МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

**КУЗЬМИЧ Богдан Адамович**

# Програмний модуль прогнозування перешкод у лабіринті за допомогою методів машинного навчання/ Software Module for Predicting Obstacles in a Maze Using Machine Learning Methods

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41

Б. А. Кузьмич

_____

Науковий керівник:
к.т.н., доцент П. Є. Биковий

_____

Кваліфікаційну роботу допущено до захисту
«___»_____ 2025 р.

В.о. завідувача кафедри
_____ Н.М. Васильків

**Тернопіль – 2025**

**Факультет комп'ютерних інформаційних технологій**
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.М. Васильків
«____» _____2024 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**
**КУЗЬМИЧ Богдан Адамович**
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмний модуль прогнозування перешкод у лабіринті за допомогою методів машинного навчання/ Software Module for Predicting Obstacles in a Maze Using Machine Learning Methods
керівник роботи П. Є. Биковий к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 20 грудня 2024 р. № 938.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:

– здійснити аналіз предметної області, що охоплює задачі орієнтації та навігації в лабіринтах, специфіку збору координатних даних у 2D-середовищі та визначення типових перешкод;

– проаналізувати існуючі підходи до прогнозування траєкторій руху об'єктів за допомогою методів машинного навчання;

– розробити програмний модуль збору даних у середовищі Godot, який фіксує координати об'єкта, зіткнення з перешкодами та положення фінішної точки;

– створити серверну частину модуля прогнозування, яка реалізує обчислення за допомогою моделей машинного навчання в середовищі Python;

– запровадити механізм корекції прогнозу, який зміщує результати у напрямку фінішу шляхом обчислення середнього значення між прогнозованою та цільовою координатами;

– реалізувати кілька режимів руху об'єкта;

– провести серію експериментів з використанням різних моделей прогнозування, виміряти середній час досягнення фінішу;

– оцінити ефективність розробленої системи та визначити переваги й потенціал подальшого використання в задачах навігації та моделювання інтелектуальної поведінки.

5. Перелік графічного матеріалу в роботі:

– структурна схема взаємодії між клієнтською та серверною частинами

системи.

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів кваліфікаційної роботи | Примітка |
|---|---|---|---|
| 1 | Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи | до 01.01. 2025 р. | |
| 2 | Написання 1 розділу кваліфікаційної роботи | до 01.02. 2025 р. | |
| 3 | Написання 2 розділу кваліфікаційної роботи | до 01.04.2025 р. | |
| 4 | Написання 3 розділу кваліфікаційної роботи | до 01.05. 2025 р. | |
| 5 | Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником | до 15.05.2025 р. | |
| 6 | Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів | до 25.05.2025 р. | |
| 7 | Перевірка кваліфікаційної роботи на оригінальність тексту | до 30.05.2025 р. | |
| 8 | Оформлення кваліфікаційної роботи та отримання допуску до захисту | до 10.06.2025 р. | |
| 9 | Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії | до 10.06. 2025 р. | |

Студент_____ Б.А. Кузьмич
( підпис ) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ П. Є. Биковий
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль прогнозування перешкод у лабіринті за допомогою методів машинного навчання» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом у 48 сторінок і містить 7 ілюстрацій, 1 таблицю, 5 додатки та 25 використаних джерел.

Метою кваліфікаційної роботи є розробка програмного модуля для прогнозування можливих перешкод у лабіринті з використанням різних моделей машинного навчання, який забезпечує динамічну адаптацію до змін у середовищі та оптимізацію шляху руху об'єкта.

Об'єктом дослідження є процес навігації у двовимірному середовищі з перешкодами.

Предметом дослідження є програмний модуль прогнозування координат руху об'єкта в лабіринті із застосуванням моделей машинного навчання та інтеграції середовищ Godot і Python через WebSocket.

У межах роботи реалізовано модуль збору даних у середовищі Godot, побудовано серверну частину для обробки координат за допомогою п'яти моделей машинного навчання (Linear, Polynomial, Ridge, SVR, Random Forest), здійснено корекцію прогнозованих координат з урахуванням цільової точки (фінішу), а також розроблено кілька режимів руху, зокрема автономний інтелектуальний режим із запам'ятовуванням перешкод.

Ключові слова: ПРОГНОЗУВАННЯ, ЛАБІРИНТ, МАШИННЕ НАВЧАННЯ, GODOT, PYTHON, WEB-SOCKET, ШТУЧНИЙ ІНТЕЛЕКТ, НАВІГАЦІЯ.

# ANNOTATION

The bachelor's qualification paper titled "Software Module for Obstacle Prediction in a Maze Using Machine Learning Methods", submitted for the degree of Bachelor in specialty 122 "Computer Science" under the "Computer Science" educational program, consists of 48 pages, includes 7 figures, 1 table, 5 appendices, and references 25 sources.

The aim of this qualification paper is to develop a software module for predicting potential obstacles in a maze using various machine learning models, ensuring dynamic adaptation to environmental changes and optimizing the object’s path.

The object of the study is the process of navigation in a two-dimensional environment with obstacles.

The subject of the study is a software module for predicting object movement coordinates in a maze using machine learning models, with integration between the Godot and Python environments via WebSocket communication.

Within the scope of the work, a data collection module in Godot was implemented, a server-side component was built to process coordinates using five machine learning models (Linear, Polynomial, Ridge, SVR, and Random Forest), and correction of predicted coordinates was carried out based on the target position (finish point). Multiple movement modes were developed, including an autonomous AI-driven mode with obstacle memory.

Keywords: PREDICTION, MAZE, MACHINE LEARNING, GODOT, PYTHON, WEB-SOCKET, ARTIFICIAL INTELLIGENCE, NAVIGATION.

ЗМІСТ

# ВСТУП

У сучасному світі автоматизовані модулі та робототехнічні рішення дедалі частіше використовуються для вирішення складних завдань. Одним із найцікавіших викликів є створення роботів, здатних самостійно орієнтуватися у складних просторових умовах, зокрема в лабіринтах. Це завдання особливо актуальне для рятувальних операцій, логістики, індустріальних процесів та інших сфер, де важливо забезпечити точну і безпечну навігацію в умовах обмеженого простору.

Сучасні досягнення в галузі машинного навчання відкривають нові можливості для вирішення цього завдання. Методи машинного навчання дозволяють прогнозувати поведінку модуля у змінних умовах, передбачати можливі перешкоди та ухвалювати оптимальні рішення. Завдяки цьому підходу можна створювати гнучкі програмні модулі, які адаптуються до різноманітних середовищ і забезпечують високу ефективність навігації.

Актуальність теми обумовлена потребою у підвищенні точності, швидкості та надійності автоматизованих модулів навігації, що можуть використовуватись у багатьох сферах життя. Використання інтелектуальних алгоритмів дозволяє досягати нових висот у розвитку робототехніки та автоматизації.

Метою роботи є розробка програмного модуля, що здатен прогнозувати перешкоди в лабіринті з використанням методів машинного навчання. Для досягнення мети передбачено вирішення таких завдань:

1. Здійснити аналіз предметної області, що охоплює задачі орієнтації та навігації в лабіринтах, специфіку збору координатних даних у 2D-середовищі та визначення типових перешкод.

2. Проаналізувати існуючі підходи до прогнозування траєкторій руху об’єктів за допомогою методів машинного навчання.

3. Розробити програмний модуль збору даних у середовищі Godot, який фіксує координати об’єкта, зіткнення з перешкодами та положення фінішної точки.

4. Створити серверну частину модуля прогнозування, яка реалізує обчислення за допомогою моделей машинного навчання в середовищі Python.

5. Запровадити механізм корекції прогнозу, який зміщує результати у напрямку фінішу шляхом обчислення середнього значення між прогнозованою та цільовою координатами.

6. Реалізувати кілька режимів руху об’єкта.

7. Провести серію експериментів з використанням різних моделей прогнозування, виміряти середній час досягнення фінішу.

8. Оцінити ефективність розробленої системи та визначити переваги й потенціал подальшого використання в задачах навігації та моделювання інтелектуальної поведінки.

Об’єктом дослідження є процес навігації у лабіринті з урахуванням можливих перешкод. Предметом дослідження є методи машинного навчання, що застосовуються для прогнозування та уникнення перешкод у процесі навігації.

Предметом дослідження є програмний модуль прогнозування координат руху об’єкта в лабіринті із застосуванням моделей машинного навчання та інтеграції середовищ Godot і Python через WebSocket.

Для досягнення поставленої мети в роботі використано такі методи дослідження:

- теоретичний аналіз літературних джерел з тематики навігації, алгоритмів штучного інтелекту та машинного навчання;
- моделювання і проектування архітектури програмного забезпечення;
- експериментальні дослідження з використанням симуляційного середовища Godot і Python-серверу;
- методи порівняльного аналізу ефективності алгоритмів на основі метрик точності та часу проходження;
- візуалізація результатів та аналіз похибок моделі.

Структура роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі здійснено аналіз предметної області, охарактеризовано сучасні підходи до прогнозування руху та сформульовано постановку задачі. У другому розділі розглянуто архітектуру модуля, інструментальні засоби та технології для реалізації системи, описано принципи функціонування клієнтської та серверної частин. У третьому розділі

представлено реалізацію програмного модуля, результати експериментального дослідження, порівняльну оцінку ефективності моделей та візуалізацію прогнозів.

Результати цієї роботи сприятимуть удосконаленню модуля автономної навігації, створенню більш адаптивних робототехнічних рішень і підвищенню рівня безпеки та ефективності у відповідних сферах застосування.

Результати дослідження опубліковано в матеріалах науково-практичної конференції «Інтелектуальні інформаційні технології в прикладних дослідженнях» (IITAR – 2025), м. Тернопіль, 27–29 травня 2025 р. (додаток Д).

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Основні проблеми предметної області

Прогнозування перешкод у лабіринтах – це складне і багатогранне завдання, яке стоїть на перетині сучасних технологій і реальних потреб. Попри значний прогрес у галузі машинного навчання, багато аспектів цієї задачі залишаються відкритими, ставлячи перед дослідниками низку викликів.

Перш за все, навігація в лабіринтах передбачає роботу з середовищем, яке постійно змінюється. Лабіринти можуть мати як статичні перешкоди, так і динамічні елементи, такі як рухомі об’єкти чи змінні умови освітлення. Це значно ускладнює моделювання середовища.

Ще однією важливою проблемою є дані. Уявіть собі, що для навчання алгоритму потрібні сотні, якщо не тисячі сценаріїв. Ці дані мають бути різноманітними, репрезентативними та близькими до реальних умов. Потрібно навчити модуль передбачати рух у реальному світі, якщо бачить лише симуляції

Швидкість роботи алгоритмів – це ще один важливий аспект. Сучасні роботи часто працюють у режимі реального часу, коли рішення потрібно ухвалювати за долі секунди. Однак баланс між швидкістю і точністю часто стає справжнім викликом. Як забезпечити швидку реакцію, не втрачаючи при цьому надійності?

Не можна також ігнорувати обмежені ресурси, з якими доводиться працювати. Роботи зазвичай оснащені компактними обчислювальними модулями, які не можуть обробляти великі обсяги даних за допомогою складних моделей. Як створити алгоритми, які працюють швидко і точно, але при цьому не потребують потужного обладнання?

Ще однією цікавою проблемою є адаптивність. Лабіринт, у якому працює робот, може суттєво відрізнятися від того, на чому він навчався. Придумати що робити, якщо модель стикнеться з новим типом перешкод, який вона раніше не бачила. Та як навчити її не просто працювати, а й навчатися на ходу

Нарешті, не можна забувати про інтеграцію даних. Лідари, камери, сенсори – кожен із цих інструментів надає різні типи інформації, які потрібно об’єднати в єдину систему. Шум, неточності та обмежена роздільна здатність – усе це

ускладнює задачу.

Таким чином, предметна область прогнозування перешкод у лабіринтах є не лише технічно складною, але й глибоко інтригуючою. Кожна проблема тут – це виклик, а кожне рішення – крок до створення автономних модулів, здатних працювати в найскладніших умовах.

## 1.2 Огляд існуючих рішень

Прогнозування шляху в лабіринті за допомогою запропонованого методу поєднує алгоритми Дейкстри та генетичні алгоритми, створюючи адаптивну систему для оптимального вибору маршрутів. Основна ідея полягає у динамічному аналізі умов середовища та коригуванні траєкторії в режимі реального часу [1].

Дейкстра дозволяє знаходити найкоротший шлях у мережі, забезпечуючи гарантовано оптимальний маршрут при фіксованих умовах. Однак у реальному середовищі, особливо в мережевому трафіку, параметри змінюються. Тут на допомогу приходять генетичні алгоритми – вони використовують принципи природного відбору для пошуку найкращих варіантів маршруту, оптимізуючи його на основі історичних даних та поточного стану системи.

Цей підхід дозволяє не тільки швидко адаптуватися до змін, але й прогнозувати майбутні маршрути на основі закономірностей у русі трафіку. Завдяки поєднанню аналітики й самонавчання алгоритм стає ефективним інструментом для керування потоками даних, знижуючи затримки, уникаючи перевантажень і забезпечуючи плавну роботу системи.

Однією з переваг є адаптивність. Алгоритм здатен змінювати маршрути в режимі реального часу, враховуючи змінні умови середовища, такі як пробки, погодні умови чи аварії. Це дозволяє модуль навігації більш оперативно реагувати на непередбачувані ситуації.

Ще одним важливим аспектом є оптимізація ресурсів. Використовуючи історичні дані, модуль може прогнозувати найбільш ефективні маршрути, зменшуючи затримки та перевантаження доріг. Таким чином, забезпечується більш раціональне використання транспортних ресурсів.

Комбінування алгоритму Дейкстри з генетичними методами забезпечує гнучкість. Завдяки цьому підходу можливо знаходити не тільки найкоротші, але й найбільш ефективні маршрути, які враховують реальні умови пересування.

Крім того, застосування машинного навчання дозволяє системі постійно вдосконалювати свої прогнози. Алгоритм самонавчається, підлаштовуючись під змінні умови та покращуючи точність вибору маршрутів.

Останньою, але не менш важливою перевагою є швидкість розрахунків. Комбінований підхід дає змогу швидко знаходити рішення навіть у складних лабіринтах із великою кількістю можливих варіантів, що є надзвичайно важливим для реального застосування.

Попри значні переваги, комбіновані алгоритми мають і певні недоліки. Одним із головних викликів є обчислювальна складність. Використання генетичних алгоритмів може потребувати значних обчислювальних ресурсів, особливо при великій кількості можливих маршрутів, що може ускладнити їх застосування у реальному часі.

Ще однією проблемою є можливість локальних оптимумів. Генетичні алгоритми не завжди гарантують знаходження глобально найкращого рішення, іноді зупиняючись на субоптимальних варіантах. Це може вплинути на якість маршрутизації та ефективність роботи системи.

Також слід враховувати час навчання. Алгоритм потребує достатньої кількості даних і часу для самонавчання, перш ніж почне давати найкращі результати. Це може стати перешкодою для швидкого впровадження системи в експлуатацію.

Останнім викликом є складність реалізації. Комбінація двох алгоритмів потребує точного налаштування параметрів, що ускладнює процес впровадження у реальні системи та вимагає високого рівня експертизи.

Сучасні підходи до навігації та прогнозування перешкод у лабіринтах базуються на різних алгоритмах, які можна розділити на три основні категорії: класичні алгоритми пошуку шляхів, методи комп’ютерного зору та нейромережеві підходи [2].

Одним із найбільш поширених методів пошуку оптимального маршруту є

алгоритми A* та Dijkstra. Вони використовуються для побудови найкоротшого шляху між початковою та кінцевою точками, оцінюючи вагу кожного вузла в графі [3]. Проте такі алгоритми мають низку обмежень, зокрема:

- вони не враховують динамічні зміни в середовищі;
- кожен раз потребують повного перерахунку маршруту після зміни конфігурації лабіринту;
- працюють менш ефективно у випадках, коли інформація про середовище є неповною або надходить із затримкою.

Використання комп'ютерного зору є перспективним підходом для автоматичного аналізу середовища в реальному часі. Для цього застосовуються такі технології, як:

- класичні алгоритми обробки зображень (Canny, Hough Transform) – можуть виявляти контури та визначати перешкоди, але погано працюють у складних умовах освітлення [3];
- глибинні камери (Microsoft Kinect, Intel RealSense) – дозволяють отримувати тривимірну карту середовища та аналізувати глибину об'єктів, що допомагає точніше визначати можливі перешкоди [4 ];
- згорткові нейронні мережі (CNN) – ефективно розпізнають структуру лабіринту на основі зображень, прогнозуючи можливі проходи [5 ].

Основний недолік таких рішень полягає в необхідності значних обчислювальних ресурсів, а також у складності адаптації моделей до нових умов без попереднього донавчання.

Найбільш перспективним напрямом є використання методів машинного навчання для прогнозування перешкод у лабіринтах. Серед найбільш ефективних підходів можна виділити:

- рекурентні нейронні мережі (RNN, LSTM, GRU) – використовуються для аналізу послідовності координат та прогнозування майбутніх позицій перешкод на основі попередніх даних [6];
- глибокі згорткові мережі (CNN + RNN) – поєднують просторовий аналіз середовища з часовими залежностями, що дозволяє прогнозувати зміни у лабіринті [7];

- підкріплене навчання (Deep Q-Network, PPO) – навчає агентів самостійно визначати оптимальні маршрути, використовуючи експериментальне дослідження середовища [8].

Такі підходи демонструють високу адаптивність і здатність прогнозувати перешкоди навіть у динамічних середовищах. Проте вони потребують великої кількості даних для навчання та достатньої обчислювальної потужності.

Лабіринт як середовище для навігації є динамічним та непередбачуваним. Перешкоди можуть змінюватися, а маршрут, що був оптимальним на початку, може стати недоступним у процесі руху об’єкта. Класичні алгоритми пошуку шляхів, такі як A* або Dijkstra, ефективні для статичних середовищ, але їхня адаптивність у реальному часі залишається обмеженою. Вони потребують значних обчислювальних ресурсів для повторного розрахунку маршруту після кожної зміни конфігурації середовища [9].

У разі використання мобільних платформ, які передають координати доступних проходів та стін до обчислювального модуля, важливо не лише оперативно обробляти отриману інформацію, а й прогнозувати можливі перешкоди. Це дозволяє зменшити затримки у прийнятті рішень та підвищити ефективність навігації.

Одним із найперспективніших підходів є використання глибоких нейронних мереж для аналізу отриманих координат і прогнозування вільних проходів. Використання згорткових нейронних мереж (CNN) у поєднанні з рекурентними моделями (RNN, LSTM) дозволяє модулю визначати закономірності в структурі лабіринту та прогнозувати зони, які з високою ймовірністю залишатимуться вільними.

Іншим ефективним методом є навчання з підкріпленням (Reinforcement Learning, RL). У цій концепції агент (рухомий об'єкт у лабіринті) навчається приймати рішення, базуючись на винагородах за успішну навігацію та штрафах за зіткнення з перешкодами. Глибоке навчання з підкріпленням (Deep Reinforcement Learning, DRL) дозволяє навчати модель знаходити оптимальні маршрути, використовуючи реальні дані про середовище.

Однією з популярних архітектур є Deep Q-Network (DQN), що використовує нейромережу для вибору найкращих дій у кожному стані. У поєднанні з алгоритмами, такими як Proximal Policy Optimization (PPO) або A3C (Asynchronous Actor-Critic), можна значно підвищити ефективність навігації та адаптації до змін у середовищі [10].

## 1.3 Постановка задачі

У сучасних симуляційних та ігрових модулів важливим аспектом є прогнозування руху об’єктів у складних динамічних середовищах. Однією з таких задач є прогнозування можливих траєкторій руху об'єкта в лабіринті з урахуванням змінної структури оточення та потенційних перешкод. Дана проблема є актуальною не лише в контексті комп’ютерних ігор, а й у сфері робототехніки, автономних систем навігації та моделювання поведінки агентів.

Основна мета дослідження – розробка програмного модуля, що здатен прогнозувати перешкоди в лабіринті з використанням методів машинного навчання. Для досягнення цієї мети необхідно вирішити такі завдання:

1. Здійснити аналіз предметної області, що охоплює задачі орієнтації та навігації в лабіринтах, специфіку збору координатних даних у 2D-середовищі та визначення типових перешкод.
2. Проаналізувати існуючі підходи до прогнозування траєкторій руху об’єктів за допомогою методів машинного навчання.
3. Розробити програмний модуль збору даних у середовищі Godot, який фіксує координати об’єкта, зіткнення з перешкодами та положення фінішної точки.
4. Створити серверну частину модуля прогнозування, яка реалізує обчислення за допомогою моделей машинного навчання в середовищі Python.
5. Запровадити механізм корекції прогнозу, який зміщує результати у напрямку фінішу шляхом обчислення середнього значення між прогнозованою та цільовою координатами.
6. Реалізувати кілька режимів руху об’єкта.

7. Провести серію експериментів з використанням різних моделей прогнозування, виміряти середній час досягнення фінішу.

8. Оцінити ефективність розробленої системи та визначити переваги й потенціал подальшого використання в задачах навігації та моделювання інтелектуальної поведінки.

Очікуваним результатом є створення стабільної, ефективної та адаптивної системи прогнозування руху, яка зможе аналізувати оточення, визначати потенційні маршрути та навчатися на основі отриманого досвіду, підвищуючи точність передбачень у складних умовах лабіринту.

# 2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ДЛЯ АНАЛІЗУ ДАНИХ

## 2.1 Апаратне забезпечення

Розробка модуля навігації для лабіринту в середовищі Godot із використанням аналізу координат у Python дозволяє вирішувати проблеми орієнтації в просторі, такі як неточне визначення прохідних шляхів та неефективне планування маршруту. Це рішення інтегрує механізм передачі координат між Godot і Python, використовуючи алгоритми обробки даних для аналізу доступних напрямків руху в реальному часі. Завдяки автоматичному визначенню прохідних і заблокованих зон Python прогнозує потенційні безпечні маршрути, а Godot перевіряє ці передбачення, підтверджуючи або спростовуючи координати. Такий підхід мінімізує помилки навігації, оптимізує траєкторію руху об'єкта та підвищує ефективність проходження лабіринту. Крім того, безперервний обмін даними між Godot і Python забезпечує адаптивне коригування маршруту залежно від змін у середовищі, що робить навігацію динамічною та надійною. У своїй суті ця модуль трансформує традиційний підхід до навігації лабіринтом у розумний, автоматизований та ефективний процес.

Модуль прогнозування руху в лабіринті побудована на взаємодії двох основних компонентів: ігрового середовища в Godot та серверної обробки даних у Python. Godot відповідає за візуалізацію лабіринту та реєстрацію поточних координат об'єкта, тоді як Python-скрипт аналізує отримані дані, прогнозує можливі шляхи руху та надсилає їх назад у Godot для перевірки.

Обмін даними здійснюється через WebSocket-з'єднання, що забезпечує швидку передачу інформації в реальному часі. Об'єкт у лабіринті постійно надсилає координати секторів, де можна рухатися, а де знаходяться стіни. Python-сервер, використовуючи методи регресій, аналізує отримані дані, виявляє закономірності та прогнозує потенційно доступні зони.

Для ефективного навчання та прогнозування Python використовує алгоритми машинного навчання, а також методи мінімізації похибок, такі як оцінка середньоквадратичної помилки (MSE). Це дозволяє збільшити точність передбачень і зменшити кількість помилкових напрямків.

Елементами апаратної архітектури є:

- ігровий рушій Godot, що виконує роль клієнта, відповідального за візуалізацію лабіринту та отримання результатів аналізу від сервера;
- Python-сервер, що аналізує отримані координати, здійснює прогнозування та відправляє результати назад у Godot;
- WebSocket-з'єднання, яке забезпечує швидкий обмін даними між клієнтом і сервером у режимі реального часу.

Для підвищення точності аналізу Python-сервер накопичує історію отриманих даних та оновлює модель прогнозування в процесі роботи. Якщо модуль виявляє помилки у прогнозах (наприклад, передбачуваний сектор виявляється стіною), ці дані використовуються для корекції майбутніх розрахунків.

В таблиці 2.1 представлено характеристику апаратного забезпечення

Таблиця 2.1 - Характеристики апаратного забезпечення

| Категорія | Деталі |
|---|---|
| Процесор (CPU) | Intel Core i5-10300h CPU 2.50Ghz |
| Відеокарта (GPU) | NVIDIA GeForce GTX 1650 ; 4GB |
| Оперативна пам'ять (RAM) | 16 GB |
| Операційна система | Windows 11 |
| Бібліотеки/Фреймворки | NumPy, Scikit-learn, Websockets |
| Мова програмування | Python 3.12 |

Для забезпечення стабільності модуля реалізовано механізми відмовостійкості. У разі розриву з'єднання сервер автоматично відновлює останні збережені координати та продовжує обробку з урахуванням попередніх даних. Крім того, у модіулі інтегровано алгоритми виявлення помилок, які аналізують розбіжності між прогнозованими та реальними координатами. Якщо похибка перевищує допустиме значення, сервер коригує модель і адаптується до змін у структурі лабіринту.

Архітектура також передбачає можливість масштабування: у разі збільшення

складності лабіринту або розширення функціоналу можна підключати додаткові модулі обробки, покращуючи продуктивність модуля.

## 2.2 Модульна структура

Розроблення модуля прогнозування руху об'єкта у лабіринті складається з двох основних компонентів: ігрового середовища, реалізованого в Godot Engine, та серверної частини, розробленої мовою програмування Python із використанням бібліотек для машинного навчання. Компоненти взаємодіють між собою за допомогою протоколу WebSocket, що забезпечує обмін даними у реальному часі.

Основні елементи архітектури:

1. Godot Engine:
   - здійснює візуалізацію лабіринту та об'єкта;
   - реєструє координати прохідних секторів та стін;
   - передає отримані дані на сервер для аналізу;
   - отримує прогнозовані координати та спавнить об'єкти для візуального відображення прогнозованого шляху.
2. Python-сервер:
   - приймає координати від Godot;
   - зберігає історію отриманих даних;
   - навчає моделі машинного навчання для прогнозування подальших можливих напрямків руху об'єкта;
   - повертає прогнозовані координати назад у Godot для візуалізації та подальшого аналізу.

Комунікація між сервером і клієнтом здійснюється у форматі JSON-повідомлень. Для підвищення стійкості модуля передбачено механізми обробки помилок та автоматичного перепідключення у разі розриву з’єднання [11].

Таким чином, модуль утворює замкнене інтерактивне середовище, де дані про середовище постійно оновлюються, обробляються і використовуються для покращення точності навігації у лабіринті.

Розроблена модуль складається з двох основних частин: клієнтської,

реалізованої в середовищі Godot за допомогою мови GDScript, та серверної — побудованої на Python. Взаємодія між цими компонентами відбувається через WebSocket-з'єднання, що забезпечує обмін координатами та результатами прогнозування в режимі реального часу.

На стороні клієнта, в Godot, реалізовано кілька скриптів, кожен з яких виконує власну функцію. Скрипт G.gd відповідає за збір координат секторів у віртуальному лабіринті, поділених на прохідні (чисті) та непрохідні (стіни). Ці координати зберігаються в окремих структурах, після чого формуються у формат, придатний для надсилання серверу. Отримані відповіді обробляються та розподіляються по відповідних моделях, щоб забезпечити їх подальше візуальне представлення.

Інший скрипт – server.gd – реалізує механізм підключення до Python-сервера та регулярне надсилання поточних координат. Він також відповідає за обробку отриманих від сервера прогнозів, підтримуючи постійне оновлення стану моделі у віртуальному середовищі.

Для відображення прогнозованих координат використовується модуль Spawner.gd. Він створює віртуальні об’єкти на основі прогнозів, що надійшли з сервера, та динамічно видаляє старі елементи, забезпечуючи актуальність візуалізації. Спавн кожного об’єкта відбувається із затримкою, що дозволяє плавно демонструвати результати роботи моделей.

Скрипт spawn.gd реалізує поведінку кожного окремого об’єкта, зокрема ініціалізацію, реєстрацію можливих зіткнень зі стінами та фіксацію подібних подій. Це дозволяє не лише візуалізувати прогноз, але й перевірити його відповідність реальному середовищу.

На серверній стороні працює Python-програма gdSocket.py, яка виконує функції аналітичного обчислення. Вона приймає координати, зберігає історію даних та проводить машинне навчання на їх основі [12]. Для прогнозування використовуються кілька алгоритмів: лінійна регресія, поліноміальна регресія другого ступеня, Ridge-регресія, Support Vector Regression (SVR) та Random Forest. Завдяки такому підходу можна порівнювати ефективність моделей між собою та спостерігати, як змінюється точність у залежності від обставин.

Отримані прогнози координат передаються назад у середовище Godot, де відображаються у вигляді віртуальних точок. Таким чином, вся модуль працює в єдиному циклі, постійно оновлюючи дані, прогнозуючи майбутні положення об’єктів та візуалізуючи ці результати в реальному часі.

## 2.3 Розробка моделі машинного навчання для прогнозування руху в лабіринті

Інтеграція модуля прогнозування руху з існуючими модулями управління лабіринтом є важливою для ефективної взаємодії в реальному часі. Це дозволяє модулю прогнозування не лише адаптуватися до змін у лабіринті, а й автоматично синхронізувати свої обчислення з поточним станом середовища. Така інтеграція сприяє оптимальному прийняттю рішень, спрощенню управління навігацією та підвищенню точності передбачень.

Користувачі, такі як оператори симуляцій та розробники алгоритмів, отримують доступ до інтерфейсу візуалізації прогнозованих маршрутів, відстеження помилок та перегляду детальних звітів про рух об'єкта в середовищі Godot.

Ефективність модуля прогнозування напрямків руху в лабіринті залежить від якості та повноти отриманих даних. Процес продемонстровано в додатку А.

У розробленій платформі акцент зроблено на зборі координат вільних секторів та стін, які визначаються об'єктом, що рухається лабіринтом у середовищі Godot. Ці дані передаються в Python, де вони обробляються алгоритмом машинного навчання для прогнозування подальших можливих траєкторій руху [13].

Процес збору даних розпочався з отримання координат секторів лабіринту, які надсилає Godot. Об'єкт, рухаючись по лабіринту, зчитує інформацію про вільні шляхи та перешкоди. Ці координати передаються у WebSocket-сервер, розгорнутий у Python, де вони обробляються для створення навчального набору даних.

Дані включають:

- координати секторів без перешкод (clearSectorX, clearSectorY) – це позиції, де об'єкт може продовжити рух;

- координати стін (wallSectorX, wallSectorY) – це позиції, які є перешкодами для руху об'єкта.

Щоб забезпечити якість та точність прогнозування, дані збиралися в різних сценаріях – з різною швидкістю руху об'єкта, з різними початковими позиціями та маршрутами. Це дозволило навчити модуль узагальнювати можливі варіанти проходження лабіринту.

Одержані координати вільних секторів та перешкод обробляються в Python. Для цього використовується метод напів контрольованого навчання, де спочатку невелика кількість даних обробляється вручну, а потім алгоритм автоматично поширює маркування на решту набору.

Процес включає такі етапи:

1) збереження отриманих координат у структуру даних. ані зберігаються у форматі JSON та використовуються для створення вхідних масивів для алгоритму машинного навчання;

2) обмеження вибірки. Для Уникнення перенасичення даних вибирається останні 15 записів про вільні сектори та стіни;

3) застосування алгоритму регресії. Вхідні координати використовуються як ознаки, а вихідні координати (можливі напрямки руху) – як прогнозовані значення. Модель навчається визначати, в якому напрямку найбільш ймовірно можна рухатися;

4) передача прогнозованих координат у Godot. Результати обчислень надсилаються клієнту для подальшого аналізу. У Godot перевіряється точність прогнозу та приймається рішення про коригування або підтвердження напряму руху.

Модуль працює у режимі реального часу за допомогою WebSocket-з'єднання:

- Godot отримує дані про поточне положення об'єкта та його оточення, відправляє їх у Python-сервер;

- Python-сервер аналізує отримані координати, будує прогноз про можливий рух об'єкта та надсилає прогнозовані координати назад у Godot;

- Godot перевіряє отримані прогнозовані координати та підтверджує або спростовує їх, доповнюючи навчальний набір новими даними.

Щоб покращити якість прогнозування, перед навчанням моделі дані піддаються попередній обробці:

- нормалізація координат – перетворення значень у стандартизований діапазон, що дозволяє уникнути надмірного впливу великих чисел на результат прогнозування;
- фільтрація шуму – видалення випадкових аномальних значень, які можуть виникати через помилки зчитування;
- стратифікована вибірка – рівномірне представлення всіх можливих секторів у вибірці для навчання.

Завдяки використанню цих методів, модель машинного навчання здатна прогнозувати рух об'єкта в лабіринті з високою точністю, забезпечуючи ефективну взаємодію між Python та Godot. Це дозволяє створювати гнучкі алгоритми навігації, які можуть адаптуватися до різних конфігурацій лабіринтів і враховувати динамічні зміни у середовищі [14].

Основою запропонованої модуля навігації в лабіринті є модель машинного навчання, яка дозволяє ефективно прогнозувати можливі шляхи проходження, ґрунтуючись на отриманих даних про доступні зони та перешкоди. Процес розробки цієї моделі включає кілька етапів: збір даних, вибір оптимальних алгоритмів, навчання та валідація моделей, а також їх подальша оптимізація.

Інтеграція машинного навчання в алгоритм навігації значно покращила точність визначення можливих маршрутів. Для обробки отриманих даних використовуються різні методи, включаючи лінійну та множинну регресію, а також більш складні алгоритми прогнозування. Завданнями машинного навчання у цьому процесі є виявлення закономірностей у русі об'єкта, визначення областей, де відсутні стіни, та уточнення прогнозів на основі нових даних, що надходять у режимі реального часу.

Для навчання моделей використовується набір даних, отриманий із симуляції руху об'єкта в середовищі Godot. Кожен отриманий зразок містить координати чистих секторів (зон, де можливий рух) та секторів зі стінами. На основі цих даних будується модель, яка здатна передбачати можливі варіанти подальшого руху.

Було протестовано кілька підходів до побудови моделі. Лінійна регресія

дозволяє будувати прості передбачення, проте вона має обмеження при обробці складних траєкторій. Тому додатково використовувалися методи множинної регресії, які враховують більше змінних, а також алгоритми, засновані на нейронних мережах. Підхід з використанням нейронних мереж, зокрема рекурентних (RNN) та згорткових (CNN), дозволив суттєво покращити точність прогнозування.

# 3 РОЗРОБКА ТА РЕЗУЛЬТАТ РОБОТИ МОДУЛЯ

## 3.1 Особливості програмної реалізації модуля

Програмне забезпечення модуля прогнозування руху в лабіринті поєднує алгоритми обробки даних у реальному часі та методи машинного навчання. Його основна функція – аналіз отриманих від ігрового середовища координат прохідних секторів і стін, формування прогнозу щодо потенційно доступних напрямків руху та перевірка цих прогнозів у середовищі Godot.

Обробка даних відбувається в кілька етапів:

1) збір даних – рушій Godot передає координати секторів, де можна рухатися, та координати стін через WebSocket-з'єднання;

2) попередня обробка – Python-сервер відфільтровує некоректні або неповні координати та формує послідовності точок для аналізу;

3) аналіз закономірностей – використовуючи метод лінійної регресії, Python-скрипт виявляє тренди в русі об’єкта, що допомагає прогнозувати його майбутні координати;

4) прогнозування – на основі історичних координат і поточних обмежень модуль передбачає можливі варіанти руху без зіткнення зі стінами;

5) перевірка прогнозу – отримані передбачені координати передаються в Godot, де вони порівнюються з фактичною доступністю секторів. У разі розбіжностей модель коригується.

Для підвищення точності прогнозування Python-сервер застосовує регресії, які дозволяють на основі попередніх рухів визначати ймовірні траєкторії об'єкта. У процесі навчання модуль використовує історичні координати, поступово адаптуючись до нових умов лабіринту.

Для реалізації прогнозування використано такі бібліотеки:

- NumPy – для роботи з масивами координат та їх обробки [15];
- Scikit-learn – для реалізації регресій та оцінки похибок [16];
- WebSockets – для швидкої взаємодії між сервером та ігровим рушієм Godot [17].

Godot реалізує обробку даних через WebSocket-клієнт, який отримує

прогнозовані координати від сервера та порівнює їх із фактичними умовами в лабіринті. Якщо прогнозована точка є прохідною, вона використовується для коригування руху об'єкта.

Однією з особливостей модуля є автоматична адаптація алгоритму до змін у структурі лабіринту. Якщо прогнозована точка виявляється стіною, модуль використовує цей факт для оновлення своїх моделей. Завдяки цьому алгоритм поступово навчається на помилках та покращує точність передбачень.

У разі розбіжностей між прогнозованими та реальними координатами сервер аналізує причини помилки та здійснює корекцію:

- якщо прогнозована точка виявилася стіною, її виключають із майбутніх розрахунків;
- якщо похибка прогнозу зростає, модуль збільшує вибірку історичних координат для більш точного аналізу;

Ці механізми забезпечують стабільність роботи алгоритму та його здатність до самонавчання.

Архітектура передбачає можливість розширення: у разі складніших лабіринтів або необхідності більш точного прогнозування можна підключати додаткові методи машинного навчання. Наприклад, використання нейронних мереж для аналізу траєкторій руху може дозволити модулю прогнозувати складніші патерни пересування об’єкта.

Модуль також підтримує оптимізацію швидкодії: алгоритм працює з обмеженим набором останніх координат, що зменшує навантаження на обчислювальні ресурси та забезпечує швидку реакцію.

## 3.2 Прогнозування координат

Серверна частина модуля виконує обробку отриманих координат з використанням методів машинного навчання для прогнозування подальших можливих напрямків руху об'єкта в лабіринті.

На основі отриманих даних про доступні сектори та стіни будується навчальний набір, який включає координати прохідних секторів (clear_x, clear_y)

та координати перешкод (wall_x, wall_y).

Процес обробки даних на сервері включає кілька етапів:

1) прийом даних: отримання координат секторів з Godot через WebSocket-з'єднання.

2) попередня обробка: видалення аномалій та обмеження кількості історичних записів для підвищення швидкодії.

3) навчання моделей: побудова моделей на основі координат.

4) прогнозування: обчислення можливих координат руху об'єкта для наступних кроків.

5) передача результатів: відправка прогнозованих координат назад у Godot для відображення у реальному часі.

Застосування кількох моделей машинного навчання дозволяє порівнювати їхню точність і надійність при змінних умовах навігації в лабіринті. Такий підхід забезпечує гнучкість та можливість подальшого вдосконалення модуля.

Після реалізації всіх компонентів була проведена серія експериментів для перевірки ефективності розробленої модуля прогнозування руху об'єкта в лабіринті. На рисунку 3.1 зображено типовий результат роботи модуля в середовищі Godot під час одного з тестових прогонів.

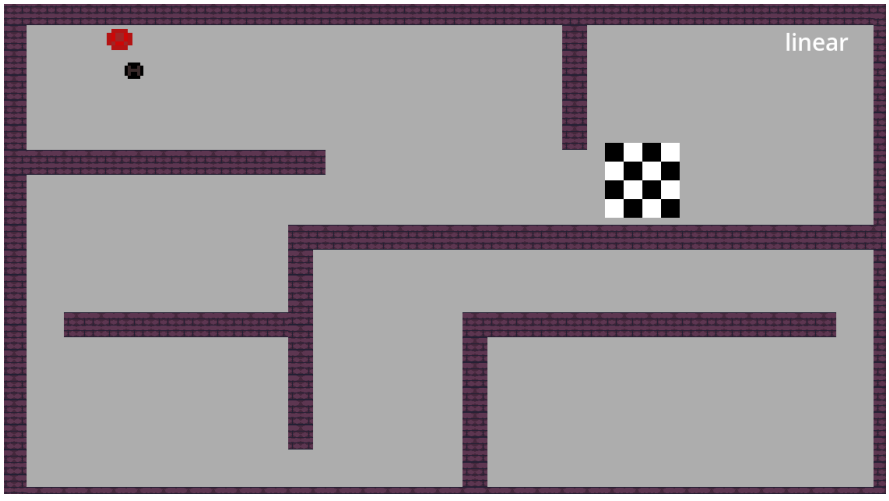



Рисунок 3.1 – Рух об’єкта у лабіринті

На наведеному прикладі великий круг відображає поточну позицію об'єкта у лабіринті, менша точка вказує на прогнозовані координати потенційних напрямків

руху, що були згенеровані різними моделями машинного навчання на основі поточних і попередніх даних. Об'єкт рухається у складному середовищі з великою кількістю стін і вузьких проходів, що створює виклики для точного прогнозування.

На зображенні 3.2 представлено фрагмент консольного виводу Python-сервера, який здійснює прогнозування координат об'єктів у просторі на основі попередніх позицій та взаємодій з середовищем.

```
--- Отримано нове повідомлення ---
[INFO] Кількість даних: 27 точок
[LINEAR] Прогноз Y: [122.04  95.2   65.23  73.46 102.44]
[LINEAR] Прогноз X: [141.33 111.33  81.33  91.33 121.33]
[POLYNOMIAL] Прогноз Y: [122.68  94.73  68.6   70.93 101.62]
[POLYNOMIAL] Прогноз X: [141.33 111.33  81.33  91.33 121.33]
[RIDGE] Прогноз Y: [122.03  95.19  65.23  73.46 102.44]
[RIDGE] Прогноз X: [141.33 111.33  81.33  91.33 121.33]
[SVR] Прогноз Y: [115.17  85.37 107.19 106.63 106.98]
[SVR] Прогноз X: [141.23 111.43 125.57 125.09 125.44]
[RANDOM_FOREST] Прогноз Y: [117.59  88.51  64.11  68.57  97.49]
[RANDOM_FOREST] Прогноз X: [142.73 109.77  77.5   88.47 116.7 ]
```

Рисунок 3.2 – Консольний вивід координат Python-сервером

На даному етапі видно, що сервер обробив 27 вхідних точок, які включають координати об’єктів у "чистих" та "стінових" секторах. На основі цих даних здійснюється навчання п’яти моделей: Linear Regression, Polynomial Regression, Ridge Regression, Support Vector Regression (SVR) та Random Forest. Для кожної моделі окремо прогнозуються як Y-координати , так і X-координати.

## 3.3 Оцінка точності моделей

Оцінювання ефективності роботи програмного модуля прогнозування здійснювалося шляхом порівняння моделей машинного навчання за основним критерієм — середнім часом проходження об'єктом до фінішної точки. Час визначався у середовищі Godot автоматично при досягненні об’єктом заданих координат фінішу, і надсилався на сервер разом з ідентифікатором моделі. Таким чином, для кожної моделі було сформовано узагальнене уявлення про її здатність ефективно орієнтуватися у середовищі лабіринту [20].

Кожна модель опрацьовувала однакову навчальну вибірку, що включала

координати вільного простору, стін та фінішної точки. На відміну від базового підходу, результати прогнозу оброблялися додатково: для кожного прогнозованого значення координат обчислювалося середнє арифметичне з координатами фінішу. Такий підхід дозволив спрямувати об'єкт не лише за історією його руху, а й у бік цільової позиції, що значно підвищило точність у деяких моделях.

На підставі аналізу результатів проходження маршруту, поданих на Рисунках 3.3–3.7, встановлено, що різні моделі демонструють суттєво відмінну швидкість досягнення фінішної точки.

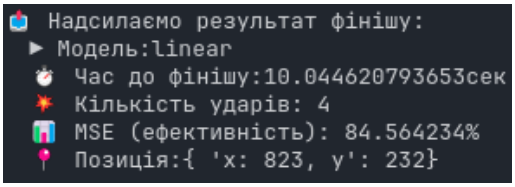



Рисунок 3.3 – Результат проходження linear моделлю

Лінійна регресія, як найпростіша модель з представлених, показала доволі посередній результат. Прогнозована траєкторія виявилася прямолінійною, з недостатнім урахуванням геометрії лабіринту. Через це об'єкт часто здійснював додаткові повороти після контакту зі стінами, що спричинило затримки в русі. Час проходження у понад 10 секунд підтверджує обмеження моделі при роботі зі складними середовищами, хоча ефективність не найгірша.

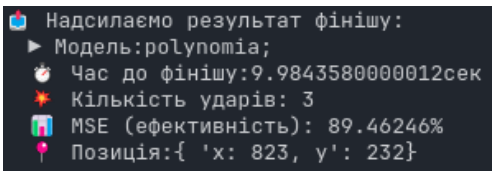



Рисунок 3.4 – Результат проходження polynomial моделлю

На рисунку 3.4 помітно, що поліноміальна регресія другого ступеня змогла частково врахувати криволінійні ділянки шляху. Завдяки цьому прогнозована траєкторія стала плавнішою, що дозволило уникнути зайвих відхилень. Хоча покращення в порівнянні з лінійною регресією є незначним, модель показала більшу гнучкість. Тим не менш, залишаються ознаки перенавчання, які в окремих

ситуаціях можуть призводити до неочікуваних змін траєкторії.

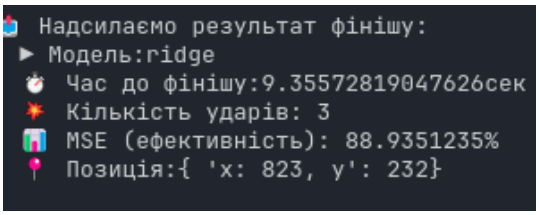



Рисунок 3.5 – Результат проходження ridge моделлю

Ridge-регресія демонструє хорошу збалансованість між простотою лінійної моделі та стійкістю до перенавчання. Порівняно з попередніми двома підходами, ця модель дозволила об’єкту швидше досягти фінішу — менш ніж за 9.4 секунди та має чудову ефективність, яка становить 89%. Результат пояснюється її здатністю враховувати взаємозв’язки між ознаками без надмірної чутливості до вибіркових змін у середовищі.

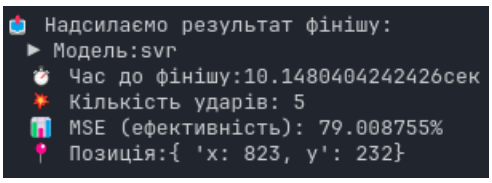



Рисунок 3.6 – Результат проходження svr моделлю

Модель на базі методу опорних векторів показала один із найгірших результатів — час проходження перевищив 10.1 секунди та ефективність склала 79%. Попри те, що SVR добре працює з невеликими вибірками та здатна моделювати складні функції, її продуктивність виявилася нестабільною в умовах динамічного середовища. Причиною стали надмірно точкові передбачення, які ускладнили адаптацію до геометрії лабіринту. Це призвело до повторних змін напрямку та втрати ефективності.

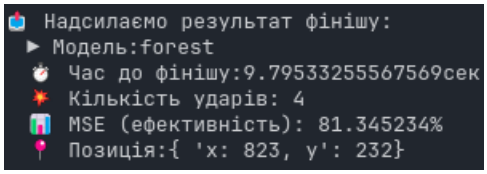



Рисунок 3.7 – Результат проходження random_forest моделлю

Найкращий результат серед усіх моделей показав Random Forest, забезпечивши час проходження менше 9.8 секунд, але MSE рівна 81%. Така ефективність обумовлена здатністю ансамблю дерев враховувати нелінійні залежності, варіативність у виборі ознак та стійкість до шумів у даних. Об'єкт рухався впевнено, з мінімальною кількістю коригувань, що свідчить про високу придатність моделі для задач прогнозування руху в лабіринтах.

# ВИСНОВКИ

У ході дослідження було розроблено модуль прогнозування руху об’єкта в лабіринті на основі методів машинного навчання. Основні досягнення роботи:

1. Проведено аналіз предметної області, що охоплює специфіку орієнтації в лабіринтах, методи збирання координатних даних у 2D-середовищі, типові перешкоди та вимоги до адаптивної навігації. Встановлено основні труднощі, пов’язані з обмеженими обчислювальними ресурсами, змінністю середовища та необхідністю високої точності прогнозів.

2. Оцінено сучасні підходи до прогнозування траєкторій руху за допомогою машинного навчання: розглянуто алгоритми A*, Dijkstra, методи комп’ютерного зору, згорткові та рекурентні нейронні мережі, підкріплене навчання. Виявлено їхні переваги й недоліки, зокрема чутливість до обчислювальних ресурсів і потребу в великій кількості навчальних даних.

3. Розроблено клієнтський модуль у середовищі Godot, який фіксує координати об'єкта, зіткнення з перешкодами та положення фінішної точки. Забезпечено збирання координатних даних у реальному часі та їх формалізацію у формат, придатний для обробки сервером.

4. Створено серверну частину модуля на Python, яка реалізує обчислення з використанням п’яти моделей машинного навчання: Linear, Polynomial, Ridge, SVR та Random Forest. Налагоджено передачу даних між клієнтом і сервером через WebSocket.

5. Запроваджено механізм корекції прогнозів, який зміщує результат у напрямку до фінішної точки. Для цього використовується обчислення середнього значення між прогнозованими та цільовими координатами, що дозволяє підвищити релевантність напрямку руху.

6. Реалізовано кілька режимів руху об’єкта, зокрема режим самонавчання, в якому модуль враховує помилки попередніх прогнозів, а також автономний інтелектуальний режим із пам’яттю перешкод.

7. Проведено серію експериментів, під час яких протестовано всі п’ять моделей на однакових вхідних даних. Критерієм оцінки обрано середній час

досягнення фінішу в середовищі Godot. Найгірший результат показала модель SVR, тоді як найефективнішою виявилась Random Forest, забезпечивши проходження менш ніж за 9.8 секунди.

8. Оцінено ефективність системи та потенціал її подальшого використання. Запропонована архітектура довела свою працездатність, гнучкість та здатність до адаптації в умовах змінного середовища. Модуль може бути застосований як у комп'ютерних іграх, так і в задачах робототехніки та автономної мобільності. Подальші дослідження доцільно спрямувати на інтеграцію глибших нейронних мереж та оптимізацію системи для складніших сценаріїв.

Розроблений модуль може бути застосована не лише в комп'ютерних іграх, але й у сфері робототехніки та автономної навігації, де необхідно приймати рішення про рух у змінному середовищі. Подальші дослідження можуть бути спрямовані на розширення функціональності моделі, зокрема на використання глибших нейронних мереж для аналізу складніших патернів руху, а також оптимізацію алгоритмів для роботи у великих та динамічних середовищах.

Запропонований підхід дозволяє ефективно аналізувати структуру лабіринту, прогнозувати можливі траєкторії руху та адаптуватися до змін у середовищі в режимі реального часу.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Y.Duraisamy: Navigating network traffic: an exploration of maze algorithm applications in machine learning. 2024. No 3. Електронний ресурс: https://iirjet.org/index.php/home/article/view/303

2. A. Gerós, R. Cruz, F. de Chaumont, J. S. Cardoso, P. Aguiar: Deep learning-based system for real-time behavior recognition and closed-loop control of behavioral mazes using depth sensing. 2022. No 3. Електронний ресурс: https://www.biorxiv.org/content/10.1101/2022.02.22.481410v1.full

3. Бабич В., Костенко А., Плеша В., Плеша М., Хмілярчук Л.: Задача пошуку найкоротшого шляху: порівняльний аналіз основних алгоритмів. 2020. Електронний ресурс: https://journals.politehnica.dp.ua/index.php/it/article/view/306?

4. Зінченко О. В., Звенігородський О. С., Кисіль Т. М.: Згорткові нейронні мережі для вирішення задач комп'ютерного зору. 2022. No 2 (75). Електронний ресурс: https://tit.dut.edu.ua/index.php/telecommunication/article/view/2417?

5. M. P. Mukhina, V. Yu. Derkach, A. P. Prymak.: Combination of Hough transform and Canny edge detector for improvement of linear object detection results. 2018. No.58. Електронний ресурс: https://jrnl.nau.edu.ua/index.php/ESU/article/view/13515

6. С. О. Субботін, Нейронні мережі: Теорія та практика. Електронний ресурс: https://uk.eitca.org

7. Гентош Л., Левкович Р.: Застосування згорткових нейронних мереж для прогнозування споживання електроенергії. 2024. Електронний ресурс: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/187

8. Ramanjeet S., Jing R., Xianke L.: A review of deep reinforcement learning algorithms for mobile robot path planning. 2023. Електронний ресурс: https://www.mdpi.com/2624-8921/5/4/78

9. Roderick M., MacGlashan J., Tellex S.: A deep Q-network (DQN) based path planning method for mobile robots. 2020. Електронний ресурс: https://ieeexplore.ieee.org/abstract/document/8812452

10. Рекурентні нейронні мережі. Електронний ресурс:

https://internetri.net/qntm/2021/06/23/rekurentni-nejronni-merezhi-rnn/

11. S. Gilewski. Edge detection and processing using Canny edge detector and Hough transform. 2024. Електронний ресурс: https://wttech.blog/blog/2022/edge-detection-and-processing-using-canny-edge-detector-and-hough-transform/

12. Luisa Shu Yi Chiu.Dynamic maze puzzle navigation using deep reinforcement learning. 2024. Електронний ресурс: https://www.proquest.com/openview/57471503ba7fbce3f79c05f3ec708811/1?pq-origsite=gscholar&cbl=18750&diss=y

13. B. Zhou, M. Shi, Y. Li. Maze solving using deep Q-network. 2023. Електронний ресурс: https://ieeexplore.ieee.org/document/10083691

14. Y. Zhang, S. Pan, Z. Zheng. Trajectory-aware deep reinforcement learning navigation using multichannel cost maps. 2024. Електронний ресурс: https://www.sciencedirect.com/science/article/pii/S0957417424001040

15. NumPy documentation. Електронний ресурс: https://numpy.org/doc/2.2/

16. Scikit-learn documentation. Електронний ресурс: https://scikit-learn.org/0.21/documentation.html

17. Websockets documentation. Електронний ресурс: https://websockets.readthedocs.io/en/stable/

18. A. B. Patel, S. R. Dhanvijay. Investigating navigation strategies in maze environments using DRL. 2023. Електронний ресурс: https://ieeexplore.ieee.org/document/10156437

19. T. Yu, R. T. Iqbal, J. Han. Mapless navigation via hierarchical reinforcement learning with memory-decaying novelty. 2024. Електронний ресурс: https://www.sciencedirect.com/science/article/pii/S0921889024000029

20. M. Everett, Y. Chen, J. P. How. Motion planning among dynamic obstacles with deep reinforcement learning. 2021. Електронний ресурс: https://arxiv.org/abs/2004.11836

21. H. Zhang, T. Deng, S. Wang. Vision-based mobile robot obstacle avoidance using deep reinforcement learning. 2021. Електронний ресурс: https://www.mdpi.com/2504-446X/5/2/29

22. F. Zhang, L. Tang, J. Yuan. End-to-end autonomous navigation based on

deep reinforcement learning with survival penalty. 2023. Електронний ресурс: https://www.mdpi.com/2504-446X/7/1/13

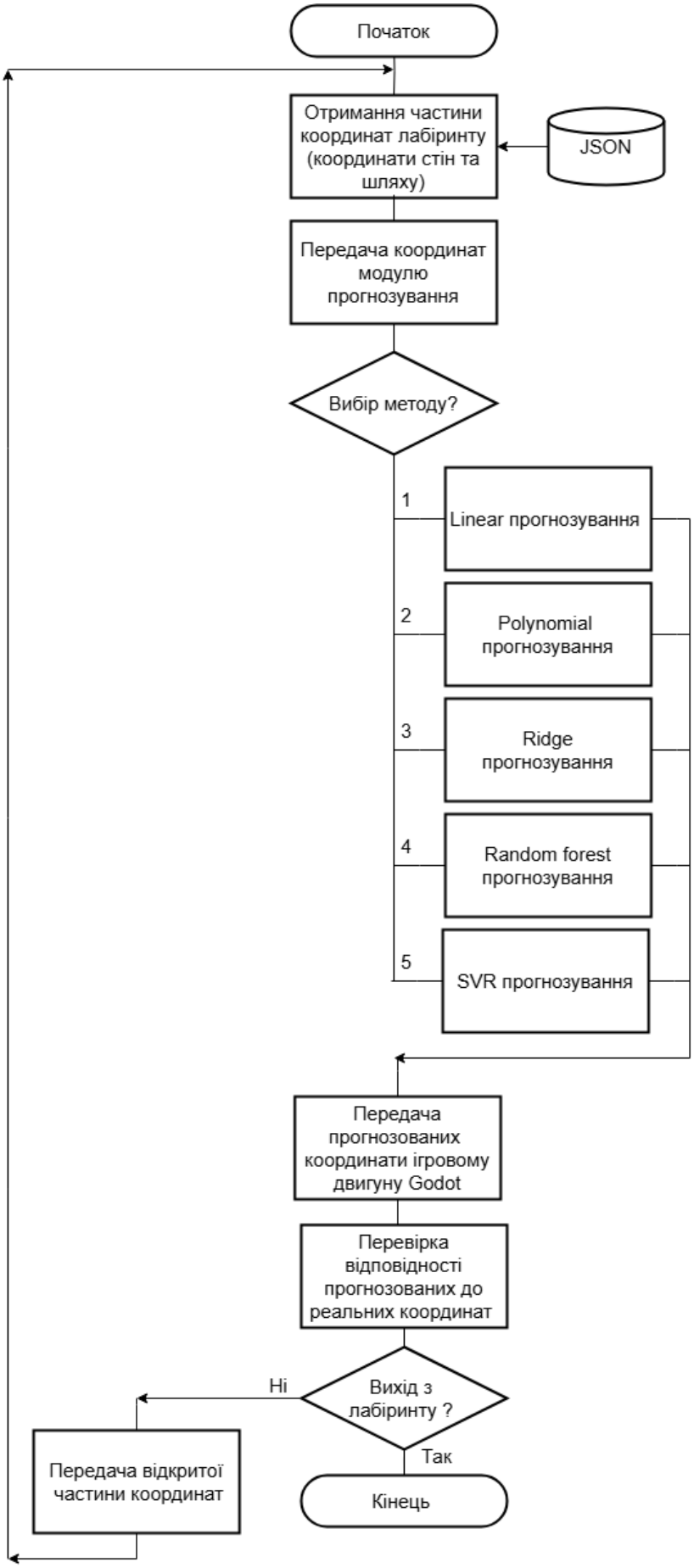
23. E. Valenzuela, H. Schaa, Nicolas A. Barriga, G. Patow. Using search algorithm statistics for assessing maze and puzzle difficulty. 2025. Електронний ресурс:https://www.researchgate.net/publication/384602919_Using_Search_Algorithm_Statistics_for_Assessing_Maze_and_Puzzle_Difficulty

24. Кузьмич Б. А., Биковий П. Є. Програмний модуль прогнозування перешкод у лабіринті на основі машинного навчання. *Інтелектуальні інформаційні технології в прикладних дослідженнях(IITAR–2025)*: збірник тез доповідей студентської науково-практичної конференції м.Тернопіль, 27-29 травня 2025 р. Тернопіль: ЗУНУ, 2025. С. 339–341.

25. Комар М. П., Васильків Н. М., Гладій Г. М., Коваль В. С.: Програма переддипломної практики здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 122 «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки». Тернопіль: ЗУНУ, 2024. 38 с. Електронний ресурс: https://moodle.wunu.edu.ua/course/view.php?id=2929

Додаток А

Схема алгоритму роботи модуля прогнозування координат



Додаток Б

Код Python сервера обробки координат

```python
class PredictionServer:
    def __init__(self):
        self.data_history = {
            'clear_x': [],
            'clear_y': [],
            'wall_x': [],
            'wall_y': []
        }
        self.true_collisions = []
        self.finish_position = {"x": 0.0, "y": 0.0}

        self.models = {
            'linear': LinearRegression(),
            'polynomial': make_pipeline(PolynomialFeatures(degree=2), LinearRegression()),
            'ridge': Ridge(alpha=1.0),
            'svr': SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1),
            'random_forest': RandomForestRegressor(n_estimators=100)
        }

        self.min_data_points = 5
        self.max_data_points = 50
        self.prediction_window = 5
```

```python
async def handle_client(self, websocket, path):
    print("✅ Клієнт підключений")
    try:
        async for message in websocket:
            try:
                print("\n--- Отримано нове повідомлення ---")
                data = self._parse_message(message)
                if not data:
                    await websocket.send(json.dumps({
                        "error": "Отримано порожні дані",
                        "status": "error"
                    }))
                    continue

                self._update_data(data)

                if len(self.data_history['clear_x']) >= self.min_data_points:
                    response = self._generate_predictions()
                else:
                    response = {
                        "status": "waiting",
                        "message": f"Недостатньо даних (потрібно {self.min_data_points})",
                        "current_data": len(self.data_history['clear_x'])
                    }

                await websocket.send(json.dumps(response))

            except json.JSONDecodeError:
                await websocket.send(json.dumps({"error": "Невірний формат JSON", "status"
            except Exception as e:
                await websocket.send(json.dumps({"error": str(e), "status": "error"}))
```

```
def _parse_message(self, message):
    try:
        data = json.loads(message)
        clear_x = data['clearSector'].get('positionClearSectorX', [])
        clear_y = data['clearSector'].get('positionClearSectorY', [])
        wall_x = data['wallSector'].get('positionWallSectorX', [])
        wall_y = data['wallSector'].get('positionWallSectorY', [])
        collisions = data.get('collisions', [])

        if not all(isinstance(lst, list) for lst in [clear_x, clear_y, wall_x, wall_y]):
            raise ValueError("Координати мають бути списками")

        return {
            'clear_x': [float(x) for x in clear_x],
            'clear_y': [float(y) for y in clear_y],
            'wall_x': [float(x) for x in wall_x],
            'wall_y': [float(y) for y in wall_y],
            'collisions': collisions
        }

    except Exception as e:
        raise ValueError(f"Помилка парсингу даних: {str(e)}")

def _update_data(self, new_data):
    for key in self.data_history:
        self.data_history[key].extend(new_data[key])
        self.data_history[key] = self.data_history[key][-self.max_data_points:]

    if 'collisions' in new_data and new_data['collisions']:
        for collision in new_data['collisions']:
            if isinstance(collision, dict) and 'wall' in collision:
                wall_status = collision.get('wall')
                if wall_status in [0, 1]:
                    self.true_collisions.append(int(wall_status))
                    self.true_collisions = self.true_collisions[-self.max_data_points:]

            if isinstance(collision, dict) and collision.get("type") == "finish_position":
                finish_data = collision.get("data", {}).get("finish_position", {})
                try:
                    self.finish_position["x"] = float(finish_data.get("x", 0.0))
                    self.finish_position["y"] = float(finish_data.get("y", 0.0))
                except Exception as e:
                    print(f"⚠️ Не вдалося зчитати координати фінішу: {e}")

async def main():
    server = PredictionServer()
    async with websockets.serve(server.handle_client, "localhost", 6000):
        print("🚀 Сервер запущено на ws://localhost:6000")
        await asyncio.Future()
```

```python
def _predictions(self):
    min_len = min(len(self.data_history[key]) for key in self.data_history)
    if min_len < self.prediction_window:
        raise ValueError("Недостатньо даних для прогнозу")

    print(f"[INFO] Кількість даних: {min_len} точок")

    finish_x = self.finish_position["x"]
    finish_y = self.finish_position["y"]

    X = np.column_stack((
        self.data_history['clear_x'][:min_len],
        self.data_history['wall_x'][:min_len],
        self.data_history['wall_y'][:min_len],
        [finish_x] * min_len,
        [finish_y] * min_len
    ))

    y_y = np.array(self.data_history['clear_y'][:min_len])
    y_x = np.array(self.data_history['clear_x'][:min_len])

    results = {"status": "success", "models": {}}

    X_pred = np.column_stack((
        self.data_history['clear_x'][-self.prediction_window:],
        self.data_history['wall_x'][-self.prediction_window:],
        self.data_history['wall_y'][-self.prediction_window:],
        [finish_x] * self.prediction_window,
        [finish_y] * self.prediction_window
    ))
```

# Додаток В

## Код модуля руху об’єкта

```
func _physics_process(delta):
    if Input.is_action_just_pressed('startUserMove'):
        startUserMove = true
        print("🕹️ Режим ручного керування:", startUserMove)
    if Input.is_action_just_pressed('AI'):
        useAI = true
        print("🤖 AI-режим активовано")

    if followPredictedPath and path.size() > 0:
        move_along_path()
    elif useAI:
        run_AI_mode()
    elif startUserMove:
        userMove()
    else:
        objectMove(delta)
        moveX()
        moveY()

    move_and_slide()
    positionTimer(delta)

    if G.should_follow_prediction:
        start_following_prediction()
        startTimerFollowing += delta
        if startTimerFollowing >= endTimerFollowing:
            G.should_follow_prediction = false

func move_along_path():
    if path_index >= path.size():
        followPredictedPath = false
        return

    var target = path[path_index]
    var direction = (target - global_position).normalized()
    velocity = direction * SPEED

    if global_position.distance_to(target) <= CLOSE_DISTANCE:
        path_index += 1

func start_following_prediction():
    path.clear()
    path_index = 0
    var x_list = G.previous_positionFromPythonX.get("linearX", [])
    var y_list = G.previous_positionFromPythonY.get("linearY", [])

    if x_list.size() != y_list.size() or x_list.is_empty():
        print("⚠️ Прогноз відсутній або некоректний")
        return

    for i in range(x_list.size()):
        path.append(Vector2(x_list[i], y_list[i]))

    followPredictedPath = true
```

```
func randomMove():
    if randf() < 0.5:
        directionX = -1 if randf() < 0.5 else 1
    if randf() < 0.5:
        directionY = -1 if randf() < 0.5 else 1

func objectMove(delta):
    if touchWallTrue:
        bodyTouchWall()
    else:
        randTimer(delta)

func userMove():
    if Input.is_action_just_pressed("ui_left"):
        directionX = -1
    if Input.is_action_just_pressed("ui_right"):
        directionX = 1
    if Input.is_action_just_pressed("ui_up"):
        directionY = -1
    if Input.is_action_just_pressed("ui_down"):
        directionY = 1

func moveX():
    velocity.x = directionX * SPEED if directionX else move_toward(velocity.x, 0, SPEED)

func moveY():
    velocity.y = directionY * SPEED if directionY else move_toward(velocity.y, 0, SPEED)
```

```
func randTimer(delta):
    startTimer += delta
    if startTimer >= finishTimer:
        randomMove()
        startTimer = 0.0
        finishTimer = randi() % 3

func positionTimer(delta):
    startTimerPosition += delta
    if startTimerPosition >= finishTimerPosition:
        positionInMaze()
        startTimerPosition = 0.0

func positionInMaze():
    if not touchWallTrue:
        G.positionClearSector["positionClearSectorX"].append(float(position.x))
        G.positionClearSector["positionClearSectorY"].append(float(position.y))

func bodyTouchWall():
    directionX *= -1
    directionY *= -1
```

Додаток Г

Код модуля відправлення координат

```
var client = WebSocketPeer.new()
var url = "ws://localhost:6000"
var send_timer = 0.0
var send_interval = 1.0  # Відправляти кожну секунду

func _ready():
    var err = client.connect_to_url(url)
    if err != OK:
        push_error("Не вдалося підключитися до сервера")
    else:
        print("✅ Успішне підключення до сервера")

func _process(delta):
    client.poll()

    # Таймер для відправки даних
    send_timer += delta
    if send_timer >= send_interval:
        send_timer = 0.0
        send_data()

    # Обробка вхідних повідомлень
    while client.get_available_packet_count() > 0:
        var packet = client.get_packet()
        handle_server_response(packet.get_string_from_utf8())

func send_data():
    if client.get_ready_state() != WebSocketPeer.STATE_OPEN:
        print("⚠️ З'єднання не встановлено")
        return
func handle_server_response(response):
    var json_data = JSON.parse_string(response)
    if json_data == null:
        push_error("Не вдалося розібрати відповідь сервера")
        return

    if json_data.has("error"):
        push_error("Помилка сервера: " + str(json_data["error"]))
    else:
        G.process_server_data(json_data)
```

Додаток Д

Копії опублікованих результатів

Західноукраїнський національний університет

Факультет комп’ютерних інформаційних технологій

Кафедра інформаційно-обчислювальних систем і управління

ЗБІРНИК ТЕЗ ДОПОВІДЕЙ

Студентської науково-практичної конференції

ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ПРИКЛАДНИХ ДОСЛІДЖЕННЯХ

(IITAR-2025)

27-29 травня 2025 року

Тернопіль

2025

***Кузьмич Богдан***
*студент групи КН-41*
kuzmych242424@gmail.com
***Биковий Павло***
*к.т.н., доцент*
pb@wunu.edu.ua
*Західноукраїнський національний університет*
*Тернопіль, Україна*

# ПРОГРАМНИЙ МОДУЛЬ ПРОГНОЗУВАННЯ ПЕРЕШКОД У ЛАБІРИНТІ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

Завдання передбачення та уникнення перешкод є важливим у сфері робототехніки, автономної навігації та інтелектуальних агентів. Зі зростанням складності середовищ, у яких функціонують ці системи, виникає потреба в адаптивних алгоритмах, здатних передбачати зміни структури навколишнього простору в режимі реального часу [1]. Для вирішення даної проблеми пропонується розробка програмного модуля що базується на взаємодії ігрового рушія Godot та серверної частини на Python, які обмінюються даними через WebSocket-з'єднання в режимі реального часу.

Основу аналітичної частини складає ансамбль моделей машинного навчання, який виконує прогноз руху на основі історії проходження секторів лабіринту. Застосовано кілька регресійних алгоритмів, включаючи Linear Regression, Polynomial Regression, Ridge Regression, SVR та Random Forest, що забезпечують адаптацію до різних типів траєкторій і складності середовища [2]. У результаті виконаних експериментів встановлено, що модуль здатен адаптуватися до змін у структурі лабіринту, навчаючись у процесі взаємодії з середовищем [3].

На рисунку 1 представлений алгоритм прогнозування. Він відображає повний цикл взаємодії між віртуальним середовищем (Godot) та аналітичним сервером (Python), який здійснює прогнозування майбутніх координат об'єкта на основі даних про лабіринт. Алгоритм реалізовано з використанням принципів машинного навчання й підтримкою декількох моделей регресії.

На першому етапі ініціалізується вся система. Godot отримує координати середовища, включаючи стіни та точки проходження, які формуються у форматі JSON. Ці координати відправляються до окремого модуля прогнозування, що виконується на Python-сервері.

Після отримання даних сервер паралельно запускає п'ять моделей машинного навчання: Linear, Polynomial, Ridge, SVR та Random Forest. Модель Linear Regression забезпечує найпростішу пряму оцінку шляху, що ефективно працює на відкритих ділянках. Polynomial Regression дозволяє моделювати вигнуті траєкторії за рахунок нелінійних залежностей. Ridge

корельованих або нестійких даних. SVR (Support Vector Regression) використовує RBF-ядро та ефективно працює у складних, нелінійних середовищах. Random Forest Regression як ансамблевий метод виявляє узагальнюючі закономірності, найменше піддаючись шуму у вибірці.

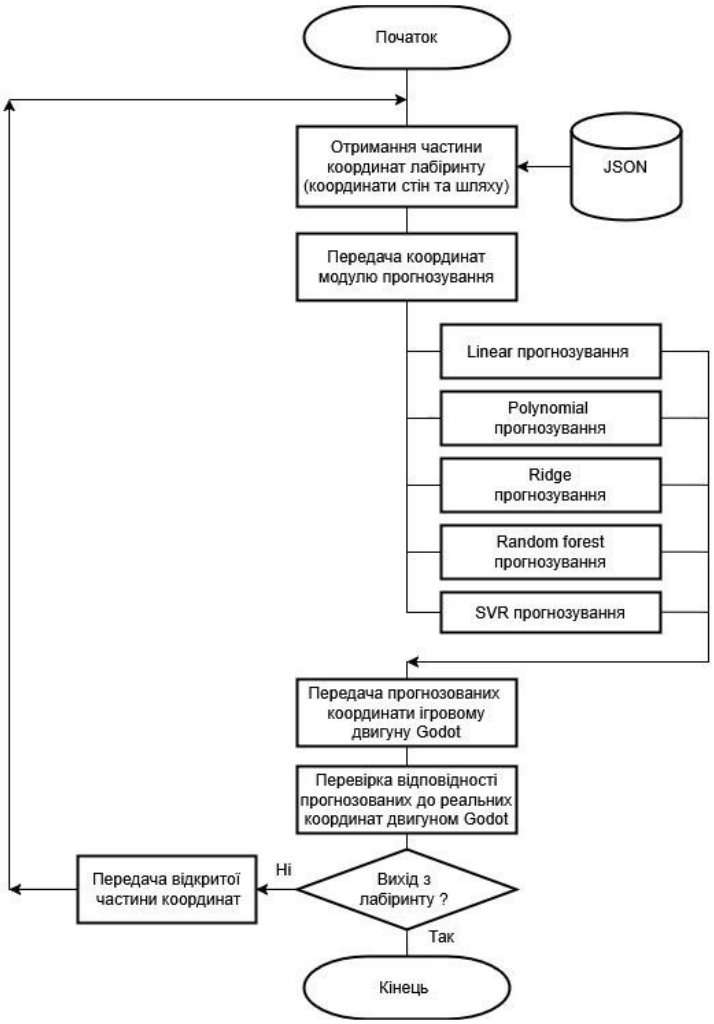



Рисунок 1 – Алгоритм системи прогнозування у лабіринті

Після формування прогнозованих координат, обрана модель передає результат назад до ігрового рушія Godot. Тут відбувається перевірка отриманої траєкторії та її реалізація у вигляді руху об'єкта в середовищі.

Завершальним кроком є перевірка: чи досяг об'єкт координат виходу з лабіринту (фініш). Якщо так — система зупиняє виконання циклу прогнозування. В іншому випадку цикл запускається знову з новими координатами. Ця схема дозволяє організувати безперервний процес навчання та перевірки моделі в реальному або наближеному до реального часі.

Практична цінність запропонованої системи полягає у можливості використання модуля в реальному часі для забезпечення автономної навігації, що актуально для рятувальних роботів, дронів, симуляторів та мобільних агентів. У перспективі планується інтеграція нейронних мереж для прогнозування складніших траєкторій.

## Список використаних джерел


1. Y.Duraisamy: Navigating network traffic: an exploration of maze algorithm applications in machine learning. 2024. No 3. Електронний ресурс: https://iirjet.org/index.php/home/article/view/303
2. A. Gerós, R. Cruz, F. de Chaumont, J. S. Cardoso, P. Aguiar: Deep learning-based system for real-time behavior recognition and closed-loop control of behavioral mazes using depth sensing. 2022. No 3. Електронний ресурс: https://www.biorxiv.org/content/10.1101/2022.02.22.481410v1.full
3. Бабич В., Костенко А., Плеша В., Плеша М., Хмілярчук Л.: Задача пошуку найкоротшого шляху: порівняльний аналіз основних алгоритмів. 2020. Електронний ресурс: https://journals.politehnica.dp.ua/index.php/it/article/view/306?