НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.057.8

До захисту допущено:
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2021 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Архітектура програмного забезпечення для навчання та тестування самокерованих транспортних засобів»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ІТ-з03мп
Іванчевська Діана Валеріївна _____

Керівник:
професор, д.т.н.
Павлов Олександр Анатолійович _____

Рецензент:
доцент кафедри ІСТ, к.т.н.
Жураковська Оксана Сергіївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2021р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Іванчевській Діані Валеріївні

1. Тема дисертації «**Архітектура програмного забезпечення для навчання та тестування самокерованих транспортних засобів**», науковий керівник дисертації Павлов Олександр Анатолійович, д.т.н., професор, затверджені наказом по університету від «25» жовтня 2021 р. № 3575-с

2. Термін подання студентом дисертації «6» грудня 2021 р.

3. Об'єкт дослідження – програмне забезпечення для навчання та тестування самокерованих транспортних засобів

4. Предмет дослідження – архітектура програмного забезпечення для навчання та тестування самокерованих транспортних засобів.

5. Перелік завдань, які потрібно розробити – аналіз проблеми та існуючих рішень; розробка архітектури програмного забезпечення; дослідження ефективності розробленої архітектури програмного забезпечення.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 2 плакати

7. Орієнтовний перелік публікацій – одна публікація

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Перевірка на співпадіння	доцент Лісовиченко О.І.		

9.Дата видачі завдання «30» вересня 2020 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Ознайомлення з предметною областю	01.10.2020 – 25.10.2020	
2	Аналіз існуючих рішень	01.12.2020 – 28.12.2020	
3	Аналіз можливостей прийняття рішень в середовищах моделювання	10.01.2021 – 25.02.2021	
4	Розробка архітектури програмного забезпечення та програмна імплементація	16.03.2021 – 23.08.2021	
5	Переддипломна практика	01.09. 2021 – 26.10.2021	
6	Виконання експериментальних досліджень	25.09.2021 – 31.10.2021	
7	Розробка стартап-проекту	01.11.2021 – 12.11.2021	
8	Підготовка доповіді на конференції	12.11.2021 – 19.11.2021	
9	Оформлення пояснювальної записки	15.11.2021 – 21.11.2021	
10	Подання дисертації на попередній захист	22.11.2021	
11	Подання дисертації на захист	6.12.2021	

Студент Діана ІВАНЧЕВСЬКА

Науковий керівник Олександр ПАВЛОВ

РЕФЕРАТ

Актуальність теми. У роботі розглянуто проблему в області програмного забезпечення автомобільної індустрії щодо створення систем автоматизації водіння для самокерованих транспортних засобів, що здатні в умовах невизначеності, різного освітлення та швидко змінюваних погодних обставин автоматично приймати миттєві рішення, планувати траєкторії руху, оминати перешкоди на шляху пересування. На сьогодні проблема прийняття рішень та прогнозування поведінки системами автоматизації водіння ще повністю не вирішена та це один з найважливіших факторів, що стримує розробку автономного транспорту з п’ятим рівнем автономності. Таким чином розробка програмного забезпечення, яке дозволить підвищити ефективність навчання таких систем є як ніколи актуальною.

Мета дослідження. Основною метою є підвищення ефективності навчання та тестування самокерованих транспортних засобів в програмованому середовищі за рахунок покращеній користувацькій взаємодії з програмним забезпеченням.

Об’єкт дослідження: програмне забезпечення для навчання та тестування самокерованих транспортних засобів.

Предмет дослідження: архітектура програмного забезпечення для навчання та тестування самокерованих транспортних засобів.

Для реалізації поставленої мети **сформульовані наступні завдання**:

- проаналізувати досліджувану предметну область та виокремити основні проблеми;
- проаналізувати існуючі рішення в досліджуваній області та виокремити їх недоліки та переваги;
- проаналізувати існуюче теоретичне забезпечення щодо можливостей прийняття рішень в середовищах моделювання;
- проаналізувати можливості засобів машинного навчання для підвищення ефективності навчання самокерованих транспортних засобів та запропонувати спосіб їх інтеграції;

- запропонувати архітектуру програмного забезпечення для найбільш ефективного навчання та тестування автономних транспортних засобів;
- виконати порівняльний аналіз розробленого архітектурного рішення з аналогами для обґрунтування його ефективності;
- узагальнити результати досліджень.

Наукова новизна результатів магістерської дисертації полягає в тому, що набули подальшого розвитку використання алгоритмів машинного навчання шляхом інтеграції відповідних засобів машинного навчання за допомогою розробки програмного забезпечення з використанням підходу компонентно-орієнтованого проектування.

Практичне значення отриманих результатів полягає у створенні архітектури простішого, більш дешевого програмного забезпечення з відкритим вихідним кодом, що використовує засоби штучного інтелекту для підвищення ефективності прийняття миттєвих рішень самокерованими транспортними засобами задля уникнення ризиків непередбачуваного розвитку подій ще на етапі тестування, та впровадженими принципами модульності для покращення можливостей інтеграції з елементами інших систем.

Зв’язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Публікації. Наукові положення дисертації опубліковані в:

Іванчевська Д.В. Етапи розробки програмного забезпечення для навчання та тестування транспорту з вбудованою підтримкою системи самокерування/ Іванчевська Д.В. // Матеріали Першої Всеукраїнської науково-практичної конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021). Секція кафедри інформатики та програмної інженерії. Матеріали конференції. – Київ. – 2021. 22–26 листопада 2021р. – С.199.

Ключові слова: автономний транспортний засіб, машинне навчання, програмне середовище моделювання, модельно – орієнтоване проектування, Unity, C#, Python, Matlab/Simulink.

Розмір пояснювальної записки – 99 аркушів, містить 35 ілюстрацій, 22 таблиць, 7 додатків.

ABSTRACT

Topicality. The paper considers the problem of creation driving automation systems for self-driving vehicles that are able to automatically make instant decisions, plan trajectories, avoid obstacles in the face of uncertainty, different lighting and rapidly changing weather conditions.

Today, the problem of decision-making and the behavior predicting by driving automation systems is not completely solved and this is one of the most important factors hindering the development of autonomous vehicles with the fifth level of autonomy. Thus, the development of software that will improve the learning efficiency of such systems is very important.

The aim of the study. The main target is to increase the efficiency of training and testing of self-driving vehicles in a programmable environment through improved user interaction with the software.

Object of research: software for training and testing autonomous vehicles.

Subject of research: software architecture for training and testing autonomous vehicles.

To achieve this goal, the **following tasks** were formulated:

- analyze the researched subject area and to single out the main problems;
- analyze existing solutions in the study area;
- test open-source software and highlight its disadvantages and advantages;
- analyze existing theoretical information on the possibilities of data processing and presentation of final results to ensure decision-making;
- analyze the possibilities of machine learning algorithms to increase the efficiency of training and testing of self-driving vehicles and suggest a way to implement them;
- propose a software architecture for the most effective training and testing of autonomous vehicles;
- substantiate the effectiveness of the proposed solution;
- summarize research results.

The scientific novelty of the results of the master's dissertation is that the use of machine learning algorithms has been further developed by integrating appropriate machine learning tools through software development using a component-oriented design approach.

The practical value of the obtained results is an architecture of simpler, cheaper open source software that uses artificial intelligence to increase the efficiency of instant decision-making by self-driving vehicles to avoid the risk of unforeseen errors at the testing stage, and implemented modularity principles to improve integration with elements of other systems.

Relationship with working with scientific programs, plans, topics. Work was performed at the Department of Informatics and Software Engineering of the National Technical University of Ukraine «Kyiv Polytechnic Institute. Igor Sikorsky».

Publications. Scientific provisions of the dissertation published in:

Ivanchevska D.V. Development stages of software for training and testing vehicles equipped with driving automation systems / D.V. Ivanchevska // Proceedings of the First All-Ukrainian Scientific and Practical Conference of Young Scientists and Students " Software Engineering and Advanced Information Technologies" (SoftTech-2021) - Kyiv: NTUU “KPI them. Igor Sikorsky”, November 22-26, 2020.

Keywords: autonomous vehicle, machine learning, software simulation, model-based design, Unity, C#, Python, Matlab/Simulink.

Explanatory note size – 99 pages, contains 35 illustrations, 22 tables, 7 applications.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HARA – аналіз загроз та оцінка ризиків (англ. Hazard Analysis and Risk Assessment);

SAE – спільнота автомобільних інженерів (англ. Society of Automotive Engineers);

ПЗ – програмне забезпечення;

ІС – інформаційна система;

HMI – людино-машинний інтерфейс (англ. Human Machine Interface);

ADAS – просунута система підтримки водія (англ. Advanced Driver Assistant Systems);

CAN – локальна мережа контролерів (англ. Controller Area Network);

DOS – атака на відмову в обслуговуванні (англ. Denial of Service);

LIDAR – лідар (англ. Light Detection and Ranging);

RADAR – радар (англ. Radio Detection and Ranging);

ПДР – правила дорожнього руху;

ШІ – штучний інтелект;

SSD – твердотільний накопичувач (англ. Solid-State Drive);

RAM – пам'ять з довільним доступом (англ. Random Access Memory);

DDT– динамічна функція водіння (англ. Dynamic Driving Task);

ASIL – рівні автомобільної інтеграційної безпеки (англ. Automotive Safety Integrity Levels);

HAZOP – небезпека та працездатність (англ. Hazard and Operability analysis);

MISRA – Motor Industry Software Reliability Association;

AUTOSAR – AUTomotive Open System ARchitecture;

FSM – скінченний автомат (англ. finite-state machine);

TTC – час до зіткнення (англ. time to collision);

HRL – ієрархічне навчання з підкріпленням (англ. Hierarchical Reinforcement Learning);

IRL – зворотнє навчання з підкріпленням (англ. Inverse Reinforcement Learning).

ВСТУП

На сьогодні надскладною задачею в області програмного забезпечення автомобільної індустрії є створення самокерованих транспортних засобів, що здатні в умовах невизначеності, різного освітлення та швидко змінюваних погодних обставин автоматично приймати миттєві та довгострокові рішення, планувати траєкторії руху, розпізнавати об’єкти на шляху пересування. Наразі шлях до повної автономності автомобіля на міських дорогах та магістралях ще досі тривалий, але вже існують просунуті системи підтримки водія, що дозволяють автоматизувати процес керування та зменшити навантаження на водія.

Створення такого програмного забезпечення є досить складним, комплексним та дуже дорогим завданням. Окрім цього визначне місце займає питання безпеки, що є чи не найважливішою складовою процесу розробки ПЗ в автомобільній індустрії. До моменту коли таке ПЗ можна випускати в комерційний обіг для користування кінцевими споживачами проходить досить багато часу. Крім того етапи розробки включають обов’язкове тестування таких програмних додатків на реальному транспорті та на реальних автошляхах.

Проблема полягає в тому, що розробник в процесі роботи над кожним нововведенням повинен мати можливість тестувати кожний фрагмент ПЗ, не відходячи від робочої станції. Це призводить до зменшення часу тестування, дозволяє проводити всі необхідні маніпуляції стосовно перевірки безпеки обладнання, а також дозволяє зменшити вартість тестування.

Сучасний підхід розробки ПЗ оснований на симуляціях і моделях покликаний подолати проблему необхідності мати дороговартісне обладнання на всіх етапах розробки та тестування. Окрім зменшення вартості безперечною перевагою такого підходу є можливість встановлювати необхідні умови в середовищі симулювання реального часу, навчати моделі застосовуючи методи машинного навчання, тестувати поведінку автомобіля в тій чи іншій потенційно небезпечній ситуації, не наражаючи на небезпеку реального водія або інших учасників дорожнього руху.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Дослідження предметної області та виділення основних проблем

Ідея застосовувати сценарії моделювання для тестування не є новою в автомобільній індустрії, але нові технології останнього десятиріччя, збільшення кількості та розміру даних, що надходять з сенсорів, встановлених на транспортному засобі, удосконалення апаратних засобів та нові виклики сьогодення потребують перегляду існуючих рішень стосовно можливості миттєвої обробки надвеликих даних, обчислення дерева можливих варіантів розвитку подій, прийняття рішень на основі отриманих результатів та програмного навчання транспортних засобів діяти в складних дорожніх умовах подібно до людської поведінки, але усуваючи такі негативні фактори, що є властивими для людини, як неуважність, погане самопочуття, втрата концентрації та ін.

Багаті та складні середовища моделювання дають інженерним командам контроль над генеруванням даних і, нарешті, навчають автономну систему транспортних засобів бути готовою до всіх сценаріїв, включаючи непередбачені та крайні випадки. Хоча вони є досить дороговартісними та не завжди відображають реальні обставини дорожнього середовища через використання прорахунків основаних на даних отриманих всередині такої системи. Є також інші системи, які є невеликими за об’ємом та складністю, вони найбільш абстраговані, використовують реальні дані автомобілів, але позбавлені важливих внутрішніх деталей середовища і можливостей встановлювати певні умови для навчання, що також не є оптимальним рішенням.

Об’єкт дослідження – програмне забезпечення для навчання та тестування самокерованих транспортних засобів.

Предмет дослідження – архітектура програмного забезпечення для навчання та тестування самокерованих транспортних засобів.

Аналіз предметної області дозволив виокремити наступні проблеми:

– висока вартість архітектурних рішень систем моделювання;

– труднощі щодо обробки надвеликих даних, що надходять з сенсорів;

– недостатні процесорні можливості обробки надвеликих даних;

– неточні дані через недосконаліть шляхів їх отримання та опрацювання;

– труднощі щодо зберігання надвеликих даних;

– недоліки застосування лише певних алгоритмів машинного навчання (необхідність комплексного застосування наявних алгоритмів);

– проблема так званого "катастрофічного забування", тобто втрати нейронними мережами знань, що були отримані раніше, у процесі імплементації нових навчальних сценаріїв водіння;

– труднощі доступу до новітніх технологій через високу комерціалізацію автомобільної індустрії;

– потреба в рішеннях з відкритим вихідним кодом задля прискорення розробок та допуску до них провідних вчених та університетів світу;

– проблема "нестійкості сенсорів" та "спаклюження даних", що надходять з них через світове засліплення, віддзеркалення, шуми, погані погодні умови та інші фактори реального оточуючого середовища;

– недостатня увага в системах моделювання на аналіз ризиків щодо прийняття миттєвих рішень, на відповідність стандарту ISO 26262 щодо функціональної безпеки просунутих систем водія;

– висока залежність результатів прийняття рішень від досвіду інженерів, а не від результатів отриманих в системах моделювання;

– зростаюча складність просунутих систем підтримки водія (ADAS) ускладнює проведення аналізу небезпек та оцінки ризиків (HARA).

Планується використання наступних методів.

Підхід, заснований на моделюванні, забезпечує автоматичний та систематизований метод для оцінки складної взаємодії об'єкта, що аналізується, з іншими функціями транспортного засобу в можливо складних експлуатаційних ситуаціях, що полегшує прогнозування небезпек.

Крім того планується застосовувати алгоритми штучного інтелекту, такі як навчання з підкріпленням – reinforcement learning.

Таким чином існує потреба у нових архітектурних рішеннях задля подолання наявних проблем, пов'язаних з високою вартістю системи, потребою в розробках з відкритим вихідним кодом, оптимальним комплексним використанням наявних алгоритмів машинного навчання для прийняття високоточних миттєвих рішень.

1.2 Огляд існуючих рішень

На сьогоднішній день існує багато готових програмних рішень, що охоплюють симулювання певної системи автомобіля, як наприклад симуляції антиблокувальної системи (АБС) гальмування, віртуальної системи розробки акумулятора, паливних елементів від компанії AVL.

Іншим прикладом є розробки компанії dSPACE, що надає користувачам різноманітні модулі симуляцій розробки сенсорів, як наприклад модулі камери, LIDAR (Light Detection and Ranging), радару, електричних компонентів. Також популярною є система ASM Diesel Engine, що є моделлю реального часу для дизельних двигунів. Ці модулі можна використовувати в продукті компанії Simulation Tool Suite. Такі окремі модулі дозволяють зосередити увагу на окремих компонентах системи в найбільш деталізованому вигляді.

Інші рішення спрямовані саме на змодельовані середовища для навчання автономного автомобіля. Наприклад компанія Unity розробляє платформу 3D-рендеринга в режимі реального часу, що використовується інженерними групами для ефективного створення середовищ моделювання для автономних тренувань на транспортних засобах, які багаті сенсорною та фізичною складністю, забезпечують вагомі когнітивні завдання та підтримують динамічну багатоагентну взаємодію [1]. Тобто розробники ПО для автомобілів можуть створювати середовища, що надають можливості впроваджувати алгоритми штучного інтелекту та забезпечувати умови функціонування подібні до реальних на основі методів ігрових симуляцій.

На рисунку 1.1 зображено схему основних компонентів автономної системи та можливостей Unity для створення навколишнього середовища для її функціонування [1].

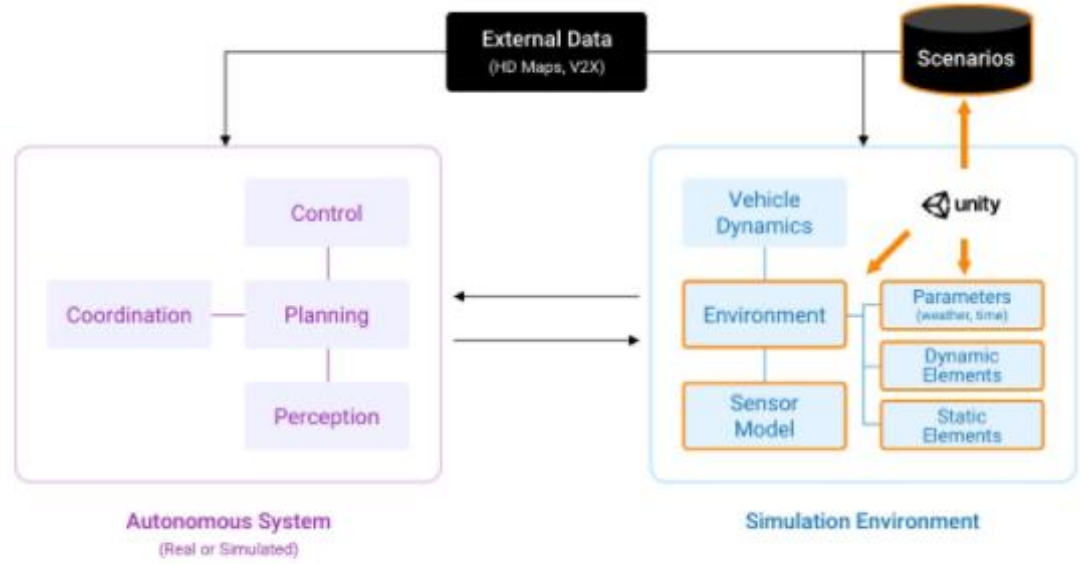


Рисунок 1.1 – Схема основних компонентів автономної системи та можливостей Unity

Також варто звернути увагу на рішення спрямовані на симуляції HMI (human machine interface). Такі симуляції дозволяють розробляти HMI автомобіля без необхідності мати панель приладів. Користувач може тестувати внутрішнє середовище автомобіля та реакції системи при виклику різних функцій ще до моменту впровадження механізмів в реальне авто. Наприклад, компанія Ford має систему підтримки маневрів заднього ходу з причіпним трейлером. Ця система дозволяє трейлеру повторювати рухи авто, коли треба виконати рух назад. Для того щоб водії могли навчитися користуватися цією системою, компанія пропонує зручне рішення у вигляді симуляції, що за наявності спеціального обладнання (окуляри віртуальної реальності) дозволяє натискати на кнопки сенсорів та проводити паркування авто разом з трейлером.

На сьогодні дуже актуальними є симуляції, що використовують алгоритми машинного навчання для тренування автономних систем. Вдалим прикладом таких

симуляцій є система VISTA (Virtual Image Synthesis and Transformation for Autonomy) розроблена дослідниками Массачусетського інституту технологій, що використовує навчання з підкріпленням та симуляції основані на реальних даних зовнішнього оточення (рис. 1.2). Вони збирають відеодані з камер автомобілів та на основі реальних траєкторій та оточення тренують свої моделі [2].

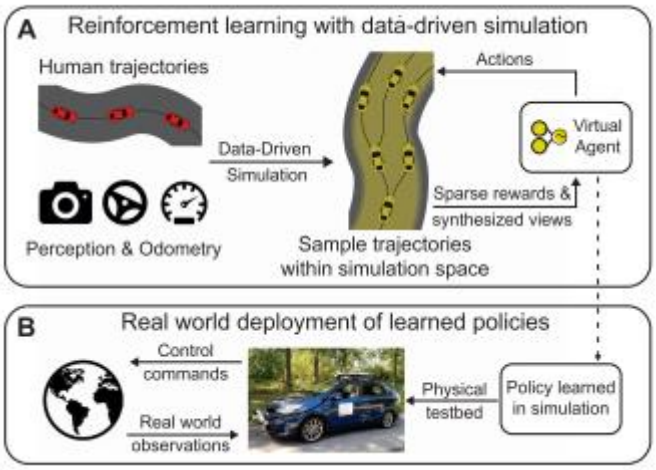


Рисунок 1.2 – Система VISTA

Розробники мали на меті усунути недоліки деяких симуляторів, що розробляють деталізоване середовище, але не мають реальних траєкторій руху та не дозволяють точно натренувати моделі, щоб вони могли реагувати в складних ситуаціях. Також їх підходом є обробка невеликого об’єму даних, але завдяки їх релевантності дослідники синтезують майже нескінченну кількість нових траєкторій конкретного реального середовища, кожна з іншим кутом обзору.

Цікавим підходом сучасних досліджень є використання біосенсорів та здобутків когнітивної психології для розроблення симуляцій. Наприклад, дослідники Мічіганського університета [3] підійшли до проблеми з точки зору когнітивних реакцій водія на можливі ситуації під час керування автомобілем в штучно симульованій сесії автономного руху. Вчені досліджували рухи очей, електричну активність шкіри та серцевий ритм, щоб оцінити психологічний стан водія. Вони поставили перед собою завдання на основі отриманих результатів передбачити, скільки знадобиться часу на те, щоб в самокерованому авто рівня SAE 3 (Society of Automotive Engineers) водій зміг в моменти складнощів з

автопілотом перемкнутися на повне самостійне керування. Піддослідні мали на собі сенсори, що дозволяли зчитувати необхідну інформацію та передавати її в симуляцію реального часу iMotions для синхронізації. Було використано метрики AOI (Area of interest) – показники відстеження очей – що дозволили відстежувати, коли очі водія слідкували за дорогою, шаблони сканування, частоту моргання та ін. Результати передбачення поведінки водія були на 70% точними.

Вчені з університету Меріленд у журналі Science Robotics у 2019 році презентували систему симуляції, що використовує фото, відео, траєкторії реального водіння та поведінкові дані [4]. Вони запевнили, що їх симуляція є більш актуальною для навчання та тестування самокерованих транспортних засобів, аніж симуляції, що використовують ігрові движки та математично вирахувані шаблони руху. Їх система має назву Augmented Autonomous Driving Simulation (AADS) (рис. 1.3). Вони запропонували використовувати лідар та камери для сканування вуличних сцен. Після цього отримані дані було декомпозовано до окремих об’єктів. Вони використали техніку синтезу оточення для відображення об’єктів на статичному тлі [4].

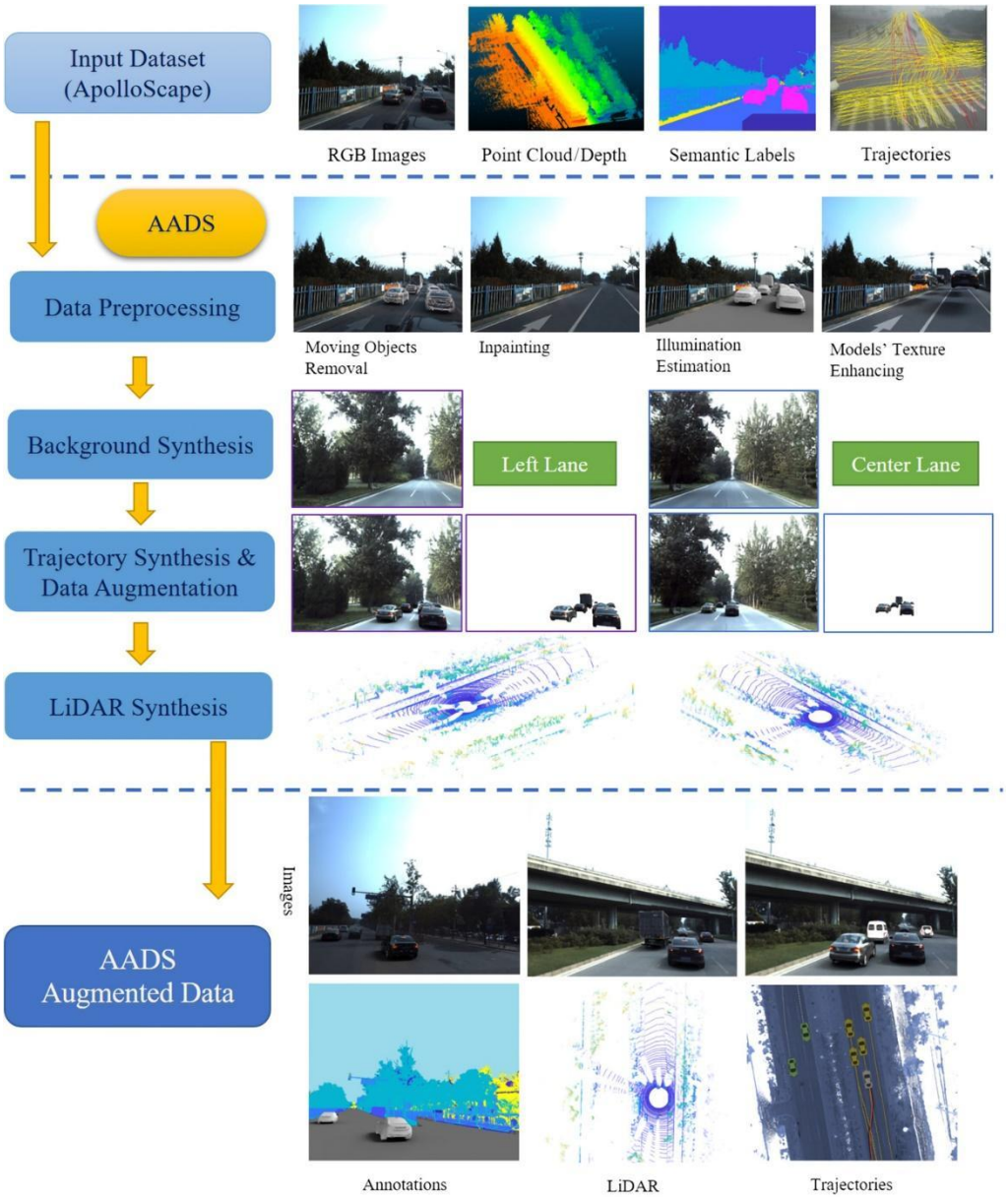


Рисунок 1.3 – Схема функціонування AADS

Відомими симуляторами також є NVIDIA Drive Constellation AV [7] та Google/Waymo's CarCraft [8, 9].

NVIDIA Drive Constellation AV – це симулятор безпілотних авто у віртуальній реальності, що генерує дані з сенсорів віртуального транспортного засобу та відправляє ці дані на цільове обладнання на іншому сервері, який в свою чергу обробляє дані та направляє відповідь на перший сервер (рис. 1.4) [10].

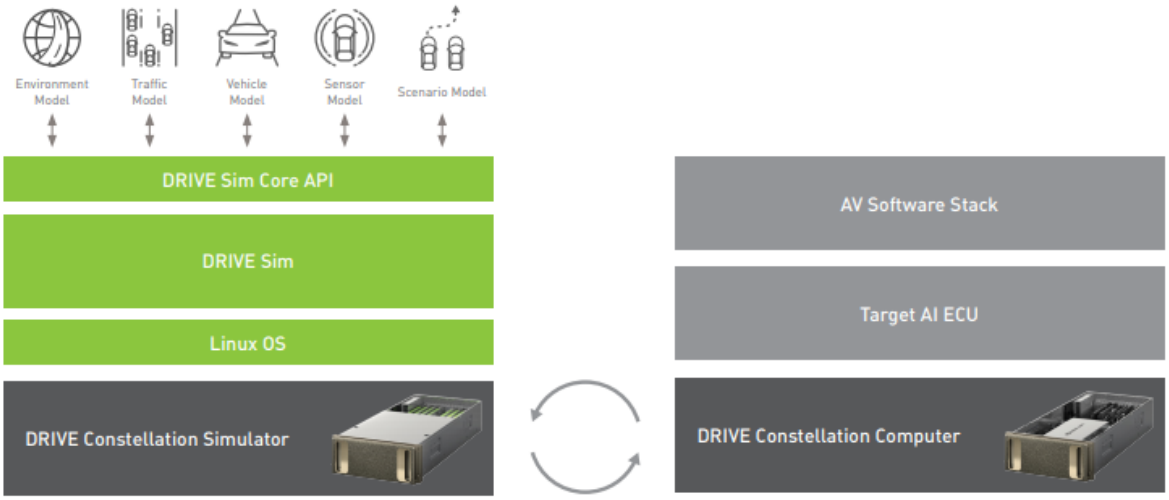


Рисунок 1.4 – Схема роботи симулятора NVIDIA Drive Constellation AV

Google/Waymo's CarCraft – це середовище для тренування, тестування та верифікації автономних систем, що відтворює реальне середовище таких міст, як Маунтін-В'ю, Каліфорнія та Остін, штат Техас у віртуальних реконструкціях. Ця система потребує багато обчислювальних ресурсів, тому у 2020 році компанія почала використовувати систему SurfelGAN, що пропонує більш простий, підхід керований даними для моделювання даних з датчиків. Спираючись на джерела даних із реальних лідарних датчиків та камер, ШІ створює та зберігає інформацію про тривимірну геометрію, семантику та зовнішній вигляд усіх об'єктів на сцені. На основі отриманих результатів SurfelGAN робить змодельовану сцену з різних відстаней та кутів огляду (рис. 1.5) [11].

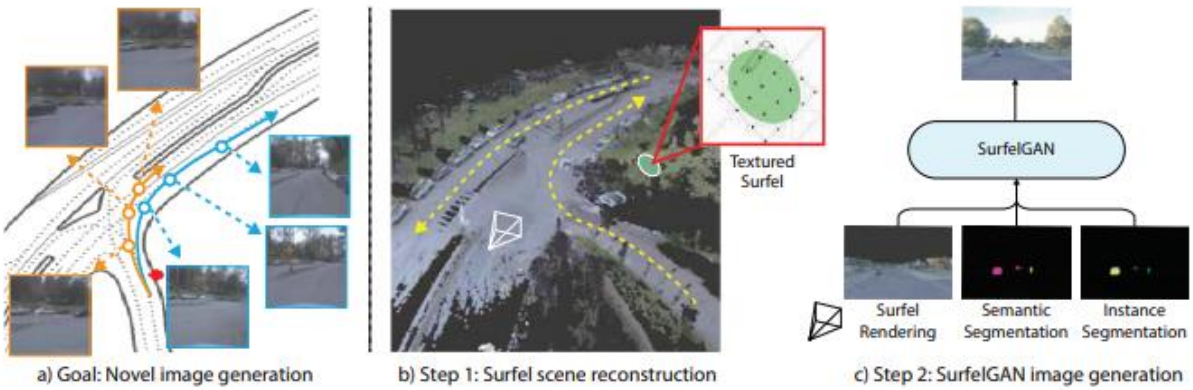


Рисунок 1.5 – Система SurfelGAN

Серед найбільш відомих симуляторів з відкритим вихідним кодом є симулятор Intel CARLA [5] та Microsoft AirSim [6]. Через те, що нас більш за все цікавить ПЗ з відкритим вихідним кодом, розглянемо їх детальніше, а також виділимо їх переваги та недоліки.

1.2.1 Середовище для навчання CARLA

CARLA це симулятор з відкритим вихідним кодом розроблений у кооперації Intel Labs, Toyota Research Institute та Barcelona Computer Vision Center для досліджень автопілотів [5].

CARLA була розроблена з нуля для підтримки розвитку, навчання та перевірки систем автономного водіння. На додаток до коду та протоколів з відкритим кодом, CARLA пропонує відкриті цифрові активи (міські плани, будівлі, транспортні засоби), які були створені для цієї мети та можуть вільно використовуватися.

Платформа моделювання підтримує гнучку специфікацію сенсорних наборів, умов навколишнього середовища, повний контроль над усіма статичними та динамічними акторами, генерацію карт та багато іншого [5]. Реалістичне графічне відображення досягається завдяки платформі Unreal Engine.

CARLA побудований на основі клієнт-серверного підходу, який дозволяє розділити навантаження на систему і забезпечити виконання кожним агентом своїх функцій. Такий підхід найбільш оптимальний для програмного забезпечення, що потребує виділення певних абстрактних прошарків для підвищення взаємної сумісності.

Перевагами такого підходу зазвичай є висока централізація основних обчислювальних потужностей, значні можливості забезпечення конфіденційності збереження даних, підвищення продуктивності, гнучкості та можливостей розширення в довгостроковій перспективі. Крім того клієнт-серверна архітектура характеризується гарними інтеграційними показниками та модульністю.

Незважаючи на істотні переваги, такий підхід має певні недоліки. По-перше, у випадку відмови серверної частини системи вся система опиниться у стані

зупинки функціонування. Крім того серверне обладнання є достатньо дороговартісним. Також даний тип архітектури може бути дуже вразливим до DOS атак через можливість надмірної кількості запитів на сервер.

У середовищі CARLA серверна сторона відповідальна за рендеринг сцени, а також за інтеграцію агентів. Запити приходять від клієнта, що реалізований за допомогою мови програмування Python та використовує сокети. У відповідь на запити сервер повертає важливі показники, такі як значення, що надходять з сенсорів, розташування досліджуваного автотранспорту та ін. Серверна частина написана за допомогою мови програмування C++ та деяких фреймворків. На рисунку 1.6 зображено концептуальну схему симулятора [12].

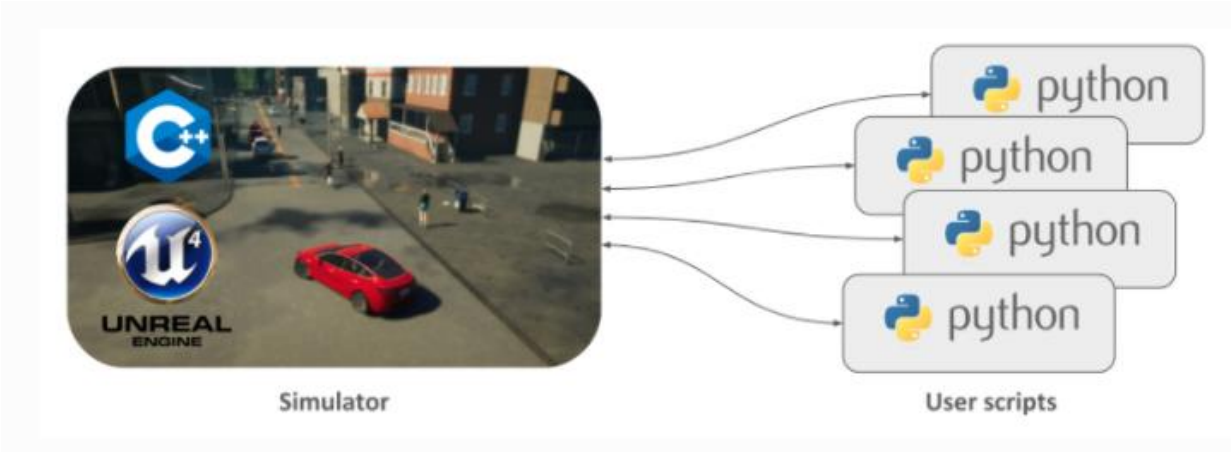


Рисунок 1.6 – Концептуальна схема симулятора CARLA

Крім того для відображення оточуючого світу, як вже було зауважено, використовується ігрова платформа Unreal Engine 4. Вона надає можливості всебічного відображення фізичного світу, реалістичні сценарії, високу якість рендерінгу (рис. 1.7) [13]. Отже значною перевагою даного симулятора є можливість тестувати та навчати самокеровані автомобілі при різних рівнях освітлення, погодних умовах, з різними агентами.

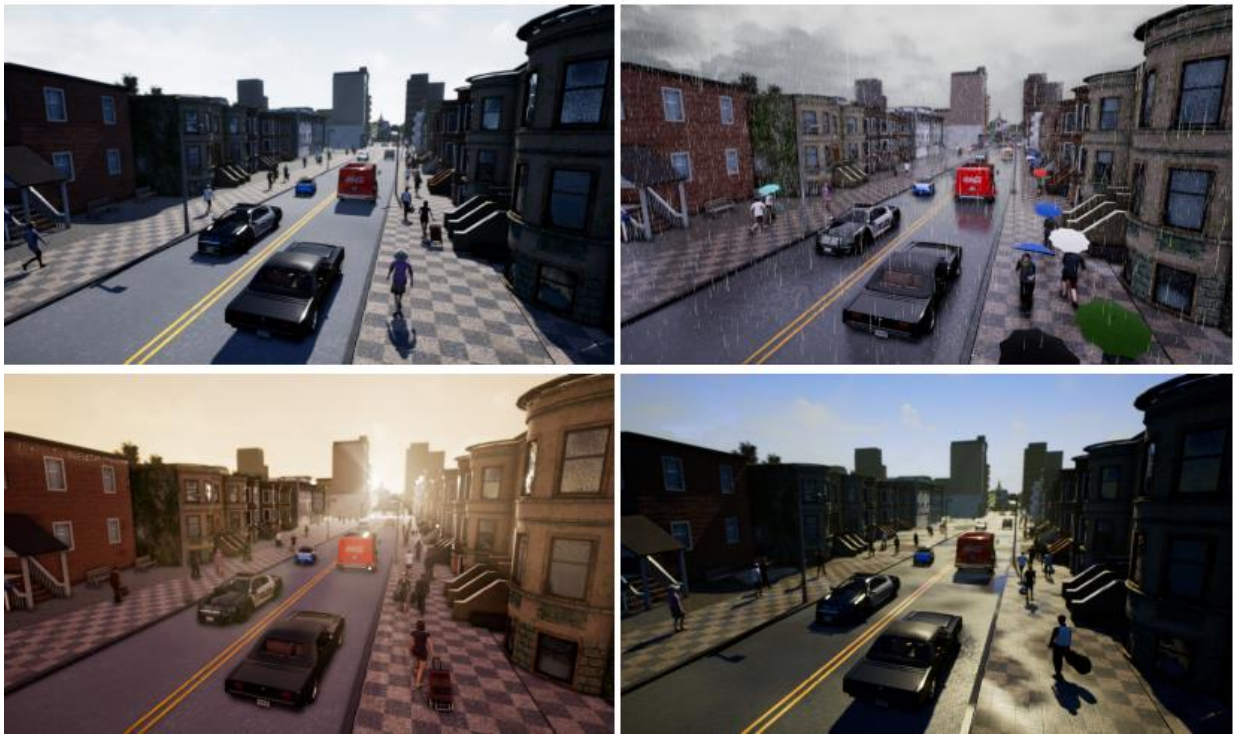


Рисунок 1.7 – Графічні можливості симулятора CARLA

До базових можливостей симулятора відносяться представлені нижче.

– Менеджер руху – вбудована система, що контролює транспортні засоби з метою досягти більш реалістичної поведінки.

– Сенсори. Автомобілі завдяки сенсорам можуть отримувати інформацію з оточуючого середовища. В даний час симулятор підтримує такі види сенсорів, як камери, радари, лідари та ін.

– Рекордер. Ця функція дозволяє отримати інформацію про кожного агента системи в любий проміжок часу.

– Міст ROS та реалізація Autoware – дозволяють інтегрувати симулятор з іншими середовищами навчання.

Щодо можливостей обчислень та відповідності правилам дорожнього руху, то у середовищі можна отримати дані щодо:

– локації та орієнтації транспорту відносно систем координат, таких як GPS та компас;

– швидкості агентів;

– вектору прискорення;

– кількісних показників після зіткнення;

– відсотку автомобіля, що може вийти за межі, дозволені ПДД;

– перевищення швидкості тестованого автомобіля відповідно до встановлених меж та ін.

Істотним недоліком симулятора є те, що він не передбачає тестування та навчання автомобілів поза межами міської місцевості.

Системні вимоги:

– 130 ГБ дискового простору (CARLA потребує близько 31 ГБ, а Unreal Engine - близько 91 ГБ) для Linux, для Windows - 165 GB;

– сервер потребує принаймні 6 ГБ пам’яті графічного процесора, хоча рекомендується 8 ГБ;

– два порти TCP і хороше підключення до Інтернету (2000 та 2001 за замовчуванням) [12].

Симулятор має підтримку для Windows та Linux. Недоліком є те, що поки що відсутня підтримка для Mac.

1.2.2 Середовище для навчання AirSim

AirSim використовує методи штучного інтелекту для систем реального часу на основі платформ Unreal Engine та Unity. Це кросплатформений симулятор з відкритим вихідним кодом, що використовується багатьма науковцями для навчання автономних систем.

Вони використовують методи глибинного, імітаційного навчання та навчання з підкріпленням. Використання таких засобів як Microsoft Project Bonsai, що є платформою ШІ, пришвидшує розробку автоматизованих процесів на основі ШІ та є частиною набору автономних систем від Microsoft, дозволяє навчати моделі в різних погодних умовах та сценаріях в хмарному середовищі Microsoft Azure.

Щодо архітектури, то AirSim спирається на модульний дизайн з упором на високі можливості розширення.

Ключові функціональні блоки архітектури:

– модель середовища;

– транспортна модель;

– фізичний движок;

– моделі сенсорів;

– інтерфейс рендерингу;

– публічний API прошарок;

– прошарок інтерфейсу для апаратного забезпечення транспортного засобу (рис. 1.8) [14].

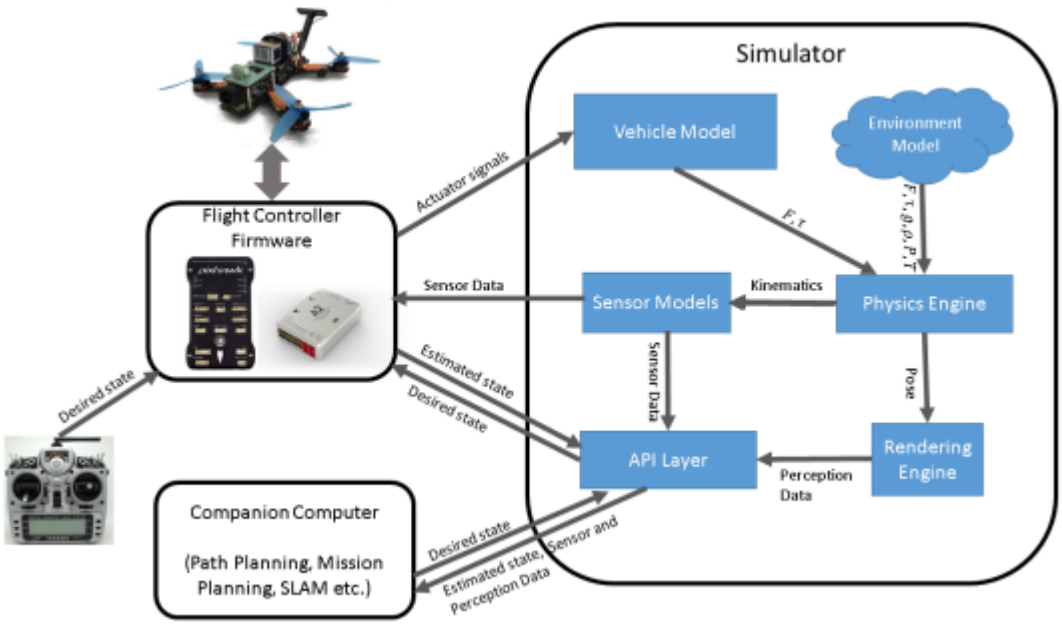


Рисунок 1.8 – Архітектура системи AirSim

Симулятор розроблений як плагін Unreal, який можна просто помістити в будь-яке середовище Unreal. Наразі з’явився також експериментальний реліз для плагіна Unity. Графічні можливості симулятора AirSim представлено на рисунку 1.9 [15].

Рисунок 1.9 – Графічні можливості симулятора AirSim

Аналіз структури вихідного коду на GitHub показав, що більша частина вищезазначеного програмного забезпечення використовує мову програмування C++. Також деякі частини написані з використанням C# та Python.

Основою проекту є бібліотека AirLib, яка компілюється за допомогою C++11 компілятора.

AirLib складається з нижченаведених компонентів.

– Фізичний движок – код, що містить лише заголовочні файли. Він розроблений таким чином, що його легко підтримувати та розширювати за потреби. Використовується для імплементації транспортних засобів.

– Моделі датчиків – це заголовочні файли для моделей барометра, IMU, GPS та магнітометра.

– Моделі транспортних засобів – це заголовочні файли для конфігурацій та моделей автомобілів.

– Файли, пов’язані з API – ця частина AirLib містить абстрактний базовий клас для всіх API та конкретну реалізацію для конкретних платформ транспортних засобів, таких як MavLink. У ньому також є класи для клієнта та сервера RPC [15].

Безперечними перевагами симулятора AirSim є висока модульність та здатність до розширення. Це досягнуто тим, що лише частина проекту залежить від Unreal Engine, вона залишається ізольованою для того, щоб можна було реалізувати симуляцію також для інших платформ, як це було зроблено для Unity.

Окрім того висока документованість та наглядність, дозволяє легко розпочати роботу з симулятором. Наприклад, на рисунку 1.10 наглядно продемонстровано, як AirSim завантажується та викликається Unreal Game Engine [15].

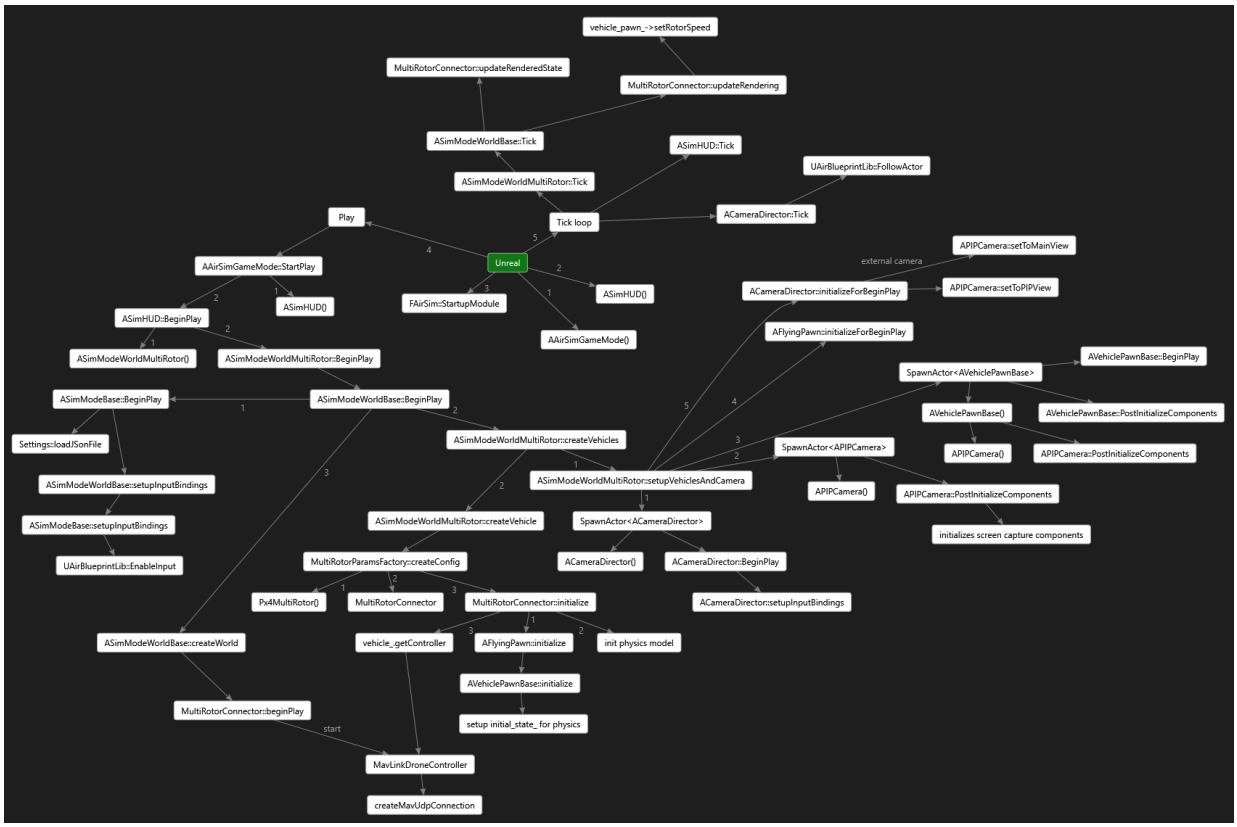


Рисунок 1.10 – Класи Unreal Game Engine симулятора AirSim

Симулятор висуває такі технічні вимоги та рекомендації до системи та обладнання:

– GPU – мінімально 4GB RAM (краще 8 GB), NVidia 1080 чи NVidia Titan;

– ядра - 6+;

– SSD;

– 2-3 монитора (4K);

– оперативна пам’ять - 64GB RAM [15].

Як ми бачимо це достатньо дороге обладнання, що є великим мінусом даного ПЗ.

Використання таких відомих графічних платформ як Unreal та Unity дозволяє досягти високої реалістичності та відчуття присутності, а можливості застосування різних алгоритмів машинного навчання в поєднанні з симулятором робить таке програмне забезпечення безперечно ефективним в навчанні автономних приладів.

На жаль, AirSim має також певні недоліки, які є істотними для тестування та навчання самокерованих транспортних засобів. Від самого початку симулятор розроблявся для тестування літальних приладів (дронів), а тому підтримка наземного транспорту все ще дещо обмежена. Не реалізована підтримка 2D та 3D маркування навколо транспортного засобу, що значно обмежує можливості розпізнавання динамічних об’єктів. Крім того набір карт більше підходить для літальних апаратів та включає гірські пейзажі, лісу.

Окрім того через те, що симулятор реалізовано як плагін до графічної платформи, його функціонування без середовища неможливе. Тобто спочатку потрібно скомпілювати плагін, додати його до проекту Unreal, а вже потім задеплоїти та запустити проект.

Суттєвим недоліком є те, що взаємодія з агентами є не повною, наприклад, взаємодія з пасажирами взагалі не реалізована. Автомобіль може збити пішохода і це ніяким чином не буде опрацьовано, що є великим порушенням питання безпеки. Тобто немає ніякої можливості зараз протестувати рух автомобіля у розрізі функціональної безпеки згідно стандарту ISO 26262.

Крім того, не реалізовано вибір рівня автономності автомобіля, який тестується, що також впливає на кінцеві результати дослідження.

Недоліком також є те, що у програмному забезпеченні не реалізовано такий тип сенсору, як радар, що також не дозволяє побачити цілісної картини реального середовища.

Як і симулятор CARLA, AirSim потребує дороговартісне потужне обладнання для повноцінних можливостей роботи з ними.

Порівняно з симулятором CARLA, окрім підтримки для Windows і Linux, AirSim також надає підтримку для macOS.

Головним істотним недоліком є те що хоча перелічене програмне забезпечення і дозволяє навчати самокеровані транспортні засоби, але це лише середовище для навчання, а компонент програмного забезпечення, яке дозволяло би навчити моделі не поставляється разом з цими середовищами.

1.3 Постановка задачі

Основною метою роботи є підвищення ефективності навчання та тестування самокерованих транспортних засобів в програмованому середовищі на основі модельно-орієнтованого проектування. Для досягнення мети необхідно вирішити наступні задачі:

- проаналізувати досліджувану предметну область та виокремити основні проблеми;
- проаналізувати існуючі рішення в досліджуваній області;
- протестувати open-source програмне забезпечення та виокремити його недоліки та переваги щодо необхідних критичних показників;
- проаналізувати існуючі теоретичні відомості щодо можливостей отримання та обробки даних та представлення кінцевих результатів для забезпечення прийняття рішень;
- на основі проведеного аналізу розглянути необхідність проведення HARA (Hazard Analysis and Risk Assessment);
- проаналізувати можливості алгоритмів машинного навчання для підвищення ефективності навчання та тестування самокерованих транспортних засобів та запропонувати спосіб його впровадження;
- запропонувати архітектуру програмного забезпечення для найбільш ефективного навчання та тестування автономних транспортних засобів;

– дослідити результати навчання самокерованих транспортних засобів для запропонованої архітектури;

– обґрунтувати ефективність запропонованого рішення;

– узагальнити результати досліджень.

Створений продукт повинен відповідати наступним вимогам:

– простота використання;

– полегшена інтеграція компонентів системи з елементами подібних систем;

– відкритість вихідного коду;

– використання новітніх галузевих технологій;

– можливість навчати моделі транспортних засобів за допомогою вбудованої підтримки засобів машинного навчання;

– легкість для підтримки та гнучкість до розширення.

Висновки по розділу

Таким чином всебічне дослідження предметної області та аналіз існуючих рішень показали, що незважаючи на те що сучасне програмне забезпечення для навчання та тестування автономних транспортних засобів є надзвичайно комплексним, всебічним та охоплює різні компоненти досліджуваної системи, але ж воно має також і недоліки, серед яких я вважаю найбільш критичними є:

– потребує великих обчислювальних можливостей, дорогої апаратної частини для можливості його використання;

– низька зручність використання та встановлення, велика кількість залежностей від інших технологічних блоків;

– відсутня вбудована підтримка можливості навчання методами машинного навчання (потрібно самостійно програмувати нейронну мережу);

– неможливість існування як окремого додатку, який можна передавати у вигляді виконуваного файлу;

– висока деталізація елементів середовища знижує ефективність навчання через необхідність направляти велику частину обчислювальних ресурсів саме на рендеринг сцени, а не на процес навчання;

– клієнт – серверний підхід, вважаю, є не найбільш оптимальним до побудови такого програмного забезпечення, через те, що для даного завдання зазвичай клієнт та сервер розміщуються на одній машині і це лише ускладнює процес навчання та користувацької взаємодії.

Таким чином існує необхідність проведення дослідження задля усунення перелічених проблем та розширення функціоналу архітектури ПЗ для навчання та тестування самокерованих транспортних засобів, яке завдяки поєднанню наявних засобів моделювання та бібліотек, що надають можливість використання алгоритмів машинного навчання, дозволило б покращити досвід користувацької взаємодії та за рахунок чого надало б можливість найбільш ефективно навчати та тестувати автопілоти.

2 АНАЛІЗ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Розробка програмного забезпечення заснована на підході модельно-орієнтованого проектування

Автомобільна індустрія є однією з найбільш дороговартісних з точки зору вартості обладнання та програмного забезпечення. Тому найбільш актуальними питаннями для розробників програмного забезпечення є зниження вартості розробки, підвищення швидкості та забезпечення найбільш ефективного методу усунення помилок на найбільш ранніх, а тобто дешевих етапах виробництва.

Модельно-орієнтований підхід (МОП) до проектування покликаний допомогти подолати вищеназвані проблеми та забезпечити найбільш оптимальний цикл розробки продукції в таких галузях, як автомобільна, аерокосмічна, виробництво промислового обладнання та ін.

Основними компонентами МОП є:

– виконувані специфікації отримані з моделей;

– побудова архітектури ПЗ в середовищі симуляції;

– постійне тестування та верифікація;

– імплементація за допомогою автоматичної генерації коду (рис. 2.1) [17].

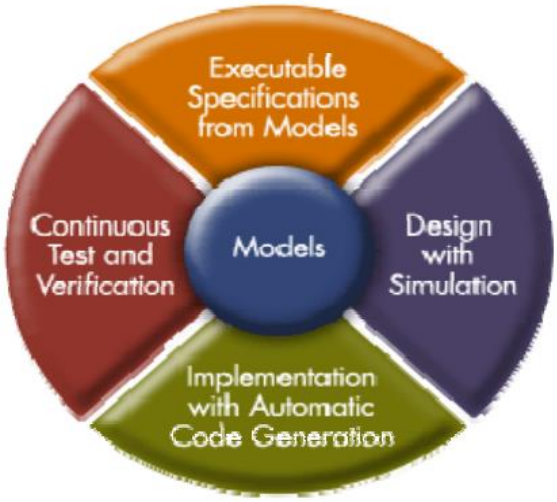


Рисунок 2.1 – Складові компоненти модельно-орієнтованого проектування

МОП використовує математичні методи для складних обчислень, а також метод візуалізації, завдяки якому складні статичні схеми, текстові технічні вимоги та діаграми перетворюються на наглядну модель, яка в свою чергу може бути за допомогою автоматизованих засобів перетворена в вихідний код. Тобто такі моделі називаються виконуваними.

Це забезпечує, по-перше, високу спроможність щодо уникнення помилок в вручну написаному коді, по-друге, дозволяє ще на етапі віртуальної побудови прототипу протестувати його поведінку та виправити помилки, які на більш пізніх етапах можуть бути занадто дороговартісними.

Крім того модель може бути розроблена як для частини системи, так і для всієї системи в цілому. І завдяки валідації моделі ще до початку її впровадження в виробництво можна прослідкувати взаємодію кожного елемента системи. Це зменшує можливі несприятливі результати.

Безперечною перевагою МОП є те, що одного разу розробивши та протестувавши модель, її можна використовувати для оптимізації існуючих продуктів, не вдаючись до вже відпрацьованих деталей реалізації.

МОП визначає структуру взаємодії основних компонентів системи та реалізує V-подібний цикл розробки, що спрямований на оптимізацію процесу розробки складних, частіше за все вбудованих систем (прикладом яких є автономний транспортний засіб) та покликаний зменшити складність розуміння етапів циклу та підвищити його ефективність. Завдяки використанню МОП досягаються більші можливості інтеграції компонентів системи та спостерігається уніфікація процедури розробки ПЗ, апаратного забезпечення та людино-машинних інтерфейсів (HMI).

На рисунку 2.2 зображено V-Model процесу розробки інформаційних систем [16].

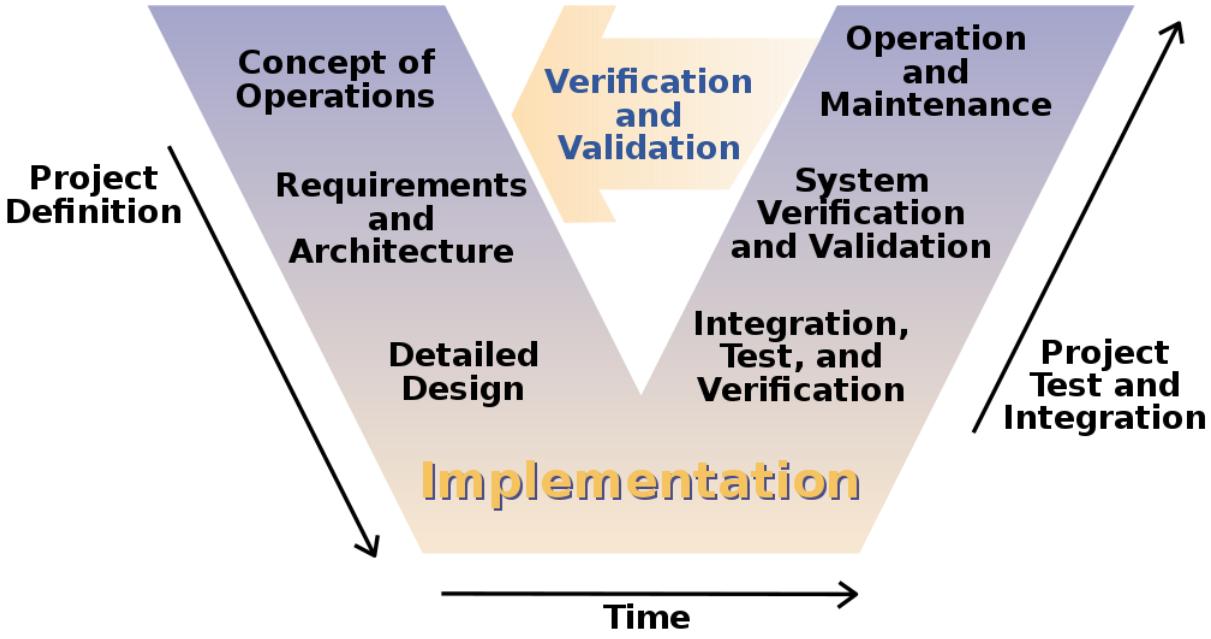


Рисунок 2.2 – V-Model процесу розробки ІС

На схемі зображено фази згідно з якими ведеться розробка ПЗ у V-Model. Читати модель потрібно зліва направо, значна увага тут приділяється питанням тестування та верифікації.

Як правило таку модель легко адаптувати до існуючого продукту, вона допомагає всім членам команди, що працює над продуктом, залишатися в курсі ходу подій, чіткі рекомендації дозволяють на кожному етапі тримати інформацію в релевантній та актуальній формі і одразу реагувати на зміни. Недоліком моделі є недостатній аналіз ризиків.

Для реалізації МОП після аналізу існуючих програмних рішень було виділено найбільш оптимальне середовище розробки моделей – MATLAB/Simulink. Це програмне забезпечення, що поставляється компанією MathWorks та сприяє можливостям візуалізації, тестування та аналізу ПЗ.

MATLAB – це обчислювальна платформа, яка використовується мільйонами інженерів та вчених для аналізу даних, розробки алгоритмів та створення моделей. Це десктопне ПЗ, органічно поєднане для ітераційного аналізу та процесів проектування з мовою програмування, яка безпосередньо працює з математичними сутностями. Воно містить редактор для створення сценаріїв, які

поєднують код, вихідні дані та форматований текст у виконуваному середовищі. MATLAB може використовуватися з такими мовами програмування як Python, C/C++, Fortran, Java та ін. [18].

В сучасній автомобільній індустрії в напрямку самокерованих авто дуже активно використовуються MATLAB, Simulink та RoadRunner для того, щоб:

- візуалізувати та маркувати дані, що надходять з сенсорів;
- моделювати сценарії пересування транспортних засобів;
- планувати алгоритми контролерів та сприйняття оточуючого середовища;
- деплоїти алгоритми за допомогою автоматичної генерації коду;
- інтегрувати та тестувати розроблювану систему [38].

Наразі MATLAB/Simulink допомагає великим виробникам програмного забезпечення для автомобільної індустрії у питаннях локалізації транспортного засобу, планування пересування, прийняття рішень та багато інших важливих аспектів.

Отже дослідження показало, що використання MATLAB/Simulink в розроблюваній архітектурі дозволить провадити всі переваги модельно-орієнтованого підходу на практиці.

2.2 Дослідження рівнів автоматизації автономних транспортних засобів за класифікацією SAE

Важливим аспектом розробки програмного забезпечення для навчання та тестування самокерованих транспортних засобів є повне розуміння, для якого рівня автономності транспортного засобу буде використовуватися ПЗ. Отже необхідним є дослідження можливих рівнів автономності та відповідно до цього встановлення вимог до розроблюваної архітектури.

Визначенням критеріїв відношення системи до того чи іншого рівня займається організація SAE (Society of Automotive Engineers). Вона розробляє та постійно оновлює систему класифікації, за якою лише транспортні засоби з відповідним забезпеченням можуть бути віднесені до автономних. Взагалі розуміння всіх особливостей того чи іншого рівня є питанням особливої важливості, так як це насамперед стосується безпеки.

Через неповне розуміння особливостей транспортного засобу, водії часто думають, що автономний транспортний засіб – це засіб, який дозволяє не приймати участі в процесі пересування, не слідкувати за дорогою, не приймати важливі миттєві рішення. Таким чином таке непорозуміння стає проблемою, як для конкретного водія, так і для галузі в цілому, тому що постають питання безпечності автономних транспортних засобів та взагалі їх заборони. Але такі критичні ситуації насправді зумовлені не небезпечністю транспорту, а лише не належною увагою до рівня автономності конкретного автомобіля.

Тому SAE дуже прискіпливо розробляє визначення автономних систем та критеріїв їх автономності. На сьогодні існує 6 рівнів автономності від 0 до 5, від транспортного засобу, який позбавлено взагалі функціональних можливостей самокерування, до максимального рівня автономності, коли водій взагалі не потрібен.

Важливим у розрізі дослідження рівня автономності є поняття «динамічна функція водіння» (dynamic driving task – DDT). Згідно з визначенням SAE, – це «усі оперативно-тактичні функції в режимі реального часу, необхідні для експлуатації транспортного засобу під час дорожнього руху, за винятком стратегічних функцій, таких як планування поїздок та вибір пунктів призначення та маршрутних точок». Крім того в класифікації зустрічається поняття ODD (operational design domain) – це «умови експлуатації, за яких певна система автоматизації водіння або її частини спеціально розроблені для функціонування в умовах екологічних, географічних обмежень та обмежень часу доби, та/або необхідну наявність чи відсутність певного руху або характеристик проїжджої частини» [19].

На рисунку 2.3 зображено всі рівні автономності транспортного засобу відповідно до класифікації SAE [20].

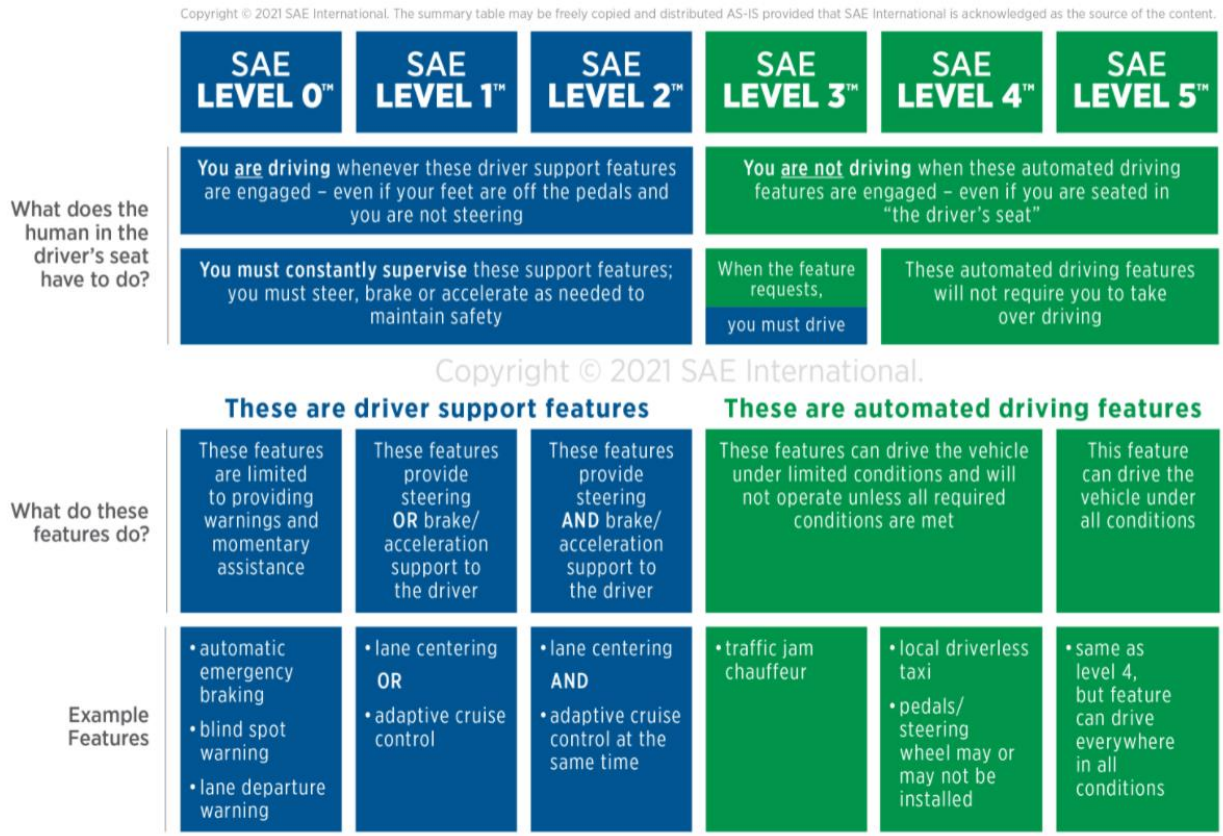


Рисунок 2.3 – Рівні автономності транспортного засобу відповідно до класифікації SAE

Рівні від 0 до 2 вимагають від водія активної залученості до процесу водіння, рівень 3 вимагає залученості водія лише у випадку, коли система цього вимагає, та рівні 4-5 не вимагають залученості водія. Лише п'ятий рівень означає повну автоматизацію водіння в будь-яких умовах та на будь-яких видах доріг.

Проаналізуємо кожний рівень.

– **Автоматизація водіння відсутня.** Виконання водієм повного DDT, навіть якщо транспортний засіб посилено системами активної безпеки.

– **Допомога водію.** Постійне та специфічне для ODD виконання системою автоматизації водіння підзадачі DDT щодо управління поперечним або поздовжнім рухом транспортного засобу (але не обох одночасно) з очікуванням, що водій виконує решту DDT.

– **Часткова автоматизація водіння.** Постійне та специфічне для ODD виконання системою автоматизації водіння підзадач DDT з боковим та поздовжнім контролем руху автомобіля з очікуванням, що водій виконує одну з підзадач DDT (моніторинг оточуючого середовища та виконання відповідної реакції на такі об'єкти та події) та контролює систему автоматизації водіння.

– **Умовна автоматизація водіння.** Система автоматизації здатна виконувати всі функції керування транспортним засобом, але водій повинен бути готовим негайно взяти управління у випадку ризикових ситуацій, коли система вимагає втручання.

– **Високий рівень автоматизації водіння.** Повна автоматизація управління в деяких умовах (ODD специфічних).

– **Повна автоматизація водіння.** Повна автоматизація управління в будь-яких умовах.

Таким чином розробка програмного забезпечення для навчання та тестування самокерованих транспортних засобів повинна проводитися з урахуванням рівня автономності і базуватися на показниках прийнятних для певного рівня.

2.3 Аналіз вимог до розробки та архітектури програмного забезпечення в автомобільній індустрії згідно стандарту функціональної безпеки ISO 26262

Питання безпеки в автомобільній індустрії є що не найважливішим для розробників програмного забезпечення. Порівнюючи з іншими типами програмного забезпечення, ПЗ для автомобілів повинно підпорядковуватися ряду суворих правил, що встановлюються державами, міжнародними організаціями,

стандартами та нормами. Така суворість не дивна, тому що ціною помилки в програмі може стати життя.

Тому середовища моделювання для навчання та тестування самокерованих транспортних засобів повинні мати можливість встановлювати прийнятні норми для тієї чи іншої країни, щоб можна було проводити дослідження в максимально наближених до реальності обставинах.

Одним з найбільш авторитетних та актуальних стандартів сьогодення в автомобільній промисловості є стандарт функціональної безпеки ISO 26262, адаптований з IEC 61508, що є стандартом функціональної безпеки для електронних безпеко-орієнтованих систем. Він включає в себе вимоги до розробки програмного забезпечення та архітектури.

Стандарт передбачає вказівні рекомендації щодо зменшення можливих ризиків виникнення помилок та містить вимоги, щодо необхідності врахування цих рекомендацій на кожному етапі розробки архітектури ПЗ, а також імплементації та валідації системи. Він покликаний усунути будь -який неприйнятний ризик для життя людини.

Важливим етапом в розробці ПЗ згідно стандарту повинен бути етап аналізу небезпеки та оцінки ризиків (HARA – Hazard Analysis and Risk Assessment). На рисунку 2.4 можна побачити належне місце даного етапу в життєвому циклу розробки продукту [21].

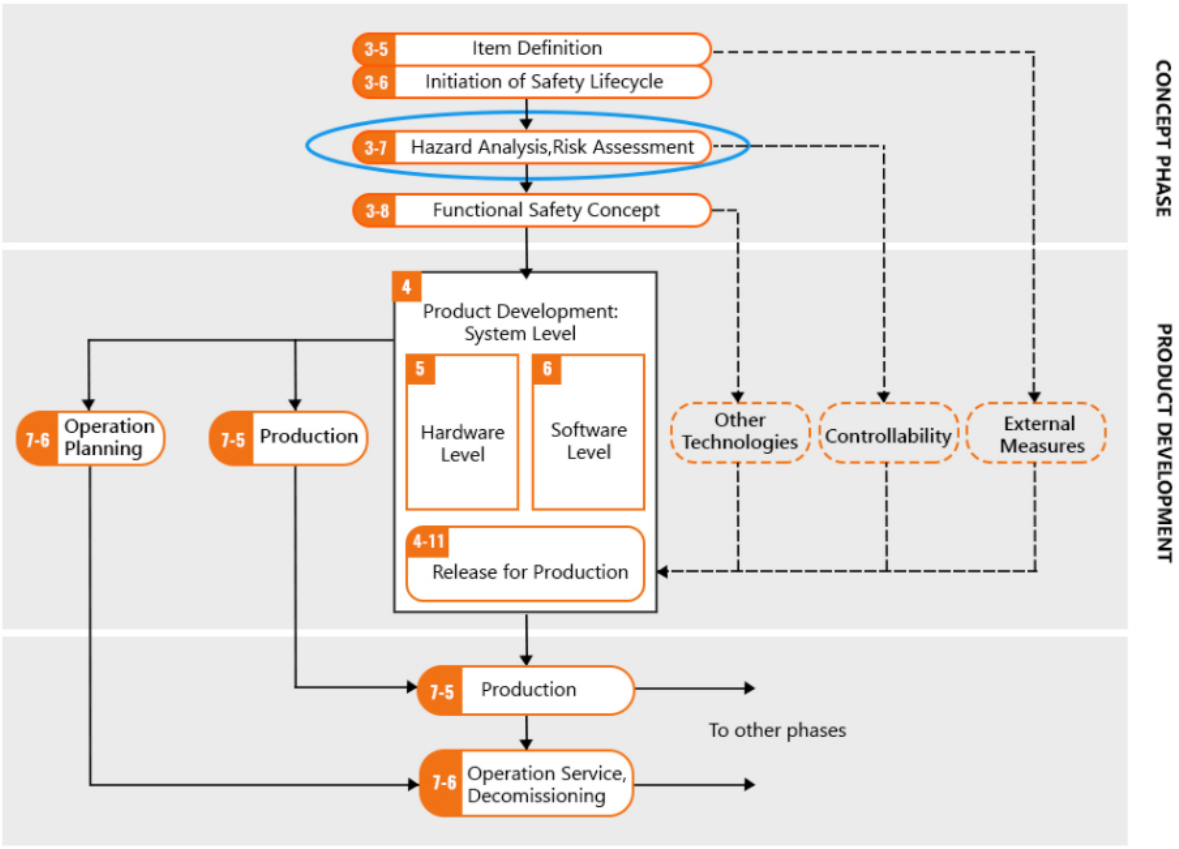


Рисунок 2.4 – Місце HARA в життєвому циклі розробки продукту

Можна побачити, що це важливий підготовчий етап, який називається умовно концептуальною фазою розробки, коли закладаються вся важливі вимоги до продукту та оцінюються ризики. На основі цих даних створюється концепція функціональної безпеки, яка є базою у подальших кроках розробки ПЗ.

Виявлення несправностей найкраще проводити за допомогою HAZOP (hazard and operability analysis). Це дослідний аналіз, який враховує відхилення системи або її операційних намірів. Теорія HAZOP передбачає, що будь -яка потенційна небезпека буде виникати щоразу, коли є відхилення від передбачуваної роботи системи [21].

Для ідентифікації відхилень застосовуються спеціальні слова, які потім використовуються в операційних сценаріях, наприклад, пришвидшення, гальмування та ін. Окрім того ідентифікуються операційні режими (такі як «транспортний засіб припарковано», «транспортний засіб рухається»).

Після ідентифікації загроз, наступає фаза їх класифікації. Для класифікації ризиків застосовується система ASIL (automotive safety integrity levels). Вона виділяє 4 рівня (A - D). A – найнижчий рівень ризику, D – найвищий. ASIL визначає, які методи команди розробників ПЗ повинні використовувати [22].

Подальшим кроком є виокремлення цілей безпеки (safety goals), які є вимогами найвищого рівня для кожного елемента системи (або всієї системи в комплексі).

Таким чином процес визначення цілей безпеки можна узагальнити у такі кроки:

– визначення всіх відповідних небезпек;

– визначення робочих сценаріїв, режимів та умов навколишнього середовища тощо;

– поєднання виявлених сценаріїв та небезпечних подій;

– класифікація небезпечних подій;

– визначення цілей безпеки, які охоплюють усі небезпечні події [21].

Процес HARA може бути виконано за допомогою спеціальних програм або на звичайних сторінках Excel. Одним з найбільш широко використовуваних інструментів для HARA є ENCO SOX. Це універсальний інструмент, який покликано допомагати автомобільним інженерам в розробках систем на основі модельно-орієнтованого проектування.

Стандарт ISO 26262 в обов'язковому порядку вимагає використання правил щодо кодування, таких як, наприклад, використання перевірених бібліотек мови програмування або конвенції з іменування.

Це спрямовується на те, щоб розробники розуміли стандарти галузі і могли створювати безпечні, легко підтримувані та здатні до розширення програми. Окрім того безперечною перевагою є те, що використання уніфікованих стандартів кодування призводить до більш легкого розуміння коду членами не тільки однієї, але і багатьох команд розробників, що зменшує час на розробку та підвищує її

ефективність, а це в свою чергу призводить до зменшення вартості кінцевого продукту.

Для автомобільної індустрії найбільш поширеною мовою програмування є C/C++, а відомими є стандарти MISRA C, MISRA C++, AUTOSAR C++14.

Для полегшення відслідковування, чи відповідає код тому чи іншому стандарту кодування, компанії використовують інструменти статичного аналізу коду [23]. Наприклад, відомими в автомобільній індустрії є CodeSonar або Helix QAC.

Helix QAC – це статичний аналізатор коду для мови програмування C/C++, який дозволяє проаналізувати код у відповідності до всіх ASIL рівнів та стандарту MISRA. Наприклад, він допоможе забезпечити потрібну складність коду за проміжок часу або строгу типізацію. Крім того він підтримує засоби відповідності вимогам до архітектурного дизайну. Наприклад, існує рекомендація щодо розміру компонентів системи або інтерфейсів, зв'язаності та зчеплення та ін.

Таким чином аналіз вимог до розробки та архітектури програмного забезпечення в автомобільній індустрії згідно стандарту функціональної безпеки ISO 26262 показав, що розробка такого ПЗ є дуже відповідальним завданням і відповідність сучасним стандартам галузі може стати вирішальним аспектом успіху продукту.

2.4 Аналіз можливих варіантів отримання вхідних даних для навчання самокерованих транспортних засобів

Для навчання самокерованих транспортних засобів необхідним є дуже великий об’єм інформації, тому що взаємодію з реальним світом неможливо передбачити і для успішного навчання система повинна бути ознайомлена з якнайбільшою кількістю різнорідних даних.

На сьогодні отримання даних не є проблемою, особливо для автомобілів, тому що велика кількість сенсорів та багато кілометрів водіння дозволяє збирати дані для навчання в режимі реального часу. Такі дані називаються «сирими», тобто

необробленими. Це дані з камер, лідарів, радарів та інших сучасних сенсорів, якими оснащено кожний автомобіль.

Такі дані є безперечно важливими для використання з цілями навчання транспортних засобів, але їх надзвичайно складно обробити. За статистикою кожну годину середньостатистичний міський автомобіль збирає близько 25 Гб даних. Їх потрібно десь зберігати, що є звичайно дорого. Крім того для обробки такої кількості різнорідних даних необхідні великі процесорні потужності, що також є досить дороговартісно.

Наприклад, компанія Tesla збирає близько 3 мільйони миль даних за день і є найбільшою компанію за цими показниками.

Але все ж таки не кожна компанія може собі дозволити отримувати та обробляти дані з реального середовища. Тому одним надзвичайно ефективним методом навчання є збір даних в середовищах моделювання. Наприклад, одним з лідерів ринку автономних транспортних засобів є компанія Waymo, що хоча і має не тільки фінансові можливості, але і реальні авто на дорогах з рівнем автопілота, що перевищує автомобілі компанії Tesla, але все ж таки більшість цінних результатів навчання, вони отримують з середовищ моделювання.

Таким чином можна зробити висновок, що дані зібрані у середовищі моделювання можуть бути не тільки ефективними, достатньо точними для успішного навчання самокерованих транспортних засобів, а також це дозволить досягти мети не витрачаючи відносно багато коштів. Отже для навчання самокерованих транспортних засобів у рамках розроблюваної архітектури обрано синтетичні дані отримані всередині віртуального середовища моделювання.

2.5 Аналіз можливостей прийняття рішень в середовищі програмної симуляції

Однією з найважливіших частин розробки автономних систем є вирішення наступних проблем:

– прогнозування дій автономної системи відповідно до умов навколишнього середовища та статичних та динамічних об’єктів, з якими вона стикається;

– прийняття миттєвих та довгострокових рішень на основі наявних вхідних даних;

– планування траєкторії руху;

– виявлення найбільш оптимального шляху пересування.

Ці проблеми потребують прискіпливої уваги, тому що є тим базисом, на якому тримається взагалі вся сутність автономних систем. Тому урахування цих аспектів в розробці програмного забезпечення для тренування автономних систем є найбільшим пріоритетом.

2.5.1 Планування руху як проблема навігації автономного транспортного засобу

У галузі дослідження планування руху для автономних транспортних засобів важливою є проблема навігації, яка активно спочатку досліджувалась в галузі робототехніки, а потім поширилась на інші суміжні галузі науки.

Ця проблема ще має назву проблеми навігації або проблеми руху піаніно. Це обчислювальна задача, яка має на меті знайти послідовність дійсних конфігурацій, які переміщують об’єкт від початкової точки до пункту призначення. Сутність завдання полягає в тому, щоб досягти безперервного пересування від стартової конфігурації до цільової, при цьому уникаючи зіткнення з будь-якими об’єктами, які виникають на шляху руху [26]. Поняття «конфігурація» широко вживається в робототехніці та представляє собою специфікацію позицій всіх точок робота відповідно до фіксованої системи координат. Як правило вона описується як вектор позицій або орієнтацій [27].

Зазвичай проблема планування руху – це комплексне завдання, що розділяється на підзадачі, які являють собою дискретні процеси руху та по можливості оптимізують весь процес пересування.

Для вирішення цієї проблеми використовують різноманітні алгоритми. Наприклад, задачі невеликого функціонального обсягу зазвичай добре піддаються розв'язанню за допомогою сіткових або геометричних алгоритмів. Що ж до систем великого розміру, то тут проблема виявляється складнішою і точне планування руху, обтяжене суворими обмеженнями, не піддається простому алгоритмічному обчисленню.

Серед алгоритмів, які використовуються для складних багатовимірних систем можна виділити алгоритм потенційного поля та оснований на вибірках.

Недоліком алгоритму потенційного поля є те, що він недостатньо ефективний стосовно локального мінімума. В свою чергу алгоритм, оснований на вибірці, усуває цю проблему, але він не може допомогти вирахувати ситуацію, коли шляху для пересування немає. Хоча застосування цього алгоритму має і свої переваги – з плином часу ймовірність невдачі наближається до нуля [26].

Розглянемо проблему навігації стосовно автономного транспортного засобу. По-перше, наведемо визначення: це завдання безпечної та комфортної навігації транспортного засобу до місця призначення, дотримуючись правил дорожнього руху [28].

Це завдання може бути декомпозовано на підзавдання, що являють собою ієрархію проблем оптимізації, кожна з яких має різні обмеження та цілі (рис. 2.5) [28]. Прикладами таких задач може бути визначення місії автономного водіння, дослідження найбільш поширених сценаріїв пересування на дорозі, необхідна поведінка для вирішення кожного сценарію та деякі важливі граничні випадки, які висвітлюють проблеми планування руху.

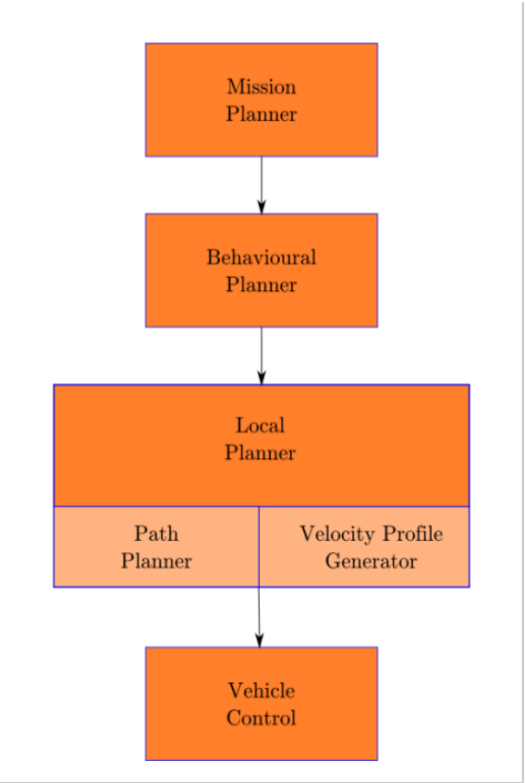


Рисунок 2.5 – Ієрархія проблем оптимізації планування руху

На найбільш високому абстрактному рівні місія автономного водіння полягає у вирішенні задачі досягнення кінцевого пункту призначення від заданої конкретної координати. Задача для автомобіля ускладняється тим, що потрібно брати до уваги можливість проїзду тієї чи іншою вулицею відповідно до міського плану, а також можливі затори на дорогах для оптимізації маршруту та часу на пересування. На першому етапі всі важливі, але побічні змінні не беруться до уваги для абстрагування від незначних для даної найбільш поверхневої задачі деталей. Такими деталями можуть бути погодні умови, структура дороги, перешкоди та інше.

Іншим важливим компонентом є планування поведінки самокерованого транспортного засобу або проблема поведінки автономної системи. Під поведінкою мається на увазі відслідковування швидкості, гальмування, прискорення, маневрування, зміна напрямку руху та ін. для кожного конкретного сценарію. Спираючись на певний маневр, локальний планувальник використовується для розрахунку шляху, який виключає можливість зіткнення, а

також профіля швидкості для необхідного цільового стану системи. Після всіх необхідних розрахунків детальний план пересування передається відповідним контролерам транспортного засобу.

Звичайно проблема планування обтяжується різними перешкодами, серед яких зазвичай виділяють найбільш критичні:

– кінематика та динаміка автомобіля;
– статичні та динамічні перешкоди;
– регуляторні елементи.

2.5.2 Можливості розв’язання проблеми прийняття рішень для частково автономних транспортних засобів

Розглянемо більш детальніше такий компонент в ієрархії архітектури системи планування руху як планування поведінки самокерованого транспортного засобу.

Як вже зазначалося, система планування поведінки – це система, яка відповідає за планування високорівневих дій або за безпечне маневрування в різних дорожніх умовах для досягнення поставленої місії автономного водіння. Ця система є важливим компонентом в ієрархії прийняття рішень самокерованим транспортним засобом.

Наприклад, в ситуації, коли автономний засіб знаходиться на перехресті дороги, саме система планування поведінки приймає миттєві рішення щодо необхідних дій, а також адаптується до змін оточуючого середовища згідно з максимально ефективним для даних умов розрахунком. Такий розрахунок в першу чергу повинен враховувати питання безпеки, а також дорожніх правил. Крім того існує велика ймовірність помилкових даних, пов'язаних з шумом вимірювання, які планувальник отримує від системи сприйняття оточуючого середовища через сенсори. З такими ложнопозитивними або ложнонегативними даними планувальник повинен вміти працювати.

Таким чином ми можемо побачити, що процес прийняття рішень повинен бути всебічно досліджений для врахування всіх деталей при розробці програмного забезпечення для навчання автономних систем.

Широкі можливості впровадження механізму прийняття рішень в середовищах симулювання надає концепція скінченного автомату (FSM – finite-state machine), яка вже давно використовується для подібного виду завдань. Ця концепція є математичною моделлю обчислення, сутністю якої є абстрактний автомат, який може бути лише в одному з скінченої кількості станів в кожний певний проміжок часу. Скінченний автомат може змінювати стани у відповідь на вхідні стимули. Таку зміну прийнято називати перехідним станом [29]. Отже для того, щоб виконати певну цільову дію, автомату потрібно зробити перехід від одного стану до іншого.

Цей механізм добре зарекомендував себе в конструюванні та тестуванні технічних приладів дискретної дії, через те що для них можна легко виокремити скінченну кількість станів завдяки їх скінченним розмірам. Концепція скінченного автомату досліджується в області теорії автоматів, яка отримала поширення на початку 50-х рр. XX сторіччя [30].

Зазвичай для відображення концепції скінченного автомату використовується діаграма станів, яка описує поведінку досліджуваної системи. Кожний можливий стан позначається колом, а переходи між ними – стрілками (рис. 2.6).

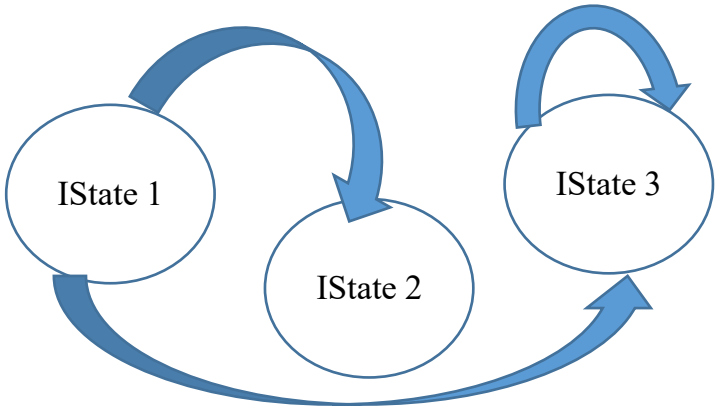


Рисунок 2.6 – Приклад діаграми станів

Для того щоб проаналізувати можливості прийняття рішень самокерованим транспортним засобом виділимо п’ять основних маневрів (станів скінченного автомату), які можуть бути виконані транспортним засобом для успішної реалізації завдання водіння:

– підтримка певного рівня швидкості, для того щоб відповідати поточному загальному показнику швидкості на дорозі відповідно до інших транспортних засобів;

– слідування за лідируючим транспортним засобом (підтримка швидкості та дистанції);

– поступове гальмування (зупинка перед певним об’єктом);

– зупинка (та перебування у даному стані на певний проміжок часу);

– перестроювання в іншу смугу руху.

Крім того потрібно виділити вхідні та вихідні дані, необхідні для коректної роботи планувальника.

Опис вхідних даних:

– дорожня карта високої чіткості;

– шлях пересування згідно з місією водіння;

– інформація стосовно визначення місцеположення транспортного засобу;

– інформація отримана від системи сприйняття (всі динамічні та статичні об'єкти в полі видимості, карта заповненості (англ. occupancy grid)).

Опис очікуваних вихідних даних:

а) опис очікуваного маневру водіння в конкретних умовах;

б) набір обмежень, згідно з якими формується необхідна траєкторія руху:

1) найбільш оптимальний шлях;

2) обмеження швидкості;

3) межа смуги дорожнього руху;

4) можливі вимушені зупинки руху;

5) можливо задіяні в маневрі інші транспортні засоби.

Таким чином ми очікуємо від планувальника вибір прийнятної для кожної конкретної ситуації поведінки, а також необхідних обмежень для безпечного та ефективного пересування самокерованого транспортного засобу.

У нашому досліджуваному випадку поняттю перехід в концепції скінченного автомату відповідає перехід від одного маневру до іншого. Він відповідальний за визначення певного правила, яке повинно бути враховано перед тим, як маневр може бути успішно виконано.

Однією з найскладніших ділянок дорожньої смуги є перехрестя, тому що тут задіяно багато об’єктів, регуляторних правил та можливих напрямків руху. Розглянемо приклад прийняття рішення на перехресті перед знаком «стоп» без дорожнього трафіку (рис. 2.7) [28].

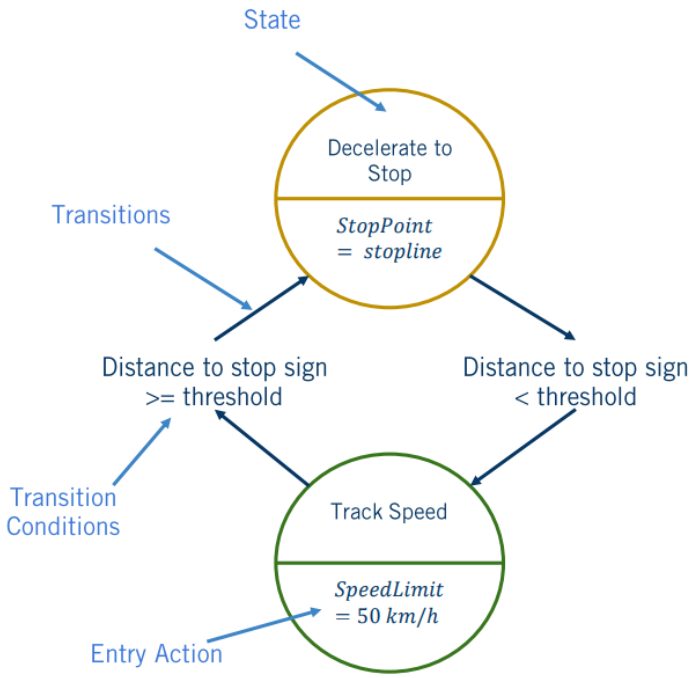


Рисунок 2.7 – Діаграма станів скінченного автомату для сценарію на перехресті без дорожнього трафіку

В ньому задіяно лише два стани – поступове гальмування та підтримка певного рівня швидкості. Окрім того нам потрібно ввести також для кожного стану певні обмеження, такі як місце, перед яким потрібно зупинитися, а також обмеження швидкості для даної ділянки дороги. Назвемо такі обмеження вхідною дією для певного стану. Щодо переходів, то в нашому випадку маємо лише дві можливості – відстеження швидкості та початок гальмування або навпаки. Для коректного переходу необхідно запровадити дотримання певних умов, таких як конкретне граничне значення відстані до знаку «стоп», яке не може бути перевищене (на діаграмі позначається як threshold).

Таким чином механізм скінченного автомату може бути дуже ефективним та простим інструментом у вирішенні завдання планування поведінки комплексної системи. Завдяки декомпозиції комплексного завдання на підзадачі досягається абстрагування від незначних деталей, що дозволяє виокремити конкретні стани, обмеження, переходи та правила для даного конкретного сценарію.

Представлений скінченний автомат дозволяє легко запрограмувати очікувану поведінку системи та отримати модель прийняття рішення в певних конкретних умовах. Такий підхід має значні перспективи навчання самокерованих транспортних засобів до 4 рівня автономності, коли автопілот ще діє в визначених конкретних умовах, а тому обмежується конкретною скінченною кількістю станів.

Хоча для автопілотів 5 рівня автономності, тобто повністю самокерованих, такий метод має важливий недолік, тому що п’ятий рівень характеризується необмеженою кількістю умов, в яких функціонує система. Тому в даному випадку потрібно звернутися до інших просунутих методів прогнозування поведінки та прийняття рішень.

Такими методами стають методи машинного навчання.

2.5.3 Аналіз можливостей розв’язання проблеми прийняття рішень для самокерованих транспортних засобів за допомогою методів машинного навчання

Сьогодні неможливо собі уявити галузі, яка не використовували б надбання штучного інтелекту. На рисунку 2.8 зображено рівень використання ШІ по галузям [24]. Можна побачити, що лідерами по залученню технологій ШІ є фінансова галузь та інформаційно-комунікаційні технології.

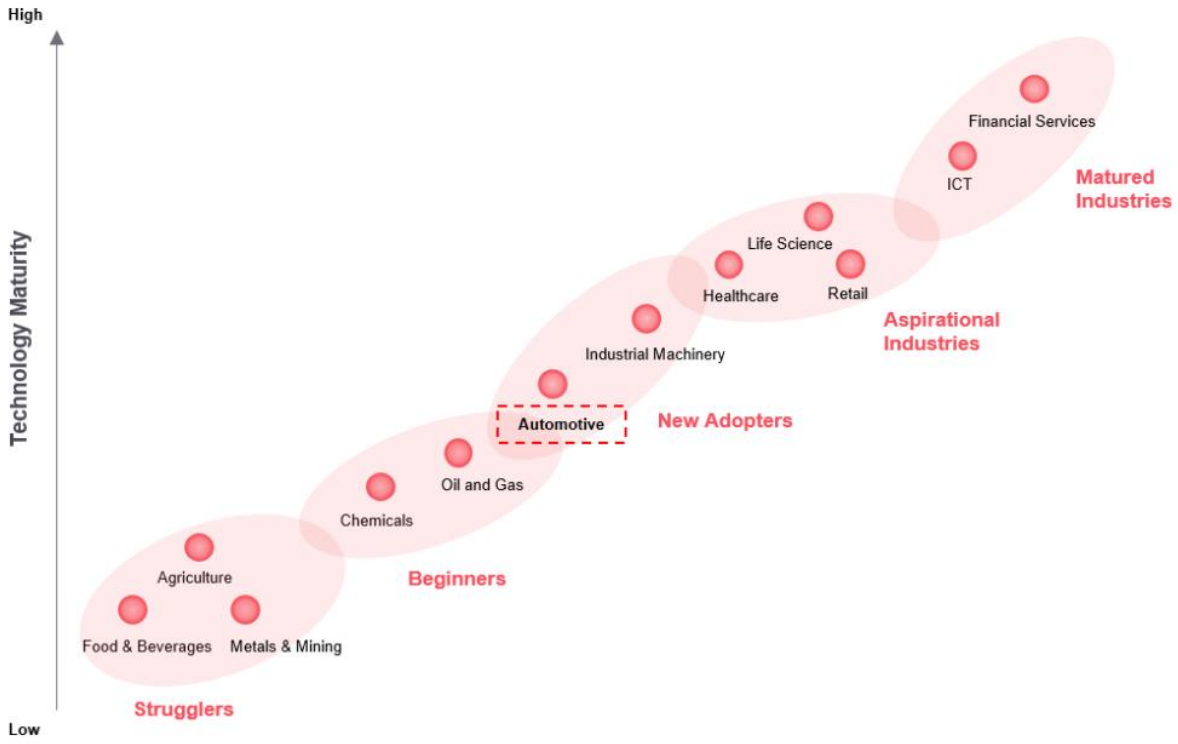


Рисунок 2.8 – Графік рівня використання ШІ по галузям

Автомобільна індустрія не є виключенням, хоча і процес адаптації технологій машинного навчання проходить тут не так швидко та гладко, як хотілось би. Причин цьому багато, але одними з них є те, що система для сучасного автомобіля містить більш ніж 100 мільйонів строчок коду, більшість з якого legacy і таким чином впровадити нові технології не завжди можна швидко та без помилок.

Окрім того найбільш важливою перепоною тут постає знову ж таки питання безпеки, а з нейронними мережами як відомо не завжди можна передбачити, як поведе себе система в тій чи іншій ситуації, тобто присутній аспект випадковості, що звичайно неприпустимий в питаннях безпеки.

За прогнозами використання технологій ШІ к 2030 року повинно досягнути 95 - 98% по галузі в цілому (рис. 2.9) [24].

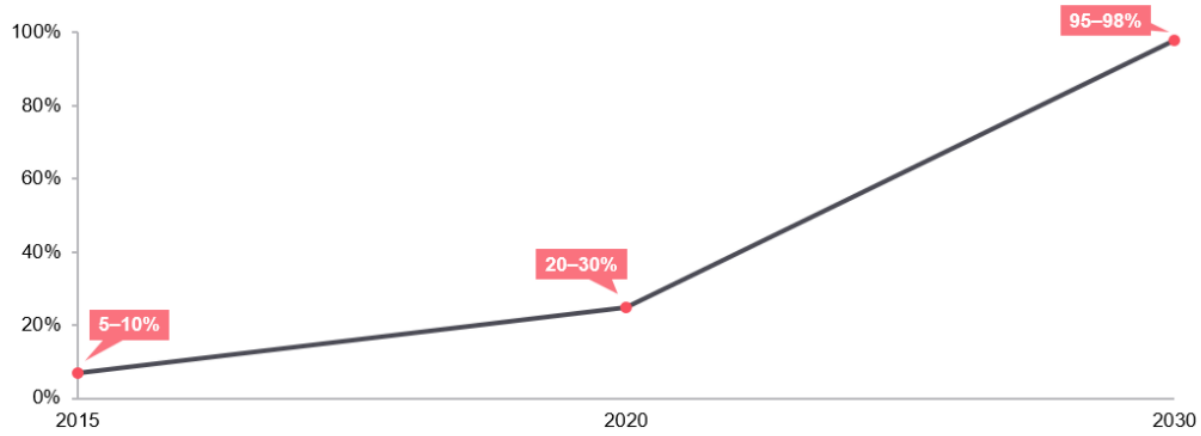


Рисунок 2.9 – Прогноз використання ШІ в автомобільній індустрії до 2030 р.

Тому для прискорення впровадження здобутків ШІ використовують середовища моделювання, в яких можна навчати моделі, не наражаючи на небезпеку людей, а також мінімізуючи можливі негативні наслідки від спроб і помилок засобів машинного навчання.

Хоча машинне навчання в розробці автономних транспортних засобів значно зросло за останні кілька років, використання методів навчання з підкріпленням (RL) застосовується лише нещодавно. Згорткові нейронні мережі (CNN) стали популярними завдяки їх властивостям точного виявлення та ідентифікації об'єктів і навіть забезпечення майже повного контроля автономного транспортного засобу. Однак однією з вимог CNN є велика кількість маркованих даних для інформування та навчання нейронної мережі. Хоча дані стають більш доступними, ці мережі все ще чутливі до формату та середовища збору, що ускладнює використання чужих даних.

На відміну від цього, RL розробляє рішення в середовищі імітації шляхом спроб і помилок без маркованих даних. Використовуючи цей метод навчання без вчителя, дослідження покликане продемонструвати здатність вивчати нові стратегії навчання, перебуваючи в змодельованому середовищі без потреби у великих обсягах реальних даних.

Використання середовищ моделювання для RL є важливим, оскільки методологія навчання без вчителя вимагає багатьох випробувань, щоб навчитися належній бажаній поведінці. Запустити це в реальному світі було б дорого і непрактично, однак моделювання дозволяє розробляти рішення з низькими витратами та часом, оскільки процес можна пришвидшити поза реальним часом.

Середовище моделювання має високу точність моделювання динаміки автомобіля, а також здатність до рендерингу навколишнього середовища, і дозволяє з високою вірогідністю успішне перенесення моделювання в реальний світ. Цей підхід призводить до недорогого рішення з порівняно низькою потребою в реальних даних, що дозволяє контролювати прийняття рішень автотранспортом з вбудованою системою самокерування.

Основним принципом навчання з підкріпленням є система дій та винагород. Агент взаємодіє з середовищем і на основі результатів такої взаємодії навчається. Агенту ніхто не каже, що робити, він повинен самостійно дізнатися, які дії потрібно виконувати через систему винагороди за успішну дію. Таким чином процес пошуку, спроби і помилок та винагороди за успіх постійно повторюється. Такий пошук ускладняється тим, що агент діє в невизначеному та потенційно складному середовищі. Результатом повинно стати досягнення встановленої мети.

В процесі навчання з підкріпленням широко використовуються ідеї теорії динамічних систем, особливо марковські процеси вирішування (МПВ), які є відомими завдяки можливостям прийняття рішень на основі математичних моделей [25].

Важливим аспектом, що відрізняє навчання з підкріпленням від інших алгоритмів машинного навчання (навчання в вчителем та без вчителя) є те, що тут постає так названа дилема «дослідження та експлуатації». Щоб отримати багато винагород, агент повинен обирати дії, що він вже спробував в минулому та які були ефективними. Але щоб знайти такі дії, він повинен спробувати те, що до цього часу ще не обирав.

Агент повинен експлуатувати те, що він уже зазнав, щоб отримати винагороду, але він також повинен досліджувати, щоб у майбутньому зробити кращий вибір дій. Дилема полягає в тому, що ані дослідження, ані експлуатацію не можна виконувати виключно без невдачі у виконанні поставленого завдання. Агент повинен спробувати різні дії і поступово віддавати перевагу тим, які здаються найкращими. У стохастичному завданні кожну дію потрібно багато разів спробувати, щоб отримати достовірну оцінку очікуваної винагороди. Дилема дослідження-експлуатації математиками інтенсивно вивчається протягом багатьох десятиліть, проте залишається невирішеною [25].

Окрім агента та середовища можна також виділити чотири важливих компонента навчання з підкріпленням:

– політика визначає модель поведінки агента в певний час;

– сигнал винагороди визначає мету навчання з підкріпленням;

– функція значення визначає загальний розмір винагороди на довгострокову перспективу;

– модель середовища дозволяє робити припущення щодо поведінки середовища.

На сьогодні багато великих автомобільних компаній використовують метод навчання з підкріпленням, але в його чистому вигляді дуже складно досягти значних результатів через велику кількість можливих сценаріїв на дорозі, з якими може стикнутися самокерований транспортний засіб та які неможливо передбачити.

Тому проблема полягає в обранні найбільш ефективного поєднання наявних алгоритмів та деяка адаптація під кожне конкретне завдання. Наприклад, такою адаптацією може стати ієрархічне навчання з підкріпленням (HRL) завдяки якому відбувається декомпозиція складних завдань прийняття рішень на підзадачі, які в свою чергу можуть оброблятися більш ефективно, а результати навчання потім поєднуються спеціальним чином задля тестування роботи всієї системи.

Наприклад, для планувальника в системі автономного автомобіля такий підхід може бути застосовуватися з точки зору високорівневих та низькорівневих політик (набору правил) для маневрів (рис. 2.10).

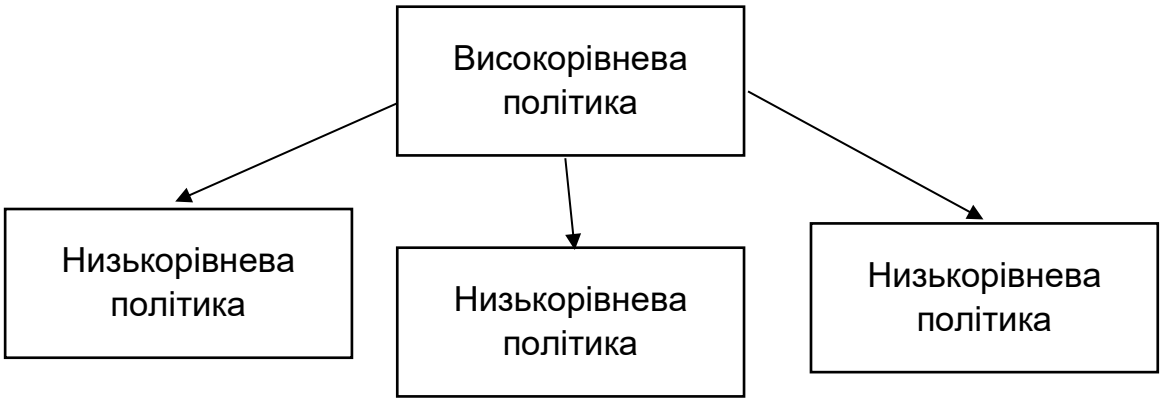


Рисунок 2.10 – Декомпозиція політик для планувальника згідно з HRL

Тобто кожна політика низького рівня самостійно навчається, і лише в разі успіху політика високого рівня може навчатися для завершення завдання.

Іншим сучасним підходом є навчання з підкріпленням засноване на моделях (model-based reinforcement learning) [31]. Його сутністю є ключовий момент згідно з яким навчається не тільки агент, але і модель його середовища. Наприклад, самокерований транспортний засіб стикається з динамічними об’єктами на своєму шляху, і якщо модель руху динамічних об’єктів може бути правильно оцінена досліджуваним транспортним засобом, планувальних матиме змогу прийняти більш точні рішення щодо пересування та досягнення місії водіння.

Звичайно у кожного метода є свої недоліки і головним недоліком навчання у середовищах симулювання є те, що вони не завжди відображають реальний світ настільки детально, різноманітно та з високим показником випадковості, як цього вимагають питання безпеки, етичних та регуляторних норм. Деякі середовища симулювання є занадто спрощеними, які звичайно мають високу ефективність в академічних цілях, але не можуть бути повністю перенесені на реальні транспортні засоби. Середовища, які володіють деталізованим описом реального світу потребують дуже високих обчислювальних потужностей, а для того щоб навчити модель потрібно не один раз повторити набір одноманітних дій, що не завжди можливо в настільки важкому програмному забезпеченню.

Дослідники намагаються подолати цю проблему спираючись на поведінку реального водія у якості еталонної політики. Для цього використовують зворотне навчання з підкріпленням (IRL), яке спирається на механізм отримання функції винагороди з оптимальної поведінки експертного агента, що діє в конкретному середовищі [32]. Результатом вивчення функції винагороди становиться можливість виконання алгоритмом маневрів подібно до реального водія.

На сьогодні все ж таки проблема прийняття рішень та прогнозування системами автоматизації водіння не вирішена та в науковому середовищі це один з найважливіших факторів, що стримує розробку автономного транспорту з п’ятим рівнем автономності. Таким чином розробка програмного забезпечення, яке дозволить підвищити ефективність навчання таких систем є як ніколи актуальною. Тому дане дослідження спрямоване на те, щоб використовуючи наявні середовища моделювання та найбільш оптимальне поєднання алгоритмів машинного навчання розробити архітектуру програмного забезпечення, яке дасть змогу наблизитися до цієї мети.

Висновки по розділу

Таким чином у другому розділі всебічно досліджено ключові аспекти, необхідні для побудови оптимальної архітектури програмного забезпечення для навчання та тестування самокерованих транспортних засобів.

Проаналізовано методологію розробки ПЗ заснованій на підході модельно-орієнтованого проектування, досліджено рівні автоматизації автономних транспортних засобів за класифікацією SAE з метою детального розуміння, для якої системи буде розроблена архітектура.

Проаналізовано вимоги до розробки та архітектури програмного забезпечення в автомобільній індустрії згідно стандарту функціональної безпеки ISO 26262.

Крім того проведено аналіз можливих варіантів отримання вхідних даних для навчання самокерованих транспортних засобів та можливостей прийняття рішень в середовищі програмної симуляції. Акцент в питанні прийняття рішень зроблено на алгоритми машинного навчання.

Проведений аналіз інформаційного забезпечення дозволив зробити висновки щодо необхідних підготовчих етапів до побудови архітектурного рішення та розробки програмного забезпечення, а також на основі обширної теоретичної бази в досліджуваній предметній області було виокремлено основні ключові компоненти, що необхідні для побудови архітектури ПЗ, яка дозволить підвищити ефективність навчання самокерованих транспортних засобів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Підготовчі етапи розробки програмного забезпечення

3.1.1 Постановка коректних вимог до програмного забезпечення для усунення конфліктів користувацької взаємодії

Першим підготовчим етапом розробки після всебічного дослідження предметної області та аналізу переваг та недоліків існуючих рішень є етап постановки вимог до розроблюваного програмного забезпечення.

По-перше, програмне забезпечення повинно відповідати стандартам якості в області автомобільної індустрії, тому необхідно додержуватися правил написання коду згідно з вимогами зрозумілості, розширюваності, гнучкості та легкої майбутньої підтримки. Крім того зважаючи на мету дослідження програмне забезпечення повинно бути простим для сприйняття, мати зручний інтерфейс та можливість використання на різних операційних системах таких як Windows, Linux, Mac.

Також важливим є те, що з огляду на недоліки проаналізованих середовищ моделювання виявилося те, що для їх використання потрібно прочитати велику кількість документації, налаштувати багату кількість залежностей з іншими бібліотеками. Тому розроблюване програмне забезпечення має на меті усунути вищеназвані недоліки та дозволити користувачеві приступити до навчання своїх моделей лише запустивши виконуваний .exe файл та створивши конфігураційний файл з відповідними параметрами навчання. Такий підхід дозволить добитися легкого перенесення такого ПЗ на інші комп’ютери за допомогою можливостей обміну файлами.

Крім того розроблюване програмне забезпечення не повинно потребувати значних обчислювальних можливостей системи, на якій відбувається навчання. Річ у тім, що проблема таких симуляторів як CARLA чи AirSim є в тому, що навколишній світ в них настільки деталізовано, що це потребує значних обчислювальних ресурсів, а відомо, що для успішних результатів навчання потрібно зробити n-кількість одноманітних повторюваних дій. Якщо система

навантажена обчисленням занадто деталізованих середовищ, то це значно впливає на ефективність навчання та збільшує час на нього.

Тому розроблювана система повинна втілювати баланс між достатньою необхідною реалістичністю навколишнього середовища, але не реалізовувати незначні деталі, які не впливають на вихідний результат. Такий підхід дозволить домогтися проміжних результатів, які можна буде використовувати для навчання на більш складних середовищах, а потом вже на міських дорогах в системах реальних автомобілів.

Варто звернути особливу увагу на необхідність впровадження можливостей навчати моделі в середовищі моделювання через користувацький інтерфейс, а не через редактор коду. Такий підхід надасть можливість зменшити кількість часу на підготовчі етапи, а одразу ж приступити до процесу навчання. Навчені моделі зберігаються до файлу на локальному диску та можуть бути повторно використані як для подальшого навчання, так і для використання для подальших наукових або комерційних потреб.

Окрім того важливим є також надання користувачу продукту можливості налагодження основних параметрів навчання задля експериментів з найбільш оптимальними показниками для нейронної мережі. З огляду на існуючі рішення такі зміни неможливо зробити в процесі навчання і це потребує більш глибоких знань у програмуванні, тому що в існуючих продуктах необхідно писати спеціальні скрипти для того, щоб можна було поміняти важливі значення змінних, що використовуються або як вхідні параметри для навчання, або як важливі архітектурні показники нейронної мережі, такі як кількість нейронів або кількість шарів.

З огляду на це потрібно проаналізувати існуючі технологічні можливості, які нададуть змогу задовольнити вищеозначені вимоги задля усунення конфліктів користувацької взаємодії з розробленим програмним забезпеченням.

3.1.2 Вибір технологічного забезпечення та обґрунтування доцільності

Для того щоб навчити модель найбільш важливим фактором є дані для навчання. Тому постає питання збору даних для навчання, їх необхідної кількості та якості. Найбільш точними даними є дані, отримані в реальному середовищі, але з огляду на те, що їх обробка – це надзвичайно складна задача, що потребує значних обчислювальних ресурсів, найкращим варіантом у розрізі розглянутого питання є використання даних, що отримано в середовищі віртуальної 3D симуляції.

Зазвичай для відображення реального світу в середовищах моделювання використовують графічні ігрові платформи. Існує декілька варіантів, які можна використовувати для рішення поставленої задачі. На рис. 3.1. надано статистичні дані станом на 2019 рік щодо поширеності різних технологічних рішень [33].

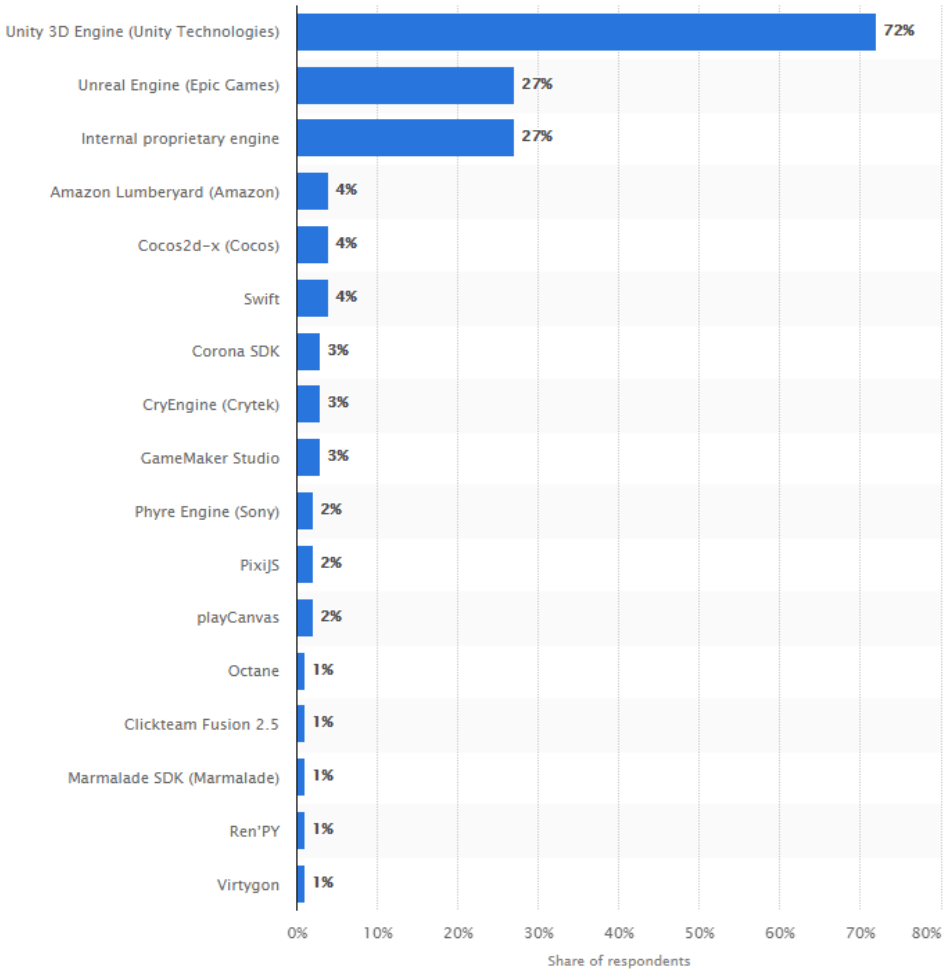


Рисунок 3.1 – Використання графічних платформ станом на 2019 рік, %

В додатку А надано порівняльну характеристику головних показників деяких графічних платформ [34].

Приведемо більш детальне порівняння двох відомих платформ – Unity та Unreal Engine для того, щоб обрати найбільш оптимальний варіант для розроблюваного ПЗ.

За показниками графічної якості обидва ігрових рушія майже не мають відмінностей, професійний розробник може досягти реалістичного графічного відображення використовуючи обидві платформи. Але як було відмічено симулятор не має на меті досягти високої реалістичності зображення, тому що це буде впливати на показники навчання моделей.

Щодо основної мови написання скриптів, то Unity використовує C#, а Unreal Engine – C++. Звичайно для кожного розробника поняття рівня складності мови відрізняється, але взагалі вважається, що C# більш легкий для написання скриптів для симулятора, а також має велику кількість бібліотек, що полегшує процес написання програми, а також зменшує час на розробку. Крім того завдяки вбудованому збирачу сміття в мові C# знижується вірогідність витоку пам’яті, тоді як C++ не має такого функціоналу і помилки витоку – не рідкість. Хоча звісно використання C++ надає більші можливості маніпулювання пам’яті, а також завдяки відсутності зайвих надбудов показує кращі показники продуктивності.

Щодо документування, то обидві платформи мають широку підтримку спільноти, а також вичерпне документування, але варто наголосити, що Unity надає велику кількість навчальних матеріалів, а також структура документів більш зрозуміла та детальна. Отже це означає, що підтримувати програмне забезпечення, побудоване на Unity буде достатньо легко.

Важливою перевагою для архітектурного рішення є те, що ця платформа відрізняється високою модульністю, та робота за нею полегшується завдяки ассетам – спеціальним пакетам, які дозволяють легко та швидко додати до проекта різноманітні об’єкти готові до використання.

Властивості модульності дозволяють змінювати окремі частини великого проекту і це не відображається на інших компонентах готового продукту. Як відомо показники зв'язності (англ. coupling) та пов'язаності (англ. cohesion) є одним з головних характеристик якості проектованої системи, тому прагнення до високої пов'язаності та слабкої зв'язності забезпечить високу можливість підтримки, розширюваності та гнучкості програми.

Таким чином платформа Unity надає можливості задовольнити висунуті вимоги та добре підходить в якості бази, на якій буде будуватися програма.

3.1.3 Вибір способу інтеграції методів машинного навчання

Для розроблюваного програмного забезпечення було обрано використання платформи Unity також завдяки тому, що Unity надає високі можливості інтеграції з різними мовами програмування. Для поставленого завдання це є дуже важливим, тому що планується використання бібліотек для машинного навчання, найбільш зручними з яких є бібліотеки мови Python. В Unity існує можливості викликати виконання скриптів, що написано на мові Python завдяки In-Process API та Out-of-Process API.

Основним елементом, який поставляється Unity та є найбільш цікавим у розрізі досліджуваного питання – це доповнення з відкритим вихідним кодом – ML Agents [35], яке поставляє функціональні можливості алгоритмів машинного навчання та дозволяє навчати моделі всередині середовища Unity за допомогою навчання з підкріпленням та імітаційного навчання через Python API.

Базою даного доповнення є бібліотека мови Python – PyTorch, яка є однією з найбільш відомих застосувань для реалізації алгоритмів машинного навчання. Інтеграція пакету mlagents з PyTorch надає можливості в реальному часі відслідковувати результати навчання, зберігати результати в файл для повторного використання, а також наблюдати статистику навчання в TensorBoard в реальному часі. На рис. 3.2 зображено схему бібліотеки для навчання Unity ML Agents [35].

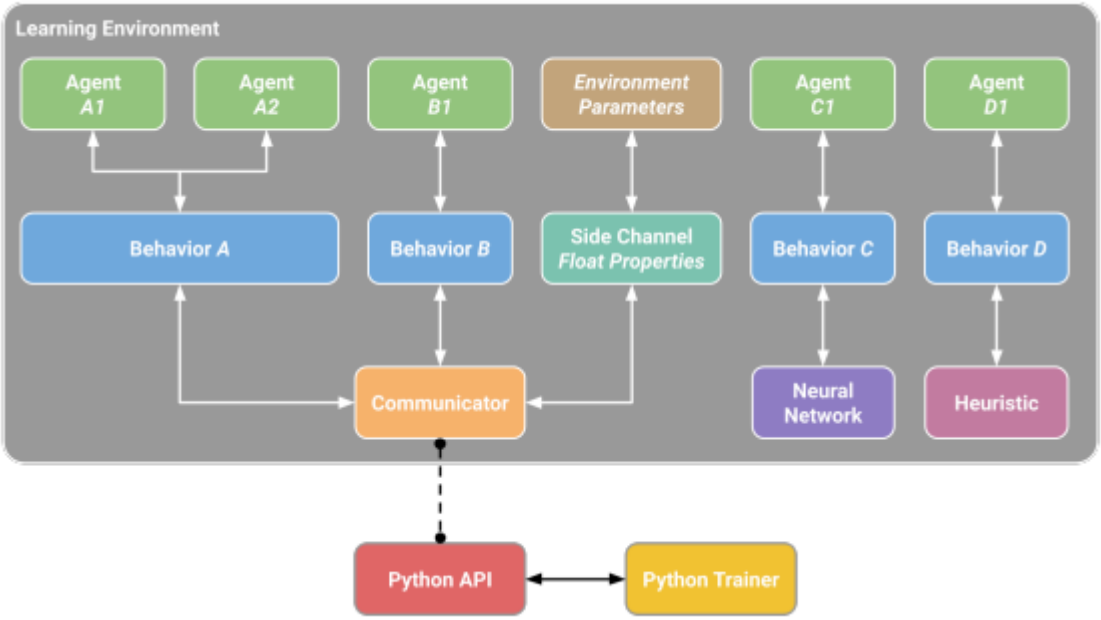


Рисунок 3.2 – Схема бібліотеки для навчання Unity ML-Agents

Інструментарій представляє собою ML-Agents SDK, який забезпечує можливість використання редактора Unity разом з базовими скриптами C# для налаштування середовища для навчання.

Наявна підтримка таких методів машинного навчання:

– навчання з підкріпленням (RL - Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO));

– імітаційне навчання (GAIL, Behavioral Cloning);

– довга короткочасна пам’ять (LSTM);

– навчання з внутрішньою мотивацією (Intrinsic CuriosityModule (ICM)).

По мірі розвитку інструментарію ML-Agents передбачається додавання додаткових алгоритмів машинного навчання, що дасть змогу з легкістю додавати можливості навчання автопілота в розроблене ПЗ за допомогою різних методів, не потребуючи глибоких знань у кожній області науки штучного інтелекту.

Крім того передбачається використання кастомних скриптів на C# без використання можливостей бібліотек, для того щоб надати можливість більш просунутим користувачам додавати до симуляції нові алгоритми та маніпулювати вхідними параметрами задля підвищення ефективності навчання.

3.2 Розробка програмного забезпечення

3.2.1 Встановлення необхідного програмного забезпечення для розробки

Першим важливим кроком для розробки програмного забезпечення є встановлення Unity. Це можна зробити різними способами (через командну строку, через вручну написаний скрипт на C#), але найбільш зручним та рекомендованим способом є встановлення Unity Hub, який представляє собою середовище для управління версіями платформи, проектами та навчальними туторіалами (рис. 3.3).

В додатку Б наведено системні вимоги для встановлення Unity 2020 LTS [36].

Перевагою використання Unity Hub є те, що він встановлює всі необхідні залежності для повноцінної роботи з кодом та візуальними компонентами. Наприклад, для роботи з кодом було обрано інтегроване середовище розробки Microsoft Visual Studio, яке має безкоштовну Community версію та надає необмежені можливості для найбільш ефективного процесу програмування. Unity Hub надає можливість відразу встановити необхідну версію Visual Studio, яка відповідає версії Unity. Тобто це скорочує час на розробку через непотрібність вивчення документації з версіонної сумісності.

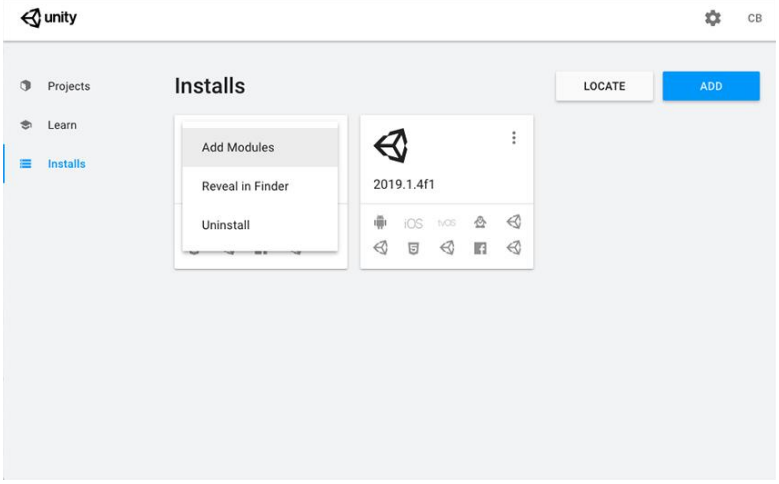


Рисунок 3.3 – Інтерфейс Unity Hub

3.2.2 Розробка інтерфейсу та контролерів

Після встановлення та налаштування всіх необхідних компонентів для розробки було розроблено програмне забезпечення, що має простий та інтуїтивно зрозумілий користувацький інтерфейс та надає можливості навчати та тестувати моделі самокерованих транспортних засобів.

Першим кроком було розроблено середовище, наближене до реального за допомогою вбудованих можливостей, що надає платформа, а також з використанням пакетів, що можна скачати та встановити з Unity Asset Store. Головні частини коду наведено у лістингу в додатку Д.

Дорожнє покриття виконане за допомогою бібліотеки Road Architect.

Найбільш важливими C# класами, що було використано в розробленому програмному забезпеченні є класи:

– GameObject;
– MonoBehavior;
– Transform;
– Vectors;
– Debug.

GameObject класс представляє собою будь-який об’єкт, що створюється в середовищі, яке має назву сцена. Таких об’єктів може бути скільки завгодно і всі візуальні компоненти, які можна побачити на екрані представляють собою об’єкти цього класу. Тут зберігаються всі правила об’єктно-орієнтованого програмування і властивості об’єктів можуть бути наслідуваними від базових класів. GameObject відповідальний не тільки за те, як об’єкт виглядає, але також і за його поведінку.

Кожний об’єкт активний за замовчуванням, але його можна відключити, він стає невидимим в симуляторі, але така можливість дуже зручна для тестування різних моделей без необхідності видаляти об’єкт взагалі. Включати та відключати об’єкт можна за допомогою метода GameObject.SetActive. Щоб додати конкретний тип об’єкту використовується AddComponent<Type>, для видалення – Object.Destroy.

MonoBehavior – це спільний класс, що являє собою фреймворк зі всіма базовими компонентами і від якого наслідуються за замовчуванням всі інші класи Unity. Завдяки MonoBehavior ми маєм можливість додати будь-який скрипт до GameObject прямо в редакторі Unity. Для цього редактор має спеціальні поля, з якими можна взаємодіяти за допомогою механізму drag-n-drop. Крім того при створенні скрипту, що наслідується від MonoBehavior, є одразу доступ до важливих івентів таких як Start (викликається, коли створюється або інстанціюється ігровий об’єкт, або завантажується сцена) та Update (викликається кожний фрейм – встановлений часовий проміжок).

Найбільш важливими класами у розрізі розроблюваного програмного забезпечення є класи Transform та Vectors, тому що вони представляють собою найважливіші математичні концепції, які необхідні для відображення реального світу, позиціонування та пересування об’єктів, обчислення напрямку та магнітуди руху.

Клас Transform представляє собою позицію, нахил та розмір об’єкту.

Для роботи з 3D об’єктами в Unity реалізовано клас Vector3. Клас Vector3 надає можливості обчислювати точну позицію об’єкта, швидкість та дистанцію між двома об’єктами. Це дані, які надзвичайно важливі для розроблюваного симулятора, тому що основною ідеєю є обчислення позиції моделі транспорту під час руху в кожний момент часу та дистанції до оточуючих об’єктів, з якими потенційно може відбуватися зіткнення. Мета навчання – навчити модель на основі цих даних уникати зіткнення з іншими об’єктами та приймати найбільш оптимальне рішення.

Таким чином найбільш важливими вхідними параметрами для навчання є показники дистанції до об’єкту в різних напрямках, а вихідними – значення типу float, які ототожнюють фізичні стимули, що представляють собою прискорення, гальмування та повороти вліво та вправо, які надаються контролеру моделі автономного транспортного засобу.

Модель для навчання – це об’єкт, що має вигляд транспортного засобу (є можливість вибору різних типів транспортних засобів, яким буде відповідати різна вага реального фізичного об’єкту, показники зчеплення з дорожнім покриттям, сили тяжіння та ін.). За управління моделлю відповідальний контролер, який написано на мові C#. Він реалізує пересування транспортного засобу та можливості взаємодії з поверхнями інших об’єктів (таких як дорожнє покриття). Для цього було реалізовано спеціальні колайдери, що являють собою механізми, які дозволяють відслідковувати взаємодію різних об’єктів та розпізнавати їх зіткнення.

Для моделі транспорту важливими є колайдери колес, які реалізовано за допомогою класу Unity.WheelCollider. Він імітує налаштування підвіски пружини та амортизатора, а також використовує модель тертя шин на основі ковзання для розрахунку зусиль контакту колеса. Виявлення зіткнення відбувається завдяки викидання променю від центра колеса по локальній осі Y [37].

Для виявлення зіткнень між моделлю та іншими об’єктами використовується технологія Physics.Raycast, яка представляє собою спрощену реалізацію лідару, який випускає промінь та за рахунок його відбиття від інших поверхонь надає інформацію щодо дистанції до них. Метод hitInfo дозволяє отримати інформацію про найближчий колайдер, з яким відбулося зіткнення. Встановлення максимальної дистанції протягом якої буде відбуватися перевірка на зіткнення відбувається за допомогою методу maxDistance. Отже було побудовано три проміння, які відображають три необхідні для навчання напрямки руху: вперед, поворот направо і наліво. Ці три значення було використано як вхідні параметри для нейронної мережі. Механізм обробки даних для навчання буде описано в розділі 4.

Вихідний код було скомпільовано засобами редактору Unity після встановлення необхідних налаштувань (назва компанії, логотип та ін.) (рис.3.4) та було отримано бінарний .exe файл для платформи Windows (через те що розробка проводилась на цій операційній системі). Додаток може бути запущено на будь-

якому іншому комп'ютері через вищеозначений файл. Наявність додаткового програмного забезпечення потребується лише для встановлення необхідних залежностей для навчання моделей за допомогою бібліотеки ML Agents. Якщо навчання проводиться за допомогою засобів кастомних скриптів написаних на C# всередині редактора Unity, то достатньо лише мати .exe файл для проведення навчання.

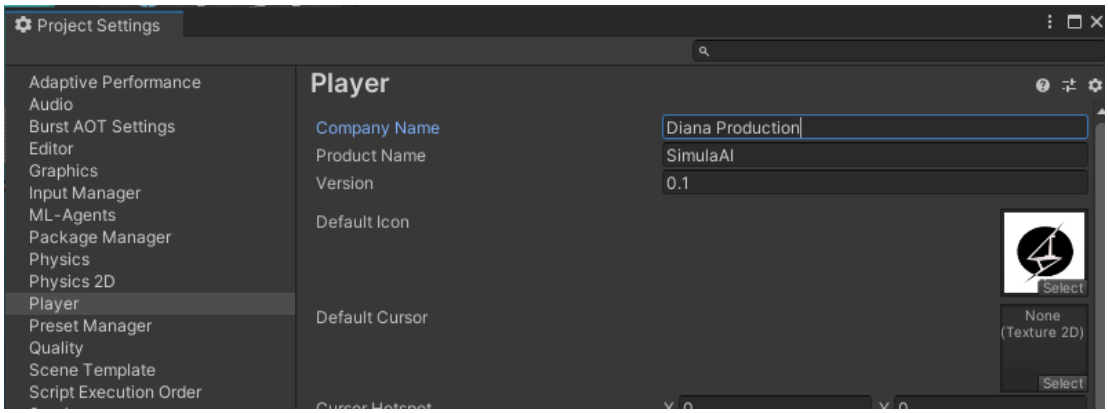


Рисунок 3.4 – Екран налаштувань для компіляції

На рис. 3.5 представлено начальний екран запуску додатка. Тут можна розмістити будь-яку інформацію про розробника або компанію.

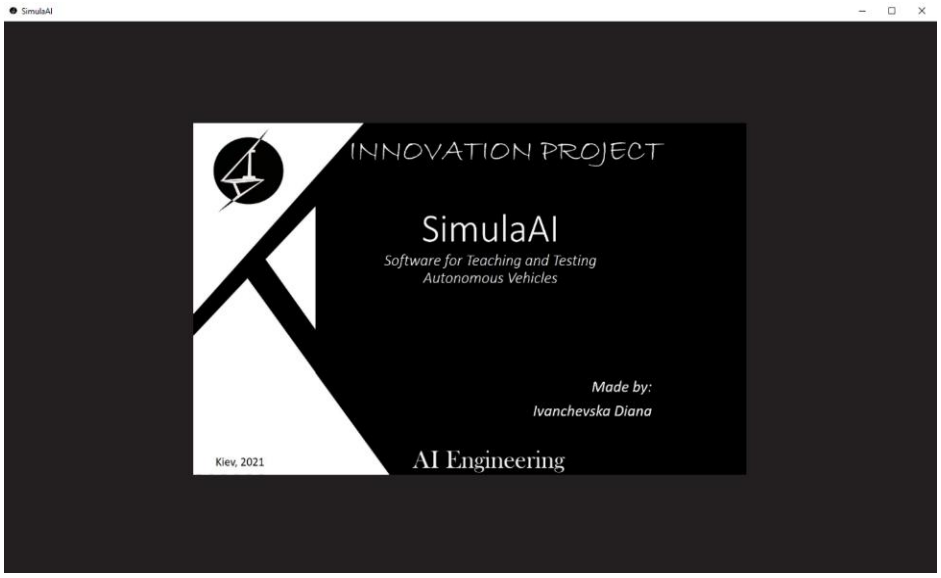


Рисунок 3.5 – Екран запуску додатка

На рис. 3.6 представлено начальний екран додатка після запуску. Він надає можливість почати роботу з додатком або вийти з нього.

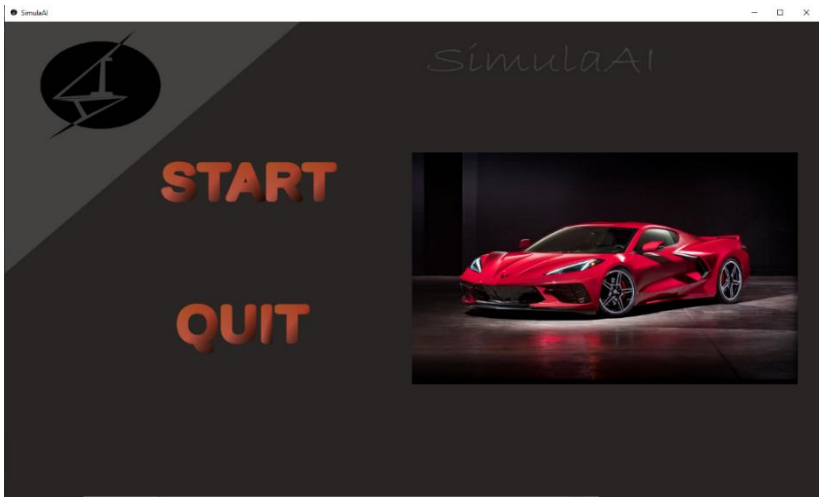


Рисунок 3.6 – Начальний екран додатка

Після вибору розпочати роботу відбувається перехід на іншу сцену, яка дозволяє обрати режим навчання та трек, на якому проводитиметься навчання. У користувача є можливість обрати трек для навчання, а також трек для тестування навченої моделі. Ручний режим може бути необхідний, якщо користувач бажає зібрати дані середовища у вигляді відео або зображень для використання в подальшому для навчання (рис. 3.7).

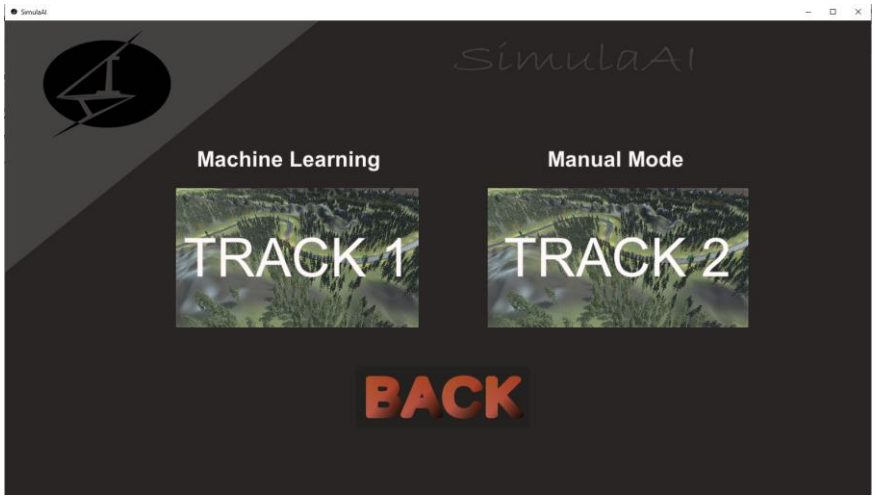


Рисунок 3.7 – Екран вибору треку

Після вибору трека відбувається процес навчання за допомогою алгоритму машинного навчання, визначеному в конфігураційному файлі (більш детально про це буде розказано в наступному пункті).

На монітор виводяться всі критичні показники, які характеризують поведінку моделі та процес навчання. Навчання можна призупинити та продовжити за допомогою кнопок Pause та Resume. Крім того візуалізовано дистанції до об’єктів зіткнення (тобто бокових обмежувачів дороги) у вигляді променів вперед, вправо та вліво (рис. 3.8).

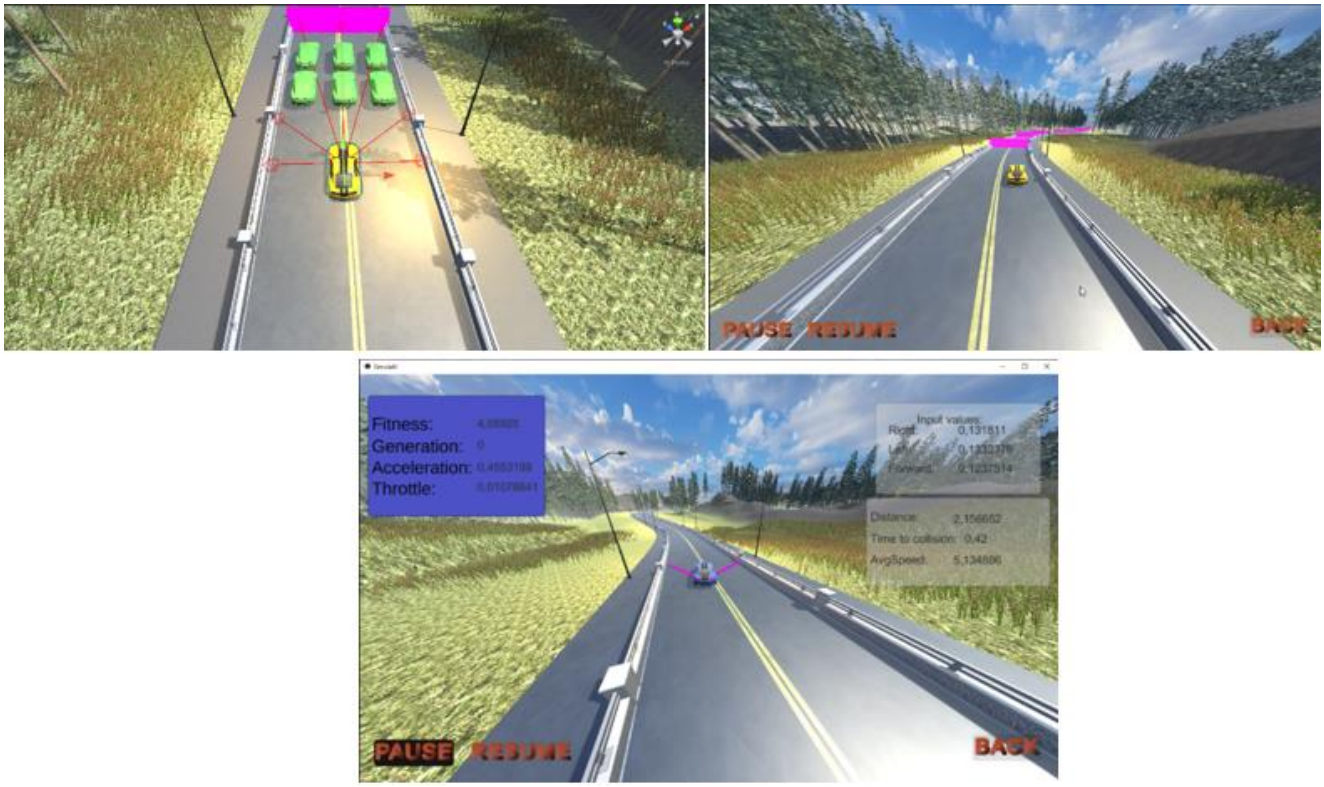


Рисунок 3.8 – Екран навчання моделі

3.2.3 Інтеграція бібліотеки ML Agents для навчання моделей

Для інтеграції з бібліотекою Unity ML Agents було виконано наступні кроки, по-перше, встановлення необхідних залежностей (рис.3.9):

- Python версії 3.9;
- пакету com.unity.ml-agents через Package Manager в Unity;
- пакету com.unity.ml-agents.extensions через Package Manager в Unity;

– PyTorch;
– пакету Python mlagents.

mlagents install

```
python -m pip install mlagents==0.27.0
```

PyTorch install

```
pip3 install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html
```

Рисунок 3.9 – Встановлення залежностей через аргументи командного рядка

Центральним компонентами ML-Agents SDK є три сутності сенсор, агент та академія. Агент використовується для встановлення залежності з досліджуваним об’єктом та збору основних даних необхідних для навчання. Він опрацьовує показники спостереження та на основі цих даних здійснює дії, а також отримує за них винагороди або покарання. Агент збирає дані навколишнього середовища за допомогою сенсорів, в нашому випадку за допомогою технології Raycast. Компонент академія (в нових версіях це компонент параметрів поведінки) відповідальний за модуль, який через Python API передає необхідні дані в нейронну мережу, його ще називають мозком агенту [35].

Для того щоб навчити модель за допомогою цих засобів було додано до моделі скрипт агенту, що відповідальний за параметри її поведінки. Для забезпечення можливості надавати агенту винагороди та покарання було реалізовано систему чекпоінтів, що представляють собою проміжні елементи проходження треку, після успішного перетинання яких агент розуміє, що дія була ефективною та загальний рівень винагороди збільшується. Якщо агент рухається не в тому напрямку або довго знаходиться на одному місці, то відбувається процес покарання, тобто вираховування винагороди.

Важливим фактором в даному підході до навчання є те, що можна обирати алгоритм машинного навчання та регулювати вхідні параметри. Це робиться за допомогою конфігураційного файлу з розширенням .yaml. Він має певну строгу структуру та потрібно дотримуватися встановленого стилю форматування задля уникнення помилок парсингу файлу. Основним параметром, який найбільш істотно впливає на процес навчання – є вибір алгоритму машинного навчання, за допомогою якого буде проводитися навчання. На даний момент пакет надає можливість використовувати PPO, SAC або POCA.

Навчати моделі можна як в редакторі Unity, так і за допомогою виконуваного файлу. Нас цікавить другий варіант, тому що завданням є підвищення ефективності взаємодії користувача з програмним забезпеченням та можливість легко передавати програму між різними комп’ютерами та надання можливості використовувати симулятор без встановлення Unity Hub та Unity. Для того щоб отримати виконуваний файл потрібно в налаштуваннях компіляції виставити необхідні параметри, встановити логотип, дозволити виконуватися програмі на задньому фоні (це важливо, тому що використання програми потребуватиме моніторингу показників в командній строчці та на графіках Tensorboard, якщо не встановити цієї можливості в симуляторі, то процес навчання та взаємодії з програмою буде неефективним, тому що модель буде навчатися лише тоді, коли вікно програми знаходиться в активному стані), а також додати всі необхідні сцени, що містяться в симуляторі. Після цього проходить процес компіляції та бінарний файл разом з необхідними даними буде розміщено в обрану папку на локальному диску комп'ютера.

Для того, щоб розпочати навчання за допомогою можливостей ML Agents, потрібно запустити командну строку та передати наступні параметри (рис.3.10):

```
mlagents-learn
mlagents-learn <trainer-config-file> --env=<env_name> --run-id=<run-identifier>
```

Рисунок 3.10 – Аргументи командного рядка для навчання за допомогою ML Agents

Першим параметром передається шлях до конфігураційного файлу, другим параметром – шлях до виконуваного файлу, третій параметр – це унікальний ідентифікатор папки, де будуть містяться всі дані про навчання. Вони зберігаються за замовчуванням до папки results та містять навчену модель (файл з розширенням .onnx) та інші дані необхідні для відображення результатів навчання у Tensorboard.

Навчена модель представляє собою дані у форматі ONNX (Open Neural Network Exchange). Цей формат, який використовується для побудови моделей нейронних мереж та дозволяє обмін ними між розробниками для використання в різних середовищах або для інтеграції як компонент готового ПЗ. Такі моделі є надзвичайно популярними сьогодні для розробки моделей та підтримуються великою кількістю інтерфейсів [39].

Після запуску навчання можна побачити стартовий екран Unity з інформацією щодо версій основних потрібних компонентів, встановлених на даний час, основними параметрами навчання з конфігураційного файлу, а також процес навчання, показники винагороди та шагу.

В додатку В можна побачити приклад налаштування конфігурації та показників навчання моделі транспортного засобу за допомогою алгоритму машинного навчання Proximal Policy Optimization.

Результати навчання графічно відображаються за допомогою Tensorboard. Для цього потрібно передати наступні аргументи до командної строки (рис. 3.11):

tensorboard

```
tensorboard --logdir results
```

Рисунок 3.11 – Аргументи командного рядка для перегляду графічних результатів навчання

Після цього можна перейти по посиланню в браузері http://localhost:6006/ та мониторити результати навчання в графічному відображенні (рис. 3.12).

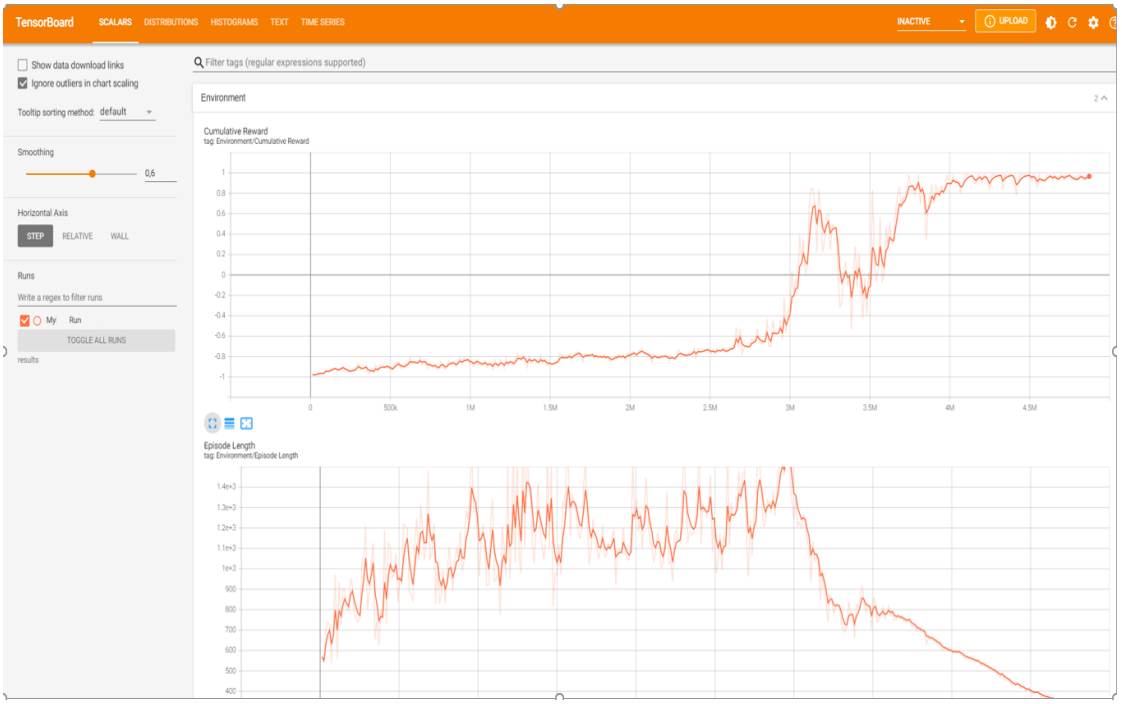


Рисунок 3.12 – Вигляд результатів навчання у Tensorboard

Висновки по розділу

Таким чином у третьому розділі розглянуто основні етапи розробки програмного забезпечення. Першим етапом була проведена постановка коректних вимог до програмного забезпечення для усунення конфліктів користувацької взаємодії. Було обґрунтовано доцільність використання платформи Unity та наведено очевидні переваги такого використання.

Було розглянуто власно процес розробки програмного забезпечення, а саме: встановлення необхідного програмного забезпечення для розробки, розробка інтерфейсу та основних контролерів для управління транспортним засобом.

Значну увагу приділено питанню інтеграції бібліотеки ML Agents для навчання моделей, тому що завдяки їй можна значно підвищити ефективність використання алгоритмів машинного навчання в розробленому програмному забезпеченні.

4 АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Архітектура програмного забезпечення

4.1.1 Схема обробки даних для механізму прийняття рішень

Для візуального відображення процесу збору та обробки даних для механізму прийняття рішень транспортними засобами з вбудованими системами самокерування було побудовано схему адаптовану на основі діаграми діяльності UML (рис. 4.1).

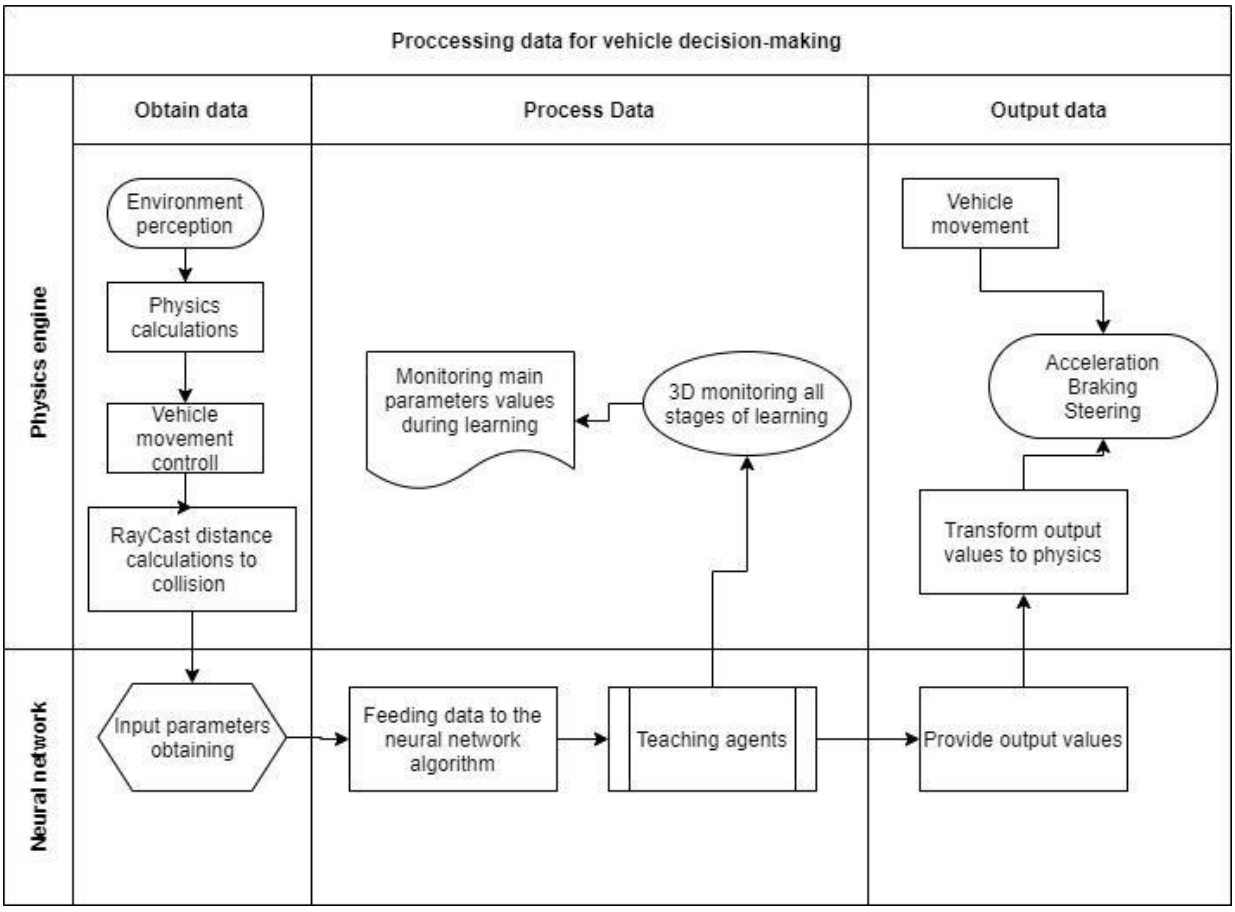


Рисунок 4.1 – Схема обробки даних для механізму прийняття рішень

На схемі зображено два рівня абстракції, на яких проходить маніпулювання з даними в певний конкретний момент. Перший рівень – це рівень візуального відображення, який програмно імплементовано за допомогою фізичного рушія.

Другий рівень виконує роль своєрідного серверу, який отримує дані, обробляє їх та надає вихідні параметри необхідні для першого рівня. Цей рівень представляє собою нейронна мережа.

Крім того схема відображає процес прийняття рішень також у розрізі іншої площини, а саме – відображення стану даних на кожному етапі обробки.

Першим етапом є збір даних в середовищі симуляції. Він відбувається завдяки можливостям фізичного рушія та містить свої під-етапи. Ці етапи імітують реальну поведінку системи самокерування. По-перше, відбувається процес сприйняття навколишнього середовища, зазвичай у реальних системах за це відповідальні сенсори, що встановлені в автотранспортному засобі (камері, лідари, радари). У нашому випадку сприйняття середовища автотранспортом можливе завдяки фізичним калькуляціям ігрової платформи, а також технології Raycast, що відповідальна за імітацію сенсорів-лідарів і робить можливим обчислення дистанції до оточуючих об'єктів. Ці обчислення є необхідними вхідними параметрами для нейронної мережі, при чому алгоритми для навчання можуть бути різними, головне - правильно обчислені параметри навколишнього середовища, які матимуть вплив на результати навчання.

Другий етап – це власне етап обробки даних, на якому обчислені параметри дистанцій до об’єктів (приведені до значень у діапазоні від 0 до 1, які необхідні для правильної роботи нейронної мережі) власне подаються до функції активації, яка продукує вихідні значення, які представляють собою власне звичайні числові змінні.

Ці змінні є просто числами лише на рівні абстракції нейронної мережі, для рівня фізичного рушія – ці дані стають відображенням руху транспортного засобу, тобто ці показники перетворюються в фізичні стимули, такі як рух вперед, гальмування, повороти.

Тобто процес навчання перетворюється у візуальну форму 3D відображення на рівні фізичного рушія. При цьому це відбувається в режимі реального часу і дозволяє користувачу значно підвищити розуміння процесу навчання і швидко змінити показники для навчання, якщо становиться зрозуміло, що нейронна мережа має помилки конструювання або потребує налаштування вхідних параметрів.

Таким чином було запропоновано процес обробки даних для механізму прийняття рішень в середовищі програмної симуляції, який дозволяє підвищити ефективність користувацької взаємодії з програмним забезпеченням.

4.1.2 Модель розгортання архітектурного рішення програмного забезпечення

З огляду на розроблене програмне забезпечення та побудовану схему обробки даних для механізму прийняття рішень було розроблено модель розгортання архітектурного рішення програмного забезпечення для навчання та тестування транспорту з вбудованою підтримкою системи самокерування (рис. 4.2).

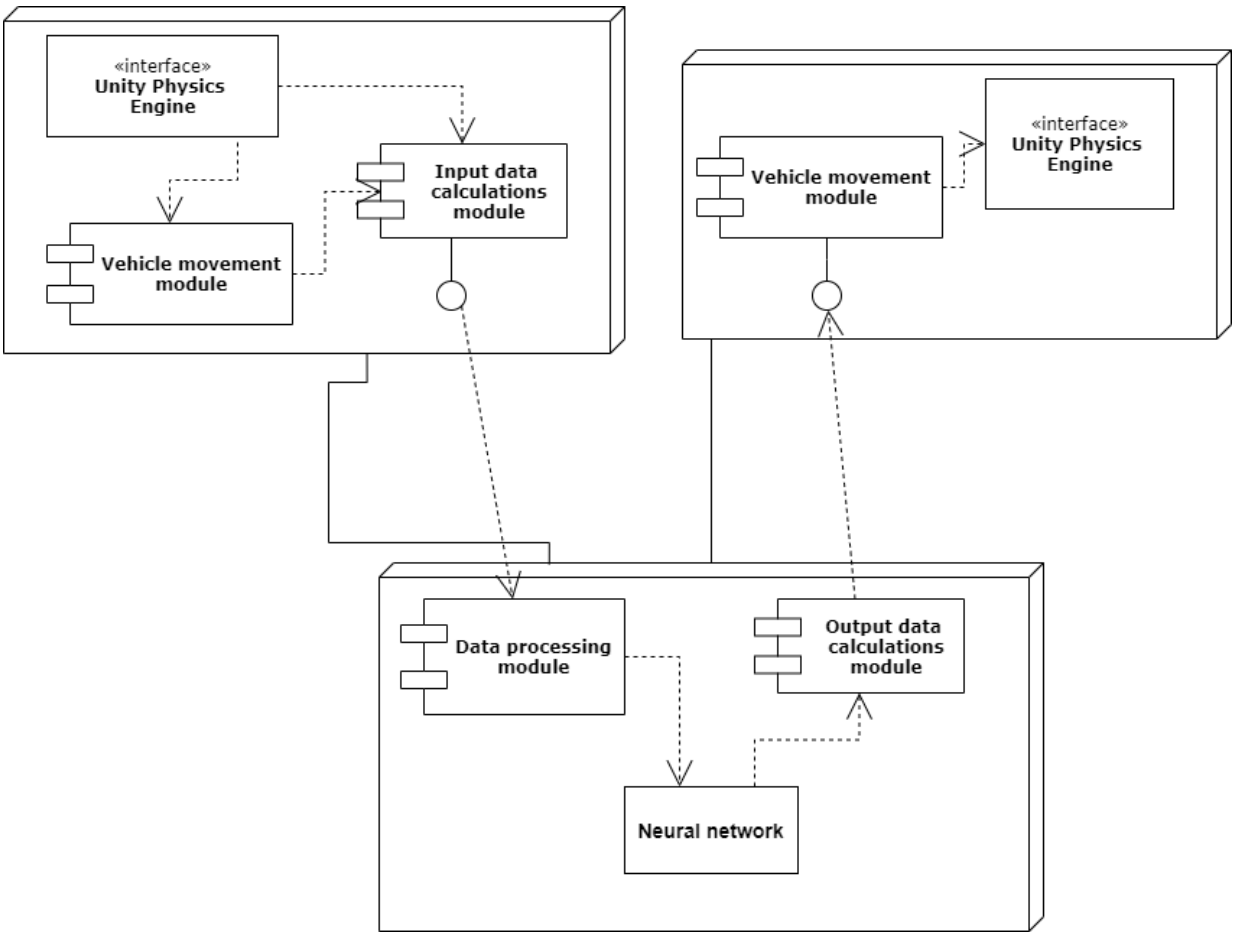


Рисунок 4.2 – Модель розгортання архітектурного рішення програмного забезпечення для навчання та тестування транспорту з вбудованою підтримкою системи самокерування

Модель графічно відображає модульний підхід до побудови архітектури програмного забезпечення. Кожний модуль виконує конкретні задачі і обмінюється даними з іншими модулями. Це дозволяє досягти великої гнучкості програмного забезпечення, легкості його подальшої підтримки, розширюваності та дозволяє знизити показник зв'язності, тобто дозволити досягти того результату, який очікується від розробленого програмного забезпечення згідно з поставленими вимогами на етапі постановки вимог.

За інтерфейс програмного додатку відповідальна платформа Unity, вона ж виконує роль своєрідного серверу, який забезпечує обчислення та візуалізацію віртуального середовища. Всередині самої платформи використовуються модулі (або компоненти середовища), які відповідальні за збір та обробку даних, а також контролери, які можуть бути використані для потрібної кількості агентів.

За навчання автопілоту відповідальна нейронна мережа, при чому маємо дві альтернативи: використання кастомних алгоритмів, що реалізовано всередині платформи за допомогою мови C# або використання можливостей машинного навчання у Python API завдяки інтеграції з пакетом Unity ML-Agents.

Для розгортання архітектурного рішення знадобиться скомпілювати всі залежності фізичного рушія Unity, а також встановити необхідні залежності Unity ML-Agents.

4.1.3 Перспективи подальших функціональних покращень та практичні можливості застосування

Проведене дослідження та практична розробка ПЗ дозволили детальніше зрозуміти, які можуть бути реальні потреби у користувачів розроблюваного рішення, а також існує необхідність врахування вимог, які можуть з'явитися у великих компаній-користувачів, що оперують в автомобільної галузі.

Таким чином важливими етапами подальшого функціонального покращення розробленого продукту повинні стати нижченаведені кроки.

1) По мірі росту продукту планується додавати підтримку більшої кількості вбудованих алгоритмів машинного навчання для можливості навчання моделей змішаним способом. Це буде зроблено засобами інтеграції з іншими бібліотеками, які надають API для використання засобів машинного навчання. Модульність додатка сприятиме успішній інтеграції. Це дозволить підвищити ефективність навчання за рахунок надання можливості різнобічних експериментальних досліджень.

2) Для підвищення зручності користування програмою планується додати можливість навчати моделі через браузер. Планується досягти це завдяки технології WebGL, яка підтримується Unity. Це технологія, яка дозволяє скомпілювати проект Unity у форматі HTML5/Javascript. Unity використовує технологію IL2CPP для того, щоб конвертувати .NET C# скрипти в WebAssembly (Wasm) [40].

3) В пункті 2.1 другого розділу даної роботи було зауважено, що компанії, які оперують в автомобільній галузі широко використовують автозгенерований код та можливості моделювання в програмах MATLAB\Simulink, тому для надання можливості практичного використання симуляції в майбутньому для комерційних цілей необхідно додати інтеграцію симулятора з вищенаведеним програмним забезпеченням. Як варіант це можливо зробити через TCP або UDP протоколи. Наприклад, використовуючи TcpListener класс у просторі імен System.Net.Sockets в C# скрипті всередині Unity, можна прослуховувати порт, що відкритий у Matlab. Потрібно спочатку запустити ігрову симуляцію в Unity, а потім запустити виконання в Matlab, результатом чого буде обмін сигналами [38].

На рис. 4.3 наведено розроблену архітектуру програмного забезпечення для навчання та тестування самокерованих транспортних засобів з майбутніми функціональними покращеннями. В додатку Е представлено схему розробленої архітектури на основі компонентно-орієнтованого підходу. Використана діаграма компонентів UML.

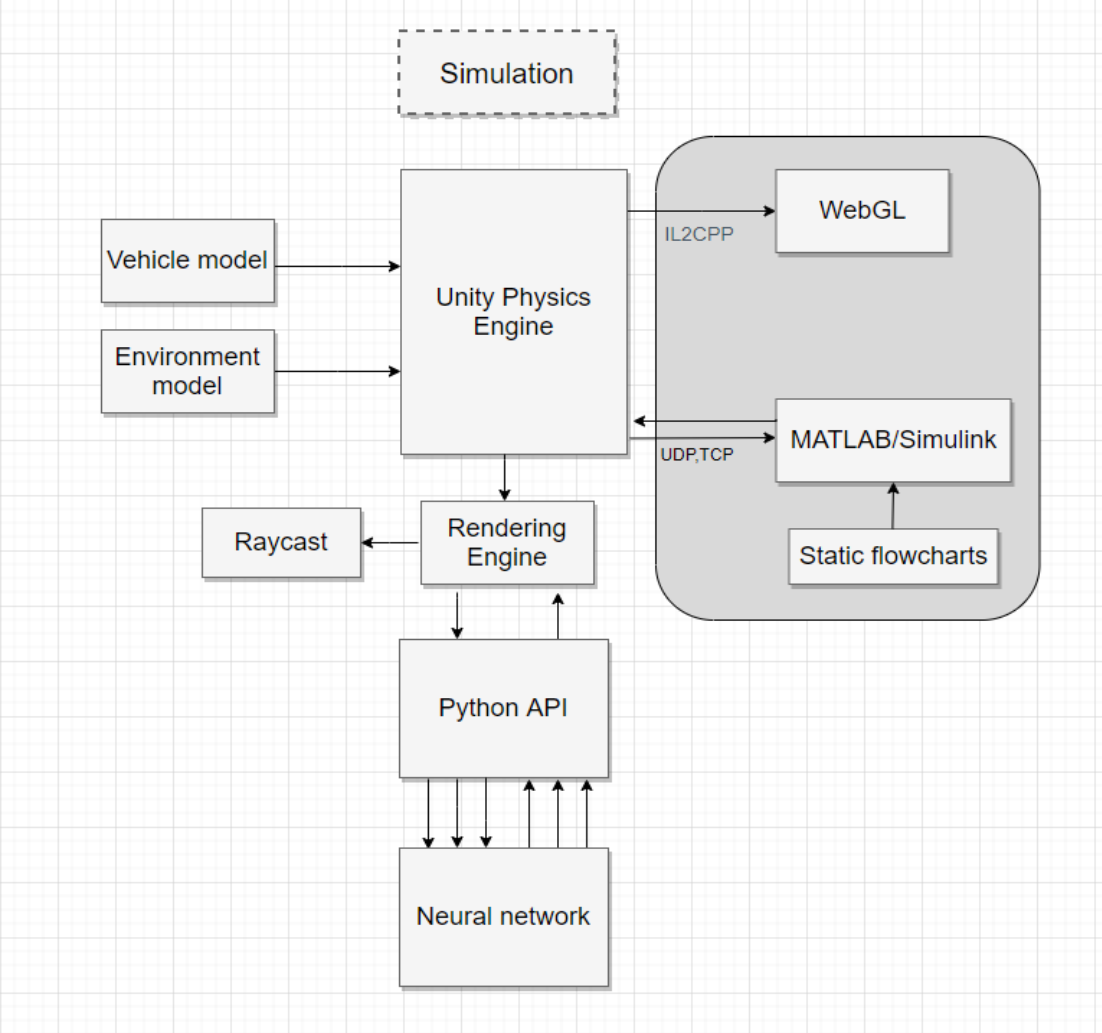


Рисунок 4.3 – Схема розробленої архітектури програмного забезпечення для навчання та тестування самокерованих транспортних засобів

4.2 Порівняння розробленого архітектурного рішення з аналогами для обґрунтування його ефективності

Таким чином для того, щоб зробити висновки щодо ефективності розробленого рішення, потрібно порівняти основні ключові показники розробленого програмного забезпечення з проаналізованими аналогами з огляду на їх переваги та недоліки. Це дозволить зрозуміти, чи було досягнуто мету дослідження, а також виокремити слабкі сторони розробленого програмного забезпечення для подальших покращень.

Детальна таблиця порівняння представлена в додатку Г.

Таким чином можна зробити висновок, що порівняно з аналогами розроблене програмне забезпечення має нижчі вимоги до апаратного забезпечення, тому що симулятор CARLA потребує 130 ГБ дискового простору, а розроблене ПЗ – всього 5MB. Крім того AirSim потребує 64GB RAM, в той час як розроблене ПЗ – 4-8GB. Вимоги до графічного процесору також значно нижчі, ніж у аналогів.

Для того щоб розпочати роботу з симулятором, наприклад, документація симулятора CARLA наголошує на необхідності великої кількості додаткового програмного забезпечення (наведено у додатку Г), а лише компіляція проекту може зайняти до 4 годин. Розроблений симулятор компілюється за 2 хвилини (якщо використовувати вихідний код), а якщо використовувати виконуваний файл, то для початку роботи з симулятором взагалі не потрібно встановлювати додаткове програмне забезпечення. Лише для навчання моделей потребується встановлення Python, PyTorch та пакет mlagents, що відбувається достатньо швидко.

Щодо зручності використання, то аналіз репортів користувачів на офіційних сторінках симулятору CARLA показав, що користувачі мають труднощі зі встановленням даного програмного забезпечення, необхідності детального вивчення документації перед початком роботи, існують проблеми крешей серверу Carla після недовгої роботи, неможливість або складність встановити комунікацію між клієнтом-сервером та ін. Розроблене рішення позбавлене даних недоліків, тому що має інтуїтивно зрозумілий інтерфейс та локально розгортається без необхідності встановлення комунікації з іншими компонентами. Запуск відбувається натисканням на бінарний файл або через командну строку. Модульний підхід до побудови архітектури дозволить легко додавати додатковий функціонал, не порушуючи існуючу логіку. Також система додавання нових сцен в Unity робить можливість дублювати сцени, робити невеликі правки та додавати нові сцени до налаштувань компіляції. Таким чином модульна архітектура розробленого рішення має більшу ефективність ніж клієнт-серверна – симулятора CARLA.

Головною істотною перевагою розробленого рішення є те, що існує вбудована підтримка методів машинного навчання, непотрібно реалізовувати власні нейронні мережі, можна приступити відразу до навчання лише регулюючі параметри конфігураційного файлу. У аналогів така підтримка відсутня. Але користувачу також надається можливість написати свою кастомну нейронну мережу в редакторі Unity та таким же чином використовувати симулятор.

Важливим аспектом, що підвищує ефективність навчання є те, що розроблене рішення не передбачає достатню деталізованість оточуючого середовища, завдяки чому досягається балансування між необхідними важливими фізичними характеристиками для навчання та непотрібними занадто деталізованими елементами, які не впливають на процес навчання. Аналоги є занадто деталізованими, а тому велика кількість обчислювальних ресурсів направлена на обрахунок фізичних даних та рендеринг середовища, а не на обчислення даних для навчання.

Було також виділено те, що поки що відсутня детальна документація для розробленого рішення, у той час як у аналогів вона на достатньо високому рівні. Тому планується розробити детальну документацію.

Висновки по розділу

У четвертому розділі було узагальнено та детально описано готове архітектурне рішення програмного забезпечення для навчання та тестування самокерованих транспортних засобів. Наведено схему та охарактеризовано процес обробки даних для механізму прийняття рішень в розробленому середовищі моделювання. Охарактеризовано процес розгортання архітектурного рішення програмного забезпечення, який підкріплено відповідним графічним матеріалом. Розглянуто перспективи подальших функціональних покращень розробленого продукту та практичні можливості його застосування.

Таким чином запропоноване рішення дозволило досягти поставленої мети дослідження, а саме: підвищити ефективність навчання самокерованих транспортних засобів в середовищі моделювання за рахунок покращення користувацької взаємодії. Такий висновок можна зробити завдяки проведеному порівняльному аналізу розробленого архітектурного рішення з аналогами, на основі якого було обґрунтовано кращі можливості користувацької взаємодії з розробленим програмним забезпеченням, та як наслідок більш ефективний процес навчання моделей автономних транспортних засобів в середовищі моделювання.

5 МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЕКТУ

5.1 Опис ідеї проекту

По результатам проведеного дослідження було знайдено потребу в оригінальній розробці архітектури програмного забезпечення, що давало б можливість навчати та тестувати самокеровані транспортні засоби найбільш ефективним чином за рахунок покращеної користувацької взаємодії.

Для досягнення мети було розроблено архітектуру програмного забезпечення, що, спираючись на можливості ігрової платформи Unity та інтеграцію бібліотеки ML-Agents для машинного навчання, дозволяє покращити користувацьку взаємодію, що в свою чергу відображається позитивним чином на ефективності навчання моделей. У таблиці 5.1 представлено опис ідеї стартап-проекту.

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
На основі розробленої архітектури було розроблено середовище моделювання, що дозволяє навчати моделі автотранспорту засобами машинного навчання, не потребуючи великих обчислювальних ресурсів, дорогого апаратного забезпечення та полегшує процес користувацької взаємодії.	Тестування моделей автопілотів перед тестуванням на міських дорогах Використання симулятора в дослідницьких цілях для навчання моделей засобами машинного навчання	Симулятор не потребує значних обчислювальних або апаратних ресурсів, займає до 5Мб місця на диску, надає можливість обирати алгоритми машинного навчання, регулювати параметри, моніторити процес навчання в 3D та графічній формах за допомогою TensorBoard. Він простий для розуміння, не потребує складної документації та встановлення додаткового ПЗ та може стати важливим проміжним етапом для подальшого навчання згенерованих моделей в деталізованих середовищах як Carla, для більш реалістичної поведінки автопілота. Головною перевагою симулятора є те, що завдяки

Продовження таблиці 5.1

		покращеній користувацькій взаємодії та можливості в реальному часі впливати на процес навчання досягається підвищення ефективності навчання.

У таблиці 5.2 проведено порівняльний аналіз основних техніко-економічних характеристик ідеї з товарами-аналогами.

Таблиця 5.2 – Порівняльна характеристика з товарами-аналогами

№	Техніко-економічні характеристики ідеї	Продукція конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	CARLA	AirSim			
1	Системні вимоги	Низькі	Високі	Середні			+
2	Зручність використання	Висока	Низька	Середня			+
3	Програмні залежності	Низькі	Високі	Середні			+
4	Спосіб запуску	Виконуваний файл	Клієнт-сервер	Плагін			+
5	Кросплатформеність	Так	Так	Так		+	
6	Основний ігровий рушій	Unity	UE	UE		+	
7	Вбудований компонент машинного навчання	Так	Ні	Ні			+
8	Документація	Ні	Так	Так	-		
9	Підтримка мов для написання скриптів	C#, Python	Python	Python, C++, C#, Java		+	
10	Деталізованість середовища	Середня	Висока	Висока			+
11	Тип архітектури	Модульна	Клієнт-серверна	Модульна (плагін)			+

5.2 Технологічний аудит ідеї проекту

Для аналізу технологічної здійсненності ідеї було проведено дослідження технологій її реалізації (табл. 5.3) та обрано найбільш оптимальний варіант.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Графічна платформа	Unity	Наявна	Доступна
		Unreal Engine	Наявна	Доступна
		Cocos	Наявна	Доступна
Обрана технологія реалізації ідеї проекту: Unity				
2	Засоби машинного навчання	Python API	Наявна	Доступна
		Unity ML-Agents	Наявна	Доступна
		mlpack	Наявна	Доступна
Обрана технологія реалізації ідеї проекту: Unity ML-Agents				
4	Мова програмування	C++	Наявна	Доступна
		C#	Наявна	Доступна
		Python	Наявна	Доступна
Обрана технологія реалізації ідеї проекту: C#				

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Для того щоб розуміти ринок, на якому буде збуватися продукт, потрібно провести аналіз потенційного ринку (табл. 5.4).

Таблиця 5.4 – Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	2
2	Загальний обсяг продаж, грн./ум.од	107.000$
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Високі стандарти щодо безпеки Висококонкурентна галузь Необхідність високих матеріальних та людських затрат Мала кількість високодосвідчених інженерів

Продовження таблиці 5.4

5	Специфічні вимоги до стандартизації та сертифікації	Немає стандартів щодо сертифікації самого додатку, але якщо симуляція не буде відповідати наявним стандартам дорожньої безпеки та ISO 26262 товар буде непотрібний
6	Середня норма рентабельності в галузі або по ринку, %	24,56%

Висновок: враховуючи кількість головних гравців по ринку, зростаючу динаміку ринку, невелику кількість конкурентів та середню норму рентабельності можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим.

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Можливість навчати та тестувати авто як проміжний етап тестування	Вітчизняні аутсорсингові компанії, що надають послуги найбільшим автовиробникам	Очікують продукт, який надає можливості якнайшвидше розробити якісний продукт для кінцевого замовника та не потребує багато часу на вивчення	Споживачі очікують, що продукт матиме простий інтерфейс, відкритий вихідний код, можливість навчати моделі та зберігати результати навчання
2	Можливість навчати та тестувати авто перед запуском в реальний світ	Закордонні автомобільні компанії	Очікують продукт, що відповідає стандарту функціональної безпеки ISO 26262 та надає можливості пришвидшити розробку кінцевого продукту	

Продовження таблиці 5.5

3	Можливість навчати та тестувати авто для дослідницьких цілей	Університети та інші дослідницькі установи	Очікують простий та дешевий продукт, що не потребує значних обчислювальних можливостей, але дає змогу використовувати алгоритми машинного навчання для навчання автопілотів

Таблиця 5.6 – Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1.	Висока конкуренція	Інші компанії, що пропонують подібні симуляції	Доповнювати функціональність, додавати підтримку різних алгоритмів машинного навчання, додавати деталі середовища, сенсори, інтеграцію з іншими модулями, в яких є потреба
2.	Високі витрати	Неможливість прорахувати всі витрати на першому етапі, а також рівень попити	Консультація з провідними фінансовими консультантами
3.	Невелика лояльність та недовіра закордонних споживачів	Великі світові OEM можуть з недовірою відноситися до невідомої маленької компанії з України	Праця над брендом компанії

Таблиця 5.7 – Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Великий попит через низьку оплату праці в Україні	Нові клієнти	Пропонувати більші функціональні можливості
2	Низький рівень оподаткування в Україні	Нові інвестори	Залучення додаткових інвестицій

Продовження таблиці 5.7

3	Високий професійний рівень українських спеціалістів	Залучення наших спеціалістів до кооперації	Робота над новими неосвоєними технологіями разом з закордонними компаніями

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: Чиста	Багато конкурентів	Кооперація
2	Рівень конкурентної боротьби: міжнародний рівень	Конкуренти по всьому світу	Проводити залучення клієнтів по всьому світу
3	Галузева ознака: міжгалузева	Конкуренція в автомобільній промисловості, а також іншими індустріями, які застосовують симуляції	Освоєння нових сфер застосування симуляції
4	Конкуренція за видами товарів: Товарно-видова	Конкуренція між товарами різного виду	Додавання нового функціоналу
5	Характер конкурентних переваг: Нецінова	Конкуренція проводиться за рахунок удосконалення якості продукту	Вдосконалення продукту
6	За інтенсивністю: Немарочна	Роль торгової марки незначна	Заохочення клієнтів якістю товару, а не брендом.

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Компанії, що надають послуги у сфері машинного навчання, автомобільні симуляції	Бар'єри входження високі, треба мати співробітників з високим професійним рівнем та великі інвестиції	Новітні актуальні технології	Намагання як можна скоріше досягти автономності автомобілів любими засобами, щоб бути першими на ринку	Товари-замінники лише дозволять збільшити прогрес, через кооперацію стратегію розвитку
Висновки	Висока здорова конкуренція	Можливості входу високі, через велику актуальність технологій	Постачальники не диктують умови, все відбувається завдяки гонці технологій	Клієнти бажають якнайскоріше досягти автономності автомобілів, отже дуже прискіпливо відносяться до симуляції	Товари-замінники дозволять налагодити кооперацію та більш швидкий розвиток нових технологій

Проведений аналіз ринкових можливостей запуску стартап-проекту дозволив зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим, потенційними клієнтами продукту є вітчизняні аутсорсингові компанії, що надають послуги найбільшим автовиробникам, закордонні автомобільні компанії та університети та інші дослідницькі установи.

Хоча існують фактори загроз, але фактори можливостей дозволяють оптимістично оцінювати запуск проекту. Аналіз конкуренції в галузі показав, що можливості входу високі, через велику актуальність технологій та застосуванню кооперацийної стратегії.

У таблиці 5.10 обґрунтовано фактори конкурентоспроможності товару.

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Актуальність товару	Застосування новітніх технологій, що привертають увагу споживачів
2	Відкритий вихідний код	Бажання працювати над продуктом великої кількості розробників зі всього світу
3	Висока готовність до кооперації	Відкритість до діалогу
4	Простота використання	Зрозумілий інтерфейс, можливість запуску ПЗ через .exe файл без додаткових програмних залежностей
5	Можливість використання не тільки користувачами з глибокими знаннями	Використання не потребує великих знань у галузі програмування та можно розпочати тренувати моделі без відповідної глибинної підготовки
6	Кросплатформеність	Можливість використовувати на різних платформах
7	Можливість обирати методи машинного навчання для підвищення ефективності навчання	Аналоги не надають рішення «прямо з коробки», потрібно самому писати нейронні мережі для навчання

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін програмного забезпечення для навчання та тестування самокерованих транспортних засобів

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з запропонованим						
			-3	-2	-1	0	+1	+2	+3
1	Актуальність товару	15				+			
2	Відкритий вихідний код	19				+			

Продовження таблиці 5.11

3	Висока готовність до кооперації	17				+			
4	Простота використання	18	+						
5	Можливість використання не тільки користувачами з глибокими знаннями	19		+					
6	Кросплатформеність	10				+			
7	Можливість обирати методи машинного навчання для підвищення ефективності навчання	20	+						

Таким чином можна зробити висновок, що розроблений продукт має значні переваги перед товарами-конкурентами та може бути з високою вірогідністю комерціалізований.

У таблицях 5.12 та 5.13 наведено характеристику слабких та сильних сторін продукту, а також альтернатив ринкового впровадження.

Таблиця 5.12 – SWOT аналіз стартап-проекту

Сильні сторони (S):	Слабкі сторони (W):
Додаток дозволяє протестувати та навчити автомобіль рухатися врегульовано без використання реальних дорожніх умов	Необхідність інвестицій, сильного позиціонування компанії та підвищення зацікавленості через високий поріг входу до індустрії
Можливості (O): – Багатозадачність – Розширюваність – Відкритий вихідний код – Реалістичне зображення	Загрози (T): – Недовіра споживачів – Необхідність висококваліфікованих спеціалістів – Світова криза, що дуже сильно вплинула на автомобільну індустрію

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Державні заходи, реклама в інтернеті, адресні розсилки	Головний ресурс – гроші, даний ресурс – необхідне залучення	5-8 місяців
3	Участь у саммітах з презентацією продукту	Головний ресурс – час, даний ресурс - наявний	2-3 тижні
4	Вести YouTube канал, що демонструє, як можна навчати моделі, записувати туторіали	Ресурс – час, наявний	1-3 місяці

5.4 Розроблення ринкової стратегії проекту

Було проведено вибір цільових груп споживачів, на яких орієнтовано продукт та визначено простоту входу у сегмент.

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Вітчизняні автомобільні компанії	Низька, отже вітчизняний автопром ще недостатньо розвинутий	40% - низький попит	Низька	Важко через недостатню розвиненість ринку
2	Закордонні автомобільні компанії	Висока, тому що активно використовують симуляції для тестування своїх автомобілів	80% - високий попит	Висока	Треба докласти зусиль, але вірогідність входу висока

Продовження таблиці 5.14

3	Університети та інші дослідницькі установи	Середня, тому що є недостатнє фінансування	60% - середній попит	Низька	Легко, бо дослідницькі установи широко використовують новаторські продукти
4	Вітчизняні аутсорсингові компанії, що надають послуги найбільшим автовиробникам	Висока, тому що активно надають послуги закордонним автовиробникам	90% - високий попит	Висока	Можна ввійти у сегмент, розробивши гарну рекламну компанію
Які цільові групи обрано: закордонні автомобільні компанії, вітчизняні аутсорсингові компанії, що надають послуги найбільшим автовиробникам, дослідницькі установи					

Таблиця 5.15 – Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Розвиток продукту та постійне оновлення функціоналу по мірі розвитку технологій	Маркетинг нових ринків	Активний розвиток технологій штучного інтелекту та жага виробити автономний автомобіль буду сприяти розвитку продукту	Стратегії концентрованого зростання

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія конкурентної поведінки
Ні	Буде шукати нових споживачів	Буде використовувати вже існуючі програмні рішення, але застосовувати та розроблювати нові модулі, буде все більше розширювати інтеграцію з іншими модулями	Кооперацийна стратегія

Таблиця 5.17 – Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Здатність системи до розширювання	Стратегії диверсифікованого зростання	Система буде побудована модульним способом, що дозволить легко додавати нові модулі без зміни існуючих	модульність відкритість до змін гнучкість
2	Багатозадачність	Стратегії концентрованого зростання	З самого початку система буде спрямована на виконання широкого спектру функцій	виконання різних функцій ефективність актуальність

Продовження таблиці 5.17

3	Можливість заміни різних платформ, що використовує система	Стратегії диверсифікованого зростання	Система буде розроблена з позиції принципів проектування, що дозволять клієнтам застосовувати різні платформи для інтеграції вже з існуючими продуктами	кросплатформеність економічна вигода залучення більшої кількості клієнтів

Відповідно до проведеного аналізу можна зробити висновок, що стартап-компанія вибирає як базову стратегію розвитку – стратегію концентрованого зростання, як базову стратегію конкурентної поведінки – кооперацийну стратегію.

5.5 Розроблення маркетингової програми стартап-проекту

Розроблення маркетингової програми передбачає визначення ключових переваг концепції потенційного товару, опис трьох рівнів моделі товару, визначення меж встановлення ціни, формування системи збуту та вибір концепції маркетингових комунікацій. Результати представлено в таблицях нижче.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Безпека дорожнього руху	Тестування руху авто у симуляції підвищить безпеку на дорогах	Висока задіяність технологій, що зазвичай використовуються в непов’язаних сферах та дозволить підвищити безпеку
2	Використання новітніх технологій прийняття рішень	Прийняття рішень автомобілем підвищиться	Методики машинного навчання підвищать точність прийняття рішень

Продовження таблиці 5.18

3	Точний розрахунок траєкторій руху	Розроблення точних карт руху	Підвищена точність призведе до автопілотов, що здатні рухатися в різних умовах

Таблиця 5.19 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>1. Товар за задумом</td><td colspan="3">Симуляція, що здатна забезпечити навчання та тестування автомобілів для розвитку самокерованості</td></tr>
<tr><td rowspan="7">2. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх/Тл/Е/Ор</td></tr>
<tr><td>1. Зручний інтерфейс
2. Багатофункціональність
3. Кросплатформеність
4. Деталізована графіка</td><td>Нм
М
Нм
М</td><td>Тх/Е/Ор
Тл
Тх
Тх/Ор</td></tr>
<tr><td colspan="3">Якість: відповідає параметрам безпеки міжнародного комітету SAE</td></tr>
<tr><td colspan="3">Пакування: Характеристики продукту будуть оформлено в офіційній документації, що буде доступна на GitHub</td></tr>
<tr><td colspan="3">Марка: AI Engineering SimulaAI</td></tr>
<tr><td rowspan="2">3. Товар із підкріпленням</td><td colspan="3">До продажу: Програмний додаток з відкритим вихідним кодом</td></tr>
<tr><td colspan="3">Після продажу: постійне розширення функціоналу, підтримка продукту</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання:

Це товар з відкритим вихідним кодом, тому має на меті забезпечити високу кооперацію зацікавлених у розвитку технологій без обмежень права власності.</td></tr>
</table>

**М/Нм – монотонні/немонотонні; *Вр/Тх/Тл/Е/Ор – вартісні/ технічні/ технологічні/ ергономічні/ органолептичні*

Таблиця 5.20 – Визначення меж встановлення ціни

Рівень цін на товари-	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на
Безкоштовні	500 - 1000$	Середній для товару з відкритим вихідним кодом і високий для можливої комерційної версії продукту	300 – 700$ для комерційної версії

Таблиця 5.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Цільові клієнти можуть долучитися до розвитку товару безоплатно, купити готовий проект у виробника або посередників	- аналіз ринку - дослідження ринків для розширення - аналіз законодавчих норм різних країн	Багатоканальний розподіл	Міжнародні торгівельні площадки

Таблиця 5.22 – Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Клієнти зацікавлені особи, що відвідують масові заходи присвячені технологіям	Міжнародні конференції, виставки, саміти	Відкритість коду, великі інвестиції, загальносоціальна мета	Залучити зацікавлених осіб	Майбутнє вже близько, треба тільки об’єднати зусилля.

Таким чином було доведено, що продукт задовольняє висунуті вимоги та потреби, було описано товар у реальному виконанні та товар із підкріпленням. Щодо ціни товару, то передбачається, що товар буде у вільному доступі з відкритим вихідним кодом, але можлива комерційна версія програми для залучення додаткових коштів на розширення функціональності. Планується використовувати найбільшу кооперацію та залучати всіх зацікавлених осіб для пришвидшення розвитку технологій галузі.

Висновки по розділу

В четвертому розділі було охарактеризовано ідею стартап-проекту, визначено можливість її комерціалізації. Було зроблено висновок, що товар є достатньо актуальним, можливість входу на ринок є високою, попит на товар буде великим, а конкуренція може стати позитивним чинником та буде добре впливати на розвиток продукту.

Було виокремлено переваги та недоліки продукту, загрози для його впровадження та підтримки. Було обрано цільові групи споживачів, якими можуть стати вітчизняні аутсорсингові компанії, що надають послуги найбільшим автовиробникам, закордонні автомобільні компанії та університети та інші дослідницькі установи. Було визначено стратегії розвитку та конкурентної поведінки, а також можливості позиціонування продукту. Аналіз показав, що реалізація проекту є доцільною.

ВИСНОВКИ

У ході роботи над магістерською дисертацією було розглянуто головні проблеми автомобільної індустрії сьогодення та на основі аналізу предметної області було зроблено висновок, що одним з актуальних напрямків для дослідження є проблема розробки програмного забезпечення для транспорту, що оснащено системами автоматизації водіння. Аналіз літературних джерел з цієї предметної області дозволив виокремити конкретні проблеми, які перешкоджають на даний момент випустити на міські дороги автотранспорт з п’ятим рівнем автономності за класифікацією SAE, тобто повністю самокеровані транспортні засоби.

Таким чином найбільш актуальним питанням постала проблема прийняття рішень самокерованими транспортними засобами в умовах невизначеності, а саме: діяти в середовищі з невизначеною кількістю змінюваних умов навколишнього середовища (таких як погодні умови, географічні ландшафти, статичні та динамічні об’єкти та ін.).

На основі дослідження предметної області та існуючих рішень було зроблено висновок, що для вирішення подібних задач найбільш оптимальним підходом є використання середовища моделювання, а також засобів машинного навчання для навчання та тестування самокерованих транспортних засобів. Аналіз існуючих рішень дозволив виокремити основні переваги та недоліки. Було зроблено висновок, що за рахунок існуючих проблем в користувацькій взаємодії з проаналізованим програмним забезпеченням ефективність навчання моделей самокерованого транспорту знижується. На основі проведеного аналізу було сформульовано основну мету та задачі дослідження. Основною метою стало підвищення ефективності навчання та тестування самокерованих транспортних засобів в програмованому середовищі за рахунок покращеній користувацькій взаємодії з програмним забезпеченням.

Для досягнення мети було розроблено архітектуру програмного забезпечення, що, спираючись на можливості ігрової платформи Unity та інтеграцію бібліотеки ML-Agents для машинного навчання, дозволяє покращити

користувацьку взаємодію, що в свою чергу відображається позитивним чином на ефективності навчання моделей.

На основі розробленої архітектури було розроблено середовище моделювання, що дозволяє навчати моделі автотранспорту засобами машинного навчання, не потребуючи великих обчислювальних ресурсів, дорогого апаратного забезпечення та полегшує процес користувацької взаємодії. Було проведено порівняльний аналіз розробленого архітектурного рішення з аналогами для обґрунтування його ефективності.

Було проведено маркетинговий аналіз стартап-проекту згідно з яким було обрано як базову стратегію розвитку – стратегію концентрованого зростання, а як базову стратегію конкурентної поведінки – кооперaційну стратегію. Було зроблено висновок, що існує достатньо висока вірогідність успіху продукту на ринку.

Статтю, що описує етапи розробки даного програмного забезпечення, було опубліковано в науковому збірнику матеріалів з напрямку програмної інженерії.

В результаті проведеного дослідження можна виокремити наукову новизну та практичну значимість одержаних результатів:

Наукова новизна результатів магістерської дисертації полягає в тому, що *набули подальшого розвитку* використання алгоритмів машинного навчання шляхом інтеграції відповідних засобів машинного навчання за допомогою розробки програмного забезпечення з використанням підходу компонентно-орієнтованого проектування.

Практичне значення отриманих результатів полягає у створенні архітектури простішого, більш дешевого програмного забезпечення з відкритим вихідним кодом, що використовує засоби штучного інтелекту для підвищення ефективності прийняття миттєвих рішень самокерованими транспортними засобами задля уникнення ризиків непередбачуваного розвитку подій ще на етапі тестування, та впровадженими принципами модульності для покращення можливостей інтеграції з елементами інших систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Simulated environments for autonomous vehicle training. URL: https://unity3d.com/simulated-environments-for-autonomous-vehicle-training (дата звернення 15.09.2020).

2) Alexander Amini, Igor Gilitschensk. Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation. *IEEE Robotics and Automation Letters*, vol. 5, no. 2, April 2020. DOI: 10.1109/LRA.2020.2966414.

3) Du, N., Zhou, F., Pulver, E., Tilbury, D. et al. Predicting Takeover Performance in Conditionally Automated Driving. *Conference: 38rd ACM Conference on Human Factors in Computing Systems*. 2020. P. 1-8. DOI: https://doi.org/10.1145/3334480.3382963.

4) W. Li, C. W. Pan, R. Zhang, J. P. Ren et al. AADS: Augmented autonomous driving simulation using data-driven algorithms. *Science Robotics*. 2019. DOI: 10.1126/scirobotics.aaw0863.

5) CARLA simulator. URL: https://carla.org/ (дата звернення 20.09.2020).

6) Microsoft Airsim now available on Unity. URL: https://www.microsoft.com/en-us/research/blog/microsoft-airsim-now-available-on-unity/ (дата звернення 25.09.2020).

7) Self driving cars. Drive constellation. URL: https://www.nvidia.com/ru-ru/self-driving-cars/drive-constellation/ (дата звернення 05.10.2020).

8) Inside Waymos secret testing and simulation facilities. URL: https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/ (дата звернення 07.10.2020).

9) Waymo engineer explains why testing self-driving cars virtually is critical. URL: https://www.businessinsider.com/waymo-engineer-explains-why-testing-self-driving-cars-virtually-is-critical-2018-8 (дата звернення 10.10.2020).

10) Nvidia drive constellation datasheet. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/self-driving-cars/drive-constellation/nvidia-drive-constellation-datasheet-2019-oct.pdf (дата звернення 12.10.2020).

11) Zhenpei Yang, Yuning Chai, Dragomir Anguelov, Yin Zhou et al. SurfelGAN: Synthesizing Realistic Sensor Data for Autonomous Driving. *Conference: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. DOI: 10.1109/cvpr42600.2020.01113.

12) CARLA. URL: https://carla.readthedocs.io/en/latest/start_introduction/ (дата звернення 15.10.2020).

13) Alexey Dosovitskiy et al. CARLA: An Open Urban Driving Simulator. In: CoRL. 2017. P. 1–16. issn: 1938-7228. arXiv: 1711.03938. URL: http://arxiv.org/abs/1711.03938 (дата звернення 17.10.2020).

14) Shital Shah et al. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In: Field and Service Robotics. 2017. P. 3–4. URL: https://arxiv.org/abs/1705.05065 (дата звернення 05.11.2020).

15) Welcome to AirSim. URL: https://microsoft.github.io/AirSim/ (дата звернення 10.11.2020).

16) V-Model. URL: https://uk.wikipedia.org/wiki/V-Model#cite_note-FHWA_05-3 (дата звернення 14.11.2020).

17) Best Practices for Establishing a Model-Based Design. URL: https://www.researchgate.net/publication/228888357_Best_Practices_for_Establishing_a_Model-Based_Design_Culture (дата звернення 12.11.2020).

18) Math. Graphics. Programming. URL: https://ch.mathworks.com/products/matlab.html (дата звернення 10.01.2021).

19) Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. URL: https://www.sae.org/standards/content/j3016_201806/ (дата звернення 15.01.2021).

20) SAE Levels of Driving Automation™ Refined for Clarity and International Audience. URL: https://www.sae.org/blog/sae-j3016-update (дата звернення 15.01.2021).

21) How HARA Helps Functional Safety (ISO 26262) Consultants to Determine ASIL Values and Formulate Safety Goals. URL:

https://www.embitel.com/blog/embedded-blog/hara-by-iso-26262-standard-for-your-functional-safety-project (дата звернення 21.01.2021).

22) How to Comply With the ISO 26262 Standard. Software Development and Design Requirements For Every ASIL. URL: https://www.perforce.com/resources/qac/how-comply-iso-26262-standard (дата звернення 21.01.2021).

23) Static Code Analysis for Automotive Software. URL: https://www.researchgate.net/publication/350429010_Static_Code_Analysis_for_Automotive_Software (дата звернення 25.01.2021).

24) Artificial Intelligence Reshaping the Automotive Industry. URL: https://www.futurebridge.com/industry/perspectives-mobility/artificial-intelligence-reshaping-the-automotive-industry/ (дата звернення 30.01.2021).

25) Richard S. Sutton and Andrew G. Barto Reinforcement Learning: An Introduction / Richard S. Sutton and Andrew G. Barto — Second edition. — A Bradford Book. The MIT Press, 2018, 2020.

26) Motion planning. URL: https://en.wikipedia.org/wiki/Motion_planning (дата звернення 07.02.2021).

27) Introduction to Mobile Robotics. Robot Motion Planning. Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras. 2011. URL: http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf (дата звернення 14.02.2021).

28) Self-Driving Cars Specialization by the University of Toronto. URL: https://www.coursera.org/learn/motion-planning-self-driving-cars/lecture/xumaJ/lesson-1-driving-missions-scenarios-and-behaviour (дата звернення 22.02.2021).

29) Finite-state machine // Вікіпедія: вільна енциклопедія. URL: https://en.wikipedia.org/wiki/Finite-state_machine (дата звернення 28.02.2021).

30) Automata theory // Вікіпедія: вільна енциклопедія. URL: https://en.wikipedia.org/wiki/Automata_theory (дата звернення 28.02.2021).

31) Thomas M. Moerland, Joost Broekens, Catholijn M. Jonker. Model-based Reinforcement Learning: A Survey. URL: https://arxiv.org/abs/2006.16712 (дата звернення 12.03.2021).

32) Pieter Abbeel, Andrew Y. Ng. Inverse Reinforcement Learning. *Encyclopedia of Machine Learning*. Springer, Boston, MA. 2011. DOI: https://doi.org/10.1007/978-0-387-30164-8_417.

33) What game engines do you currently use? URL: https://www.statista.com/statistics/321059/game-engines-used-by-video-game-developers-uk/ (дата звернення 21.03.2021).

34) List of game engines. URL: https://en.wikipedia.org/wiki/List_of_game_engines (дата звернення 22.03.2021).

35) Arthur Juliani et al. Unity: A General Platform for Intelligent Agents. 2020. URL: https://arxiv.org/abs/1809.02627 (дата звернення 27.03.2021).

36) System requirements for Unity 2020 LTS. URL: https://docs.unity3d.com/Manual/system-requirements.html (дата звернення 28.03.2021).

37) WheelCollider. URL: https://docs.unity3d.com/ScriptReference/WheelCollider.html (дата звернення 11.04.2021).

38) Іванчевська Д.В. Етапи розробки програмного забезпечення для навчання та тестування транспорту з вбудованою підтримкою системи самокерування/ Іванчевська Д.В. // Матеріали Першої Всеукраїнської науково-практичної конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021). Секція кафедри інформатики та програмної інженерії. Матеріали конференції. – Київ. – 2021. 22–26 листопада 2021р. – С.199.

39) ONNX models. URL: https://docs.microsoft.com/en-us/windows/ai/windows-ml/get-onnx-model (дата звернення 23.06.2021).

40) Getting started with WebGL development. URL: https://docs.unity3d.com/Manual/webgl-gettingstarted.html (дата звернення 03.08.2021).

ДОДАТКИ

ДОДАТОК А

Порівняння основних графічних 3D платформ

Назва	Написана з допомогою мови програмування	Мова програмування для написання скриптів	Цільові платформи
Cocos2d-x	C++, Python, Objective-C, JavaScript	C++, JavaScript, Java, Lua	Windows, Linux, macOS, iOS, Android, BlackBerry, Tizen
Unity 3D	C++	C#, Visual Scripting (Bolt)	Windows, macOS, Linux, Xbox, PlayStation, Windows Phone, iOS, Android, BlackBerry, Tizen, Unity Web Player, Windows Store, WebGL, Oculus Rift, Gear VR, Android TV, Samsung Smart TV, Google Stadia
Unreal Engine	C++	C++, Blueprints	Підтримка практично всіх відомих
Lumberyard	C++	Lua	PlayStation 4, Xbox One, Windows
CryEngine	C++	C++, C#	Windows, macOS, Linux, PlayStation 3, PlayStation 4, Wii U, Xbox 360, Xbox One, iOS, Android
GameMaker Studio	GML	Game Maker Language, JavaScript, C++, GLSL, HLSL	Windows, Windows 8, Xbox 360, Xbox One, PlayStation 3, PlayStation 4, PlayStation Vita, macOS, Ubuntu, HTML5, Android, iOS, Windows Phone 8, Tizen, Amazon Fire TV, Nintendo Switch
PhyreEngine	C++	-	PC, PlayStation Portable, PlayStation Vita, PlayStation 3, PlayStation 4
PlayCanvas	JavaScript	JavaScript	Windows, Linux, macOS, iOS, HTML5, Android

ДОДАТОК Б

Системні вимоги для роботи з Unity 2020 LTS

Мінімальні вимоги	Windows	macOS	Linux (Support in Preview)
Версія операційної системи	Windows 7 (SP1+), Windows 10 та Windows 11, лише 64-bit.	High Sierra 10.13+	Ubuntu 20.04, Ubuntu 18.04, and CentOS 7
CPU	Архітектура X64 з підтримкою набору інструкцій SSE2	Архітектура X64 з підтримкою набору інструкцій SSE2	Архітектура X64 з підтримкою набору інструкцій SSE2
Графічний API	DX10, DX11, DX12 GPUs	Графічні процесори Intel і AMD з підтримкою Metal API	OpenGL 3.2+ чи Vulkan-підтримувані, Nvidia та AMD GPUs.
Додаткові вимоги	Драйвери, що офіційно підтримує постачальник обладнання	Драйвери, що офіційно підтримує постачальник обладнання	Середовище робочого столу Gnome, що працює поверх віконної системи X11, офіційного власного графічного драйвера Nvidia або графічного драйвера AMD Mesa. Інша конфігурація та користувальницьке середовище, з підтримуваним дистрибутивом (ядро, композитор тощо)

ДОДАТОК В

Приклад налаштування конфігурації та показників машинного навчання засобами пакету mlagents

```
[INFO] VehicleAgent. Step: 12000. Time Elapsed: 111.159 s. Mean Reward: 0.054. Std of Reward: 0.015. Training.
[INFO] VehicleAgent. Step: 24000. Time Elapsed: 213.718 s. Mean Reward: -0.114. Std of Reward: 0.379. Training.
[INFO] VehicleAgent. Step: 36000. Time Elapsed: 307.850 s. Mean Reward: -0.042. Std of Reward: 0.289. Training.
[INFO] VehicleAgent. Step: 48000. Time Elapsed: 412.745 s. Mean Reward: 0.045. Std of Reward: 0.021. Training.
[INFO] VehicleAgent. Step: 60000. Time Elapsed: 565.779 s. Mean Reward: 0.042. Std of Reward: 0.020. Training.
[INFO] VehicleAgent. Step: 72000. Time Elapsed: 704.696 s. Mean Reward: 0.030. Std of Reward: 0.011. Training.
[INFO] VehicleAgent. Step: 84000. Time Elapsed: 792.758 s. Mean Reward: 0.049. Std of Reward: 0.016. Training.
[INFO] VehicleAgent. Step: 96000. Time Elapsed: 880.579 s. Mean Reward: 0.044. Std of Reward: 0.024. Training.
[INFO] VehicleAgent. Step: 108000. Time Elapsed: 968.541 s. Mean Reward: 0.049. Std of Reward: 0.022. Training.
[INFO] VehicleAgent. Step: 120000. Time Elapsed: 1056.366 s. Mean Reward: 0.063. Std of Reward: 0.010. Training.
[INFO] VehicleAgent. Step: 132000. Time Elapsed: 1143.996 s. Mean Reward: 0.070. Std of Reward: 0.011. Training.
[INFO] VehicleAgent. Step: 144000. Time Elapsed: 1231.642 s. Mean Reward: 0.068. Std of Reward: 0.016. Training.
[INFO] VehicleAgent. Step: 156000. Time Elapsed: 1320.814 s. Mean Reward: 0.063. Std of Reward: 0.014. Training.
[INFO] VehicleAgent. Step: 168000. Time Elapsed: 1409.588 s. Mean Reward: 0.059. Std of Reward: 0.020. Training.
[INFO] VehicleAgent. Step: 180000. Time Elapsed: 1499.691 s. Mean Reward: 0.066. Std of Reward: 0.017. Training.
[INFO] VehicleAgent. Step: 192000. Time Elapsed: 1587.647 s. Mean Reward: 0.070. Std of Reward: 0.015. Training.
[INFO] VehicleAgent. Step: 204000. Time Elapsed: 1676.805 s. Mean Reward: 0.076. Std of Reward: 0.014. Training.
[INFO] VehicleAgent. Step: 216000. Time Elapsed: 1765.901 s. Mean Reward: 0.063. Std of Reward: 0.016. Training.
[INFO] VehicleAgent. Step: 228000. Time Elapsed: 1857.066 s. Mean Reward: 0.066. Std of Reward: 0.024. Training.
[INFO] VehicleAgent. Step: 240000. Time Elapsed: 1945.349 s. Mean Reward: 0.078. Std of Reward: 0.015. Training.
[INFO] VehicleAgent. Step: 252000. Time Elapsed: 2032.366 s. Mean Reward: 0.078. Std of Reward: 0.031. Training.
[INFO] VehicleAgent. Step: 264000. Time Elapsed: 2118.676 s. Mean Reward: 0.078. Std of Reward: 0.027. Training.
[INFO] VehicleAgent. Step: 276000. Time Elapsed: 2211.481 s. Mean Reward: 0.097. Std of Reward: 0.042. Training.
[INFO] VehicleAgent. Step: 288000. Time Elapsed: 2308.510 s. Mean Reward: 0.097. Std of Reward: 0.038. Training.
[INFO] VehicleAgent. Step: 300000. Time Elapsed: 2403.683 s. Mean Reward: 0.117. Std of Reward: 0.057. Training.
[INFO] VehicleAgent. Step: 312000. Time Elapsed: 2501.103 s. Mean Reward: 0.136. Std of Reward: 0.053. Training.
[INFO] VehicleAgent. Step: 324000. Time Elapsed: 2594.598 s. Mean Reward: 0.114. Std of Reward: 0.032. Training.
[INFO] VehicleAgent. Step: 336000. Time Elapsed: 2687.792 s. Mean Reward: 0.195. Std of Reward: 0.086. Training.
[INFO] VehicleAgent. Step: 348000. Time Elapsed: 2780.254 s. Mean Reward: 0.316. Std of Reward: 0.307. Training.
[INFO] VehicleAgent. Step: 360000. Time Elapsed: 2874.527 s. Mean Reward: 0.142. Std of Reward: 0.099. Training.
[INFO] VehicleAgent. Step: 372000. Time Elapsed: 2969.312 s. Mean Reward: 0.106. Std of Reward: 0.041. Training.
[INFO] VehicleAgent. Step: 384000. Time Elapsed: 3074.910 s. Mean Reward: 0.231. Std of Reward: 0.229. Training.
[INFO] VehicleAgent. Step: 396000. Time Elapsed: 3172.292 s. Mean Reward: 0.258. Std of Reward: 0.238. Training.
[INFO] VehicleAgent. Step: 408000. Time Elapsed: 3270.879 s. Mean Reward: 0.277. Std of Reward: 0.237. Training.
[INFO] VehicleAgent. Step: 420000. Time Elapsed: 3374.260 s. Mean Reward: 0.302. Std of Reward: 0.298. Training.
[INFO] VehicleAgent. Step: 432000. Time Elapsed: 3471.081 s. Mean Reward: 0.258. Std of Reward: 0.243. Training.
[INFO] VehicleAgent. Step: 444000. Time Elapsed: 3590.039 s. Mean Reward: 0.268. Std of Reward: 0.294. Training.
[INFO] VehicleAgent. Step: 456000. Time Elapsed: 3695.978 s. Mean Reward: 0.329. Std of Reward: 0.294. Training.
[INFO] VehicleAgent. Step: 468000. Time Elapsed: 3793.619 s. Mean Reward: 0.278. Std of Reward: 0.251. Training.
[INFO] VehicleAgent. Step: 480000. Time Elapsed: 3898.595 s. Mean Reward: 0.282. Std of Reward: 0.242. Training.
[INFO] VehicleAgent. Step: 492000. Time Elapsed: 4006.197 s. Mean Reward: 0.273. Std of Reward: 0.234. Training.
[INFO] Exported results\24Training\VehicleAgent\VehicleAgent-499060.onnx
[INFO] VehicleAgent. Step: 504000. Time Elapsed: 4107.307 s. Mean Reward: 0.474. Std of Reward: 0.292. Training.
[INFO] VehicleAgent. Step: 516000. Time Elapsed: 4208.478 s. Mean Reward: 0.294. Std of Reward: 0.238. Training.
```

ДОДАТОК Г

Таблиця Г.1 – Порівняльний аналіз розробленого ПЗ з аналогами

Показник для порівняння	SimulaAI (розроблене ПЗ)	CARLA	AirSim
Системні вимоги	– 5MB дискового простору; – 4-8GB оперативна пам'ять; – 4GB RAM GPU	– 130 ГБ дискового простору (CARLA потребує близько 31 ГБ, а Unreal Engine - близько 91 ГБ) для Linux, для Windows - 165 GB; – сервер потребує принаймні 6 ГБ пам’яті графічного процесора, хоча рекомендується 8 ГБ; – два порти TCP і хороше підключення до Інтернету (2000 та 2001 за замовчуванням)	– GPU мінімально 4GB RAM (краще 8 GB), NVidia 1080 чи NVidia Titan; – ядра - 6+; – SSD; – 2-3 монитора (4K); – оперативна пам’ять - 64GB RAM
Зручність використання	Симулятор надає можливість обирати алгоритми машинного навчання, регулювати параметри, моніторити процес навчання в 3D та графічній формах, для цього потрібно лише запустити .exe файл та натиснути на відповідну кнопку додатку. Параметри змінюються в окремому конфігураційному файлу.	Після аналізу користувацьких проблем на GitHub зроблено висновок, що існують проблеми крешей серверу Carla після недовгої роботи, неможливість або складність встановити комунікацію між клієнтом-сервером, потрібно детально вивчити документацію навіть для простих маніпуляцій	Середовище зарекомендувало себе з дуже гарного боку для навчання та тестування літальних транспортних засобів, але навчання автомобілів дещо обмежується.

Продовження таблиці Г.1

Передумови для роботи	Потрібно встановити Python, PyTorch та пакет mlagents. Але симулятор надає можливість також навчати моделі без засобів mlagents, тому встановлення додаткових залежностей в даному випадку не потрібно.	Згідно з документацією перед встановленням симулятора потрібно встановити: Python x64, NumPy, Carla client library (потрібно обрати формат .egg file чи .whl file, важливо вивчити вимоги щодо версіонності, а також цей файл неможливо скачати, він специфічний для поточної ОС). Якщо симулятор компілюється вручну: CMake, Make, 7Zip, Visual Studio 2019 (інші версії можуть спричинити конфлікти), Windows 8.1 SDK, x64 Visual C++ Toolset, .NET framework 4.6.2., Unreal Engine 4.26. (потрібно обов'язково прилінкувати GitHub акаунт до акаунта Unreal Engine, а потім скомпілювати модифіковану версію UE4). Згідно з документацією для компіляції Carla потрібно 4 години.	Epic Games Launcher, Unreal Engine, Visual Studio 2019, Python, пакет airsim.

Продовження таблиці Г.1

Спосіб запуску	Запуск через виконуваний файл.	Запуск сервера через виконуваний файл, але комунікація з симулятором неможлива лише через нього, потрібно підключити до нього Python клієнт.	Не має можливості запуску окремо, представляє собою плагін для Unreal Engine і функціонує лише як компонент його середовища.
Кросплатформеність	Windows, Linux, Mac	Windows, Linux	Windows, Linux, Mac
Основний ігровий рушій	Unity	Unreal Engine	Unreal Engine
Вбудований компонент машинного навчання	Так	Ні	Ні
Документація	Відсутня	Деталізована	Деталізована
Підтримка мов для написання скриптів	C#, Python	Python	Python, C++, C#, Java
Деталізованість середовища	Середня	Висока	Висока
Тип архітектури	Модульна	Клієнт-серверна	Модульна (плагін)

ДОДАТОК Д

Лістинг головних частин реалізації ПЗ

Vehicle контролер

```
…
public class VehicleController : Monobehaviour
{
…

    public void Awake()
    {
        rigidBody = GetComponent<Rigidbody>();
        startPosition = transform.position;
        startRotation = transform.eulerAngles;
    }

    private void VehicleMovementInputs()
    {
        leftRightMovementInput = Input.GetAxis("Horizontal");
        forwardBackMovementInput = Input.GetAxis("Vertical");
        stopping = Input.GetKey(KeyCode.Space);
    }

    public void ApplyAcceleration(float acceleration)
    {
        rigidBody.constraints = RigidbodyConstraints.None;
        colliderFLW.motorTorque = acceleration * accelerationForceChangeableInInsp;
        colliderFRW.motorTorque = acceleration * accelerationForceChangeableInInsp;
        vehicleBreakForceCurr = stopping ? breakingForceChangeableInInsp : 0f;
        ManageVehicleBreaking();
    }

    private void ManageVehicleBreaking()
    {
        colliderFRW.brakeTorque = vehicleBreakForceCurr;
        colliderFLW.brakeTorque = vehicleBreakForceCurr;
        colliderRLW.brakeTorque = vehicleBreakForceCurr;
        colliderRRW.brakeTorque = vehicleBreakForceCurr;
    }

    private void FullStop()
    {
        rigidBody.constraints = RigidbodyConstraints.FreezePosition;
    }

    public void ApplySteering(float steering)
    {
        vehicleSteerAngleCurr = steerAngleMaxChangebleInInspector * steering;
        colliderFLW.steerAngle = vehicleSteerAngleCurr;
        colliderFRW.steerAngle = vehicleSteerAngleCurr;
        ManageAllWheelsUpdate();
    }

    public void Respawn()
    {
        FullStop();
        transform.position = startPosition;
        transform.eulerAngles = startRotation;
    }
}
```

Vehicle агент

```
…
public class VehicleAgent : Agent
{
…

    public override void OnEpisodeBegin()
    {
        roadblocksHandler.ResetRoadblocks();
        vehicleController.ToStartingPosition();
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        Vector3 vehiclePosition = roadblocksHandler.nextRoadBlock.transform.position -
transform.position;
        sensor.AddObservation(vehiclePosition/20f);
    }

    public override void OnActionReceived(ActionBuffers actions)
    {
        var possibleVehicleActions = actions.ContinuousActions;
        vehicleController.ApplyAcceleration(possibleVehicleActions[1]);
        vehicleController.ApplySteering(possibleVehicleActions[0]);
    }

    public override void Heuristic(in ActionBuffers actionsOut)
    {
        var manualActions = actionsOut.ContinuousActions;
        manualActions.Clear();

        manualActions[0] = Input.GetAxis("Horizontal");
        manualActions[1] = Input.GetAxis("Vertical");
    }

}
```

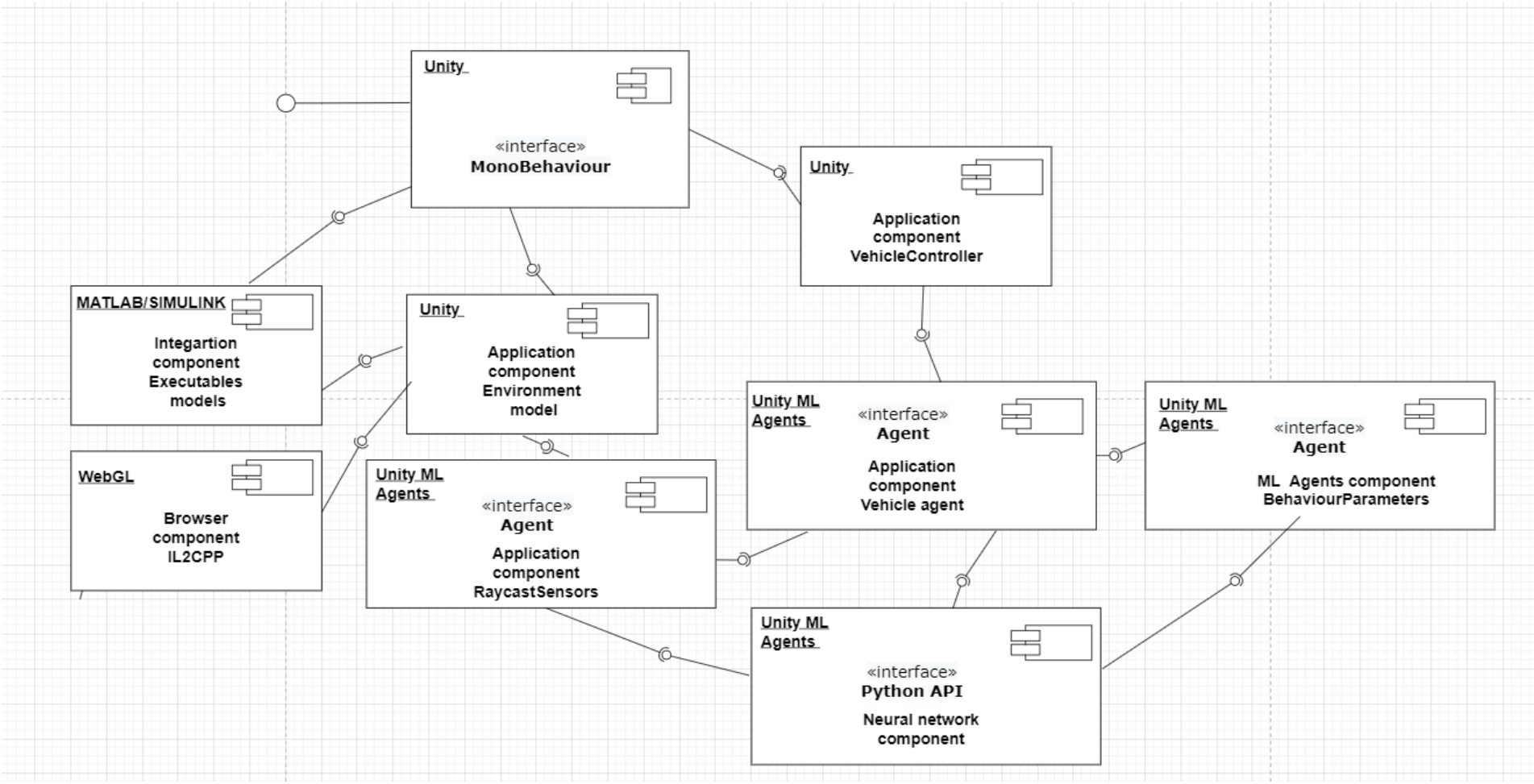
Config parameters Vehicle-Agent.yaml

```
default_settings: null
behaviors:
  VehicleAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 120
      buffer_size: 12000
      learning_rate: 0.0003
      beta: 0.001
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
    reward_signals:
      extrinsic:
```

```
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
    init_path: null
    keep_checkpoints: 5
    checkpoint_interval: 500000
    max_steps: 500000000
    time_horizon: 1000
    summary_freq: 12000
    threaded: true
    self_play: null
    behavioral_cloning: null
env_settings:
  env_path: null
  env_args: null
  base_port: 5005
  num_envs: 1
  seed: -1
engine_settings:
  width: 84
  height: 84
  quality_level: 5
  time_scale: 20
  target_frame_rate: -1
  capture_frame_rate: 60
  no_graphics: false
environment_parameters: null
checkpoint_settings:
  run_id: 18Training
  initialize_from: null
  load_model: false
  resume: false
  force: false
  train_model: false
  inference: false
  results_dir: results
torch_settings:
  device: null
debug: false
```

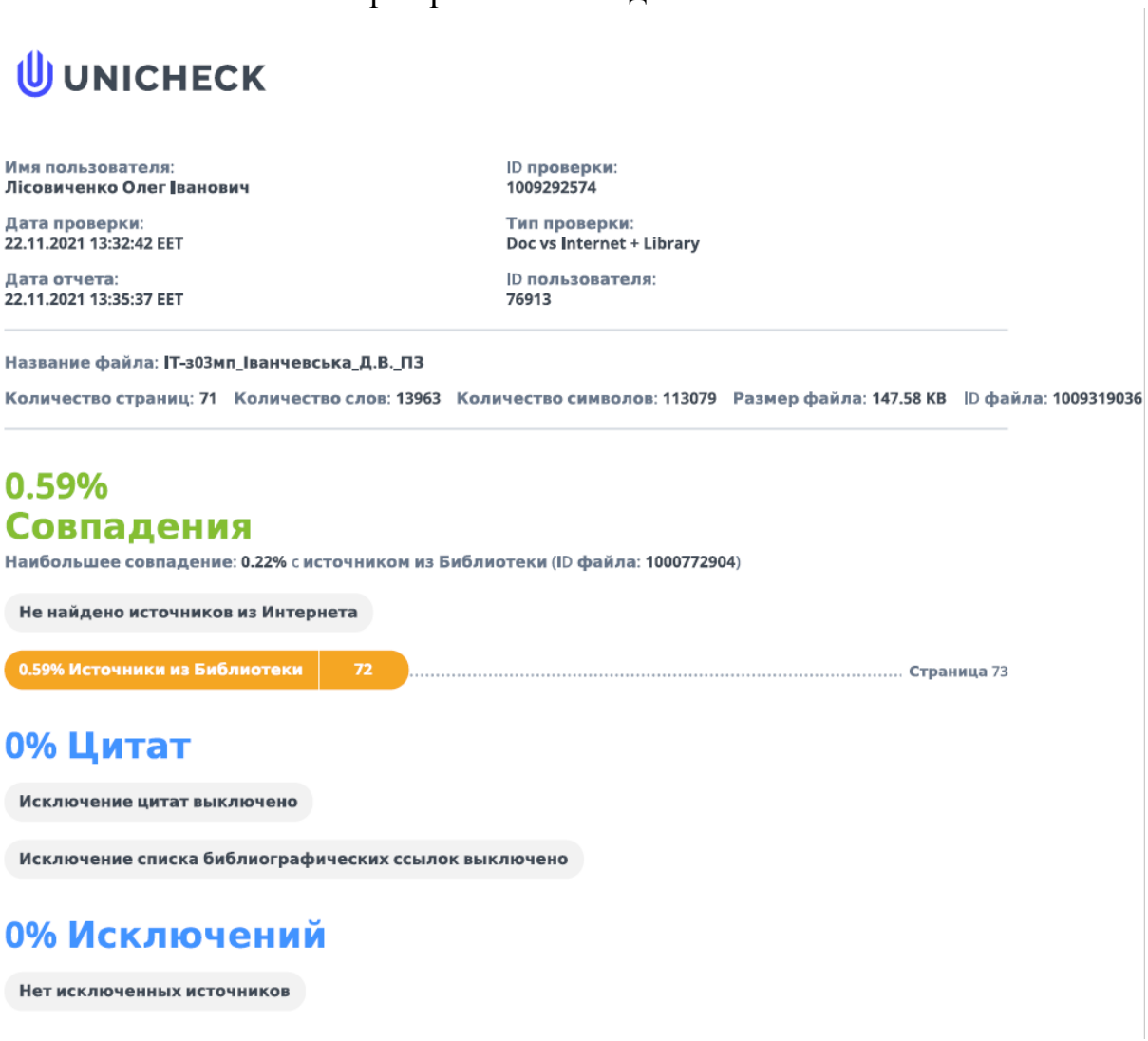
ДОДАТОК Е

Архітектура програмного забезпечення для навчання та тестування самокерованих транспортних засобів на основі компонентно-орієнтованого підходу



ДОДАТОК Ж

Перевірка на співпадіння



UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1009292574

Дата проверки:
22.11.2021 13:32:42 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
22.11.2021 13:35:37 EET

ID пользователя:
76913

Название файла: ІТ-з03мп_Іванчевська_Д.В._ПЗ

Количество страниц: 71 Количество слов: 13963 Количество символов: 113079 Размер файла: 147.58 KB ID файла: 1009319036

0.59%
Совпадения

Наибольшее совпадение: 0.22% с источником из Библиотеки (ID файла: 1000772904)

Не найдено источников из Интернета

0.59% Источники из Библиотеки 72 ..... Страница 73

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников