3.4 – Головне меню

Для того, щоб закрити меню завантаження потрібно ще раз натиснути на кнопку «Load» або натиснути на дві інші кнопки, які відкривають свої меню.

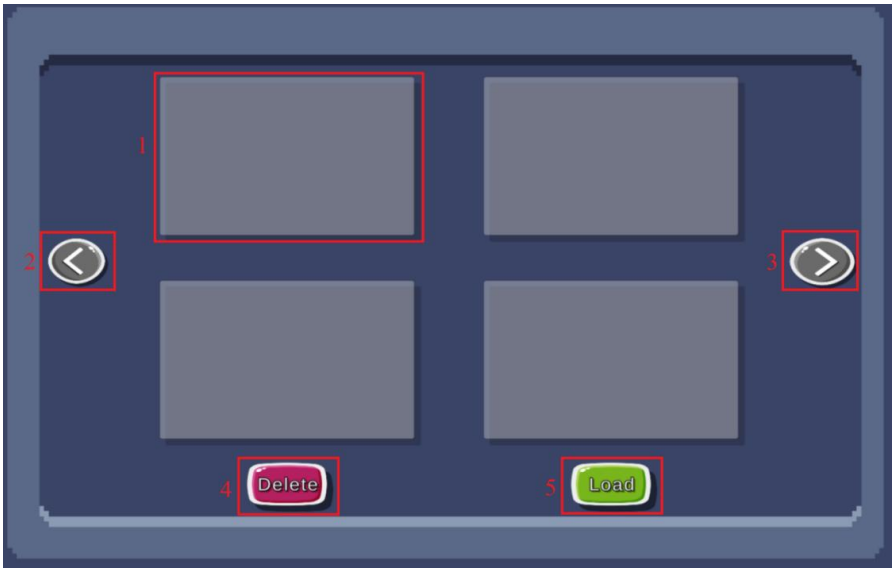


Рисунок 3.5 – Меню завантаження

В меню налаштування користувач може змінити деякі налаштування

додатку, а саме:

1. Гучність. За допомогою ползунка користувач може регулювати гучність від 0 до 100.

2. Розширення. Користувач може змінити розширення гри, доступні всі найпопулярніші розширення.

3. Формат екрану. Користувач може змінити формат екрану, в наявності є такі формати, як: у вікні; у вікні без рамки; повний екран.

4. Мова. У гру доступен вибір двох мов: українська та англійська.

Меню налаштування зображене на рисунку 3.6.

Для того, щоб закрити меню завантаження потрібно ще раз натиснути на кнопку «Options» або натиснути на дві інші кнопки, які відкривають свої меню.

Рисунок 3.6 – Меню налаштування

В меню про розробника користувач може побачити інформацію про розробника. Меню про розробника зображене на рисунку 3.7.

Для того, щоб закрити меню завантаження потрібно ще раз натиснути на кнопку «About» або натиснути на дві інші кнопки, які відкривають свої меню.

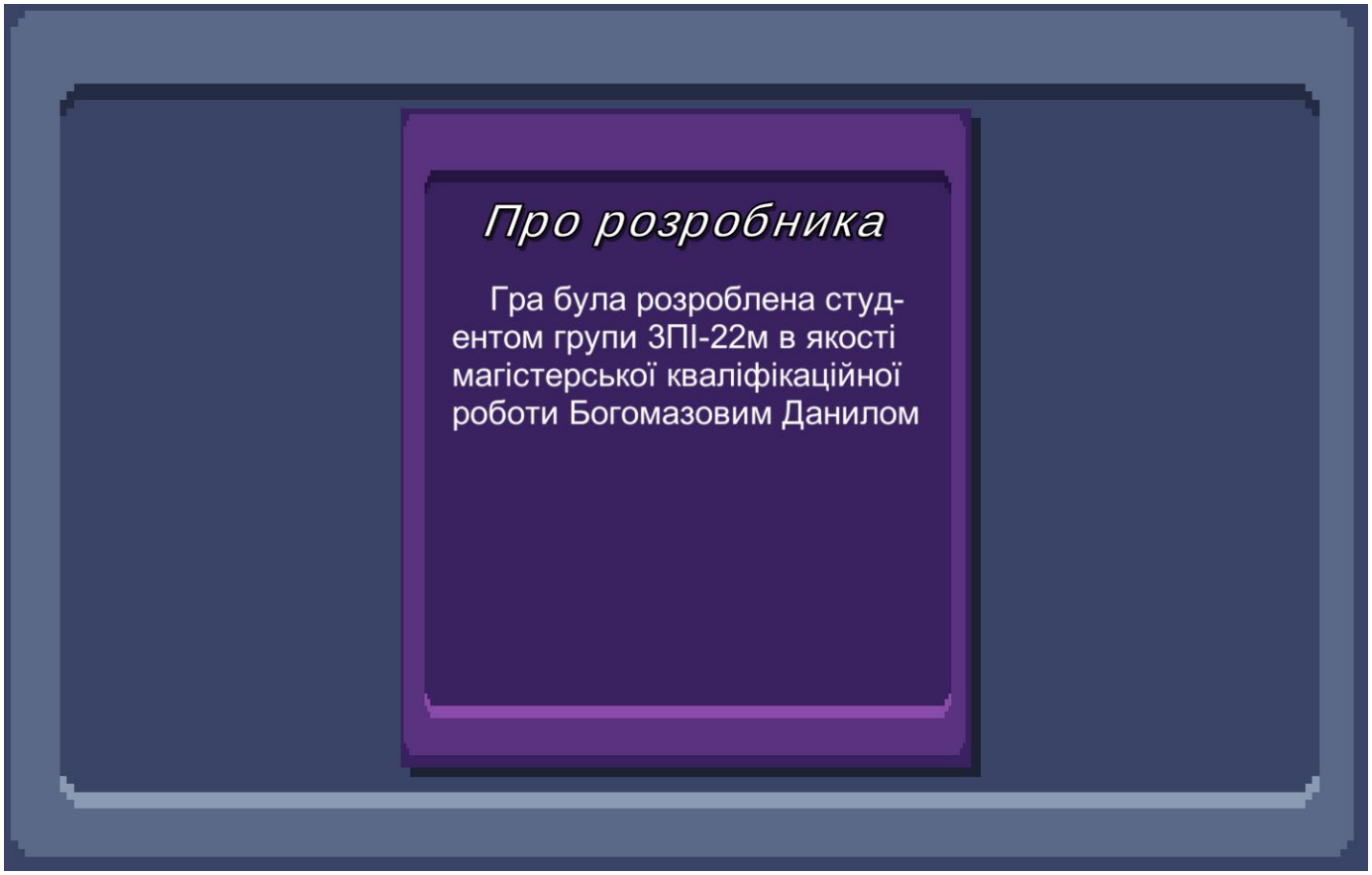


Рисунок 3.7 – Меню про розробника

Інструкція для сцени «Меню вибору гравців» (рис 3.8).

1. Обрати кількість гравців – для цього потрібно натиснути кнопку «Створити» (рисунок 3.8 – пункт 1). Максимальна кількість гравців – 8. Після натиснення кнопки з’являється панель з характеристиками (рисунок 3.8 – пункт 2):

1) Номер гравця – не можна змінити.

2) Портрет гравця – натиснувши можна обрати один з можливих, при умові, що такий портрет вже не було обрано.

3) Ім’я гравця – натиснувши на поле потрібно ввести бажане ім’я з клавіатури.

4) Колір гравця – натиснувши можна обрати один з можливих, при умові, що такий колір вже не було обрано.

5) Видалити гравця – при натисканні видаляє гравця.

6) Вибір статусу – можна обрати чи буде гравець ботом чи людиною, якщо зображена людина – гравець буде людиною, якщо комп'ютер – ботом.

2. Обрати кількість початкових коштів (рисунок 3.8 – пункт 3).

3. Обрати рівень складності (рисунок 3.8 – пункт 4).

4. Перейти безпосередньо до гри натиснувши кнопку «Почати гру» (рисунок 3.8 – пункт 5). Сцена «Меню вибору гравців» закривається і відкривається сцена «Гра».

5. Вийти з гри натиснувши на червоний хрест в правому верхньому кутку екрана (рисунок 3.8 – пункт 6)..

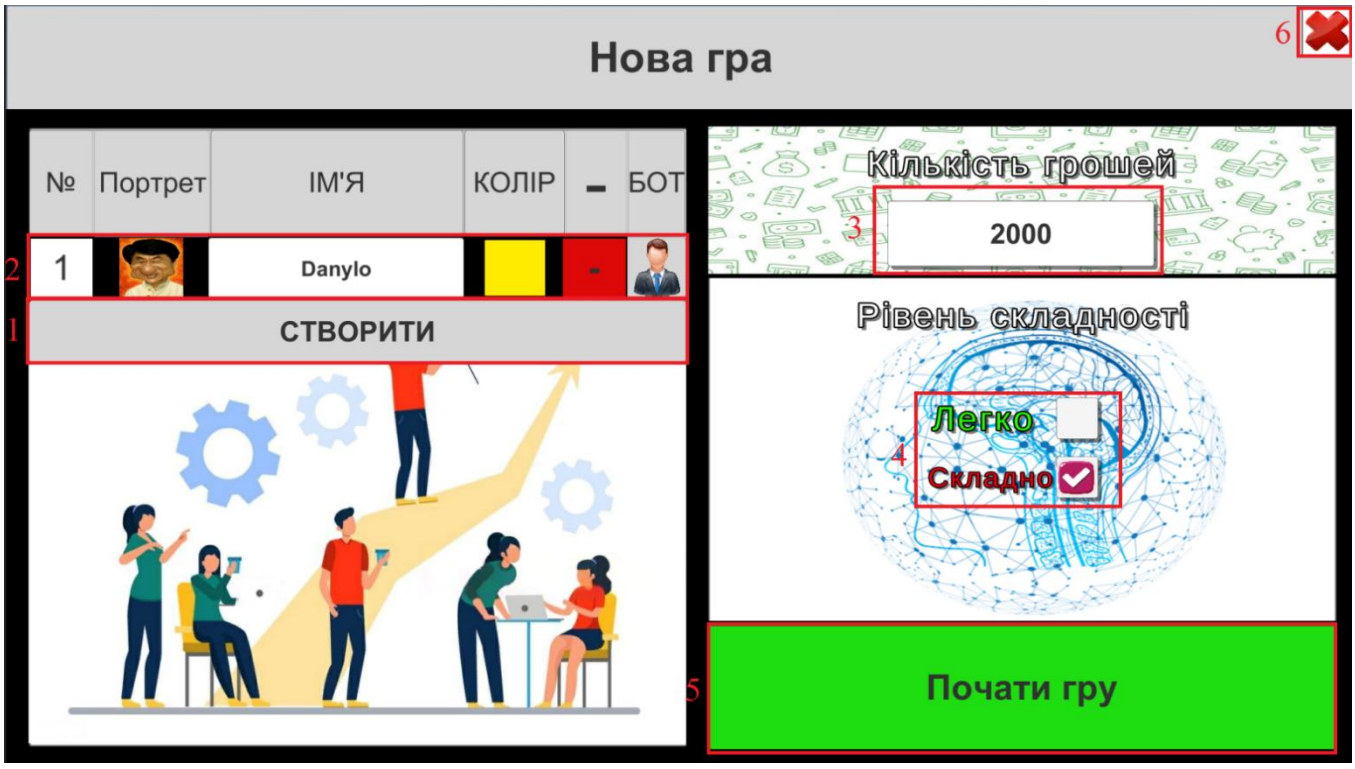


Рисунок 3.8 – Меню вибору гравців

Інструкція сцени «Гра» (рис. 3.9).

1. Для того, щоб брати участь у грі, необхідно зачекати свій черговий хід. Інформацію про це можна отримати через ліву панель (див. рисунок 3.9 – пункт 1). Зелене підсвічування вашого профілю вказує на вашу можливість розпочати хід, у той час як червоне світіння сигналізує про хід іншого гравця. Послідовність дій

відбувається зверху донизу, коли останній (найнижчий) гравець завершує свій хід (у грі це умовно називається «Кінець кола»), черга знову переходить до першого, і так повторюється до завершення гри.

Рисунок 3.9 – Гра

2. Для того, щоб розпочати свій хід, слід натискати кнопку "Кинути кубики" на правій панелі (див. рисунок 3.9 – пункт 2). Після відтворення анімації гравець переміщується на кількість секторів, яка випала на кубиках.

3. Після того, як гравець потрапить на один із секторів, відкривається відповідне меню сектору на правій панелі. Для кожного сектору передбачене власне меню (див. рисунок 3.9 – пункт 3). Сектори можна класифікувати за 8 типами:

1) "Старт": Це початковий сектор гри. Він не має власного меню, але якщо гравець потрапив на цей сектор після кидка кубиків або пройшов повз нього, йому нараховуються певні кошти.

2) "Поле"/"Бізнес": Це сектори, які можуть бути придбані гравцем. Якщо "Поле"/"Бізнес" вільний, гравець може купити його, натиснувши кнопку "Купити

Сектор" (див. рис. 3.10). Якщо "Поле"/"Бізнес" вже належить комусь іншому, гравець зобов'язаний заплатити зазначену у меню сектора суму коштів, натискавши кнопку "Заплатити" (див. рис. 3.11). Якщо гравець не може внести відповідну суму, він програє.

3)

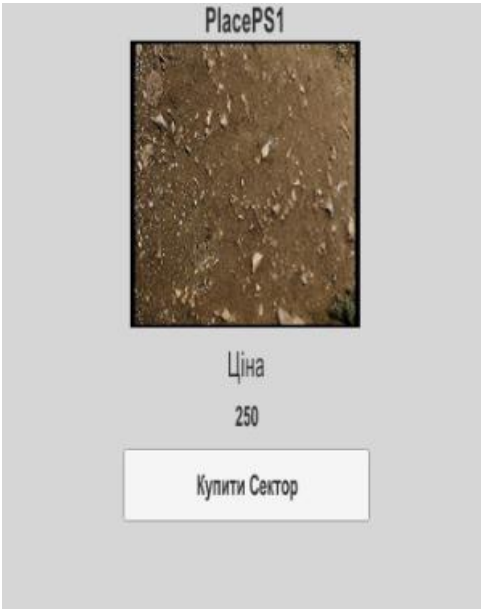


Рисунок 3.10 – Меню покупки

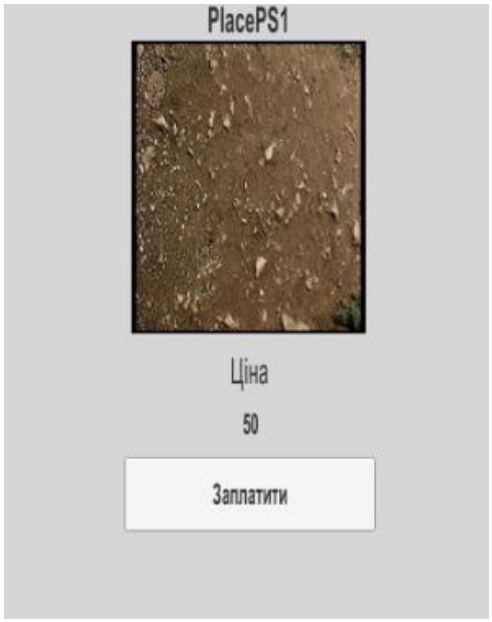


Рисунок 3.11 – Меню купленого сектора

4) «Бонус» (рис. 3.12). Потрапивши на цей сектор гравець отримує можливість обрати один з трьох типів бонусів на свій вибір.

Зелений бонус. В ньому трапляються карти, які допомогають гравцю економічно. Там може бути знижка на район або дім, єдиноразова винагорода або можливість взяти кредит на декілька днів більше тощо.

Червоний бонус. В червоних бонусах трапляються карти, які дозволяють якось послаблювати супротивників, наприклад знизити комусь капітал, позбавити хода, знищити будинок тощо.

Жовтий бонус. Завдяки жовтим картам можна уникнути деяких випадковостей у грі, наприклад погода не впливає на вас цього раунду, ви можете переміститись до 3 клітинок в обрану сторону, у вас є можливість вийти з в’язниці тощо.

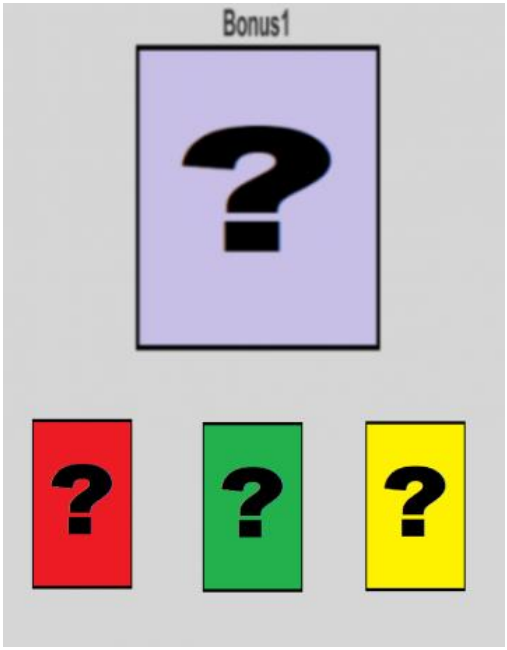


Рисунок 3.12 – Меню бонуса

5) "Зупинка": Цей сектор дозволяє гравцеві переміститися до іншої зупинки за допомогою меню, при умові, що у нього є достатня кількість коштів, а маршрут до необхідної зупинки відкритий (див. рис. 3.13). На початку гри всі маршрути зачинені, за винятком північного. Після двох "Кінців кола" відкривається східна зупинка, після наступних двох – південний, і після ще двох –

західний (під час такого переходу не нараховуються кошти за перехід через сектор "Старт").

Рисунок 3.13 – Меню зупинки

6) "В’язниця": Якщо гравець потрапив на цей сектор після кидка кубиків, то нічого не відбувається, і в такому випадку взаємодія з цим сектором недоступна. Якщо гравець потрапив до в’язниці через події, такі як невиплата кредиту або три подряд дублі на кубиках (дублем вважається, коли обидва кубики показують однакове число), тоді відкривається меню цього сектору (див. рис. 3.14). Гравець не може рухатися, поки не відсидить у «В’язниці» три ходи або не сплатить визначену суму коштів. Також у гравця є можливість викинути три дублі поспіль, що автоматично виведе його з в’язниці. Проте, якщо це не вдасться хоча б один раз, хід гравця закінчується.

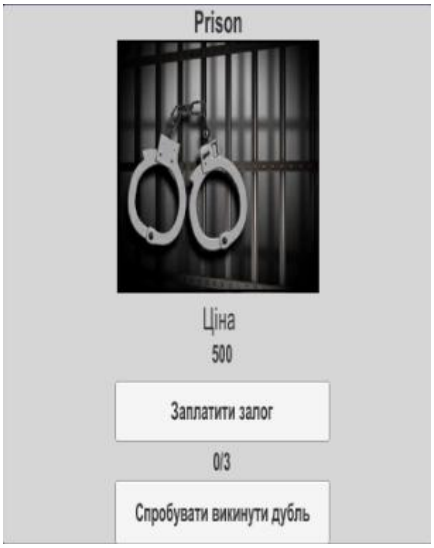


Рисунок 3.14 – Меню в’язниці

7) "Казино": На цьому секторі гравець має зробити важливий вибір – ризикувати на удачу чи ні. Якщо гравець вірить у свої здібності, він може обрати певну суму коштів (не перевищуючи загальної суми у гравця) і натискати кнопку "Зіграти" (див. рис. 3.15). У випадку успіху сума подвоїться, а якщо ні, кошти будуть втрачені.

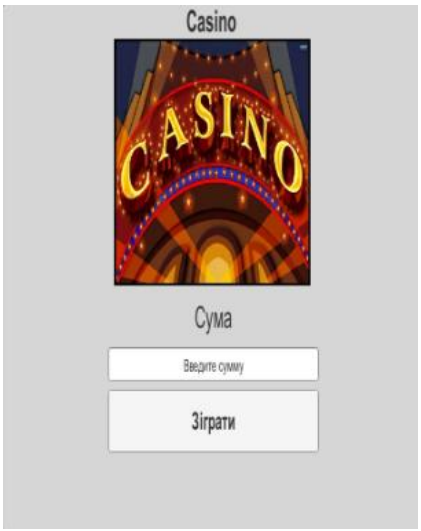


Рисунок 3.15 – Меню казино

8) "Таксі": У цьому секторі гравець може вибрати будь-який сектор на карті (за винятком сектору "Таксі") і, сплативши відповідну суму коштів, перенестися до цього сектору (при цьому не нараховуються кошти за перехід через сектор "Старт") (див. рис. 3.16).

Рисунок 3.16 – Меню таксі

4. Якщо гравець не перебуває у в'язниці, він може взаємодіяти з центральним меню гри (див. рисунок 3.9 – пункт 4). Центральне меню включає наступні секції:

1) "Банк" (див. рис. 3.17): Гравець може взяти кредит на певну суму грошей, за умови, що кредит ще не був взятий. Для цього він повинен ввести необхідну суму коштів у відповідному меню та натискати кнопку "Взяти кредит" (див. рис. 3.17 – пункт 1). В банку вказується відсоток, який гравець повинен буде виплатити (див. рис. 3.17 – пункт 2). Після цього гравець отримає обрану суму коштів, і в банку з'являться три рядки із сумами, які гравець повинен виплатити (див. рис. 3.17 – пункт 3). Під час ходу, на якому гравець взяв кредит, і на наступному, сума дорівнює першому дню, на другому ході після взяття дорівнює другому дню, і на третій хід буде дорівнювати третьому дню. Щоб виплатити кредит, гравець повинен натискати кнопку "Виплатити кредит" (див. рис. 3.17 – пункт 4). Якщо на третій день гравець не виплатить кредит і завершить хід, йому заберуть ті кошти, які він повинен був виплатити. Якщо у гравця недостатньо коштів, його власність буде продана, поки не буде набрана необхідна сума, після чого гравець потрапить у в’язницю. Якщо навіть після продажу майна гравця не вистачає коштів, він автоматично програє.

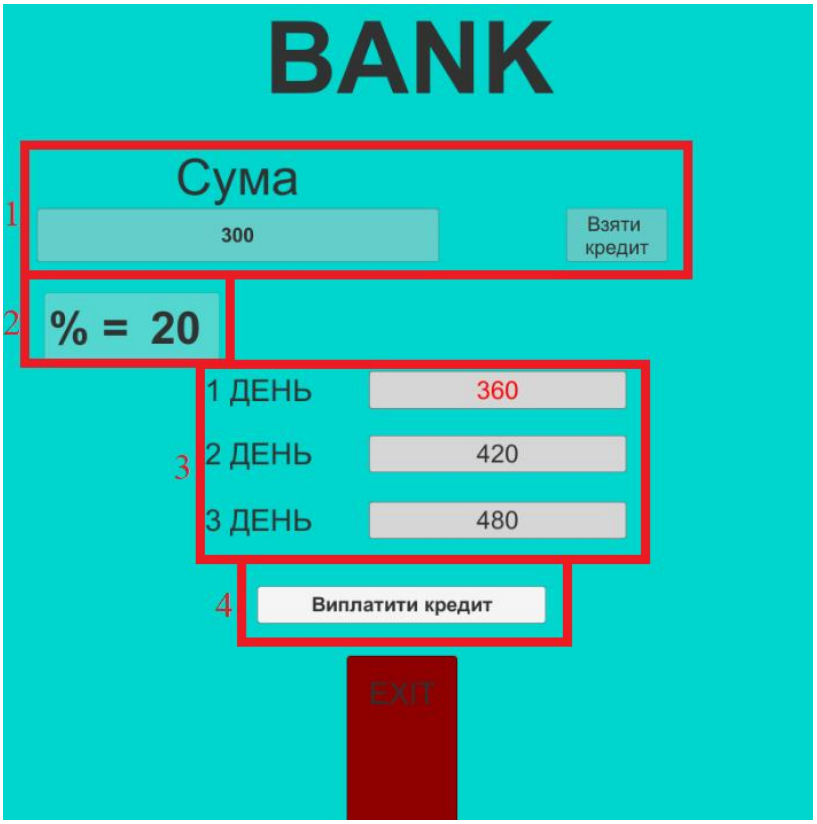


Рисунок 3.17 – Меню банку

2) "Обмін" (див. рис. 3.18): Під час обміну активний гравець може вибрати будь-якого іншого гравця та розпочати з ним торгівлю. Сектори та гроші активного гравця розташовані справа, а обраного гравця — зліва. Щоб вибрати гравця, необхідно натискати на випадаюче меню та обирати потрібного (див. рис. 3.18 – пункт 1). Для вибору сектора для обміну слід натискати на його зображення; після цього на ньому з'явиться зелений маркер (див. рис. 3.18 – пункт 2), для відміни слід ще раз натискати на зображення. Для вибору суми грошей, слід натискати на відповідне поле зі свого боку та вписати необхідну кількість коштів (див. рис. 3.18 – пункт 3). Після того, як гравці узгодили свої умови, кожен з них повинен натискати кнопку "Готовності" (див. рис. 3.18 – пункт 4). Коли обидва гравці готові, кнопка "Обмін" стає активною (див. рис. 3.18 – пункт 5). При натисканні на неї обрані сектори переходять до іншого власника, а гроші кожного знімаються та додаються відповідно до узгоджених умов.

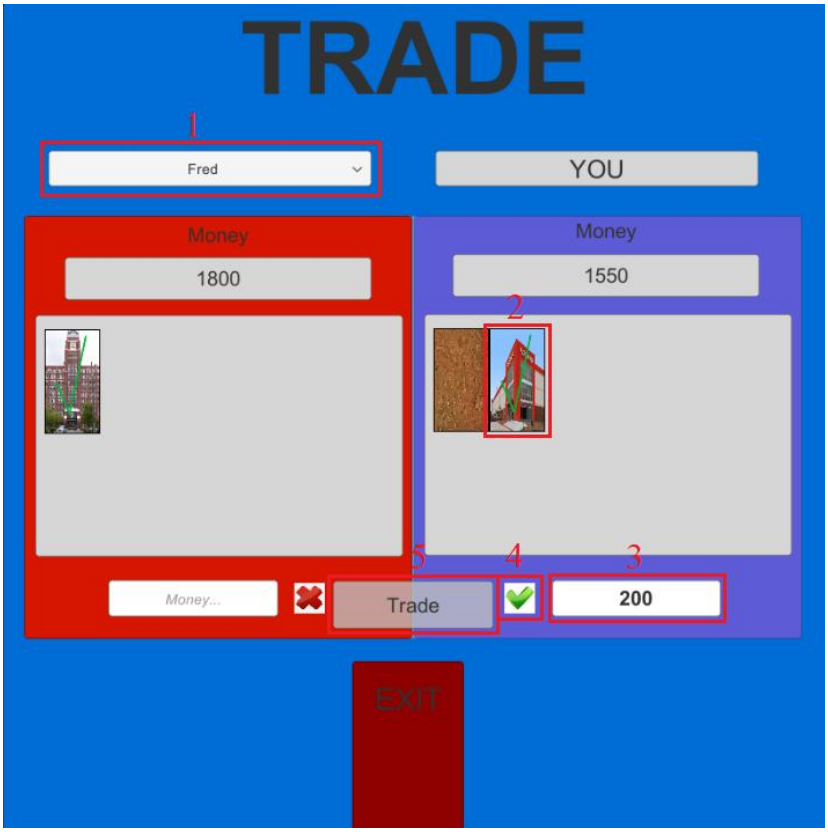


Рисунок 3.18 – Меню обміну

3) "Здатися" (див. рис. 3.19): Заходячи сюди, гравець може завершити гру і визнати свою здачу.

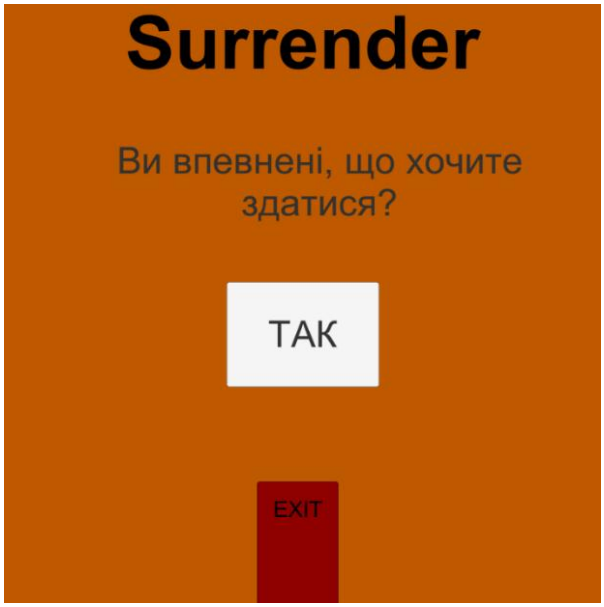


Рисунок 3.19 – Меню здатися

4) "Кубик надії" (див. рис. 3.20): В цьому розділі гравець може випробувати удачу і кинути "Кубик надії". Ймовірність перемоги залежить від

відставання гравця від інших, але завжди залишається дуже низькою. Після того, як гравець кидає кубик, є лише два можливих результати: зображення зупиняється на черепі, і гравець автоматично програє, або зображення зупиняється на "$", і гравець отримує певну кількість грошей.

Рисунок 3.20 – Меню кубику надії

5) "Продаж" (див. рис. 3.21): Якщо гравець бажає продати придбані сектори "Поле" або "Бізнес", він може зробити це в цьому розділі. Натисканням на зображення відповідного сектора на ньому з'являється маркер (див. рис. 3.21 – пункт 1), після чого обрано інший сектор неможливо, не забравши маркер з попереднього. Далі, для здійснення продажу необхідно натискати кнопку "Продати" (див. рис. 3.21 – пункт 2); сектор буде вилучено у гравця, і він отримає певну кількість коштів.

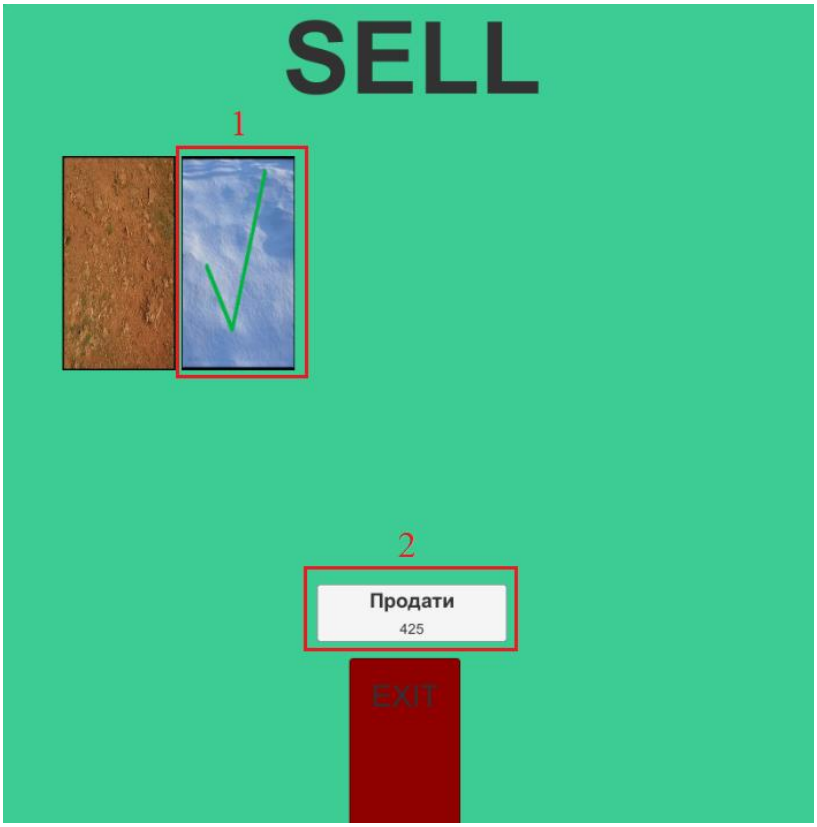


Рисунок 3.21 – Меню продажу

6) "Дома" (див. рис. 3.22): У грі є можливість збільшувати цінність сектору "Поле" шляхом будівництва домів на цьому секторі. Для того, щоб мати можливість будувати дома на секторі "Поле", гравець повинен мати всі сектори з його "Району". "Район" (див. рис. 3.23) — це умовне згрупування трьох секторів "Поле" і одного "Бізнес" на відповідній ділянці. Якщо у гравця є "Район", після натискання кнопки "Дома" з'явиться панель із назвою району та секторами в ньому (див. рис. 3.22 – пункт 1).

Кожен сектор розділений на 6 рівних частин. Спочатку на всіх частинах зображена цифра (див. рис. 3.22 – пункт 2) — це вартість будівництва дома на цій частині. Після натискання на одну з частин, з'явиться зелений дім (див. рис. 3.22 – пункт 3), і після повторного натискання дім зникає. Після вибору частини, на якій буде розташований дім, гравець повинен натискати кнопку "Почати будівництво" (див. рис. 3.22 – пункт 4); дім перестає бути зеленим і стає кольоровим (див. рис. 3.22 – пункт 5), і гравцю знімаються гроші. У грі сектор візуально також зміниться.

Якщо гравець бажає продати дім, він може натискати на дім, і той стане червоним (див. рис. 3.22 – пункт 6). Щоб продати, треба натискати кнопку "Продати дім" (див. рис. 3.22 – пункт 7); після цього з сектору зникне дім, і знову з'являться цифри, а гравцю буде видана певна кількість грошей.

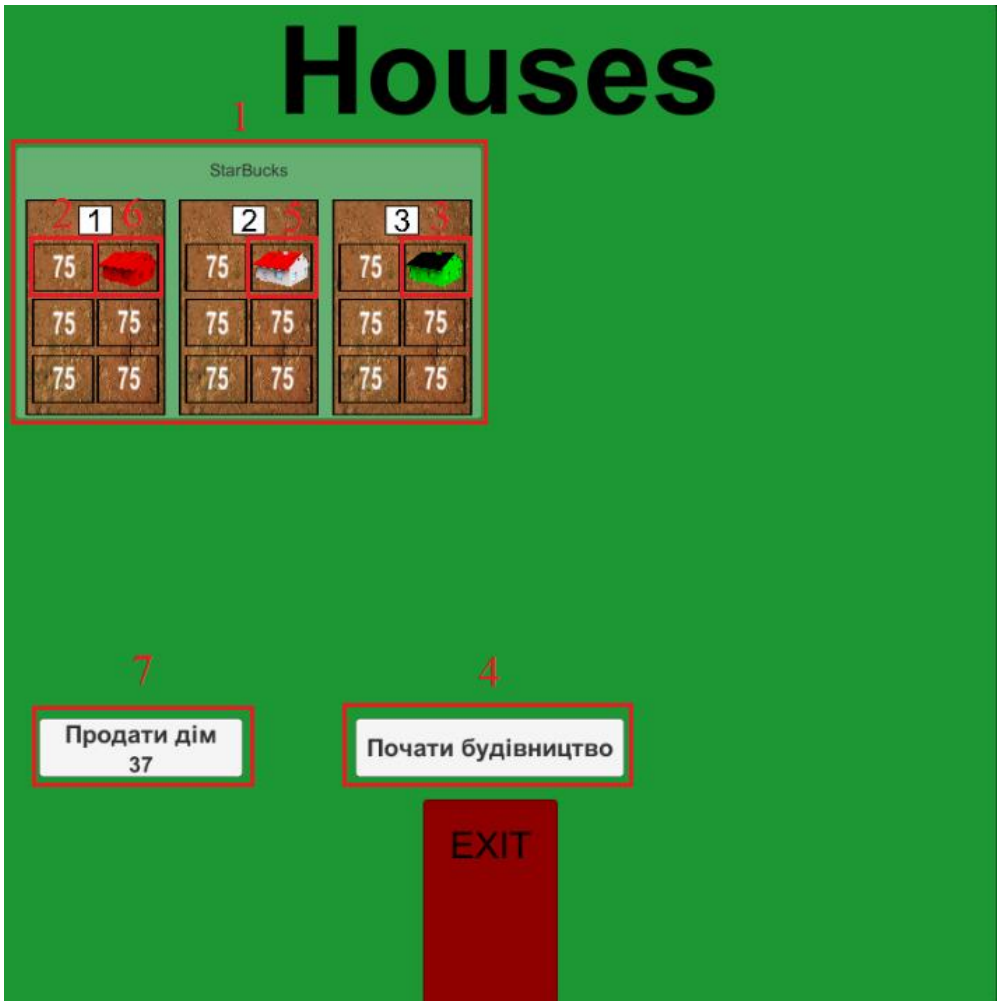


Рисунок 3.22 – Меню дома

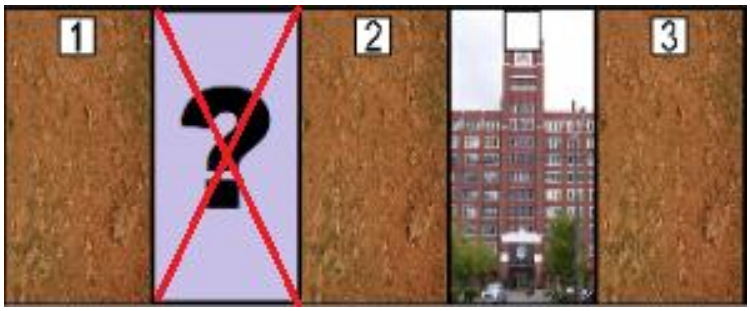


Рисунок 3.23 – Район

7) "Бізнес" (див. рис. 3.24): Якщо гравець володіє трьома секторами "Поле" з одного "Району", він може зайти в цю секцію і, за умови, що ніхто не володіє потрібним бізнесом, купити потрібний бізнес з будь-якої частини карти. Якщо гравець може купити бізнес, він стає кольоровим (див. рис. 3.24 – пункт 1), в іншому випадку бізнес залишається сірий (див. рис. 3.24 – пункт 2).

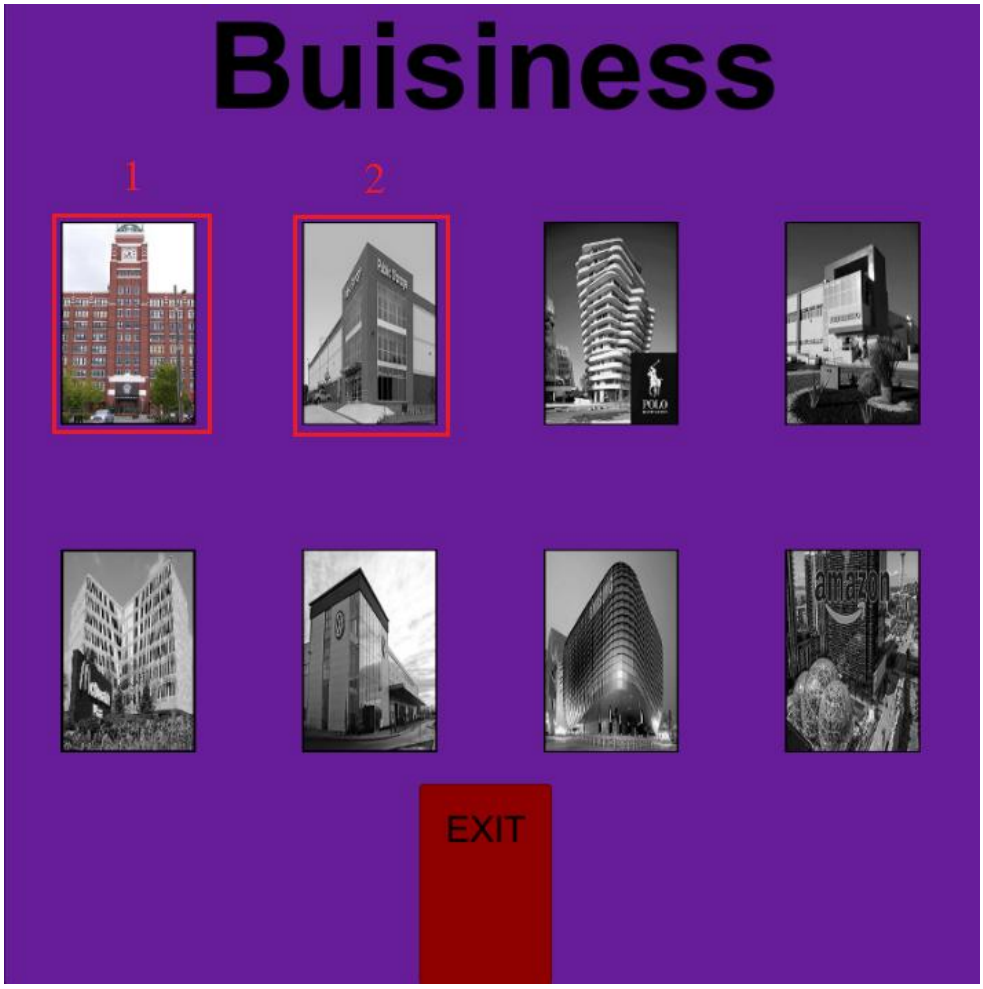


Рисунок 3.24 – Меню бізнес

5. Коли гравець завершує всі свої дії і готовий завершити свій хід, йому слід натискати кнопку "Завершити хід" на правій панелі (див. рисунок 3.9 – пункт 5). Після цього профіль гравця стає червоним, і хід передається наступному учаснику.

У грі існує механіка "Погоди", яка змінюється після "Кінця кола". Для отримання інформації про погоду гравець може скористатися центральним меню

(див. рисунок 3.9 – пункт 6). Залежно від погоди відбуваються зміни в переміщенні гравців. Гра реалізує п'ять типів погоди, а саме:

- Ясно – гравці залишаються без змін.
- Вітер – від суми очок, випавших на кубиках гравців, віднімається 2.
- Сонце – існує можливість того, що гравець вибере неочікуваний напрямок.
- Блискавка – існує ризик пройти менше, ніж вказує сума очок на кубиках.
- Дощ – враховується лише шлях, вказаний одним з кубиків.

3.3 Висновки

В данному розділі виконано аналіз різноманітних ігрових рушіїв, і в результаті вибір зупинився на Unity. Також було проведено порівняльний аналіз різних мов програмування, і з цього аналізу виявилось, що найбільш підходящою є мова C#. У контексті розробки обраної гри було детально ознайомлено з можливостями інтегрованого середовища розробки Visual Studio 2019.

Окрім того, в цьому розділі розроблено інструкцію користувача, яка подробно розглядає всі аспекти гри, щоб надати гравцям повне розуміння її функціоналу.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Вибір методики тестування

Тестування є ключовим етапом у процесі розробки програмного забезпечення і визначається багатьма корисними аспектами. Ось деякі з них:

- Виявлення помилок. Тестування дозволяє виявляти та виправляти помилки та дефекти в програмному коді на ранніх етапах розробки, перед тим як програма потрапить в руки користувачів.
- Забезпечення якості продукту. Тестування гарантує, що продукт відповідає вимогам та специфікаціям, і вирішується якісно та надійно.
- Вдосконалення надійності. Тестування допомагає перевірити стійкість програми до різних умов та навантажень, підвищуючи надійність системи.
- Зменшення ризику виходу на ринок. Ретельне тестування допомагає зменшити ризик виникнення серйозних проблем чи помилок при виході продукту на ринок.
- Підтримка рефакторингу. Тестові набори дозволяють впевнено внесені зміни у код, забезпечуючи, що вони не порушать існуючої функціональності.
- Економія часу та ресурсів. Автоматизовані тести можуть швидко та ефективно виявляти помилки, зменшуючи кількість часу та ресурсів, витрачених на вручне тестування.
- Покращення здатності до масштабування. Тестування дозволяє визначити, наскільки добре програма адаптується до зростаючого обсягу даних чи користувачів.
- Впровадження неперервної інтеграції та постійного впровадження. Тестування є необхідною складовою для успішної реалізації практик неперервної інтеграції та постійного впровадження (CI/CD), що сприяє швидкому та стабільному впровадженню змін.

Тестування є критично важливим елементом у забезпеченні високої якості програмного забезпечення і впевненості у його працездатності для кінцевих користувачів.

Для вибору правильного методу тестування потрібно розглянути і порівняти декілька з них.

1. Інтеграційні тести – це вид тестування програмного забезпечення, який спрямований на перевірку взаємодії між різними компонентами або модулями системи. Ці тести перевіряють, як добре компоненти взаємодіють один з одним під час їх об'єднання в єдиний функціональний блок [28].

Плюси:

- Виявлення дефектів взаємодії. Інтеграційні тести дозволяють виявляти помилки та неправильну взаємодію між компонентами програми.
- Перевірка коректності об'єднання. Тести допомагають переконатися, що при об'єднанні компонентів система працює правильно та ефективно.
- Вдосконалення якості коду. Розробники можуть виявити та виправити помилки, які можуть виникнути при інтеграції коду різних частин системи.
- Сприяє стабільності системи. Інтеграційне тестування допомагає забезпечити стабільність системи, особливо при великих змінах в програмному коді.

Мінуси:

- Складність налаштування. Налаштування інтеграційних тестів може бути трудомістким завданням через потрібність роботи з різними компонентами системи.
- Часові та ресурсні витрати. Виконання інтеграційних тестів може займати багато часу, особливо великих проектах.
- Підтримка. Підтримка інтеграційних тестів може бути важливим завданням, оскільки система може змінюватися, і тести повинні адаптуватися до цих змін.

2. Системні тести – це вид тестування програмного забезпечення, який спрямований на перевірку всієї системи як єдиної інтегрованої одиниці. Ці тести

оцінюють, чи відповідає весь продукт заданим вимогам та специфікаціям перед його випуском в експлуатацію [29].

Плюси:

- Повність відображення функціональності. Системні тести оцінюють весь функціонал системи, надаючи повний вигляд на її роботу в цілому.
- Виявлення проблем інтеграції. Тести допомагають виявляти проблеми взаємодії між різними компонентами системи.
- Визначення відповідності вимогам. Системні тести перевіряють, чи відповідає система визначеним вимогам та специфікаціям.
- Оцінка продуктивності та навантаження. Можливість визначити, як система працює при різних навантаженнях та в певних умовах.

Мінуси:

- Великі витрати часу. Виконання системних тестів може займати значно більше часу порівняно з іншими видами тестів.
- Складність виявлення точного місця помилки. Якщо система не працює правильно, може бути важко визначити конкретну причину помилки через велику кількість компонентів.
- Залежність від конфігурації. Результати тестів можуть змінюватися в залежності від конфігурації системи, що може зробити важчим розуміння реальної працездатності.

3. Автоматизовані тести – це використання програмних скриптів та інструментів для виконання тестів на програмному забезпеченні. Ці тести здатні автоматично виконувати, порівнювати та аналізувати результати, спрощуючи процес тестування [30].

Плюси:

- Швидкість виконання. Автоматизовані тести виконуються швидше, ніж ручні, що дозволяє їх використання в циклі розробки та постійному впровадженні.
- Повторюваність. Результати автоматизованих тестів однакові при кожному їх виконанні, що дозволяє виявляти неправильності та забезпечує

надійність тестування.

- Ефективність ресурсів. Зменшення необхідності у великій кількості людських ресурсів для тестування, особливо у великих проектах.
- Раннє виявлення помилок. Автоматизовані тести дозволяють виявляти помилки на ранніх етапах розробки, що полегшує їх виправлення.

Мінуси:

- Вартість впровадження. Початкова вартість створення автоматизованих тестів може бути високою.
- Складність обслуговування. Підтримка та обслуговування автоматизованих тестів може вимагати додаткових витрат.
- Неефективність для деяких видів тестів. Деякі види тестів, зокрема тести на користувацький інтерфейс, можуть бути складно автоматизувати та вимагати ручного тестування.
- Залежність від змін у програмі. Якщо програма зазнає частих змін, автоматизовані тести можуть вимагати постійного оновлення.

4. Юніт-тести – це вид тестування програмного забезпечення, який спрямований на перевірку окремих компонентів програми для визначення їхньої коректної роботи. У контексті об'єктно-орієнтованого програмування юнітами можуть бути класи, методи чи функції [31].

Плюси :

- Ізольованість. Тести проводяться ізольовано від інших частин системи, що дозволяє ефективно виявляти та виправляти помилки в окремих компонентах.
- Швидкість виконання. Юніт-тести виконуються дуже швидко, що дозволяє їх виконання в кілька секунд або менше, надаючи миттєвий звіт про стан коду.
- Автоматизованість. Юніт-тести легко автоматизовані, що дозволяє їх виконання в ході розробки без значної ручної праці.
- Полегшення рефакторингу. Юніт-тести допомагають забезпечити, що зміни в коді не призводять до введення нових помилок.

Мінуси:

- Не відображення взаємодії. Юніт-тести не виявлять проблем взаємодії між компонентами або класами, які можуть виникнути при їх об'єднанні в єдиний функціональний блок.
- Неабсолютна повнота. Неможливо покрити всі можливі шляхи в коді, тому покриття може бути неповним.
- Витрати на обслуговування. Підтримка та обслуговування тестів може вимагати часу та зусиль, особливо при змінах у коді.
- Можливість вигляду тестів як завдання відволікання. Іноді розробники можуть вважати написання тестів завданням, відволіканням від основної роботи.

Зваживши плюси і мінуси різноманітних типів тестування було обрано юніт-тестування, оскільки вони найбільш вдало підходять для невеличкої гри накшталт розроблюваного додатку. Окрім того, ігровий рушій Unity має в своїх інструментах вбудовану можливість юніт-тестування.

4.2 Тестування програмного забезпечення

Ігровий рушій Unity має вбудовану функцію юніт-тестів, який реалізується за допомогою вікна Test Runner. Він дозволяє створити папку з тестами, скрипти в якій будуть відокремлені від інших скриптів, щоб не впливати на основний код програмного додатку [32].

Основна логіка у грі відбувається з гравцем, саме тому тестування буде саме основних методів гравця, а саме Buy() – покупка сектору, Move() – перехід на інший сектор, LostPlace() – втрата сектору та Bankrupt() – поразка гравця.

Окрім того було протестовано функції центрального меню, а саме Trade(), Credit(), BuyHouse(), SellHouse(), BusinessBuy().

Після того, як тести були написані, Unity автоматично перевіряє їх в модулі Test Runner. Якщо горить зелена галочка, то тест був пройдений. Результат тестування зображений на рисунку 4.1.

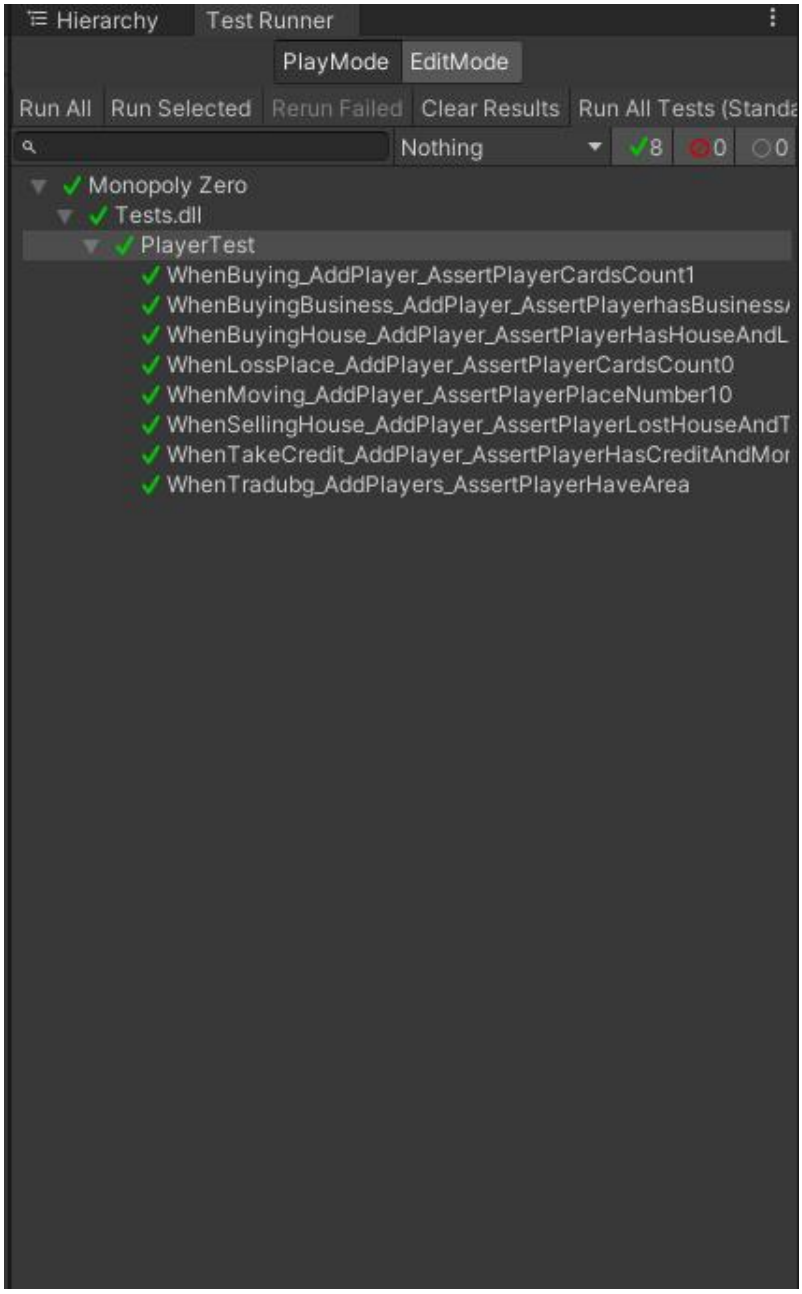


Рисунок 4.1 – Результат роботи юніт-тестів

4.3 Висновки

У цьому розділі було порівняно декілька методик тестування, серед яких було обрано юніт-тестування, оскільки воно більш підходить для розроблюваного програмного додатку та більш зручне у використанні за допомогою вбудованих функція ігрового рушія Unity.

Також було описано створені юніт-тести для основних функцій гравця в програмному додатку.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [33].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	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 працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
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					

Продовження таблиці 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

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 і з 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
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

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	3	4
2. Ринкові переваги (наявність аналогів)	3	3	4
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	4	4
10. Практична здійсненність (необхідність нових матеріалів)	3	4	3
11. Практична здійсненність (термін реалізації)	3	3	3
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	38	40	41
Середньоарифметична сума балів $СБ_с$	39,7		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [33].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ ,розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»» становить 39,7 бала, що, відповідно до таблиці

5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [33]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \ t_i}{T_p}, \qquad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

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

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=24 дні.

$$З_о = 18500{,}00 \cdot 5 / 24 = 46250{,}00 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	18500	770,83	55	46250,00
Інженер-розробник програмного забезпечення	16000	666,67	55	37333,33
Науковий співробітник дослідження проблем програмного забезпечення	12500	520,83	10	5208,33
Всього				89479,17

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \ t_i, \tag{5.2}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \ K_i \ K_c}{T_p \ t_{зм}}, \tag{5.3}$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [33];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 24 дні;

$t_{зм}$ – тривалість зміни, год.

$С_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (24 \cdot 8) = 63{,}34$ грн.
$З_{р1} = 63{,}34 \cdot 6{,}00 = 380{,}02$ грн.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка електронно-обчислювального обладнання	6	2	1,1	63,34	380,02
Підготовка робочого місця дослідника	2,4	2	1,1	63,34	152,01
Інсталяція програмного забезпечення	2,2	5	1,7	97,88	215,34
Всього					747,36

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р)\ \frac{Н_{дод}}{100\%}, \qquad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 10%.

$З_{дод} = (89479{,}17 + 747{,}36) \cdot 10 / 100\% = 9022{,}65$грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод})\ \frac{Н_{зп}}{100\%} \qquad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$Зн = (89479{,}17 + 747{,}36 + 9022{,}65) \cdot 22 / 100\% = 21834{,}82$ грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \ \ Ц_j \ \ K_j - \sum_{j=1}^{n} B_j \ \ Ц_{вj}, \qquad (5.6)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1 = 2,00 \cdot 200,00 \cdot 1,1 - 0,000 \cdot 0,00 = 440,0$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір, уп	200	2	0	0	440
Папір для записів, уп	110	1	0	0	121
Органайзер офісний,шт	210	2	0	0	462
Канцелярське приладдя (набір офісного працівника),шт	175	2	0	0	385
Картридж для принтера	1100	1	0	0	1210
Диск оптичний NewLine CD-RW	15	3	0	0	49,5
Flesh-пам'ять Kingston 16 GB	130	1	0	0	143
Тека для паперів	82	2	0	0	180,4
Всього					3490,63

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \quad C_{пр.i} \quad K_i, \qquad (5.7)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10...1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 550000 \cdot 1 \cdot 1,1 = 60500$ грн.

Отримані результати зведемо до таблиці:

Таблиця 5.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Персональний комп'ютер ASUS S500SC-51140F0030 SFF Intel i5-11400F, 16GB, F512GB, (90PF02K2-M02E5016)	1	25000,00	27500
Комп'ютер персональний HP Pro 400-G9 SFF, Intel i5-12500, 8GB, F512GB, UMA, WiFi, кл+м, 3р (8N8V2AA)	1	30000,00	33000
Всього			60500

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \quad C_{прг.i} \quad K_i \ , \qquad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 5420,00 \cdot 1 \cdot 1,1 = 6070,4$ грн.

Отримані результати зведемо до таблиці:

Таблиця 5.8– Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows 11	1	5420	6070,4
Прикладний пакет Microsoft Office 2019	1	5230	5857,6
Система розробки Project Rider	1	5400	6048
Всього			17976

Прирозробці також використовувалось і безкоштовне програмне забезпечення Unity та Visual Studio.

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \frac{t_{вик}}{12}, \quad (5.9)$$

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;

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

$A_{обл} = (24370{,}00 \cdot 2) / (2 \cdot 12) = 2030{,}83$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер ASUS	27500	2	2	2291,67
Персональний комп'ютер HP	33000	2	2	2750,00
Робоче місце дослідника	7880	5	2	262,67
Оргтехніка	8675	4	2	361,46
ОС Windows 11	5450	2	2	454,17
Прикладний пакет Microsoft Office 2019	3795	2	2	316,25
Всього				6436,21

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi}\ t_i\ Ц_e\ K_{вni}}{\eta_i}, \qquad (5.10)$$

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

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,50$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,3 \cdot 452,0 \cdot 7,50 \cdot 0,95 / 0,97 = 996,03$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.10 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер ASUS	0,3	440	969,59
Персональний комп'ютер HP	0,3	440	969,59
Робоче місце дослідника	0,15	440	498,02
Оргтехніка	0,25	10	18,36
Всього			2442,33

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р)\ \frac{Н_{св}}{100\%}, \qquad (5.11)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = *(89479,17+ 747,36) · 20 / 100% = 18045,31 грн.*

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{сп} = (З_о + З_р) \frac{Н_{сп}}{100\%}, \quad (5.14)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$В_{сп}$ = *(89479,17+ 747,36) · 30 / 100% = 27067,96 грн.*

5.2.11 Інші витрати

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» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \frac{Н_{ів}}{100\%}, \quad (5.12)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = *(89479,17+ 747,36) · 50 / 100% = 45113,27 грн.*

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з

освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \frac{H_{нзв}}{100\%}, \quad (5.13)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 120%.

$B_{нзв}$ = (89479,17+ 747,36) · 120 / 100% = 108 271,84 грн.

Витрати на проведення науково-дослідної роботи на тему «Підвищення захищеності процесу розробки проектів на основі удосконаленої моделі розподілу секрету та розмежування доступу між учасниками проекту» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (5.14)$$

$B_{заг}$=89479,17+ 747,36+ 9022,65+21834,82044+3490,63+0,00 +60500+17976+ 6436,21 +2442,33+18045,31+27067,96+45113,27+108271,84= 410427,54 грн.

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

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.15)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,8.

ЗВ = 410427,54 / 0,8 = 513034,43 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим 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.

Результати дослідження проведені за темою «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Показник	1-й рік	2-й рік	3-й рік
Збільшення кількості споживачів, користувачів	20000	15000	8000

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 100000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 1200,00 грн;

$\pm\Delta Ц_o$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [34]:

$$\Delta П_i = (\pm\Delta Ц_o \;\; N + Ц_o \;\; \Delta N)_i \;\; \lambda \;\; \rho \;\; (1 - \frac{\vartheta}{100}), \qquad (5.16)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо $\rho = 38\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (500,00 \cdot 100000,00 + 1700,00 \cdot 20000) \cdot 0,83 \cdot 0,38 \cdot (1 - 0,18/100\%) = 21803275,2$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (500,00 \cdot 100000,00 + 1700,00 \cdot (20000 + 15000)) \cdot 0,83 \cdot 0,38 \cdot (1 - 0,18/100\%) =$ 28422126,6грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (500,00 \cdot 100000,00 + 1700,00 \cdot (20000 + 15000 + 8000)) \cdot 0,83 \cdot 0,38 \cdot (1 - 0,18/100\%) =$ 31952180,68 грн.

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

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.17)$$

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

T – 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, $\tau = 0,25$;

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

$ПП = 21803275,2/(1+0,25)^1+28422126,6/(1+0,25)^2+31952180,68/(1+0,25)^3=51992297,69$грн

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \ \ ЗВ, \tag{5.18}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=3;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 513034,43грн.

$PV = k_{інв} \ \ ЗВ = 3 \cdot 513034,43 = 1539103,285$грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.19}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 51992297,69 грн;

PV – теперішня вартість початкових інвестицій, 1539103,285грн.

$E_{абс} = ПП - PV = 51992297,69 - 1539103,285 = 50453194,41$грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.20)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 50453194,41грн грн;

PV – теперішня вартість початкових інвестицій, 1539103,285грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+50453194,41/1539103,285)^{1/3} = 2,23.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.21)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,2.

$\tau_{мін}$ = 0,11+0,2 = 0,31 < 1,05 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (5.22)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 2,23 = 0,45 року.

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

Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія» становить 39,7 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Також термін окупності становить 0,45 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»».

ВИСНОВКИ

У магістерській кваліфікаційній роботі був розроблений програмний додаток гра «Монополія» з елементами штучного інтелекту. Під час роботи було розглянуто розвиток гри «Монополія» та розвиток штучного інтелекту. Була описана проблема класичної гри і методи їх вирішення під час розробки програмного додатку. Окрім того було розглянуто існуючи аналоги, порівняні з розроблювальним додатком.

Для більш якісної взаємодії з користувачем був обраний найбільш придатний інтерфейс, а саме графічний. Також був ретельно розписаний інтерфей для головних вікон гри, а саме «Головне меню», «Меню вибору гравців», «Гра». Були розроблені UML діаграми та блок-схеми алгоритмів головних сцен. Нарешті був побудований алгоритм роботи штучного інтелекту для легкої версії гри з заданими параметрами гри. Окрім того був розписаний процес навчання штучного інтелекту у грі для забезпечення більш цікавої гри зі штучним інтелектом. Для того, щоб грати з більш навчиним супротивниками з’явилася можливість обирати складну складність гри.

Для розробки було проаналізовано та порівняно різноманітні мови програмування серед яких було обрано об’єктно-орієнтовну мову C#. Серед ігрових рушіїв було вирішено обрати саме ігровий рушій Unity. Для написання коду було обрано середовище розробки Visual Studio. Для користувачів була сформована максимально детальна інтсрукція користувача в якій описані всі можливі дії користувача під час використання ігрового додатку.

Було проведено тестування ігрового додатку. Серед декількох варіантів методів тестування було обрано юніт-тестування. Завдяки інструментам в середені ігрового рушія Unity були розроблені юніт-тести для декількох методів програми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Д.В. Богомазов, Д.І. Катєльніков, "Розробка ігрового застосунку з елементами штучного інтелекту з використанням технології Unity та мови C#" в Матеріали конференції «LI Науково-технічна конференція підрозділів Вінницького національного технічного університету (2022)», Вінниця, 2022. [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/allvntu/index/pages/view/zbirn2022 Дата звернення: Черв. 2022.

2. Катєльніков Д.І., Богомазов Д.В. Розробка модуля мережевого обміну для ігрового застосунку з елементами штучного інтелекту з використанням технології Unity та мови C#. Матеріали II Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації - 2022» Одеса, 29-30 вересня 2022 р. - Одеса, Видавництво ОНАХТ, 2022 р. – С. 117-119.

3. Богомазов Д.В., Катєльніков Д.І. Розробка методу та програмного забезпечення модуля штучного інтелекту для гри "Монополія". // Матеріали III Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації - 2023», Одеса, 28-29 жовтня 2023 р. Одеса, Видавництво ОНТУ, 2023 р. С.162-165.

4. Богомазов Д. В., Катєльніков Д. І. Розробка модуля штучного інтелекту для гри "Монополія". Збірник матеріалів Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ», 20 – 21 листопада, 2023. Суми/Вінниця. – С. 35– 38.

5. Monopoly (game)[Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Monopoly_(game)

6. History of Monopoly [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/History_of_Monopoly

7. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Francis Bach. Deep Learning (Adaptive Computation and Machine Learning series). The MIT Press, 2016. 776c.

8. Nikhil Buduma, Joe Papa, Nithin Buduma. Fundamentals of Deep Learning. Designing Next-Generation Machine Intelligence Algorithms. 2nd Edition. O'Reilly, 2022, 388c.

9. Monopoly Silvergames [Електронний ресурс]. URL: https://www.silvergames.com/en/monopoly.

10. Capitalista [Електронний ресурс]. URL: https://gidd.io/games/capitalista/#:~:text=Capitalista%20is%20a%20classical%20dice,with%20up%20to%2020%20players.

11. Monopoly star [Електронний ресурс]. URL: https://monopolystar.com.

12. Кононюк А.Ю. Нейроні мережі і генетичні алгоритми: Науково-практичне видання. – Київ: Корнійчук, 2008. 446 с.

13. А.С. Савченко, О.О. Синельніков. Методи та системи штучного інтелекту. Київ, 2017. -177 с.

14. UML [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Unified_Modeling_Language.

15. Текстовий інтерфейс [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Текстовий_інтерфейс_користувача.

16. Графічний інтерфейс [Електронний ресурс]. URL: https://www.wikiwand.com/uk/Графічний_інтерфейс_користувача.

17. Веб-інтерфейси [Електронний ресурс]. URL: https://avada-media.ua/ua/services/webinterface/.

18. Троелсен, Эндрю. Мова програмування C# 2005 та платформа .NET 2.0. 3 видання.: Пер. с англ. – М.: ООО ”Дім Вільямс”, 2007. 1168 с..

19. Марк Лутц. Вивчаємо Python (5 видання, том 1).: Пер. с англ. – М.: ООО ” Диалектика”, 2020. 833 с.

20. Блинов, И.Н., Романчик, В. С. Java. Методи програмування.: Пер. с англ. – М.: ООО ” Чотири чверті”, 2013. 896 с.

21. Бьярне Страуструп. Програмування. Принципи і практика з використанням C++.: Пер. с англ. – М.: ООО ”Дім Вільямс”, 2018. 1329 с.

22. Joe Hocking. Unity in Action, Third Edition: Multiplatform game development in C# 3rd Edition. O'Reilly, 2019. 352 с.

23. Aram Cookson, Ryan DowlingSoka, Tim Johnson, Clinton Crumpler. Unreal Engine 4 Game Development in 24 Hours. Sams, 2016. 496 с.

24. Godot Engine [Електронний ресурс]. URL: https://godotengine.org.

25. Bruce Johnson. Professional Visual Studio 2019. Wrox, 2019. 1032с.

26. Ed Burnette. Eclipse IDE: Pocket Guide. O'Reilly, 2005. 163с.

27. Carlo Leonardini, Vincent Massol. IntelliJ IDEA: The Definitive Guide. Apress, 2020. 443с.

28. Інтеграційні тести [Електронний ресурс]. URL: https://qalight.ua/baza-znaniy/integratsijne-testuvannya/.

29. Системні тести [Електронний ресурс]. URL:https://qalight.ua/baza-znaniy/sistemne-testuvannya/.

30. Автиматизовані тести [Електронний ресурс]. URL:https://qalight.ua/baza-znaniy/avtomatizovane-testuvannya/.

31. Модульне тестування [Електронний ресурс]. URL: https://qalight.ua/baza-znaniy/modulne-testuvannya/.

32.Unity unit testing [Електронний ресурс]. URL: https://docs.unity3d.com/Manual/testing-editortestsrunner.html

33. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

34. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

ДОДАТКИ

Додаток А
(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
"20" вересня 2023 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія» » за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доцент Д. І. Катєльніков
"20" вересня 2023 р.
Виконав:
_____ студент гр. 3ПІ-22м Д. В. Богомазов
"20" вересня 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія»».

Галузь застосування – розробка комп'ютерних ігор.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою даної магістерської роботи є розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія», який здатен конкурувати з гравцями та надавати їм високий рівень виклику.

Призначення роботи – методи та засоби розробки модулів штучного інтелекту для покрокових стратегічних економічних ігрових додатків «Монополія».

4. Вихідні дані для проведення НДР

1. History of Monopoly [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/History_of_Monopoly

2. Кононюк А.Ю. Нейроні мережі і генетичні алгоритми: Науково-практичне видання. – Київ: Корнійчук, 2008. 446 с.

3. Троелсен, Эндрю. Мова програмування C# 2005 та платформа .NET 2.0. 3 видання.: Пер. с англ. – М.: ООО "Дім Вільямс", 2007. 1168 с..

4. Joe Hocking. Unity in Action, Third Edition: Multiplatform game development in C# 3rd Edition. O'Reilly, 2019. 352 с.

5. Unity unit testing [Електронний ресурс]. URL: https://docs.unity3d.com/Manual/testing-editortestsrunner.html

5. Технічні вимоги

Детальний огляд офіційних правил гри «Монополія»; визначення основних аспектів гри; огляд існуючих підходів до створення штучного інтелекту для настільних ігор; створення моделі гравця з використанням різних підходів машинного навчання або імітації поведінки гравців; розробка алгоритмів прийняття рішень; проведення тестів штучного інтелекту; підготовка інструкції користувача.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

пояснювальна записка до МКР;

технічне завдання;

лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз вибору методу розробки та постановка задачі дослідження	20.09. 2023 - 01.10.2023

2	Розробка архітектури та алгоритмів програмного продукту	02.10.2023 - 16.10.2023
3	Розробка функціоналу та модулів програми	17.10.2023 - 11.11.2023
4	Тестування програми	12.11.2023 - 20.11.2023
5	Економічна частина	21.11.2023 - 8.12.2023

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б

(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка методу та програмного забезпечення модуля штучного інтелекту для гри «Монополія».

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Катєльніков Д. І.

Оригінальність	96.2
Схожість	3.8

Аналіз звіту подібності

Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Богомазов Д.В.

Керівник роботи _____ Катєльніков Д. І.

Додаток В. Лістинг програмного коду

(довідниковий)

Bot.cs

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

public class Bot : Player
{
    public void StartTurn()
    {
        Debug.Log("Deep1");
        GameObject.Find("UI").GetComponent<Buttons>().ThrowCube();
        StartCoroutine(ChooseBehavior());
    }

    public IEnumerator ChooseBehavior()
    {
        Debug.Log("Deep2");
        yield return new WaitForSeconds(3f);
        Card card = Info.S.cardActive.GetComponent<Card>();
        switch (card.cardtype)
        {
            case Cardtype.Place:
                CardPlace place = (CardPlace)card;
                if (place.boss == null)
                {
                    if (money > place.money && districts[place.districtNumber].counter != 0)
                    {
                        GameObject.Find("NewButtonMenu").GetComponent<Place>().Buy();
                    }
                    else if (money > place.money && cards.Count < 6)
                    {
                        GameObject.Find("NewButtonMenu").GetComponent<Place>().Buy();
                    }
                }
                else
                {
                    //Have money
                    if (money > place.price)
```

```
GameObject.Find("EnemyButtonMenu").GetComponent<BuyPlace>().Price();
                        else
                            StartCoroutine(NoMoney(place));
                    }
                    break;
                case Cardtype.Buisiness:
                    if (card.boss == null) { if (money > card.money)
{ GameObject.Find("NewButtonMenu").GetComponent<Place>().Buy(); } }
                    else { StartCoroutine(NoMoney(card)); }
                    break;
                case Cardtype.Stay:
                    //
                    break;
                case Cardtype.Casino:
                    float rand = Random.value;
                    if(rand > 0.6f)
                    {
                        rand = Random.value;
                        InputField casinoInput =
GameObject.Find("CasinoField").GetComponent<InputField>();
                        if(rand <= 0.5f)
                        {
                            casinoInput.text = ((int)(money * 0.05f)).ToString();
                            yield return new WaitForSeconds(1f);

GameObject.Find("CasinoButtonMenu").GetComponent<Casino>().GoPlay();
                        }
                        else if(rand <= 0.85f)
                        {
                            casinoInput.text = ((int)(money * 0.1f)).ToString();
                            yield return new WaitForSeconds(1f);

GameObject.Find("CasinoButtonMenu").GetComponent<Casino>().GoPlay();
                        }
                        else if (rand <= 0.95f)
                        {
                            casinoInput.text = ((int)(money * 0.2f)).ToString();
                            yield return new WaitForSeconds(1f);

GameObject.Find("CasinoButtonMenu").GetComponent<Casino>().GoPlay();
                        }
                        else
                        {
                            casinoInput.text = ((int)(money * 0.5f)).ToString();
                            yield return new WaitForSeconds(1f);
```

```
GameObject.Find("CasinoButtonMenu").GetComponent<Casino>().GoPlay();
                        }
                    }
                    break;
                case Cardtype.Taxi:
                    for(int i = 0; i < 8; i++)
                    {
                        if(districts[i].cardsCounter == 2)
                        {
                            int name = 0;
                            string kostname = "zero";
                            for (int j = 0; j < 3; j++)
                            {
                                if (districts[i].cards[j] == null)
                                {
                                    name = j;
                                }
                                else
                                {
                                    kostname = districts[i].cards[j].name;
                                }
                            }

GameObject.Find("TaxiDrop").GetComponent<Dropdown>().captionText.text                        =
kostname.Substring(0, kostname.Length - 1) + name;
                            Debug.Log("What???" + kostname.Substring(0, kostname.Length -
1) + name);
                            GameObject.Find("TaxiButtonMenu").GetComponent<Taxi>().Find();
                            yield return new WaitForSeconds(2f);
                            GameObject.Find("TaxiButtonMenu").GetComponent<Taxi>().Move();
                        }
                    }
                    break;

            }
            StartCoroutine(ChooseCenter());
        }

        public IEnumerator ChooseCenter()
        {
            Debug.Log("Deep3");
            //Bank
            if(takeCredit && money > creditCount)
            {
                GameObject.Find("UI").GetComponent<Buttons>().BankButActive();
                yield return new WaitForSeconds(2f);
```

```
                GameObject.Find("Bank").GetComponent<BankMenu>().GoPay();
                yield return new WaitForSeconds(2f);
                GameObject.Find("Bank").GetComponent<BankMenu>().DestroyThis();
            }
            yield return new WaitForSeconds(2f);
            //Trade
            List<int> numbers = new List<int>();
            for (int i = 0; i < 8; i++)
            {
                if (districts[i].cardsCounter == 2)
                {
                    numbers.Add(i);
                }
            }
            if (numbers.Count > 0)
            {
                GameObject.Find("UI").GetComponent<Buttons>().BankButActive();
                yield return new WaitForSeconds(2f);
            }
            yield return new WaitForSeconds(2f);
            //Houses
            if (haveDistricts.Count > 0)
            {
                foreach(District d in haveDdistricts)
                {
                    if(d.cards[0].houses[0].GetComponent<House>().moneyInt * 2 < money)
                    {
                        GameObject.Find("UI").GetComponent<Buttons>().HousesButActive();
                        yield return new WaitForSeconds(2f);
                        int place = 0;
                        for(int i = 0; i < 3; i++)
                        {
                            if (d.cards[place].houseCounter > d.cards[i].houseCounter)
                                place = i;
                        }
                        Panels panel = new Panels();
                        for(int i = 0; i < 4; i++)
                        {
                            if
(GameObject.Find("HouseMenu").GetComponent<HousesMenu>().panels[i].GetComponent<Panels>().d
istrict == d.district)
                            {
                                panel                                                    =
GameObject.Find("HouseMenu").GetComponent<HousesMenu>().panels[i].GetComponent<Panels>();
                                break;
                            }
                        }
```

```
                        foreach(GameObject h in panel.cards)
                        {
                            if(!h.GetComponent<House>().isBuy)
                            {
                                h.GetComponent<House>().GoTry();
                                break;
                            }
                        }
                        yield return new WaitForSeconds(1f);
                        GameObject.Find("HouseMenu").GetComponent<HousesMenu>().GoBuild();
                        yield return new WaitForSeconds(1f);

GameObject.Find("HouseMenu").GetComponent<HousesMenu>().DestroyThis();
                    }
                }
            }
            yield return new WaitForSeconds(3f);
            Debug.Log("Deep4");
            GameObject.Find("EndTurn").GetComponent<Button>().onClick.Invoke();
        }
        public IEnumerator NoMoney(Card card)
        {
            //Have money
            if (money > card.price)
                GameObject.Find("EnemyButtonMenu").GetComponent<BuyPlace>().Price();
            //Bank credit
            else if (card.price - money < Info.S.money)
            {
                GameObject.Find("UI").GetComponent<Buttons>().BankButActive();
                yield return new WaitForSeconds(2f);
                GameObject.Find("BankInput").GetComponent<InputField>().text            =
((card.price - money) + 100).ToString();
                GameObject.Find("Bank").GetComponent<BankMenu>().TakeCredit();
                GameObject.Find("Bank").GetComponent<BankMenu>().DestroyThis();
                yield return new WaitForSeconds(2f);
                GameObject.Find("EnemyButtonMenu").GetComponent<BuyPlace>().Price();
            }
            //Sell houses
            else if (card.price - money >= Info.S.money && haveDistricts.Count > 0)
            {
                foreach (District d in haveDistricts)
                {
                    if (d.cards[0].houses[0].GetComponent<House>().moneyInt * 2 < money)
                    {
                        GameObject.Find("UI").GetComponent<Buttons>().HousesButActive();
                        yield return new WaitForSeconds(2f);
                        int place = 0;
```

```
                            for (int i = 0; i < 3; i++)
                            {
                                if (d.cards[place].houseCounter < d.cards[i].houseCounter)
                                    place = i;
                            }
                            Panels panel = new Panels();
                            for (int i = 0; i < 4; i++)
                            {
                                if
(GameObject.Find("HouseMenu").GetComponent<HousesMenu>().panels[i].GetComponent<Panels>().d
istrict == d.district)
                                {
                                    panel                                                    =
GameObject.Find("HouseMenu").GetComponent<HousesMenu>().panels[i].GetComponent<Panels>();
                                    break;
                                }
                            }
                            foreach (GameObject h in panel.cards)
                            {
                                if (h.GetComponent<House>().isBuy)
                                {
                                    h.GetComponent<House>().GoTry();
                                    break;
                                }
                            }
                            yield return new WaitForSeconds(1f);
                            GameObject.Find("HouseMenu").GetComponent<HousesMenu>().Sell();
                            yield return new WaitForSeconds(1f);

GameObject.Find("HouseMenu").GetComponent<HousesMenu>().DestroyThis();
                        }
                    }
                }
                //Sell places
                else if (haveDistricts.Count == 0 && cards.Count > 0)
                {
                    GameObject.Find("UI").GetComponent<Buttons>().SellButActive();
                    yield return new WaitForSeconds(2f);
                    StartCoroutine(SellPlaces(card));
                }
                else
                {
                    StartCoroutine(LastChance());
                }
            }

            public IEnumerator SellPlaces(Card card)
```

```
        {
            if(money <= card.price && cards.Count > 0)
            {
                int number = -1;
                for(int i = 0; i < 8; i++)
                {
                    if(districts[i].counter > 0 && ((number != -1) ||
districts[number].counter > districts[i].counter))
                    {
                        number = i;
                    }
                }
                string name = "";
                for(int i = 0; i < 3; i++)
                {
                    if(districts[number].cards[i] != null)
                    {
                        name = districts[number].cards[i].name;
                    }
                }
                if (name == "")
                    name = districts[number].buisiness.name;
                SellCardmini need = new SellCardmini();
                foreach(GameObject g in
GameObject.Find("Sell").GetComponent<SellMenu>().cards)
                {
                    if (g.GetComponent<SellCardmini>().card.name == name)
                    {
                        need = g.GetComponent<SellCardmini>();
                        break;
                    }
                }
                need.Active();
                GameObject.Find("Sell").GetComponent<SellMenu>().SellButton();
                yield return new WaitForSeconds(2f);
                StartCoroutine(SellPlaces(card));
            }
            else if(money > card.price)
            {
                GameObject.Find("EnemyButtonMenu").GetComponent<BuyPlace>().Price();
            }
            else if(cards.Count == 0)
            {
                StartCoroutine(LastChance());
            }
            yield return new WaitForSeconds(2f);
        }
```

```
    public IEnumerator LastChance()
    {
        GameObject.Find("UI").GetComponent<Buttons>().HopeCubeButActive();
        yield return new WaitForSeconds(2f);
        GameObject.Find("HopeCube").GetComponent<HopeCubeMenu>().Throw();
        yield return new WaitForSeconds(5f);
        if (!bankrupt)
            GameObject.Find("EnemyButtonMenu").GetComponent<BuyPlace>().Price();
    }
}
```

Info.cs

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

//Enum with weather type
public enum Weather
{
    sun,
    rain,
    wind,
    lighting,
    shine
}

//Enum with card type
public enum Cardtype
{
    Start,
    Prison,
    Casino,
    Taxi,
    Stay,
    Place,
    Buisiness,
    Bonus
}

//Main data
public class Info : MonoBehaviour
{
    static public Info S;
    [Header("Main Data")]
```

```
//Players count
public int playersCount = 1;
//Money count
public int money;
//Player active number
public int numberHod;
//Max players in the game
public int maxPlayers;
//Number of turn
public int turn;
//Card active
public GameObject cardActive;
//Player active
public GameObject player;
//Districts (no need now)
public List<District> districts;
//Bonuses
public List<int> bonuses;

[Header("Players Data")]
//Players list
public GameObject[] players = new GameObject[8];
//Plyaers colors
public Color[] colors = new Color[8];
//Players names
public string[] names = new string[8];
//Player portraits
public Sprite[] portraits = new Sprite[8];
//Bot status
public bool[] botStatus = new bool[8];

[Header("Weather Data")]
//Weathers sprite
public Sprite[] weathers = new Sprite[5];
//Wather objects
public GameObject[] weatherObjects = new GameObject[5];
//Weather count
public int weatherCount;
//Weather type in this turn
public Weather weather;

[Header("Card Menu Data")]
//Card menu panel
public GameObject cardMenu;
//Card menu buttins
public GameObject[] buttonsMenu = new GameObject[7];
```

```
            [Header("Cards Data")]
            //All cards in game
            public GameObject[] cards = new GameObject[48];

            [Header("Stay Stops")]
            //Stops
            public List<GameObject> stops;
            int counter = 0;

            [Header("Prefabs")]
            //Prefab card place
            public GameObject prefabCardPlace;
            //Parent for card place
            public GameObject prefabRightPanel;

            //Initial parameters
            private void Awake()
            {
                S = this;
                numberHod = 1;
                maxPlayers = PlayerPrefs.GetInt("MaxPlayers");
                money = PlayerPrefs.GetInt("Money");
                for(int i = 0; i < maxPlayers; i++)
                {
                    names[i] = PlayerPrefs.GetString("NamePlayer" + i);
                    portraits[i] = Resources.Load<Sprite>("Sprites/NewSprites/Characters/" +
PlayerPrefs.GetString("PortraitPlayer" + i));
                    botStatus[i] = bool.Parse(PlayerPrefs.GetString("BotPlayer" + i));
                    colors[i] = CreateColor(i);
                }
                turn = 1;
                weather = Weather.shine;
                weatherCount = 1;
                player = players[0];
                districts = new List<District>();
                //bonuses
                bonuses.Add(100);
                bonuses.Add(200);
                bonuses.Add(300);
                bonuses.Add(500);
                bonuses.Add(1000);
                //stops = new List<GameObject>();
            }

            //Convert string to Color
            Color CreateColor(int i)
            {
```

```
            Color newColor;
            string[] newString;
            newString = PlayerPrefs.GetString("ColorPlayer" + i).Split(new char[]
{ ';' });
            print(PlayerPrefs.GetString("ColorPlayer" + i));
            float r, g, b;
            r = float.Parse(newString[0]);
            g = float.Parse(newString[1]);
            b = float.Parse(newString[2]);
            newColor = new Color(r, g, b, 255);
            print(newColor);
            return newColor;
        }

        //
        public void YourTurn(int cubeThrow)
        {
            player = GameObject.Find("player" + numberHod);
            if (!player.GetComponent<Player>().inPrison)
            {
                CubesThrow(cubeThrow);
            }
            else
            {
                Move();
            }
        }

        //Cubes throw
        public void CubesThrow(int cubeThrow)
        {
            //Dubl status
            bool both = false;
            int chance;
            //Take number from buttons script
            cubeThrow = GameObject.Find("UI").GetComponent<Buttons>().firstThrow +
GameObject.Find("UI").GetComponent<Buttons>().secondThrow + 2;
            if (GameObject.Find("UI").GetComponent<Buttons>().firstThrow ==
GameObject.Find("UI").GetComponent<Buttons>().secondThrow)
                both = true;
            //remove player from previous card

cards[player.GetComponent<Player>().place].GetComponent<Card>().players.Remove(player);

            //Check weather status
            switch(weather)
            {
```

```
                case Weather.shine:
                    break;
                case Weather.rain:
                    break;
                case Weather.lighting:
                    chance = Random.Range(0, 2);
                    if (chance == 1)
                    {
                        int count = Random.Range(0, cubeThrow);
                        cubeThrow -= count;
                        Debug.Log("Lighting - " + count);
                    }
                    break;
                case Weather.wind:
                    cubeThrow -= 2;
                    break;
                case Weather.sun:
                    chance = Random.Range(0, 2);
                    if(chance == 1)
                    {
                        cubeThrow = (-cubeThrow);
                        Debug.Log("Sun");
                    }
                    break;
            }
            //Player move
            player.GetComponent<Player>().place += cubeThrow;
            if(player.GetComponent<Player>().place < 0)
            {
                player.GetComponent<Player>().place += 48;
            }
            //Player end circle
            if (player.GetComponent<Player>().place > 47)
            {
                player.GetComponent<Player>().place -= 48;
            }
            //Check if dubl, if its true player can moving again
            /*if (!both)
            {
                GameObject.Find("CubesButton").GetComponent<Button>().interactable      =
false;
                GameObject.Find("EndTurn").GetComponent<Button>().interactable = true;
            }*/
            Move();
        }

        //Change weather
```

```
        public void NewWeather()
        {
            int randNumber = Random.Range(1, 7);
            Weather previos = weather;
            switch(randNumber)
            {
                case 1:
                    weather = Weather.sun;
                    break;
                case 2:
                    weather = Weather.rain;
                    break;
                case 3:
                    weather = Weather.wind;
                    break;
                case 4:
                    weather = Weather.lighting;
                    break;
                case 5:
                case 6:
                    weather = Weather.shine;
                    break;
            }
            Sprite check;
            for(int i = 0; i < 4; i++)
            {
                //Change weather
                if(weatherObjects[i].GetComponent<ChangeWeather>().weather == weather)
                {
                    check = weatherObjects[4].GetComponent<Image>().sprite;
                    weatherObjects[i].GetComponent<ChangeWeather>().weather = previos;
                    weatherObjects[4].GetComponent<ChangeWeather>().weather = weather;
                    weatherObjects[4].GetComponent<Image>().sprite                          =
weatherObjects[i].GetComponent<Image>().sprite;
                    weatherObjects[i].GetComponent<Image>().sprite = check;
                }
            }
            //Check if weather repeated
            if (weather == previos && weather != Weather.shine)
            {
                weatherCount++;
            }
            else
            {
                weatherCount = 1;
            }
        }
```

```
        //Open stops
        public void OpenGate()
        {
            counter++;
            if(stops.Count > 0 && counter == 3)
            {
                counter = 0;
                Destroy(stops[0]);
                stops.RemoveAt(0);
            }
        }

        //Start next turn (before cubes throw)
        public void NextPlayer()
        {
            GameObject.Find("EndTurn").GetComponent<Button>().interactable = false;
            GameObject.Find("CubesButton").GetComponent<Button>().interactable = true;
            //Exit card menu
            for(int i = 0; i < cardMenu.GetComponent<CardMenu>().buttonsMenu.Length; i++)
            {
                cardMenu.GetComponent<CardMenu>().buttonsMenu[i].SetActive(false);
            }
            Debug.Log("Nextplayer");
        }

        //Player move
        public void Move(bool active = true)
        {
            cardActive = cards[0];
            for (int i = 0; i < 48; i++)
            {
                if (cards[i].GetComponent<Card>().number == player.GetComponent<Player>().place)
                {
                    //Change card active
                    cardActive = cards[i];
                    //Change player location (Check if some players stop in this place)
                    if (cardActive.GetComponent<Card>().players.Count > 0)
                    {
                        Vector3 another;
                        if (i > 12 && i < 24)
                        {
                            another = new Vector3(cards[i].GetComponent<Card>().playerPlace.x,
                                cards[i].GetComponent<Card>().playerPlace.y - 10 * cardActive.GetComponent<Card>().players.Count,
```

```
                                cards[i].GetComponent<Card>().playerPlace.z);
                            }
                            else if (i > 36 && i <= 47)
                            {
                                another                             =                             new
Vector3(cards[i].GetComponent<Card>().playerPlace.x,
                                cards[i].GetComponent<Card>().playerPlace.y    +    10    *
cardActive.GetComponent<Card>().players.Count,
                                cards[i].GetComponent<Card>().playerPlace.z);
                            }
                            else if (i > 24 && i < 36)
                            {
                                another                             =                             new
Vector3(cards[i].GetComponent<Card>().playerPlace.x             +             10             *
cardActive.GetComponent<Card>().players.Count,
                                cards[i].GetComponent<Card>().playerPlace.y,
                                cards[i].GetComponent<Card>().playerPlace.z);
                            }
                            else
                            {
                                another                             =                             new
Vector3(cards[i].GetComponent<Card>().playerPlace.x             -             10             *
cardActive.GetComponent<Card>().players.Count,
                                cards[i].GetComponent<Card>().playerPlace.y,
                                cards[i].GetComponent<Card>().playerPlace.z);
                            }
                            player.transform.position = another;
                        }
                        else
                        {
                            player.transform.position                                           =
cards[i].GetComponent<Card>().playerPlace;
                        }
                        break;
                    }
                }
                //Add player to card
                cardActive.GetComponent<Card>().players.Add(player);
                for (int i = 0; i < 7; i++)
                {
                    buttonsMenu[i].SetActive(false);
                }
                //Check if it is default move or its move from another place
                if (active)
                    DefaultMove();
                else
                    StayMove();
```

```
        }

        //Default move
        public void DefaultMove()
        {
            //Check card and open required card menu
            switch (cardActive.GetComponent<Card>().cardtype)
            {
                case Cardtype.Bonus:
                    cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 6);
                    Bonus();
                    break;
                case Cardtype.Start:
                    cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 6);
                    break;
                case Cardtype.Prison:
                    if                    (GameObject.Find("player"                    +
numberHod).GetComponent<Player>().inPrison)
                    {
                        cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 5);
                    }
                    else
                    {
                        cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 6);
                    }
                    break;
                case Cardtype.Casino:
                    cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 3);
                    break;
                case Cardtype.Taxi:
                    cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 4);
                    break;
                case Cardtype.Stay:
                    cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 2);
                    break;
                case Cardtype.Buisiness:
                case Cardtype.Place:
                    if (!cardActive.GetComponent<Card>().isBuy)
                    {
```

```
                        cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 0);
                    }
                    else            if(cardActive.GetComponent<Card>().boss            !=
player.GetComponent<Player>())
                    {
                        cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 1);
                    }
                    else
                    {
                        cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 6);
                    }
                    break;
                }
            }

            //Moving to stay
            public void StayMove()
            {
                cardMenu.GetComponent<CardMenu>().ChangeCardMenu(cardActive.name,
cardActive.GetComponent<Image>().sprite, 6);
            }

            //Check if player pulled the credit, if not player moving to prison
            public void CheckPrison()
            {
                if (player.GetComponent<Player>().takeCredit)
                {
                    player.GetComponent<Player>().dayCredit--;
                    if (player.GetComponent<Player>().dayCredit == 0)
                    {

cards[player.GetComponent<Player>().place].GetComponent<Card>().players.Remove(player);
                        player.GetComponent<Player>().place = 12;
                        Move();
                        player.GetComponent<Player>().inPrison = true;
                        player.GetComponent<Player>().TakeMoney(-
player.GetComponent<Player>().creditPrice[2]);
                    }
                }
            }

            public void Bonus()
            {
                float rand = Random.value;
```

```
        int bonus = 0;
        if (rand <= 0.05)
            bonus = bonuses[4];
        else if (rand <= 0.20)
            bonus = bonuses[3];
        else if (rand <= 0.40)
            bonus = bonuses[2];
        else if (rand <= 0.65)
            bonus = bonuses[1];
        else
            bonus = bonuses[0];

        Debug.Log("Chance = " + rand + " Bonus = " + bonus);
        player.GetComponent<Player>().TakeMoney(bonus);
    }

}
```

Додаток Д

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДУ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОДУЛЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ГРИ «МОНОПОЛІЯ»

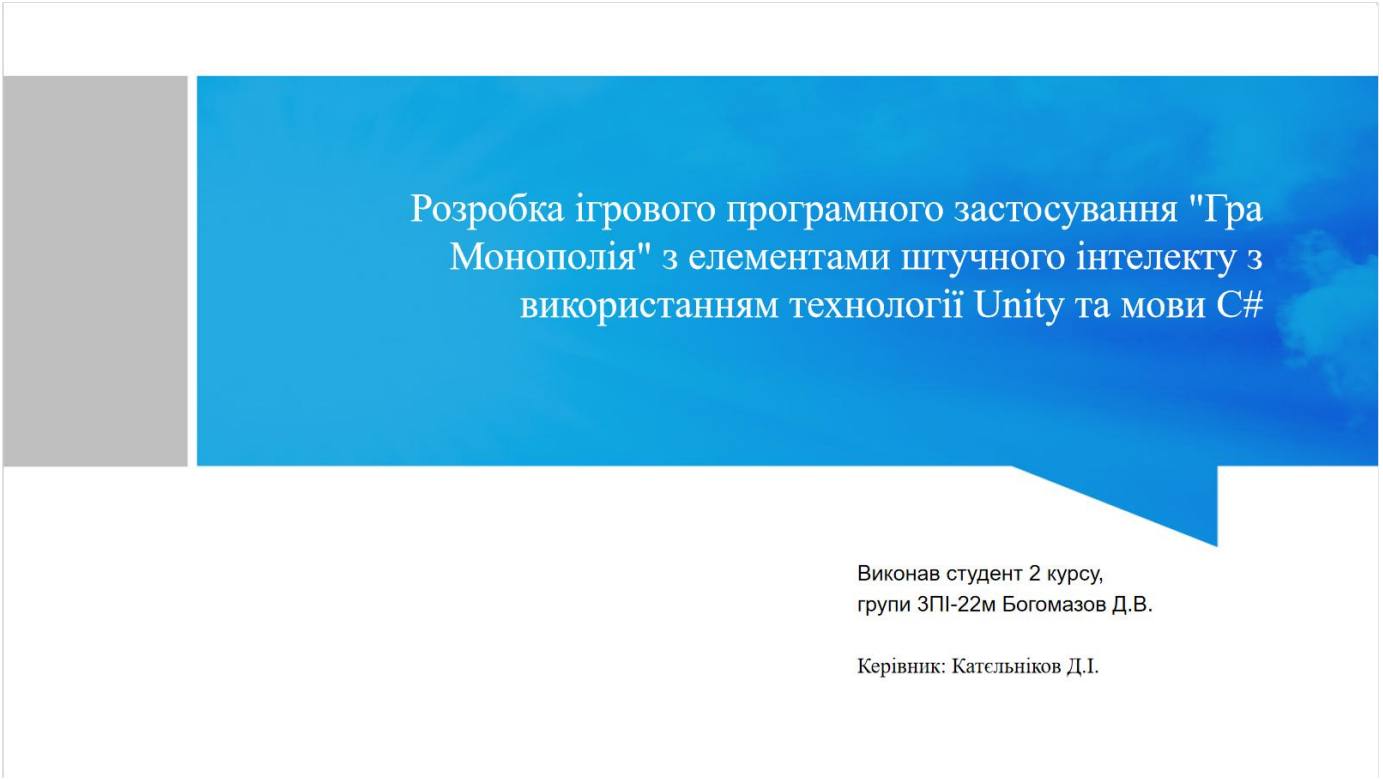


Рисунок Д.1 – Титульний слайд

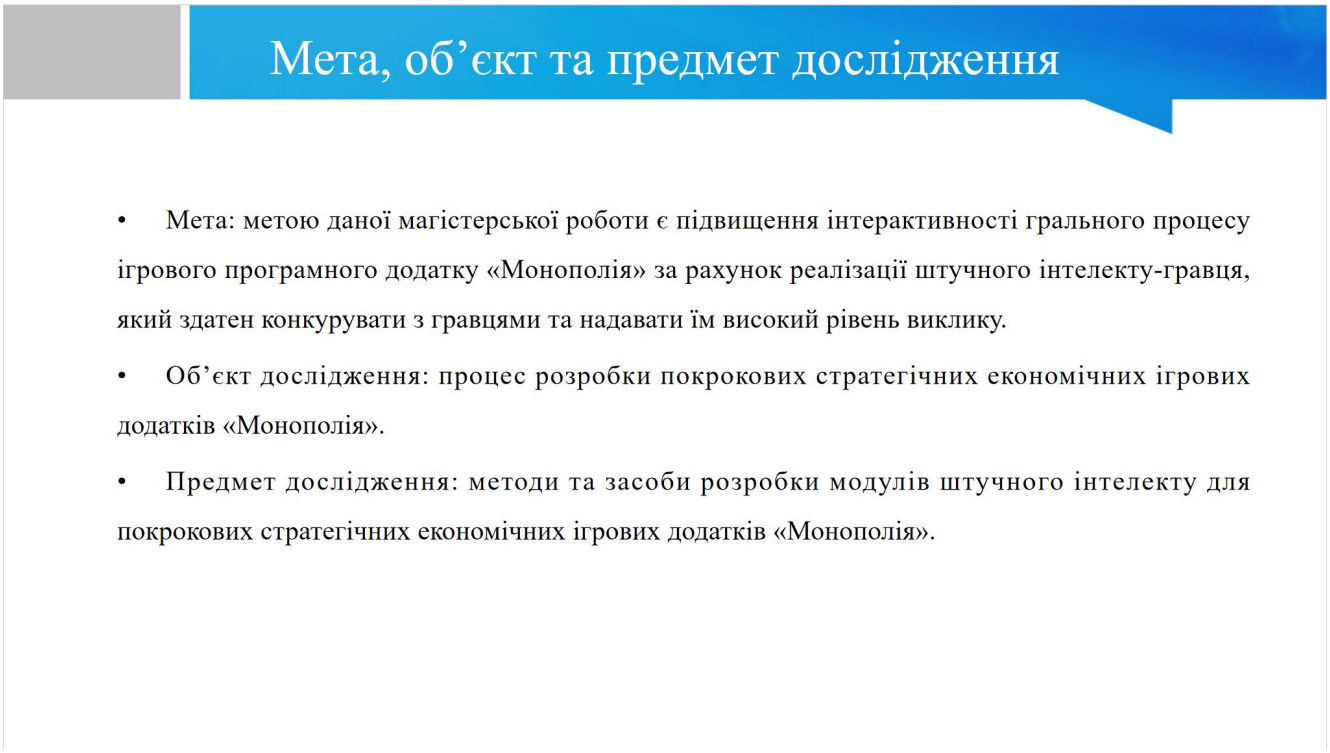


Рисунок Д.2 – Мета, об’єкт та предмет дослідження

Наукова новизна

- Подальшого розвитку отримав метод побудови модуля штучного інтелекту на основі дерева ухвалення рішень, у якому, на відміну від існуючих методів, у вузлах використовуються не жорсткий набір умов, а розширений набір продуктивних правил ЯКЩО-ТОДІ, що дозволило не тільки зробити процес прийняття рішень більш прозорим, але й надало можливість додавати нові правила по мірі накопичення досвіду.

- Уперше запропоновано систему використання методів машинного навчання для ідентифікації та адаптації до індивідуальних стилів гри гравців у «Монополії». Розроблений підхід використовує аналіз дій гравців та автоматизоване визначення їхніх стратегій для покращення взаємодії штучного інтелекту з гравцями.

Рисунок Д.3 – Наукова новизна

Постановка задач розробки

1. Проаналізувати існуючі підходи до побудови моделей штучного інтелекту, порівняти різні методи прийняття рішень.
2. Дослідити структуру гри і ключові ситуації, коли учасник (людина чи модуль ШІ) повинні приймати рішення:
    1. можливість обміну ігровими цінностями між користувачами;
    2. можливість будувати і продавати нерухомість;
    3. можливість стати монополістом і перемогти у грі.
    4. розробити алгоритми прийняття рішень в ключових ситуаціях;
    5. розробити програмну реалізацію модуля штучного інтелекту;
    6. розробити користувацький інтерфейс;
    7. розробити програмний продукт;
    8. провести тестування розробленого програмного забезпечення.

Рисунок Д.4 – Постановка задач розробки

Інструменти для розробки ігрового додатку

Unity – багатоплатформовий інструмент для розробки відеоігор і рушій, на якому вони працюють.

Visual Studio 2019 – це оригінальне середовище запуску, яке дозволяє редагувати, налагоджувати і створювати код, а потім публікувати програмні додатки.

Мова програмування C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research.

Рисунок Д.5 – Інструменти для розробки ігрового додатку

Структура ігрового додатку «Монополія»

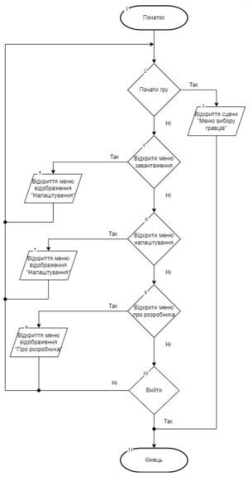
Ігровий програмний додаток «Монополія» складається з трьох основних вікон:

- Головне меню.
- Меню вибору гравців.
- Гра.

Рисунок Д.6 – Структура ігрового додатку

Головне меню

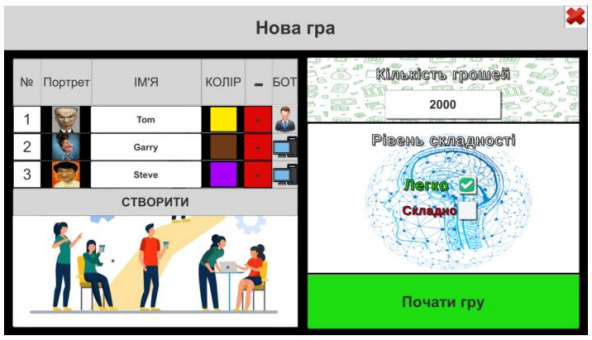
Інтерфейс головного меню



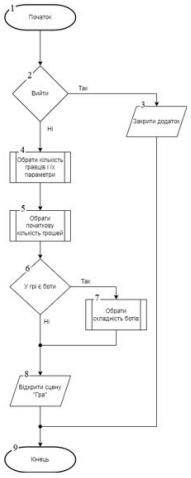
Алгоритм роботи головного меню

Рисунок Д.7 – Головне меню

Меню вибору гравців



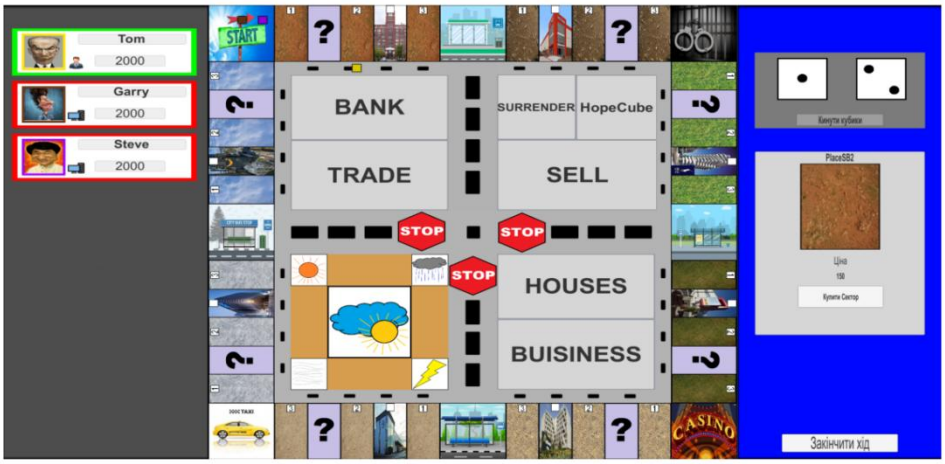
Інтерфейс меню вибору гравців



Алгоритм роботи меню вибору гравців

Рисунок Д.8 – Меню вибору гравців

Гра



Інтерфейс гри

Рисунок Д.9 – Інтерфейс гри

Розробка штучного інтелекту

Для того, щоб грати в гру самому, або додати ще гравців в ігровому додатку реалізован штучний інтелект у вигляді ботів.

Бот імітує дії реального гравця, а саме:

- після того, як хід переходить до бота він автоматично кидає кубики і рухається до потрібного сектору;
- зупинившись на секторі бот обирає чи хоче він взаємодіяти з меню сектору, якщо він потрапляє на куплений сектор він одразу сплачує, якщо вистачає коштів, якщо не вистачає намагається всіма способами заробити шляхом взяття кредиту, продажу будинків та секторів, та навіть кидання кубика надії;
- після взаємодії з сектором в нього є можливість використати функції центрального меню, якщо потрібно;
- в кінці бот натискає клавішу завершити хід.

Рисунок Д.10 – Розробка штучного інтелекту

Рисунок Д.11 – Розробка штучного інтелекту

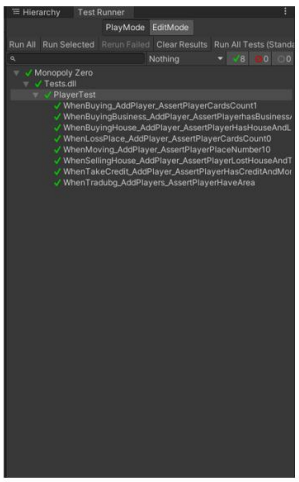
Рисунок Д.12 – Розробка штучного інтелекту

Тестування ігрового додатку

Ігровий рушій Unity має вбудовану функцію юніт-тестів, який реалізується за допомогою вікна Test Runner. Він дозволяє створити папку з тестами, скрипти в якій будуть відокремлені від інших скриптів, щоб не впливати на основний код програмного додатку.

Основна логіка у грі відбувається з гравцем, саме тому тестування буде саме основних методів гравця, а саме Buy() – покупка сектору, Move() – перехід на інший сектор, LostPlace() – втрата сектору та Bankrupt() – поразка гравця.

Окрім того було протестовано функції центрального меню, а саме Trade(), Credit(), BuyHouse(), SellHouse(), BusinessBuy().



Результат тестування

Рисунок Д.13 – Розробка штучного інтелекту

Публікації

- **Публікації.** Основні результати досліджень опубліковано в 2 наукових працях: 1 – у матеріалах конференцій, 1 - авторських свідоцтв про реєстрацію авторського права на комп'ютерну програму.

Рисунок Д.14 – Публікації

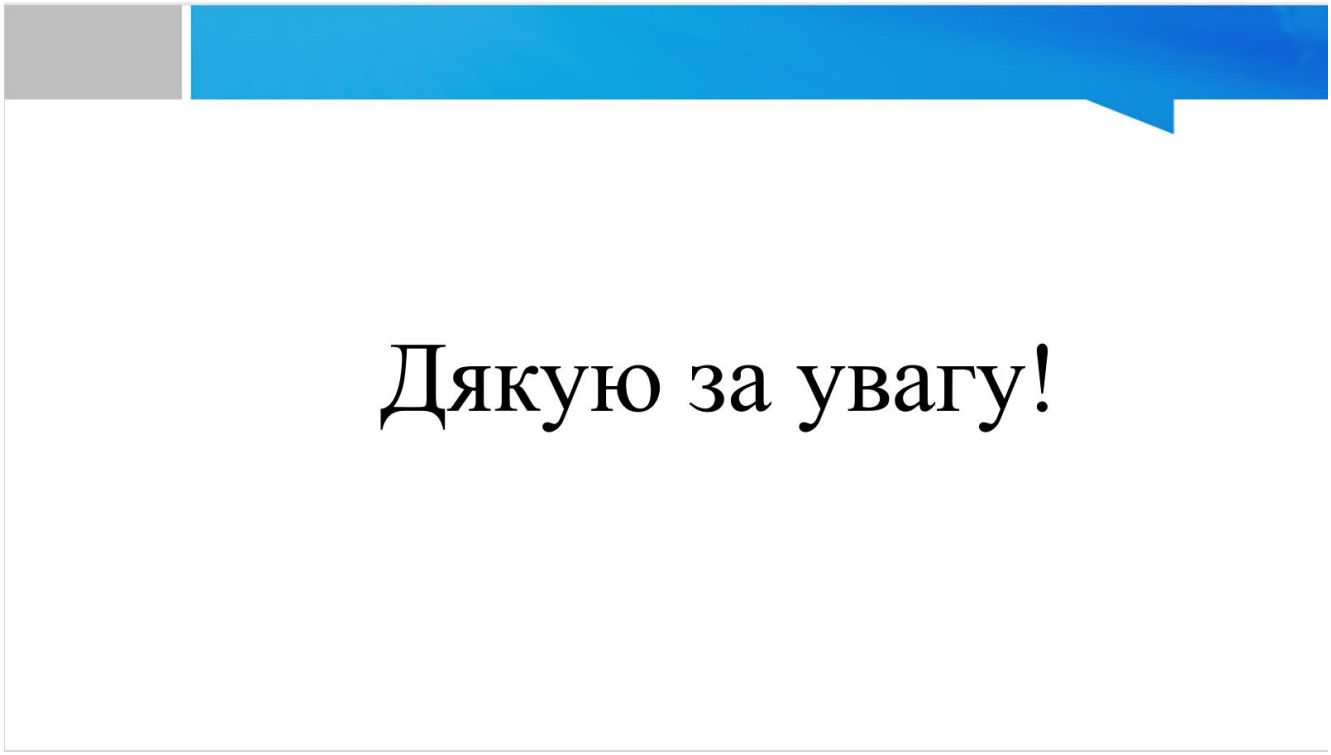


Рисунок Д.15 – Фінальний слайд