ПРИКЛАДНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК,
БЕЗПЕКИ ТА МАТЕМАТИКИ, 2025, № 5

УДК 123.45

ПОРІВНЯННЯ НАВЧАННЯ З ПІДКРІПЛЕННЯМ В ІГРОВИХ РУШІЯХ UNREAL ENGINE 5 ТА UNITY

Лайтарук І.Ф. (ORCID: 0009-0002-9832-7759),
Гришанович Т.О. (ORCID: 0000-0002-3595-6964)
Онищук О.О. (ORCID: 0000-0002-8342-3011)
Волинський національний університет імені Лесі Українки

COMPARISON OF REINFORCEMENT LEARNING IN GAME ENGINES UNREAL ENGINE 5 AND UNITY

Laitaruk I.F., Hryshanovych T.O., Onyshchuk O.O.
Lesya Ukrainka Volyn National University

Abstract. This paper presents a comparative analysis of reinforcement learning implementation in two leading game engines: Unreal Engine 5 and Unity. The study evaluates the integration simplicity, training effectiveness, and tool support of each platform by examining similar open-source projects where autonomous car agents learn to navigate racetracks using the Proximal Policy Optimization (PPO) algorithm. In Unreal Engine 5, training is achieved through the new Learning Agents plugin, leveraging Blueprint scripting and real-time physics-based simulation, while in Unity, the ML-Agents Toolkit is used with C# scripting and external Python-based training. The analysis includes environment setup, reward structures, agent perception systems, and debugging tools. Additionally, user experience and technical documentation quality are assessed. The results provide insights for developers and researchers aiming to choose an optimal platform for integrating RL into interactive simulations or games.

Keywords: Reinforcement learning (RL), Unreal Engine 5 (UE5), Unity, machine learning (ML), agents, Blueprints, C#, Proximal Policy Optimization (PPO).

Вступ. Поняття навчання з підкріплення в ігрових рушіях. *Навчання з підкріпленням* (reinforcement learning) – це галузь машинного навчання, яка базується на процесі обрання конкретних дій програмними агентами в середовищі задля максимізації кумулятивної винагороди. Якщо в керованому навчанні модель навчається на заданих даних, то тут вона навчається на послідовності своїх спроб та помилок. Вперше навчання з підкріпленням у комп'ютерній грі використовується Річардом Саттоном та Джеральдом Тесауро, коли гра TD-нарди (1992) [1] показала значний потенціал росту штучного інтелекту за допомогою методу часових різниць. Без явного програмування стратегій, комп'ютер зміг досягнути рівня професійних гравців.

Для ознайомлення необхідно розглянути базові поняття, які вводиться в навчанні з підкріпленням (рис. 1). Агент (agent) – образний діяч, який обирає дію для переходу в наступний стан. У комп'ютерних іграх агентом може виступати будь-яка сутність, яка керує ігровим процесом (наприклад, суперник гравця-людини або неігрові персонажі NPC). Середовище (environment) – контекст, у якому діє агент. Це все з чим взаємодіє агент та дає винагороду незалежно від її ефекту (наприклад, ігровий світ, правила гри, персонажі). Стан середовища (state) – це інформація про середовище в момент часу (якщо покрокова гра, то хід), яка важлива для формування винагороди агента. Спостереження (observations) – це підмножина стану середовища, тобто та інформація про середовище, яка важлива для формування винагороди і відома агенту. Дія (action) – це акт зміни стану середовища агентом задля отримання винагороди. Винагорода (reward) – це позитивний або негативний результат зміни стану середовища дією агента, який повинен зрозуміти, яку винагороду він може отримати "підкріплена" вибором його дії. Стратегія (policy) – поведінка (множина дій), якої навчається агент зі спостережень.

Рис. 1. Модель навчання з підкріпленням з усіма його елементами

Дію обирає агент із запрограмованої функції, яка приймає спостереження як вхідні дані, а стан середовища може визначатися стохастично, тому така модель є марковським процесом вирішення (МПВ) [2]. Ця концепція лежить в основі більшості алгоритмів навчання з підкріпленням.

Існує безліч прикладів успішного застосування таких моделей у комп'ютерних іграх. Штучний інтелект гри AlphaGo (рис. 2 – а) переміг чемпіона світу в настільній грі Ґо в 2016 році [3, 4]. За допомогою дерева ухвалень рішень Монте-Карло, гра моделювала потенційні кроки та їхні результати, щоб знайти найбільш перспективні дії. Спочатку розробники використовували кероване навчання для імітації дій схожих до професійних гравців, а пізніше покращенню сприяли партії гри самі з собою, що допомогло відшліфувати стратегію. AlphaGo підкреслила здатність RL працювати в суміші з іншими моделями, а також обробляти багатовимірні простори станів і планувати на великі горизонти.

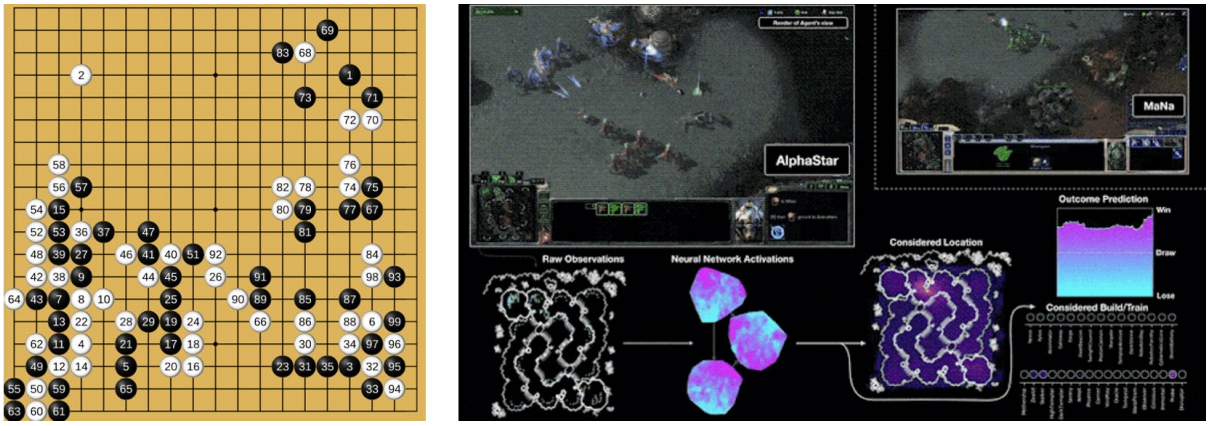

Рис. 2. А (ліворуч) – перші 99 ходів гри в AlphaGo;
Б – принцип роботи AlphaStar з усіма покроковими етапами

AlphaStar – це програмне забезпечення, розроблене як RL агент для гри в StarCraft II у 2018 році [5]. Особливістю є те, що це стратегія в реальному часі з високим рівнем складності та глибини ігрових механік. AlphaStar використовує глибоке навчання з підкріпленням, багатоагентне навчання та імітаційне навчання з даних ігрового процесу реального гравця (рис. 2 – б). Це продемонструвало вміння довгострокового планування та багатозадачності, що дозволило штучному інтелекту досягти рівня гросмейстера в грі StarCraft II.

Актуальність використання та дослідження навчання з підкріпленням в ігрових рушіях Unreal Engine 5 та Unity зростає швидкими темпами, оскільки гравці очікують більш адаптивну та реалістичну поведінку NPC та складніші нетривіальні ігрові локації. Навчання з підкріпленням є одним з методів вирішення поставлених проблем за допомогою інтеграції в ігрові рушії. Вперше RL як окремий плагін "Learing Agents" для UE представлено у версії 5.3 (вересень 2023 року), а для Unity ще у 2017 році в "ML Agents Toolkit" та стрімко вдосконалюється з кожним роком. Ці інструменти використовуються в багатьох академічних дослідженнях, зокрема для навчання стратегічних та кооперативних агентів, модифікацій рівнів, які змінюються залежно від стилю гри користувача, а також у симуляції дронів (AirSim) [6] та дослідження RL в реалістичних фізичних умовах. Наразі активно досліджується комбінація RL з традиційним AI, нейронними мережами та еволюційними алгоритмами. Це дає змогу створювати геймдизайнерські інструменти нового покоління: AI-агенти, які самі навчаються створювати квести, оптимізувати баланс, або навіть допомагають розробникам.

Методологія дослідження. Робота проводиться на основі аналізу простоти та якості інтеграції RL бібліотек з Unreal Engine 5 та Unity. Для порівняння якості навчання агента, кількості ітерацій до досягнення цілей та поведінки агента в нових умовах використовуються схожі проекти з відкритих ресурсів, а саме навчання агента-автомобіля водінню по гоночній трасі. На UE5 код для навчання написаний на візуальній мові скриптування Blueprint, а на Unity – C#. В обох проєктах навчання нейронної мережі відбувається на основі алгоритму PPO (Proximal Policy Optimization). Опис роботи відбувається покроково, але лише з ключовими етапами, тому його не можна сприймати як повноцінний туторіал для точного відтворення результатів і враховується оцінка зручності роботи, доступності інструментів та якості підтримки. Розглянуті аспекти алгоритмів, бібліотек та ігрових рушіїв зрештою узагальнені в невелику таблицю (табл. 1).

Отже, *мета дослідження* — провести порівняльний аналіз якості, зручності інтеграції та ефективності навчання з підкріпленням у рушіях Unreal Engine 5 та Unity на основі практичних прикладів тренування агентів-автомобілів з використанням алгоритму PPO.

Результати роботи. Навчання з підкріпленням в Unreal Engine 5 та Unity. У результаті проведеного дослідження було проаналізовано процес навчання з підкріпленням в ігрових рушіях Unreal Engine 5 та Unity. В обох середовищах агент-автомобіль навчався проходити гоночну трасу за допомогою алгоритму PPO (Proximal Policy Optimization). Для Unreal Engine 5 використовувався плагін Learning Agents, що дозволяє задавати спостереження, дії, винагороди та стратегії безпосередньо через Blueprint, з подальшою взаємодією з Python-моделлю на основі PyTorch. У Unity був застосований набір інструментів ML-Agents Toolkit, де агент реалізовувався через клас Agent, а навчання проводилося за допомогою зовнішнього Python-сервера. Обидва рушії надали можливість гнучко керувати параметрами навчання, проте різняться за рівнем складності налаштування, зручністю візуального дебагу та швидкістю досягнення результатів. Узагальнені результати порівняння можна переглянути в таблиці 1.

Таблиця 1.
Порівняння навчання з підкріпленням в UE5 та Unity

Критерій	**Unreal Engine 5**	**Unity**
Інструмент RL	Learning Agents (з версії 5.3)	ML-Agents Toolkit (з 2017 року)
Мови реалізації	C++, Blueprint, Python	C#, Python
Інтеграція з RL-бібліотеками	PyTorch через зовнішній процес	Пряма інтеграція з Python/PyTorch
Налаштування агента	Менеджер агентів + інтерфейси спостереження, дій, винагород	Наслідування від класу Agent + методи CollectObservations, OnActionReceived, Heuristic
Візуальний дебаг	F8 для візуалізації, Visual Logger	Ray Cast Gizmos, HUD, Unity консоль, Heuristic Mode
Час навчання (для прикладу з авто)	≈ 2 години для базового проходження	≈ 5–6 днів для повного проходження траси (70 чекпоінтів)
Паралельні агенти	До 32 в одному сеансі	У прикладі — лише один агент, але підтримується --num-envs для паралельності
Гнучкість у зміні середовища	Висока (сплайни, фізика, великі сцени)	Висока (можна додавати колайдери, чекпоінти, сенсори тощо)
Пороговість входу	Вища (необхідні знання C++/Blueprint + налаштування менеджера)	Нижча (докладна документація, простіша структура коду)
Підтримка і документація	Новий інструмент, менше туторіалів	Велика спільнота, докладна документація, відкриті приклади
Алгоритм навчання	PPO + вручну налаштовані винагороди	PPO + GAIL + CheckPoint-система
Застосування стратегій (Inference Mode)	Через Data Asset, без Python	Через .nn файл, запуск без Python
Сфера використання	NPC, анімація на фізиці, автотестування, ігровий AI	NPC, поведінкові моделі, симуляції, ігрові прототипи, навчання стратегій

Порівняльна таблиця узагальнює ключові аспекти реалізації навчання з підкріпленням в Unreal Engine 5 та Unity, зокрема: інструменти, мови реалізації, структуру агента, дебаг-можливості, складність інтеграції, підтримку паралельних середовищ, швидкість навчання, алгоритмічну гнучкість та якість документації. Як показує таблиця, Unity забезпечує ширший доступ до навчальних матеріалів, легший поріг входу та стабільніший робочий процес для початківців. Натомість Unreal Engine 5 демонструє більший потенціал у складних фізичних симуляціях, завдяки високій візуальній якості, широкій інтеграції з Blueprints та можливості глибокої оптимізації агента в реальному часі. Такий аналіз дозволяє розробникам свідомо обирати платформу відповідно до цілей та ресурсів проєкту.

Навчання з підкріпленням в Unreal Engine 5 реалізується за допомогою плагіна "Learning Agents" [7, 8]. Він дає змогу розширити або замінити традиційний ігровий штучний інтелект, наприклад, створений за допомогою дерев поведінки (Behaviour Tree) або стейт-машин. Окрім того, плагін дозволяє використовувати підходи імітаційного

навчання (IL). У довгостроковій перспективі "Learning Agents" прагне бути корисним у низці модулів, включаючи ігрових NPC, анімацію на основі фізики та автоматизоване тестування якості. Плагін не є системою машинного навчання загального призначення. Кожен аспект плагіна було створено з урахуванням прийняття рішень персонажем, тому його не доцільно використовувати як Generative AI зображень, аудіо, рівнів та для спілкування з NPC.

Плагін написаний на C++ (з підтримкою Blueprint) і скриптами Python. Він надає функціональні можливості для визначення спостережень/дій і структури нейронної мережі, а також керування навчанням та формуванням стратегій. Під час навчання процес UE співпрацює із зовнішнім процесом Python, на якому запущений PyTorch.

Для реалізації використовується шаблонний проект Vehicle із автомобілями та гоночною трасою. Встановлення плагіна відбувається в UE редакторі, в меню Edit → Plugins та обирається "Learning Agents".

Основний клас для управління агентами це Learning Agents Manager. Менеджер є акторським компонентом, навколо якого будується решта агентів навчання. Він діє як структура даних, яка зберігає посилання на різні агенти, а також як місце для визначення логіки навчання. Оскільки дії відбуваються в реальному часі з використанням симуляції фізики, потрібно, щоб дія встигла вплинути на стан гри, тому необхідно змінити інтервал тіку менеджера (0.1 сек), що дозволить ефективно контролювати, як часто агенти приймають рішення. Кожен агент повинен зареєструватися в менеджері, для того, щоб його було розпізнано. Декілька агентів можуть оброблятися в пакеті задля оптимізаційних заходів. Агентами можуть бути будь-які екземпляри класу UObject. У цьому прикладі використовується SportsCar_Pawn клас для агентів та BP_SportsCarManager для менеджера агентів. Після цього менеджер ставиться на рівень, і автомобілі реєструються як агенти. Значна частина функціональних можливостей менеджера надходить від Manager Listeners. Ці об'єкти розширюють менеджер, дозволяючи йому виконувати різноманітні завдання, такі як збір спостережень, виконання дій, навчання та запис даних у файли (Interactor, Policy та Trainer).

Інтерактор відповідає за визначення того, як агенти менеджера взаємодіють зі світом через спостереження та дії. Усі агенти для даного менеджера поділяють спільну схему спостережень і дій. Інтерактор має чотири основні функції, які потрібно перевизначити для гри: SpecifyAgentObservation, SpecifyAgentAction, GatherAgentObservation і PerformAgentAction. Під час виконання функції SpecifyAgentObservation додаються спостереження до схеми агента, викликаючи його функцію Specify ___ Observation (рис. 3). Ця функція буде викликана менеджером один раз під час початкового налаштування. Далі визначається, як кожне із спостережень збирається зі стану гри всередині функції GatherAgentObservation. Для спостереження за місцезнаходженням потрібно викликати MakeLocationAlongSplineObservation.

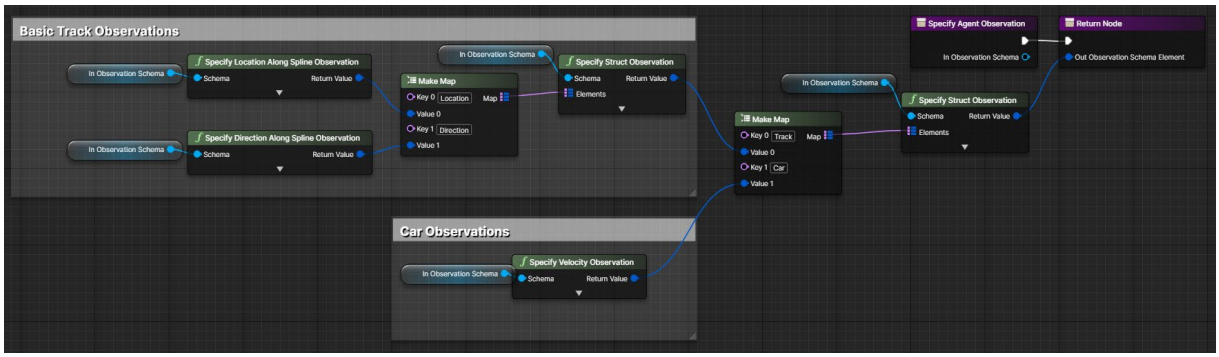

Рис. 3. Blueprint скрипт функції SpecifyAgentObservation

Для того, щоб автомобіль міг успішно проїхати трасою, потрібно надати йому інформацію про те, де він знаходиться відносно траси, та інші стани, такі як його поточна швидкість. Гоночна траса представлена у вигляді сплайну, а "Learning Agents" надає допоміжний компонент сплайнів, щоб спростити спостереження за положенням уздовж сплайну (рис. 4 – а). Після завершення налаштування спостереження агенти тепер зможуть відчути, де вони знаходяться на трасі (рис. 5).

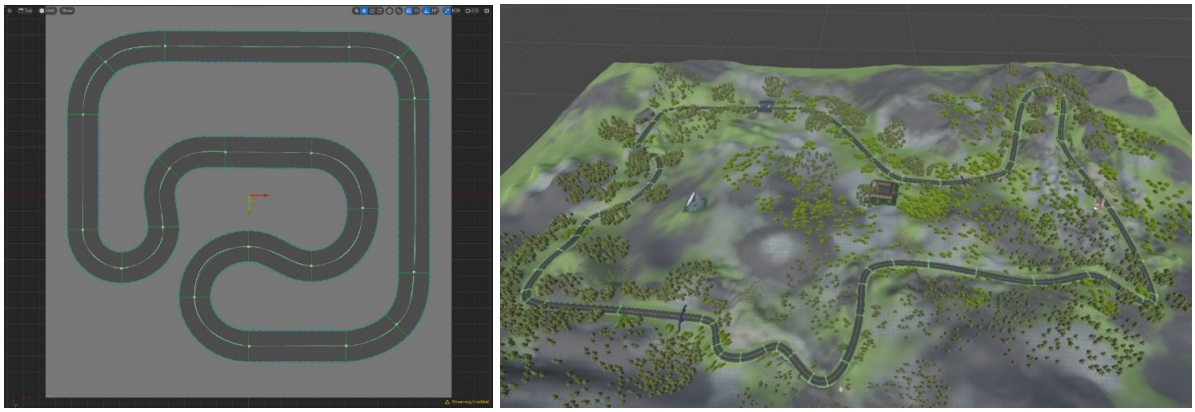

Рис. 4. А (ліворуч) – Гоночна траса в проєкті Unreal Engine 5;
Б (праворуч) – Гоночна траса в проєкті Unity

Дії схожі на спостереження – реалізується подія SetupAction (рис. 6), яка оголошує дії, які можуть виконувати агенти, а потім подію GetActions, де фактично береться значення дії з нейронної мережі та застосовується до конкретного агента. Для простоти налаштовуються дві дії: газ/гальмування та рульове керування.

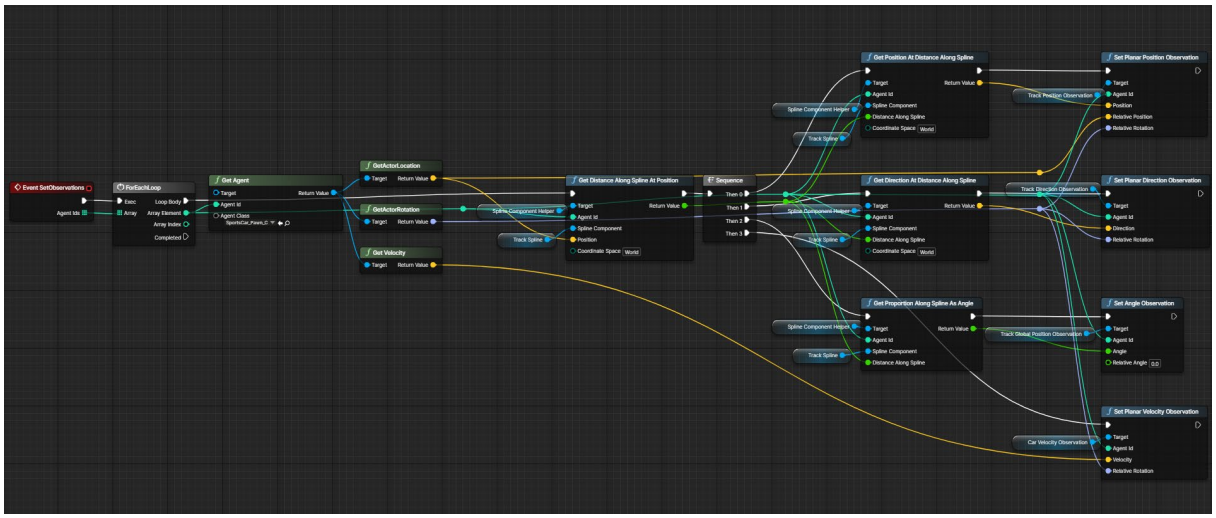

Рис. 5. Blueprint скрипт налаштування спостереження агентів

Стратегія (policy) – це компонент, який відповідає за втілення спостережень агентів у дії. Це робиться шляхом введення даних спостережень у нейронну мережу, яка оброблятиме значення для формування дій. Для цього створюється клас BP_DrivingPolicy з батьківським класом LearningAgentsPolicy.

Останнім компонентом є клас BP_DrivingRLTrainer від LearningAgentsTrainer. Як випливає з назви, цей клас відповідає за навчання агентів виконувати правильні дії в грі,

враховуючи спостереження, які він бачать. Для навчання з підкріпленням реалізовуються дві речі: винагороди та завершення. Для винагород створюється подія SetupRewards, яка позитивно впливає на швидкість уздовж сплайну та негативно (штраф) за виїзд із траси.

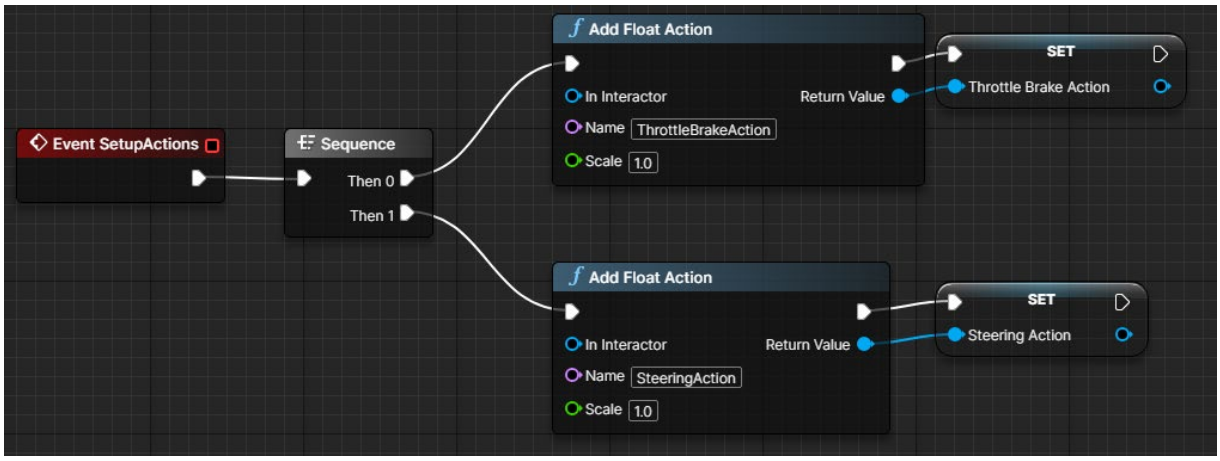

Рис. 6. Blueprint скрипт події SetupActions

UE використовує поняття епізоду як послідовності спостережень, дій і винагород від початку до кінця гри. На додаток до звичайних умов завершення гри (наприклад, завершення гонки), зазвичай прискорюється навчання та додається логіка для раннього завершення навчання, якщо агенти потрапили або в поганий стан, коли неможливо отримати значуще навчання, або в хороший стан, коли додаткові винагороди недоступні. У цьому прикладі додається логіка для виявлення, коли автомобіль з'їхав з траси, і епізод при цьому завершується раніше.

Решту вихідного коду та деталі налаштування можна переглянути в "Learning to Drive (5.3)" [8].

Для відлагодження процесу написання RL моделі в Unreal Engine 5 присутній інструмент Visual Logger (рис. 7), де можна побачити інформацію про кожне окреме спостереження агентів у різні моменти часу.

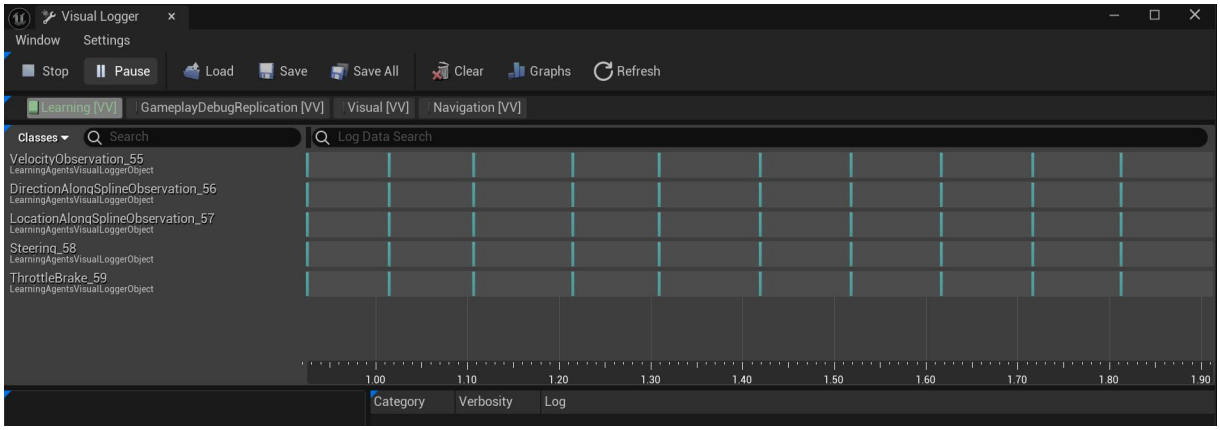

Рис. 7. Інструмент Visual Logger в Unreal Engine 5

За допомогою клавіші F8 можна увімкнути візуальне відображення спостережень на сплайні гоночної траси (рис. 8). Навчання агентів відбувається безпосередньо в редакторі після запуску ігрової симуляції. Оптимальна кількість, яка враховує ресурси

комп'ютера та швидкість навчання, це 32 агенти (автомобілі). Побачити помітне покращення у водінні можна вже через 15 хвилин, і приблизно через 2 години агенти будуть повністю навчені. Навчені ваги зберігаються в спеціальному файлі інформації нейронної мережі (Data Asset). Отримавши повністю навчену модель, її можна запустити в заключному режимі (inference mode), щоб використовувати навчену поведінку. Під час заключного режиму Python більше не використовується, і весь код нейронної мережі виконується виключно в процесі гри Unreal. Коли гра пакується, жоден навчальний код не включається в програму.

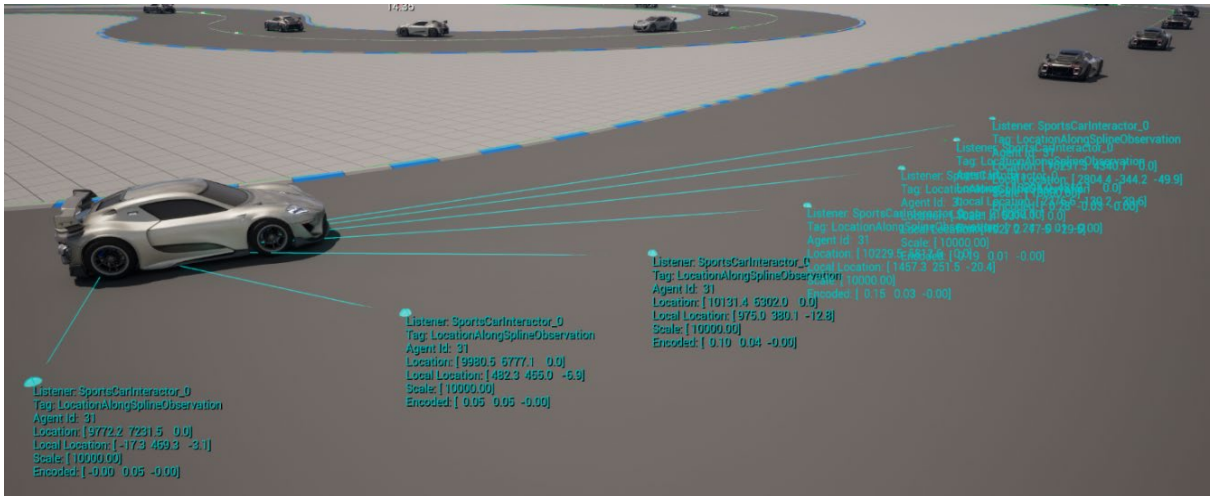

Рис. 8. Дебагінг спостережень за допомогою клавіші F8

Навчання з підкріпленням в Unity реалізується за допомогою набору інструментів "ML-Agents Toolkit" [9]. Вони підтримують навчання агентів за допомогою навчання з підкріпленням, імітаційного навчання, нейроеволюції та інших методів машинного навчання через той же інструмент, що і в UE – Python API та бібліотека PyTorch. Навчених агентів можна використовувати для багатьох цілей, зокрема для контролю поведінки NPC, автоматизованого тестування збірок гри та оцінки різних рішень щодо дизайну гри перед випуском. ML-Agents Toolkit є взаємовигідним як для розробників ігор, так і для дослідників штучного інтелекту, оскільки він забезпечує центральну платформу, на якій можна оцінити досягнення ШІ в багатих середовищах Unity, а потім зробити їх доступними для більш широких спільнот дослідників і розробників ігор.

Інсталяція ML-Agents Toolkit є важчою ніж аналогічного плагіна UE та включає такі кроки:

1. Інсталяція Python (рекомендовано 3.10.12);
2. Стягування репозиторію ML-Agents Toolkit з GitHub;
3. Встановлення пакету com.unity.ml-agents в Unity;
4. Встановлення mlagents-envs та mlagents для Python.

Для оцінки навчання з підкріпленням в Unity використовується проєкт з відкритим кодом MLAgentsAtSpa [10].

Тут гоночна траса згенерована складніше, оскільки містить підйоми та схили (рис. 4 – б). Окрім цього, використовується Check Point система, яка дозволяє контролювати прогрес агента на трасі, формувати винагороди, а також структурувати навчання. Це система тригерів (колайдерів), які розміщені послідовно вздовж треку. Коли автомобіль проїжджає через такий чекпоінт — скрипт фіксує її положення, напрямок руху та факт проходження цієї точки. Всього у проєкті створено 70 чекпоінтів, які розташовані по всій довжині траси. Агент отримує винагороду за проходження чекпоінтів, особливо

останнього. Якщо він не встигає пройти певну кількість чекпоінтів за час — отримає штраф. Вектор напрямку треку між двома чекпоінтами використовується для оцінки, чи рухається машина в правильному напрямку. Обчислюється скалярний добуток між вектором руху машини і напрямком між чекпоінтами. Навчання починається з 1 чекпоінта та після 20 успішних проходжень — додається ще один чекпоінт, і так до повного проходження 70, що робить навчання стабільним та поступовим (curriculum learning).

Для взаємодії між трасою, агентом (автомобілем) та системою навчання використовується клас CheckPointMonitor. Він відстежує прогрес гоночного агента на трасі, перевіряє проходження контрольних точок, оцінює вирівнювання машини з треком та передає винагороди або покарання до агента. Метод, який викликається при проходженні чекпоінта агентом називається PassedCheckPoint (лістинг 1).

```
public void PassedCheckPoint(int CheckPointPassed)
   {
      if(CheckPointPassed != NextExpectedCheckPoint)
      {
         Debug.Log("[ERROR]: Failed Check Point");
      }
      else
      {

         TheCarHUDDisplay.SetLastCP(CheckPointPassed);

         NextExpectedCheckPoint = NextExpectedCheckPoint + 1;

         TheRaceCarAgent.AddCheckPointReward(1.0f / MaxCheckPoint);

         if (NextExpectedCheckPoint > MaxCheckPoint)
         {
            CurrentLapTime = Time.time - LapStartTime;
            Debug.Log("[INFO] Completed a Lap: " + CurrentLapTime.ToString());
            TheCarHUDDisplay.SetLapDistance(CurrentLapTime);
            TheRaceCarAgent.LapCompleted(CurrentLapTime / MaxLapTime);

         }
          }
   }
```

Лістинг 1. Реалізація функції PassedCheckPoint

Якщо це правильний чекпоінт (той, який очікується), то агент отримує часткову винагороду (1.0 / MaxCheckPoint) та HUD оновлює останній пройдений чекпоінт. Якщо це був останній чекпоінт — агенту нараховується винагорода за коло і викликається метод LapCompleted(). При проходженні неправильного чекпоінту фіксується помилка ([ERROR]: Failed Check Point) та винагорода не нараховується.

Іншим важливим класом для цієї моделі є клас агента RaceCarAgent. Він успадковується від базового класу Agent, і саме тут реалізована логіка взаємодії з навчальним середовищем. Дії агента визначені за допомогою двох гілок: рух вперед / гальма / реверс та рух вліво / вправо. Метод CollectObservations передає спостереження

до нейромережі, а саме швидкість машини (CarSpeed), нахил траси (PitchIncline) — щоб враховувати схили та вирівнювання з трасою — передається з CheckPointMonitor (лістинг 2).

```
        public override void CollectObservations(VectorSensor sensor)
           {
              Vector3 LocalVelocity =
TheRaceCarRigidBody.transform.InverseTransformDirection(TheRaceCarRigidBody.velocity);
              CarSpeed = LocalVelocity.z * 2.0f;
              TheHUDDisplayController.SetSpeed(CarSpeed);

              var right = transform.right;
              right.y = 0;
              right *= Mathf.Sign(transform.up.y);
              var fwd = Vector3.Cross(right, Vector3.up).normalized;
              CarPitchIncline = Vector3.Angle(fwd, transform.forward) *
Mathf.Sign(transform.forward.y);
              TheHUDDisplayController.SetIncline(CarPitchIncline);

              sensor.AddObservation(CarSpeed/100.0f);
              sensor.AddObservation(CarPitchIncline/10.0f);
              sensor.AddObservation(TheRaceTrackMonitor.CarToRaceTrackAlignment);

           }
```

Лістинг 2. Реалізація функції CheckPointMonitor

Агент використовує 3D Ray Cast Sensor для розпізнавання чекпоінтів перед собою та формування спостереження. Його можна відобразити задля відлагодження моделі (рис. 9). Окрім цього, можна увімкнути Heuristic режим для перевірки реакції агента та ручне керування, вивести логи через Debug.Log() або переглянути зміну значення кумулятивної винагороди, щоб переконатися в її збільшенні.

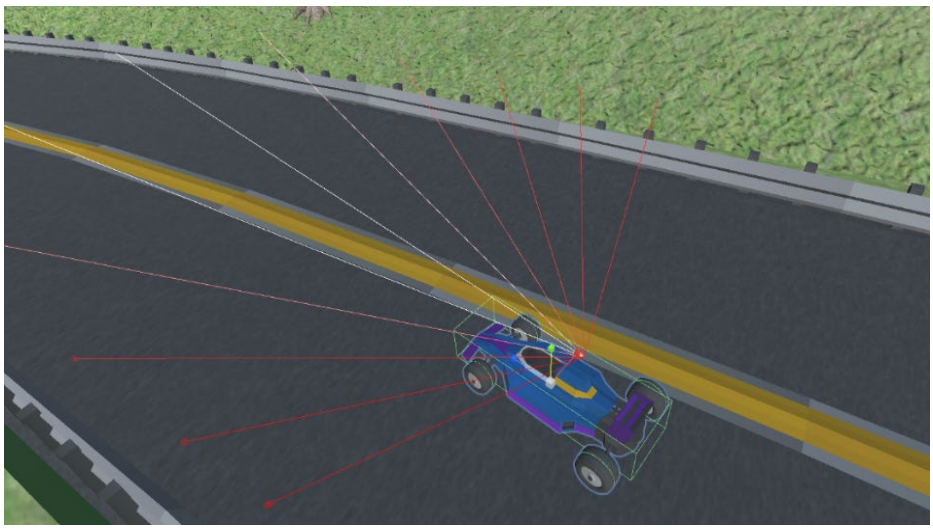

Рис. 9. Візуалізований 3D Ray Cast Sensor агента-автомобіля. Трасування потрібне для розпізнавання автомобілем свого місця на трасі – формуванні спостережень

Решту вихідного коду та деталі налаштування можна переглянути в "MLAgentsAtSpa" [9].

Щоб навчити агента проходити одне ціле коло, навчання відбувалося на 70 рівнях. Початкова ціль і позитивна винагорода — просто дістатися до контрольної точки 1 на першому рівні навчання. Рівень навчання підвищувався після 20 послідовних позитивних досягнень мети, а також було реалізовано зниження рівня навчання після 10 невдач (однак жодного зниження рівня навчання не спостерігалося). На кожному рівні навчання довжина траси перегонів збільшувалася на одну контрольну точку, тому для тренування агента по всій трасі було необхідно мати понад 70 рівнів підготовки, що відповідає повній довжині траси – 70 коллайдерів контрольних точок. Цю модель можна було назвати успішною після 20 мільйон епох навчання. Для відшліфування роботи модель тренувалася ще приблизно 5 мільйонів епох (рис. 10).

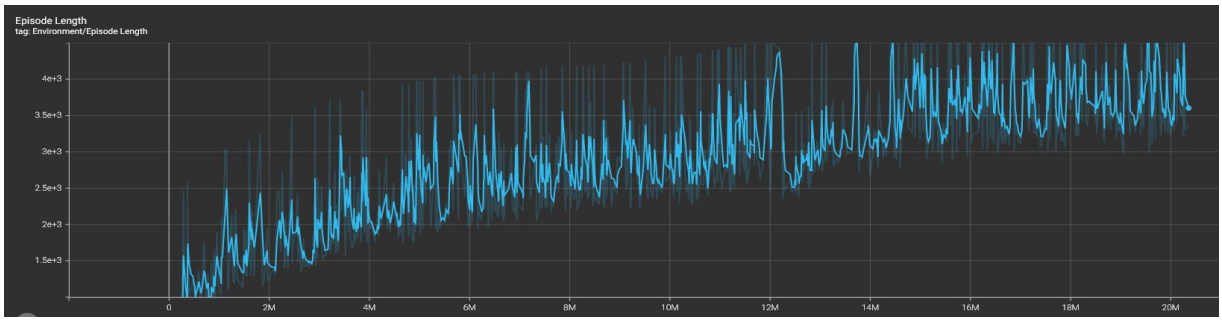

Рис. 10. Графік результатів навчання моделі в Unity

Виходячи з наданих даних, навчання моделі в проєкті тривало досить довго через особливості реалізації. Агент тренувався за допомогою алгоритму PPO з мінімальним налаштуванням гіперпараметрів: розмір пакету (batch size) складав 256, розмір буфера – 2048, а часовий горизонт – 256. Нейронна мережа складалася з двох шарів по 512 нейронів для основної моделі та мережі GAIL (імітаційного навчання). Особливістю навчання було те, що воно проводилось лише в одному середовищі Unity, без паралельного запуску декількох копій сцени, що значно сповільнювало загальний процес. Оскільки сцена була досить важкою — включала великі ландшафти, об'єкти оточення, HUD-елементи та фізичну модель автомобіля з WheelCollider-ами — кожна ітерація могла тривати орієнтовно 20 мілісекунд. За таких умов приблизний час навчання становив близько 500 000 секунд, тобто майже 139 годин або приблизно 5–6 днів безперервного тренування. Основними причинами тривалого навчання були відсутність паралельного запуску кількох агентів, поступове додавання складності лише по одному чекпоінту за раз та складна сцена. Попри це, агент успішно навчився проходити трасу, що свідчить про ефективність застосованого підходу, хоч і за рахунок значного часу тренування.

Висновки. У результаті дослідження було проаналізовано використання навчання з підкріпленням в ігрових рушіях Unreal Engine 5 та Unity. Обидва середовища чудово підходять для Reinforcement Learning, але вибір залежить від цілей проєкту, досвіду в рушіях та потреб у графіці/фізиці. Unity ML-Agents Toolkit — зрілий інструмент із гнучкою екосистемою, зручний для досліджень і навчання, але повільніше працює без паралельності. Unreal Engine 5 Learning Agents — новий потужний інструмент, оптимізований для симуляцій з реалістичною фізикою та графікою, але потребує глибшого занурення в рушій. Узагальнені результати роботи подані в табл.1.

Бібліографія

1. TD-Gammon / W. Uther et al. Encyclopedia of machine learning. Boston, MA, 2011. P. 955–956. URL: https://doi.org/10.1007/978-0-387-30164-8_813.
2. Gronauer S., Diepold K. Multi-agent deep reinforcement learning: a survey. Artificial intelligence review. 2021. URL: https://doi.org/10.1007/s10462-021-09996-w.
3. Egri-Nagy A., Törmänen A. The cost of passing - using deep learning AIs to expand our understanding of the ancient game of Go. International journal of networking and computing. 2023. Vol. 13, no. 2. P. 258–272. URL: https://doi.org/10.15803/ijnc.13.2_258.
4. Yadav A. Reinforcement Learning in Games: A Complete Guide. Medium. URL: https://medium.com/@amit25173/reinforcement-learning-in-games-a-complete-guide-24d1cab79317.
5. Risi S., Preuss M. Behind DeepMind's AlphaStar AI that Reached Grandmaster Level in StarCraft II. KI - Künstliche Intelligenz. 2020. Vol. 34, no. 1. P. 85–86. URL: https://doi.org/10.1007/s13218-020-00642-1.
6. Welcome to AirSim. AirSim. URL: https://microsoft.github.io/AirSim/.
7. Mulcahy B. Learning agents (5.5). Unreal Engine Dev Community. URL: https://dev.epicgames.com/community/learning/courses/GAR/unreal-engine-learning-agents-5-5/7dmy/unreal-engine-learning-to-drive-5-5.
8. Mulcahy B. Learning Agents Introduction (5.3). Unreal Engine Dev Community. URL: https://dev.epicgames.com/community/learning/tutorials/8OWY/unreal-engine-learning-agents-introduction-5-3.
9. ML-Agents Overview. Unity ML-Agents Toolkit. URL: https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/.
10. JulesVerny/MLAgentsAtSpa: Unity ML Agent Racing Car learns to Drive at Spa !. GitHub. URL: https://github.com/JulesVerny/MLAgentsAtSpa.

References

1. TD-Gammon / W. Uther et al. Encyclopedia of machine learning. Boston, MA, 2011. P. 955–956. URL: https://doi.org/10.1007/978-0-387-30164-8_813.
2. Gronauer S., Diepold K. Multi-agent deep reinforcement learning: a survey. Artificial intelligence review. 2021. URL: https://doi.org/10.1007/s10462-021-09996-w.
3. Egri-Nagy A., Törmänen A. The cost of passing - using deep learning AIs to expand our understanding of the ancient game of Go. International journal of networking and computing. 2023. Vol. 13, no. 2. P. 258–272. URL: https://doi.org/10.15803/ijnc.13.2_258.
4. Yadav A. Reinforcement Learning in Games: A Complete Guide. Medium. URL: https://medium.com/@amit25173/reinforcement-learning-in-games-a-complete-guide-24d1cab79317.
5. Risi S., Preuss M. Behind DeepMind's AlphaStar AI that Reached Grandmaster Level in StarCraft II. KI - Künstliche Intelligenz. 2020. Vol. 34, no. 1. P. 85–86. URL: https://doi.org/10.1007/s13218-020-00642-1.
6. Welcome to AirSim. AirSim. URL: https://microsoft.github.io/AirSim/.
7. Mulcahy B. Learning agents (5.5). Unreal Engine Dev Community. URL: https://dev.epicgames.com/community/learning/courses/GAR/unreal-engine-learning-agents-5-5/7dmy/unreal-engine-learning-to-drive-5-5.
8. Mulcahy B. Learning Agents Introduction (5.3). Unreal Engine Dev Community. URL: https://dev.epicgames.com/community/learning/tutorials/8OWY/unreal-engine-learning-agents-introduction-5-3.
9. ML-Agents Overview. Unity ML-Agents Toolkit. URL: https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/.
10. JulesVerny/MLAgentsAtSpa: Unity ML Agent Racing Car learns to Drive at Spa !. GitHub. URL: https://github.com/JulesVerny/MLAgentsAtSpa.