Державний вищий навчальний заклад   
«Прикарпатський національний університет імені Василя Стефаника»  
Факультет математики та інформатики  
Кафедра комп’ютерних наук та інформаційних систем

**ДИПЛОМНА** **РОБОТА**

на здобуття другого (магістерського) рівня вищої освіти  
на тему “Поведінкова модель неігрового персонажа комп’ютерної гри”

Виконав: студент II курсу, групи КНМ-21  
спеціальності   
122 Комп’ютерні науки  
Ільницький М. А.  
Керівник Горєлов В.О.  
Рецензент Іляш Ю. Ю.

Івано-Франківськ   
2021р.

Державний вищий навчальний заклад  
«Прикарпатський національний університет імені Василя Стефаника»

Факультет математики та інформатики  
Кафедра комп’ютерних наук та інформаційних технологій   
Освітньо-кваліфікаційний рівень магістр  
Напрям підготовки (Спеціальність): 122 Комп’ютерні науки

**ЗАТВЕРДЖУЮ   
Завідувач кафедри:**Петришин Л. Б. \_\_\_\_\_\_\_\_\_\_   
(підпис)   
“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ р.

**ЗАВДАННЯ  
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ**

Ільницькому Миколі Анатолійовичу   
1. Тему роботи «Поведінкова модель неігрового персонажа комп’ютерної гри»  
керівник роботи: доц. Горєлов В. О.   
Затверджені наказом університету від “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_р. №\_\_\_\_\_\_.

2. Термін здачі студентом закінченого проекту: 11.12.2021   
3. Перелік питань, які потрібно розробити: проаналізувати поняття поведінкової моделі, вплив алгоритмів машинного навчання на її формування, вибрати програмну платформу та практично реалізувати модель.   
4. Дата видачі завдання: 23.09.2020

Студент \_\_\_\_\_\_\_\_\_\_\_ Ільницький М. А.

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_ Горєлов В. О.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів роботи | Строк виконання етапів роботи | Примітка |
| 1 | Формування мети та завдання щодо об’єкту дослідження роботи | 15.10.2020 | Виконав |
| 2 | Дослідження літератури та джерел по темі роботи | 19.02.2021 | Виконав |
| 3 | Вивчення впливу алгоритмів машинного навчання на поведінкову модель | 22.06.2021 | Виконав |
| 4 | Вибір технічних засобів та вивчення їхнього функціоналу для виконання задачі | 16.07.2021 | Виконав |
| 5 | Формування поведінкової моделі та її імплементація в комп’ютерну гру | 22.08.2021 | Виконав |
| 6 | Візуальний дизайн проекту | 17.09.2021 | Виконав |
| 7 | Оформлення дипломної роботи | 20.10.2021 | Виконав |

Студент \_\_\_\_\_\_\_\_\_\_\_ Ільницький М. А.

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_ Горєлов В. О.

**АНОТАЦІЯ**Ільницький Микола Анатолійович  
Поведінкова модель неігрового персонажа комп’ютерної гри

Дана дипломна робота присвячена дослідженню видів машинного навчання та їх впливу на поведінкову модель комп’ютерного агента.   
 В цій роботі був проведений аналіз поняття поведінкової моделі, різновидів машинного навчання та їх вплив на поведінкову модель при розробці комп’ютерної гри.  
 Розроблено комп’ютерну гру з застосуванням агента побудованим на базі поведінкової моделі навченій певним способом.  
 Робота складається з 64 сторінок, 32 рисунків, 2 таблиць, 1 додатку на 1 сторінку.   
 Ключові слова: поведінкова модель, машинне навчання, unity, комп’ютерні агенти.

**ANNOTATION**Ilnickiy Mykola Anatoliyovych  
Behavioral model of a non-playable computer game character

This thesis is devoted to the study of types of machine learning and their impact on the behavioral model of the computer agent.  
 This work analyzed the concept of behavioral models, types of machine learning and their impact on the behavioral model in the development of computer games.  
 Videogame was developed by using of computer agent which was built on the basis of a behavioral model that was trained by particular way.  
 This work consists of 64 pages, 32 pictures, 2 tables, 1 applications with 1 page.  
 Tags: behavioral model, machine learning, unity, computer agents.

**ЗМІСТ**

**ВСТУП** 7

**РОЗДІЛ 1. ДЕКОМПОЗИЦІЯ ПОНЯТТЯ   
ПОВЕДІНКОВОЇ МОДЕЛІ** 9  
 1.1 Загальне поняття 9  
 1.2 В контексті нейронних мереж 11  
 1.3 Категорії, принципи та класифікація систем   
 машинного навчання 11  
 1.4 Динамічне та пакетне машинне навчання 21  
 1.5 Постановка задачі та проміжні висновки 24

**РОЗДІЛ 2. ПРОГРАМНІ ПЛАТФОРМИ ТА ІНСТРУМЕНТИ   
МАШИННОГО НАВЧАННЯ** 26  
 2.1 Формування вимог до програмного забезпечення   
 та вибір програмної платформи 26  
 2.2 Аналіз та вибір ігрових рушіїв 33   
 2.3 Порівняння Unity та Unreal Engine 40  
 2.4 Проміжні висновки та наступні кроки 43

**РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА РОЗРОБКА   
ПОВЕДІНКОВОЇ МОДЕЛІ** 46  
 3.1 Розробка концепції та вибір жанру 46  
 3.2 Оформлення та побудова середовища 50  
 3.3 Розробка логіки та формування поведінкової моделі 54  
 3.4 Візуальне оформлення та додаткові функції 62  
**ВИСНОВКИ** 65  
**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 66  
**ДОДАТОК** 68

**ВСТУП**

В контексті комп’ютерних агентів які керуються нейронними мережами завжди є шанс того, що агент почне приймати хаотичні, не очевидні або непередбачувані рішення, тому в зв’язку з цим з’являється потреба у додаткових засобах контролю над його поведінкою. При розробці такого агента слід враховувати особливості його роботи та особливості середовища в якому він проводить навчання і взаємодію.

Впровадження конкретного паттерну поведінки та певних обмежень для такого агента є одним з вирішення цієї проблеми. Оскільки логіка сучасних комп’ютерних ігор, в більшій мірі, опирається на логіку та взаємодію об’єктів з реального світу, то відповідно види поведінкових моделей агентів, які знаходяться в цих комп’ютерних іграх, можуть братись з реального світу.

Об’єктом дослідження в даній дипломній роботі є види машинного навчання, їх вплив на поведінкову модель та можливість їх впровадження в комп’ютерну гру. В цій роботі будуть дослідженні та проаналізовані певні види машинного навчання, будуть вивчені теоретичні можливості для їх імплементації в комп’ютерну гру та реалізація цих знань на реальному проекті.

Мета роботи – розробка проекту який представляє з себе комп’ютерну гру, в якій буде використовуватись комп’ютерний агент зі сформованою поведінковою моделлю яка буде оптимально працювати в контексті вибраного жанру комп’ютерної гри.

Завданням цього дослідження полягає у вивчені та аналізі поведінкової моделі, встановленню шляхів та формулювання пропозицій щодо їх впровадження в відеогру.

Практичне значення одержаних результатів полягає в розробці комп’ютерної гри в якій функціонує комп’ютерний агент з поведінковою моделлю яка була оптимально навчена відповідно до типу гри, що в результаті, якісно покращує загальний ігровий досвід гравця.

Апробація результатів роботи. Роботу було апробовано на Міжнародній науково-технічній конференції здобувачів вищої освіти та молодих вчених «Комп’ютерні науки, інформаційні технології та системи управління» (CSYSC-2020).

1. ДЕКОМПОЗИЦІЯ ПОНЯТТЯ ПОВЕДІНКОВОЇ МОДЕЛІ
   1. Загальне поняття.

В широкому розумінні поняття «поведінкова модель» може трактуватися як   
характерні дії та взаємодія певного суб’єкту з навколишнім середовищем. В більш вузькому розумінні ці дії приносять певний результат та мають певний вплив на це навколишнє середовище. Поведінкова модель формується шляхом накопичення інформації щодо дій та результатів суб’єкту.

В контексті людської поведінки широко відома система оцінювання поведінки індивіда під назвою «DISC», суть якої отримати загальні переваги в поведінці та, також, стиль цієї поведінки. Ця система була винайдена американським психологом Джоном Гайєром, робота якого, в свою чергу, ґрунтувалась на наробках Віль’яма Марстона. Назва цієї системи являє собою акронім та позначає кожен з чотирьох секторів які, в свою чергу, характеризують конкретного індивіда.

Перелік секторів та їх загальна характеристика:

1. Dominance (домінування) – риси поведінки які властиві особам які мають здатність до більш активної роботи з викликами та проблемами з якими стикаються.
2. Influence (вплив) – даний сектор вказує на особи в яких поведінка характеризується вирішенням задач шляхом дипломатії, діалогу та холодного розрахунку.
3. Steadiness (стійкість) – характеризує безпечність, стабільність та схильність до неприйняття змін. Цьому сектору характерна послідовність і тенденція до без упереджених висновків.
4. Compliance (відповідність) – позначає структурованість і внутрішню організацію індивіда. Особи в яких домінує ця характеристика виділяються жорстким дотримуванням правил та законів.

На рис. 1.1 зображена загальна схема оцінювання по системі «DISC».

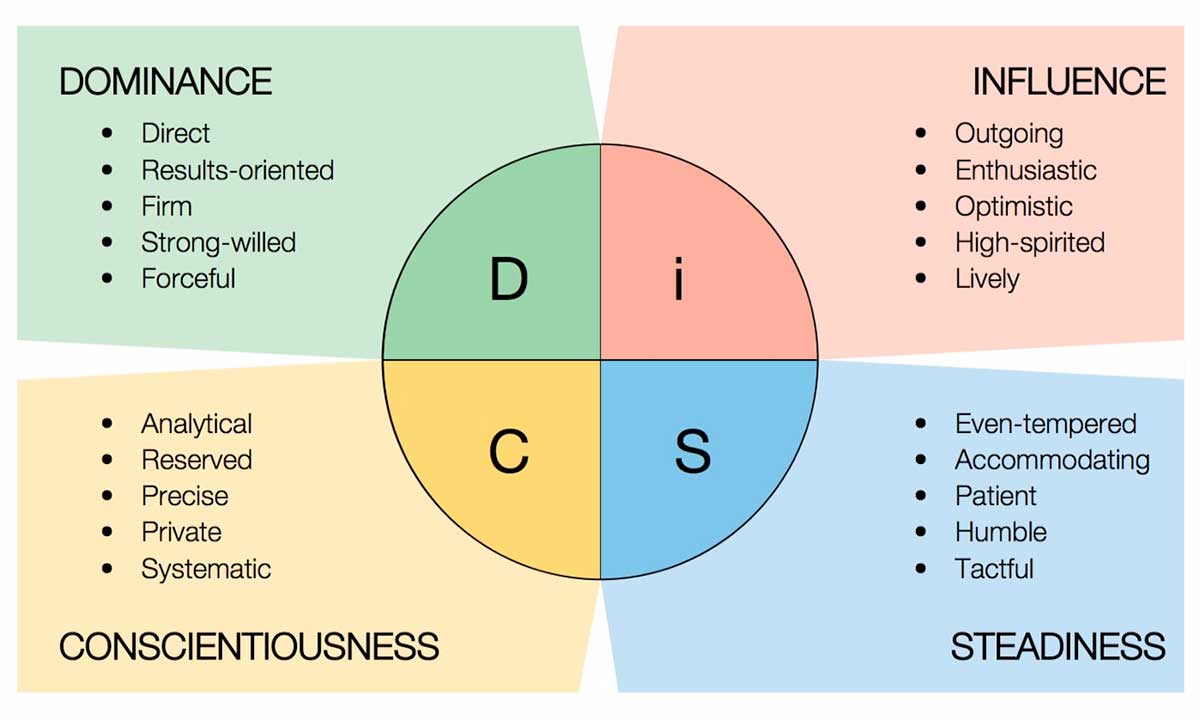


Рисунок 1.1 – Система «DISC»  
Автор: <https://united-partners.com/how-disc-personality-assessment-can-make-our-life-at-work-better/>

В результаті роботи цієї схеми отримується значення яке може трактуватись як характеристика певного індивіда. Висока характеристика певного сектору означає яскраву вираженість відповідної поведінки, низька – його протилежність. В загальному, схема не вказує на будь-які конкретні та точні дані про особу, однак використовується як первинна основа для оцінки поведінки та психологічного типу людини.

* 1. В контексті нейронних мереж

На модель поведінки комп’ютерного агента впливає тип та будова нейронної мережі яка ним керує. В свою чергу, на саму нейронну мережу – принцип за яким вона була навчена та яким способом проводилась взаємодія з навколишнім середовищем під час навчання. Для більш чіткого розуміння цього впливу виникає потреба в детальному огляді видів та принципів машинного навчання. Оскільки на даний момент існує безліч варіантів машинного навчання, для зручності їх розподіляють на кілька категорій по типу роботи. В книзі «Прикладне машинне навчання за допомогою Scikit-Learn, Keras та TensorFlow» за авторством Орель’єн Жерона [1] категорії систем машинного навчання розділялись по таким признакам:

1. Чи навчаються вони з людським контролем (навчання з вчителем, навчання без вчителя, часткове навчання (semi-supervised learning) і навчання з підкріпленням));
2. Чи можуть вони навчатись поступово в реальному часі (динамічне чи пакетне навчання);
3. Чи працюють вони тільки шляхом порівняння нових точок даних зі відомими точками даних, чи взамін знаходять патерни в навчальних даних та будують прогнозуючу модель подібну тому, як роблять вчені (навчання на основі зразків або на основі моделей).

* 1. Категорії, принципи та класифікація систем машинного навчання

Вищевказані критерії, по своїй суті, не є монолітними та можуть в тому чи іншому варіанті комбінуватись між собою. Прикладом такого поєднання є система навчання DPL (Direct policy learning), навчання прямої політики де, в силу своєї будови, використовуються елементи пакетного навчання з вчителем та динамічного навчання в реальному часі. Для більш ґрунтовного розуміння відмінностей та особливостей типів машинного навчання, нижче буде приведений детальний опис цих систем.

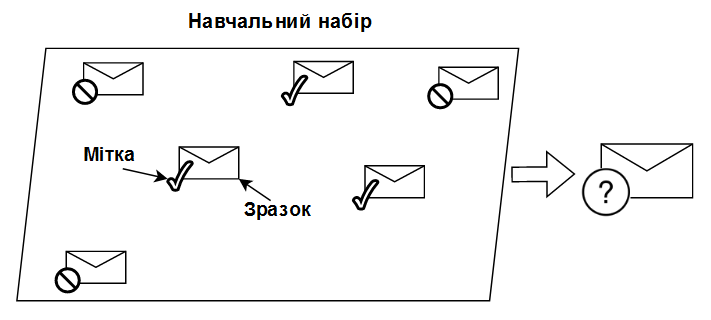
 Перш за все, слід виділяти системи МН (машинного навчання) за типом контролю під час навчання. Так, навчання з вчителем (або Supervised learning), глобально, відноситься до процесу навчання яке строго контролюється безпосередньо людиною або певним компонентом, який відповідає за досягнення кінцевого результату після завершення навчання. В загальному, суть даного виду машинного навчання полягає в обробці великої кількості інформації на предмет відмінностей, що постачається від коректних прикладів, тобто, по типу «відомий вхід – відомий вихід». Іншими словами, цей тип машинного навчання працює не тільки з конкретними атрибутами та векторами в прикладах які постачаються, а ще з інформацією про результат, який має видаватись в кінці роботи алгоритму. Типовою задачею даного типу машинного навчання є класифікація. Типовим прикладом такої задачі є навчання спам-фільтра, де системі надаються приклади (спам або не спам) листів для аналізу, результатом чого система здатна відрізнити листи опираючись на атрибути які вона попередньо вивчила. На рисунку 1.2 продемонстрована діаграма з принципом роботи такої схеми.

Рисунок 1.2 – Процес навчання спам-фільтру.   
Автор: авторська робота

Як було вказано вище, навчальний набір для спам-фільтру представляв з себе набір зразків (безпосередньо самих листів) та міток, які вказували на належність цих листів до тієї чи іншої категорії. Інший тип задач де задіються алгоритми машинного навчання з вчителем – задачі регресії. Задача регресії, на відміну від задач класифікації, не просто вибрати мітку з кінцевого набору значень, а змоделювати залежність міток від зв’язаних побічних компонентів та визначити закономірність змін значень міток при різних значеннях компонентів. В результаті алгоритм регресії визначає функцію, яка вміє прогнозувати значення мітки при будь-яких нових наборах вхідних компонентів. Типові задачі в яких застосовуються регресії:

1. Прогнозування ціни автомобіля, де вхідні компоненти це марка, пробіг, вік та інше.
2. Прогнозування ціни акцій на основі трендів, тенденцій, історичних та інших даних.
3. Прогнозування погоди, де вхідними даними є тиск, швидкість вітру, вологість, пора року і т.п.

Однак деякі алгоритми регресії також можуть застосовуватись і для класифікації. Так, наприклад, алгоритм логістичної регресії в силу того, що він може видавати значення, яке відповідає вірогідності належності до конкретного класу, може застосовуватись як інструмент для класифікації.

Навчання без вчителя (англ. Unsupervised learning) – як зрозуміло з назви, даний тип машинного навчання не припускає залучення попередньо поміченого навчального набору для роботи алгоритмів навчання. В широкому розумінні успішність навчання алгоритму без вчителя оцінюється тільки від кінцевого результату без поправки на спосіб отримання цього результату. Показовим прикладом роботи алгоритмів машинного навчання без вчителя є задачі кластеризації, де в масиві даних для об’єктів проводиться пошук та вибудовуються закономірності на основі певних параметрів, закономірність в яких бачить сам алгоритм та розподіляє ці об’єкти на певні групи саме по цим параметрам. Така техніка кластеризації може використовуватись в соціальних мережах при групуванні людей за певними ознаками, такими як стать, вік, спільні друзі або для зниження розмірності певних параметрів, тобто, зменшення кількості параметрів та звуження їх до якогось конкретного значення, яке є результатом виявленої закономірності між значеннями деяких об’єктів, наприклад, ступінь зносу автомобіля буде результатом зведення його віку та пробігу. В таблиці 1.1 сформовані основні концептуальні відмінності між принципами навчання з вчителем та без вчителя.

Таблиця 1.1 – Порівняння SL та UL

|  |  |  |
| --- | --- | --- |
| Параметр | Навчання з вчителем  (Supervised Learning) | Навчання без вчителя (Unsupervised Learning) |
| Контроль зі сторони людини під час навчання | Присутній. Різниться від слабкого до строгого | Відсутній |
| Тип даних та навчальний набір | З самого початку поділений на «правильні» та «неправильні» варіанти | Навчальний набір ніяк не класифікований та не розсортований |
| Типовий принцип навчання | Пошук відмінностей та/або схожості по строгому алгоритму | Пошук нових алгоритмів які б приводили до певного результату |
| Сильні сторони | Ефективне виконання задач оптимізації та класифікації | Робота без шаблонів, пошук закономірностей які можуть бути не очевидні |
| Слабкі сторони | Залежність в попередній розмітці | Менша ефективність в специфічних задачах |

Часткове навчання (англ. Semi-supervised learning) – компромісний метод машинного навчання. Оскільки ситуації де масив даних, в силу свого великого розміру, може складатись як і з помічених зразків так і зі зразків без поміток є доволі поширеними, виникає потреба в використанні алгоритмів машинного навчання, які вміють працювати в обох цих ситуаціях. Алгоритми часткового навчання, здебільшого, це комбінації алгоритмів навчання з вчителем та без. Так як за своєю суттю навчальний набір, який постачається алгоритмам часткового навчання, здебільшого, складається з елементів які не є попередньо поміченими, а помічені елементи складають лише малу долю серед всієї маси, алгоритми часткового машинного навчання працюють за таким принципом: спочатку навчання проводиться на елементах які розмічені (навчання з вчителем) в результаті чого формується навчена модель, з допомогою якої розпізнається або передбачається клас для не розмічених даних. Дане навчання проводяться по колу та з кожною ітерацією цього алгоритму навчена модель оновлює свої дані щодо класифікацій та типів даних. Прикладом алгоритму часткового навчання є Глибинна мережа довіри (англ. deep belief network), її схема зображена на рисунку 1.3.



Рисунок 1.3 – Глибинна мережа довіри.   
Автор: авторська робота

Даний алгоритм побудований на основі компонентів навчання без вчителя під назвою машини Больцмана (англ. restricted Boltzmann machine), які накладені один на другого, після чого ця конструкція проводить корегуванням зовнішнім елементом з використанням принципів навчання з вчителем.

Навчання з підкріпленням (англ. Reinforcement learning) – сукупність методів головним принципом яких є використання винагороди для подальшого навчання та збільшення ефективності роботи агента шляхом взаємодії з навколишнім середовищем. Агентом в даному методі машинного навчання називають систему яка навчається та отримує винагороди, самостійно вибудовуючи на основі цього алгоритми дій. Загальна схема даного виду машинного навчання продемонстрована на рисунку 1.4.

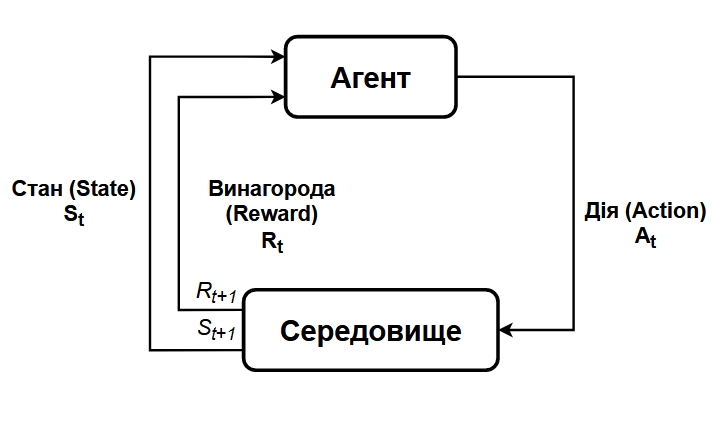


Рисунок 1.4 – Навчання з підкріпленням.   
Автор: авторська робота

Навчання з підкріпленням є одним з трьох базових парадигм навчання нейронних мереж, на ряду з навчанням з вчителем (англ. Supervised learning) та навчанням без вчителя (англ. Unsupervised learning), по ходу розробки систем з машинного начання можуть частково використовуватись всі види за потребою. Дана концепція навчання нейронних мереж містить багато різноманітних алгоритмів які відрізняються по принципу реалізації та сценаріїв використання. Найчастіше використовуються такі алгоритми:

1. Q-навчання (англ. Q-learning) – алгоритм який дозволяє реалізувати навчання з підкріпленням без моделі. Основана на рівнянні Белмана і в представлені Q-value має такий вигляд (рис. 1.5).

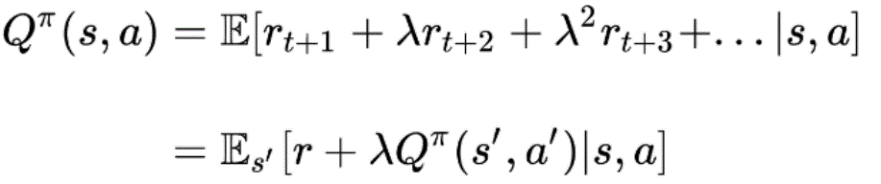


Рисунок 1.5 – Рівняння Белмана в представлені Q-value.   
 Автор: <https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287>

Метою алгоритму є максимізація значення Q\* яке, в свою чергу, є аналогом значенню Value (V) та позначає очікувану винагороду. Основна різниця – прийом додаткового параметру *a*. Даний алгоритм в своїй найпростішій реалізації зберігає дані в таблиці та при накопичені великої кількості вхідних даних про стани середовища і дії агента відрізняється зменшеною ефективністю, оскільки, з часом послідовність дій агента стає одноманітною і він виробляє один і той самий паттерн поведінки. Приклад таблиці з застосуванням алгоритму Q-learning наведений на рис. 1.6.

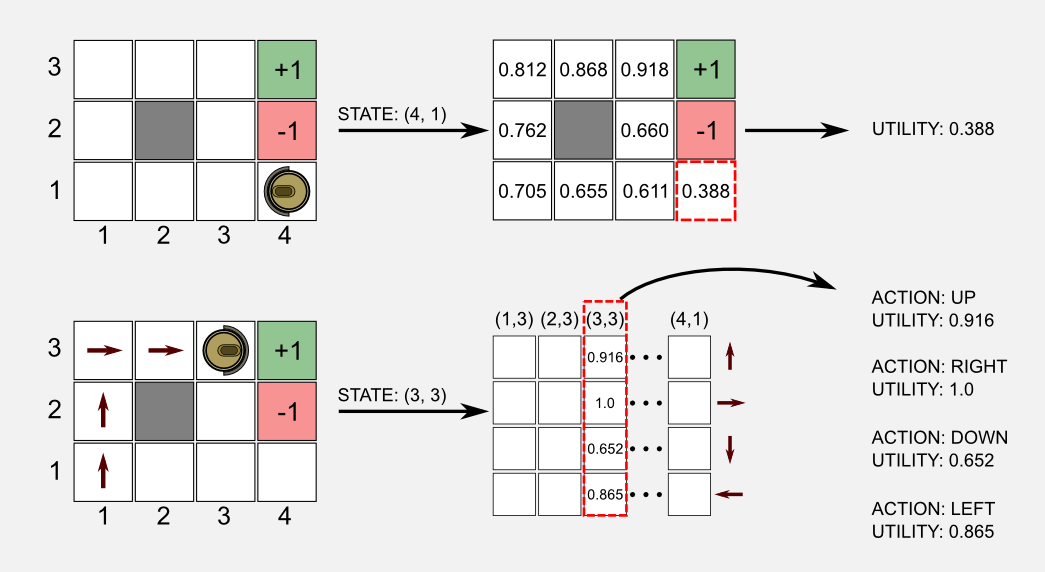


Рисунок 1.6 – Приклад простої Q-learning таблиці.   
 Автор: https://mpatacchiola.github.io/blog/2017/12/11/dissecting-reinforcement-learning-7.html

На даній ілюстрації зображена таблиця після завершення виконання програми для пошуку та проходження шляху.

1. Поглиблене Q-навчання (англ. Deep Q-learning) – по суті, є вдосконаленим варіантом попереднього алгоритму. Відрізняється тим, що при реалізації не використовується таблиця, а сама нейронна мережа. Зокрема, замість співставлення пари стан-дія з q-значенням – виконується співставлення вхідних станів з парами дія та q-значення.
2. Навчання часовим різницям (англ. Temporal difference learning) – метод навчання де прийнятті рішення при виконанні задачі засновані на вже присутньому до цього досвіді. Тобто, даний метод має властивість коригувати свою поведінку тільки після пройденого кроку в певний *t* момент часу засновуючись на тому що саме сталось після цієї дії.
3. Метод SARSA (англ. акрн. State-action-reward-state-action) – алгоритм де Q-функції залежать від поточного стану агента S1, дії яка вибирається A1, винагороди яку агент отримує R, стан який отримується агентом в результаті цієї дії S2 та наступної дії яка залежить від другого стану (A2). На рисунку 1.7 зображена формула методу SARSA.

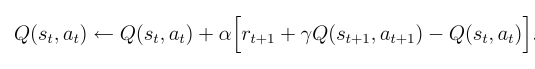
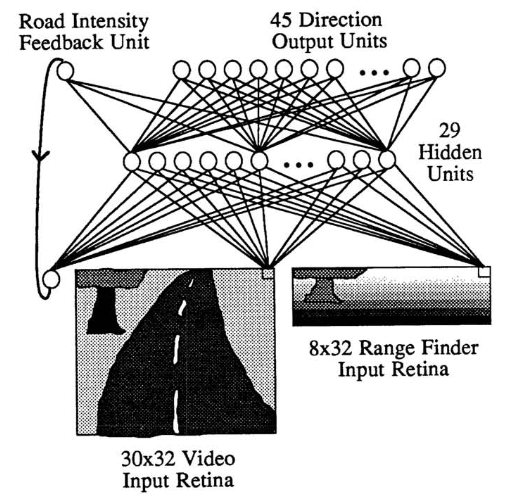


Рисунок 1.7 – Алгоритм SARSA  
Автор: http://incompleteideas.net/book/ebook/node64.html

Даний вид машинного навчання знаходить своє застосування в системах які припускають виконання певних шаблонів з мінімальною похибкою при конкретних ситуаціях та отримання максимальної винагороди під час навчання шляхом точного виконання певної роботи. Прикладом таких систем є технології автономного керування автомобілем, де даний комплекс не має на меті повністю вивчити кожну дорогу, місцевість або перехресток, а виробити певний універсальний патерн ефективної роботи.

Навчання шляхом імітації (англ. Imitation learning) – метод навчання нейронної мережі шляхом виконання певної задачі експертом з метою демонстрації бажаної моделі поведінки агента. Даний метод навчання може засновуватись на принципах навчання з підкріпленням, однак за зворотнім шляхом, де винагорода видається не за будь-яке досягнення мети, а тільки методом виконання задачі по аналогії з експертом. Так, до виду навчання шляхом імітації можна занести метод поведінкового клонування (англ. Behavioral cloning). Суть даного способу навчання полягає в використанні експертної політики (policy) та застосуванні цієї політики до нейронної мережі шляхом навчання зі спостерігачем. Прикладом застосування такого типу навчання можуть бути технології автопілоту для автомобілів, роботів, тощо. Одною з перших робіт на тему навчання шляхом імітації є робота 1988 року за авторства Діна Померлоу під назвою «ALVINN: An autonomous land vehicle in a neural network» [24]. На рисунку 1.8 зображена архітектура проекту ALVINN.



Дана архітектура функціонувала на базі прихованого одно-шарового компоненту нейронів, та отримувала інформацію з попередньо обладнаних сенсорів, які знаходились на транспортному засобі, в результаті чого, базуючись на дорожній розмітці відбувалось автономне керування автомобілем.

Рисунок 1.8 - Архітектура ALVINN   
Автор: https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

Навчання прямої політики (англ. Direct policy learning) – метод навчання який базується на навчанні шляхом імітації та є, по суті, його покращеною версією. Основною відмінністю даного методу від звичайного IL є можливість звертатись до експерта під час процесу навчання. По аналогії з IL, з кожною ітерацією роботи даний метод використовує механізми навчання з вчителем, щоб вивчити нову політику. Далі для перевірки нової політики робиться запит до експерта, який, в свою чергу, надає додаткову інформацію для компоненту навчання з вчителем. Даний алгоритм продовжує свої ітерації доки нейронна мережа не почне виконувати свою роботу по аналогії з експертом. До переваг такого методу навчання можна занести:

1. В силу особливостей методу, модель навчання має здатність до довгострокового планування.
2. Висока ефективність при достатніх кількостях ітерацій роботи методу.

Основним недоліком даного методу навчання є критична необхідність у наявності інтерактивного експерта, однак, саме наявність цього компоненту є найбільш важливим у функціонуванні такої системи.

* 1. Динамічне та пакетне машинне навчання

В процесі виконання будь-якого виду алгоритму машинного навчання виникає питання постачання навчальної бази для опрацювання алгоритмом, тобто, чи будуть дані постачатись в реальному часі, чи будуть існувати в вигляді вже готових навчальних пакетів.

Динамічне навчання (або Машинне навчання в режимі онлайн), вид навчання при якому інформація та дані для навчання постачаються поступово або в вигляді малих порційних частин які називаються «міні-пакети». Даний підхід до постачання даних для машинного навчання показує свою ефективність в задачах, де присутнє обмеження на час виконання кожної ітерації та обчислювальні ресурси апаратного забезпечення. Висока ефективність динамічного навчання зумовлена тим, що кожен набір даних, який постачається системі, може бути відкинутим як тільки з’явиться більш актуальний та новий набір. На рисунку 1.9 зображена діаграма яка демонструє принцип роботи динамічного навчання.

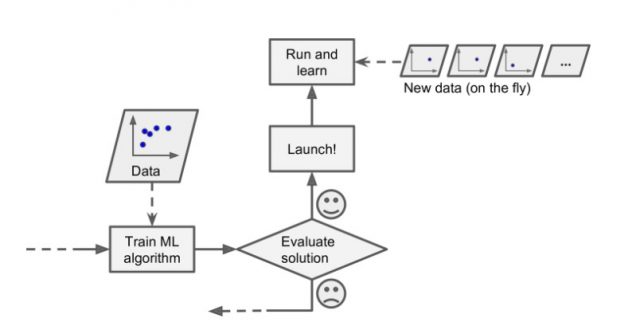


Рисунок 1.9 – Типова схема динамічного навчання.   
Автор: <https://vitalflux.com/difference-between-online-batch-learning/>

Оскільки даний підхід до машинного навчання передбачає можливість швидко проводити процес навчання шляхом потокового постачання новими даними, виникає проблема адаптації мережі до кожного нового міні-пакету навчальних даних. Так, для корегування точності та швидкості навчання в системах динамічного навчання існує такий параметр як «швидкість навчання» (англ. learning rate). Високе значення даного параметру приводить до швидкого адаптування системи до нових даних, однак всі дані, які вже обробились мають властивість «забуватись» системою. Протилежністю цього є низькі значення параметру швидкості навчання, де система навчається з більш низькою швидкістю, але стає менш чутливою до різноманітних шумів в нових пакетах даних. До недоліків такого принципу машинного навчання можна занести:

1. Потреба в грамотному конфігуруванні швидкості навчання, оскільки занадто високе значення цього параметру приводить до зменшеної ефективності роботи в подальшій перспективі, а при занадто низьких його значеннях – до, загально, низької швидкості навчання.
2. Зменшення ефективності при початково некоректних даних, які постачаються по ходу навчання. Для запобігання такого ефекту з’являється потреба в додатковій ревізії навчальних даних.

Пакетне навчання, за своїм принципом, є протилежністю динамічному (або онлайн) навчанню. Процес навчання при даному підході полягає в використанні сформованого монолітного пласту даних без динамічного постачання нових зразків. Оскільки навчальний набір для систем пакетного навчання розповсюджується в вигляді зразків відносно великих розмірів, час виконання та обробки таких наборів оцінюється як доволі значний та потребує великих апаратних витрат, перед впровадженням безпосередньо в виробниче середовище, прийнято проводити навчання в вигляді автономного процесу.   
Такий підхід до машинного навчання обмежений своєю низькою здатністю в плані адаптації до нових змін в навчальному наборі, оскільки для того, щоб занести нові дані в систему, потрібно проводити процес навчання з самого початку. Враховуючи особливості даного підходу до машинного навчання можна сформувати такі тези:

1. Пакетне навчання ефективно проявляє себе в ситуаціях де навчальний набір строго контрольований та не потребує оновлення.
2. Застосування систем машинного навчання які тренуються шляхом пакетного навчання є корисними в задачах, де основною метою є робота в статичних системах де присутні чітко сформовані сценарії дій.
3. Підхід коли для адаптації потрібно проводити повторне навчання з нуля може бути вигідним, якщо це навчання проводити регулярно. Таким чином можна частково отримати ефект динамічного навчання при правильній конфігурації.

До недоліків пакетного машинного навчання можна занести:

1. Підвищені вимоги до апаратних ресурсів якщо порівнювати динамічне машинне навчання.
2. В ситуаціях коли виникає потреба оновити навчальний набір, проявляються великі витрати в часі для процесу повторного навчання, що може бути неприпустимим в випадку комерційної експлуатації таких систем.
3. Загальна низька гнучкість такого підходу машинного навчання в порівнянні з динамічним навчанням та застосуванням «міні-пакетів».
   1. Постановка задачі та проміжні висновки

В даному розділі була проведена робота по декомпозиції поняття «поведінкова модель» з загальної точки зору та в контексті технологій машинного навчання та нейронних мереж. Дослідженні особливості різних алгоритмів машинного навчання нейронних мереж, що допоможе в подальшому виборі програмної платформи для розробки та, безпосередньо, самого процесу розробки.   
 Після поглибленого аналізу цієї теми можна привести такі проміжні висновки:

1. Поведінкова модель формується шляхом вибору принципу навчання, так навчання з вчителем та з підкріпленням, на відміну від навчання без вчителя, добре підходять для ситуацій коли потребується чітке слідування алгоритмам або певним сценаріям поведінки. При цьому, саме в випадку навчання з підкріпленням та з частковим залученням вчителя (semi-supervised learning) є в наявності достатньо інструментів для ефективного впливу та коректування поведінкової моделі агента.
2. Альтернативою навчанню з вчителем та навчанню з підкріпленням може бути навчання шляхом імітації. Враховуючи особливості роботи цього алгоритму та методу навчання, цей вид машинного навчання можна розглядати як один з варіантів імплементації в проект.
3. При подальшій розробці слід приділити увагу яким чином будуть поступати навчальні набори, тобто, чи буде навчання засновуватись на динамічному або пакетному принципі, оскільки в даному випадку присутня значна різниця в часі та загальній продуктивності навчання.

Оскільки тематикою роботи було є розробка певної комп’ютерної гри, в якій би були присутні елементи машинного навчання, зокрема комп’ютерні агенти в ролі суперників, основною задачею роботи є створення максимально ефективної системи машинного навчання на базі конкретної програмної платформи, з можливістю тим чи іншим чином впливати на процес та результати навчання комп’ютерного агента, шляхом коректування поведінкової моделі цього агента інструментами які надаються обраними програмними платформами.

2. ПРОГРАМНІ ПЛАТФОРМИ ТА ІНСТРУМЕНТИ МАШИННОГО НАВЧАННЯ

2.1 Формування вимог до програмного забезпечення та вибір програмної платформи

На сьогоднішній день існує велика кількість програмних рішень які так чи інакше спеціалізуються на галузях де використовується машинне навчання. Перш за все варто відмітити платформи хмарного обчислення, які надаються в вигляді спеціальних сервісів, та працюють в більшій мірі тільки на певному обладнанні розробника. Нижче приведені певні їх представники де будуть відзначені їхні переваги та недоліки.

Amazon SageMaker – платформа, яка розроблена компанією Amazon, а точніше, її підрозділом Amazon Web Services. Даний сервіс надає в своє розпорядження платформу для машинного навчання, дозволяє тренувати та впроваджувати натреновані моделі в різноманітні системи та обладнання, наприклад, вбудоване. Сферами застосування даної платформи є технології автономного керування автомобілем, систем «машинного зору», системи персоналізованих комерційних рекомендацій, прогнозування відтоку клієнтів за допомогою методів глибинного навчання, розпізнавання рукописного тексту, та ін. В контексті теми та задачі роботи можна виділити такі переваги даної платформи:

1. Наявність окремого вбудованого середовища розробки під назвою SageMaker Studio.
2. Достатньо широка підтримка різноманітних мов програмування.
3. Можливість розгортки розробленої моделі на технологіях хмарного обчислення.
4. Висока швидкість навчання та широкий вибір алгоритмів машинного навчання.

Недоліки Amazon SageMaker:

1. Висока ціна на послуги хмарного обчислення та машинного навчання.
2. Недостатні можливості до імплементації платформи у розробку комп’ютерної гри.
3. Відносно мала кількість документації та навчальної бази, а також прикладів застосування даної платформи при моделюванні поведінкової моделі агента та розробці комп’ютерної гри.

Хоча основний недолік, а саме – висока ціна на сервіс, частково вирішується використанням Amazon SageMaker Neo (проектом з відкритим кодом), однак це не вирішує питань недостатньої функціональності для створення комп’ютерних ігор.

Microsoft Azure Machine Learning – в цілому, близький по функціоналу з SageMaker сервіс та є реалізацією машинного навчання на програмній платформі компанії Microsoft. Являється компонентом сервісу Azure та надає доступ до більшості можливостей цієї платформи. До переваг цієї платформи можна віднести:

1. Наявність безкоштовної версії.
2. Велика кількість доступних алгоритмів машинного навчання, в тому числі від компанії Microsoft та сторонніх компаній.
3. Дозволяє створювати на проводити навчання власних моделей з їх подальшим використанням в інших сервісах та API.

Недоліки:

1. Відносно мала кількість доступного сховища для зберігання навчальних моделей в безкоштовній версії.
2. Низька адаптованість до роботи з комп’ютерною графікою та ігровими рушіями.

Саме низька адаптованість до роботи з проектами в яких фігурують ігрові рушії є основною причиною, чому дана програмна платформа машинного навчання не може задовільнити певні потреби при розробці комп’ютерної гри.

Оскільки використання онлайн платформ для машинного навчання і імплементації натренованої моделі в комп’ютерну гру по ряду причин є не зовсім доцільним, можливим вирішенням цієї проблеми може стати використання окремого фреймворку або плагіну, який би дозволяв проводити навчання прямо на персональному комп’ютері та, по можливості, був би інтегрованим з ігровим рушієм. Надалі буде розглянуто певна кількість таких фреймворків та будуть проаналізовані їхні особливості.

mlpack – спеціальна бібліотека для мови програмування C++, основана на математичній бібліотеці Armadillo. Дозволяє вести розробку та навчання за допомогою таких мов програмування як: C++, Go, Python, Julia та CLI. Дана бібліотека підтримує велику кількість алгоритмів машинного навчання, такі як лінійна регресія, логістична регресія та такі моделі як прихована марковська модель та інші. До переваг даного фреймворку можна занести:

1. Висока швидкодія, оскільки основною мовою програмування проекту являється C++.
2. Широкий перечень доступних алгоритмів машинного навчання та підтримка багатьох видів моделей.

Недоліки mlpack:

1. Даний фреймворк, в першу чергу, розроблений та адаптований для специфічних математичних задач, тому кількість доступних алгоритмів, які можна використати для навчання комп’ютерного агента для роботи в комп’ютерній грі вкрай мала.
2. Відсутність відомостей та офіційної документації на предмет інтеграції фреймворка в будь-який відомий ігровий рушій.

Хоч фреймворк mlpack показує одну з найкращих швидкодій серед інших фреймворків машинного навчання (рис. 2.1), в контексті даної роботи він не є найкращим вибором, оскільки він мало адаптований до роботи з ігровими рушіями.

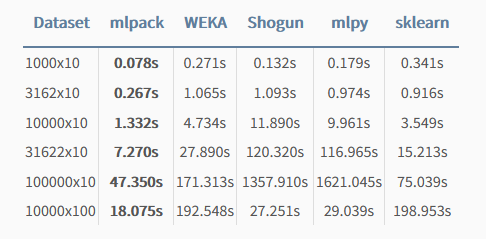


Рисунок 2.1 - Порівняння швидкодії mlpack.   
Автор: <https://www.mlpack.org/>

PyTorch – фреймворк з відкритим кодом, який розробляється силами компанії Samsung та дозволяє проводити прискорені обчислення за допомогою GPU. Надає достатньо широкий функціонал при побудові навчальної системи та має в наявності модулі для тензорних обчислень, алгоритмів глибинного навчання та навчання з підкріпленням. Основною мовою програмування є Python, але є можливість використання мов C та C++ шляхом задіяння API-розширень. До переваг фреймворку PyTorch можна віднести:

1. Відкритий вихідний код фреймворку.
2. Велика кількість готових навчених моделей, які готові до подальшого використання.
3. Велика кількість модулів. Наприклад для додаткової оптимізації багатьох популярних алгоритмів машинного навчання та побудови нейронних мереж.

До недоліків PyTorch відноситься:

1. Мала кількість документації та в загальному мала кількість задокументованих можливостей фреймворка.
2. Потреба у самостійному написанні кожного тренувального коду, оскільки готових рішень для цих задач немає.
3. За умовчуванням немає візуального відображення графів та інших інструментів візуалізації даних. Задля такого функціоналу прийдеться використовувати сторонні рішення.

В загальному, мала кількість документації та низька адаптованість фреймворку до роботи в ігрових рушіях не дозволяє використовувати його як платформу для машинного навчання в даному випадку.

Apache Spark – фреймворк який реалізує можливості розподілених обчислень для обробки даних структур будь-якого виду. Спеціалізується, в першу чергу, для роботи в хмарних обчисленнях, однак, в силу особливості принципу роботи з оперативною пам’яттю – ефективно показує в алгоритмах машинного навчання. Фреймворк написаний, в більшій мірі, на мовах програмування Java та Scala, базується на апаратній платформі Java Virtual Machine. Окрім цього є підтримка мов Python, R, та стандартів SQL і Java Database Connectivity. Серед переваг Apache Spark можна виділити:

1. Висока ефективність роботи з оперативною пам’яттю обчислювальної системи, що надає певні переваги в продуктивності.
2. Велика кількість доступних алгоритмів машинного навчання.

Слабкі сторони Apache Spark:

1. Низька адаптованість до алгоритмів машинного навчання, які частіше за все використовуються для роботи з ігровими рушіями.
2. Загальна «громіздкість» фреймворку.

Не зважаючи на всі переваги даного фреймворку, його використання для розробки комп’ютерного агента, який би тренувався та функціонував в ігровому рушії, є декілька недоцільною, оскільки дана платформа являє собою не тільки основу для машинного навчання, а ще й певним інструментом для розподілених обчислень, що є декілька надлишковим в контексті даної роботи.

TensorFlow – програмна платформа, яка бере свій початок як проект підрозділу компанії Google – Google Brain. Представляє з себе бібліотеку, яка написана, в більшій мірі, мовами програмувань Python та C++ та націлена на забезпечення роботи машинного навчання. По аналогіями з багатьма фреймворками які забезпечують роботу з машинного навчання, працює на основі C++ та C, а для більш зручного користування використовує високорівневий Python. Даний фреймворк за умовчуванням має в наявності інструменти для графічного відображення графів, коефіцієнтів та іншої візуалізації даних про поточне навчання (рис. 2.2) під назвою TensorBoard.

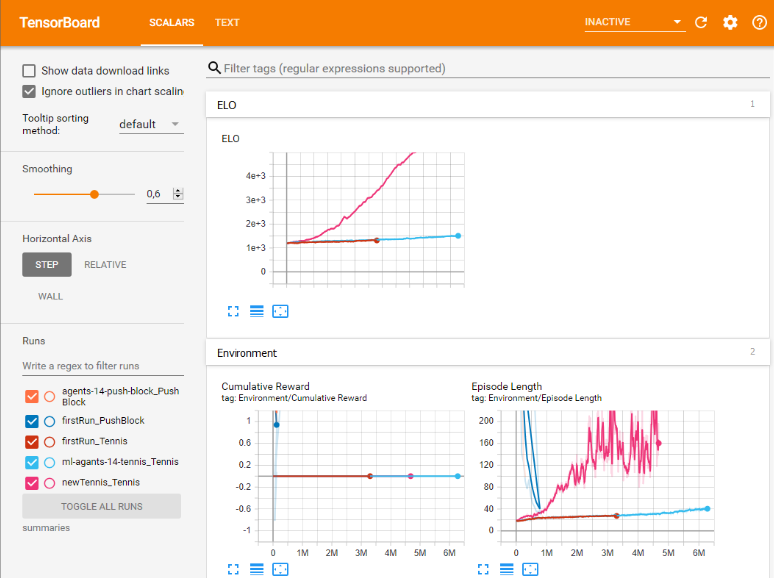


Рисунок 2.2 - Графічний інтерфейс TensorBoard.  
 Автор: авторська робота

Оскільки даний фреймворк є одним з найпопулярніших серед інших інструментів машинного навчання, існує велика кількість адаптацій та імплементацій цієї бібліотеки в різні проекти, в тому числі в вигляді плагіну для ігрових рушіїв, наприклад в Unity. Підсумовуючи, можна виділити такі переваги TensorFlow:

1. Висока популярність, з чого випливає велика кількість готових рішень різних проблем, наявність обширної навчальної бази, широка спільнота та постійний вихід нових версій.
2. В силу особливостей внутрішньої архітектури, оптимізація роботи проводиться силами самого фреймворку.
3. Великий список підтримуваних платформ, серед них мобільні платформи, персональні комп’ютери та серверне обладнання.

Виявлені недоліки TensorFlow:

1. Достатньо високе споживання пам’яті відеоадаптера при проведенні навчання.
2. Далеко не завжди стабільна робота, особливо це виділяється при використанні нового функціоналу.

Враховуючи всі переваги даного фреймворку, його можна занести як претендентом для побудови системи машинного навчання. В даному випадку, в першу чергу, на це впливає той факт, що даний фреймворк має свою реалізацію для ігрового рушія Unity та може бути адаптований для Unreal Engine.   
 Опираючись на вищевказані дані можна сформувати такі вимоги до вибору програмної платформи по машинному навчанню:

1. Фреймворк або бібліотека повинна мати офіційну реалізацію для певного ігрового рушія або може бути якимось чином адаптована для роботи в ігровому рушії.
2. Можливість роботи вихідної моделі на різних платформах без її глобальної модифікації.
3. Обширна документація.
4. Стабільна робота та достатньо широкий набір алгоритмів машинного навчання.
5. Опціонально – можливість коректувати поведінкову модель та конфігурацію навчання можливостями мови програмування ігрового рушія та його інструментами.

Беручи до уваги вимоги які були вказані вище, найкращим варіантом для побудови моделі машинного навчання та подальшої роботи з ними є саме фреймворк TensorFlow, оскільки він, так чи інакше, задовольняє кожну з вищевказаних вимог. Оскільки ніяка з розглянутих платформ не надає можливості створити на їх базі комп’ютерну будь-якого вигляду, з’являється потреба у виборі ігрового рушія. Надалі будуть переглянуті певні їхні представники та проаналізовані в контексті поточної задачі.

2.2 Аналіз та вибір ігрових рушіїв

Враховуючи характер розроблювального проекту слід зазначити, що ігровий рушій повинен мати широку спеціалізацію та повинен працювати з двовимірною та тривимірною графікою. По цій ж причині, вибраний ігровий рушій повинен мати стабільну версію, велику кількість документації та активну підтримку з боку його розробника. Опираючись на це, в категорію претендентів не можуть входити рушії, які є експериментальними, тестовими або слабо-розвинутими на даний момент. Таким чином, можна відмітити декілька ігрових рушіїв, а саме:

Unreal Engine – платформа для створення комп’ютерних ігор, спецефектів та іншого графічного контенту, розробником якої є компанія Epic Games. Перша версія даного рушія вийшла в 1998 році одразу з виходом однойменної гри «Unreal». На сьогоднішній день рушій нараховує свою четверту ітерацію (вже анонсована п’ята версія), розповсюджується шляхом комерційної ліцензії та доступна безплатно в разі якщо сума прибутку від розробленої гри не буде перевищена до певного розміру. Основною мовою програмування для даного рушія є C++. Альтернативою класичному способу створення логіки, в Unreal Engine присутня система візуального програмування під назвою «Blueprints», простий приклад зображений на рисунку 2.3.

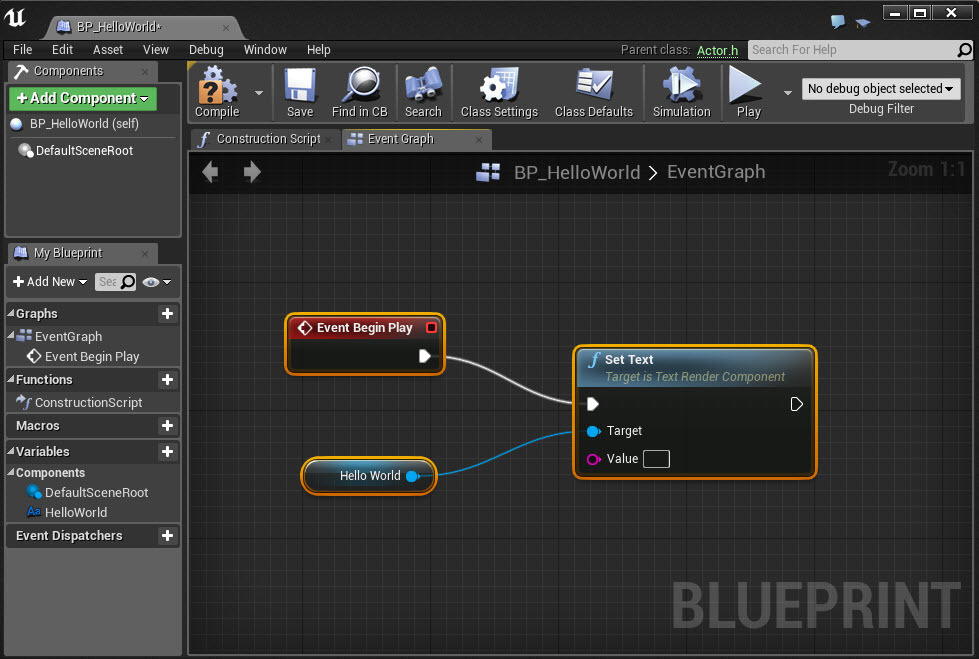


Рисунок 2.3 - Приклад системи Blueprints.   
Автор: <https://docs.unrealengine.com/4.27/en-US/Resources/ContentExamples/Blueprints/1_1/>

По своїй суті Blueprint є візуальним відображенням коду який може бути написаний на C++ та є з ним сумісним. Даний компонент рушія дозволяє швидко створювати проекти в якості прототипів шляхом взаємодії з інтерфейсом. Unreal Engine підтримує більшість сучасних платформ та операційних систем, в тому числі мобільні. До переваг даного ігрового рушія можна віднести:

1. Висока якість тривимірної графіки, в порівнянні з іншими доступними рушіями.
2. Система графічного програмування Blueprints.
3. Оптимальна швидкодія та продуктивність.
4. Висока розширюваність та велика кількість безкоштовних плагінів, доповнень, фреймворків, тощо.
5. Широка документація, велика кількість навчального матеріалу та прикладів.
6. Прийнятний спосіб розповсюдження.
7. Власний репозиторій (магазин) з великою кількістю тривимірних моделей, текстур, шейдерів та інших ресурсів.
8. Присутній функціонал для роботи з алгоритмами машинного навчання та безпосередньо з тренуванням моделей.

Недоліки рушія Unreal Engine:

1. Всі доступні варіанти машинного навчання не є вбудованими в сам рушій або офіційно не розповсюджуються самими розробниками Unreal Engine.
2. Система Blueprints не завжди задовольняє всі задачі. Випадку з машинним навчанням, більшість логіки має бути написана вручну.
3. В деякий випадках не найкраща загальна продуктивність.

Підсумовуючи про даний рушій, можна відмітити той факт, що більшість вищезазначених проблем вирішуються додатковою оптимізацією проекту або можуть зустрічатись тільки в значно більших по розміру проектах.

Source Engine – ігровий рушій компанії Valve перша ітерація якого була випущена в 2004 році, а останньою версією на даний момент є друга. Рушій створений на мові C++ та доступний як на персональних комп’ютерах так і на мобільних платформах. До ключових особливостей Source Engine відносять детальну і ефективну систему фізичної симуляції та просунутий рівень графіки на час виходу. Рушій розповсюджується у вигляді наборів розробника під назвою Source SDK, де крім основних функціональних файлів рушія знаходиться редактор сцен Hummer Editor, з допомогою якого можна створювати ігрові карти в форматі BSP (Binary space partitioning). До сильних сторін Source Engine можна віднести:

1. Просунута фізична модель.
2. Висока стабільність та загальна продуктивність.
3. В силу свого віку присутня детальна документація та більшість проблем, з якими можна стикнутись, так чи інакше, мають готові варіанти вирішення.

В свою чергу можна виділити такі недоліки:

1. Остання ітерація Source Engine 2 не є загальнодоступна в своєму повному варіанті.
2. Доступна версія Source не підтримує багато сучасних API.
3. Відсутня офіційна підтримка машинного навчання та фреймворків.
4. Низька здатність рушія до розширюваності.
5. Відсутня підтримка мобільних платформ в доступній версії рушія.
6. Слабка функціональність основного редактору в порівнянні з середовищами інших рушіїв.

В даному випадку Source Engine не є оптимальним рішенням, оскільки для даного рушія не існує можливості повноцінного машинного навчання та не існує певного необхідного функціоналу, зокрема, створення агента.

Unity – тривимірний ігровий рушій розроблений компанією Unity Technologies. З сучасних рушіїв вважається одним з найбільш універсальних, оскільки дозволяє розробляти і проводити адаптацію додатків практично на кожну з сучасних та актуальних платформ. Рушій підтримує більшість сучасних графічних API, серед яких DirectX 12 та Vulkan. Основною мовою програмування є C# код якої транслюється в код C++ при запуску проекту в реальному часі, до версії 2017.1 підтримував створення скриптів з допомогою JavaScript. Основною ідеєю даного рушія є спрощення процесу розробки комп’ютерних ігор шляхом візуальної взаємодії з графічним інтерфейсом, так, в редакторі сцен реалізована підтримка технології drag’n’drop для розташування моделей, текстур та інших об’єктів одразу на ігрову сцену в потрібну позицію. Даний ігровий рушій зарекомендував себе як оптимальну платформу для створення комп’ютерних ігор, математичних та фізичних моделей, та як середовище для машинного навчання.

До переваг рушія Unity можна занести:

1. Простий та при цьому функціональний графічний інтерфейс та середовище розробки.
2. Можливість зручної розгортки проекту на будь-яку з підтримуваних платформ без його глобальної модифікації або оптимізації.
3. Наявність вбудованого репозиторію (магазину) з різноманітними ресурсами для розробки.
4. Офіційна реалізація фреймворку для підтримки машинного навчання під назвою ML-Agents.
5. Велика кількість безкоштовного навчального матеріалу, в тому числі від самих розробників. Наявність широкої документації.
6. Оптимальна форма розповсюдження рушія.

В свою чергу серед недоліків можна відмітити:

1. Слабку реалізацію інструментів для створення графічного інтерфейсу (GUI).
2. В порівнянні з іншими доступними рушіями, більш слабкі графічні можливості.
3. Відсутність інструментів для графічного програмування по аналогії системи Blueprint.

Основною перевагою цього рушія, в контексті даного завдання, є офіційна підтримка інструментів для створення середовища машинного навчання та комп’ютерних агентів і їх впровадження, безпосередньо, в комп’ютерну гру. Зваживши вищевказані переваги та недоліки можна розглядати даний ігровий рушій як претендент на роль платформи машинного навчання та створення комп’ютерної гри на його базі.

Panda3D – платформа для розробки графічних програм на базі мов програмування C++ та C, є безкоштовною і з відкритим кодом. На відміну від популярних рушіїв не є повноцінним ігровим рушієм сам по собі та розповсюджується у вигляді фреймворку для мови програмування Python. По аналогії з іншими рушіями такого типу, даний рушій не володіє вбудованим конструктором сцен та рівнів, однак дозволяє задавати їх безпосередньо в коді. Panda3D підтримує більшість сучасних технологій рендерінгу, а саме: HDR, Normal Mapping, Parallax Bump Mapping, прості моделі шейдингу та інші. Основним графічним API виступає OpenGL, однак, існує підтримка OpenGL ES, Software Rendering та присутня експериментальна версія API Vulkan. Присутня підтримка різних фізичних модель, в тому числі PhysX і PandaPhysics (власна реалізація фізичної системи). Panda3D являється крос-платформеним, підтримує роботу на більшості сучасних операційних системах, також присутня підтримка роботи в браузері. До переваг Panda3D можна віднести:

1. Модульна структура рушія. Дозволяє створювати додатки тільки з потрібним функціоналом, що може економити ресурси обчислювальної системи.
2. Безкоштовність та відкритий код.
3. Базова підтримка машинного навчання.
4. Підтримка основних форматів зображень, текстур та тривимірних моделей (у вигляді плагінів для експорту).

В свою чергу, до недоліків Panda3D відносяться:

1. Порівняно низька якість графіки.
2. Відсутність графічного конструктора.
3. Відсутність повноцінної підтримки мобільних операційних систем, зокрема, Android.
4. Відсутність підтримки розподілених обчислень.

Взявши за основу той факт, що фреймворк Panda3D має базову підтримку технологій машинного навчання, використання даного рушія не є зовсім доцільним, оскільки, в рушії не передбачений функціонал для створення агентів а також відсутні більшість популярних алгоритмів навчання. Додатковою проблемою рушія Panda3D, в контексті даної роботи, є відсутність графічного конструктора який би дозволяв більш зручно створювати рівні та оперувати об’єктами в проекті.

В результаті аналізу доступних рушіїв можна прийти до висновку, що саме рушії Unreal Engine та Unity можна розглядати як основу для проекту. Основні аргументи для використання саме цих рушіїв:

1. Обидва цих рушія представляють з себе як інструмент для конструювання комп’ютерної гри (рівнів та сцен) так і, безпосередньо, як інтегроване середовище для розробки.
2. Unreal Engine та Unity мають достатньо високу популярність, відповідно існує велика кількість навчальної бази, прикладів, документації та інших ресурсів, які б допомогли при розробці.
3. Обидва рушія підтримують інструменти для машинного навчання. Зокрема, присутній функціонал для створення агентів.
4. Кожен з рушіїв є мультиплатформеним та дозволяє розгортувати проект на мобільні операційні системи.
5. Обидва рушії розповсюджуються безкоштовно. Безплатні версії функціонально нічим не обмежені.

Оскільки при первинному аналізі дані рушії показали себе платформами які схожі по функціоналу, виникає потреба виконати більш детальне порівняння між цими двома рушіями, проаналізувавши їхні можливості в контексті виконання поточної задачі.

2.3 Порівняння Unity та Unreal Engine

Перш за все варто відмітити, що різниця в фундаментальному принципі цих рушіїв полягає в їх загальному позиціонуванні розробниками. Так, існує думка, що Unity, в основному, адаптований для швидкої прототипізації проектів та для створення невеликих ігор, на відміну від Unreal Engine який позиціонується як платформа для проектів великого масштабу зі складною архітектурою. Однак, в даній ситуації обидва рушія показують себе як хороші інструменти для створення проектів будь-якої величини. Нижче сформована порівняльна таблиця (2.1) з обома рушіями, в примітках вказані певні висновки щодо позиції:

Таблиця 2.1 – Загальне порівняння Unity та Unreal Engine

|  |  |  |  |
| --- | --- | --- | --- |
| Функція/Параметр | Unity | Unreal Engine | Примітки  автора |
| Основна мова програмування | C#, користувацькі системи візуального програмування (платно) | C++, система Blueprint (візуальне створення скриптів) | Вибір фаворита залежить від суб’єктивних вподобань та попереднього досвіду. |
| Спосіб розповсюдження | Безкоштовно. Ліцензійні виплати стягуються тільки при доході більше ніж $1000000 в рік. | Безкоштовно. Стягнення ліцензійних виплат аналогічна Unity | Паритет між двома рушіями в контексті даного проекту. |
| Можливість до створення прототипів | Суб’єктивно хороша. Середня кількість коду. | Непоганий. Відсутній або мала кількість коду. | В даному випадку фаворитом є Unreal Engine |
| Функція/Параметр | Unity | Unreal Engine | Примітки  автора |
| Рівень графіки та підтримка графічних API | Хороший рівень графіки. Широка підтримка API. | Один з найкращих рівнів графіки. Підтримка більшості API. | За рівнем графіки об’єктивно лідером є Unreal Engine. |
| Кількість та якість документації і навчальних матеріалів | Обширна документація, велика кількість навчальних матеріалів та інших ресурсів | Достатня документація, відносно Unity невелика кількість навчальних матеріалів | Фаворитом є Unity |
| Кількість підтримуваних платформ | Більше 20 підтримуваних платформ. Серед них ПК, ігрові системи, мобільні платформи та VR-гарнітури | До 10 підтримуваних платформ. Стандартний набір серед платформ | Рушій Unity є більш оптимальним вибором при крос-платформеній розробці. |

Існує ще один параметр який не був внесений в таблицю, оскільки він потребує більш детального порівняння тому що є одним з найбільш критичних в контексті виконання роботи. Цим параметром є здатність до машинного навчання, створення тренувального середовища та, безпосередньо, агентів, які в ньому навчаються та функціонують.

Unreal Engine пропонує кілька сторонніх реалізацій алгоритмів машинного навчання. Один з найпопулярніших плагінів є UnrealCV. Його основною задачею є використання в ролі агента компонент Camera рушія, реалізовуючи таким чином роботу комп’ютерного зору. На рисунку 2.4 зображений приклад роботи по розпізнаванню ігрових об’єктів за допомогою плагіну UnrealCV.



Рисунок 2.4 - Робота UnrealCV.   
Автор: <http://docs.unrealcv.org/en/latest/reference/model_zoo.html>

Одним з головних недоліків реалізації машинного навчання в Unreal Engine є низька адаптованість до реалізації мульти-агентних систем, оскільки більшість плагінів та фреймворків які доступні, в ролі агента використовують тільки якийсь один компоненті рушія, наприклад камеру, що погано впливає на гнучкість редактору при розробці.

В свою чергу, рушій Unity має в наявності свою власну реалізацію бібліотеки машинного навчання під назвою ML-agent. Дана бібліотека розроблена як рішення для прискореного налагодження машинного навчання, створення агентів і їх безпосереднє впровадження в гру. На рисунку 2.5 зображена робота мультиагентної системи з використанням плагіну ML-Agents.



Рисунок 2.5 – Мультиагентна система яка впроваджена в комп'ютерну гру.   
Автор: авторська робота

На даній ілюстрації зображений скріншот комп’ютерної гри де водночас працюють два агенти, які ведуть гру один проти одного. Кожен з агентів контролюється своєю моделлю поведінки. Алгоритм навчання який використовувався при тренуванні – навчання з підкріпленням. Приклад такої імплементації демонструє гучність даного інструменту при імплементації, навчених з його допомогою, агентів в гру. Також, дана бібліотека має підтримку розгортки робочої моделі на інші платформи без додаткової модифікації, по аналогії з будь-яким іншим проектом на рушії Unity.

2.4 Проміжні висновки та наступні кроки

Зваживши всі за та проти при порівнянні рушіїв Unreal Engine та Unity, були сформовані такі тези:

1. По критичному параметру, а саме, наявність зручних інструментів та бібліотек для машинного навчання та створення агентів з моделями, саме рушій Unity має найбільшу перевагу, оскільки ML-Agents є найбільш повноцінним та найадаптованішим для подальшої імплементації в комп’ютерну гру.
2. Основна мова програмування для рушія Unity є C#, таким чином створення агента та коректування поведінкової моделі, а також сама розробка комп’ютерної гри проводиться однією мовою. Гра на Unreal Engine, на відміну від Unity, може бути написана водночас на кількох мовах (наприклад логіка агента на Python, а ігрові механіки на C++), що негативно позначається на внутрішній архітектурі проекту та загальній оптимізації.
3. Рівень графіки рушія Unity хоч і поступається Unreal Engine, проте є цілком достатнім для більшості задач. Крім цього даний факт певним чином випливає в перевагу з наступного пункту.
4. Загальний розмір. Перш за все це відноситься до розміру зібраної програми в кожному з рушіїв. Так, типовий розмір запускного файлу простого проекту на UE, зазвичай, складає близько 100 мегабайт, тоді як в Unity не перебільшує 10-20. Збільшений розмір також відноситься і до інших елементів рушія Unreal Engine, тому його використання для проекту такого виду можна вважати доволі надлишковим.
5. Навіть зі стандартним набором інструментів рушія, Unity надає вкрай широкі можливості по створенню ігрових сцен та, в загальному, рівнів і середовищ.
6. При порівнянні доступних сторонніх ресурсів на обох рушіях, можна привести до висновку, що саме для рушія Unity створено найбільша кількість різноманітних моделей, текстур, об’єктів, плагінів, шейдерів та інших бібліотек в порівнянні з Unreal Engine.

Оскільки саме рушій Unity задовольняє більшість вимог, а також є оптимальним вибором для поточної задачі в порівнянні з Unreal Engine, логічним рішенням буде вибір саме Unity як основу для майбутнього проекту.

Так як платформа для машинного навчання та основний рушій були вибрані, наступним кроком буде розробка комп’ютерної гри на його базі та, відповідно, розробка і імплементація комп’ютерного агенту, паралельно визначити та створити оптимальну поведінкову модель доступними інструментами.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА РОЗРОБКА ПОВЕДІНКОВОЇ МОДЕЛІ

3.1 Розробка концепції та вибір жанру

Так як основною тематикою даної роботи є розробка комп’ютерної гри, де б функціонував комп’ютерний агент з певною поведінковою моделлю, слід визначитись з жанром та проробити концепцію, в якій б це було доречним та, так чи інакше, впливало на ігровий процес.

Опираючись на вищевказані обставини, з’являється потреба до формування певного списку вимог для обраної концепції та жанру. Сформований список має такий вигляд:

1. Жанр проекту повинен передбачати безпосереднє задієння агента у ігровий процес в тому чи іншому вигляді.
2. Концепція та ігрова логіка не повинна бути занадто перевантажена зайвими механіками, оскільки збільшення кількість змінних в проекті негативно відобразиться на процесі тренування агента та на загальному рівні продуктивності.
3. Загальний дизайн гри повинен втілювати як легкий до освоєння ігровий процес так і хорошу розважальну функцію.

Таким чином, розглядаючи жанри скрізь дані вимоги, можна прийти до висновку, що розроблена гра може бути виконана в так званому «казуальному» жанрі.

Казуальний ігри – жанр комп’ютерних ігор де основною ідеєю є спрощення ігрового процесу в силу коротких ігрових сесій. Інакше кажучи, даний жанр має на меті робити акцент на доступності для більш широкого кругу користувачів. На рисунку 3.1 зображений приклад сучасної гри такого жанру.



Рисунок 3.1 – «Alto Adventure». Типова сучасна гра казуального жанру.   
Автор: <https://www.irishtimes.com/culture/games/game-reviews/alto-s-adventure-game-review-1.2119852>

Жанр казуальних ігор, історично, бере початок з аркадних жанрів, де не передбачувався надмірно перевантажений правилами ігровий процес, зазвичай, суть таких ігор знаходилась на поверхні та була зрозуміла всім без будь-яких пояснень або інструкцій.

В останній час набрав популярності під-жанр казуальних ігор під назвою «гіпер-казуал» (від англ. Hyper casual). На відміну від класичних казуальних ігор, даний жанр ще більш спрямований на спрощення ігрового процесу, часто шляхом зменшення елементів керування та ступенів свободи для гравця. Найбільше поширення даний під-жанр получив на мобільних платформах та портативних ігрових системах. Частіше за все, гіпер-казуальному жанру властиве максимальне спрощення, наприклад, обмежуючись тільки однією кнопкою або тільки кількома базовими жестами на сенсорному екрані. Зокрема, максимальна простота керування зробила жанр гіпер-казуальних ігор популярним на мобільних платформах. Нижче на рисунку 3.2 зображений скріншот ігрового процесу гри «Stack», популярної гри гіпер-казуального жанру.



Рисунок 3.2 - Гіпер-казуальна гра «Stack».   
Автор: https://www.uplabs.com/posts/stack-ios-app

Оскільки жанр гіпер-казуальних ігор поєднує в собі простоту ігрового процесу та високий потенціал до створення гідного виклику для гравця, при цьому відповідає вищеописаним вимогам, було прийняте рішення розробити гру саме такого жанру.

Так як жанр гіпер-казуальних ігор не має конкретних строгих критеріїв, логічним рішенням буде обрати жанр зі списку звичайних казуальних або аркадних ігор та розробити гру саме по гіпер-казуальним «лекалам». Одним з таких аркадних ігор можна вважати ігри жанру runner (або endless runner). Концепцію даного виду казуальних ігор можна сформувати за такими тезами:

1. Персонаж або об’єкт гравця рухається постійно. Гравцеві надаються вкрай обмежені можливості до керування персонажем (наприклад, тільки стрибок).
2. Зазвичай, гра є безкінечною та припиняється тільки в випадку програшу гравця по тим чи іншим причинам.
3. Частіше за все, метою гри є набирання найбільшої кількості очок та попадання в таблицю рекордів.
4. В проектах такого жанру зазвичай присутні вороги які постійно наздоганяють гравця.

Типовою грою цього жанру є «I Must Run!» компанії Huuuge Games, знімок екрану зображений на рисунку 3.3.

Рисунок 3.3 - Скріншот гри «I Must Run!»   
Автор: http://www.allaboutsymbian.com/reviews/item/12640\_I\_Must\_Run.php

Тут так як і в класичних проектах жанрів runner вся маніпуляція з персонажем відбувається лише у вигляді натискання на екран для стрибка. Таким чином, враховуючи вищевказані тези та особливості жанру runner-ів, даний тип казуальних ігор можна вибирати як основний по ходу розробки жанр.

3.2 Оформлення та побудова середовища

Отже, вибраним жанром став endless runner, тому в зв’язку з цим потрібно продумати дизайн середовища в якому буде проходити ігровий процес та тренування агента, який буде в подальшому імплементований.   
 Даний жанр передбачає високу різноманітність концепцій в плані оформлення зовнішнього вигляду, зокрема, до розташування камери. Прикладом вдалого позиціонування камери для такого жанру є застосування ізометричного виду, який продемонстрований на рисунку 3.4.



Рисунок 3.4 - Приклад ізометричної камери.   
Автор: <https://forum.gdevelop-app.com/t/isometric-platformer-game-example/24757/1>

Одною з ключових переваг такої організації камери полягає в тому, що вона дозволяє легко адаптувати проект для роботи як у вертикальній так і в горизонтальній орієнтації екрану, завдяки чому проект де реалізований такий тип камери буде правильно функціонувати на платформах де застосовуються різні підходи до проведення ігрового процесу, наприклад, на мобільних платформах, де популярним рішенням є використання вертикальної орієнтації екрану. Саме по цій причині вибір ізометричної камери є оптимальним в даному випадку.  
 Наступним кроком є вибір ресурсів та їх адаптація для створення сцени та середовища для навчання. Оскільки жанр гіпер-казуальних ігор передбачає низькі системні вимоги, оскільки орієнтований на широку аудиторію, логічним рішенням буде використання низько-полігональних моделей в стилі «low poly». Таким чином, для побудови основної дороги, по якій буде рухатись головний персонаж, вибір припав на безкоштовний набір асетів під назвою «Racing Kit», а саме, на готовий префаб секції дороги. Її модель зображена на рисунку 3.5.

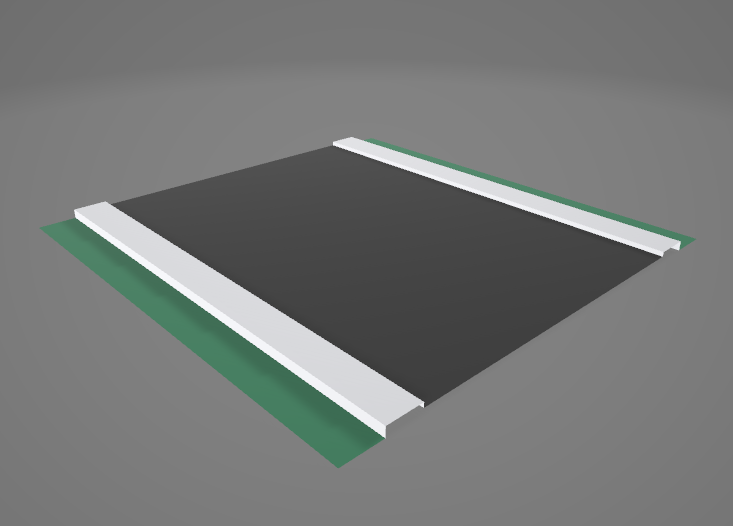


Рисунок 3.5 - Модель "roadStraight".  
 Автор: авторська робота

Сегменти цієї моделі будуть виступати в ролі головної дороги. Для кожного сегменту застосовано стандартний компонент рушія «Box Collider» для обробки колізії.

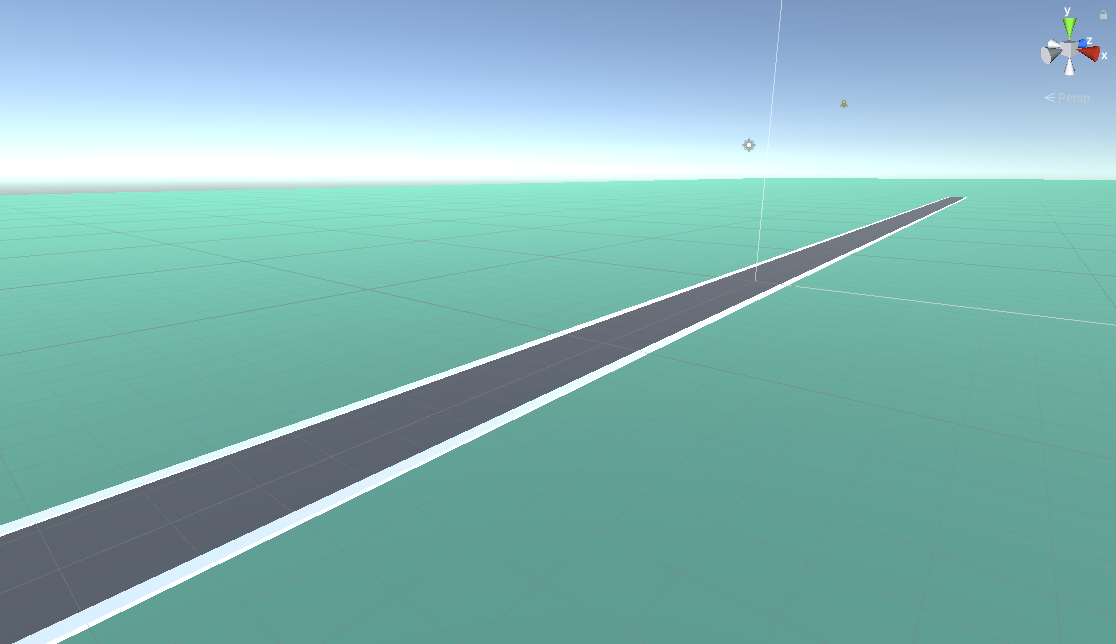
Крім розташування об’єкту дороги на сцену, був розташований базовий об’єкт з текстурою зеленого кольору для імітації землі. Базове середовище для навчання має такий вигляд.

Рисунок 3.6 – Вигляд середовища в редакторі.  
 Автор: авторська робота

Далі логічним рішенням буде обрати та розташувати модель гравця. Оскільки з самого початку були обрані асети з набору для гоночної гри, закономірним буде використати модель автомобіля в ролі головного персонажа, яким буде керувати гравець. Для цього можна використати низько-полігональну модель з безкоштовного набору «Car Kit». Оскільки вона виготовлена тим самим автором, що і попередній набір, не виникає проблеми з несумісною стилістикою. Зображення моделі показано на рисунку 3.7.



Рисунок 3.7 - Модель автомобіля головного героя.   
Автор: авторська робота

Основним ворогом в грі вирішено зробити автомобілі які будуть рухатись в бік гравця, для них буде використана інша модель з вище вказаного набору об’єктів. Підсумовуючи про середовище навчання, можна сформувати такі тези:

1. Розроблене середовище виступатиме «сирцем» та базою для подальшої модифікації.
2. Сформована ігрова сцена з її положенням камери буде коректно відображатись як на горизонтальній, так і на вертикальній орієнтації екрану.
3. В подальшому можна урізноманітнити середовище додавши більше графічних елементів, об’єктів та будь-яких інших додаткових елементів для урізноманітнення ігрового процесу.

Сформована база для навчання комп’ютерного агенту зображена на рисунку 3.8.

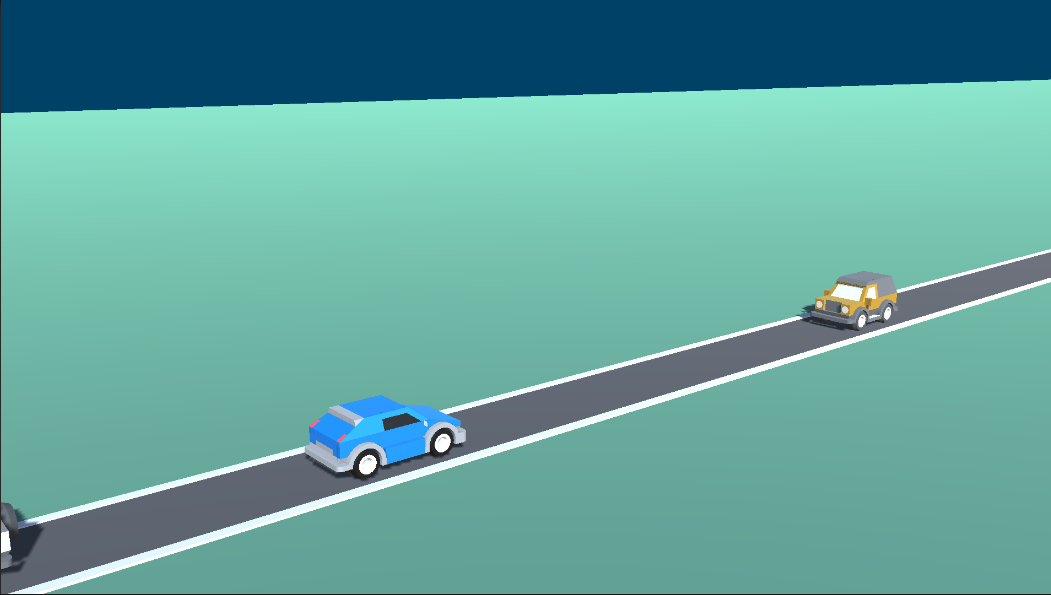


Рисунок 3.8 - Базове навчальне середовище.   
Автор: авторська робота

Оскільки ця сцена, поки що, є навчальною базою, будь-яке додаткове оформлення їх не потрібне.

3.3 Розробка логіки та формування поведінкової моделі

Першою задачею в проектах даного виду виступає створення основної логіки взаємодії об’єктів в середовищі. Оскільки ігровий процес даної гри вирішено зробити безкінечним, логічним рішенням було створити автоматичну систему для створення ворогів від яких би ухилявся персонаж гравця. Засоби рушія Unity дозволяють реалізувати таку функцію шляхом створення пустого об’єкту, присвоєння йому виконуваного скрипта та його розташування на сцені. Скрипт, в свою чергу, реалізує список, який можна заповнити потрібними для створення об’єктами. Вся ця система спрацьовує в діапазоні, який можна самостійно задати у інтерфейсі рушія. На рисунку 3.9 зображено фінальної конфігурації об’єкту «Spawner».

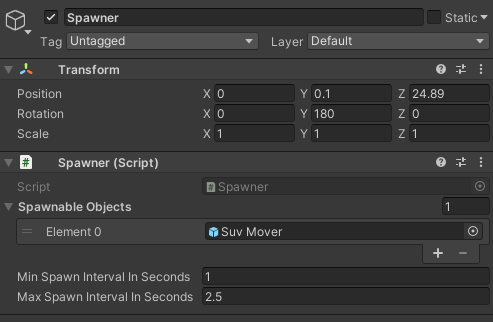


Рисунок 3.9 - Об'єкт "Spawner".   
Автор: авторська робота

Наступним кроком є задання логіки для ворогів, яких буде створювати об’єкт «Spawner». Для цього використовується стандартний метод рушія Unity Rigidbody.velocity для задання вектору об’єкту. В загальному, дане рішення являється доволі поширеним для таких жанрів, тому не вартує більш детального огляду. Більш цікавою є реалізація системи підрахунку балів, тут була застосована система виявлення колізії шляхом прикріплення до моделі ворога невидимої стіни. Зображення префаба продемонстровано на рисунку 3.10.

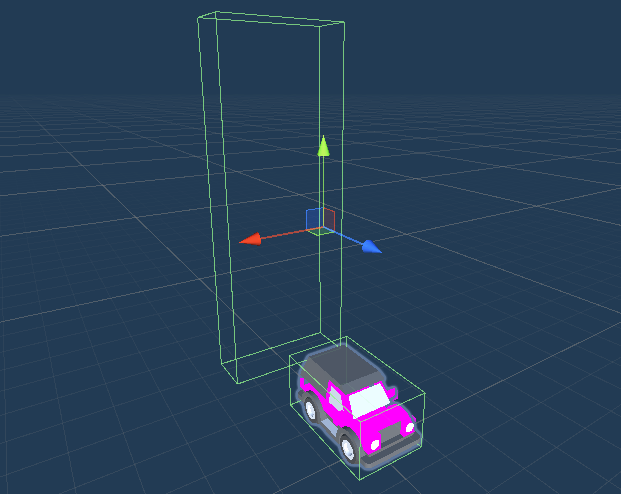


Рисунок 3.10 – Модель противника.   
Автор: авторська робота

Останнім елементом впровадження для початку функціонування навчального процесу став, власне, сам агент. Першою задачею для його функціонування стала система стрибків, виявлення колізій та зарахування балів. Враховуючи досвід попередніх робіт та вивченої документації, найбільш ефективним рішенням стало повторне використання системи тригерів та визначень колізії об’єктів. Основним механізмом стрибків став метод Jump(), в якому компоненту rBody, який приймає фізичний об’єкт рушія, застосовується метод AddForce() з певними параметрами, за допомогою яких визначається сила імпульсу стрибка. Паралельно з цим була додана система визначення колізій та прописана система винагород для нейронної мережі. Зокрема, для колізії між гравцем, вище вказаної невидимої стіни, а також для самих противників. Головні елементи системи зображені на рисунку 3.11.



Рисунок 3.11 – Система колізій.  
Автор: авторська робота

Крім цього, була розроблена запису балів та їх збір для утворення підсумкової статистики під кінець тренування агента.

Останнім кроком для повноцінного функціонування агента стала система його взаємодії та реакції на зовнішні подразники. Опираючись на досвід власної роботи [8], яка велась по близькій тематиці, логічним рішенням було застосування стандартного методу для функціонування агента, тобто, навчання з вчителем. Після проведення тестового навчання, з допомогою інструменту візуалізації TensorBoard, був отриманий такий результат (рис. 3.12).

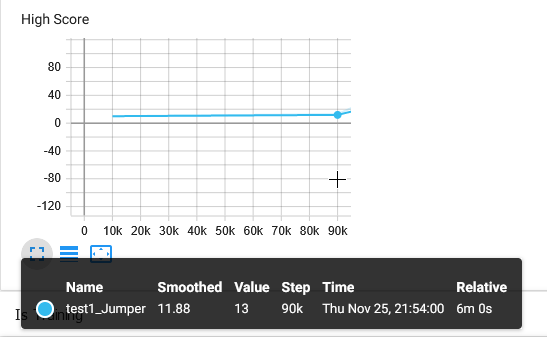


Рисунок 3.11 - Результати тестового "пробігу".   
Автор: авторська робота

Загальний час тестового навчання складав близько семи хвилин. Опираючись на статистику можна було б прийти до висновку, що стандартний підхід до навчання агента показує себе доволі ефективно, однак таке враження є декілька оманливим, оскільки не відображають всю суть роботи такої системи. Нижче сформована певна кількість тез, які опираються на суб’єктивні спостереження автора:

1. Фінальна поведінка після навчання не була хоч якось організована та була схожа на хаотичні дії без будь якої логіки.
2. Більшість результатів даної системи були досягнуті за допомогою удачі.
3. При подальшому навчанні результати ставали кращими, однак методи якими вони досягались були не зовсім коректними. Так, складалось враження, що система намагається «вгадати» проміжок часу коли об’єкт «Spawner» створить нового ворога і намагається його перестрибнути.
4. В цілому, даний підхід залишав враження того, що модель не підпорядковується під правила навчання (навчання з підкріпленням), а працює сама по собі та видає вкрай непередбачувані результати, які більш схожі на результати навчання по алгоритмам без вчителя.

Таким чином виникала певна кількість очевидних проблем з якими можна стикнутись в випадку якщо використовувати такий підхід як основний. Серед них була неможливість використання даної моделі поведінки з застосуванням інших параметрів середовища, наприклад, діапазоном появлення ворогів, в даному випадку для кожної такої конфігурації середовища доведеться проводити повторне навчання агента. Ще однією очевидною проблемою такого підходу є значні витрати часу, оскільки для формування поведінкової моделі мінімально базового рівня функціональності, доведеться провести вкрай велику кількість ітерацій навчання, що демонструє низьку ефективність навчання та, відповідно, високі витрати ресурсів.

Для вирішення цієї проблеми, слід звернутись до прикладів з реального життя. На даний момент бурно розвиваються та користуються великою популярністю системи автономного керування транспортом. Схематичний приклад таких систем зображений на рисунку 3.12.

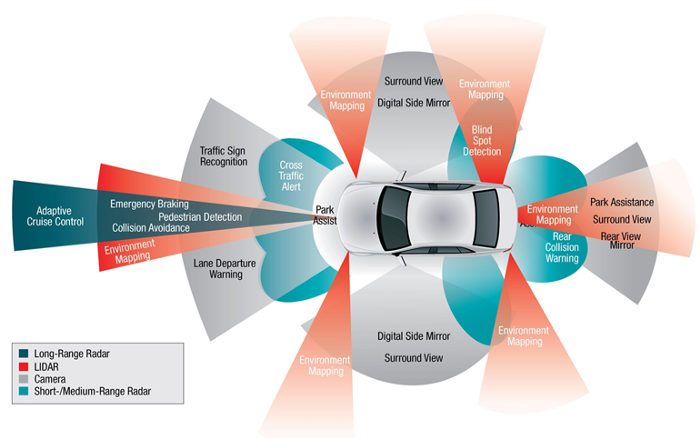


Рисунок 3.12 - Комплекс апаратних пристроїв для автономного керування.   
Автор: <https://towardsdatascience.com/how-to-make-a-vehicle-autonomous-16edf164c30f>

Частіше за все, дані системи виступають в ролі «очей» транспортного засобу, де інформація з них обробляється бортовою обчислювальною системою та після цього приймаються рішення про подальші дії. Виходячи з цього, можливим рішенням вищевказаної проблеми може бути використання певного аналогу радара для визначення відстані від противника.

Фреймворк ML-Agents має в наявності компонент під назвою «Ray Perception Sensor» який, по суті, реалізовує технологію рейкастинга (англ. Raycasting), яка, в свою чергу, працює за принципом проекції променів на поверхню для визначення глибини. Даний компонент не є повноцінним замінником систем радарів, однак, може служити його оптимальним аналогом в даному випадку. В загальному, компонент Ray Perception Sensor, застосовується для імітації роботи складних систем, наприклад, симуляції ходьби та не використовується при розробці ігор в жанрі runner.

Основною ідеєю є застосування компоненту Ray Perception Sensor на основного агента для його коректної реакції при оцінці відстані до противника та, відповідно, утворення правильної поведінкової моделі. При додаванні даного компоненту на агента отримуємо такий результат (рис 3.13).

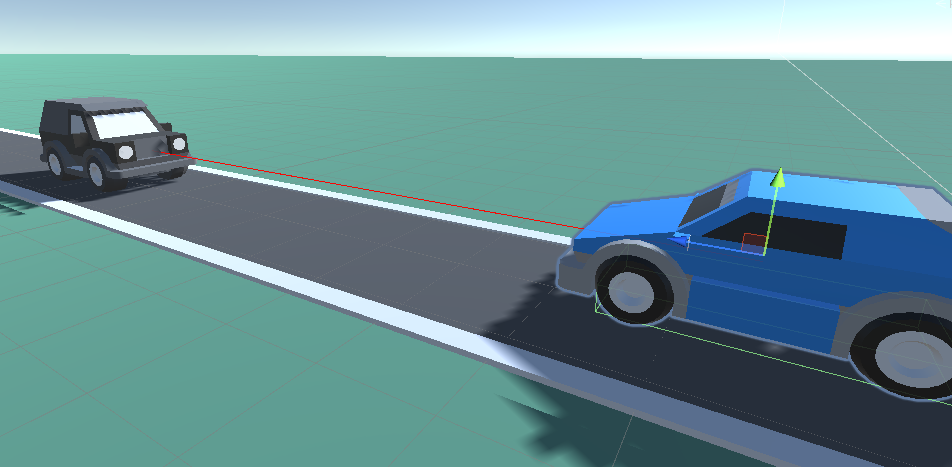


Рисунок 3.13 – Робота Ray Perception Sensor 3D.  
Автор: авторська робота

На ілюстрації помітно, як при приближені противника промінь, який виходить з переднього бампера автомобіля головного героя, став червоного кольору, це означає, що сенсор розпізнав приближення та подав сигнал мережі. Даного результату вдалось досягти шляхом задавання компоненту Ray Perception Sensor в графі «Detectable Tags» тегу «Mover», яким були позначенні префаби всіх зустрічних автомобілів. Одним з переваг цього компоненту є спрощене налаштування для коректної роботи, так, в даному випадку не знадобилось додатково переписувати логіку, компонент автоматично зв’язався з фреймворком та буде передавати йому дані по ходу навчання.

Отже, після проведення навчання з компонентом Ray Perception Sensor 3D були отримані такі результати (рис. 3.14).



Рисунок 3.14 – Наслідок застосування Ray Perception Sensor.   
Автор: авторська робота

Судячи по статистиці середній максимальний результат даного методу більший на 26,22. Суб’єктивна оцінка при роботі:

1. Робота агента стала стабільною та очікуваною.
2. Кількість похибок або не коректної поведінки зведені до мінімуму.
3. Загальне враження полягає в тому, що поведінкова модель скоректована правильно та повноцінно функціонувати в даному проекті.

Крім цього, не зайвим буде визначити наскільки гнучкою є дана модель поведінки в контексті різних ігрових ситуацій. Для цього було створено ще одного противника, який знаходиться перед гравцем та переслідує його. При колізії з ним головний герой програє. Його модель зображена на рисунку 3.15.

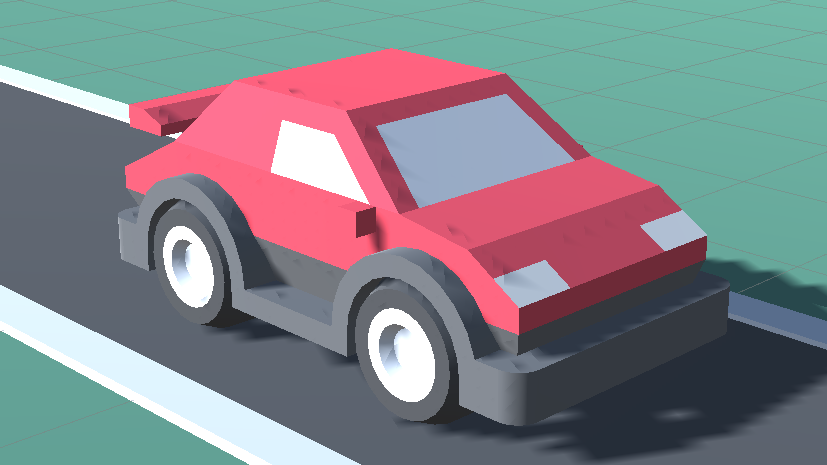


Рисунок 3.15 – Модель переслідувача (неігровий персонаж).   
Автор: авторська робота.

Паралельно з розташуванням переслідувача, ми відберем у об’єкта гравця натреновану поведінкову модель та застосуємо її на переслідувача. Базовий ігровий процес матиме такий вигляд (рис. 3.16)

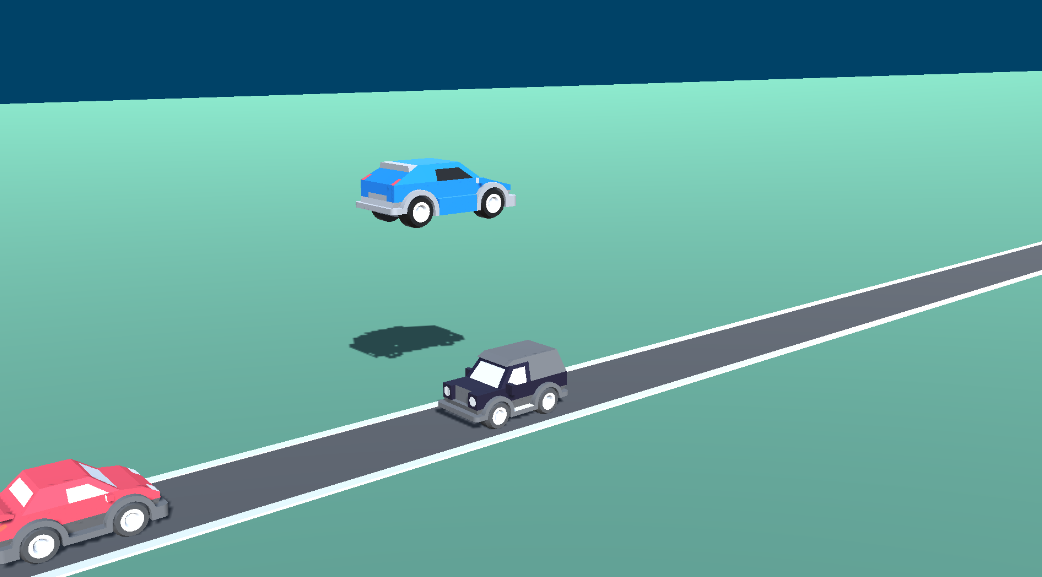


Рисунок 3.16 – Ядро ігрового процесу.  
Автор: авторська робота.

В результаті, без будь якої адаптації, поведінкова модель яка натренована шляхом Ray Perception Sensor 3D, змогла коректно функціонувати.

3.4 Візуальне оформлення та додаткові функції

Оскільки базове ядро гри було розроблено, прийшла черга візуального оформлення гри та поглиблення геймплею.

Однією з найголовніших функцій аркадних та казуальних жанрів ігор було запис найкращого результату. Оскільки дана функція, попередньо, була реалізована в коді, залишилось тільки його вивести. Для цього використовується плагін під назвою TextMesh Pro, який є, де-факто, стандартом для Unity. Текст з кількістю очок розташовується на екран методом використання стандартного компоненту «Canvas» (полотно).

Наступним рішенням є додавання динамічних елементів, які б рухались на фоні. Дані дії спрямовані для забезпечення додаткової динамічності ігрового процесу. Для цього можна застосувати об’єкти з вищевказаного набору «Racing Models» які представляють з себе низько-полігональні моделі ліхтарного стовпа та дерев. Для імітації руху був застосований стандартний компонент Unity під назвою Animator та створено просту анімацію, яка працює по колу. Її схема зображена на рисунку 3.17.

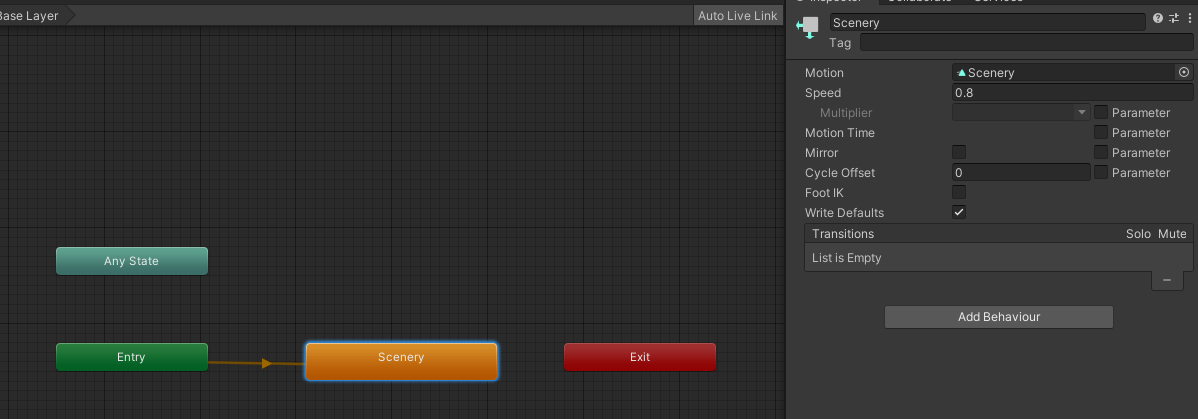


Рисунок 3.17 - Анімація для компоненту Scenery.   
Автор: авторська робота

Щоб поглибити геймплей було вирішено додати бонусну систему. Суть її полягає у додаванні спеціального об’єкту в елемент «Spawner», де після створення, по аналогії з противником, об’єкт створюється та починає рух в сторону гравця. При колізії гравця з об’єктом проводиться збільшення дистанції від переслідувача, а, оскільки, занадто сильне зближення приводить до кінця гри, дана механіка виступає в ролі додаткової глибини ігрового процесу. На рисунку 3.18 зображена система бонусів в дії.

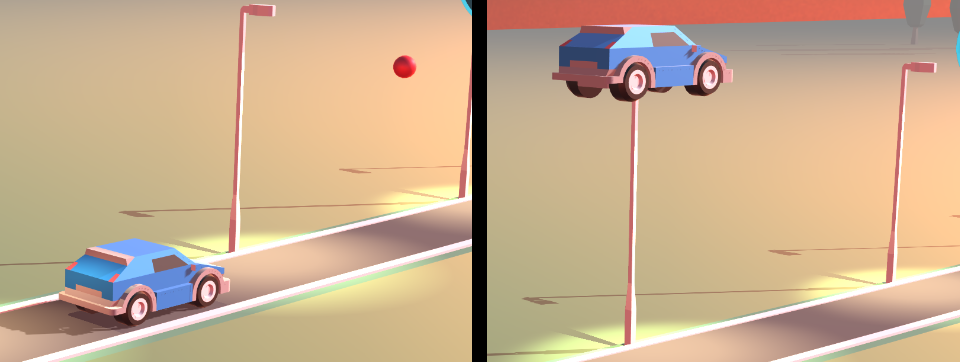


Рисунок 3.18 – Робота бонусної системи.   
Автор: авторська робота

При колізії бонусний об’єкт знищується, а об’єкту гравця надається невеликий поштовх.  
 Останнім штрихом стало налаштування освітлення для сцени. Перш за все, було замінено стандартний скайбокс користувацьким з безкоштовного набору «Skybox Series Free», який був доступний в фірмовому репозиторії Unity Asset Store. Кожен скайбокс з набору постачається разом із налаштованою кольоровою схемою освітлення, таким чином, при його застосуванні можна отримати оптимально скореговане за гаммою освітлення. Додатковою деталлю буде додавання джерел світла кожному з стовпів. Вони присутні на рисунку вище, однак є мало помітними на статичному скріншоті.

В загальному, після впроваджених змін до базового ігрового процесу, натренована методом рейкастингу модель показала себе задовільно в роботі. На рисунку А.1 зображений скріншот кінцевого ігрового процесу, в режимі гри проти комп’ютера. Також доступний режим демонстрації, де головним гравцем керує натренована модель поведінки, його можна активувати шляхом мінімальної маніпуляції в редакторі рушія.

**ВИСНОВКИ**

В ході виконання роботи були вивчені та проаналізовані методи машинного навчання та їх вплив на поведінкову модель комп’ютерного агента. На базі модифікованого алгоритму машинного навчання була натренована і сформована поведінкова модель, яка була імплементована в комп’ютерну гру жанру runner у вигляді суперника.

Під час розробки модифікація у вигляді компоненту Ray Perception Sensor довела свою ефективність в порівнянні з класичними підходами до конфігурування навчального середовища та агента. Сформована за цим підходом поведінкова модель вдало виконувала поставлені для неї задачі, в тому числі, задачі які попередньо не передбачались.

З іншої сторони, даний підхід не став повністю універсальним рішенням для будь якої задачі. Так, в ході виконання було виявлено декілька проблем, які можуть потребувати додаткового опрацювання, наприклад, некоректна реакція при агента при великій кількості зовнішніх подразників.

Особливості вищевказаної поведінки можна усунути кількома шляхами, серед яких є збільшення кількості сенсорів для більш детального контролю над навколишньою ситуацією або створення власного аналогу сенсора з більш ефективними алгоритмами визначення.

В цілому, даний підхід до формування поведінкової моделі агента має значний потенціал до розвитку та імплементації в комп’ютерні ігри різних жанрів та різної механіки взаємодії, обмежуючись не тільки жанром runner.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Aurélien Géron. Hands-On Machine Learning with Scikit-Learn and TensorFlow, 2017.
2. А.П. Еремеев, А.А. Кожухов. Реализация методов обучения с подкреплением на основе темпоральных различий и мультиагентного подхода для интеллектуальных систем реального времени, 2016.
3. Я. А. Туровский, С.Е. Гриднев, Ю.А. Ипполитов. Моделирование поведения человека на основе анализа отклонения от оптимального поведенческого паттерна, 2016.
4. Николаенко С., Кадурин А., Архангельская Е. Глубокое обучение, 2018.
5. Chelsea Finn, Sergey Levine, Pieter Abbeel, Guided Cost Learning: Deep Inverce Optimal Control via Policy Optimization, 2016.
6. Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, Anind K. Dey, Maximum Entropy Inverse Reinforcement Learning, 2008.
7. Gross C. G., Genealogy of the “grandmother cell”, 2002.
8. Ільницький М., Методи глибинного навчання при побудові агентів комп’ютерної гри, 2020.
9. И. Ю. Сотников, И. В. Григорьева, Адаптивное поведение програмных агентов в мультиагентной компьютерной игре, 2014.
10. Рыбина Г. В., Паронджанов С. С., Технология построения динамических интеллектуальных систем, 2014.
11. Осипов Г. С. Методы искусственного интеллекта, 2011.
12. Sarenko A., Detlor B., Intelligent agents as innovation, 2004.
13. Shteingart H., Neiman T., Loewenstein Y., The Role of First Impression in Operant Learning, 2013.
14. Sebastiano M. Cossu, Beginning Game AI with Unity, 2020.
15. Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra, Continuous control with deep reinforcement learning, 2016
16. Sellers M., Advanced Game Design: A System Approach 1st Edition, 2017
17. Lanham M. Hands-On Deep Learning for Games: Leverage the power of neural network and reinforcement learning to build intelligent games, 2019.
18. Pachall A., Developing AI with Unreal Engine 4, 2020.
19. Andreas C. Muller, Guido S., Introduction to Machine Learning with Python: A Guide for Data Scientists, 2016.
20. Hocking J., Shell J., Unity in Action: Multiplatform Game Development in C# with Unity 5, 2018.
21. Флах П., Машинное обучение, 2015.
22. Sapio F., Hands-On Artificial Intelligence with Unreal Engine: Everything you want to know about Game AI using Blueprints or C++, 2019.
23. Sanders A., An Introduction to Unreal Engine 4 (Focal Press Game Design Workshops), 2016.
24. Dean A. Pomerleau, ALVINN: An autonomous land vehicle in a neural network, 1986.

**ДОДАТОК**



Рисунок А.1