Міністерство освіти і науки України

Одеський національний політехнічний університет

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Місько Дмитро Андрійович

студент групи АС-151

**КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА**

Ігровий штучний інтелект суперників, що навчається на базі дій гравця,

в іграх жанру "Гонки"

Спеціальність:

121 – Інженерія програмного забезпечення

Спеціалізація:

Інженерія програмного забезпечення

Керівник:

Рувінська Вікторія Михайлівна,

к.т.н., професор

Одеса – 2020

|  |  |
| --- | --- |
|  | Міністерство освіти і науки України |
| Одеський національний політехнічний університет | |
| Навчально-науковий інститут комп’ютерних систем | |
| Кафедра системного програмного забезпечення | |
| Рівень вищої освіти: | другий (магістерський) |
| Спеціальність: | 121 – Інженерія програмного забезпечення |
| Спеціалізація: | Інженерія програмного забезпечення |

|  |
| --- |
| ЗАТВЕРДЖУЮ |
| Завідувач кафедри |
| Крісілов В. А. |
| « » 20 р. |

|  |
| --- |
| **ЗАВДАННЯ** |
| **НА КВАЛІФІКАЦІЙНУ РОБОТУ** |

Місько Дмитра Андрійовича, група АС-151

|  |
| --- |
| 1. Тема роботи: Ігровий штучний інтелект суперників, що навчається на базі дій гравця, в іграх жанру "Гонки" |

|  |  |
| --- | --- |
| Керівник роботи: | Рувінська Вікторія Михайлівна, к.т.н., професор |

затверджені наказом ректора від « 29 » жовтня 2020 р. № 412-в

1. Зміст роботи: Аналіз вимог до програмного продукту, план виконання проекту, побудова методу адаптації рівня гри суперників під рівень гри гравця, проектування програмної системи, програмна реалізація системи, тестування програмної системи, аналіз результатів, розгортання програмного продукту, написання пояснювальної записки
2. Перелік ілюстративного матеріалу: згідно зі слайдами презентації
3. Консультанти розділів роботи

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
| завдання видав | завдання  прийняв |
| ОПтаБНС | Москалюк А.Ю., доц. |  | 10.11.2020 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Дата видачі завдання: « 1 » вересня 20 20 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів кваліфікаційної роботи | Строк  виконання | Примітка |
| 1 | Аналіз вимог до програмного продукту | 18.09.2020 – 26.09. 2020 | Виконано |
| 2 | План виконання проекту | 19.09.2020 – 26.09.2020 | Виконано |
| 3 | Аналіз існуючих підходів та аналогів | 23.09.2020 – 26.09.2020 | Виконано |
| 4 | Побудова методу адаптації рівня гри суперників під рівень гри гравця | 10.10.2020 – 15.10.2020 | Виконано |
| 5 | Проектування програмної системи | 15.09.2020– 20.10.2020 | Виконано |
| 6 | Програмна реалізація системи | 20.10.2020 – 23.10.2020 | Виконано |
| 7 | Тестування програмної системи та аналіз результатів | 23.10.2020 – 30.10.2020 | Виконано |
| 8 | Розгортання програмного продукту | 01.11. 2020 – 25.11.2020 | Виконано |
| 9 | Написання пояснювальної записки | 25.11.2020 – 04.12.2020 | Виконано |
| 10 | Розгляд питань охорони праці безпеки у надзвичайних ситуаціях | 10.11.2020 – 01.12.2020 | Виконано |

|  |  |
| --- | --- |
| **Здобувач вищої освіти** | Д.А. Місько |
| **Керівник роботи** | В. М. Рувінська |

**ЗАВДАННЯ**

на розробку розділу «Охорона праці та безпека в надзвичайних ситуаціях»

|  |
| --- |
| Місько Дмитра Андрійовича, група АС-151 |
| Навчально-науковий інститут комп’ютерних систем |
| Кафедра системного програмного забезпечення |

|  |  |
| --- | --- |
| Тема роботи: | Ігровий штучний інтелект суперників, що навчається на базі дій гравця, в іграх жанру "Гонки" |

Зміст розділу:

* 1. Аналіз умов праці і вибір заходів і засобів захисту від небезпечних і шкідливих виробничих факторів.
  2. Аналіз техногенних небезпек і вибір заходів і засобів забезпечення без- пеки у надзвичайних ситуаціях.
  3. Складання рекомендацій щодо поліпшення умов праці.

|  |  |
| --- | --- |
| Керівник роботи | Консультант з охорони праці та БНС |
| В. М. Рувінська | А. Ю. Москалюк |
| « » 20 р. | « » 20 р. |

РЕФЕРАТ

Пояснювальна записка до дипломної роботи: 96 с., 22 рис., 14 табл., 1 додаток, 22 джерел.

Робота присвячена розробці гри жанру «Гонки», у якій рівень гри суперників адаптується під рівень гри гравця.

Метою дослідження є підвищення зацікавленості гравця в іграх жанру "Гонки" на основі машинного навчання моделей суперників на діях гравця.

Як результат розроблено метод адаптації ігрового штучного інтелекту в іграх жанру «Гонки» під рівень гри гравця, а також на цій основі створена програмна система, що являє собою десктопну гру.

У роботі використовується мова програмування С++, бібліотека для нейронних мереж MiniDNN, а також графічна бібліотека SFML.

Ключові слова: гонки, ігровий штучний інтелект, нейронні мережі.

ABSTRACT

The work is devoted to the development of a game of the genre "Race", in which the level of the opponent's game is adapted to the level of the player's game.

The purpose of the study is to increase the player's interest in games of the genre "Race" based on machine learning of rival models on the actions of the player.

As a result, a method of adapting the game's artificial intelligence in games of the "Race" genre to the level of the player's game was developed, and on this basis a software system was created, which is a desktop game.

The paper uses the C++ programming language, the library for MiniDNN neural networks, as well as the graphic library SFML.

Key words: races, game artificial intelligence, neural networks.

**ЗМІСТ**

[ВСТУП 8](#_Toc58532593)

[1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ 11](#_Toc58532594)

[1.1 Основні поняття 11](#_Toc58532595)

[1.2 Історія ігрового інтелекту 12](#_Toc58532596)

[1.3 Історія розвитку ігор типу «Гонки» 13](#_Toc58532597)

[1.4 Критичний огляд ігор типу «Гонки» з ігровим інтелектом 15](#_Toc58532598)

[2 ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ 17](#_Toc58532599)

[2.1 Мета проведення роботи 17](#_Toc58532600)

[2.2 Призначення розробки 17](#_Toc58532601)

[2.3 Вимоги до програми 18](#_Toc58532602)

[2.4 Вимоги до програмної документації 18](#_Toc58532603)

[2.5 Технічно-економічні показники 19](#_Toc58532604)

[2.6 Технічні вимоги 19](#_Toc58532605)

[2.7 Стадії та етапи розробки програмної системи 19](#_Toc58532606)

[3 РОЗРОБКА МЕТОДУ АДАПТАЦІЇ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ В ІГРАХ ЖАНРУ «ГОНКИ» ПІД РІВЕНЬ ГРИ ГРАВЦЯ 21](#_Toc58532607)

[3.1 Нейронні мережі суперників для гри жанру «Гонки» 21](#_Toc58532608)

[3.1.1 Загальні положення про штучні нейронні мережі 21](#_Toc58532609)

[3.1.2 Конфігурація вхідного шару 25](#_Toc58532610)

[3.1.3 Конфігурація вихідного шару 28](#_Toc58532611)

[3.1.4 Конфігурація проміжного шару 28](#_Toc58532612)

[3.2 Міра адаптованості нейронної мережі суперників до гри гравця 29](#_Toc58532613)

[3.3 Кроки розробленого методу 30](#_Toc58532614)

[3.3.1 Збір бази дій гравця 30](#_Toc58532615)

[3.3.2 Нормалізація зібраних даних 31](#_Toc58532616)

[3.3.3 Фільтрація зібраних даних 32](#_Toc58532617)

[3.3.4 Тренування нейронних мереж суперників 36](#_Toc58532618)

[3.3.5 Застосування нейронних мереж суперників 37](#_Toc58532619)

[3.4 Висновки до розділу 3 37](#_Toc58532620)

[4 ПРОЕКТУВАННЯ СИСТЕМИ 38](#_Toc58532621)

[4.1 Опис нефункціональних вимог 38](#_Toc58532622)

[4.2 Опис функціональних вимог 39](#_Toc58532623)

[4.3 Архітектура системи 42](#_Toc58532624)

[4.4 Проектування інтерфейсу користувача 45](#_Toc58532625)

[4.5 Детальне проектування логічного представлення системи 48](#_Toc58532626)

[5 РЕАЛІЗАЦІЯ ТА ВИПРОБУВАННЯ СИСТЕМИ 56](#_Toc58532627)

[5.1 Опис програмних технологій 56](#_Toc58532628)

[5.2 Інструкція з встановлення 56](#_Toc58532629)

[5.3 Інструкція з використання 57](#_Toc58532630)

[5.4 Функціональне тестування 61](#_Toc58532631)

[5.5 Випробування системи і аналіз результатів 63](#_Toc58532632)

[6 ОХОРОНА ПРАЦІ 67](#_Toc58532633)

[ВИСНОВКИ 68](#_Toc58532634)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70](#_Toc58532635)

# ВСТУП

У сучасному світі комп'ютерні ігри стали не тільки розвагою, але і частиною культури. Вони мало схожі на інші прояви сучасних видів мистецтв, але точно так само фіксують сучасну мораль, етику і уявлення про минуле і майбутнє більшості людей.

Комп'ютерні ігри дають людям нову унікальну можливість перенестися в світ фантазій. Раніше не було способу глибоко занурюватися в нереальний світ і мати там свободу дій. Перевага над кіно або книгами комп'ютерних ігор дає їх інтерактивність: ігри втягують у спільну діяльність, гравець перестає бути пасивним спостерігачем, він активно впливає на поточні події.

Існує багато різних жанрів комп’ютерних ігор серед яких є «Гонки». У реальному житті прийняти участь у гонках на машинах складно та небезпечно. Комп’ютерні ігри гоночного жанру дозволяють користувачу приміряти на себе роль гонщика без загрози для свого життя та без складної підготовки.

Для імітації розумної поведінки персонажів розробники ігор наділяють їх ігровим штучним інтелектом. Ігровий штучний інтелект – набір програмних методик, які використовуються у відеоіграх для створення ілюзії інтелекту в поведінці персонажів, керованих комп'ютером. Ігровий штучний інтелект, крім методів традиційного штучного інтелекту, включає також алгоритми теорії керування, робототехніки, комп'ютерної графіки та інформатики у цілому [16].

Існуючі рішення по створенню ігрового штучного інтелекту для суперників(ботів) в іграх жанру «Гонки» використовують два основні типи моделей: заздалегідь написаний на етапі розробки скінченний автомат [16], а також заздалегідь навчена нейронна мережа [11]. У першому випадку рівень гри бота може регулюватися штучним покращенням характеристик транспорту, котрим штучний інтелект керує, а також додаванням більш складної поведінки у різних ігрових ситуаціях. У другому випадку рівень гри бота регулюється надаванням ботам більш або менш навченої нейронної мережі.

В обох випадках штучний інтелект та його рівні складності створені на етапі розробки програмної системи, що не дозволяє точно змінити рівень складності гри під рівень гри конкретного гравця. Розробники ігор намагаються обійти це обмеження надаючи змогу гравцю вибрати рівень складності гри, зазвичай це: новачок, середній, складний. Проте, це далеко не завжди допомагає досить точно адаптувати гру під гравця. Справжній рівень гравця може бути, наприклад, серед «новачком» та «середнім» або серед «середнім» та «складним».

Ця проблема призводить до того, що гравець або часто програє штучному інтелекту, або навпаки – занадто часто виграє у нього. В обох випадках гравець втрачає інтерес до гри, оскільки він або доволі довго не може досягнути перемоги у гонці серед ботів-суперників, або занадто багато всю ігрову сесію виграє у них.

У роботі проведено розробку гри “AAIRace” жанру «Гонки» з ботами, штучний інтелект яких представляє собою нейронну мережу, яку навчають на базі дій гравця під час процесу гри. Це дозволяє створювати штучний інтелект, наближений до рівня гри гравця, безпосередньо коли гравець грає, що має вирішити проблему неадаптованого рівня складності і таким чином зробити гру більш збалансованою.

Метою дослідження є підвищення зацікавленості гравця в іграх жанру "Гонки" на основі машинного навчання моделей суперників на діях гравця.

Підвищення зацікавленості гравця – це зробити так, щоб він не втрачав інтерес до гри, а для цього суперники повинні мати наближені до гравця рівні гри. Представлена у роботі міра адаптованності нейроної мережі суперників до гри гравця являє собою суму квадратів різниці фактичної вірогідності зайняти і-те місце та теоретичної у випадку, коли рівень гри усіх персонажів однаковий.

Для досягнення цієї мети було виконано наступні завдання:

* проаналізовано історію ігрового інтелекту та ігор типу "Гонки";
* проведено аналіз різних підходів у створенні ігрового штучного інтелекту в іграх жанру «Гонки» та різноманітних варіантів програм-аналогів, оцінивши їх переваги та недоліки;
* розроблено метод адаптації ігрового штучного інтелекту в іграх жанру «Гонки» під рівень гравця;
* сформульовано та узгоджено технічне завдання на розробку гри жанру «Гонки»;
* на основі даних досліджень створено гру жанру «Гонки»;
* випробувано та проведено аналіз результатів роботи системи та розробленого методу.

Об’єктом дослідження є ігровий штучний інтелект в іграх жанру "Гонки".

Предметом дослідження є адаптація суперників до рівня гри гравця в іграх жанру "Гонки".

У роботі отримано нові наукові результати: отримав подальший розвиток метод адаптації ігрового штучного інтелекту в іграх жанру «Гонки» під рівень гри гравця, який на відміну від існуючих, навчається під час гри, що дозволяє створювати ботів, рівень гри яких максимально наближений до рівня гри гравця.

У першому розділі даної роботи проводиться аналіз існуючих гоночних ігор та їх способи адаптації рівня складності під рівень гри гравця.

Другий розділ містить технічне завдання на розробку.

У третьому розділі розглядається розробка методу адаптації ігрового штучного інтелекту в іграх жанру «Гонки» під рівень гри гравця.

Четвертий розділ містить у собі проектування програмної системи гри жанру «Гонки», штучний інтелект ботів якої представляє собою нейронну мережу, яка навчається на базі дій гравця під час гри.

У п’ятому розділі відображено тестування та випробування реалізованої системи, що створена за розробленим методом, проводиться оцінка отриманих результатів.

Шостий розділ містить виконане завдання з охорони праці та безпеки в надзвичайних ситуаціях.

# 1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Важливим етапом розробки інформаційних систем є аналіз вже існуючих рішень, який дозволяє зрозуміти наявний науковий прогрес та ситуацію на ринку у заданій предметній області. В даній роботі розглянуто існуючі рішення для системи в цілому, а також реалізацію штучного інтелекту суперників в них.

## 1.1 Основні поняття

Ігровий штучний інтелект (ШІ, англ. Game artificial intelligence) – набір програмних методик, які використовуються в комп'ютерних іграх для створення ілюзії інтелекту в поведінці персонажів, що управляються комп'ютером [1].

Персонажів комп'ютерних ігор, що управляються ігровим штучним інтелектом, ділять на:

* неігрові персонажі ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)Non-player character, NPC) – як правило, ці персонажі є дружніми або нейтральними до людського гравця; це такі персонажі в комп'ютерній грі, які не протидіють гравцю [2];
* боти (англ.Bot) – ворожі до гравця персонажі, що наближаються за можливостями до ігрового персонажу; проти гравця в будь-який конкретний момент борються невелика кількість ботів [3];
* моби (англ.Mob) – ворожі до гравця «низькоінтелектуальні» персонажі. Моби вбиваються гравцями в великих кількостях заради очок досвіду, артефактів або проходження території [4].

Ігровий ШІ використовує деякі з напрямків академічної науки «Штучний інтелект», часто складається з декількох емпіричних правил і евристик, яких достатньо, щоб надати гравцеві хороший геймплей, відчуття і враження від гри. Найочевидніше застосування ігрового ШІ проявляється в контролюванні ботів або мобів. Пошук шляху є іншим широко поширеним застосуванням ігрового ШІ, – він особливо проявляється в стратегіях реального часу. Пошук шляху є методом для визначення того, як персонажу, що управляється комп’ютером, перейти з однієї точки на карті до іншої: потрібно враховувати ландшафт, перешкоди і, можливо, «туман війни» [1]. Туман війни – військовий термін, що позначає відсутність достовірної інформації про поточну обстановку на полі бою в силу тих чи інших об'єктивних причин. В контексті ігор туман війни являє собою ефект, коли ховаються від опонентів типи і властивості фігур. Наприклад, якщо предмети кладуться написом вниз або повертаються так, щоб їх опису не бачив противник.

## 1.2 Історія ігрового інтелекту

Концепт ШІ в іграх зародився в 1956 році. Система ШІ в той час була набагато простіше, ніж в сучасних відеоіграх. Перший бот у відеогрі був введений в 1951 р, за 5 років до самого визначення концепції ШІ. Він створювався для гри в шахи та мав порівняно простий список алгоритмів, так що не міг скласти велику конкуренцію людям.

У 1980 році вийшла всесвітньо відома гра Pac-Man, суть якої – з'їсти всі точки на карті, обходячи привиди, що активно переміщаються. ШІ всіх привидів було поставлено таким чином, щоб всі вони рухалися за певним шаблоном. Це дозволяло гравцеві при досить великій кількості проведеного часу в грі передбачати поведінку ворогів.

1990 роки – це час масового виходу донині відомих ігор жанру «файтинг» (наприклад Mortal Kombat, Tekken, Soul Calibur і т.д.). Головна інновація для ігрового ШІ того часу – модернізація алгоритмів в реальному часі. Ворог міг адаптуватися до шаблонних дій гравців, наприклад: гравець знайшов дуже потужну комбінацію за персонажа з Tekken і повторює її вже 3 рази за раунд, в цей час ШІ «усвідомлює», що потрібно підбирати інший метод підходу до гравця і почне або атакувати в стрибку, або блокувати удари з подальшою контратакою. Іншими словами, персонажі, що управляються комп’ютером, починають шукати альтернативні шляхи виходу з ситуації, аналізуючи дії гравця, що часто ускладнює гру. Хоча файтинг того часу, як і зараз, був розрахований більше на протистояння двох реальних гравців.

У 1998 році вийшла гра Golden Eye 007, підносить геймеру досить просунутий штучний інтелект. Персонажі що керуються комп’ютером активно реагували на рухи і стрілянину з боку гравця, ховаючись в укриття. Також вороги відмінно підбирали момент для метання гранат під ноги, але величезний мінус ШІ в грі був в тому, що вороги точно знали місце розташування гравця, де б він не був.

FEAR, що вийшов в 2005-му, дуже порадував гравців і ігрових аналітиків своїм штучним інтелектом. Відрізняло FEAR від інших ігор схожого типу – чудово продумана поведінка ворогів. Головна перевага над комп’ютерними персонажами інших проектів полягало в тому, що вони діяли спільно. Не просто бездумно йшли в одну точку натовпом, а тактично розподілялися між укриттями, використовуючи певні тактики для провокації персонажа гравця, ставлячи його в незручне становище [5].

Далі в 2008 році вийшов новий мережевий кооперативний шутер від першої особи Left 4 Dead, використовував нову систему ігрового ШІ під назвою «Режисер» (англ. The Director). Система використовувалася для процедурного генерування різного ігрового досвіду для гравців при кожному запуску гри. Розробники гри назвали спосіб, згідно з яким працює Режисер, «процедурним наративом». Замість строго встановлених і статичних рівнів складності «Режисер» аналізує дії і «ступінь виживання» гравців і відповідно до цього динамічно додає наступні події, роблячи гру цікавою, але також і прохідною. Проте, поряд з «Режисер» в грі присутні і рівні складності, які впливають на стійкість і ступінь пошкоджень ігрових персонажів [5].

Таким чином, однією з задач ігрового ШІ є балансування гри. Ігровий баланс в комп'ютерних іграх – це баланс між числами, що описують різні характеристики в грі, визначає складність, інтерес і плавність ігрового процесу [22].

## 1.3 Історія розвитку ігор типу «Гонки»

Першою гоночною грою прийнято вважати Space Race, яку в 1973 році презентувала світу легендарна компанія Atari.

Гравець в ній керував космічним кораблем, якому необхідно було дістатися до лінії фінішу максимально швидко. При цьому важливо не зіткнутися з кораблем суперника або в літальні апарати, керовані комп'ютером. На базі цієї ідеї з'явилися інші ігри, де навпаки потрібно зіткнутися із суперником і набирати так бонуси, або просто влаштувати перегони [6].

Ігри жанру «Гонки» швидко перекочували з космосу на трасу. Першою стала Speed ​​Race від компанії Taito. Її поява зайняла всього рік. Далі пішли варіації автосимуляторів на трасі: з видом зверху, з видом з кабіни водія, з повною трасою або відображається частково. Деякі розробники додали навіть карти-лабіринти.

Велика частина ігор аж до половини 90-х років випускалася для ігрових автоматів. Як елементи управління пропонувалися джойстики, керма і навіть невеликі автомобілі або мотоцикли. Але з розвитком комп'ютерної техніки гонки почали перебиратися з ігрових центрів на комп'ютери користувачів, а потім і в Інтернет.

Головними віхами 1990-х стала поява популярної серії The Need for Speed, перша гра якої вийшла в 1993 році. На сьогоднішній день компанія випустила вже 10 нових версій. У 1999 році з'явилися нові ідеї: все більше розробників відмовляються від трас на користь відкритих карт, де гравцю необхідно вибрати максимально короткий шлях [6].

У 1980-х роках розробники гоночних ігор розробили перший реалістичний автосимулятор. Звичайно, до сучасних симуляторів машин роботі Джеффа Креммонда REVS було далеко і цьому заважали апаратні обмеження. Автор постарався максимально наблизити управління в грі до управління реальним транспортним засобом.

З'явилася тривимірна графіка. Її додали в 1989 році в автомат Hard Drivin.

У 2000 році і далі гоночні ігри в основному не змінювалися в плані геймплея. В основному були зміни в плані швидкодії, поліпшення графіки, доповнення функціоналу, необхідного для управління [6].

## 1.4 Критичний огляд ігор типу «Гонки» з ігровим інтелектом

В даний час існують багато ігор в жанрі типу «Гонки». Для порівняння розглянемо деякі з них.

**Crossout** – гра в жанрі «Гонки на виживання». Головна особливість гри – бронемобілі, які гравець створює самостійно з різних частин і матеріалів, отриманих в бою або куплених у інших гравців на ринку. Основні частини машини – кабіна, платформа і різноманітні ходові – обов'язкові для створення бронемобіля. Основна мета – захоплення чужої бази або знищення всіх супротивників [7].

У ботів в Crossout простий ШІ: бот шукає і знищує ворожих гравців. Якщо бот не може знайти ворожі машини, він робить спробу захопити ворожу базу або захищає свою [8].

**FlatOut** – гоночна відеогра, розроблена Bugbear Entertainment. Відмінною особливістю FlatOut є майже повна руйнація об'єктів, розставлених на трасі, – будь то дерев'яний парканчик, який огороджує складний поворот, або водонапірне спорудження [9].

ШІ суперників в цій грі заздалегідь натренований на певний рівень гри, також кожен суперник має свій автомобіль з певними характеристиками, і чим далі гравець проходить гру, тим більше натренований ШІ, і більш швидкі автомобілі суперників його чекають. Відповідно складність гри зростає від дуже легкого до складного по мірі проходження гри [10].

**Colin McRae Rally 2.0** – відеогра в жанрі автосимулятор, розроблена і видана Codemasters для PlayStation, Windows і Game Boy Advance. Це друга гра серії Colin McRae Rally, в ній представлені справжні машини і ралі сезону 2000, Світового чемпіонату з ралі. У грі є три рівні складності: Новачок, Любитель і Експерт [11].

Суперники мають заздалегідь натренований ШІ, представлений штучною нейронною мережею, проте машини схожі своїми характеристиками [12].

Всі представлені вище аналоги не мають функції адаптації рівня гри під конкретного гравця або реалізують її частково. У Crossout рівень гри бота регулюється лише швидкістю машини, точністю знаряддя, швидкістю перезарядки і т.д. Такий ШІ реалізований з використанням кінцевих автоматів і не може демонструвати високу гнучкість в адаптації під рівень гри гравця шляхом навчання на його діях. Рівень гри такого ШІ регулюється, наприклад, регулюванням часу прицілювання (пауза між знаходженням цілі і пострілом) у ворога, і/або точності стрільби, швидкості машини і т.д. Це погано імітує протистояння сильнішому (або рівному) реальному гравцеві, тому що дає більше можливостей ботам в порівнянні зі звичайним гравцем, в той час як реальний суперник використовує ті ж умови середовища. У FlatOut і Colin McRae Rally 2 ШІ представлений заздалегідь натренованими моделями, це дозволяє імітувати досить складну реальну поведінку гравця. Однак в цих іграх штучні нейронні мережі заздалегідь натреновані на підставі даних, які були взяті із загальної бази, ніяк не пов'язаної з конкретним гравцем. Таким чином, адаптація рівня складності до конкретного гравця в цих іграх лише умовна, тому що гра ніяк не враховує і не аналізує для своїх подальших дій поведінку гравця і його реакцію на конкретні події. Крім цього, в FlatOut, автомобілі суперників мають інші характеристики, ніж у гравця, що в будь-якому випадку робить боротьбу спочатку нерівною.

Таким чином, в розглянутих іграх хоч і використовуються нейронні мережі для управління ботів, наприклад як це реалізовано у грі Colin McRae Rally 2.0, але вони навчаються заздалегіть і тому не можуть адаптуватися під дії конкретного гравця. У роботі пропонується навчати нейронні мережі суперників динамічно по ходу гри на базі дій гравця, навчаючись на підставі того, що бачить гравець і його подальшої реакції на конкретні події. Таким чином, ШІ суперника намагається «наслідувати» поведінку гравця.

# 2 ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ

## 2.1 Мета проведення роботи

Підставою розробки є завдання на дипломне проектування. Темою роботи є «Ігровий штучний інтелект суперників, що навчається на базі дій гравця, в іграх жанру "Гонки"». Метою дослідження є підвищення зацікавленості гравця в іграх жанру "Гонки" на основі машинного навчання моделей суперників на діях гравця. Пропонується розробка гри жанру «Гонки», суперники котрої навчаються під час гри. Таким чином, суперники наслідують реакції на ігрові події гравця та на основі цих реакцій екстраполюють дії, які б виконав гравець у відповідь на ігровий стан з точки зору суперників. Тим самим суперники грають близько рівню гравця, що автоматично регулює гру під рівень гри гравця.

## 2.2 Призначення розробки

Задачею розробки є створення десктопної програмної системи, яка представляє собою гру жанру «Гонки», суперники котрої мають штучний інтелект, створений нейронною мережею, яку навчають на базі дій гравця під час гри.

Десктопна програма дає користувачу змогу створити аккаунт або увійти під існуючим. Наступним кроком система дозволяє обрати один із 30ти розблокованих рівнів. Коли користувач проходить останній розблокований рівень, система розблоковує наступний та навчає штучний інтелект суперників на цьому рівні на базі дій гравця за останні декілька рівней.

Створення десктопної програмної системи, яка представляє собою гру, не ставиться за головну мету, а лише розглядається як розробка ігрових умов, у яких штучний інтелект адаптується, навчаючись грі у гравця. Для автоматизації процесу навчання нейронних мереж треба збирати базу дій гравця під час гри та після закінчення поточного рівня починати тренування суперників. Для реалізації нейронної мережі треба визначити формат, у якому надається ігровий стан на вхід, а також формат, у якому нейронна мережа видає дії. Необхідно проаналізувати існуючі методи тренування нейронних мереж, їх початкову конфігурацію, а також рекомендації її структурування. На основі отриманих результатів потрібно спроектувати та розробити дескотпну програму для Windows, яка буде виконувати описані нижче функції.

## 2.3 Вимоги до програми

Розроблена програмна система повинна мати наступні функціональні можливості:

* створювати аккаунт нового користувача та зчитувати інформацію з аккаунту існуючого;
* давати можливість користувачу обирати один із 30ти рівнів;
* надавати можливість користувачу розібратися із управлінням автомобілем;
* надавати можливість користувачу керувати автомобілем у грі.

Нефункціональні вимоги – це вимоги до [програмного забезпечення](https://uk.wikipedia.org/wiki/Програмне_забезпечення), які задають критерії для оцінки якості його роботи [14].

До програмної системи висуваються наступні нефункціональні вимоги:

* у процесі роботи з програмою користувач не повинен спостерігати помітних затримок у відклику графічного інтерфейсу; якщо виконання деякої операції потребує багато часу, то програма повинна зробити так, щоб користувачу було зрозуміло, що програма зайнята.

## 2.4 Вимоги до програмної документації

В документації до програмної системи повинна міститься наступна інформація:

* специфікація функціональних вимог до системи;
* опис формату, у якому дається на вхід до нейронної мережі інформація про ігровий стан з точки зору автомобіля;
* опис формату, у якому нейронна мережа видає на виході інформацію про дії, котрі треба зробити супернику, зважаючи на ігровий стан довкола нього;
* опис архітектури десктопної програми;
* опис графічного інтерфейсу десктопної програми.

## 2.5 Технічно-економічні показники

Розроблена система може бути застосована звичайними користувачами задля розваги у вільний час. Окремі модулі системи можуть бути повторно використані у схожих застосуваннях, а метод може бути застосований до різних предметних областей.

## 2.6 Технічні вимоги

До застосування висунуті наступні технічні вимоги:

* десктопне застосування повинно працювати на ПК з операційною системою не меншою, ніж Windows 7;
* навчання нейронних мереж суперників повинно бути у різних потоках, задля прискорення процесу.

## 2.7 Стадії та етапи розробки програмної системи

Стадії та етапи розробки програмної системи наведено у таблиці 2.1.

Таблиця 2.1 – Стадії та етапи розробки

|  |  |  |  |
| --- | --- | --- | --- |
| Стадія | Етап | Зміст робіт | Строки |
| Огляд існуючих рішень | Вивчення існуючих рішень ігор жанру «Гонки» | Вивчення функціональних можливостей та особливостей існуючих рішень. | 18.09.2020 – 26.09. 2020 |

Продовження таблиці 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| Стадія | Етап | Зміст робіт | Строки |
| Технічне завдання | Визначення даних, пошук яких підтримуватиме система, та можливих варіантів їх формулювання. | Опис форматів входу та виходу для конфігурації нейронної мережі. | 19.09.2020 – 26.09.2020 |
| Створення специфікації вимог до програмної системи. | Складання документації щодо вимог до програмної системи. | 15.09.2020– 20.10.2020 |
| Технічний проект | Побудова методики для адаптації рівня гри суперників під рівень гри гравця | Розробка оцінки конфігурації нейронної мережі | 10.10.2020 – 15.10.2020 |
| Проектування десктопної програми | Детальне проектування та створення діаграм класів | 20.10.2020 – 23.10.2020 |
| Робочий проект | Реалізація програмної системи | Кодування і налагодження програми, завершення розробки програмної документації | 01.11.2020 – 25.11.2020 |
| Готовий проект | Випробування системи | Апробація методики, випробування системи, аналіз результатів | 25.11.2020 – 01.12.2020 |

# 3 РОЗРОБКА МЕТОДУ АДАПТАЦІЇ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ В ІГРАХ ЖАНРУ «ГОНКИ» ПІД РІВЕНЬ ГРИ ГРАВЦЯ

В цьому розділі представлено розробку методу адаптації ігрового штучного інтелекту в іграх жанру «Гонки» під рівень гри гравця, яка буде використовуватись у десктопній програмі.

Одним з найважливіших етапів розроблюваного методу є збір бази дій гравця. Для розробки методу потрібно розглянути такі проблеми, як попередня підготовка формату надавання інформації щодо ігрового стану відносно автомобіля суперника до нейронної мережі, опрацювання вхідних даних нейронної мережі, а також визначити, якою саме буде архітектура нейронної мережі і як саме будуть відбиратися дані для її тренування, які кроки розробленого методу.

## 3.1 Нейронні мережі суперників для гри жанру «Гонки»

### 3.1.1 Загальні положення про штучні нейронні мережі

Штучні нейронні мережі – це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин. Такі системи навчаються вирішуванню задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування [17].

Штучна нейронна мережа складається із штучних нейронів.

Вхідними сигналами штучного нейрона є вихідні сигнали інших нейронів, кожний з яких узятий зі своєю вагою.

Вхідний оператор перетворює зважені входи й подає їх на оператор активації. Вихідний сигнал нейрона уявляє собою перетворений вихідним оператором вихідний сигнал оператора активації. Вхідний оператор (вхідна функція) нейрона задає вигляд використовуваного в нейроні перетворення зважених входів. Зазвичай вхідний оператор – це сума добутків сигналу входу на відповідну до нього вагу [18].

Функція активації трансформує вихідні дані вхідного оператора у сигнал, який видається на виході нейрону.

Існують такі основні функції активації:

* лінійна: ;
* сигмоїда: ;
* гіперболічний тангенс (TanH): ;
* випрямлена лінійна (ReLU): );
* SoftPlus: .

Алгоритм навчання – це метод або математична модель, яка підвищує продуктивність штучної нейронної мережі. Це робиться шляхом поновлення ваг на входах штучних нейронів [19].

На сьогодні відомо три парадигми навчання нейронних мереж, в основу яких покладено особливості машинного навчання:

* навчання з вчителем (supervised learning);
* навчання без вчителя (unsupervised);
* навчання з підкріпленням (reinforcement leaning).

Навчання з вчителем – передбачає, що для кожного вхідного вектора існує вектор вихідних значень. Разом ці два вектора називають навчальною парою, а множину навчальних пар – навчальною вибіркою. Процес навчання зводиться до почергового подавання на вхід нейронної мережі навчальних пар, вираховування похибки між дійсним і бажаним значенням нейронної мережі, та корегування параметрів мережі в бік зменшення цієї похибки.

Навчання без вчителя – один із способів машинного навчання, коли випробовувана система навчається виконувати поставлене завдання без втручання з боку експериментатора. Як правило, це підходить тільки для задач, в яких відомий опис множини об'єктів (навчальна вибірка), і необхідно виявити внутрішні взаємозв'язки, залежності, закономірності, що існують між об'єктами [19].

Навчання з підкріпленням є проміжним варіантом двох попередніх парадигм. Замість «вчителя» в схему навчання вводиться блок «критика», який відслідковує реакцію середовища на вхідний сигнал і опираючись на нього визначає евристичну похибку, яку покладено в процес навчання мережі.

Теорія навчання розглядає три фундаментальні властивості, пов’язані з навчанням за прикладами: ємність, складність зразків і обчислювальна складність. Під ємністю розуміється, скільки зразків може запам’ятати мережа, і які функції і межі ухвалення рішень можуть бути на ній сформовані. Складність зразків визначає число навчальних прикладів, необхідних для досягнення здатності мережі до узагальнення. Дуже мале число прикладів може викликати «перенавчання» мережі, коли вона добре функціонує на прикладах навчальної вибірки, але погано – на тестових прикладах, які підлягають тому ж статистичному розподілу.

Відомі 4 основних типи правил навчання: корекція за помилкою, навчання Больцмана, Дельта-правило і метод зворотного поширення помилки.

Правило корекції за помилкою використовується при навчанні з вчителем. Для кожного вхідного прикладу заданий бажаний вихід d. Реальний вихід мережі у може не збігатися з бажаним. Це правило являє собою такий метод навчання, при якому вага зв'язку не змінюється до тих пір, поки поточна реакція перцептрона залишається правильною. При появі неправильної реакції вага змінюється на одиницю, а знак (+/-) визначається протилежним від знаку помилки. Навчання має місце тільки у разі, коли перцептрон помиляється. Відомі різні модифікації цього алгоритму навчання: метод корекції помилок без квантування, метод корекції помилок з квантуванням, метод корекції помилок з випадковим знаком підкріплення, метод корекції помилок з випадковими збуреннями.

Правило навчання Больцмана є стохастичним правилом навчання, яке виходить з інформаційних теоретичних і термодинамічних принципів. Метою навчання Больцмана є таке налаштування вагових коефіцієнтів, при якому стани видимих нейронів задовольняють бажаний розподіл вірогідності. Навчання Больцмана може розглядатися як спеціальний випадок корекції за помилкою, в якому під помилкою розуміється розбіжність кореляцій полягань в двох режимах [19].

Дельта-правило навчання було запропоновано Б. Уідроу й М. Е. Гоффом і найбільше відповідає одношаровим штучним нейронним мережам прямого поширення. Ідея його полягає в тому, що якщо під час навчання мережі можна встановити розбіжність між її бажаною й наявною реакціями, ця розбіжність може бути усунута або зменшена шляхом зміни певним чином вагових коефіцієнтів зв’язку.

Метод зворотнього поширення помилки – метод навчання багатошарового перцептрону. Це ітеративний градієнтний алгоритм, який використовується з метою мінімізації помилки роботи багатошарового перцептрону та отримання бажаного виходу. Основна ідея цього методу полягає в поширенні сигналів помилки від виходів мережі до її входів, в напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи. Барц і Охонін запропонували відразу загальний метод («принцип подвійності»), який можна застосувати до ширшого класу систем, включаючи системи з запізненням, розподілені системи, тощо [20].

Одним із методів для поліпшення процесу навчання штучних нейронних мереж є початкова ініціалізація ваг.

Окрім класичного методу, де ваги початково встановлюються у випадкову величину у проміжку [-1, 1], є такі методи ініціалізації [21]:

* Метод Ксавьє. Основна ідея цього методу – спростити проходження сигналу через шар під час як прямого, так і зворотного поширення помилки для лінійної функції активації. При обчисленні ваг цей метод спирається на імовірнісний розподіл (рівномірний чи нормальний) з дисперсією, яка дорівнює , де – кількість нейронів у вхідному шарі, а – кількість нейронів у наступному шарі.
* Метод Кайминга Хе – це варіація методу Ксавьє, більше підходить для функції активації ReLU, це компенсує той факт, що ця функція повертає нуль для половини області визначення. А саме, в цьому випадку .

### 3.1.2 Конфігурація вхідного шару

Під ігровим станом розуміється те, що бачить гравець і його суперники щодо свого оточення, тобто параметри, включаючи положення всіх видимих юнітів та їх характеристики. Положення визначається за допомогою відстані та кута між гравцем та суперниками та видимими юнітами.

Для визначення вхідного формату до нейронної мережі потрібно визначити, від яких ігрових параметрів юнітів у конкретному ігровому стані залежать дії гравця.

Розглянемо *особливості*, які треба взяти до уваги при визначенні вхідних даних до нейронної мережі.

1) У грі є два види автомобілей: суперник та міський транспорт. Реакція гравця на два цих види машин перед ним може бути різною. Так, якщо перед гравцем звичайна міська машина він може охайно об'їхати її, але якщо бачить перед собою машину суперника він може захотіти прискоритися аби обігнати її. Це означає, що потрібно надати можливість нейронній мережі відокремлювати два цих види автомобілей.

2) Реакція гравця також може залежати від відстані до автомобіля. Так, наприклад, якщо гравець під’їхав дуже близько до авто перед ним і встиг зреагувати – він певне пригальмує, а якщо знаходиться досить далеко, то або не встигне зреагувати або навмисно буде продовжувати деякий час прискорюватися. Тобто нейронна мережа повинна знати відстані до автомобілей.

3) Реакція гравця залежить від швидкості автомобіля, який він бачить перед собою. Наприклад, якщо якийсь автомобіль загальмував, а гравець їде на повній швидкості – автомобіль перед ним буде наближатися до нього значно швидше ніж, якби авто перед ним їхало б зі швидкістю гравця (у такому випадку авто не рухалося відносно гравця). Тобто нейронна мережа повинна враховувати швидкість, з котрою їде автомобіль нейронної мережі, а також швидкість авто, котрий у її полі зору.

4) Ще два важливих параметра, від котрих залежить реакція гравця – відстані до лівого та правого країв дороги. Гравець може мати тактику, наприклад, їзди у районі центру дороги, щоб мати більше простору для маневрів, чому і повинна наслідувати нейронна мережа.

5) Якщо виникає ситуація, що гравець не встигнув зреагувати та врізався у авто перед ним, машина гравця втратить швидкість та відстане від автомобіля перед ним. Але гравець може запам’ятати, що спереду буде автомобіль та їхати більш охайно. Це означає, що нейронній мережі потрібно надавати інформацію, стикалася або ні машина суперника, котрою вона керує з машиною, яку вона бачить.

У роботі *пропонується* такий *алгоритм* збору інформації щодо автомобілей довкола конкретної гоночної машини:

1. Із машини, відносно котрої програма збирає інформацію, випускається N проміней, кут між якими однаковий та рахується як ;

2. Якщо на відстані MAX\_OBSERVED\_DISTANCE у напрямку променя немає автомобіля, відстань до авто у напрямку цього променя виставляється рівною MAX\_OBSERVED\_DISTANCE. У протилежному випадку, якщо у напрямку променя є авто відстань до якого менше, інформація про цей автомобіль відносно машини нейронної мережі збирається та запам’ятовується.

У роботі пропонується MAX\_OBSERVED\_DISTANCE = 1300. Це значення отримане як відстань від нижнього лівого до правого верхнього кута вікна програми, яка відображає гру.

Такий алгоритм з одного боку простий у реалізації, дає змогу зменшити кількість променів, щоб пришвидшити роботу алгоритму, стискає інформацію, котра надається до нейронної мережі шляхом виключення автомобілей, які стоять далі відносно напрямку променя, ніж перша машина, що зустрілась з променем. Треба також відмітити, що такий алгоритм дуже легко розпаралелити та виповнювати у декількох потоках, якщо це необхідно.

Для відокремлення двох видів автомобілей та окремої обробки кожного з них нейронною мережею пропонується виконати алгоритм, описаний вище, для двох видів автомобілей. Тобто спершу проміні «бачать» лише місцеві авто, а потім вони фіксують лише гоночні автомобілі.

Далі розглянемо *параметри ігрового стану*, які передаються на вхід до нейронної мережі.

Відносно кожного автомобіля, у напрямку якого «дивиться» скануючий промінь, маємо три параметри: відстань, швидкість та флаг, який відображає інформацію, чи стикалась машина з цим авто раніше. Кількість променів дорівнює N. А також промені сканують довкілля два рази: перший раз для фіксування місцевих авто, другий – гоночних автомобілей. Також три параметри має сама машина, відносно якої збирається інформація: відстань до лівого та правого краю дороги, швидкість авто. Це означає, що загальна кількість параметрів рахується за такою формулою: C .

Для досягнення мети роботи пропонується взяти N=20, якщо взяти значення більше, програмна система буде повільно працювати на старих компьютерах, а якщо менше, то даних буде недостатньо для навчення нейронної мережі.

Тобто фінальна кількість параметрів становить C=123, яка дорівнює кількості входів нейронної мережі. Маємо структуру датасету вхідних даних, відображену у таблиці 3.1.

Таблиця 3.1 – Структура датасету вхідних даних

|  |  |
| --- | --- |
| Дані | Кількість у датасеті |
| Швидкість поточного (гравця та суперників) гоночного автомобіля | 1 |
| Відстань до лівого краю дороги поточного автомобіля | 1 |
| Відстань до правого краю дороги поточного автомобіля | 1 |
| Швидкість спостережуваного авто |  |
| Відстань до спостережуваного авто | 120 |
| Чи стикалося поточне авто із спостережуваним раніше |  |

### 3.1.3 Конфігурація вихідного шару

У грі при керуванні гоночним автомобілем можливі чотири дії, які можуть комбінуватися між собою:

* прискорення;
* гальмування;
* поворот наліво;
* поворот направо.

Отже, нейронна мережа повинна мати можливість на надану їй інформацію щодо довкілля відносно машини, якою вона керує, видавати дії котрі вона хоче примінити у даний проміжок часу. Це означає, що кількість нейронів на вихідному шарі повинна дорівнювати кількості можливих дій, а саме чотирьом.

### 3.1.4 Конфігурація проміжного шару

Є такі загальні поради щодо організації проміжкових шарів нейронної мережі [24]:

1. Для більшості задач достатньо одного проміжного шару.
2. Кількість прихованих нейронів має бути між розміром вхідного шару і розміром вихідного шару.
3. Кількість прихованих нейронів має становити 2/3 розміру вхідного шару плюс розмір вихідного шару.
4. Кількість прихованих нейронів має бути менше ніж в два рази більше розміру вхідного шару.

Отже, враховуючи ці загальні рекомендації, пропонується створити нейронну мережу із одним проміжковим шаром, кількість нейронів у котрому дорівнює 2/3 від кількості нейронів у вхідному шарі, плюс кількість нейронів у вихідному шарі. Тобто, кількість нейронів у єдиному проміжному шарі дорівнює C=2/3 \* 123 + 4 = 86 нейронів.

На рисунку 3.1 представлена фінальна конфігурація нейронної мережі суперників, що навчаються на базі дій гравця.

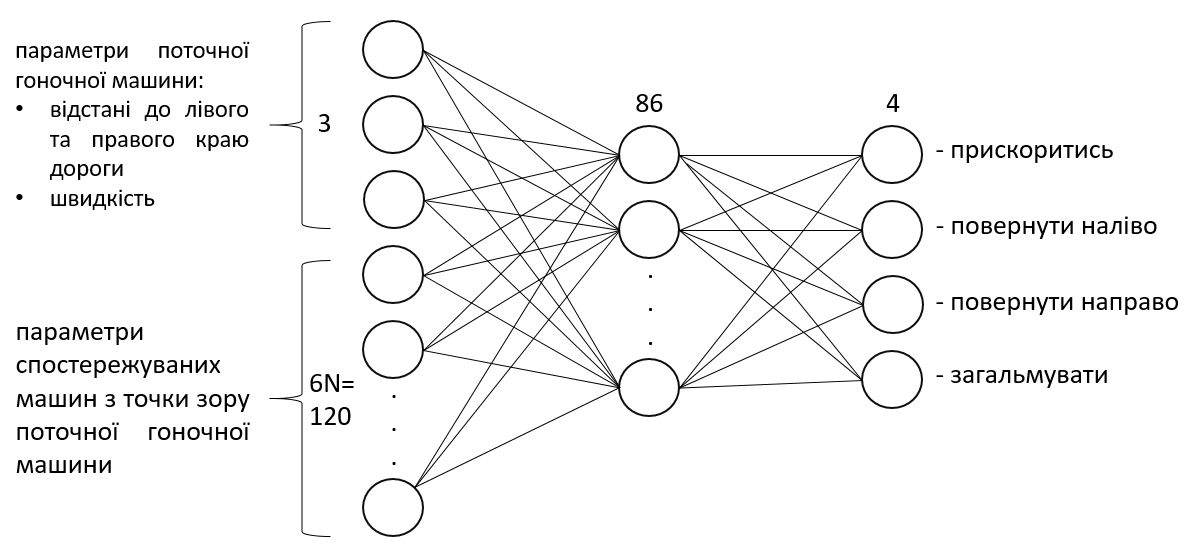


Рисунок 3.1 – Конфігурація нейронної мережі

## 3.2 Міра адаптованості нейронної мережі суперників до гри гравця

У попередньому розділі була підібрана конфігурація нейронної мережі не враховуючи функцію активації. Це було зроблено за допомогою загальних рекомендацій. Але підбір функції активації, засобів навчання та інших параметрів нейронної мережі найчастіше має бути експериментальним. Тобто для виявлення найкращої навченої нейронної мережі для даної задачі потрібно розробити оцінку її ефективності.

Оскільки мета даної роботи є розробка штучного інтелекту, рівень гри котрого буде адаптуватися під рівень гри гравця та грати на його рівні, то у цьому випадку, якщо усі гравці, включаючи ботів, котрі керуються комп’ютером, та користувач, повинні мати однакову вірогідність зайняти кожне можливе у результаті гонки місце. Позначимо цю вірогідність як (теоретична вірогідність):

де n – кількість усіх гравців у грі (суперники та користувач).

Маємо кількість можливих місць у результаті гонки n та теоретичну вірогідність зайняти кожне з них . Тоді VT = () = (), |VT| = n, – це точка у гіперпросторі розмірністю n, яка представляє собою теоретичні вірогідності зайняти місця у гонках з точку зору досягненої мети у роботі.

Тепер позначимо – як фактичну вірогідність зайняти і-те місце. Цю вірогідність можна порахувати, наприклад за формулою:

Ci – скільки разів було зайнято і-те місце,

CTotal – скільки усього було ігрових сесій.

Тепер маємо VF = () = (), |VF| = n – точка у гіперпросторі розмірністю n, яка представляє собою фактичні вірогідності зайняти місця у гонках, які були пораховані після проведення експерименту.

У якості оцінки конфігурації нейронної мережі, будемо використовувати відстань від точки з фактичними вірогідностями до точки з теоретичними вірогідностями:

.

## 3.3 Кроки розробленого методу

### 3.3.1 Збір бази дій гравця

Для тренування нейронних мереж суперників є необхідність збору даних про ігрові стани під час ігрової сесії та дії гравця, що відповідають кожному ігровому стану, тобто дані для нейронної мережі подаються у вигляді пар: (ігровий стан, дії гравця у відповідь). Для можливості регулювання кількості FPS (Frames Per Second, кількість кадрів у секунду) у роботі пропонується зберігати ігровий стан довкола гравця, а також примінені їм на даний час дії кожні 500 мілісекунд. Це значення було підібране таким чином, щоб знайти компроміс між кількістю зібраних даних та швидкістю роботи системи. Якщо взяти менший час, то даних будет більше, але система буде працювати менш швидче, а я якщо взяти час більше, то система буде працювати швидче, але даних буде недостатньо. Також для подальшої обробки у роботі використовуються дані за останні три пройдені рівні. Це робиться для усереднення рівня гри гравця, так щоб уникнути випадку, коли одну ігрову сесію гравець пограв дуже погано, програмна система навчила нейронні мережі грати теж погано, але у наступному рівні гравець зіграв навпаки дуже добре. Якщо ж ми використовуємо данні за останні декілька рівнів, ми маємо можливість взяти середній рівень гри, який показав гравець за ці останні ігрові сесії.

### 3.3.2 Нормалізація зібраних даних

Нормалізація даних дозволяє масштабувати числові значення в зазначений діапазон. Нижче представлені найбільш поширені методи нормалізації [23]:

• нормалізація методом мінімакса: лінійне перетворення даних в діапазоні, наприклад, від 0 до 1, де мінімальне і максимальне значення масштабованих даних відповідають 0 і 1 відповідно;

• нормалізація за Z-показником: масштабування даних на основі середнього значення і стандартного відхилення: ділення різниці між даними і середнім значенням на стандартне відхилення;

• десяткове масштабування: масштабування даних шляхом видалення десяткового роздільника значення атрибута.

Оскільки у грі заздалегіть відомі усі максимальні значення, найбільш зручним методом нормалізації у даній роботі буде нормалізація методом мінімакса. Таким чином, всі вхідні дані до нейронної мережі, показані в таблиці 3.1, нормалізуються, тобто кожне вхідне значення ділиться на його можливе максимальне значення. Так, наприклад, швидкість гоночного автомобіля ділиться на максимальну можливу швидкість гоночного авто, а відстань до спостережуваного автомобіля суперника ділиться на максимальну довжину MAX\_OBSERVED\_DISTANCE скануючого проміня. Після цього дані становляться нормалізованими.

### 3.3.3 Фільтрація зібраних даних

Оскільки гра дозволяє застосовувати декілька дій одночасно, є вірогідність, що гравець використовує одну або декілька із дій частіше, ніж інші. Наприклад, гравець може частіше хотіти прискоритися, ніж пригальмувати. У такому випадку, якщо тренувати нейронні мережі на таких даних, вони можуть «завчити» у будь яких ситуаціях використовувати ці самі найчастіші дії. Щоб уникнути цієї ситуації і навчити нейронні мережі, у яких саме ситуаціях гравець використовує кожну із дій, потрібно зробити фільтрацію початково зібраних даних. Мета такої фільтрації – залишити якомога більш рівну кількість використаних усіх дій у виборці.

Існують різні методи, за допомогою яких можна зменшити розмір даних (відфільтрувати) для спрощення обробки даних. Можна застосувати такі методи [23]:

* Вибірка записів: створення вибірки записів даних і вибір репрезентативної підмножини із загального набору даних.
* Вибірка атрибутів: вибір в даних набору найважливіших атрибутів.
* Агрегування: поділ даних на групи і зберігання числових значень для кожної групи.

Оскільки у вхідних даних вже зібрані найважливіші атрибути – у даному випадку слід примінити метод вибірки записів.

У роботі розроблений алгоритм фільтрації даних, який має такі константи та змінні:

* MinimumActionCount – змінна, яка розраховується в алгоритмі та означає мінімальну кількість кожної дії, яка повинна залишитися у відфільтрованому датасеті у випадку, коли загальна кількість конкретної дії більша MinimumActionCount, інакше у відфільтрований датасет включаються усі дії цього типу із загального датасету.
* MinimumActionDatasetSize – це константа, яка означає мінімальне допустиме значення змінної MinimumActionCount. Чим вона більше – тим довше працює навчання, але тим більше даних буде дано для навчання. У роботі пропонується MinimumActionDatasetSize = 10, це значення підходить до довжини траси у грі, а також частоти збору даних, для подальшого навчання нейронних мереж.
* MAX\_INT – максимальне можливе цілочисельне число.

На вхід алгоритму подається вхідний датасет у вигляді набору пар (ігровий стан, дія гравця у відповідь).

Алгоритм фільтрації даних виконується за наступними кроками:

1. Підраховуємо кількість кожної дії у вхідному датасеті, позначемо кількість i-тої дії як .

2. Розраховуємо MinimumActionCount за формулою ( – проміжне значення для і-тої дії):

3. Відсортуємо дії із зростанням . Почнемо перебирати їх з першої (найбільш рідкої) до останньої (найбільш частішої).

4. Якщо лічильник взятих дій для поточної дії більше або рівний MinimumActionCount, переходимо до наступної за зростанням частоти дії, інакше, якщо це не була остання дія, виконуємо крок 5, та переходимо до кроку 6 у іншому випадку.

5. Беремо із вхідного датасету пару (ігровий стан, дія гравця у відповідь), котра має поточно застосовану дію. Збільшуємо лічильник взятих дій для усіх дій, котрі були застосовані у взятій парі. Перехід до кроку 4.

6. Перемішуємо усі відібранні дії.

У результаті роботи такого алгоритму отримаємо збалансований відфільтрований датасет, у якому найчастіші дії не будуть значно превалювати над іншими, і, таким чином, на навчання нейронної мережі будуть впливати всі дані, а не тільки найчастіші.

Розберемо описаний алгоритм на прикладі фрагменту датасета, який наведений у таблиці 3.2 (у даниму випадку не приведені ігрові стани).

Таблиця 3.2 – Приклад вхідних даних

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Прискорення | Поворот наліво | Поворот направо | Гальмування |
| 1 | 1 | 0 | 1 | 0 |
| 2 | 0 | 0 | 0 | 1 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 1 | 1 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 |
| 6 | 1 | 1 | 1 | 0 |
| 7 | 1 | 1 | 0 | 0 |

Тут 1 або 0 у стовпці дії означає, чи була застосована користувачем дія у цей момент часу.

У цьому прикладі візьмемо MinimumActionDatasetSize = 2 для того, щоб продемонструвати крайні випадки у роботі алгоритму на малому вхідному датасеті.

1. Порахуємо кількість кожної дії: прискорення – 6, поворот наліво – 4, поворот направо – 3, гальмування – 1.

1. Розраховуємо MinimumActionCount: T1 = 6, T2 = 4, T3 = 3, T4 = .

MinimumActionCount = min(T1, T2, T3, T4) = min(6, 4, 3, ∞) = 3;

3. Відсортуємо дії із зростанням кількості разів, котрі вони застосувалися у датасеті. У прикладі дії слід відортувати так: гальмування, поворот направо, поворот наліво, прискорення.

4. Заведемо лічильник взятих дій. Переберемо дії у відсортованому порядку та будемо брати пару (ігровий стан, дія гравця у відповідь), яка містить у собі поточну дію у випадку, якщо кількість взятих дій менше MinimumActionCount, інакше переходимо до наступної дії.

Підбираємо дію «гальмування». Лічильник взятих дій має вигляд, відображений у таблиці 3.4. Відфільтрований датасет відображений у таблиці 3.5.

Таблиця 3.4 – Кількість кожної взятої дії

|  |  |  |  |
| --- | --- | --- | --- |
| Ускорение | Наліво | Вправо | Гальмування |
| 0 | 0 | 0 | 1 |

Таблиця 3.5 – Взяті дії

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Прискорення | Поворот наліво | Поворот направо | Гальмування |
| 1 | 0 | 0 | 0 | 1 |

Підбираємо дію «поворот направо». Лічильник взятих дій має вигляд, відображений у таблиці 3.6. Відфільтрований датасет відображений у таблиці 3.7.

Таблиця 3.6 – Кількість кожної взятої дії

|  |  |  |  |
| --- | --- | --- | --- |
| Ускорение | Наліво | Вправо | Гальмування |
| 3 | 2 | 3 | 1 |

Таблиця 3.7 – Взяті дії

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Прискорення | Поворот наліво | Поворот направо | Гальмування |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 0 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 1 | 1 | 1 | 0 |

Підбираємо дію «поворот наліво». Лічильник взятих дій має вигляд, відображений у таблиці 3.8. Відфільтрований датасет відображений у таблиці 3.9.

Таблиця 3.8 – Кількість кожної взятої дії

|  |  |  |  |
| --- | --- | --- | --- |
| Ускорение | Наліво | Вправо | Гальмування |
| 4 | 3 | 3 | 1 |

Таблиця 3.9 – Взяті дії

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Прискорення | Поворот наліво | Поворот направо | Гальмування |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 0 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 1 | 1 | 1 | 0 |
| 5 | 1 | 1 | 0 | 0 |

Дію «прискорення» підбирати не потрібно, оскільки на данному етапі лічильник для цієї дії дорівен 4, тобто більше ніж MinimumActionCount.

6. Перемішуємо отриманий результат випадковим чином (рівномірний розподіл), отримаємо відфільтрований датасет, який наведений у таблиці 3.10.

Таблиця 3.10 – Перемішаний результат фільтрації

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Прискорення | Поворот наліво | Поворот направо | Гальмування |
| 1 | 1 | 1 | 1 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 1 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 1 |

### 3.3.4 Тренування нейронних мереж суперників

Після фільтрації зібраних даних можна починати процес навчання нейронних мереж. Коли гравець закінчує грати останній розблокований рівень, він відкриває наступний (якщо це не був останній), та система навчає нейронні мережі суперників грати на рівні гравця, використовуючи данні, які були зібранні про гру користувача за останні три рівня.

### 3.3.5 Застосування нейронних мереж суперників

Після того, коли нейронні мережі суперників були навчені, вони застосовуються для керування автомобілем. Реалізовано це таким чином:

1. Кожен ігровий цикл система надає інформацію про ігровий стан відносно кожної із машин суперників.
2. Система отримує на виході нейронних мереж необхідні дії у відповідь на ігровий стан та застосовує їх до автомобілей, котрими вони керують.

## 3.4 Висновки до розділу 3

Таким чином, метод адаптації ігрового штучного інтелекту в іграх жанру "Гонки" під рівень гравця застосовується під час гри і містить наступні етапи:

1. Нормалізація і фільтрація зібраних даних – виконується після збору даних між рівнями.

2. Тренування нейронних мереж суперників – виконується після збору даних між рівнями. Конфігурація нейронних мереж:

* + вхідний шар: 123 нейрона;
  + вихідний шар: 4 нейрона;
  + проміжний шар: 86 нейронів.

3. Застосування нейронних мереж суперників – виконується починаючи з рівня 2 на базі навчених нейронних мереж на даних не більше 3х останніх рівнів.

Для аналізу адаптованості гри під рівень гравця запропоновано оцінку, що являє собою суму квадратів різниці фактичної вірогідності зайняти і-те місце та теоретичної у випадку, коли рівень гри усіх персонажів однаковий.

# 4 ПРОЕКТУВАННЯ СИСТЕМИ

Важливим етапом розробки інформаційних систем є передпроектний аналіз, одним з ключових завдань якого є виявлення вимог до інформаційної системи, що будується. Формулювання вимог дозволяє створити закінчений опис поведінки системи, яку потрібно розробити, та список повністю сформульованих функціональних і нефункціональних вимог, що дає можливість точної оцінки етапів, планів та термінів проекту.

Специфікація вимог програмного забезпечення згідно до IEEE 830 (Software requirements specification) [13] – закінчений опис поведінки програми, яку потрібно розробити. Включає ряд користувальницьких сценаріїв, які описують всі варіанти взаємодії між користувачами і програмним забезпеченням.

Призначені для користувача сценарії є засобом представлення функціональних вимог. Окрім сценаріїв для користувача, специфікація також містить нефункціональні вимоги, які накладають обмеження на дизайн або реалізацію (такі як вимоги продуктивності, стандарти якості або проектні обмеження).

## 4.1 Опис нефункціональних вимог

Нефункціональні вимоги – це вимоги до [програмного забезпечення](https://uk.wikipedia.org/wiki/Програмне_забезпечення), які задають критерії для оцінки якості його роботи. На відміну від [функціональних вимог](https://uk.wikipedia.org/wiki/Функціональні_вимоги), нефункціональні вимоги визначають, якою система повинна бути [14].

Нефункціональні вимоги:

1. Надійність – застосування забезпечує користувачу інтерфейс для роботи із системою, тому про всі помилки та нештатні ситуації користувачу виводяться повідомлення.
2. Зручність використання – клієнт забезпечує користувачу інтерфейс для роботи із системою, тому він має бути зручним та привабливим. Інтерфейс має бути доступним, іконки, кнопки та надписи мають бути інтуїтивно зрозумілим та трактуватися однозначно. Елементи інтерфейсу мають легко сприйматися. Десктопне застосування має працювати швидко та інформувати користувача про те, що зараз відбулося або відбувається у системі.

Основними атрибутами якості є:

* Функціональність: система повинна бути здатною при заданих умовах виконувати поставлену задачу.
* Ефективність: система повинна підтримувати працездатність, бути стійкою до відмов.
* Зручність модифікації: система повинна бути гнучкою до змін, які в разі додання нового критерію не повинні викликати хвильового ефекту.

## 4.2 Опис функціональних вимог

Для того, щоб описати загальні вимоги до функціональної поведінки десктопного застосування AAIRace, яке представляє собою гру жанру «Гонки», застосуємо діаграму варіантів використання, представлену на рис. 4.1.

На діаграмі варіантів використання можна виділити одного актора, який взаємодіє з системою через варіанти використання: користувач системи, який має доступ до функціоналу системи.

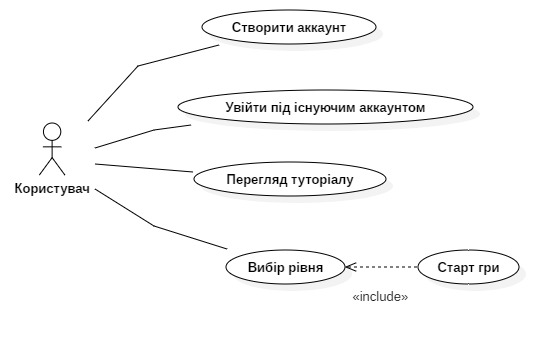


Рисунок 4.1 – Діаграма варіантів використання

Розглянемо докладно сценарії варіантів використання десктопного застосування, яке представляє собою гру жанру «Гонки».

**Варіант використання «Створити аккаунт»**

Діючі особи: Користувач.

Передумова: Система працює.

Мінімальні гарантії: Аккаунт не створено, система сповістить про проблему.

Гарантії успіху: Аккаунт створено, система покаже головне меню.

Тригер: Користувач натискає кнопку «Sign Up».

Основний успішний сценарій:

1. Користувач натискає кнопку «Sign Up».

2. Користувач вводить унікальне ім’я.

3. Користувач натискає кнопку «Create».

3.1. Система перевіряє коректність введеного імені.

3.2. Система завантажує пов’язані із аккаунтом дані.

3.3. Система показує головне меню.

**Варіант використання «Увійти під існуючим аккаунтом»**

Діючі особи: Користувач.

Передумова: Система працює.

Мінімальні гарантії: Дані пов’язані з аккаунтом не будуть завантажені, система сповістить користувача про помилку.

Гарантії успіху: Система завантажить дані пов’язані з аккаунтом та покаже головне меню.

Тригер: Користувач натискає кнопку «Sign In».

Основний успішний сценарій:

* + - 1. Користувач натискає кнопку «Sign In».
      2. Користувач вводить ім’я аккаунта.
      3. Користувач натискає кнопку «GO!».

3.1. Система перевіряє коректність введеного імені.

3.2. Система завантажує дані пов’язані з аккаунтом.

3.3. Система показує головне меню.

**Варіант використання «Перегляд туторіалу»**

Діючі особи: Користувач.

Передумова: Користувач увійшов у систему під аккаунтом, головне меню на екрані.

Мінімальні гарантії: Туторіал не буде показаний.

Гарантії успіху: Туторіал буде показаний.

Тригер: Користувач натиснув кнопку «Tutorial».

Основний успішний сценарій:

1. Користувач натиснув кнопку «Tutorial».

2. Система показує повідомлення із інформацією про керування гоночним автомобілем.

**Варіант використання «Вибір рівня»**

Діючі особи: Користувач.­­

Передумова: Користувач увійшов у систему під аккаунтом, головне меню показане на екрані.

Мінімальні гарантії: Екран вибору рівня не буде показаний.

Гарантії успіху: Буде виконаний варіант використання «Старт гри».

Тригер: Користувач натиснув кнопку «Play».

Основний успішний сценарій:

1. Система показує екран вибору рівня.

2. Користувач обирає один із розблокованих рівней.

3. Система виконує варіант використання «Старт гри».

**Варіант використання «Старт гри»**

Діючі особи: Користувач.

Передумова: Система виконала варіант використання «Вибір рівня».

Мінімальні гарантії: Інтерфейс гри не буде відображений.

Гарантії успіху: Інтерфейс гри буде відображений, користувач зможе керувати гоночним автомобілем.

Тригер: Користувач натискає кнопку з номером рівня.

Основний успішний сценарій:

1. Користувач натискає кнопку з вибором рівня.

1.1. Система завантажує дані пов’язані з рівнем.

1.2. Система показує інтерфейс гри.

1.3. Система дає можливість керувати автомобілем гравцю.

## 4.3 Архітектура системи

Програмна система складається із трьох основних компонентів: інтерфейс користувача, компоненти ядра гри, компонент штучного інтелекту. Структура системи та зв’язки між компонентами наведені у вигляді UML діаграми компонентів на рисунку 4.2.

Як показано на діаграмі, система складається з наступних високорівневих компонентів:

* Компонент інтерфейсу користувача – графічний інтерфейс користувача десктопної програми. Містить GUI класи, а також правила переходів між екранами – реалізація паттерну проектування «скінченний автомат».
* Компонент ядра гри – це компонент системи, у якому зберігається аккаунт користувача, а також код бізнес-логіки гри та її юнітів (авто, дорога, полоса прогресу, тощо).
* Компонент штучного інтелекту – це компонент системи, який складається із компонента бібліотеки нейронних мереж MiniDNN, а також містить алгоритми обробки даних перед тренуванням нейронних мереж, структури даних, які використовуються класами штучного інтелекту.

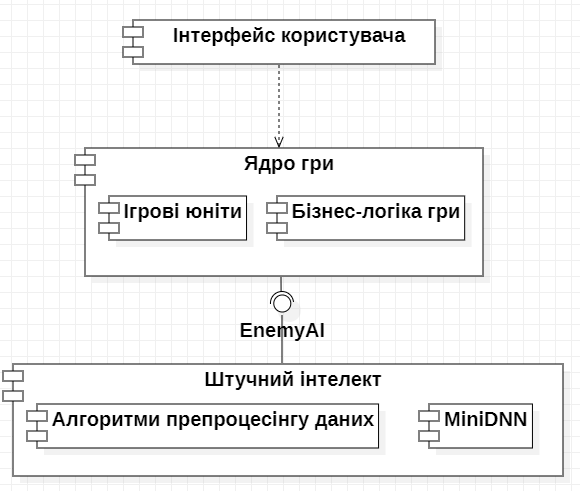


Рисунок 4.2 – Діаграма компонентів програмної системи

Десктопна програма, яка представляє собою гру жанру «Гонки», спроектована за архітектурним шаблоном MVC [15]. Цей шаблон застосовується для відокремлення даних системи від інтерфейсу користувача так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача. MVC ділить систему на три взаємопов'язані частини: модель даних, вигляд та модуль управління. Останні дві створюють інтерфейс користувача. Структура системи представлена діаграмою концептуальних класів, яка показана на рис. 4.3.

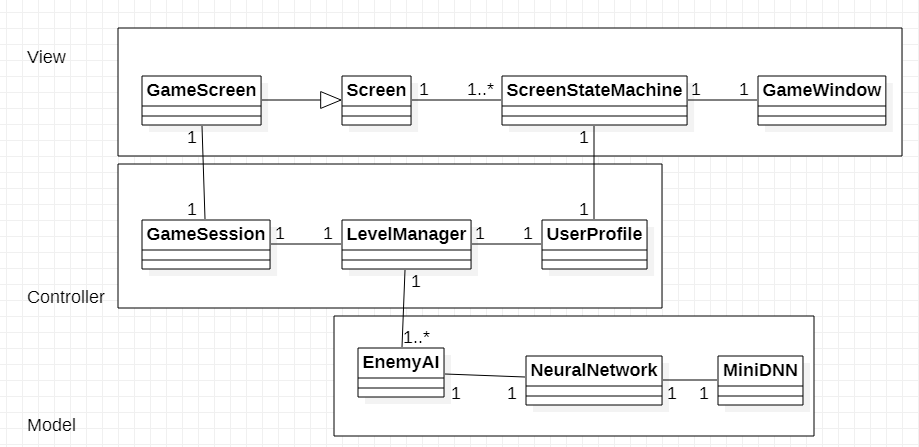


Рисунок 4.3 – Діаграма концептуальних класів

На діаграмі класи GameScreen, Screen, ScreenStateMachine, GameWindow – класи інтерфейсу системи. Вони відповідають за відображення даних у грі. Клас GameWindow – являє собою клас головного вікна програми, у якому показуються усі екрани, у тому числі, екран ігрової сесії. ScreenStateMachine – клас, який реалізує патерн проектування «стан». Він зберігає клас UserProfile, а також клас екрану, котрий відображається зараз у вікні. Клас Screen – інтерфейс до усіх класів екранів, одним з котрих є екран GameScreen. GameScreen – клас, який представляє собою екран, у якому відображається інтерфейс гри під час ігрової сесії.

Класи GameSession, LevelManager та UserProfile є контролерами системи. Класи GameSession та UserProfile – контролери елементів, які відповідають за зв’язок з елементами компонента вигляду. GameSession – контролює інтерфейс гри, UserProfile – контролює інтерфейс користувача у меню вибору рівня. LevelManager є контролером штучного інтелекту а також він генерує контролер GameSession. LevelManager також відповідний за дані про гру гравця, їх препроцесінг та подальшу їх передачу до класів-моделей EnemyAI для подальшої обробки та навчання нейронних мереж.

Класи EnemyAI, NeuralNetwork та MiniDNN являють собою модель системи. Клас EnemyAI являється класом-адаптером до класу NeuralNetwork, він відповідає за адаптацію даних від класу-контролера LevelManager до класу нейронної мережі NeuralNetwork. NeuralNetwork являє собою фасад до бібліотеки нейронних мереж MiniDNN. Клас MiniDNN – це основний клас бібліотеки MiniDNN, конфігуруючи котрий можна гнучно реалізовувати різні конфігурації штучних нейронних мереж та навчати їх.

## 4.4 Проектування інтерфейсу користувача

У цьому розділі зображені макети графічного інтерфейсу, що були створені перед початком розробки програми, а не графічний інтерфейс закінченої програми.

На рисунку 4.4 зображено початковий екран системи. Його призначення полягає у тому, щоб користувач увійшов у систему під існуючим аккаунтом або створив новий.

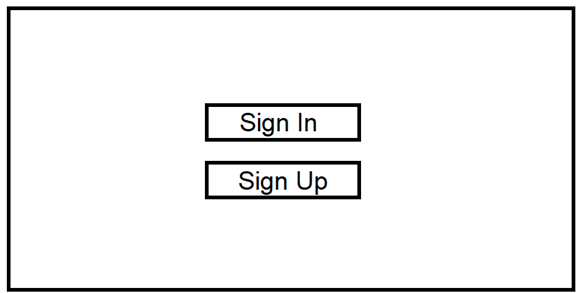


Рисунок 4.4 – Макет початкового екрану програми

Якщо користувач натискає кнопку «Sign In», він попадає на екран входу від існуючим аккаунтом, який зображений на рисунку 4.5. У випадку, коли користувач натискає кнопку «Sign Up», він попадає на екран реєстрації нового аккаунту, макет якого зображений на рисунку 4.6. У обох випадках користувач має можливість повернутися на екран початкового екрану, натиснувши клавішу «Back».

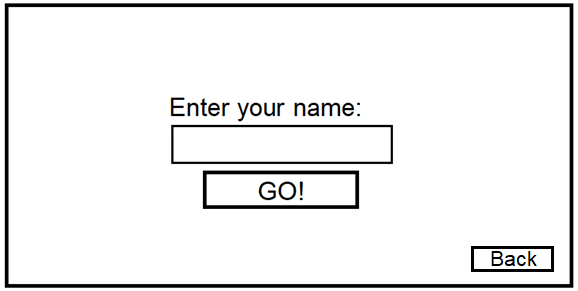


Рисунок 4.5 – Макет екрану входу під існуючим аккаунтом

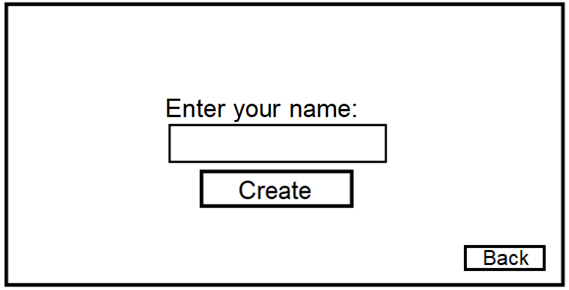


Рисунок 4.6 – Макет екрану створення нового аккаунту

Коли користувач натискає «GO!» або «Create» відповідно до того, на якому екрані користувач знаходиться, він потрапляє на екран головного меню гри, він зображений на рисунку 4.7.

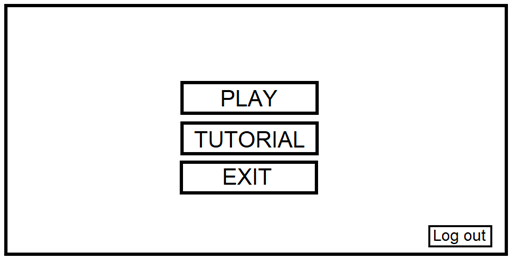


Рисунок 4.7 – Макет екрану головного меню гри

Якщо користувач натискає клавішу «Tutorial», йому показується спливаюче вікно із зображенням інструкції управління автомобілем. Якщо користувач натискає «Exit», вікно гри закривається. Якщо була натиснута кнопка «Play» – показується екран вибору рівня, його макет зображений на рисунку 4.8. Також користувач має можливість повернутися на початковий екран, натиснувши клавішу «Log out».

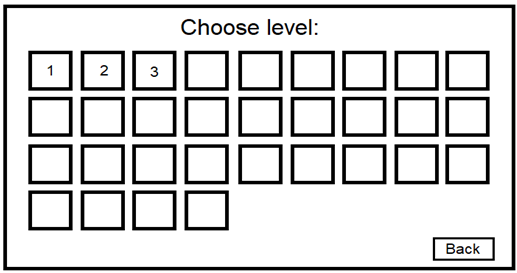


Рисунок 4.8 – Макет екрану вибору рівня

Користувач має можливість повернутись у головне меню, натиснувши «Back». Коли користувач натискає на кнопку з номером рівня, він переходить на екран ігрової сесії. Макет цього екрану зображений на рисунку 4.9.

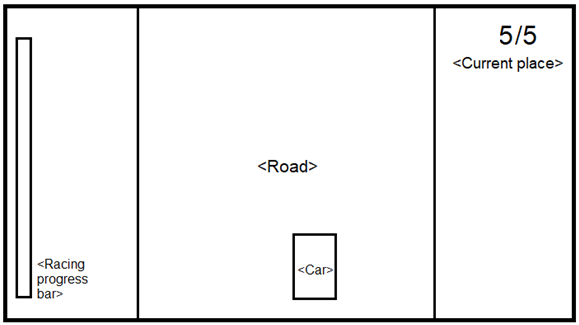


Рисунок 4.9 – Макет екрану ігрової сесії

Тут зображені такі основні деталі:

* <Road> – відображення дороги, по котрій їздить гравець на авто;
* <Car> – автомобіль гравця, котрим він керує;
* <Current place> – поточне місце у гонці;
* <Racing progress bar> – тут відображені положення автомобілей суперників та гравця відносно довжини траси.

## 4.5 Детальне проектування логічного представлення системи

На рисунку 4.10 представлена діаграма програмних класів частини архітектурного шаблону MVC, а саме Model.

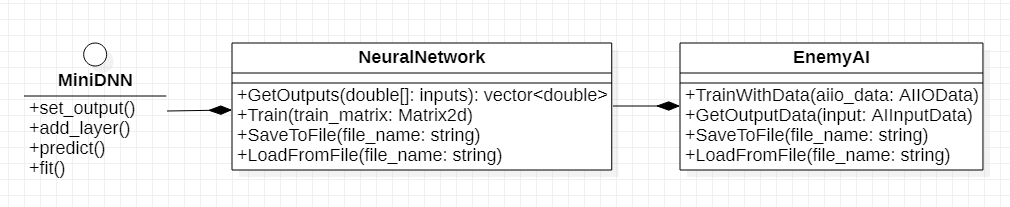


Рисунок 4.10 – Діаграма програмних класів частини Model

Тут інтерфейс MiniDNN представляє собою бібліотеку нейронних мереж MiniDNN, у нього є такі основні функції:

* set\_output() – встановити вихідний шар;
* add\_layer() – додати шар у конфігурацію нейронної мережі;
* predict() – екстраполювати вхідні дані до нейронної мережі у вихідні;
* fit() – тренувати нейронну мережу на матриці вхідних та вихідних даних.

Клас NeuralNetwork представляє собою фасад до бібліотеки MiniDNN, він має такі функції:

* GetOutputs – видає значення вихідного шару після надавання до нейронної мережі вхідних даних;
* Train – тренувати нейронну мережу на вхідних та вихідних даних;
* SaveToFile – зберегти нейрону мережу та її конфігурацію у файл;
* LoadFromFile – загрузити нейронну мережу із файлу.

Клас EnemyAI представляє собою адаптер від класів ядра гри до класу нейронної мережі NeuralNetwork. Він має такі функції:

* TrainWithData – тренувати нейронну мережу на вхідних та вихідних даних;
* GetOutputData – видає значення вихідного шару після надавання до нейронної мережі вхідних даних;
* SaveToFile – зберегти нейрону мережу та її конфігурацію у файл;
* LoadFromFile – загрузити нейронну мережу із файлу.

Далі на рисунку 4.11 зображена діаграма класів частини архітектурного шаблону «Controller».

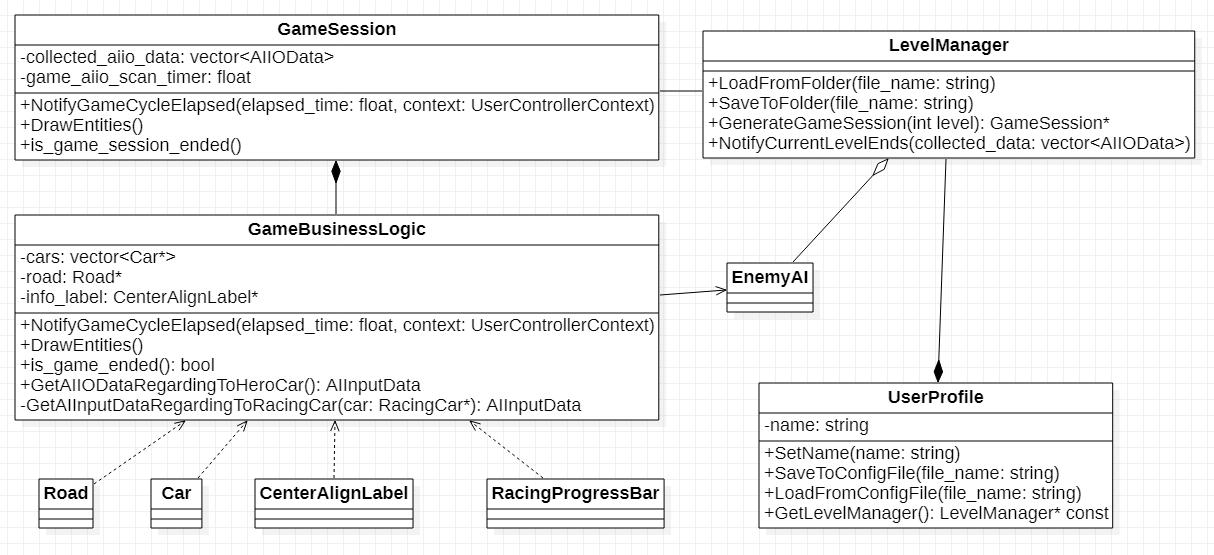


Рисунок 4.11 – Діаграма програмних класів частини Controller

Класи Road, Car, CenterAlignLabel, RacingProgressBar – елементи графічного інтерфейсу, котрі керуються класом GameBusinessLogic.

Клас GameBusinessLogic відповідає за бізнес-логіку гри, тобто він імплементує керування автомобілями суперників за допомогою наданих йому екземплярів класу EnemyAI. Також він зчитує пристрої вводу користувача для керування автомобілем гравця. В його відповідальність також входить опрацювання колізій, у тому числі, із фінішною прямою. Він має такі основні функції:

* NotifyGameCycleElapsed – повідомляє екземпляру класу, що був пройдений ігровий цикл і потрібно оновити зображення на екрані;
* DrawEntites – повідомляє, що потрібно відобразити елементи ігрового інтерфейсу на екрані;
* is\_game\_ended – повертає true, якщо гра закінчилась, false у іншому випадку;
* GetAIIODataRegardingToHeroCar – повертає інформацію про ігрове оточення гравця та його поточні дії. Використовується для збору інформації про гру гравця та подальшого навчання нейронних мереж;
* GetAIInputDataRegardingToRacingCar – повертає інформацію про ігрове оточення гоночної машини.

На рисунку 4.12 зображена діаграма класів програмної системи, яка представляє собою частину MVC – View (елементи UI).

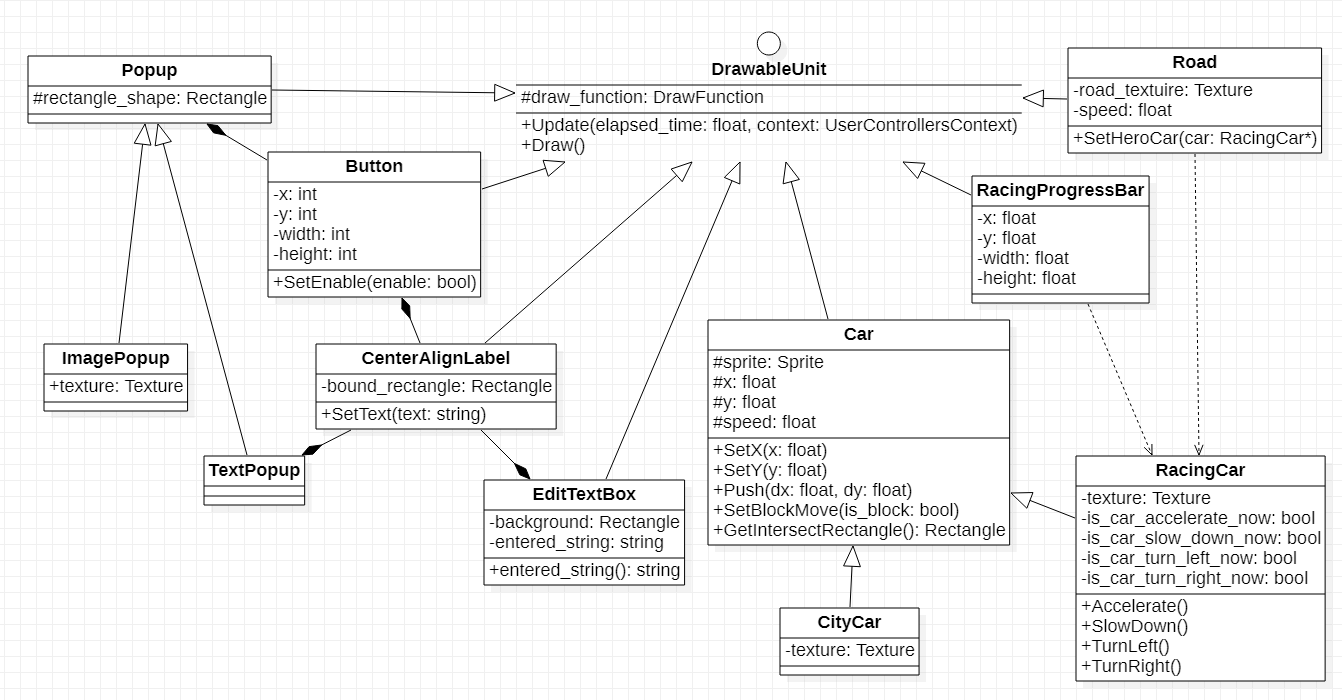


Рисунок 4.12 – Діаграма програмних класів частини View (елементи UI)

Клас DrawableUnit представляє собою інтерфейс для класів, які являються елементами графічного інтерфейсу. Він складається з таких абстрактних методів:

* Update – нотифікує елемент UI, що треба оновити свій стан для подальшої прорисовки;
* Draw – нотифікує елемент UI, що треба прорисувати себе на екрані.

Клас Popup імплементує елемент графічного інтерфейсу – спливаюче повідомлення, а саме, вікно з кнопкою «ОК».

Класи ImagePopup та TextPopup наслідують клас Popup. ImagePopup – додає до вікна із повідомленням зображення, а TextPopup – додає текст.

Клас Button імплементує кнопку. Він має єдиний основний метод:

* SetEnable – робить кнопку активною або не активною.

CenterAlignLabel – клас, який імплементує текст, котрий вирівнюється відносно центру прямокутника. Має єдиний основний метод:

* SetText – встановлює текст для відображення.

Клас EditTextBox представляє собою поле для вводу тексту. Він має єдиний метод:

* entered\_string – повертає введений користувачем текст.

Клас Car – імплементує автомобіль. Він має такі основні методи:

* SetX – встановлює Х координату;
* SetY – встановлює У координату;
* Push – товкає машину у заданому напрямку;
* SetBlockMove – блокує пересування авто;
* GetIntersectRectangle – повертає прямокутник колізії авто.

Класи RacingCar та CityCar наслідують клас Car. RacingCar – клас, який представляє собою гоночний автомобіль. Він має такі основні методи:

* Accelerate – виконати дію «прискорення»;
* SlowDown – виконати дію «гальмування»;
* TurnLeft – виконати дію «поворот наліво»;
* TurnRight – виконати дію «поворот направо».

Клас CityCar являє собою міську машину.

Клас RacingProgressBar представляє собою елемент ігрового інтерфейсу, який являє собою схематичне зображення положення гравця та суперників відносно гоночної траси.

Клас Road представляє собою дорогу у грі. Він має такий основний метод:

* SetHeroCar – встановлює машину гравця. Це потрібно для того, щоб дорога рухалася у напрямку гравця синхронно з його швидкістю.

Далі на рисунку 4.13 зображена діаграма програмних класів частини архітектурного шаблону MVC – View (система екранів).

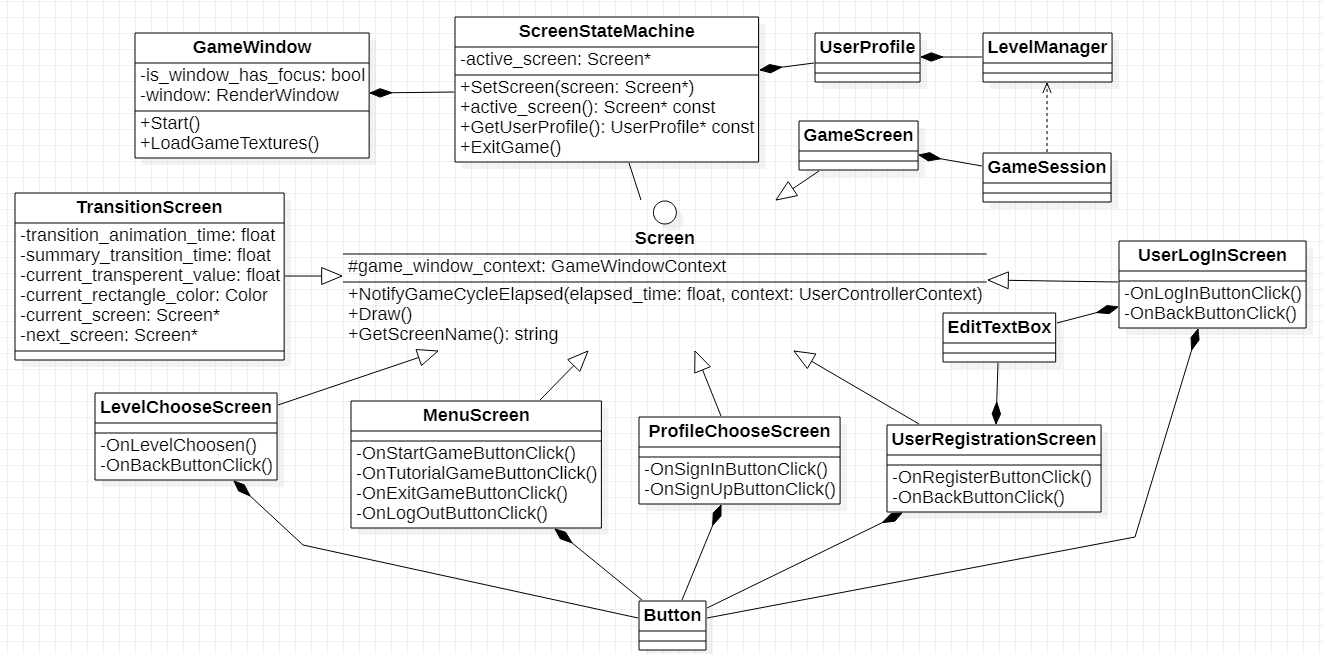


Рисунок 4.13 – Діаграма програмних класів частини View (система екранів)

Клас GameWindow представляє собою клас головного вікна програми. Має такі функції:

* Start – відкрити вікно;
* LoadGameTextures – загрузити усі шрифти, текстури, зображення гри.

Зв’язка класу ScreenStateMachine та класу-інтерфейсу Screen являє собою реалізацію шаблону паттерну проектування «стан». У даному випадку у системі таким чином реалізований перехід між різними екранами. Клас ScreenStateMachine має такі основні методи:

* SetScreen – встановити новий екран та закрити попередній;
* active\_screen – повернути вказівник на поточний активний екран;
* GetUserProfile – повернути вказівник на клас-профайл користувача;
* ExitGame – нотифікує, що користувач закриває гру.

Клас Screen представляє собою інтерфейс класів екранів. Він має такі методи:

* NotifyGameCycleElapsed – нотифікує екран про необхідність оновлення своїх елементів;
* Draw – нотифікує клас екрану про те, що треба прорисовувати елементи на екрані;
* GetScreenName – повертає унікальне ім’я екрану.

TransitionScreen – клас екрану, який представляє собою анімацію переходу між поточним та наступним екраном.

LevelChooseScreen – клас екрану, який представляє собою екран вибору рівня. Має такі два методи-колбеки:

* OnLevelChoosen – нотифікує екран про те, що користувач натиснув на кнопку з номером рівня;
* OnBackButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Back».

MenuScreen – клас екрану, який являє собою головне меню. Має чотири функції-колбеки:

* OnStartGameButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Play»;
* OnTutorialButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Tutorial»;
* OnExitGameButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Exit»;
* OnLogoutButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Log out».

ProfileChooseScreen – клас-екран, який являє собою початковий екран вибору методу входу у гру. Має два колбеки:

* OnSignInButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Sign in»;
* OnSignUpButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Sign up».

Клас UserRegistrationScreen – це екран реєстрації нового користувача. Має два колбеки:

* OnRegisterButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Create»;
* OnBackButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Back»;

Клас UserLoginScreen – це екран входу під існуючим аккаунтом. Має два колбеки:

* OnLoginButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «GO!»;
* OnBackButtonClick – нотифікує екран про те, що користувач натиснув на кнопку «Back»;

Клас GameScreen – це екран ігрової сесії для відображення елементів гри.

# 5 РЕАЛІЗАЦІЯ ТА ВИПРОБУВАННЯ СИСТЕМИ

## 5.1 Опис програмних технологій

Для розробки програми був використаний швидкий редактор коду Visual Studio Code, який представляє собою інструмент для написання та редагування коду. У якості мови програмування була обрана мова С++14, оскільки написання на цій мові дозволяє розроблювати швидкі та оптимізовані з точки зору часу та пам’яті програми. Для сбірки проекту був використаний інструмент для сбірки MVS проектів – MSBuild. Також для розробки графічної складової програми був використаний фреймворк – SFML, який являє собою фасад до API OpenGL.

Для тестування програми під час розробки був використаний ПК із операційною системою Windows 10.

## 5.2 Інструкція з встановлення

Для встановлення десктопної програми, яка представляє собою гру жанру «Гонки», треба:

* завантажити архів, який містить exe-файлы, а також DLL-бібліотеки;
* розпакувати архів;
* запустити та встановити VC++ redistributable 2015.

Розглянемо процес встановлення програми на пристрій з операційною системою Windows 10.

Завантажили архів, знаходимо його у файловій системі пристрою та розпаковаємо у поточну папку. Запускаємо інсталяційний файл VC++ redistributable 2015. Приймаємо вимоги ліцензії та натискаємо «встановити».

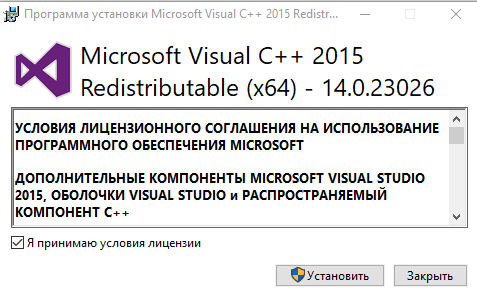


Рисунок 5.1 – Перший етап інсталяції

Після закінчення інсталяції матимемо готове до роботи застосування. Відкривши його побачимо:



Рисунок 5.2 – Встановлене застосування

## 5.3 Інструкція з використання

Після натиску на кнопку «Sign In» відкриється екран входу в систему під існуючим ім’ям, який зображений на рисунку 5.3. Також з цього екрану можна перейти на екран реєстрації, який зображений на рисунку 5.4.

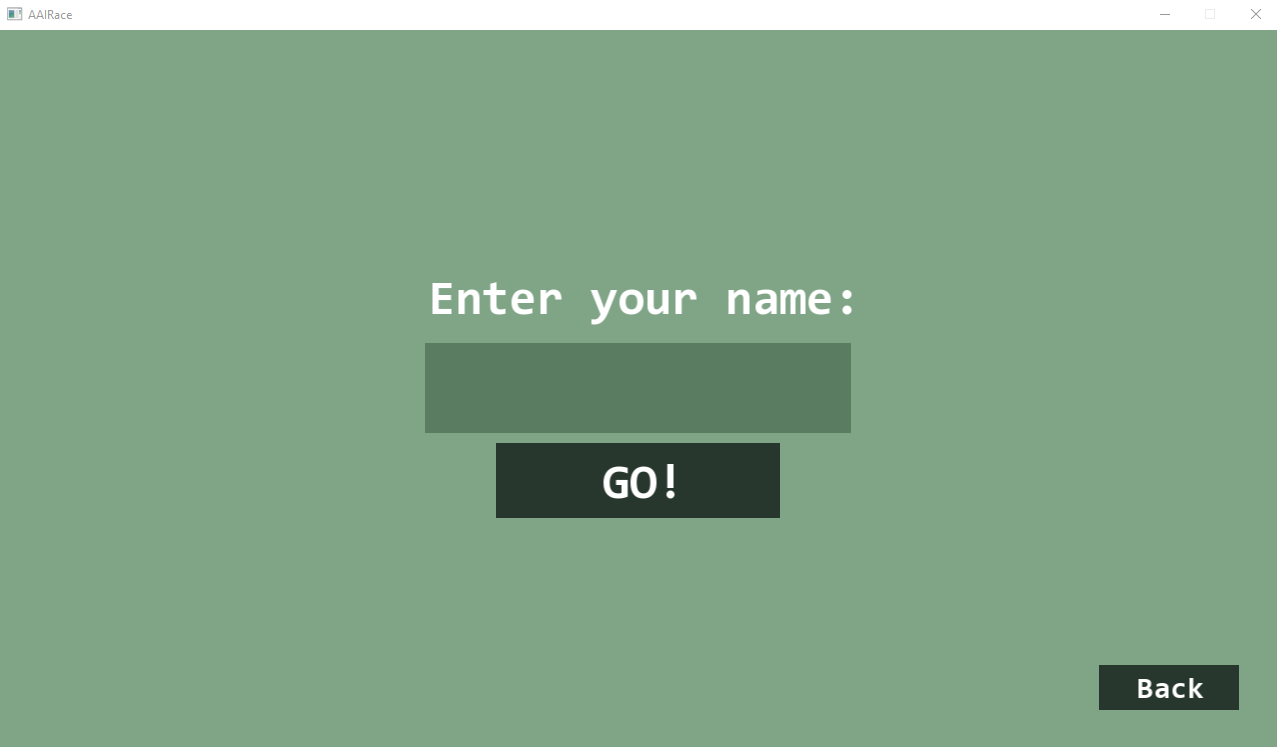


Рисунок 5.3 – Екран входу у систему під існуючим ім’ям

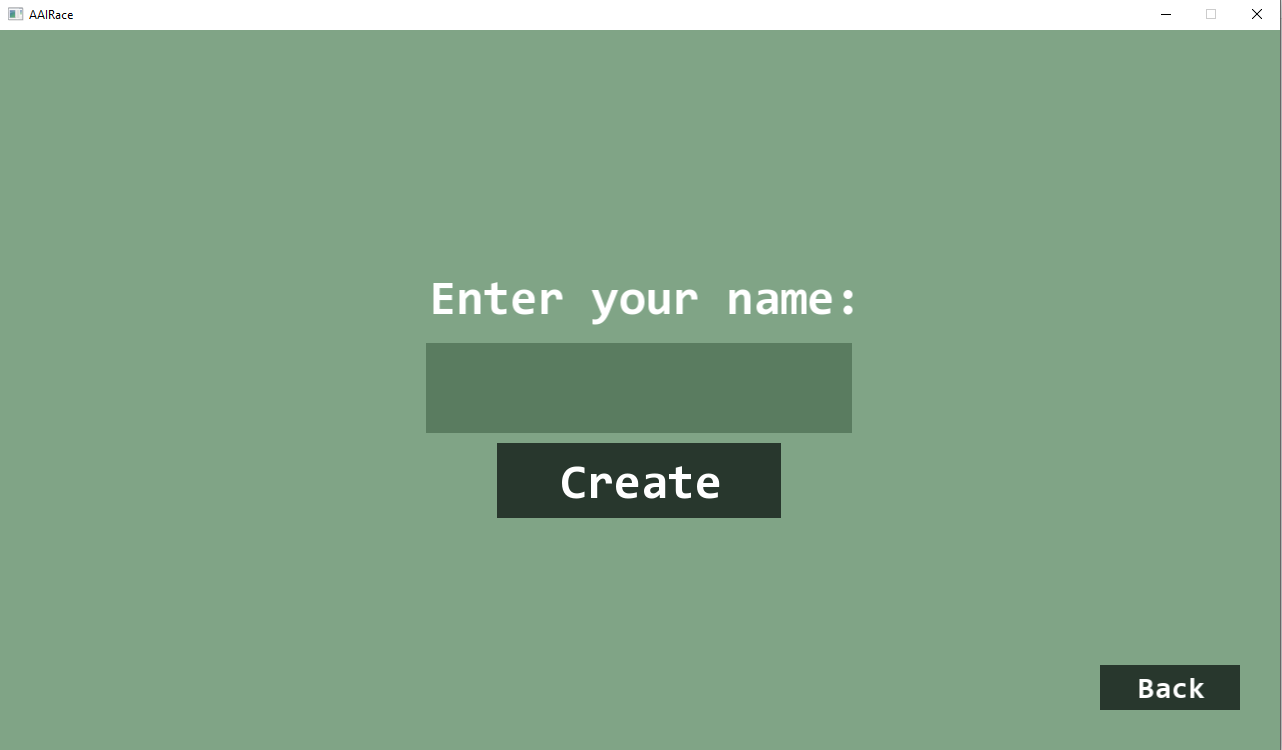


Рисунок 5.4 – Екран реєстрації нового користувача

Після входу у систему одним з описаних вище двох способів користувач попадає на екран головного меню, який зображений на рисунку 5.5.



Рисунок 5.5 – Екран головного меню

Якщо користувач натискає «Play», він попадає на екран вибору рівня, який зображений на рисунку 5.6. Якщо користувач натискає кнопку «Tutorial» – показується повідомлення із позначенням клавіш для керування автомобілем, який зображено на рисунку 5.7. Натиснувши кнопку «Exit» – вікно програми закриється.

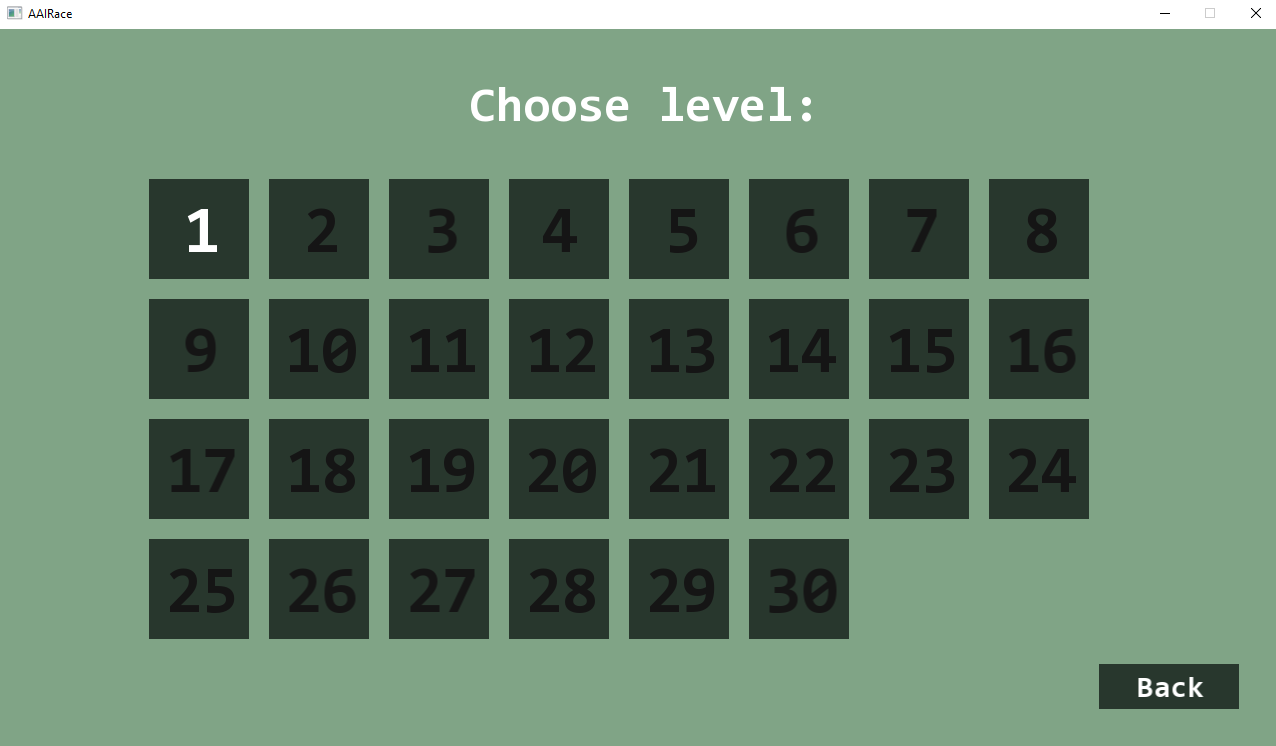


Рисунок 5.6 – Екран вибору рівня



Рисунок 5.7 – Повідомлення з туторіалом

Коли користувач обирає рівень, натиснувши кнопку з числом на екрані вибору рівня, він попадає у інтерфейс гри, який виглядає так як зображено на рисунку 5.8.

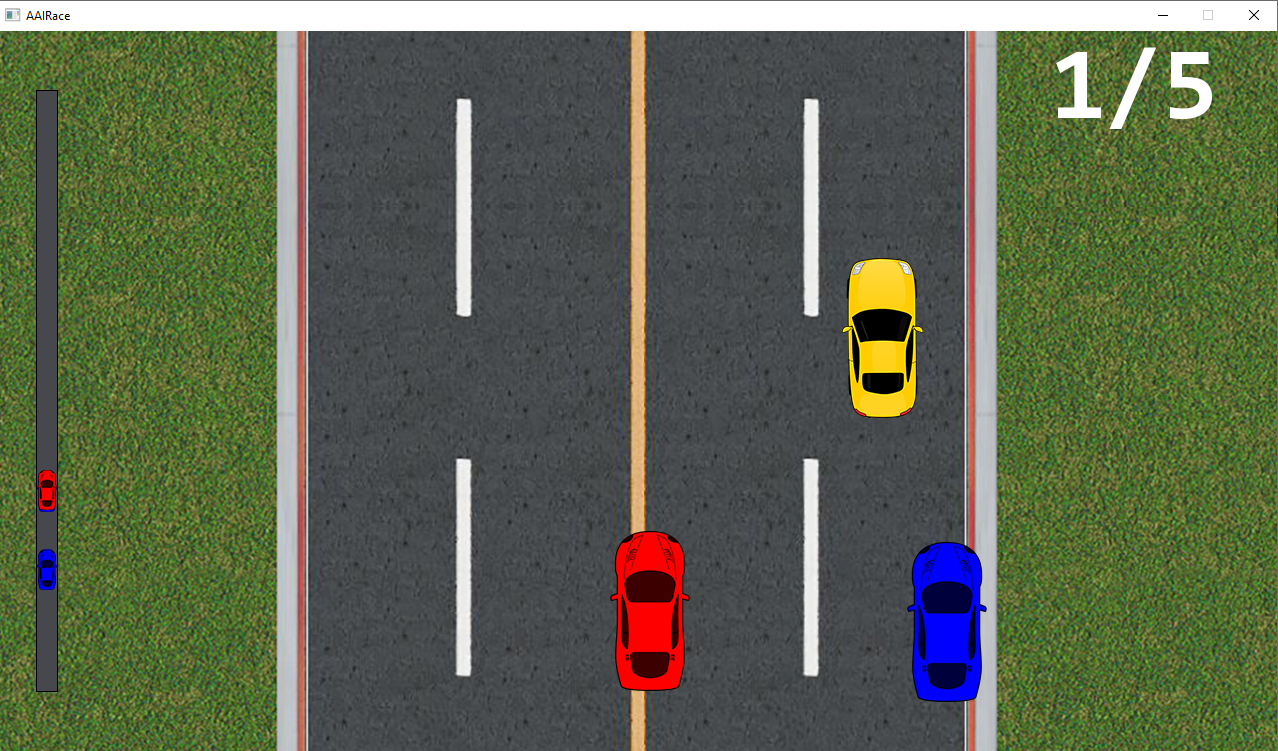


Рисунок 5.8 – Екран ігрової сессії

По центру екрану зображена дорога, по якій їдуть автомобілі. Зліва зображена полоса прогресу гонки, а також положення гоночних автомобілей відносно усієї довжини дороги та одне одного. У верхньому правому куту показується поточне місце гравця у гонці.

## 5.4 Функціональне тестування

Для проведення функціонального тестування програмного продукту необхідно скласти тестові сценарії на основі варіантів використання з урахуванням сценаріїв (табл. 5.1 – 5.5).

Таблиця 5.1 – Тестування сценарію «Створити аккаунт»

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | Очікуваний результат | Результат |
| 1 | Користувач знаходиться на екрані входу у систему | Система показує дві кнопки: «Sign In» та «Sign Up» | Система показує дві кнопки: «Sign In» та «Sign Up» |
| 2 | Користувач натискає кнопку «Sign Up» | Система показує поле вводу для імені, а також кнопку «Create» | Система показує поле вводу для імені, а також кнопку «Create» |
| 3 | Користувач вводить унікальне ім’я | Система відображає введене ім’я у полі вводу | Система відображає введене ім’я у полі вводу |
| 4 | Користувач натискає кнопку «Create» | Система показує головне меню програми | Система показує головне меню програми |

Таблиця 5.2 – Тестування сценарію «Увійти під існуючим аккаунтом»

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | Очікуваний результат | Результат |
| 1 | Користувач знаходиться на екрані входу у систему | Система показує дві кнопки: «Sign In» та «Sign Up» | Система показує дві кнопки: «Sign In» та «Sign Up» |

Продовження таблиці 5.2.

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | Користувач натискає кнопку «Sign In» | Система показує поле вводу для імені, а також кнопку «GO!» | Система показує поле вводу для імені, а також кнопку «GO!» |
| 3 | Користувач вводить зареєстроване ім’я у системі | Система відображає введене ім’я у полі вводу | Система відображає введене ім’я у полі вводу |
| 4 | Користувач натискає кнопку «GO!» | Система показує головне меню програми | Система показує головне меню програми |

Таблиця 5.3 – Тестування сценарію «Перегляд туторіалу»

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | Очікуваний результат | Результат |
| 1 | Користувач знаходиться на екрані головного меню | Система показує три кнопки: «PLAY», «TUTORIAL», «EXIT». | Система показує три кнопки: «PLAY», «TUTORIAL», «EXIT». |
| 2 | Користувач натискає кнопку «TUTORIAL» | Система показує повідомлення із зображенням управління автомобілем | Система показує повідомлення із зображенням управління автомобілем |

Таблиця 5.4 – Тестування сценарію «Вибір рівня»

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | Очікуваний результат | Результат |
| 1 | Користувач знаходиться на екрані головного меню | Система показує три кнопки: «PLAY», «TUTORIAL», «EXIT». | Система показує три кнопки: «PLAY», «TUTORIAL», «EXIT». |
| 2 | Користувач натискає кнопку «PLAY» | Система меню вибора рівня гри | Система меню вибора рівня гри |
| 3 | Користувач обирає одиз із розблокованих рівнів | Система показує екран ігрової сесії | Система показує екран ігрової сесії |

Таблиця 5.5 – Тестування сценарію «Старт гри»

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дія | Очікуваний результат | Результат |
| 1 | Користувач знаходиться на екрані головного меню | Система показує три кнопки: «PLAY», «TUTORIAL», «EXIT». | Система показує три кнопки: «PLAY», «TUTORIAL», «EXIT». |
| 2 | Користувач натискає кнопку «PLAY» | Система меню вибора рівня гри | Система меню вибора рівня гри |
| 3 | Користувач обирає одиз із розблокованих рівнів | Система показує екран ігрової сесії | Система показує екран ігрової сесії |
| 4 | Користувач натискає кнопки: «вліво», «вправо», «назад», «уперед». | Гоночний автомобіль гравця відповідно: повертає вліво, повертає направо, гальмує та прискорюється | Гоночний автомобіль гравця відповідно: повертає вліво, повертає направо, гальмує та прискорюється |

## 5.5 Випробування системи і аналіз результатів

Метою розробки програмної системи, яка являє собою гру жанру «Гонки», було підвищення зацікавленості гравця в іграх жанру «Гонки» на основі машинного навчання моделей суперників на діях гравця. У попередніх розділах була розроблена архітектура нейронної мережі. Для того, щоб підібрати функцію активації у фінальну версію програми та для оцінки розробленої системи, було проведено експерименти, результати яких представлено нижче.

Для підбору функції активації проведемо випробування різних конфігурацій нейронної мережі у грі, у якості оцінки кожної з отриманих конфігурацій будемо використовувати описаний раніше показник D2, чим він менший тим краще. У таблиці 5.6 наведені результати експерименту.

Таблиця 5.6 – Результати випробування для різних функцій активації

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Функції активації | | | | | | | |
| Місце | Sigmoid | | Tanh | | SoftPlus | | ReLU | |
|  | C | P | C | P | C | P | C | P |
| 1 | 17 | 0.41 | 16 | 0.40 | 14 | 0.3 | 12 | 0.29 |
| 2 | 8 | 0.19 | 10 | 0.24 | 10 | 0.25 | 10 | 0.25 |
| 3 | 8 | 0.2 | 6 | 0.15 | 6 | 0.16 | 8 | 0.19 |
| 4 | 6 | 0.13 | 4 | 0.1 | 5 | 0.15 | 6 | 0.14 |
| 5 | 3 | 0.07 | 4 | 0.1 | 4 | 0.11 | 6 | 0.15 |
| D2 | 0.066 | | 0.068 | | 0.024 | | 0.016 | |

Тут С – кількість разів для кожної конфігурації, скільки було зайняте і-те місце, Р – розрахована вірогідність зайняти і-те місце для кожної з протестованих функцій активації.

З наведених вище результатів тестування можна зробити висновок, що для даної у роботі мети найкраще підходить функція активації – ReLU, оскільки конфігурація саме із цією функцією отримала найменший показник D2. Оскільки оптимальна функція активації – ReLU, можна примінити метод ініціалізації ваг ШІ Кайминга Хе (розглянуто у розділі 3). Після застосування методу Кайминга Хе маємо результати тестування, наведені у таблиці 5.7.

Таблиця 5.7 – Результати випробування для різних методів ініціалізації ваг

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Місце | ReLU (нормальний розподіл) | | ReLU (Каймінг Хе) | |
|  | C | P | C | P |
| 1 | 12 | 0.29 | 11 | 0.27 |
| 2 | 10 | 0.25 | 10 | 0.25 |
| 3 | 8 | 0.19 | 8 | 0.2 |
| 4 | 6 | 0.14 | 7 | 0.16 |
| 5 | 6 | 0.15 | 5 | 0.11 |
| D2 | 0.016 | | 0.0149 | |

Можна помітити, що при всіх конфігураціях нейронної мережі, у тому числі і у випадку найденої оптимальної конфігурації, вірогідність зайняти і-те місце (і=1…5) знижується із ростом і. Це добре видно на графіку, який зображений на рисунку 5.9 для випадку ReLU + метод ініціалізації Кайминга Хе.

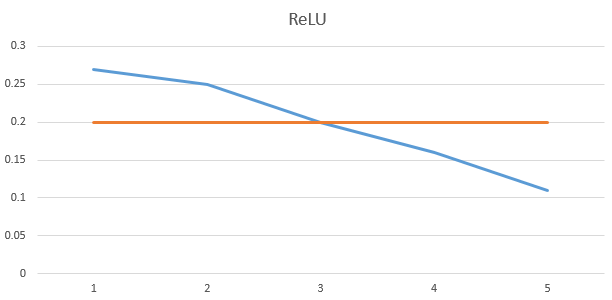


Рисунок 5.9 – Графік зниження вірогідності зі зростанням номера місця у гонці (вісь ординат – вірогідність зайняти місце, вісь абсцис – номер місця)

Це можна пояснити тим що, нейронна мережа навчаєтеся на даних про гру гравця, які були зібрані на основі декількох останніх пройдених гравцем рівней, але гравець вдосконалює навички гри, у тому числі, і на поточному рівні, який буде взятий до уваги при навчанні нейронних мереж ботів лише на наступному рівні. І, таким чином, гравець грає “трохи” краще, ніж навченні нейронні мережі суперників.

Також для порівняння результатів проведемо подібне тестування у іншій грі цього жанру. У якості гри для порівняння візьмемо гру Colin McRae Rally 2.0, яка була описана у розділі 1 (оскільки моделі ШІ суперників також являють собою нейронні мережі, але навчені заздалегідь). У грі можна налаштовувати кількість суперників, отже візьмемо таку ж саму кількість, як у розробленій у дипломній роботі грі, тобто 4. Маємо результати тестування, наведені у таблиці 5.8.

Таблиця 5.8 – Результати порівняння ігор AAIRace та Colin McRae Rally 2.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Місце | AAIRace | | Colin McRae Rally 2.0 | |
|  | C | P | C | P |
| 1 | 11 | 0.27 | 15 | 0.35 |
| 2 | 10 | 0.25 | 9 | 0.21 |
| 3 | 8 | 0.2 | 9 | 0.21 |
| 4 | 7 | 0.16 | 7 | 0.16 |
| 5 | 5 | 0.11 | 2 | 0.04 |
| D2 | 0.0149 | | 0.0724 | |

Для того, щоб порівняти результати, порахуємо різницю у процентах між показниками D = √D2,які отримані як результати тестування гри Colin McRae Rally 2.0, та як результат тестування найкращої конфігурації програмної системи AAIRace, розробленої у дипломі. Розрахуємо це за формулою:

тут – показник D2 для гри Colin McRae Rally 2.0, а – для гри AAIRace.

Маємо:

.

Отже при порівнянні мір адаптованості нейронних мереж до рівня гри суперників отримали що результат AAIRace кращий на 53%, ніж для гри Colin McRae Rally 2.0.

# 6 ОХОРОНА ПРАЦІ

Оскільки розроблена програмна система передбачає роботу із користувачем, слід розглянути питання охорони його праці.

Охорона праці – це регульована предметна область, для якої потрібно визначити види заходів та відповідні умови їх проведення, необхідні для успішної реалізації і завершення кваліфікаційної роботи магістра зі спеціальності інженерія програмного забезпечення.

Об’єктом дослідження було обрано робоче місце у навчальній лабораторії, що знаходиться у аудиторії за номером 908ф. Аудиторія знаходиться на дев’ятому поверсі будівлі Інституту комп'ютерних систем.

Робоче місце студента, де він виконує навчальні завдання, складається із стільця та стола, на якому розміщено монітор, системний блок, клавіатура та миша.

Вивчення найважливіших заходів з охорони праці та безпеки у надзвичайних ситуаціях для працівників галузі інженерія програмного забезпечення наведено у Додатку Б та висвітлює наступні питання:

1. Аналіз умов праці і вибір заходів і засобів захисту від небезпечних і шкідливих виробничих факторів.
2. Аналіз техногенних небезпек і вибір заходів і засобів забезпечення без- пеки у надзвичайних ситуаціях.
3. Складання рекомендацій щодо поліпшення умов праці.

Спираючись на проаналізовані дані, було розраховано захисне занулення та визначення категорії пожежонебезпечних приміщень.

# ВИСНОВКИ

В ході дипломної роботи проведено дослідження в області розробки комп’ютерних ігор жанру «Гонки» зі штучним інтелектом. Виконані наступні завдання:

* проаналізовано предметну область та знайдено процес для автоматизації;
* проаналізовано існуючі аналоги та подібні застосування;
* проаналізовано методи адаптації ігрового штучного інтелекту у іграх жанру «Гонки»;
* розроблено метод адаптації ШІ суперників до рівня гри гравця під час гри, що включає у якості моделі нейронні мережі суперників, вибір конфігурації яких базується на запропонованій мірі адаптованості гри суперників під рівень гри гравця D2, а також фільтрації та нормалізації даних, що являють собою зібрані дії гравця в залежності від станів гри;
* розроблено десктопну програму AAIRace, яка представляє собою гру жанру «Гонки», що використовує розроблений метод адаптації:
  + проаналізовано та обрано інструментальні засоби для створення десктопного застосування;
  + проаналізовані вимоги до десктопного застосування, тобто зафіксовані функціональні та нефункціональні вимоги;
  + представлена архітектура всієї системи і описані інтерфейси користувача програмної системи;
  + виконано проектування десктопної програми;
  + проведено тестування важливих для програмної системи функцій та підтверджено факт того, що система відповідає описаним бізнес-вимогам і вимогам користувача.
* проведено випробування методу адаптації та системи.

Таким чином, розроблена програмна система AAIRace, що представляє собою гру жанру «Гонки», у якій штучний інтелект суперників навчається на базі дій гравця.

Результат випробування демонструє, що програмна система адаптує рівень складності гри під гравця, що виражається у більш рівній вірогідності зайняти кожне із можливих у гонці місць. Для функції активації ReLU разом з ініціалізацією методом Каймига Хе показиний результат D2 найкращий і становить 0.0149 в порівнянні з іншими функціями активаціями, де найближче до найкращого значення – 0.024 для функції активації SoftPlus.

Також у роботі було проведено порівняння гри AAIRace з розробленим методом адаптації із іншою грою жанру «Гонки» – Colin McRae Rally 2.0, ШІ суперників котрої також базується на нейронних мережах, але навчених заздалегіть. Міра адаптованості суперників до рівня гри гравця у розробленій програмній системі краще на 53%, ніж для гри Colin McRae Rally 2.0.

Розроблена система працює під операційною системою Windows 10, розгортається швидко та легко.

У майбутньому планується розширити функціонал налаштувань для користувача та розробити більш різноманітні рівні. А також допрацювати метод адаптації штучного інтелекту для того, щоб компенсувати покращення навичок гри гравця під час гри.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Игровой искусственный интеллект [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://ru.wikipedia.org/wiki/Игровой\_искусственный \_интеллект – Загол. з екрану.
2. Rouse, Richard. Game Design: Theory & Practice : [англ.]. – 2. – Los Rios Boulevard, Plano, Texas, USA : Wordware Publishing, 2004. – 698 с. – ISBN 1-55622-912-7.
3. The Quake III Arena bot is an artificial player for the computer game Quake III Arena [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://quake.fandom.com/wiki/Quake\_3\_Bots – Загол. з екрану.
4. Моб (компьютерные игры) [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://ru.wikipedia.org/Моб\_(компьютерные игры) – Загол. з екрану.
5. История развития ИИ в играх: эволюция, алгоритмы [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://stopgame.ru/blogs/topic/93248 – Загол. з екрану.
6. История компьютерных гонок [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://most.tv/news/75496.html – Загол. з екрану.
7. Crossout [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://ru.wikipedia.org/wiki/Crossout – Загол. з екрану.
8. Боты в crossout [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://crossout.fandom.com/wiki/Bot\_controlled\_vehicles – Загол. з екрану.
9. FlatOut [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://ru.wikipedia.org/wiki/FlatOut – Загол. з екрану.
10. Водители в FlatOut [Електронний ресурс]– Режим доступу до ресурсу: www / URL: https://flatout.fandom.com/wiki/Drivers – Загол. з екрану.
11. Colin McRae Rally 2.0 [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://ru.wikipedia.org/wiki/Colin\_McRae\_Rally\_2.0 – Загол. з екрану.
12. Philip Hingston, Believable Bots: Can Computers Play Like People? – 294 c.
13. Брауде Э.Дж. Технология разработки программного обеспечения. Питер, 2004. – 656 с.
14. Chen, Lianping Characterizing Architecturally Significant Requirements [Текст] / Lianping Chen, Muhammad Ali Babar, Bashar Nuseibeh – 2013.
15. Android Architecture Patterns Part 1: Model-View-Controller [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/upday-devs/android-architecture-patterns-part-1-model-view-controller-3baecef5f2b6 – Загол. з екрану.
16. Ігровий штучний інтелект [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://uk.wikipedia.org/wiki/Ігровий\_штучний\_інтелект – Загол. з екрану.
17. Штучні нейронні мережі [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://uk.wikipedia.org/wiki/Штучна\_нейронна\_мережа – Загол. з екрану.
18. В. М. Коцовський, Нейронні системи, конспект лекцій, 2013 – 25 с.
19. Правило навчання ШНМ [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://uk.wikipedia.org/wiki/Правило\_навчання\_ШНМ – Загол. з екрану.
20. Бодянский Е.В., Руденко О.Г. Искусственные нейронные сети: архитектуры, обучение, применения / Харьков: Телетех, 2004. - 369 с.
21. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение, погружение в мир нейронных сетей / 2018 – 145 с.
22. Oleksanrd A. Blazhko. Features of using the canvas-oriented approach to game design / Applied Aspects of Information Technology, 2018 / Vol.1 No.1:66–77;
23. Препроцесінг даних для навчання нейронних мереж [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://docs.microsoft.com/ru-ru/azure/machine-learning/team-data-science-process/prepare-data – Загол. з екрану.
24. Підбір кількості нейронів у проміжковому слої ШНМ [Електронний ресурс] – Режим доступу до ресурсу: www / URL: https://qastack.ru/stats/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw – Загол. з екрану.