Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення

(повна назва кафедри (предметної, циклової комісії)

Пояснювальна записка
До бакалаврської дипломної роботи
бакалавр
(ступінь вищої освіти)

на тему: Розробка ігрового програмного застосування "Гра Монополія" з елементами штучного інтелекту з використанням технології Unity та мови C#.

Виконав: студент 4 курсу, групи 2ПІ-18б
спеціальність
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Богомазов Д.В.
(прізвище та ініціали)

Керівник к.т.н., доц. Катєльніков Д.І.
(прізвище та ініціали)
Рецензент к. т. н., доц. Василевський О.М
(прізвище та ініціали)

Вінниця – 2022 року

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Богомазову Данилу Вікторовичу

1. Тема роботи – «Розробка ігрового програмного застосування "Гра Монополія" з елементами штучного інтелекту з використанням технології Unity та мови C#»

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент, затверджені наказом вищого навчального закладу від 24 березня 2022 р. № 66

2. Строк подання студентом роботи 13 червня 2022 р.

3. Вихідні дані до роботи: інструмент для розробки відеоігор Unity; інтегроване середовище розробки – Microsoft Visual Studio 2019; мова розробки C#.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; розробка архітектури та алгоритмів програмного продукту; розробка програмного продукту; тестування програми, висновки; перелік посилань, додатки.

5. Перелік графічного матеріалу: мета; об'єкт та предмет дослідження; діаграми варіантів використання; структурна схема компонентів додатку; загальний алгоритм роботи; приклади аналогів; тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Технічний розділ	Катєльніков Д.І., к.т.н., доцент		

7. Дата видачі завдання __25 березня 2022 року_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз сучасного стану питання та обґрунтування завдання на роботу	26.03.2022 - 10.04.2022	Вик.
2	Розробка структури серверного додатку	11.04.2022- 25.04.2022	Вик.
3	Розробка алгоритмів програмного додатку	26.04.2022- 30.04.2022	Вик.
4	Розробка інтерфейсу програмного додатку	01.05.2022- 06.05.2022	Вик.
5	Програмна реалізація додатку	07.05.2022- 25.05.2022	Вик.
6	Тестування роботи додатку та алгоритму	26.05.2022- 01.06.2022	Вик.
7	Оформлення матеріалів до захисту БДР	01.06.2022- 10.06.2022	Вик.

Студент _____ **_Богомазов Д.В.**
(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Катєльніков Д.І.**
(підпис) (прізвище та ініціали)

Рецензент бакалаврської дипломної роботи _____ **Василевський О.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 56 сторінках формату А4, на яких є 51 рисунок, 1 таблиця, список використаних джерел містить 12 найменувань.

В бакалаврській дипломній роботі розроблене програмне застосування гра «Монополія» з використанням штучного інтелекту. Даний програмний продукт призначений для розваг.

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

Annotation

The bachelor's thesis consists of 56 A4 pages, which have 51 figures, 1 table, the list of sources used contains 12 items.

In the bachelor's thesis, the software application of game "Monopoly" with use of artificial intelligence is developed. This software product is intended for entertainment.

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

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. На сьогоднішній день прогрес технологій людства досяг рішучих позитивних змін. На даний момент в нашому світі існує дуже багато новітніх і сучасних технологій, а саме: доповнена реальність, віртуальна реальність, штучний інтелект, тощо.

Перший штучний інтелект з’явився приблизно в середині XX століття. На той час вони могли роботи тільки найпростіші міркування.

Вже дивлячись на сьогоднішній день можна сказати, що ця технологія дуже еволюціонувала, тому що, порівняно з тим що було раніше: маємо вже штучний інтелект, який здатен на комплексні міркування, наприклад, якісь дії з користувачем, інтерактивність, обчислювання, логістику та будь що, що цікаво розробнику.

Отже, у вік технологій людей почала приваблювати технологія штучного інтелекту. Однак алгоритми штучного інтелекту й досі потребують вдосконалення, наближення до людської майстерності прийняття рішень, аналізу власних помилок тощо. Це робить актуальною розробку випробувальних полігонів, де можна спостерігати застосування алгоритмів штучного інтелекту, накопичувати статистику їх використання, перевіряти гіпотези та вдосконалювати різні аспекти автоматичного прийняття рішень. Одним з найбільш відомих полігонів є настільні стратегічні економічні ігри, наприклад, “Монополія”.

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

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є підвищення інтерактивності грального процесу за рахунок реалізації штучного інтелекту-гравця при розробці ігрового програмного додатку «Монополія».

Основними задачами дослідження є:

1. проаналізувати існуючі підходи до побудови моделей штучного інтелекту, порівняти різні алгоритми прийняття рішень;

2. дослідити структуру гри і ключові ситуації, коли учасник (людина чи модуль ШІ) повинні приймати рішення:

2.1. можливість обміну ігровими цінностями між користувачами;

2.2. можливість зробити інтервенцію капіталу;

2.3. можливість будувати і продавати нерухомість;

2.4. можливість стати монополістом і перемогти у грі.

2.5. розробити алгоритми прийняття рішень в ключових ситуаціях на основі дерево ухвалення рішень та правил ЯКЩО-ТОДІ;

2.6. розробити програмну реалізацію модуля штучного інтелекту;

2.7. розробити користувацький інтерфейс;

2.8. розробити програмний продукт;

2.9. провести тестування розробленого програмного забезпечення.

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, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

Подальшого розвитку отримала модель штучного інтелекту на основі дерева ухвалення рішень, у якому, на відміну від існуючих реалізацій, 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.

Практична цінність отриманих результатів.

Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби модуля штучного інтелекту для гри в покрокову стратегічну економічниху гру “Монополія”.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованій праці [1], опублікованій у співавторстві, автору належать такі результати: користувацький інтерфейс, архітектура компонентів ігрового додатку. Де авторське свідоцтво про реєстрацію авторського права на комп’ютерну програму.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на всеукраїнській конференції: LI Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (ВНТУ, 2022).

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

1 АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану проблеми

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 [2].

Поступово, з часом технології ставали все краще і краще, і можливість реалізувати штучний інтелект з кожним роком ставало все легше і більш можливо.

Дивлячись в сьогодення можна з впевненістю сказати те, що людство вже не може жити без штучного інтелекту, адже він майже завжди навколо нас.

Ігрова індустрія з кожним роком прогресувала. Для того, щоб зробити процес гри цікавим і насиченим гравцю потрібно з кимось взаємодіяти. Самі перші ігри, такі як Spacewar!, Pong і Gotcha були розраховані на гру гравець проти гравця, але зрозуміло, що не завжди знайдеться людина, з якою можна було б пограти. Саме для цього розробили ігровий штучний інтелект, за допомогою якого гравець може грати проти або разом із комп’ютером.

У сучасних ігор майже всюди присутній штучний інтелект, в залежності від штучного інтелекту гра може ставати, як цікавішою так і нудніше, адже можна розробити такий штучний інтелект, якого не можливо бути перемогти або

навпаки він буде занадто не розумним. Саме тому створювати штучний інтелект є не легкою задачею.

Розглянемо основні види методів, що використовуються для знаходження нового знання на основі даних інформаційного сховища. Метою інтелектуальних інформаційних технологій є знаходження нового знання, що користувач може надалі застосувати для поліпшення результатів своєї діяльності. Результат моделювання – це виявлення типу відношень у даних.

Можна визначити шість методів виявлення й аналізу знань:

- класифікація;
- регресія;
- прогнозування тимчасових послідовностей (рядів);
- кластерізація;
- послідовність.

Послідовність має місце, якщо існує ланцюжок зв'язаних у часі подій [3].

У програмному додатку буде використано метод «Послідовність».

Додання штучного інтелекту до програмного додатку «Монополія» дозволяє почати гру самостійно аж з сімома ботами, які можуть легко замінити реальних гравців.

1.2 Порівняльний аналіз аналогів

Було проведено порівняння існуючих аналогів та майже всі із них можна розкласти на певні плюси та мінуси.

Monopoly online – безкоштовна онлайн гра в яку можна грати безпосередньо з самого браузеру, що є безсумнівним плюсом [4]. З мінусів те, що для того, щоб почати гру потрібно мати що-найменш 2 живих гравця. В розроблюваному ігровому додатку буде присутній штучний інтелект, за допомогою я кого буде вирішена дана проблема.

Зображення інтерфейсу додатку «Monopoly online» зображено на рисунку 1.1.

Рисунок 1.1 – Зображення головного екрану додатку «Monopoly online»

Monopoly one. В грі є штучний інтелект [5]. Але, як і в класичній монополії тут дуже сильно впливає не те як гравець грає, а те як йому пощастить. Візьмемо такий приклад, гравець має не багато коштів, але вже наступним ходом може закінчити круг і получити винагороду, але перед цим йому не пощастило потрапити на сектор ворога і тепер йому треба продати все своє майно і тим самим зменшити свої шанси на перемогу. У розроблюваному програмному додатку «Монополія» є можливість взяти кредит у ігровому банку, тим самим забезпечити себе коштами і не втратити майно.

Інтерфейс гри «Monopoly one» зображений на рисунку 1.2.

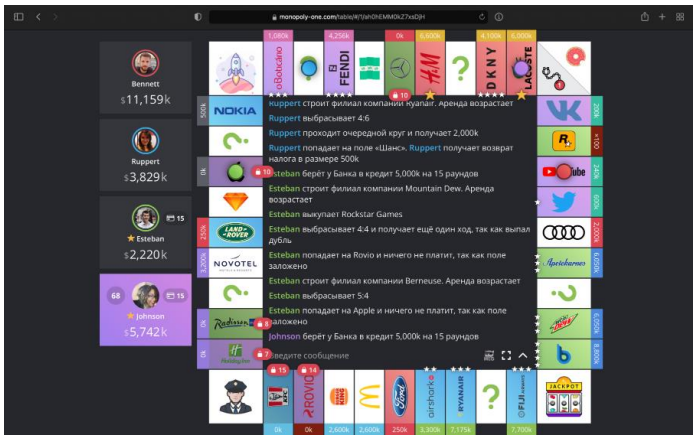


Рисунок 1.2 – Зображення інтерфейсу головного екрану додатку «Monopoly one»

Monopoly star. В цій версії гри вже присутня можливість брати кредит, що є плюсом [6]. Але, все ж таки цій грі не вистачає азарту і можливості перевернути гру в будь-який момент. Для цього у програмному додатку «Монополія» є можливість кинути «Кубик надії», якщо повезе гравця чекає винагорода, але з більшою вірогідністю гра для цього гравця може закінчитись. За допомогою цієї можливості можна навіть в самій безвихідній ситуації повернутись гру.

Інтерфейс гри «Monopoly star» зображений на рисунку 1.3.

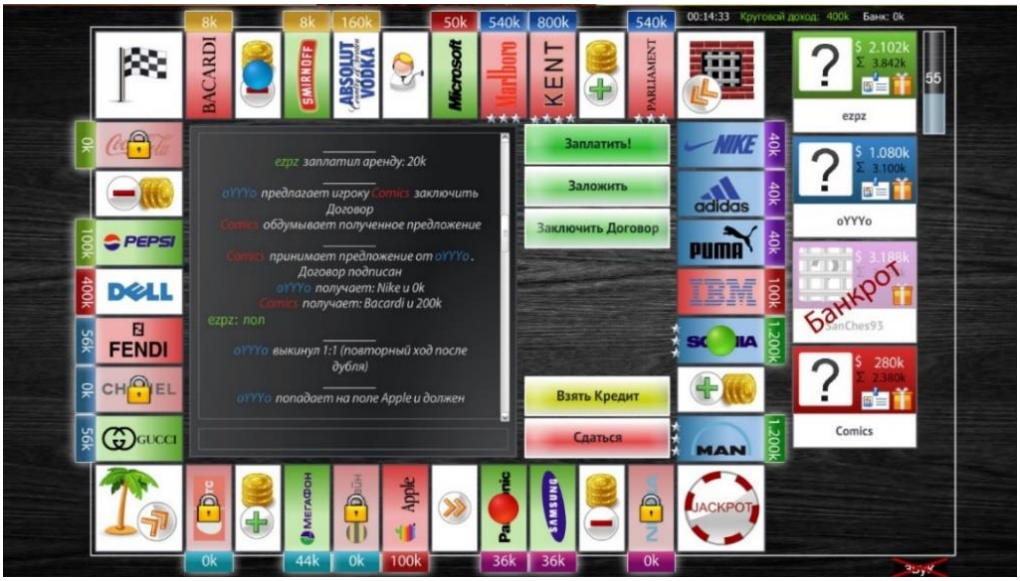


Рисунок 1.3 – Зображення інтерфейсу додатку «Monopoly star»

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

1.3 Розвиток штучного інтелекту

Розвиток штучного інтелекту можна умовно поділити на три етапи.

Перший етап. Кінець 50-х років XX ст. У цей час почалися перші дослідження у галузі штучного інтелекту. В 1956 році з’явилась перша програма зі штучним інтелектом «Логік-Теоретик», вона була створена для доведення теорем в численні висловів.

1957 рік – з’явилась програма для гри в шахи, яка надалі привела до концепції універсального вирішення задач. Ця програма добре вирішувала

головоломки типу «Ханойська вежа» та могла обчислювати невизначені інтеграли [7].

На той час було два основних методи вирішення задач.

Евристичний метод – властивий людському мисленню «взагалі», для якого характерне виникнення припущень про шлях вирішення задачі з подальшою їх перевіркою.

Алгоритмічний метод – він інтерпретувався як механічне здійснення заданої послідовності кроків, що детерміновано приводить до правильної відповіді.

У 1965 році Дж. Робінсон створив метод резолюцій, який був заснований на доведенні теорем у логіці предикатів шляхом приведення до суперечностей.

Дослідницьким полігоном для розвитку методів штучного інтелекту на першому етапі стали ігри, головоломки, математичні задачі. Вибір таких задач обумовлювався простотою і зрозумілістю проблемного середовища, її відносно малою громіздкістю, можливістю достатньо легкого підбору і навіть штучного конструювання «під метод». Розквіт таких досліджень припадає на кінець 60-х років XX ст., після чого було зроблено перші спроби використання розроблених методів для задач, розв’язуваних не в штучних, а в реальних проблемних середовищах.

Другий етап. Другий етап припадає на початок 70-тих років XX ст., у цей час відбувся якісний стрибок на дослідженнях.

Дослідники зрозуміли, що раніше створеним програмам бракує глибоких знань у відповідній природній галузі. Для цього необхідно було використовувати методи логічних міркувань і накопичені в досвіді знання, подані у символьній формі.

Тоді виникає проблема: як передати ці знання програмі, якщо її безпосередній творець ними не володіє. Тобто сама програма повинна їх виділяти з даних, одержуваних від експерта. Дослідники стикнулися з необхідністю забезпечити системи штучного інтелекту можливостями, яких немає в звичних мовах програмування. У даному випадку є розмежування між висновком про

якийсь факт і використовуванням цього факту. На противагу цьому звична мова програмування дозволяє виражати лише здійснимі завдання або вказівки.

До 1970 р. була створена безліч програм, заснованих на цих ідеях. Перша з них – програма DENDRAL. Вона призначена для породження структурних формул хімічних з'єднань на основі інформації, що поступає від мас-спектрометра.

Третій етап. Він почався з середини 70-х років XX століття. Характерною ознакою цього етапу стало зміщення центру уваги дослідників зі створення автономних функціонуючих систем, які самостійно вирішували поставлені перед ними задачі в реальному середовищі, до створення людино-машинних систем, інтегруючих в єдине ціле інтелект людини і здатності обчислювальної машини для досягнення загальної мети – вирішення задачі, поставленої перед системою.

Можна виділити три основні підходи в моделюванні штучного інтелекту.

Перший підхід. Об'єктом дослідження в даному підході є структура і механізми роботи мозку людини, а кінцева мета полягає в розкритті таємниць мислення. Необхідними етапами досліджень у цьому напрямі є побудова моделей на основі психофізіологічних даних, проведення експериментів із ними, висунення нових гіпотез щодо механізмів інтелектуальної діяльності, вдосконалення моделей.

Другий підхід. У даному підході як об'єкт дослідження розглядає штучний інтелект. Тут йдеться про моделювання інтелектуальної діяльності за допомогою обчислювальних машин. Метою робіт у цьому напрямі є створення алгоритмічного та програмного забезпечення обчислювальних машин, що дозволяє вирішувати інтелектуальні задачі не гірше за людину.

Третій підхід. Він орієнтований на створення змішаних людино-машинних, або, як ще говорять, інтерактивних інтелектуальних систем, на симбіоз можливостей природного і штучного інтелекту. Найважливішими проблемами в цих дослідженнях є оптимальний розподіл функцій між природним та штучним інтелектом і організація діалогу між людиною і машиною [8].

В розроблюваному ігровому додатку буде використаний саме другий підхід, адже потрібно створити бота, який буде повністю імітувати дії людини і буде вирішувати проблеми не гірше за людину гравця.

Метод вирішення задач був обраний – алгоритмічний. Для бота буде задана певна послідовність кроків, яку він буде виконувати на своєму ході.

1.4 Постановка задачі розробки

Оскільки завданням є розробка програмного додатку гра «Монополія» з елементами штучного інтелекту задачами розробки є:

- розробити алгоритм створення об’єктів гри;
- розробити алгоритм поведінки живих гравців;
- розробити алгоритм поведінки гравців зі штучним інтелектом;
- розробити взаємодію між гравцями;
- створити логістику постачання ігрової валюти гравцям;
- розробити алгоритм завершення гри.

1.5 Висновки

У цьому розділі було проаналізовано проблеми і обґрунтовано, як створюваний програмний додаток вирішує їх.

В результаті аналізу вищенаведених аналогів, було виявлено те, що у всіх є свої мінуси, які вирішуються у розроблюваному програмному додатку «Монополія».

Було розглянуто історію розвитку штучного інтелекту, обрано підхід і метод розробки бота для розроблюваного ігрового додатку.

2 РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Обґрунтування вибору інтерфейсу

Для програмного додатку потрібно розробити інтерфейс. Існує декілька видів інтерфейсів розглянемо деякі з них:

1. Інтерфейс командного рядка (рис 2.1) – різновид текстового інтерфейсу користувача й комп'ютера, в якому інструкції комп'ютеру можна дати тільки введенням із клавіатури текстових рядків (команд). Також відомий під назвою консоль. Інтерфейс командного рядка може бути протиставлений системам управління програмою на основі меню чи різних реалізацій графічного інтерфейсу. Формат виводу інформації в інтерфейсі командного рядка не регламентується; звичайно це простий текстовий вивід, але може бути й графічним, звуковим виводом тощо [9].

Зазвичай такий вид інтерфейсу використовується в системах з обмеженими ресурсами.

В заданому ПЗ не виникає проблем з використанням ресурсів в системі. Також ПЗ повинно бути забражено графічно, тому інтерфейс командного рядка не підходить.

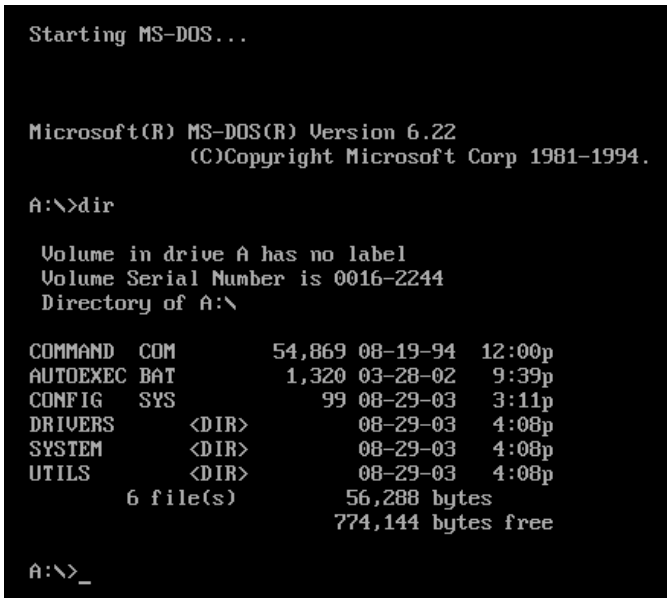


Рисунок 2.1 – Командний рядок в MS-DOS

2. Веб-інтерфейси (рис 2.2) – це сукупність засобів, за допомогою яких користувач взаємодіє з веб-сайтом або веб-застосунком через браузер. Веб-інтерфейси отримали широке поширення у зв'язку зі зростанням популярності всесвітньої павутини і відповідно повсюдного розповсюдження веб-браузерів [10].

Для використання веб-інтерфейсів завжди повинен бути доступ до інтернету, що є доволі великим мінусом даного виду інтерфейсу, він також не підходить.

3. Графічний інтерфейс користувача (рис 2.3) – тип інтерфейсу, який дає змогу користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки, на відміну від текстових інтерфейсів, заснованих на використанні тексту, текстовому наборі команд та текстовій навігації [11].

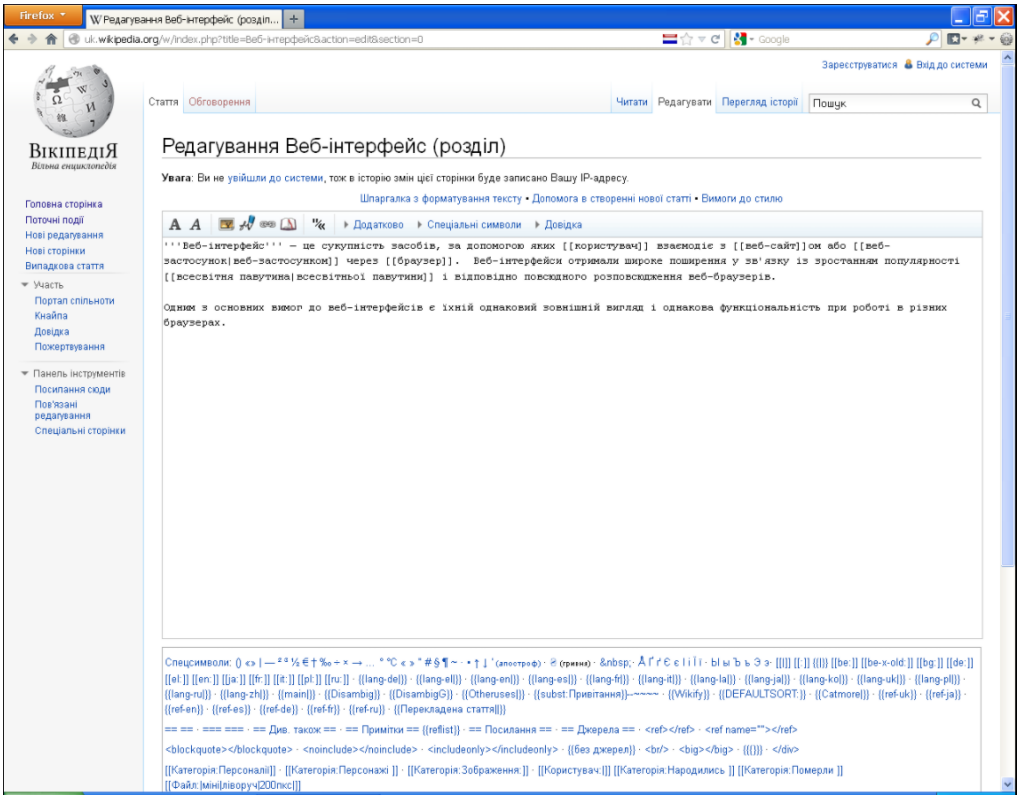


Рисунок 2.2 – Приклад веб-інтерфейсу: інтерфейс редагування вікі-тексту

Графічний інтерфейс користувача задовольняє всі вимоги до поставленого ПЗ.

Рисунок 2.3 – Графічний інтерфейс програми Gnome

Серед наведених видів інтерфейсів було прийнято рішення обрати саме графічний інтерфейс користувача. Перевага графічного інтерфейсу над текстовим в більш високому фунціоналі та більш простішому в користуванні, що необхідно для якісного продукту. Також, не потрібно буде мати доступ до інтернету, що є великим плюсом, якого нема в веб-інтерфейсах.

2.2 Опис інтерфейсу

Інтерфейс програмного додатку складається з трьох основних вікон.

На рисунку 2.4 зображена графічна схема інтерфейсу «Головного меню».

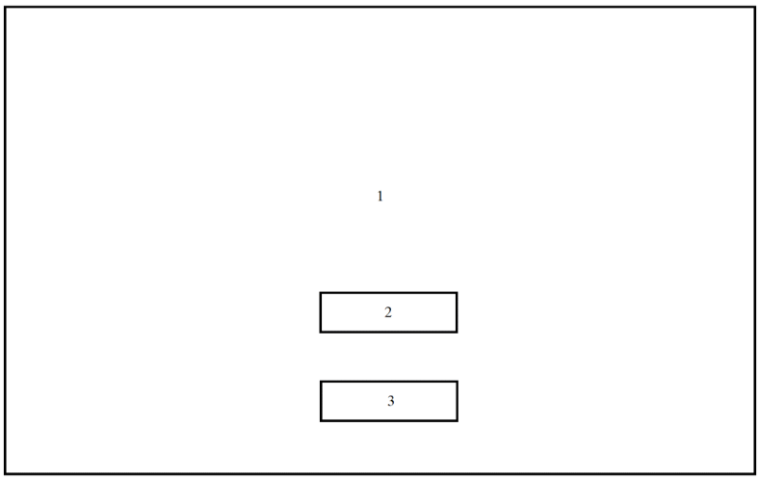


Рисунок 2.4 – Графічний інтерфейс головного меню

Елементи головного меню :

1. Головне вікно із зображенням логотипу.
2. Кнопка «Game».
3. Кнопка «Exit».

На рисунку 2.5 зображена графічна схема інтерфейсу «Меню вибору гравців».

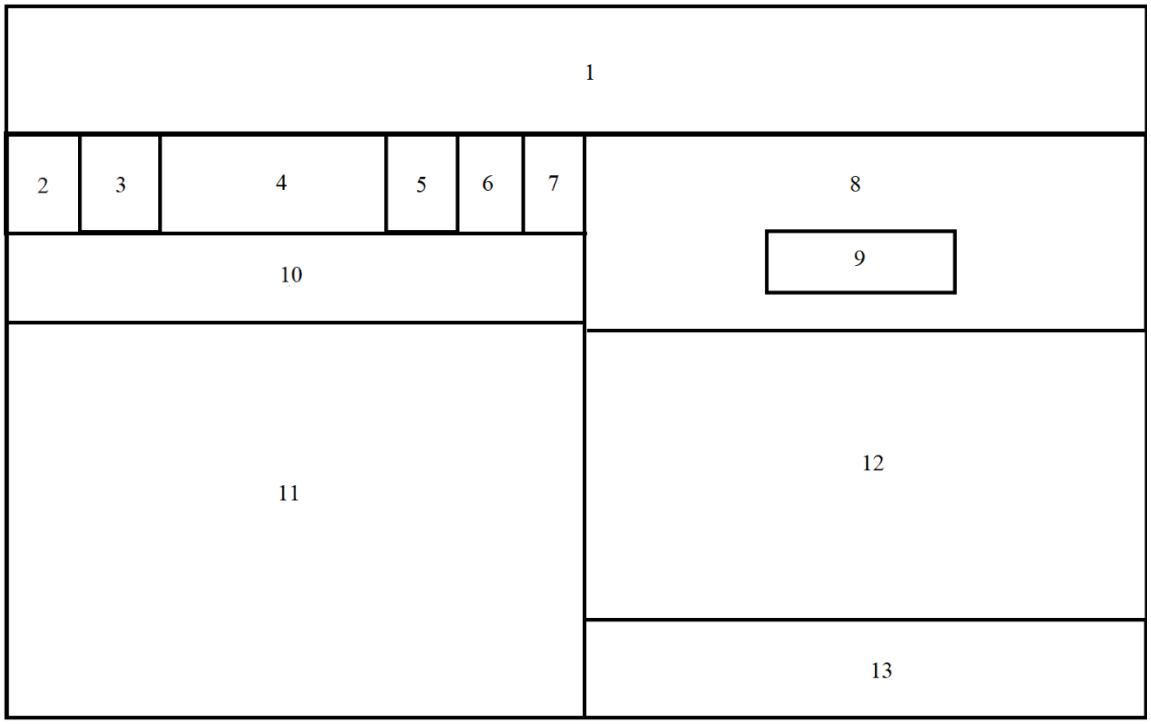


Рисунок 2.5 – Графічний інтерфейс меню вибору гравців

Елементи меню вибору гравців :

1. Надпис «New Game».
2. Початок стовпця «№».
3. Початок стовпця «Портрет».
4. Початок стовпця «Ім’я».
5. Початок стовпця «Колір».
6. Початок стовпця «Людина/Бот».
7. Початок стовпця «Видалити».
8. Поле з вибором грошей.
9. Зміна кількості грошей.

10. Кнопка «Створити».
11. Поле для нових гравців.
12. Поле для майбутніх опцій.
13. Кнопка «Почати гру».

На рисунку 2.6 зображена графічна схема інтерфейсу самої гри «Гра».

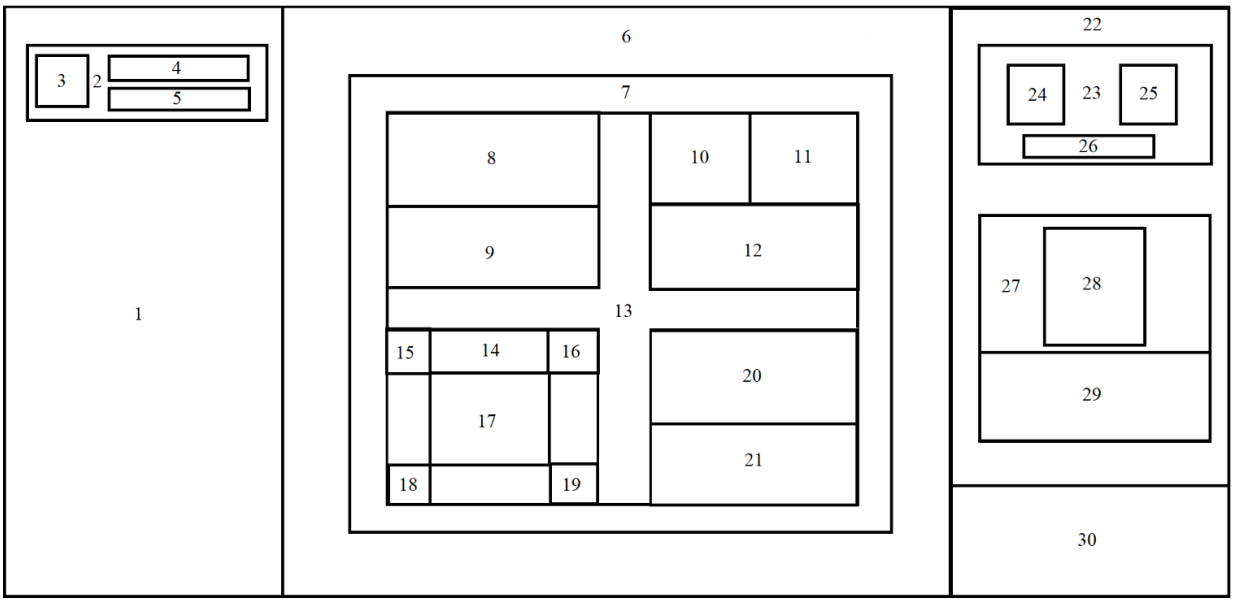


Рисунок 2.6 – Графічний інтерфейс гри

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.

2.3 Розробка діаграм

Для ігрового додатку було створено дві UML діаграми. UML – це мова моделювання, яка використовується у ООП парадигмі та є невід'ємною частиною процесу розробки програмного забезпечення. UML був створений для визначення, візуалізації, проєктування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація [12].

Існує чимало різноманітних UML діаграм для різного застосування. На рисунку 2.7 зображена схема усіх можливих діаграм.

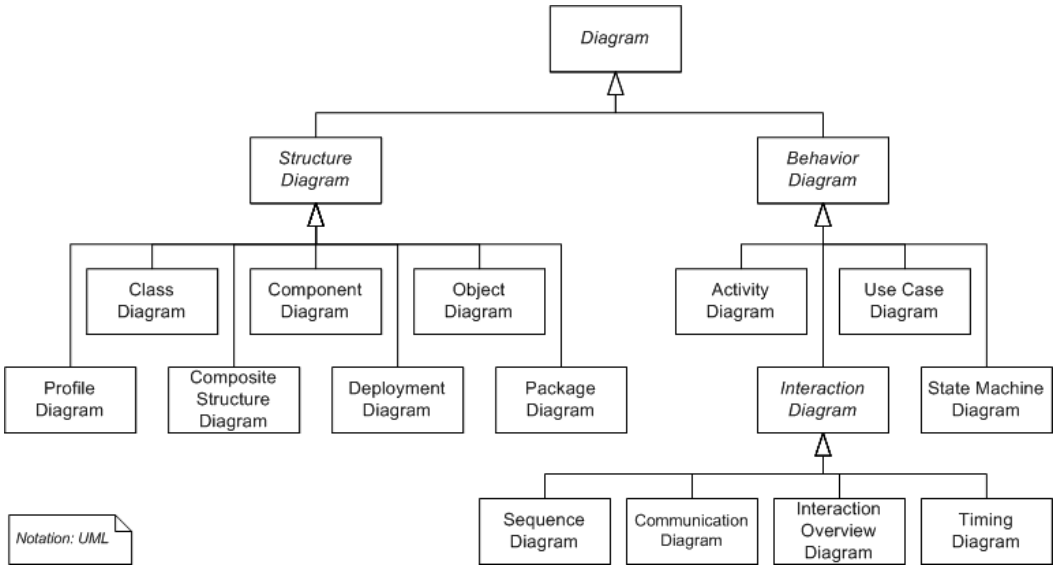


Рисунок 2.7 – Класифікація UML діаграм

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

Діаграма прецедентів зображена на рисунку 2.8. В ній можна побачити, що гравець може обрати параметри для гри, такі як: кількість гравців та кількість коштів. Окрім того, він може почати грати і тоді в нього з’явиться можливість: взаємодіяти з секторами, з іншими гравцями, з центральним меню та зі штучним інтелектом, або ботами. Після вибору гравцем тієї чи іншої опції система оброблює дані передані гравцем.

Рисунок 2.8 – Діаграма прецедентів

Діаграма послідовностей зображена на рисунку 2.9. 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.

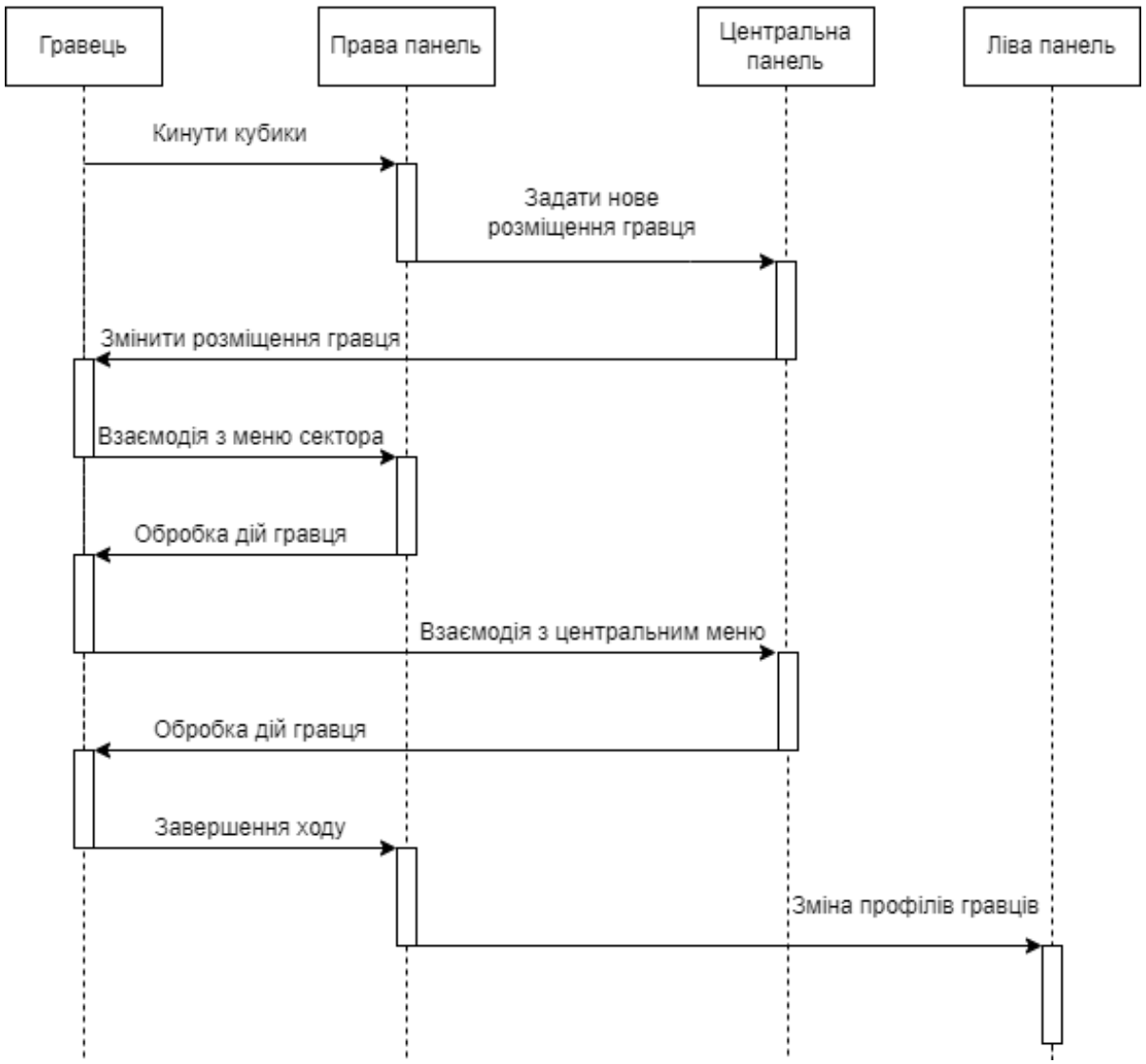


Рисунок 2.9 – Діаграма послідовностей

2.4 Розробка блок-схем алгоритмів роботи програми

Для того, щоб розпочати розробку додатку спочатку необхідно розробити алгоритми роботи програми за допомогою яких буде написаний код програми. Для зображення алгоритму було обрано блок-схеми.

Блок-схема – поширений тип схем, що описують алгоритми чи процеси, у

яких окремі кроки зображуються як блоків різної форми, з'єднаних між собою лініями, вказують напрям послідовності.

Програмний додаток гра «Монополія» з елементами штучного інтелекту складається з трьох частин, або сцен. Отже, для кожної сцени потрібно розробити власну блок-схему.

Перша сцена – це «Головне меню». На цій сцені у користувача є два варіанти, або почати гру, або вийти з неї. Блок-схема головного меню зображена на малюнку 2.10.

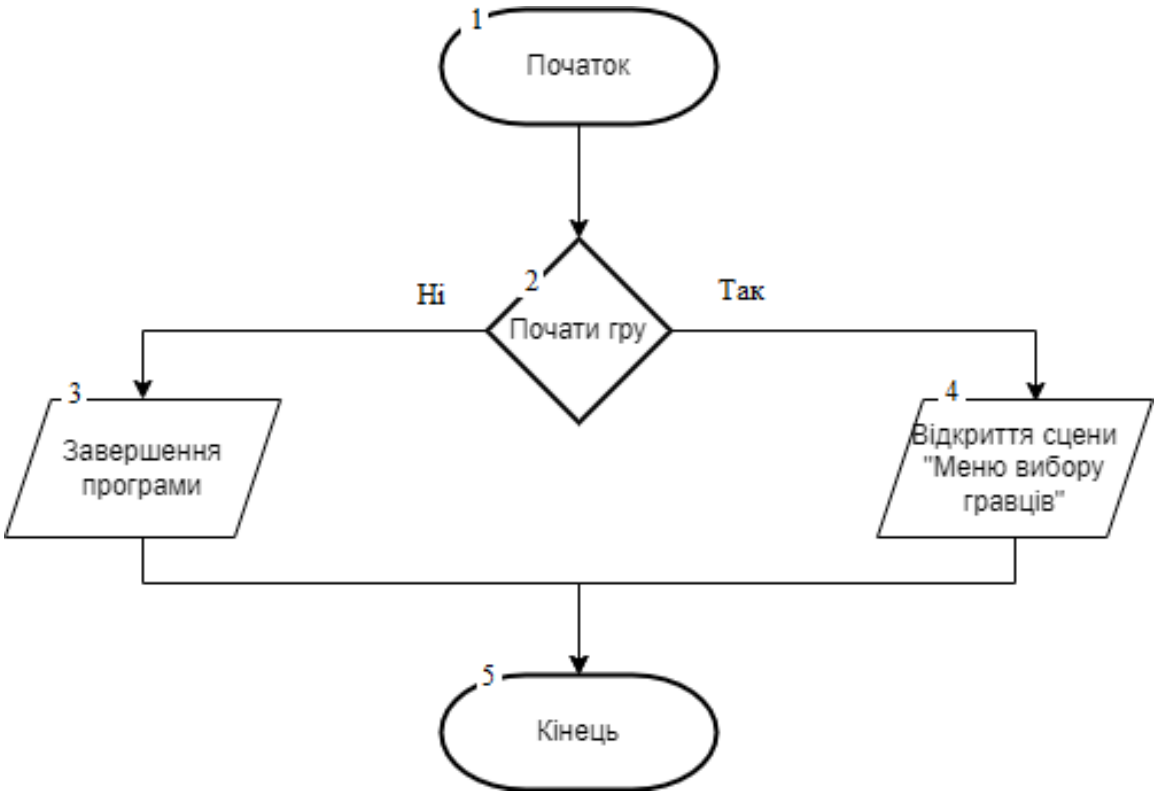


Рисунок 2.10 – Алгоритм головного меню

Друга сцена – це «Меню вибору гравців». В цьому меню користувач може обрати кількість гравців та їх характеристики, а саме: портрет, ім’я, колір та чи є цей гравець людиною або ботом. Окрім того, для початку гри потрібно ввести кількість початкових грошей у кожного гравця. Також, в цьому меню є можливість закінчити гру і вийти з програми. Алгоритм роботи «Меню вибору гравців» зображено на рисунку 2.11.

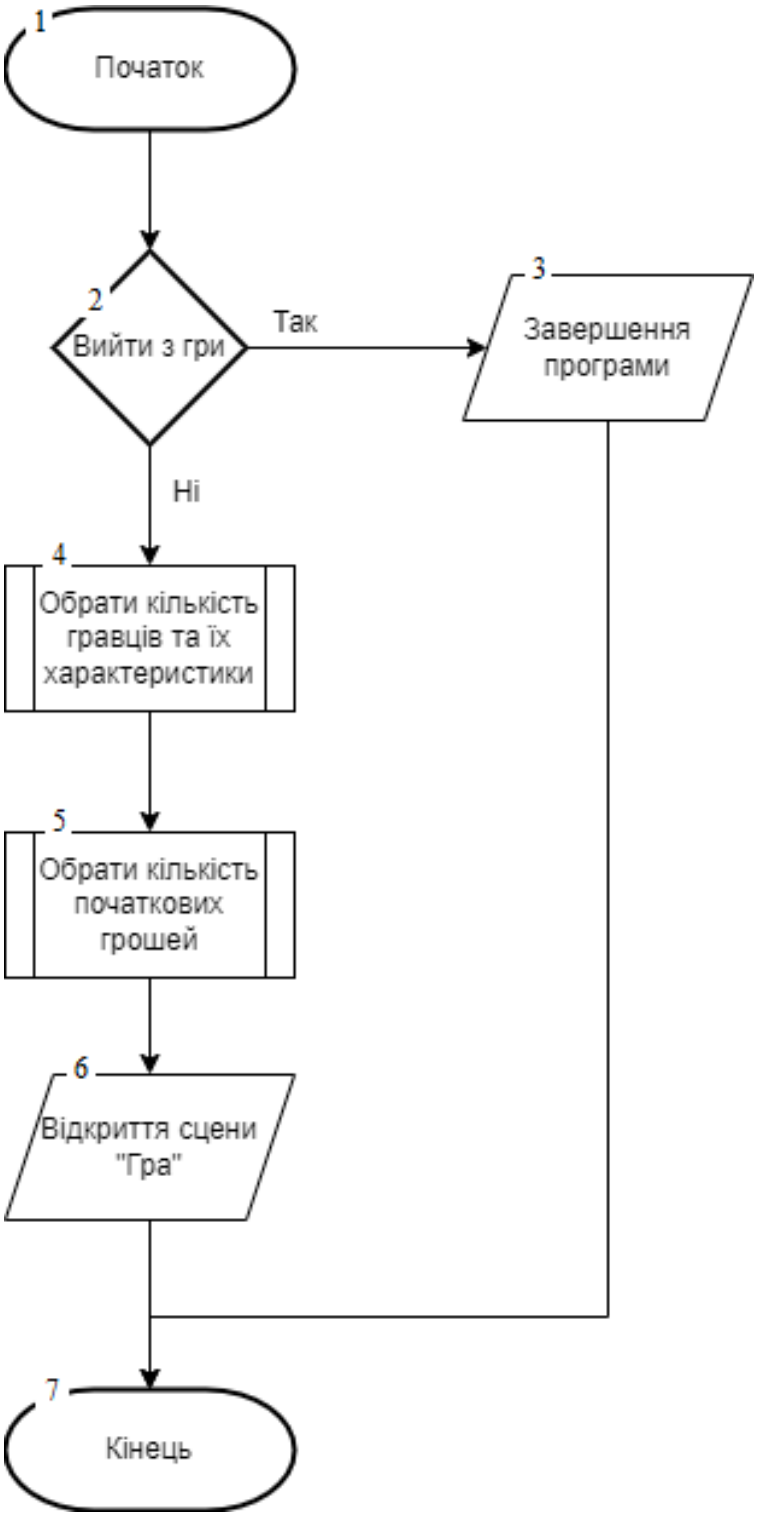


Рисунок 2.11 – Алгоритм меню вибору гравців

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 2.12 та 2.13.

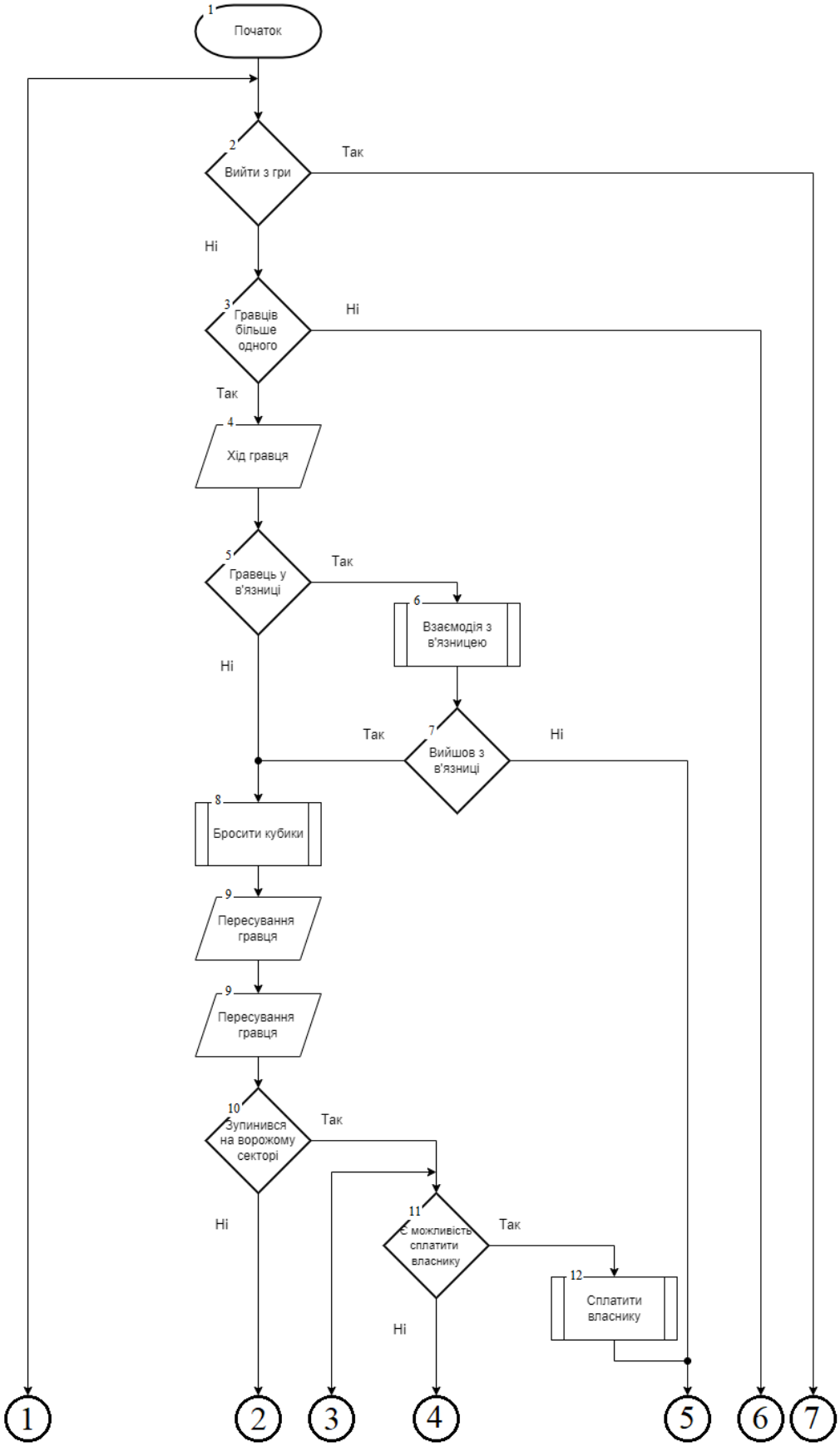


Рисунок 2.12 – Основний алгоритм роботи сцени «Гра»

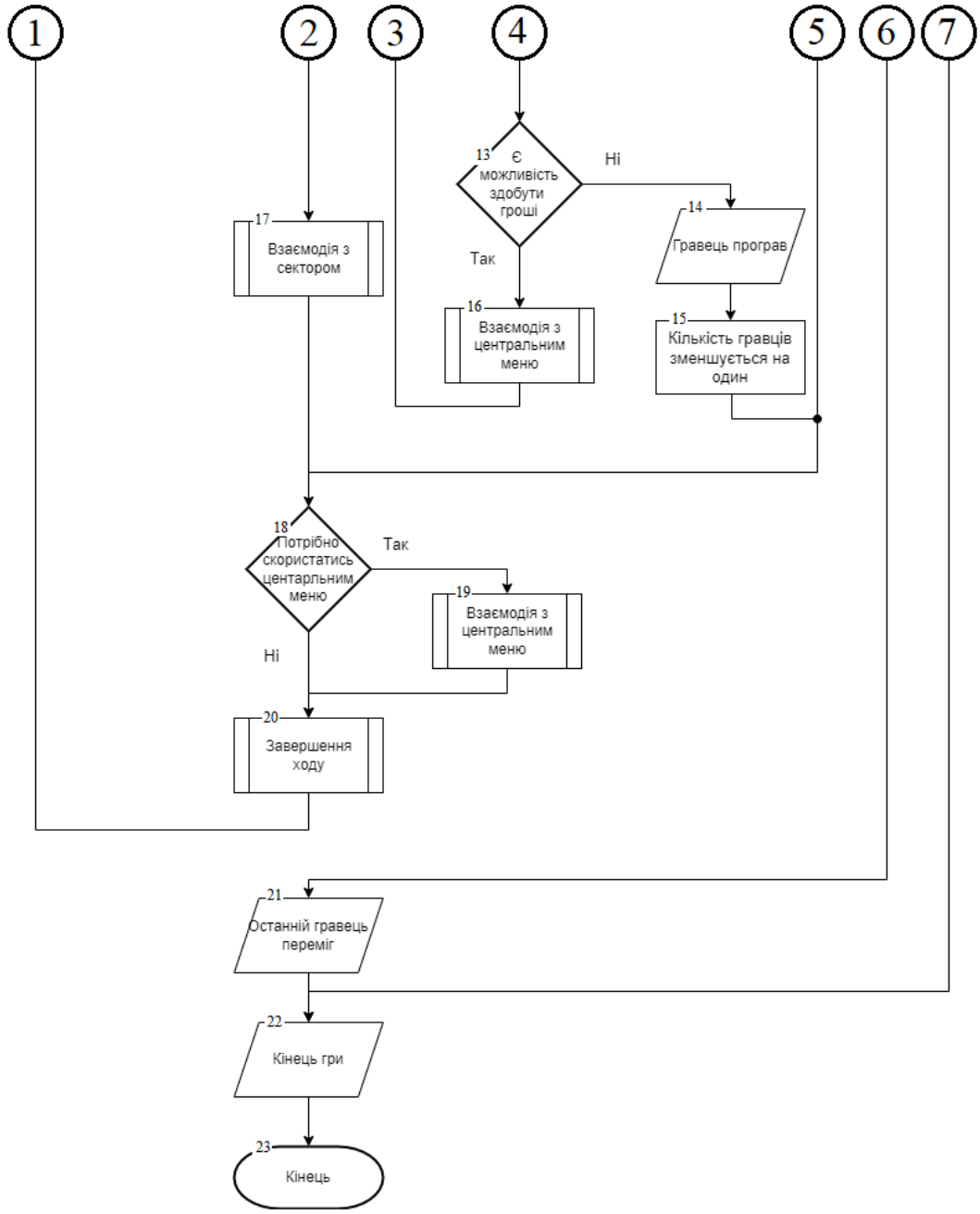


Рисунок 2.13 – Основний алгоритм роботи сцени «Гра»

2.5 Розробка модуля штучного інтелекту

Важливою частиною гри є штучний інтелект. Алгоритм роботи бота зображений на рисунку 2.14.

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:

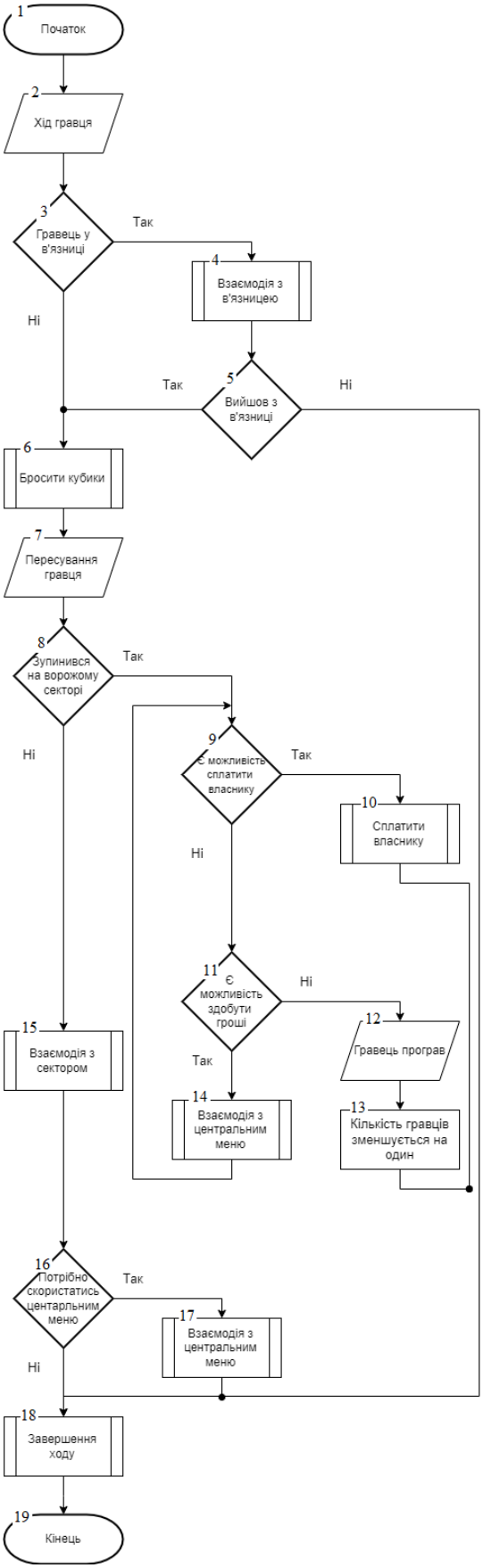


Рисунок 2.14 – Алгоритм бота

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».

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.

2.6 Висновки

В цьому розділі було розглянуто основні можливі інтерфейси для програмного додатку, серед яких був обраний графічний інтерфейс.

Описаний графічний інтерфейс усіх головних вікон, а саме «Головне меню», «Меню вибору гравця» та «Гра».

Було розроблено UML діаграми та блок-схеми алгоритмів програми.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

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.

Програмний додаток гра «Монополія» розробляється для ОС Windows.

Перед розробкою гри потрібно спочатку обрати інструмент в якому безпосередньо буде створена гра, іншими словами потрібно обрати ігровий рушій.

Ігровий рушій (англ. Game engine) – програмний рушій, центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну сторону, дозволяє полегшити розробку гри шляхом уніфікації та систематизації її внутрішньої структури. Важливим значенням рушія є можливість створення багатоплатформових ігор (сьогодні найчастіше одночасно для ПК, PS4 та Xbox One). [13]

Основну функціональність гри зазвичай забезпечує її рушій, до якого входить рушій рендерингу («візуалізатор»), фізичний рушій, звук, система скриптів, анімація, ігровий штучний інтелект, мережевий код, керування пам'яттю, багатонитевість і граф сцени. Часто на процесі розробки можна заощадити шляхом повторного використання одного рушія гри для створення декількох різних ігор. [13]

Існує чимало ігрових рушіїв, але потрібно розглянути ті, які найбільше підходять для розв’язання поставленої задачі. Розглянемо найпопулярніші з них.

Unreal Engine – це ігровий рушій та редактор, розроблений компанією Epic Games для створення ігор та програм від мультиплатформенних ігор з мільйонними бюджетами до мобільних інді розробок. Unreal Engine працює на операційних системах Windows та macOS та розроблені в ньому проекти можуть бути розгорнуті на платформах Windows, Mac, PlayStation 4, Xbox One, iOS,

Android, HTML5 та Linux. У найпростішій формі Unreal Engine 4 є колекцією редакторів, що використовуються в різних стадіях виробництва ігор або програм [14].

Інтерфейс Unreal Engine зображено на рисунку 3.1.

Рисунок 3.1 – Інтерфейс ігрового рушія Unreal Engine

Розроблюваний програмний додаток «Монополія» – це покрокова 2D гра у жанрі «стратегія». Через це вона не потребує всього того функціоналу, якого надає Unreal Engine. Для цієї гри набагато ліпше буде використовувати ігровий рушій Unity.

Додаток Unity – це професійний ігровий рушій, за допомогою якого створюються відеоігри для різних платформ. Ним щодня користуються досвідчені розробники, та й разом з тим це один із найдоступніших інструментів для новачків. Unity має дві основні переваги над іншими передовими інструментами розробки ігор. Це дуже продуктивний візуальний робочий процес і сильна міжплатформова підтримка. [15]

Інтерфейс Unity зображено на рисунку 3.2.

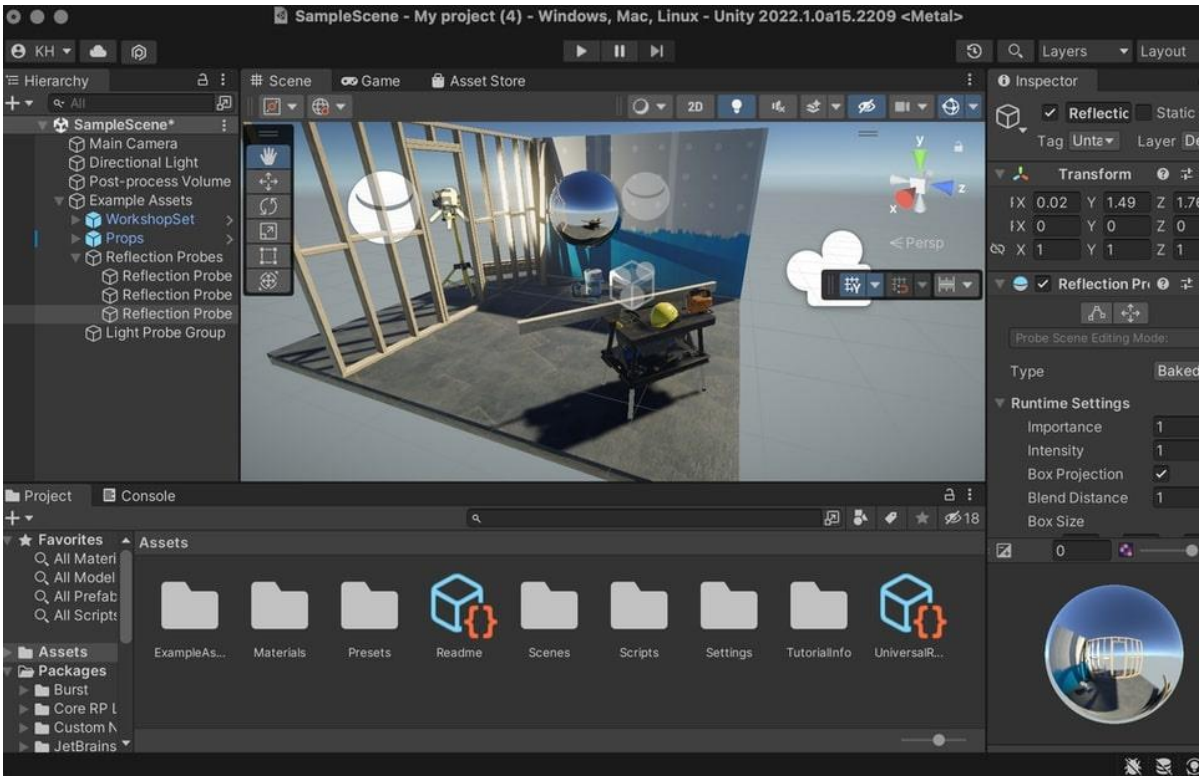


Рисунок 3.2 – Інтерфейс ігрового рушія Unity

Отже, для реалізації гри «Монополія» було обрано ігровий рушій Unity. Оскільки Unity підтримує тільки мову програмування C# то і програмний додаток буде написаний саме на цій мові.

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Мова C# є мовою програмування, за синтаксисом дуже схожим на Java (але не ідентичним йому). Однак називати C# "переробленим" варіантом Java буде неправильно. C#, як і Java, заснований на синтаксичних конструкціях C++. Так само, як і Java, C# можна називати "рафінованою" версією C++ - зрештою, це мови однієї родини [16].

Мова C++ є однією з найбільш широко поширених мов програмування. Він застосовується в багатьох прикладних сферах. Програми, написані мовою C++, можна зустріти всюди на дні океану і на поверхні Марса. Крім того, існує точний і повний загальнодоступний міжнародний стандарт мови C++, не захищений правом власності. Якісні та/або безкоштовні реалізації цієї мови доступні для будь-яких комп'ютерів [17].

Об'єктно-орієнтована мова Java, розроблена в компанії Sun Microsystems в 1995 році для пожвавлення графіки на стороні клієнта за допомогою аплетів, в даний час використовується для створення переносних на різні платформи та операційні системи програм. Мова Java знайшла широке застосування в Інтернет-додатках, додавши на статичні та клієнтські веб-сторінки динамічну графіку, покращивши інтерфейси та реалізувавши обчислювальні можливості [18].

Python є універсальною мовою програмування, яка часто виступає у ролях, пов'язаних із написанням сценаріїв. Він зазвичай визначається як об'єктно-орієнтована мова написання сценаріїв - визначення, яке комбінує підтримку ООП із загальною орієнтацією на сценарні ролі. Якщо висловитись однією рядком, то я б сказав, що Python ймовірно краще відомий як універсальна мова програмування, ніж суміш процедурної, функціональної та об'єктно-орієнтованої парадигм - формулювання, яке захоплює багатство та сфери застосування сучасної мови Python [19].

Проаналізувавши мови програмування, було складено таблицю (табл.3.1), яка демонструє відмінності Java, Python, C#, C++.

Таблиця 3.1 – Функціональні характеристики мов програмування

	Java	Python	C#	C++
Значення параметрів за замовчуванням	0	1	1	1
Функції першого класу	0	1	1	1
Динамічна типізація	0	1	1	0
Параметричний поліморфізм	1	0	1	0
Статична типізація	1	0	1	1
Ручне управління пам'яттю	0	0	1	1
Сумарний коефіцієнт	2	3	6	4

Проаналізувавши дані таблиці, C# краща за Java на 66,7% (100%- 2/6*100% = 66,7%), за Python на 50% (100%-3/6*100% = 50%) та на 33,3% (100% - 4/6*100% = 33,3%) краща ніж C++.

Отже, C# задовольняє не тільки через ігровий рушій Unity, а й через первагу над іншими мовами програмування.

3.2 Вибір середовища розробки

Оскільки було обрано ігровий рушій Unity, то обирати інтегроване середовище розробки не потрібно, адже Unity підтримує саме Visual Studio 2019.

Переваги Visual Studio 2019:

- кросплатформеність;
- надійна архітектура;
- безкоштовне програмне забезпечення.

Visual Studio 2019 – це оригінальне середовище запуску, яке дозволяє редагувати, налагоджувати і створювати код, а потім публікувати програмні додатки [20].

Крім стандартного редактора і відладчика, які існують в більшості середовищ IDE, Visual Studio включає в себе компілятори, засоби виконання коду, графічні конструктори і багато інших функцій для спрощення процесу розробки програмного забезпечення. На рисунку 3.3 представлено середовище Visual Studio з відкритим проектом і декількома вікнами основних інструментів, які вам, швидше за все, знадобляться.

- Оглядач рішень(вгорі праворуч) дозволяє переглядати файли коду, пересуватися по ньому і управляти ними.Оглядач рішеньдозволяє впорядкувати код шляхом об'єднання файлів врішення і проекти.
- Увікні редактора(центр), відображається вміст файлу.Тут ви можете редагувати код або розробляти призначений для користувача інтерфейс, наприклад вікно з кнопками або текстові поля.

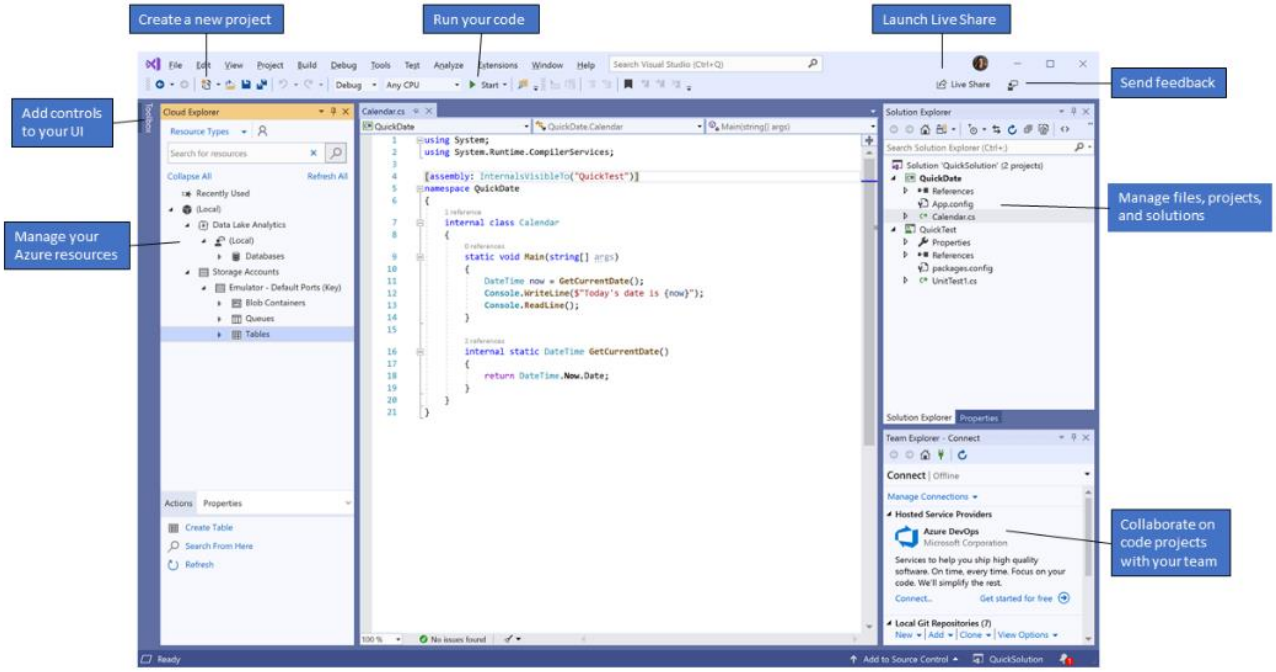


Рисунок 3.3 – Відкритий проект

- Team Explorer (правий нижній кут) дозволяє відслідковувати робочі елементи і використовувати код спільно з іншими користувачами за допомогою технологій управління версіями, таких якGiticистема управління версіями TeamFoundation (TFVC).

Отже, для реалізації програмного додатку гра «Монополія» з елементами штучного інтелекту було обрано інтегроване середовище розробки Visual studio 2019.

3.3 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 – користувач переходить зі сцени «Головне меню» до сцени «Меню вибору гравців». Рисунок 3.4 – 1 пункт.

2. Вийти з гри – користувач виходить з гри. Рисунок 3.4 – 2 пункт.

Рисунок 3.4 – Головне меню

Інструкція для сцени «Меню вибору гравців» (рис 3.5).

1. Обрати кількість гравців – для цього потрібно натиснути кнопку «Створити» (рисунок 3.5 – пункт 1). Максимальна кількість гравців – 8. Після натиснення кнопки з’являється панель з характеристиками (рисунок 3.5 – 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 3.5 – пункт 3).

3. Перейти безпосередньо до гри натиснувши кнопку «Почати гру» (рисунок 3.5 – пункт 4). Сцена «Меню вибору гравців» закривається і

відкривається сцена «Гра».

4. Вийти з гри натиснувши на червоний хрест в правому верхньому кутку екрана (рисунок 3.5 – пункт 5)..

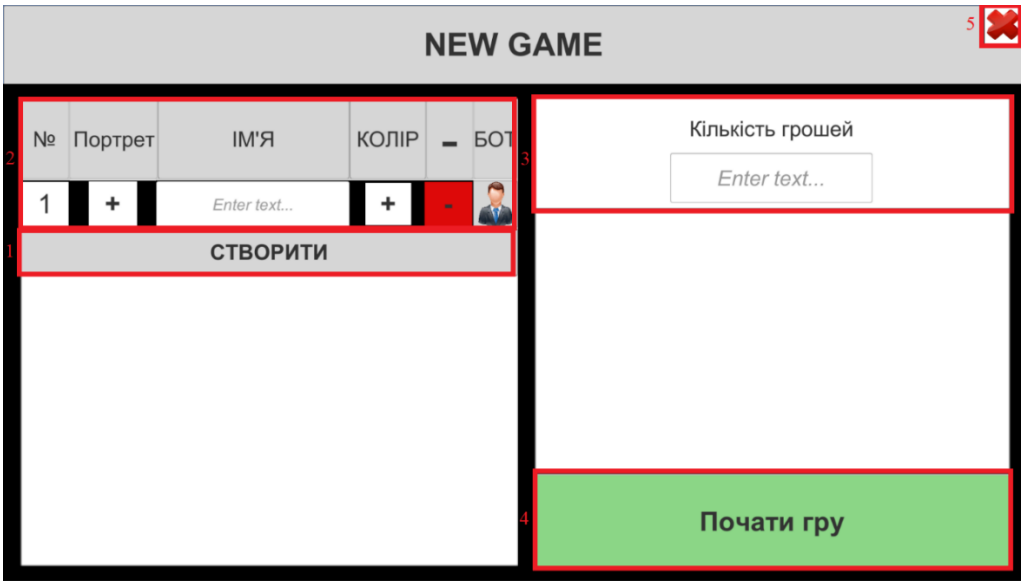


Рисунок 3.5 – Меню вибору гравців

Інструкція сцени «Гра» (рис. 3.6).

Рисунок 3.6 – Гра

1. Для того, щоб взаємодіяти з грою потрібно дочекатись свого ходу. Дізнатись про це можна за допомогою лівої панелі (рисунок 3.6 – пункт 1). Якщо ваш профіль горить зеленим – це означає, що ви можете починати ходити, якщо красним – ходить інший гравець. Черговість відбувається зверху до низу, після того, як останній (найнижчий) гравець закінчує хід (у грі це умовно називається «Кінець кола»), черга знов переходить до першого і так поки гра не закінчиться.

2. Щоб почати свій хід потрібно натиснути кнопку на правій панелі «Кинути кубики» (рисунок 3.6 – пункт 2). Після анімації гравець перейде на стільки секторів, скільки сумісно випало на кубиках.

3. Коли гравець опиниться на одному з секторів в нього з’являється меню сектору на правій панелі. Для кожного сектору своє меню (рисунок 3.6 – пункт 3). Сектори можна поділити на 8 типів:

1) «Старт». Сектор з якого починається гра. В нього нема свого меню, однак якщо після кидка кубиків гравець опинився на ньому, або пройшов повз, тому нараховується певна кількість коштів.

2) «Поле»/«Бізнес» – це сектори, які можуть бути куплені гравцем. Якщо «Поле»/«Бізнес» вільне, гравець може заплатити написану в меню сектора суму коштів натиснувши кнопку «Купити Сектор» (рис. 3.7). Якщо «Поле»/«Бізнес» куплене, гравець зобов’язаний заплатити написану в меню сектора суму коштів натиснувши кнопку «Заплатити» (рис. 3.8), якщо гравець не може заплатити він програє.

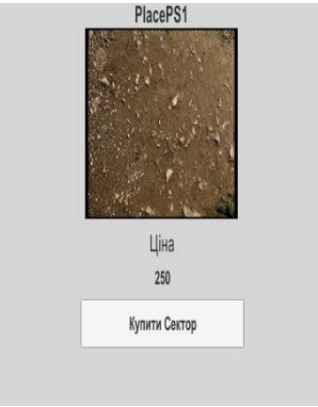


Рисунок 3.7 – Меню покупки

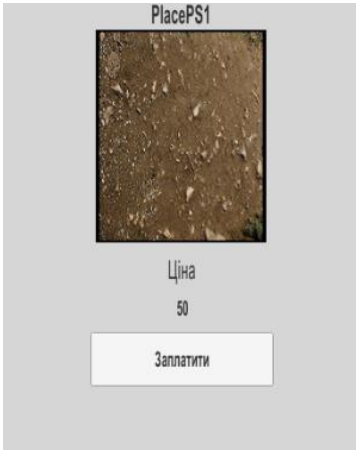


Рисунок 3.8 – Меню купленого сектора

3) «Бонус». Потрапивши на цей сектор гравець отримує винагороду, або втрачає певну кількість коштів (рис. 3.9).

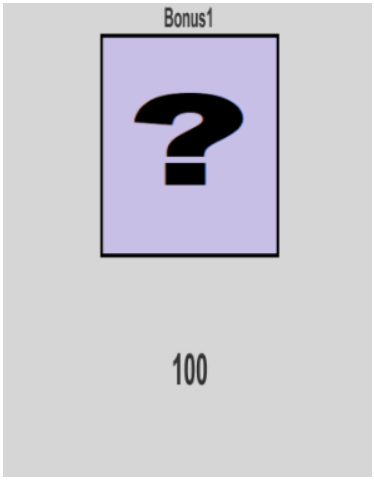


Рисунок 3.9 – Меню бонуса

4) «Зупинка». За допомогою меню цього сектору можна відправитись до іншої зупинки, при умові, що у гравця є певна кількість коштів і шлях до потрібної зупинки відкритий (рис. 3.10). На початку гри всі шляхи закриті, окрім північного. Після того, як пройде два «Кінця кола» відкриється східна зупинка, після наступних двох – південний, і після ще двох західний (під час такого переходу не нараховуються кошти за перехід сектору «Старт»).

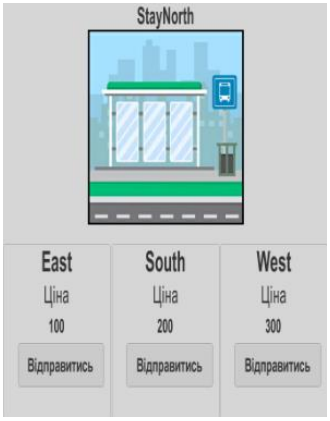


Рисунок 3.10 – Меню зупинки

5) «В’язниця». Якщо гравець опинився на цьому секторі після кидка кубиків, то нічого не відбувається, в такому разі взаємодіяти з цим сектором не можна. Якщо гравець потрапив до в’язниці через якусь подію, а саме: не виплата кредиту або якщо гравець викинув 3 рази поспіль дубль на кубиках (дублем вважається коли обидва кубики показуються однакове число), тоді з’являється меню цього сектору (рис. 3.11). Гравець не зможе ходити, поки не пробудить у «В’язниці» три хода або поки не виплатить певну суму коштів. Також у гравця є можливість викинути три дубля поспіль, і тоді гравець автоматично вийде з в’язниці, однак, якщо хоч раз не пощастить хід гравця одразу закінчується.

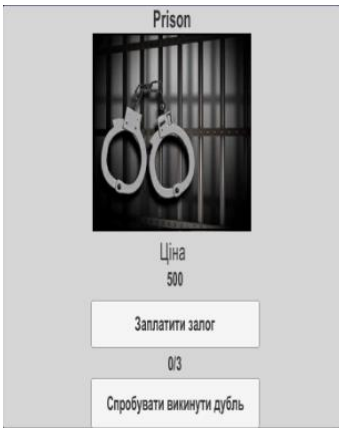


Рисунок 3.11 – Меню в’язниці

6) «Казино». На цьому секторі гравцю потрібно зробити вибір: понадіятись на удачу, або ні. Якщо ж гравець вірить у свої сили, він може обрати

певну суму коштів (яка не буде перевищувати кількість коштів гравця) і натиснути кнопку «Зіграти» (рис. 3.12). Якщо пощастить, сума подвоїться, якщо ні кошти знімуться.

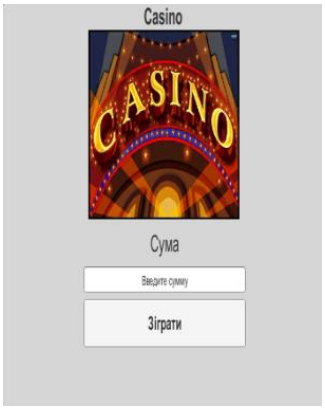


Рисунок 3.12 – Меню казино

7) «Таксі». Знаходячись на цьому секторі гравець може обрати будь-який сектор на карті (окрім сектору «Таксі»), і заплативши відповідну суму коштів перенестись до цього сектору (під час такого переходу не нараховуються кошти за перехід сектору «Старт») (рис. 3.13).

Рисунок 3.13 – Меню таксі

4. Якщо гравець не знаходиться у в’язниці він може взаємодіяти з центральним меню гри (рисунок 3.6 – пункт 4). В центральному меню є такі секції:

1) «Банк» (рис. 3.14). В банку можна взяти кредит на певну суму грошей,

при умові, що він вже не був взятий, для цього потрібно в відповідне меню ввести потрібно кількість коштів і натиснути кнопку «Взяти кредит» (рисунок 3.14 – пункт 1). В банку знаходиться відсоток, який потрібно буде виплатити (рисунок 3.14 – пункт 2). Після цього гравець отримає обрану кількість коштів і в банку з’явиться 3 строки з кількістю грошей, яку потрібно виплатити гравцю (рисунок 3.14 – пункт 3). Під час ходу на якому гравець взяв кредит і на наступному, кількість дорівнює першому дню, на 2 ходу після взяття дорівнює другому дню, і на 3 хід буде дорівнювати третьому дню. Для того, щоб виплатити кредит, потрібно натиснути кнопку «Виплатити кредит» (рисунок 3.14 – пункт 4). Якщо на третій день гравець не виплатить кредит і закінчить хід в нього заберуть ті кошти, які він повинен був виплатити, якщо в нього стільки немає продасться майно гравця поки не буде набрана потрібна сума, після чого гравець потрапить у в’язницю. Якщо навіть після продажу майна в гравця не вистачає коштів він автоматично програє.

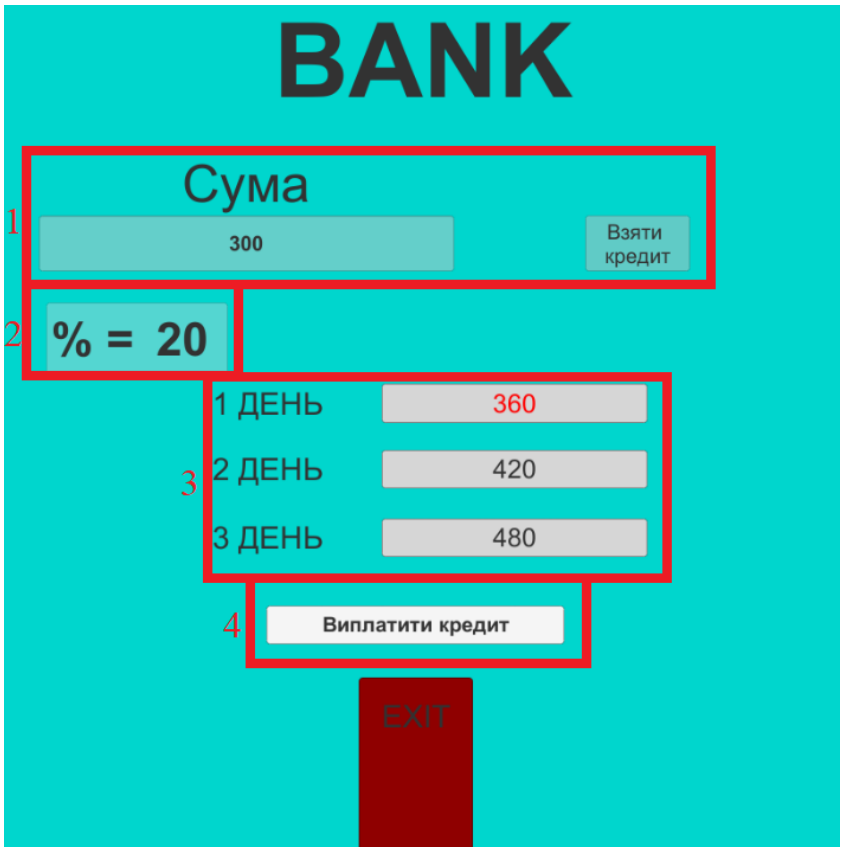


Рисунок 3.14 – Меню банку

2) «Обмін» (рис. 3.15). Під час обміну активний гравець може обрати будь-якого іншого гравця і почати з ним торгувати. Сектори і гроші активного гравця знаходяться з права, а обраного іншого гравця зліва. Щоб обрати гравця потрібно натиснути на випадаюче меню і обрати потрібного (рисунок 3.15 – пункт 1). Для того, щоб обрати сектор для обміну потрібно натиснути на його зображення після чого на ньому з'явиться зелений маркер (рисунок 3.15 – пункт 2), для відміни потрібно ще раз натиснути на зображення. Для вибору грошей, потрібно натиснути на відповідне поле зі своєї сторони і вписати потрібну кількість коштів (рисунок 3.15 – пункт 3). Після того, як гравці визначились, кожен повинен натиснути кнопку «Готовності» (рисунок 3.15 – пункт 4). Коли обидва гравця готови кнопка «Обмін» стає активною (рисунок 3.15 – пункт 5), якщо її натиснути обрані сектори перейдуть до іншого власника, а гроші кожного знімуться і додадуться відповідно до обраних.

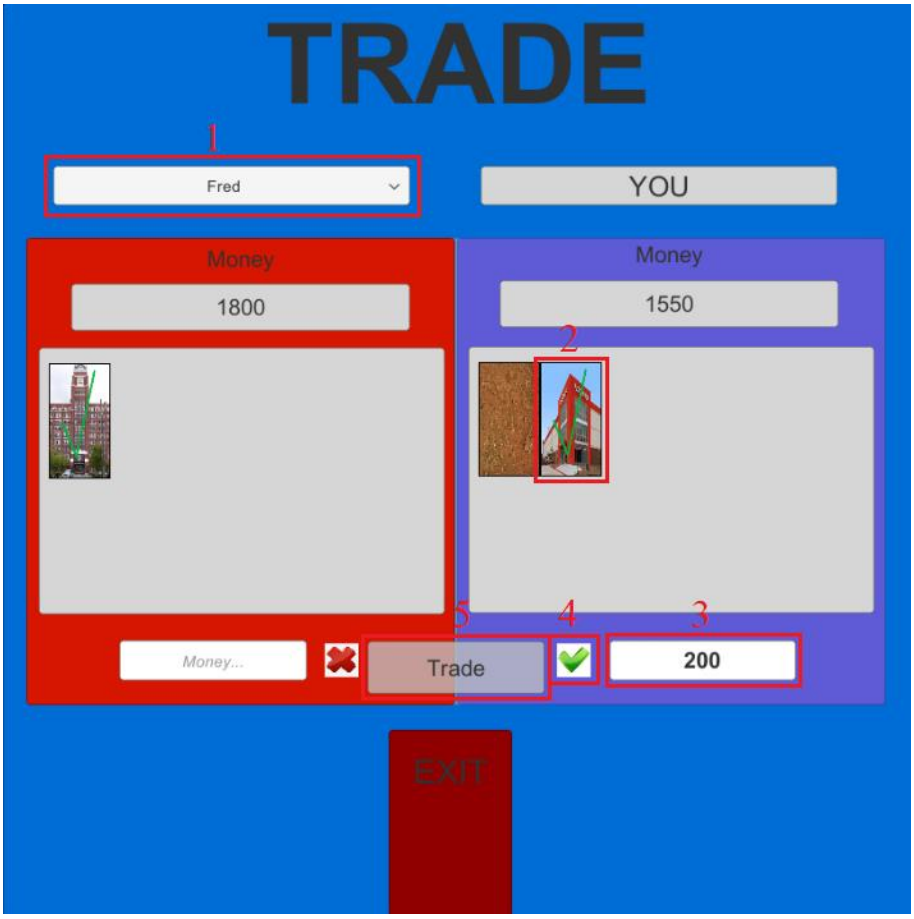


Рисунок 3.15 – Меню обміну

3) «Здатися» (рис. 3.16). Зайшовши сюди гравець може завершити гру і здатися.

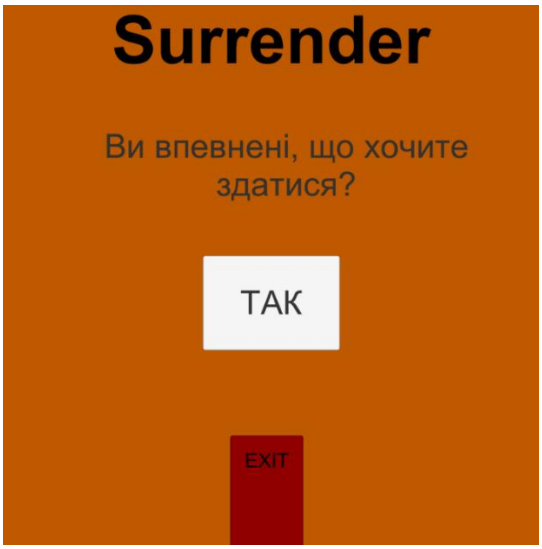


Рисунок 3.16 – Меню здатися

4) «Кубик надії» (рис 3.17). В цій секції гравець може випробувати удачу і кинути кубик надії. Відсоток на перемогу залежить від відставання від інших гравців, але завжди дуже малий. Після того, як гравець кидає кубик є лише два можливих варіанти: картинка зупиняється на черепі і гравець автоматично програє, або картинка зупиняється на $ і гравець отримує певну кількість грошей.

Рисунок 3.17 – Меню кубику надії

5) «Продаж» (рис. 3.18). Якщо гравець хоче продати куплені сектори «Поле» або «Бізнес» тут він може це зробити. Натиснувши на зображення потрібного сектору на ньому з’являється маркер (рисунок 3.18 – пункт 1), після цього не можна обрати інший сектор не забравши маркер з минулого. Далі, щоб продати потрібно натиснути кнопку «Продати» (рисунок 3.19 – пункт 2) в гравця забереться сектор і він отримає певну кількість коштів.

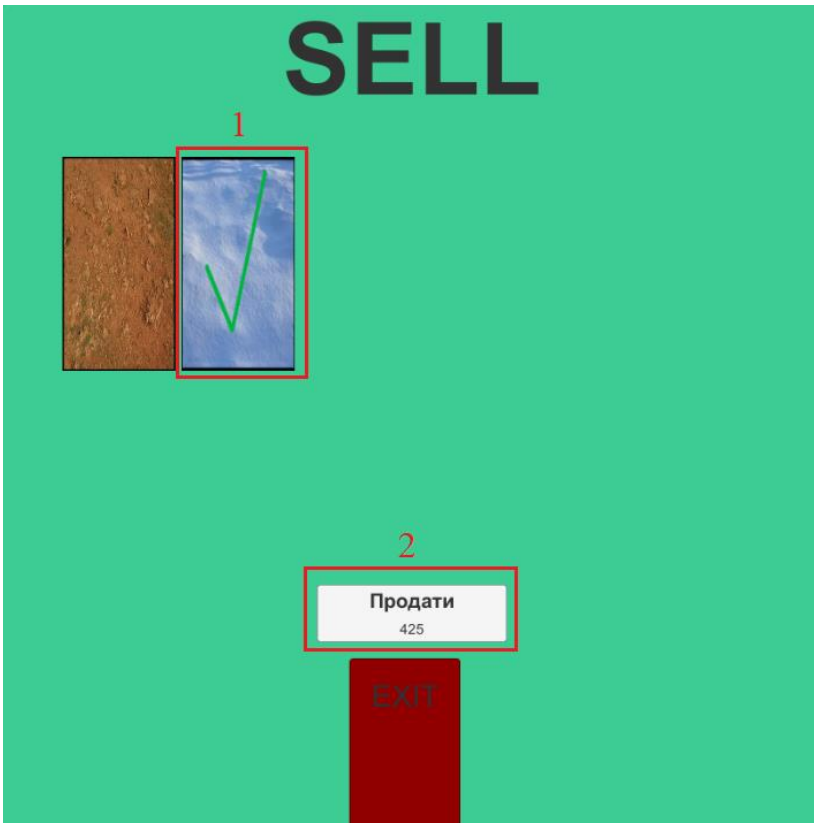


Рисунок 3.18 – Меню продажу

6) «Дома» (рис. 3.19). У грі є можливість збільшувати цінність сектору «Поле» шляхом будівництва домів на цьому секторі. Для того, щоб мати можливість будувати дома на секторі «Поле» потрібно мати всі сектори з його «Району». «Район» (рис. 3.20) – умовне згрупування трьох секторів «Поле» і одного «Бізнес» на відповідній ділянці. Якщо у гравця є «Район» після натискання кнопки «Дома» з’явиться панель з назвою району і секторами в ньому (рисунок 3.19 – пункт 1). Кожний сектор розділений на 6 рівних частин. Спочатку на всіх

частинах зображена цифра (рисунок 3.19 – пункт 2) – це скільки коштує будівництво дома на цій частині. Після натискання на одну з частин з’являється зелений дім (рисунок 3.19 – пункт 3), після повторного натискання дім пропадає. Після вибору частини на якій буде розташований дім потрібно натиснути на кнопку «Почати будівництво» (рисунок 3.19 – пункт 4) дім перестане бути зеленим і стане кольоровим (рисунок 3.19 – пункт 5), у гравця знімуться гроші. У грі сектор візуально також зміниться. Якщо гравець хоче продати дім, він може натиснути на дім, після чого той стане червоним (рисунок 3.19 – пункт 6), щоб продати потрібно натиснути кнопку «Продати дім» (рисунок 3.19 – пункт 7) після чого з сектору зникне дім і знову з’являться цифри, а гравцю буде видана певна кількість грошей.

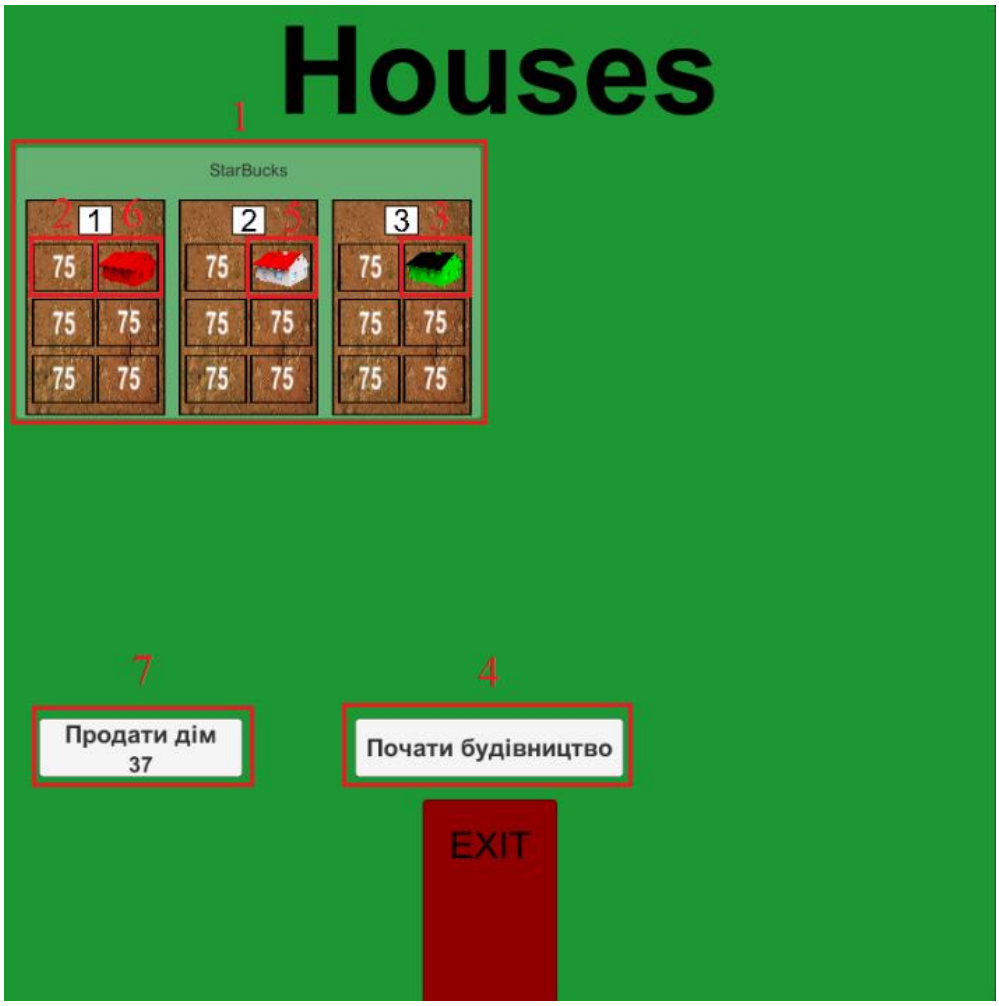


Рисунок 3.19 – Меню дома

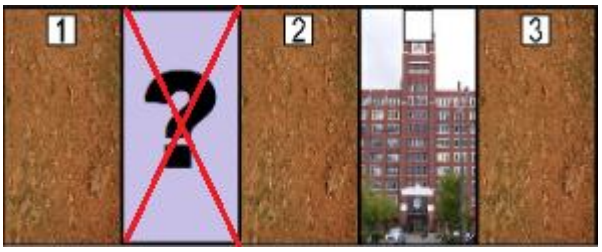


Рисунок 3.20 – Район

7) «Бізнес» (рис. 3.21). Якщо гравець володіє трьома секторами «Поле» з одного «Району» він може зайти в цю секцію і, при умові, що ніхто не володіє потрібним бізнесом, купити потрібний бізнес з будь-якої частини карти. Якщо гравець може купити бізнес він стає кольоровим (рисунок 3.21 – пункт 1), в іншому випадку бізнес сірий (рисунок 3.21 – пункт 2).

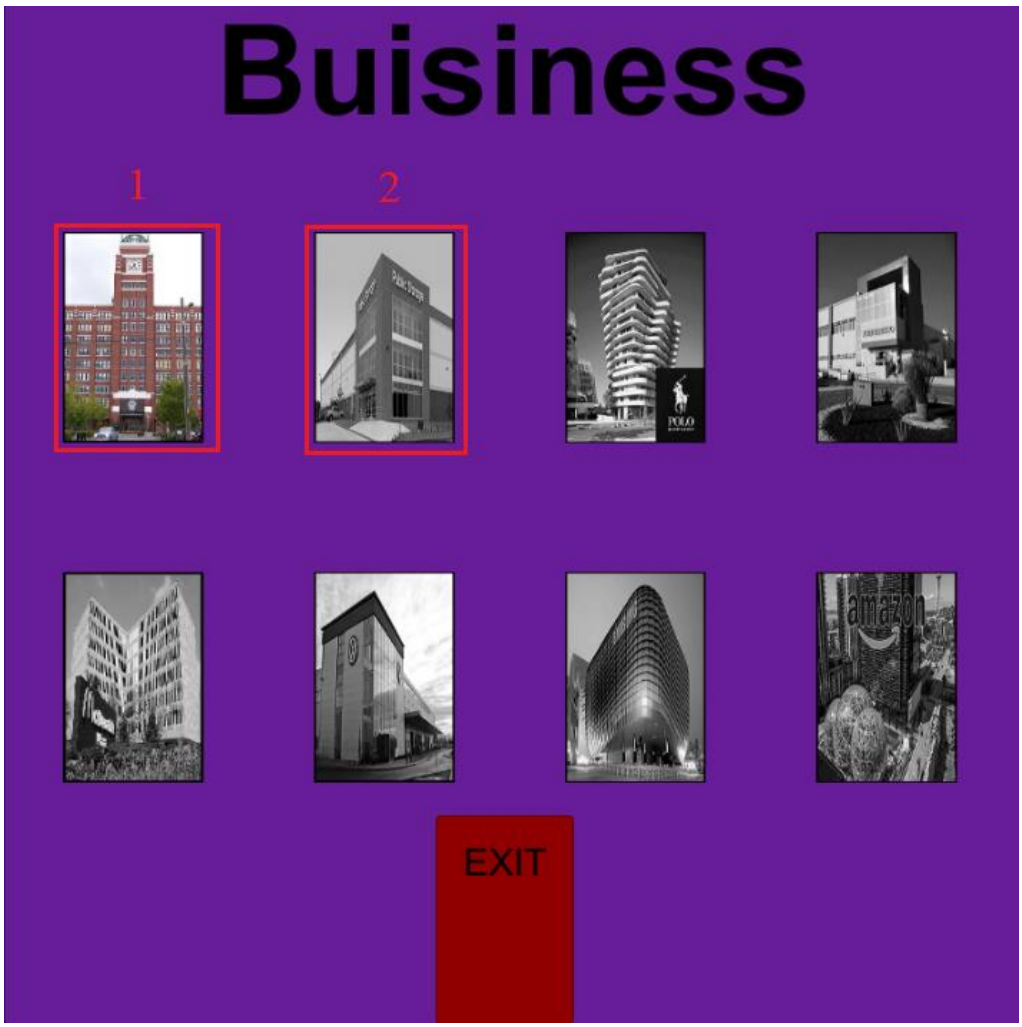


Рисунок 3.21 – Меню бізнес

5. Коли гравець закінчує всі свої справи і готовий завершити хід, йому потрібно натиснути кнопку «Завершити хід» на правій панелі (рисунок 3.6 – пункт 5). Після цього профіль гравця стане червоним і хід перейде до наступного гравця.

У грі присутня механіка «Погоди». Погода змінюється після «Кінця кола». Дізнатись про погоду можна в центральному меню (рисунок 3.6 – пункт 6). В залежності від погоди змінюється пересування гравців. У грі реалізовано 5 типів погоди, а саме:

1. Ясно – з гравцями нічого не відбувається.
2. Вітер – від суми кидка кубиків гравців віднімається 2.
3. Сонце – є шанс, що гравець піде не туди.
4. Блискавка – є шанс пройти менше ніж випало на кубиках.
5. Дощ – враховується шлях тільки з одного кубика.

3.4 Програмна реалізація

Оскільки гра складається з трьох частин розглянемо програмну реалізацію кожної частини окремо.

Перша частина. Головне меню – в ньому користувач може обрати, або почати гру, а бо вийти з неї. Якщо користувач захоче почати гру йому потрібно буде натиснути кнопку «Game» . Після чого користувач опиниться в «Меню вибору гравців». Програмна реалізація зображена на рисунку 3.22.

Друга частина. Меню вибору гравців – в цьому меню користувач може обрати кількість гравців, змінити кожному: ім’я, портрет, колір та обрати чи буде цей гравець ботом. Після цього потрібно обрати кількість початкових грошей у гравців і натиснути кнопку «Начать игру». Після цього користувач опиниться в «Грі». Програмна реалізація зображена на рисунку 3.23.

Третя частина. Гра – тут безпосередньо відбувається гра.

```
//Main menu - Start Button

public void StartGame()
{
    SceneManager.LoadScene("ChangeMenu");
}
```

Рисунок 3.22 – Програмна реалізація переходу до «Меню вибору гравців»

В цьому методі програма закриває сцену «Головне меню» та відкриває сцену «Меню вибору гравців».

```
//Start Game - Button, Save global parameters
Ссылок: 0
public void StartGame()
{
    //Save money
    PlayerPrefs.SetInt("Money", int.Parse(money.text));
    //Save max players
    PlayerPrefs.SetInt("MaxPlayers", playersCount);
    //Need, because new create panel not created (Need to fix)
    int counter;
    if (characters.Count != 8)
    {
        counter = characters.Count-1;
    }
    else
    {
        counter = 8;
    }
    //Save player parameters
    for(int i = 0; i < counter; i++)
    {
        PlayerPrefs.SetString("NamePlayer" + i, characters[i].GetComponent<CharacterMenu>().namePlayer);
        PlayerPrefs.SetString("PortraitPlayer" + i, characters[i].GetComponent<CharacterMenu>().image.sprite.name);
        PlayerPrefs.SetString("BotPlayer" + i, characters[i].GetComponent<CharacterMenu>().isBot.ToString());
        PlayerPrefs.SetString("ColorPlayer" + i, characters[i].GetComponent<CharacterMenu>().color.color.r + ";"
            + characters[i].GetComponent<CharacterMenu>().color.color.g + ";" + characters[i].GetComponent<CharacterMenu>().color.color.b);
    }
    //Load Game
    SceneManager.LoadScene("Game");
}
```

Рисунок 3.23 – Програмна реалізація переходу до «Гри»

Цей метод зберігає інформацію яку вносив користувач щодо гравців і відкриває сцену з грою.

Після переходу до сцени «Гра» відбувається завантаження інформації, яку надав користувач у «Меню вибору гравців» і зберігається у головному скрипті Info. Метод завантаження зображений на рисунку 3.24. Також, разом з цим за допомогою скрипту Markup створюються об’єкти секторів, які розміщуються навколо центральної панелі (див розділ 2.2); гравців, які розміщуються на секторі «Start»; профілів гравців, вони розміщуються на лівій панелі. Реалізація розміщення об’єктів зображена на рисунках 3.25, 3.26, 3.27.

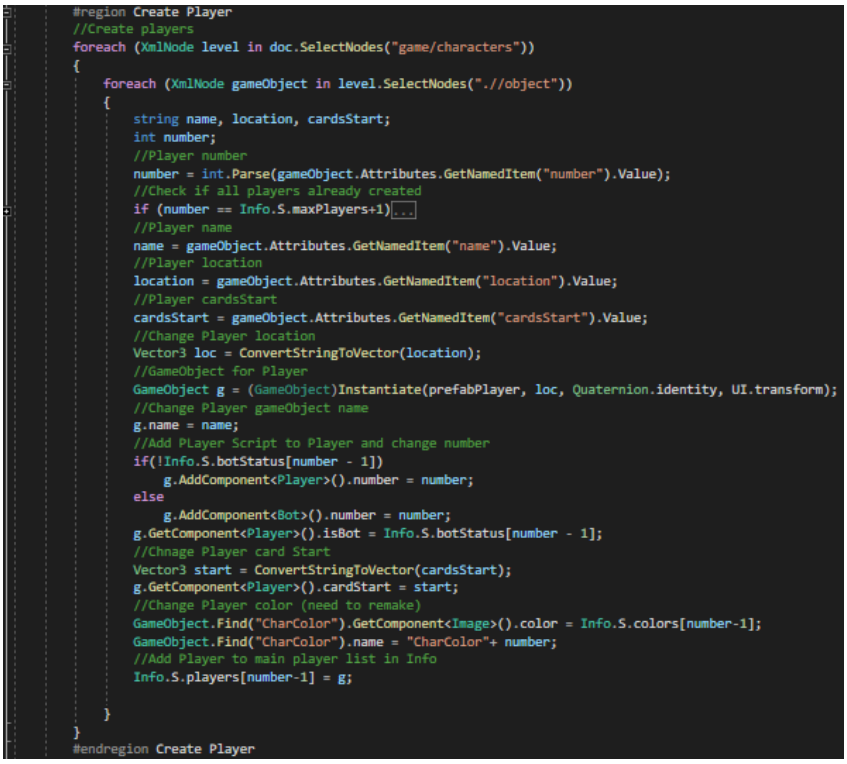
```
//Initial parameters
@ Сообщение Unity | Ссылок: 0
private void Awake()
{
    S = this;
    numberHod = 1;
    maxPlayers = PlayerPrefs.GetInt("MaxPlayers");
    money = PlayerPrefs.GetInt("Money");
    for(int i = 0; i < maxPlayers; i++)
    {
        names[i] = PlayerPrefs.GetString("NamePlayer" + 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>();
}
```

Рисунок 3.24 – Програмна реалізація завантаження інформації про гравців

```
//Create cards
#region CreateCards
foreach (XmlNode level in doc.SelectNodes("game/maintable"))
{
    foreach (XmlNode gameObject in level.SelectNodes(".//object"))
    {
        string name, location, rotation, scale, sprite, playerLoc, type;
        int number, money;
        bool flag;
        //Card name
        name = gameObject.Attributes.GetNamedItem("name").Value;
        //Card location
        location = gameObject.Attributes.GetNamedItem("location").Value;
        //Card rotation
        rotation = gameObject.Attributes.GetNamedItem("rotation").Value;
        //Card scale
        scale = gameObject.Attributes.GetNamedItem("scale").Value;
        //Card sprite
        sprite = gameObject.Attributes.GetNamedItem("sprite").Value;
        //Card playerPlace
        playerLoc = gameObject.Attributes.GetNamedItem("playerPlace").Value;
        //Card type
        type = gameObject.Attributes.GetNamedItem("type").Value;
        //Card number
        number = int.Parse(gameObject.Attributes.GetNamedItem("number").Value);
        //Card cost
        money = int.Parse(gameObject.Attributes.GetNamedItem("money").Value);
        //Change flag object for business and places (mb no need)
        flag = bool.Parse(gameObject.Attributes.GetNamedItem("flag").Value);
        //Create Transform for card
        Vector3 loc = ConvertStringToVector(location);
        Vector3 rot = ConvertStringToVector(rotation);
        Vector3 scl = ConvertStringToVector(scale);

        //GameObject for new card
        GameObject g;
        //Change card location/rotation (mb need to remake)
        if(gameObject.Attributes.GetNamedItem("type").Value == "Place")...
        else if(gameObject.Attributes.GetNamedItem("type").Value == "Buisiness")...
        else...
        //Change card sprite
        g.GetComponent<Image>().sprite = Resources.Load<Sprite>("Sprites/NewSprites/" + sprite);
        //Change card number
        g.GetComponent<Card>().number = number;
        //Change card playerPlace
        g.GetComponent<Card>().playerPlace = ConvertStringToVector(playerLoc);
        //Change card scale
        g.transform.localScale = scl;
        //Change card object name
        g.name = name;
        //Change card cost
        g.GetComponent<Card>().money = money;
        //Add card price
        g.GetComponent<Card>().price = (int)((float)money * 0.2f);
        //Need change MainCard prefab and delete this
        if (gameObject.Attributes.GetNamedItem("flag").Value == "false")
            g.GetComponent<Card>().flag.SetActive(false);
        //Change card object tag
        g.tag = "Card";
        //Add card to main list in Info
        Info.S.cards[number] = g;
        //Change card type
        switch(type)...
    }
}
#endregion CreateCards
```

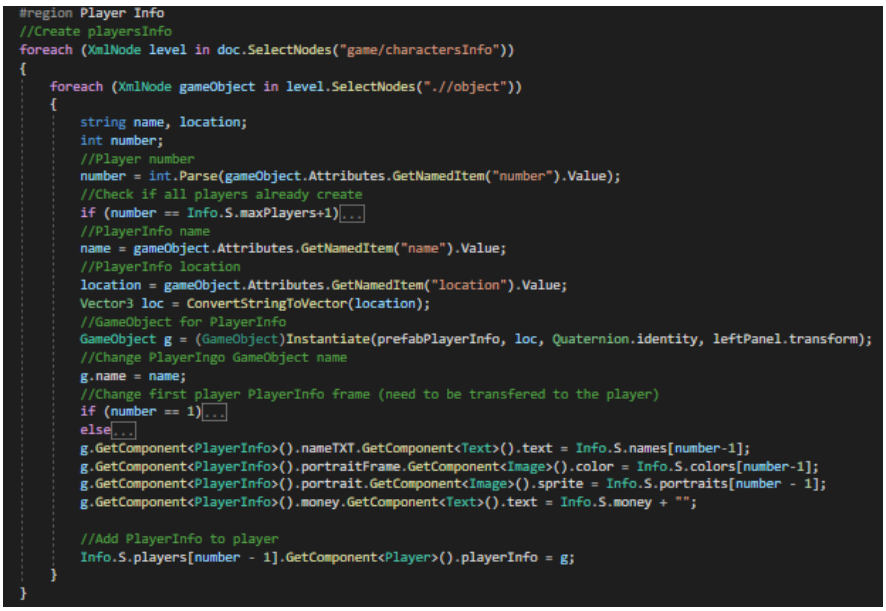
Рисунок 3.25 – Програмна реалізація розміщення секторів



```
#region Create Player
//Create players
foreach (XmlNode level in doc.SelectNodes("game/characters"))
{
    foreach (XmlNode gameObject in level.SelectNodes(".//object"))
    {
        string name, location, cardsStart;
        int number;
        //Player number
        number = int.Parse(gameObject.Attributes.GetNamedItem("number").Value);
        //Check if all players already created
        if (number == Info.S.maxPlayers+1)...
        //Player name
        name = gameObject.Attributes.GetNamedItem("name").Value;
        //Player location
        location = gameObject.Attributes.GetNamedItem("location").Value;
        //Player cardsStart
        cardsStart = gameObject.Attributes.GetNamedItem("cardsStart").Value;
        //Change Player location
        Vector3 loc = ConvertStringToVector(location);
        //GameObject for Player
        GameObject g = (GameObject)Instantiate(prefabPlayer, loc, Quaternion.identity, UI.transform);
        //Change Player gameObject name
        g.name = name;
        //Add PLayer Script to Player and change number
        if(!Info.S.botStatus[number - 1])
            g.AddComponent<Player>().number = number;
        else
            g.AddComponent<Bot>().number = number;
        g.GetComponent<Player>().isBot = Info.S.botStatus[number - 1];
        //Chnage Player card Start
        Vector3 start = ConvertStringToVector(cardsStart);
        g.GetComponent<Player>().cardStart = start;
        //Change Player color (need to remake)
        GameObject.Find("CharColor").GetComponent<Image>().color = Info.S.colors[number-1];
        GameObject.Find("CharColor").name = "CharColor"+ number;
        //Add Player to main player list in Info
        Info.S.players[number-1] = g;

    }
}
#endregion Create Player
```

Рисунок 3.26 – Програмна реалізація розміщення гравців



```
#region Player Info
//Create playersInfo
foreach (XmlNode level in doc.SelectNodes("game/charactersInfo"))
{
    foreach (XmlNode gameObject in level.SelectNodes(".//object"))
    {
        string name, location;
        int number;
        //Player number
        number = int.Parse(gameObject.Attributes.GetNamedItem("number").Value);
        //Check if all players already create
        if (number == Info.S.maxPlayers+1)...
        //PlayerInfo name
        name = gameObject.Attributes.GetNamedItem("name").Value;
        //PlayerInfo location
        location = gameObject.Attributes.GetNamedItem("location").Value;
        Vector3 loc = ConvertStringToVector(location);
        //GameObject for PlayerInfo
        GameObject g = (GameObject)Instantiate(prefabPlayerInfo, loc, Quaternion.identity, leftPanel.transform);
        //Change PlayerIngo GameObject name
        g.name = name;
        //Change first player PlayerInfo frame (need to be transfered to the player)
        if (number == 1)...
        else...
        g.GetComponent<PlayerInfo>().nameTXT.GetComponent<Text>().text = Info.S.names[number-1];
        g.GetComponent<PlayerInfo>().portraitFrame.GetComponent<Image>().color = Info.S.colors[number-1];
        g.GetComponent<PlayerInfo>().portrait.GetComponent<Image>().sprite = Info.S.portraits[number - 1];
        g.GetComponent<PlayerInfo>().money.GetComponent<Text>().text = Info.S.money + "";

        //Add PlayerInfo to player
        Info.S.players[number - 1].GetComponent<Player>().playerInfo = g;
    }
}
```

Рисунок 3.27 – Програмна реалізація розміщення профілів гравців

Однією з найголовніших функцій програмного додатку є функція завершення ходу. Алгоритм її дій наступний:

- спочатку вона перевіряє чи сплатив діючий гравець кредит (якщо він у

нього є), якщо ні і заборгованість вже 3 день, діючий гравець відправляється у в’язницю.

- Далі йде зміна профілів гравців. У неактивних гравців рамка профілю червона, а у активного – зелена.
- Після цього очищується права панель, адже вона створена саме для діючого гравця.
- Далі змінюється статус активного гравця на неактивного і наступного гравця на активного.
- Перевіряється чи сидить новий діючий гравець у в’язниці, для нього окрема логіка.
- Перевіряється чи є новий діючий гравець ботом, якщо є, то запускається скрипт у бота.

Програмна реалізація функції «Кінець ходу» зображений на рисунку 3.28.

```
ссылка: 1
public void EndTurn()
{
    Info.S.CheckPrison();
    //Change PlayerInfo - no active
    Info.S.player.GetComponent<Player>().playerInfo.GetComponent<Image>().color = new Color(255, 0, 0, 255);
    //Change CardMenu
    Info.S.cardMenu.GetComponent<CardMenu>().nameC.GetComponent<Text>().text = "";
    Info.S.cardMenu.GetComponent<CardMenu>().image.GetComponent<Image>().sprite = null;
    //Change Player
    Info.S.players[Info.S.numberHod - 1].GetComponent<Player>().hodit = false;
    Info.S.players[Info.S.numberHod - 1].GetComponent<Player>().ShowCards();
    //Transfer of the cource
    if (Info.S.numberHod != Info.S.maxPlayers)
    {
        Info.S.numberHod++;
        CheckBankrupt();
    }
    else
    {
        Info.S.numberHod = 1;
        CheckBankrupt();
        onNewTurn.Invoke();
        Info.S.turn++;
    }
    Info.S.players[Info.S.numberHod - 1].GetComponent<Player>().hodit = true;
    Info.S.players[Info.S.numberHod - 1].GetComponent<Player>().ShowCards();
    if(Info.S.players[Info.S.numberHod - 1].GetComponent<Player>().inPrison)
    {
        GameObject.Find("EndTurn").GetComponent<Button>().interactable = true;
        GameObject.Find("CubesButton").GetComponent<Button>().interactable = false;
        onCubeThrow.Invoke(10);
        GameObject.Find("SecondChance").GetComponent<Button>().interactable = true;
        Debug.Log("EndTurn");

    }
    GameObject.Find("playerInfo" + Info.S.numberHod).GetComponent<Image>().color = new Color(0, 255, 0, 255);
    Waiting.SetActive(true);
    if (Info.S.players[Info.S.numberHod - 1].GetComponent<Player>().isBot)
        StartCoroutine(BotStart());
}
```

Рисунок 3.28 – Програмна реалізація завершення ходу.

Частина програмної реалізації бота зображена рисунку 3.29.

```
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)...
            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());
```

Рисунок 3.29 – Програмна реалізація бота

3.5 Висновки

У цьому розділі було проведено аналіз ігрових рушіїв серед яких було був обраний Unity. Було порівняно декілька мов програмування, серед яких найліпшим був C#. Було ознайомлено з інтегрованим середовищем розробки Visual Studio 2019.

Була створена інструкція користувача, в якій детально розглянуто всі можливості гравця.

Також було створено алгоритм роботи програмного додатку та розглянуто головні частини програмної реалізації.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Вибір методики тестування

Існує багато методик для тестування ПЗ. Для аналізу було обрано такі методики функціонального тестування:

- тестування методом білого ящика;
- тестування методом чорного ящика;
- ручне тестування;
- юніт-тестування.

Тестування чорного ящика – спеціальний метод перевірки працездатності програмного забезпечення, при якому вся функціональність продукту досліджується без аналізу вихідного коду. Тестувальники створюють логічно зрозумілі тест-кейси, спираючись виключно на вимоги з специфікації на проекті.

Переваги:

- Дозволяє швидко знаходити баги в розробленій функціональності ПО;
- Тестувальник не обов'язково повинен володіти вузькопрофільної спеціальністю;
- Перевірка проходить з позиції кінцевого споживача;
- Розробляти тест-кейси можна відразу ж після завершення роботи зі специфікацією.

Недоліком даного методу є ризик пропустити серйозні помилки [21].

Тестування білого ящика – особливий метод перевірки ПО, який має на увазі, що внутрішня структура і технічні особливості ПО досконально відомі перевіряючому.

Переваги:

- Оптимізація програмного коду шляхом пошуку прихованих багів;
- Створення автоматизованих тест-кейсів;
- Використання найбільш підходящого виду вхідних даних, що застосовуються для якісного процесу тестування.

До недоліків можна віднести:

- Для виконання тестування білого ящика необхідна велика кількість спеціальних знань.
- При використанні автоматизації тестування на цьому рівні, підтримка тестових скриптів може виявитися досить накладної, якщо програма часто змінюється [22].

Ручне тестування – це тип тестування програмного забезпечення, при якому тестери вручну виконують тестові випадки без використання будь-яких засобів автоматизації. Ручне тестування є найбільш примітивним з усіх типів тестування і допомагає знаходити помилки в програмній системі.

До переваг ручного тестування можна віднести:

- Тестування в реальному часі. Незначні зміни можуть бути досліджені відразу, без написання коду і його виконання.
- Можливість дослідного тестування. Його метою є перевірка різноманітних можливостей програми. Важливо, що використовуються не заздалегідь складені тест-кейси, а придумані на льоту сценарії.

Недоліки методу:

- Людський фактор. Хоча UI і може бути протестований тільки вручну, люди часто схильні до неефективності. Деякі помилки можуть залишитися непоміченими.
- Неможливість навантажувального тестування. Не можна змоделювати велику кількість користувачів вручну [23].

Юніт-тестування – процес, завдяки якому можна перевірити коректність окремо виділених методів. За допомогою юніт-тестування можна легко перевірити чи не призвело нововведення до помилок в минулих частинах коду.

Переваги юніт-тестування:

- Допомагає усунути сумніви, щодо окремих частин коду.
- Дозволяє проводити рефакторинг, адже завдяки юніт-тестуванню розробник впевнений, що модуль, як і раніше, працює коректно.

Недоліки юніт-тестування:

- Погано працює зі складним кодом.
- Потрібно вести записи, щодо всіх проведених тестів, щоб при виявленні проблеми можна було подивитись, що стало причиною поламки [24].

Отже, серед вищенаведених методик тестування було обрано юніт-тестування, адже програмний додаток гра «Монополія» є не дуже великим для тестування білим або чорним ящиком, та не дуже малим, щоб можна було легко перевірити все ручним тестуванням. Також, не менш важливе є те, що в Unity є вбудована функція юніт-тестування. Саме тому було обрано юніт-тестування, за допомогою якого будуть тестуватись окремі ділянки коду.

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 [25g].

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.

Програмна реалізація тесту для методів зображені на рисунках 4.1, 4.2, 4.3, 4.4.

```
[Test]
Ссылок: 0
public void WhenBuying_AddPlayer_AssertPlayerCardsCount1()
{
    //Addrange
    Player player = new GameObject().GetComponent<Player>();
    player.SetPlayer();
    Card card = new GameObject().GetComponent<Card>();

    //Act
    player.Buy(0, true, 0, false, card);

    //Assert
    Assert.IsTrue(player.cards.Count.Equals(1));
}
```

Рисунок 4.1 – Програмна реалізація тесту для методу Buy()

```
[Test]
Ссылок: 0
public void WhenMoving_AddPlayer_AssertPlayerPlaceNumber10()
{
    //Addrange
    Player player = new GameObject().GetComponent<Player>();
    player.SetPlayer();
    Card card = new GameObject().GetComponent<Card>();
    card.number = 10;

    //Act
    player.Move(card.number);


    //Assert
    Assert.IsTrue(player.number.Equals(card.number));
}
```

Рисунок 4.2 – Програмна реалізація тесту для методу Move()

```
[Test]
Ссылок: 0
public void WhenLossPlace_AddPlayer_AssertPlayerCardsCount0()
{
    //Addrange
    Player player = new GameObject().GetComponent<Player>();
    player.SetPlayer();
    Card card = new GameObject().GetComponent<Card>();
    player.cards.Add(card);

    //Act
    player.LossPlace(card);
    //Assert
    Assert.IsTrue(player.cards.Count.Equals(0));
}
```

Рисунок 4.3 – Програмна реалізація тесту для методу LostPlace ()

```
[Test]
Ссылок: 0
public void WhenBankrupt_AddPlayer_AssertPlayerBankruptTrue()
{
    //Addrange
    Player player = new GameObject().GetComponent<Player>();
    player.SetPlayer();
    player.money = 0;

    //Act
    player.Bankrupt();
    //Assert
    Assert.IsTrue(player.bankrupt);
}
```

Рисунок 4.4 – Програмна реалізація тесту для методу Bankrupt ()

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 4.5.

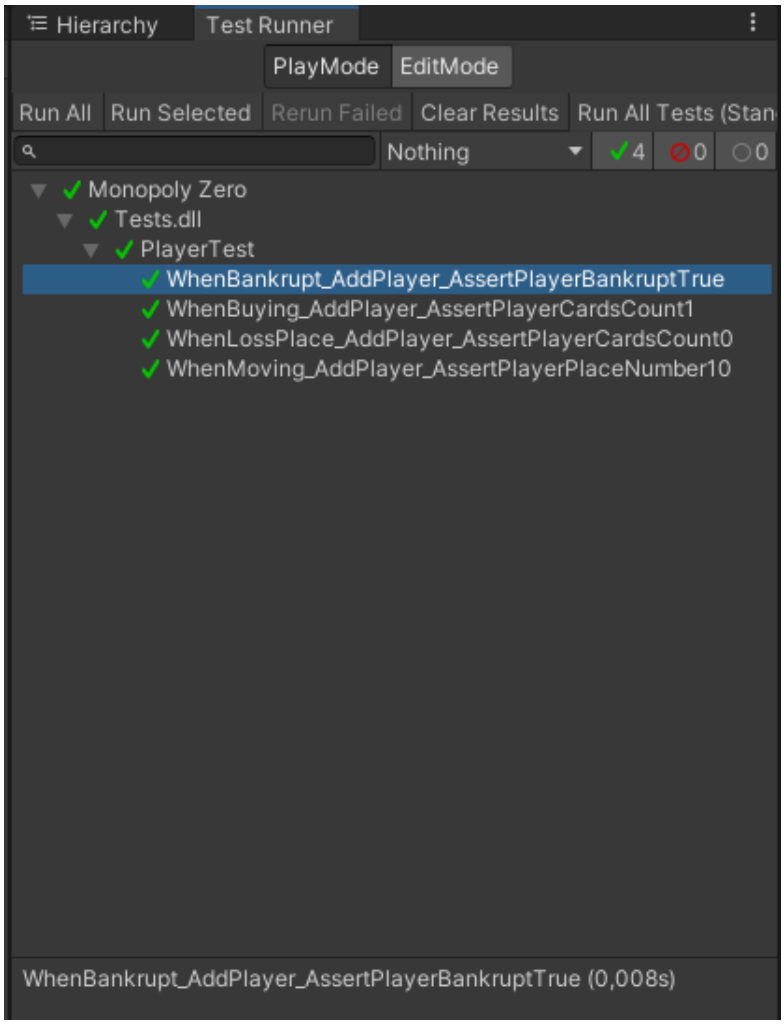


Рисунок 4.5 – Результат роботи юніт-тестів

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.

ВИСНОВКИ

У бакалаврській дипломній роботі був розроблений програмний додаток гра «Монополія» з елементами штучного інтелекту. Під час роботи було ознайомлена зі станом проблеми, порівняно розроблюваний програмний додаток з існуючими аналогами та описано алгоритми для вирішення поставлених проблем.

Було обрано графічний інтерфейс. Описані графічні інтерфейси програмного додатку. Розроблено блок-схеми алгоритмів програмного додатку. Розроблено і описано роботу штучного інтелекту.

Була описана розробка програмного додатку. Пояснений вибір ігрового рушія Unity, мови програмування C# та інтегрованої середи розробки Microsoft Visual Studio 2019. Була створена детальна інструкція користувача. Також була описана взаємодія між головними вікнами програми та описана частина коду в основній грі.

Для програмного додатку була обрана методика тестування юніт-тестами, яка була порівняна з такими методиками, як: тестування методом білого ящика, тестування методом чорного ящика та ручним тестуванням. Після порівняння юніт-тестування більш за інші методики підійшла для програмного додатку, оскільки більш спрямована на тестування окремих блоків і ігровій рушій Unity має вбудовану можливість створювати юніт-тести.

ПЕРЕЛІК ПОСИЛАНЬ

1. Д.В. Богомазов, Д.І. Катєльніков. Розробка ігрового застосунку з елементами штучного інтелекту з використанням технології Unity та мови C#. Матеріали LI Науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії. Вінниця:ВНТУ, 2022.

2. Штучний інтелкт [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Штучний_інтелект.

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

4. Monopoly online [Електронний ресурс] – Режим доступу до ресурсу: https://ru.monopoly-club.com.

5. Monopoly one [Електронний ресурс] – Режим доступу до ресурсу: https://monopoly-one.com.

6. Monopoly star [Електронний ресурс] – Режим доступу до ресурсу: https://monopolystar.ru.

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

8. Розвиток штучного інтелекту [Електронний ресурс] – Режим доступу до ресурсу: https://dut.edu.ua/ua/news-1-576-8835-poyava-ta-perspektivi-rozvitku-shtuchnogo-intelektu_kafedra-shtuchnogo-intelektu

9. Інтерфейс командного рядка [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Інтерфейс_командного_рядка.

10. Веб-інтерфейси [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Вебінтерфейс.

11. Графічний інтерфейс користувача [Електронний ресурс] – Режим доступу до ресурсу: uk.wikipedia.org/wiki/Графічний_інтерфейс_користувача.

12. UML [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Unified_Modeling_Language

13. Ігровий рушій [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Ігровий_рушій.

14. Арам Куксон, Райан Даулингсока, Клинтон Крамплер. Разработка игр на Unreal Engine 4 за 24 часа. : Пер. с англ. – М.: ООО ”Бомбора”, 2019.-528 с.: ил.

15. Хокинг Джозеф. Unity в действии. Мультиплатформенная разработка на C#. – Питер: ВДТУ, 2019 – 352 с.

16. Троелсен, Эндрю. Язык программирования C# 2005 и платформа .NET 2.0. 3-е издание.: Пер. с англ. – М.: ООО ”Издательский дом Вильямс”, 2007.-1168 с.: ил.

17. Бьярне Страуструп. Программирование. Принципы и практика с использованием C++.: Пер. с англ. – М.: ООО ”Издательский дом Вильямс”, 2018.-1329 с.: ил.

18. Блинов, И.Н., Романчик, В. С. Java. Методы программирования.: Пер. с англ. – М.: ООО ” Четыре четверти”, 2013.- 896 с.: ил.

19. Марк Лутц. Изучаем Python (5 издание, том 1).: Пер. с англ. – М.: ООО ” Диалектика”, 2020.-833 с.: ил.

20. Visual Studio [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio

21. Чорний ящик [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Чорний_ящик

22. Білий ящик [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Структурне_тестування

23. Ручне тестування [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Ручне_тестування

24. Юніт-тестування [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Модульне_тестування

25. Unity unit testing [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Manual/testing-editortestsrunner.html

ДОДАТКИ

ДОДАТОК А. Технічне завдання

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

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"__25__ " березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка ігрового програмного застосування "Гра Монополія" з елементами штучного інтелекту з використанням технології Unity та мови C#» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ д.т.н., доц. каф Д.І. Катєльніков
"____"_____2022 р.
Виконав:
_____студент гр.2ПІ-18б Д.В. Богомазов
"____"_____2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування.

Розробка ігрового програмного застосування "Гра Монополія" з елементами штучного інтелекту з використанням технології Unity та мови C#

Кінцевий продукт використовується лише у рамках бакалаврської дипломної роботи.

2. Підстава для розробки.

Підставою для розробки бакалаврської дипломної роботи є рішення засідання кафедри програмного забезпечення (протокол №13 від 11.02.2022).

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є створення програмного додатку в якому буде використаний штучний інтелект, для цього було обрано розробляти гру «Монополія».

4. **Вихідні дані для проведення БДР**

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. Кононюк А.Ю. Нейроні мережі і генетичні алгоритми: Науково-практичне видання. – Київ: Корнійчук, 2008. – 446 с.
2. Хокинг Джозеф. Unity в действии. Мультиплатформенная разработка на C#. – Питер: ВДТУ, 2019 – 352 с.
3. Арам Куксон, Райан Даулингсока, Клинтон Крамплер. Разработка игр на Unreal Engine 4 за 24 часа. : Пер. с англ. – М.: ООО ”Бомбора”, 2019.-528 с.: ил.
4. Троелсен, Эндрю. Язык программирования C# 2005 и платформа .NET 2.0. 3-е издание.: Пер. с англ. – М.: ООО ”Издательский дом Вильямс”, 2007.-1168 с.: ил.
5. Бьярне Страуструп. Программирование. Принципы и практика с использованием C++.: Пер. с англ. – М.: ООО ”Издательский дом Вильямс”, 2018.-1329 с.: ил.
6. Блинов, И.Н., Романчик, В. С. Java. Методы программирования.: Пер. с англ. – М.: ООО ” Четыре четверти”, 2013.- 896 с.: ил.
7. Марк Лутц. Изучаем Python (5 издание, том 1).: Пер. с англ. – М.: ООО ” Диалектика”, 2020.-833 с.: ил.

5. Технічні вимоги.

5.1. Вимоги до середовища розробки:

5.1.1. Операційна система – Windows 8/8.1/10.

5.1.2. Фреймворк – .NET 4.7.

5.1.3. Середовище розробки – Visual Studio 2019.

5.1.4. Мова програмування – C#.

5.2. Мінімальні вимоги до системи:

5.2.1. Операційна система – Windows 8/8.1/10.

5.2.2. Процесор – 1,0 ГГц.

5.2.3. Оперативна пам’ять – 128 МБайт і вище.

5.2.4. Об’єм жорсткого диску – 256 Мбайт вільного простору

5.2.5. Графічний пристрій – Об’єм пам’яті 256 Мбайт та сумісність з DirectX 9.0b

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати всім діючим стандартам України.

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

7.1.1. Пояснювальна записка до бакалаврської дипломної роботи.

7.1.2. Лістинг програми.

8. Стадії та етапи розробки

№з/п	Назва етапів дипломного проекту	Термін виконання етапів проекту (роботи)	Примітка
1	Аналіз сучасного стану питання та обґрунтування завдання на роботу	26.03.2022 - 10.04.2022	Виконано
2	Розробка структури серверного додатку	11.04.2022-25.04.2022	Виконано
3	Розробка алгоритмів програмного додатку	26.04.2022-30.04.2022	Виконано
4	Розробка інтерфейсу програмного додатку	01.05.2022-06.05.2022	Виконано
5	Програмна реалізація додатку	07.05.2022-25.05.2022	Виконано
6	Тестування роботи додатку та алгоритму	26.05.2022-01.06.2022	Виконано
7	Оформлення матеріалів до захисту БДР	01.06.2022- 10.06.2022	Виконано

9. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту. Корегування технічного завдання допускається з дозволу керівника бакалаврської дипломної роботи.

ДОДАТОК Б. Лістинг програмного коду

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 (haveDdistricts.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 В. Ілюстративний матеріал

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ

Завідувач кафедри ПЗ, д. т. н., професор _____ О. Н. Романюк

Науковий керівник, к. т. н., доц. кафедри ПЗ _____ Д.І. Катєльніков

Рецензент, к. т. н., доцент кафедри КН _____ О.М. Василевський

Нормоконтроль, к. т. н., доц. кафедри ПЗ _____ Д.І. Катєльніков

Виконавець, студент групи 2ПІ-18б _____ Д.В. Богомазов

РОЗРОБКА ІГРОВОГО ПРОГРАМНОГО ЗАСТОСУВАННЯ "ГРА МОНОПОЛІЯ" З ЕЛЕМЕНТАМИ ШТУЧНОГО ІНТЕЛЕКТУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ UNITY ТА МОВИ C#.

Виконав:
Богомазов Д.В.

Науковий керівник
к.т.н., доцент каф. ПЗ Катєльніков Д.І.

Мета, об’єкт та предмет дослідження

- Мета дослідження. Метою бакалаврської дипломної роботи є реалізація штучного інтелекту при розробці ігрового програмного додатку «Монополія».
- Об’єкт дослідження: процес розробки ігрового додатку «Монополія» з елементами штучного інтелекту.
- Предмет дослідження: методи та засоби розробки ігрових додатків.

Постановка задач розробки

Оскільки завданням дипломної роботи є розробка програмного додатку гра «Монополія» з елементами штучного інтелекту задачами розробки є:

- розробити БД;
- розробити алгоритм створення ігрових об'єктів;
- розробити алгоритм поведінки живих гравців;
- розробити алгоритм поведінки гравців зі штучним інтелектом;
- розробити взаємодію між гравцями;
- створити логістику постачання ігрової валюти гравцям;
- розробити алгоритм завершення гри.

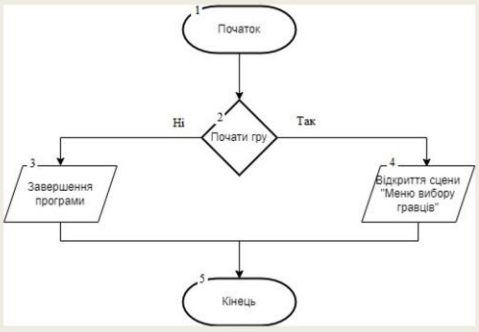
Структура ігрового додатку «Монополія»

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

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

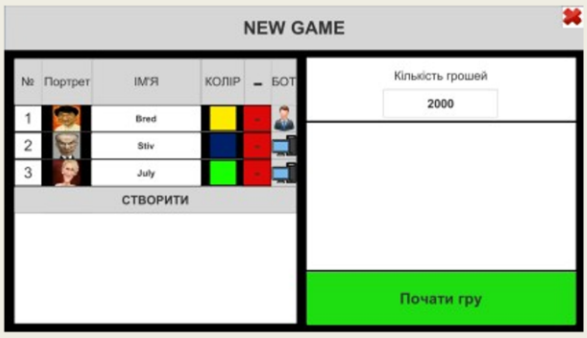
Головне меню

Інтерфейс головного меню



Алгоритм роботи головного меню

Меню вибору гравців



Інтерфейс меню вибору персонажів

Алгоритм роботи меню вибору гравців

Гра

Інтерфейс гри

Реалізація 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 обираючи 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 секторомв нього є можливість використати функції центрального меню, якщо потрібно;
- в кінці бот натискає клавішу завершити хід.

Графічне представлення штучного інтелекту

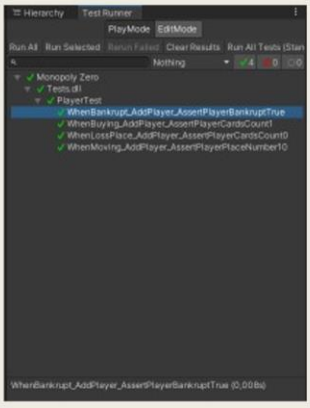
Для того, щоб у грі з'явився штучний інтелект, або бот, потрібно в меню вибору гравців натиснути на спеціальну кнопку на панелі гравця.

Після вибору гравців і переходу безпосередньо до гри дізнатись чи є гравець штучним інтелектом можна за допомогою індекатора на його профілі.

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.



Результат тестування

Дякую за увагу!

ДОДАТОК Г. ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник:

Оригінальність	89.9%
Схожість	10.1%

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.