DOI 10.36074/logos-19.12.2025.024

БЕК-ЕНД АРХІТЕКТУРА СИСТЕМИ АДАПТИВНОЇ ПОВЕДІНКИ NPC У МУЛЬТИАГЕНТНОМУ СЕРЕДОВИЩІ НА ОСНОВІ Q-LEARNING

Юхатов Артур Вікторович[1]

1. Senior Software Enginee
УКРАЇНА
ORCID ID: 0009-0003-6268-3038

Анотація. *У статті представлено бек-енд архітектуру системи адаптивної поведінки неігрових персонажів (NPC) у мультиагентному середовищі, побудовану на основі алгоритму Q-learning. Розглянуто модель марківського процесу прийняття рішень (MDP), механізм формування та збереження Q-таблиці, систему винагород, а також архітектурні рішення серверної частини, що забезпечують навчання агентів у прискореній симуляції. Запропонований підхід інтегровано у рушій Unreal Engine 5, що дозволяє організувати кероване та відтворюване середовище для множини NPC. Проведено аналіз експериментальних результатів, що підтверджують стабільність навчання та можливість масштабування системи до більших мультиагентних конфігурацій.*

Вступ. Формування правдоподібної, самостійної та адаптивної поведінки віртуальних агентів (NPC) є фундаментальним завданням у розробці передових інтерактивних систем та симуляційних середовищ. Однак, класичні детерміновані підходи, такі як архітектури скінченних автоматів (FSM) або поведінкові дерева (BT), незважаючи на забезпечення певного рівня прогнозованості, суттєво обмежують спроможність віртуальних сутностей гнучко адаптуватися до динамічних та непередбачуваних змін у навколишньому середовищі. В умовах мультиагентних систем, що характеризуються одночасною взаємодією численних суб'єктів, ці недоліки набувають особливої виразності та критичного значення. [1]. У мультиагентних системах, де багато персонажів взаємодіють одночасно, ці обмеження стають критичними.

Навчання з підкріпленням (Reinforcement Learning, RL) дозволяє агентам самостійно формувати політику поведінки через взаємодію з середовищем,

отримуючи винагороди за успішні дії. Q-learning, один із базових RL-алгоритмів, вирізняється простотою, гарантованою збіжністю та можливістю зберігати політику у табличному вигляді [2].

Мета статті полягає у розробці, описі та експериментальній верифікації бек-енд архітектури системи адаптивної поведінки неігрових персонажів (NPC), що базується на табличному алгоритмі Q-learning для управління багатовимірною системою потреб у симуляційному середовищі, реалізованому в Unreal Engine 5.

Виклад основного матеріалу. Найпростішими підходами до поведінки NPC є скриптові сценарії та автомати скінченних станів (FSM). FSM визначає обмежений набір станів та переходів між ними (рис. 1). Такі моделі мають низьку обчислювальну складність, але масштабуються погано – кількість станів росте експоненційно із зростанням складності поведінки [3].

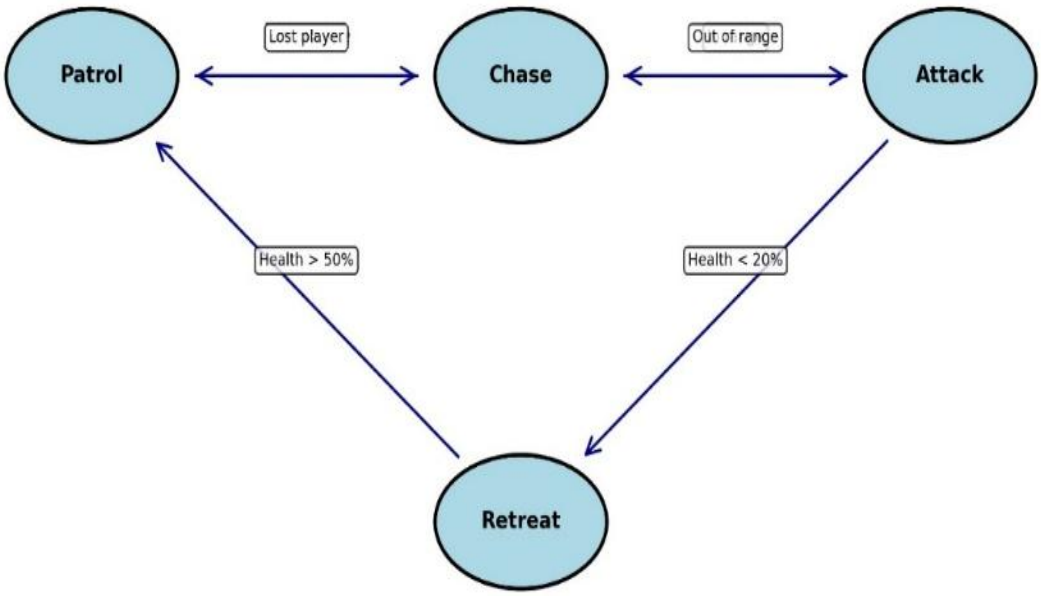

Рис 1. **Приклад FSM**

Незважаючи на притаманну їм лаконічність та структурну зрозумілість, такі моделі виявляються недостатньо адаптивними за умов ескалації складності поведінкових парадигм. Це об'єктивно детермінувало формування більш архітектурно організованих підходів, зокрема концепції поведінкових дерев. З метою нівелювання фундаментальних обмежень, властивих зокрема моделям скінченних автоматів, галузь імплементувала парадигму поведінкових дерев, що забезпечує значно вищий рівень модульності та масштабованості для репрезентації логіки функціонування.

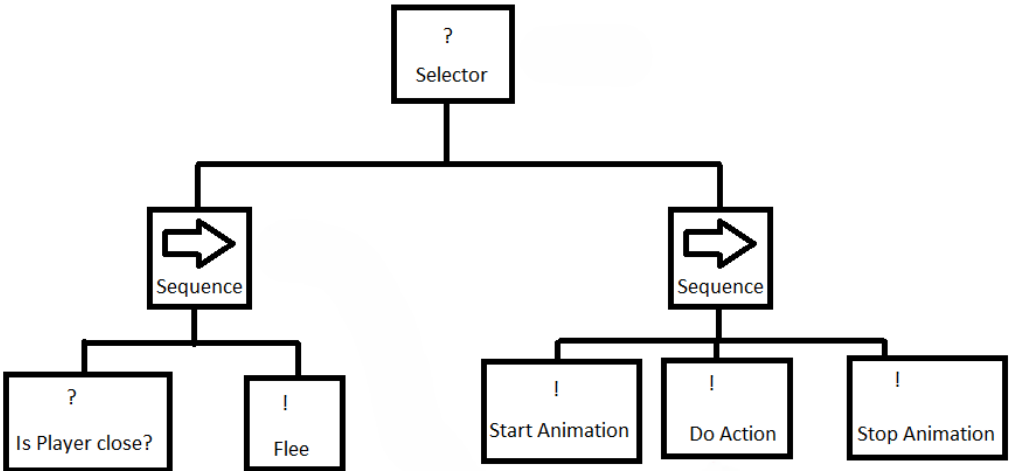

Рис. 2. **ВТ з подвійною дією**

Поведінкові дерева (Behavior Trees, BT) стали широко застосовуватися у великих проєктах завдяки модульності та можливості реактивного переобчислення логіки (рис. 2). Однак BT залишається детермінованим підходом, не здатним до самонавчання [4].

Для подолання детермінованості BT розробники почали застосовувати підходи, що дозволяють агентам самостійно формувати плани досягнення цілей, що і стало основою GOAP. Підхід Goal-Oriented Action Planning (GOAP) дозволяє будувати плани на основі визначених цілей і доступних дій (рис. 3).

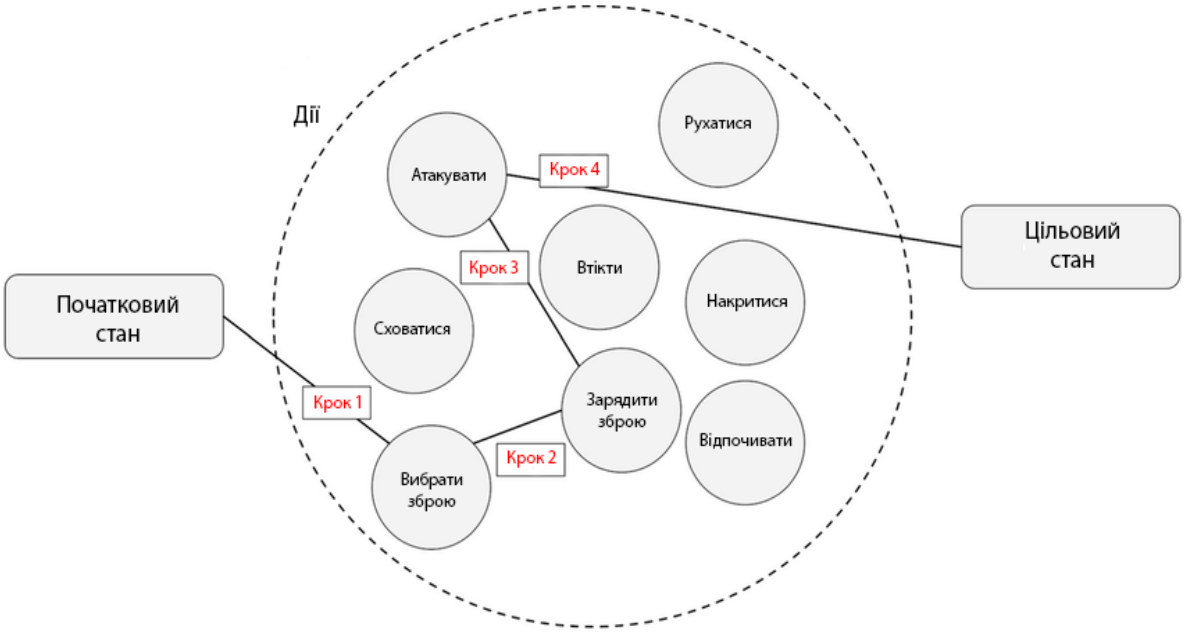

Рис 3. **Схематичне представлення роботи GOAP-планувальника**

Проте GOAP залежить від вручну створених правил і не здатний адаптуватися без участі розробника [5]. Хоча BT забезпечують архітектурну гнучкість, вони є декларативними, жорстко запрограмованими структурами, нездатними до автоматичної адаптації. Спроба подолати це обмеження призвела до появи GOAP, який вводить цілеспрямоване планування. Однак, ключовим недоліком GOAP залишається залежність від ручного налаштування правил та умов середовища, що обмежує здатність системи самостійно виявляти оптимальні стратегії. Ця неможливість самоадаптації у rule-based підходах (як GOAP) стимулювала перехід до навчання з підкріпленням (RL). На відміну від попередніх методів, RL дозволяє агентам формувати ефективну поведінку емпірично (через взаємодію із середовищем та максимізацію винагород). Для бек-енд системи це відкриває можливості для централізованого зберігання знань, тренування поза реальним геймплеєм та незалежного масштабування політик.Одним із найпоширеніших алгоритмів такого типу є Q-learning, який може бути ефективно реалізований у бек-енд системі.

Запропонована архітектура бек-енд підсистеми Q-learning реалізовується на основі моделі марківського процесу прийняття рішень (MDP). Отже, для формалізації процесу навчання використовується MDP, що дозволяє описати динаміку станів NPC у вигляді чітких параметрів: Hunger, Bladder, Energy, Social, Hygiene, Fun. Кожна з шести потреб була дискретизована на три рівні (Low, Medium, High), що в сукупності визначає простір із $3^6 = 729$ можливих станів. Узагальнена схема роботи алгоритму Q-learning в даному середовищі представлена на рис. 4.

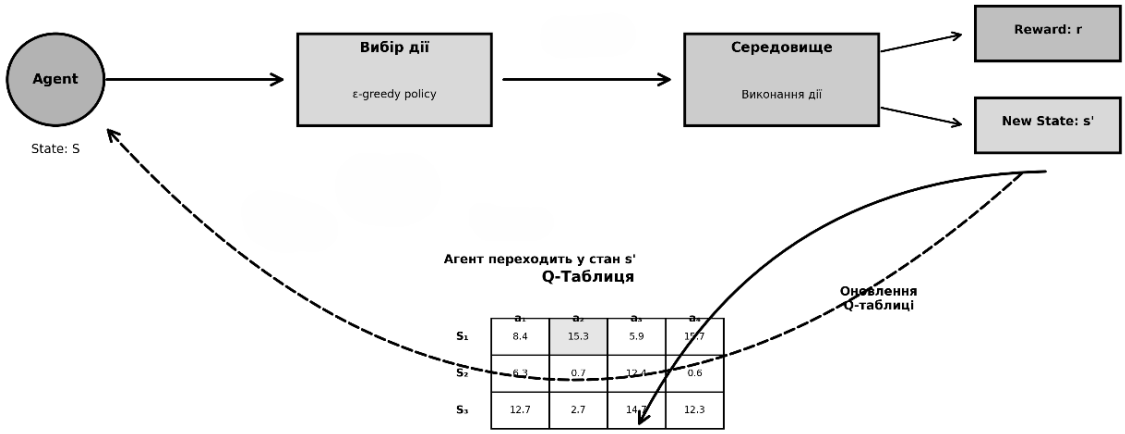

Рис. 4. **Процес Q-learning**

Представлена модель працює на основі алгоритму Q-learning, де агент поступово оновлює Q-таблицю, прагнучи досягти оптимальної стратегії

вибору дій у кожному дискретному стані. Дії агента відповідають високорівневим взаємодіям з об'єктами середовища (холодильник, душ, ліжко та ін.). Перехід між станами є детермінованим, оскільки залежить від двох факторів: модифікаторів, пов'язаних із характеристиками об'єктів взаємодії, та природної деградації потреб. Для забезпечення персистенції досвіду та ефективного керування політикою, Q-таблиця серіалізується у форматі JSON, що дозволяє швидко зберігати/завантажувати навчену політику та передавати знання між послідовними поколіннями NPC. Кожен запис у таблиці – це оцінка пари «стан–дія». Дані накопичуються протягом симуляцій та оновлюються за класичною формулою Q-learning [6]. Для повного розуміння роботи системи розглянемо цикл обробки даних у бек-енд ядрі.

Ядро бек-енду виконує повторювану схему, що включає оцінку стану NPC, вибір дії за ε-greedy, застосування дії, обчислення винагороди, оновлення Q-значення та збереження таблиці.

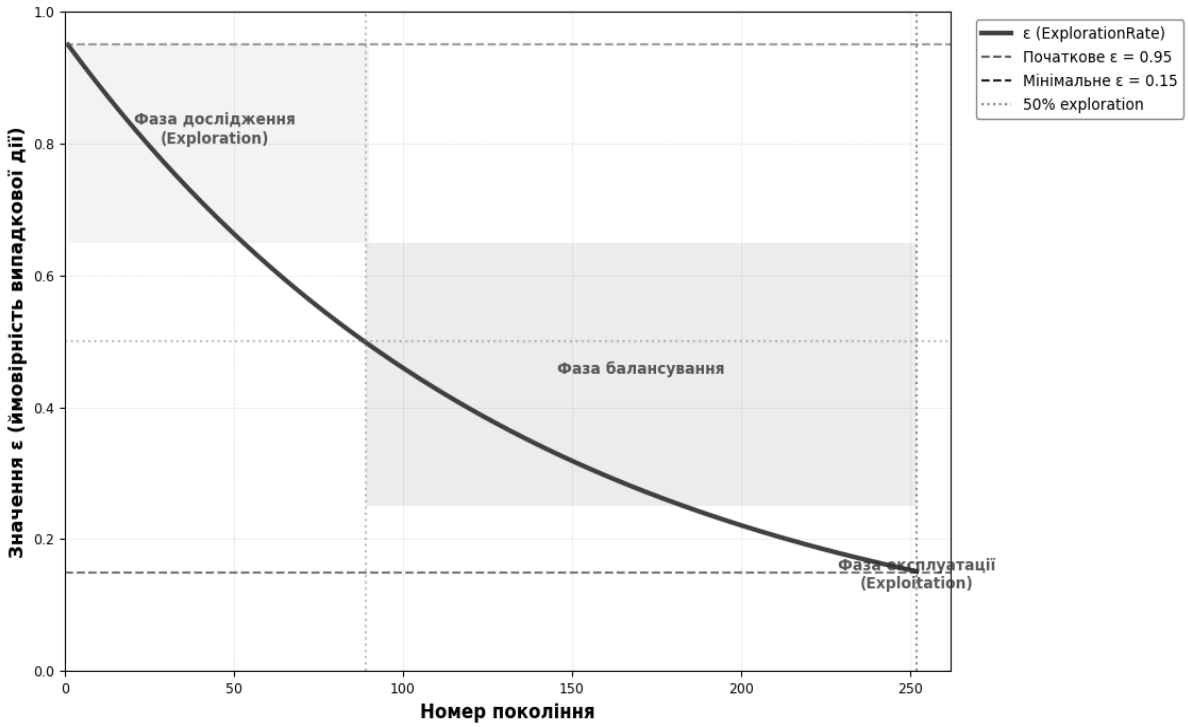

Рис. 5. **Графік спадання коефіцієнту дослідження ε за поколіннями**

Поведінка агента значною мірою залежить від параметрів навчання. На рис. 5 продемонстровано, як коефіцієнт ε поступово зменшується від високих значень (дослідження) до нижчих (експлуатація політики). Таке зниження сприяє більш стабільним стратегіям на пізніх етапах.

Параметри моделі у цьому дослідженні було встановлено такими: learning rate $\alpha = 0.3$, discount factor $\gamma = 0.95$, $\varepsilon_{\text{початкове}} = 0.95$, спадання до 0.15. Таке поєднання параметрів забезпечує баланс між швидкістю навчання та стабільністю політики.

Щоб реалізувати навчання NPC у реальному ігровому середовищі, необхідно забезпечити належну організацію бек-енд архітектури.

У бек-енд частині реалізовано таймери прийняття рішень, менеджер поколінь NPC, систему деградації потреб та модулі навігації.

Архітектура побудована на компонентній структурі C++ (рис. 6), що дозволяє підтримувати високу модульність і масштабованість системи.

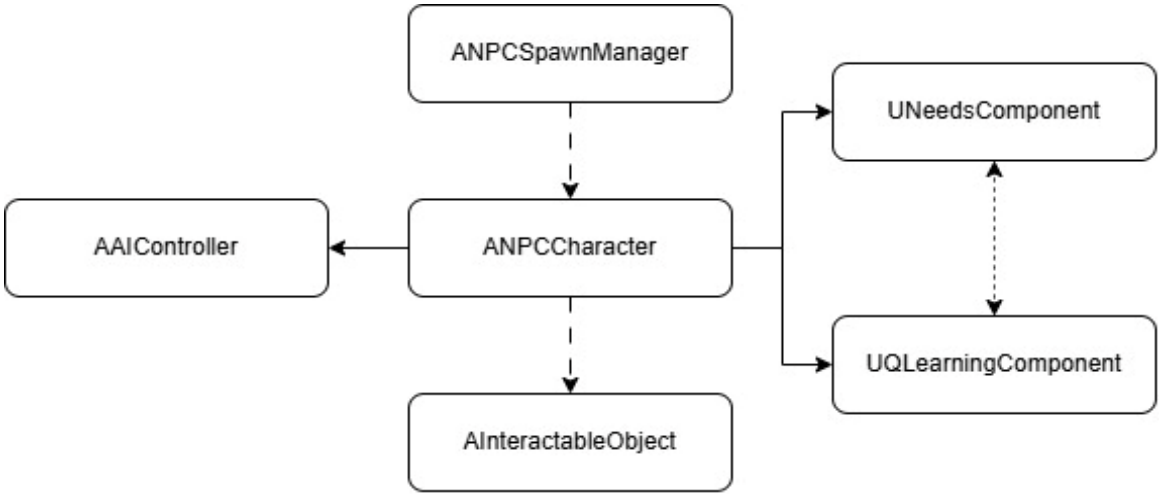

Рис. 6. **Взаємодія класів**

Для забезпечення довготривалого накопичення досвіду використано механізм «policy persistence».

Це означає, що після завершення життєвого циклу персонажа всі напрацьовані ним Q-значення зберігаються, а наступне покоління NPC успадковує політику.

Таким чином, система здатна навчатися протягом десятків тисяч поколінь, накопичуючи дедалі ефективнішу поведінку.

Для моделювання поведінки NPC було використано Unreal Engine 5.

Це середовище забезпечує високопродуктивне C++ ядро, інтегровані засоби навігації, підтримку JSON-серіалізації та зручні інструменти налагодження, що робить його придатним для складних мультиагентних симуляцій [7].

У запропонованій моделі потреб NPC знижуються на 0.4 одиниці щосекунди, що створює постійний дефіцит ресурсів. Такий тиск змушує агента

приймати рішення, спрямовані на підтримання життєздатності, і формує природні умови для навчання.

У середовищі наявні 21 інтерактивних об'єктів різних типів (рис. 7), кожен із яких має власну тривалість та вплив на потреби NPC.

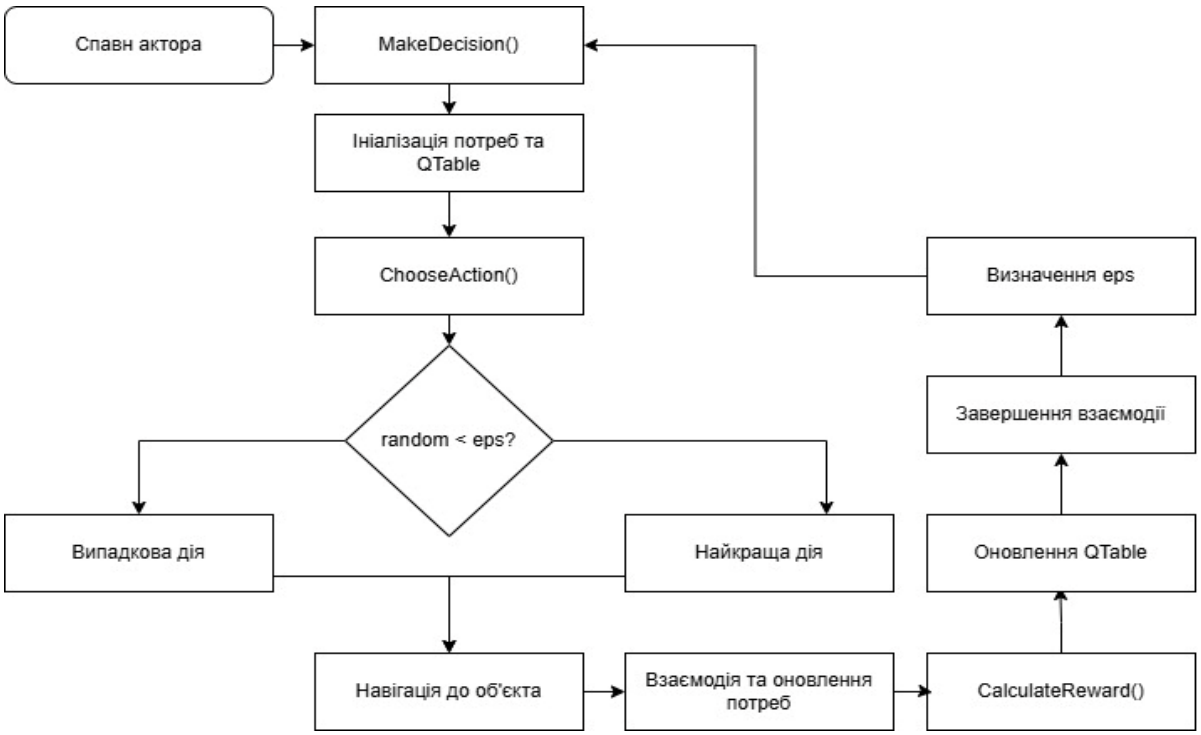

Рис. 7. **Цикл навчання NPC**

Для верифікації моделі було здійснено експеримент, що проводився протягом 252 поколінь, а це відповідає приблизно восьми годинам реального часу з урахуванням прискорення симуляції (коефіцієнт 15 ×). Загальний обсяг даних склав понад 38 тисяч записів про окремі дії та зміни станів. Аналіз тривалості життя персонажів виявив чітку нелінійну динаміку навчального процесу, що підтверджує чутливість табличного Q-learning до параметрів ϵ – $greedy$ політики. Динаміку чітко розділено на три фази (табл. 1):

Таблиця 1

Фази навчання NPC за методом Q-learning та їх характеристика

Фаза	Покоління	Коефіцієнт ϵ	Середня тривалість життя	Ключові характеристики
Рання (Exploration)	1–50	0.95 до 0.67	2249 с (37.5 хв)	Домінування випадкового вибору (висока ϵ). Висока варіативність результатів (StdDev 2179 с).

Продовження табл. 1

Фаза	Покоління	Коефіцієнт є	Середня тривалість життя	Ключові характеристики
Пікова (Balancing)	60–90	0.67 до 0.33	**4600 с (76.7 хв)**	Найвища ефективність, оптимальний баланс дослідження/експлуатації. Покращення на **104.5%** порівняно з ранньою фазою.
Пізня (Exploitation)	151–252	0.16 (мінімум)	1700 с (28.3 хв)	Деградація продуктивності через **перенавчання (overfitting)** та втрату адаптивності.

Графічне представлення зміни тривалості життя та середнього рівня потреб, що ілюструє цю динаміку, наведено на рис. 8 та рис. 9 відповідно:

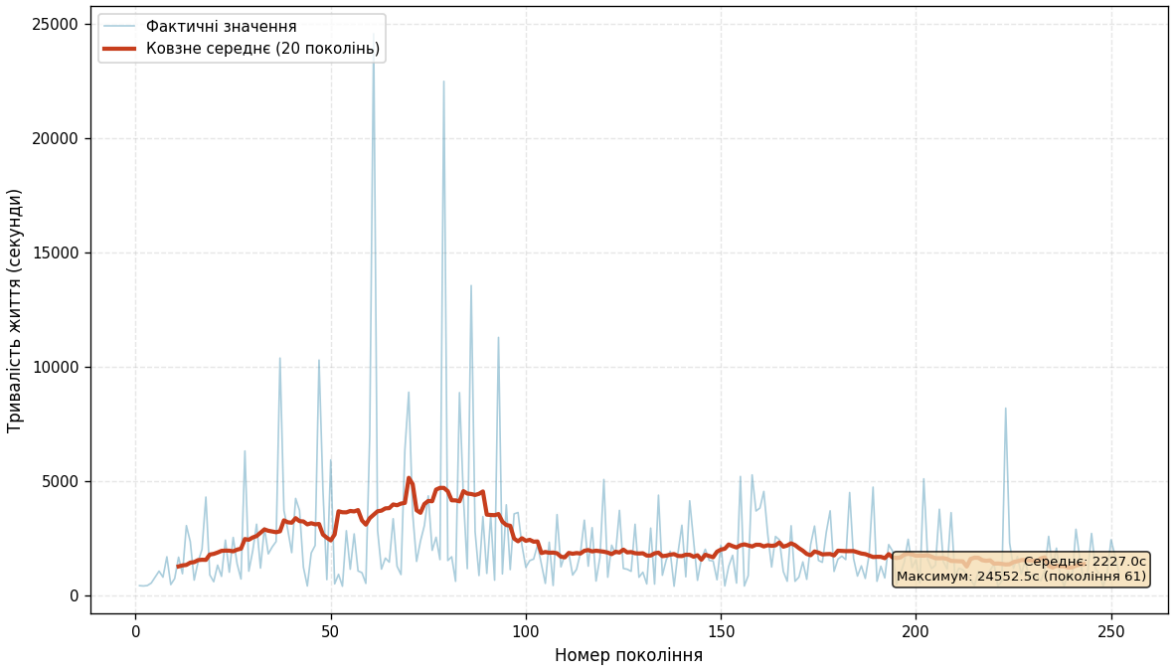

Рис. 8 **Тривалість життя NPC**

Отримані коливання тривалості життя відображають не лише ефективність дій агента, але й якість управління його внутрішніми станами. Щоб краще зрозуміти причини цих змін, проаналізуємо поведінку потреб NPC, представлену на рис. 9. Як видно з графіку, стабілізація тривалості життя супроводжується вирівнюванням середніх рівнів потреб. Це свідчить про те,

що агент починає розпізнавати пріоритети й формувати більш збалансовані стратегії виживання.

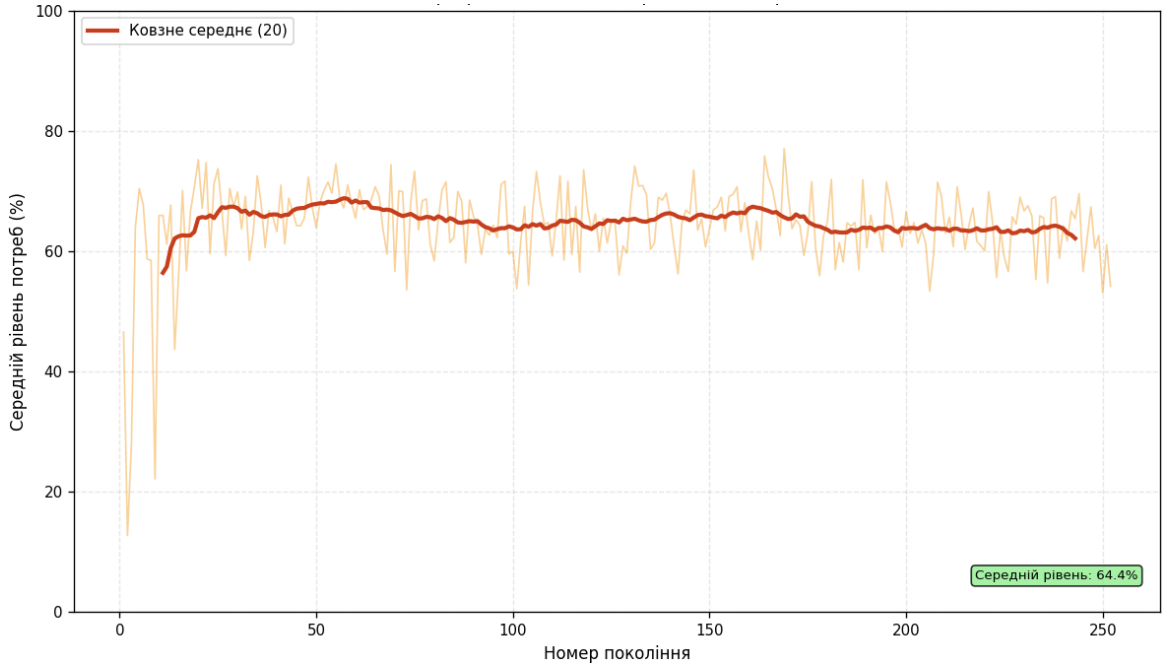

Рис. 9 **Якість підтримки потреб**

Щоб оцінити, як ці стратегії формалізуються у термінах алгоритму, розглянемо підсумковий розподіл Q-значень. Він демонструє, які дії агент вважає найбільш перспективними у різних станах. Для деталізації цієї картини наведено фрагмент Q-таблиці (табл. 2).

Таблиця 2

Фрагмент Q-таблиці після навчання NPC

Стан NPC (дискретизовані потреби)	Дія	Q-значення
Hunger = Low, Energy = Medium, Fun = Low	GoToTV	**912.4**
Hunger = Low, Energy = Medium, Fun = Low	GoToFridge	845.2
Hunger = Low, Energy = Medium, Fun = Low	GoToBed	430.1
Hunger = High, Energy = Low, Hygiene = Medium	GoToBed	**901.8**
Hunger = High, Energy = Low, Hygiene = Medium	GoToShower	544.6
Hunger = High, Energy = Medium, Fun = High	GoToFridge	**887.9**
Hunger = Medium, Energy = Medium, Fun = Low	GoToTV	**903.3**

взято з [авторська розробка]

Аналіз її структури та значень підтверджує ефективність алгоритму. З 729 теоретично можливих станів (дискретизація 6 потреб на 3 рівні), система відвідала 487 унікальних станів (66.8% покриття), сформувавши 2847 пар "стан-дія". Це достатній рівень покриття, оскільки не відвідані стани є екстремальними комбінаціями, які рідко виникають при оптимальній політиці.

84.5% Q-значень є позитивними, що вказує на загальну успішність навчання. Середнє Q-значення склало 456.3 при медіані 523.7. Максимальне значення досягло 1024.7 для дії відновлення критично низької потреби Energy. Ці дані відображають високу цінність негайного задоволення життєво важливих потреб. Розподіл Q-значень показано на рис. 9.

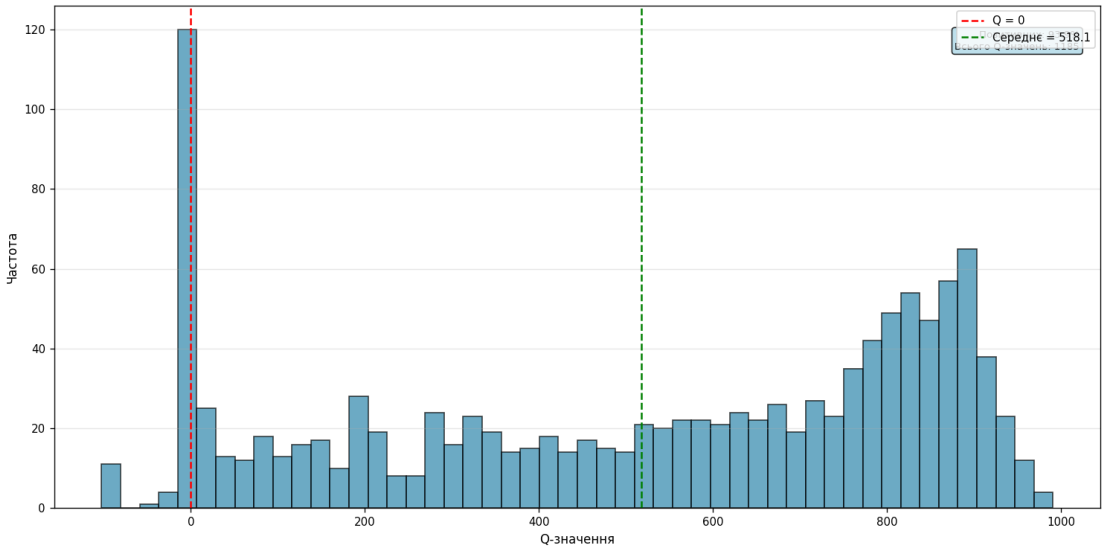

Рис. 9 **Розподіл Q-значень**

Фрагмент Q-таблиці (табл. 2) узагальнює найтиповіші рішення NPC після завершення навчання та ілюструє, як Q-learning формує ієрархію дій залежно від потреб агента та демонструє, що для заданого стану Hunger = Low, Energy = Medium, Fun = Low найкращою дією є GoToTV (Q-значення 912.4) – дія, спрямована на задоволення найнижчої потреби (Fun). Це підтверджує формування превентивної стратегії, спрямованої на підтримку балансу потреб, а не лише реакцію на критичні виснаження.

Найчастіше відвідуваним станом є "222222" (усі потреби на високому рівні), що становить 17.0% від усіх дій. Це є цільовим станом (goal state), досягнення якого забезпечує максимальну стійкість. Щоб оцінити, наскільки стабільною є поведінка NPC у часі, проаналізуємо частоту відвідуваних станів. Це дозволяє зрозуміти, чи дійсно агент прямує до цільових конфігурацій та уникає критичних ситуацій (рис. 10).

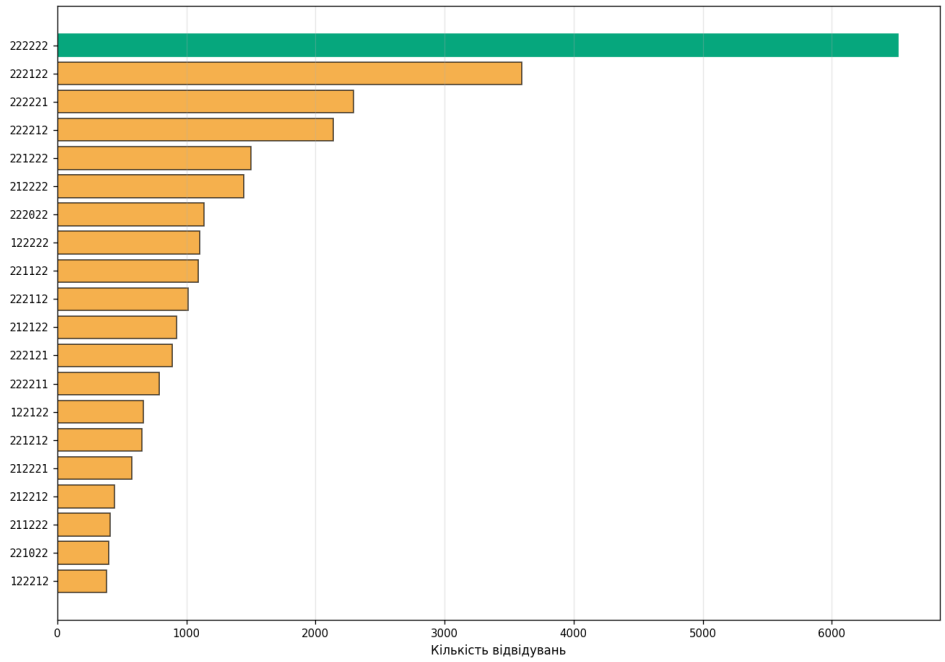

Рис. 10 **20 найбільш частих значень**

Найчастішою дією є UseBed (6842 рази) через високу швидкість деградації Energy. Водночас важливо оцінити, як часто NPC обирає ті чи інші дії. Рівномірність або перекоси в цьому розподілі відображають сформовану стратегію та ступінь її оптимальності (рис. 11).

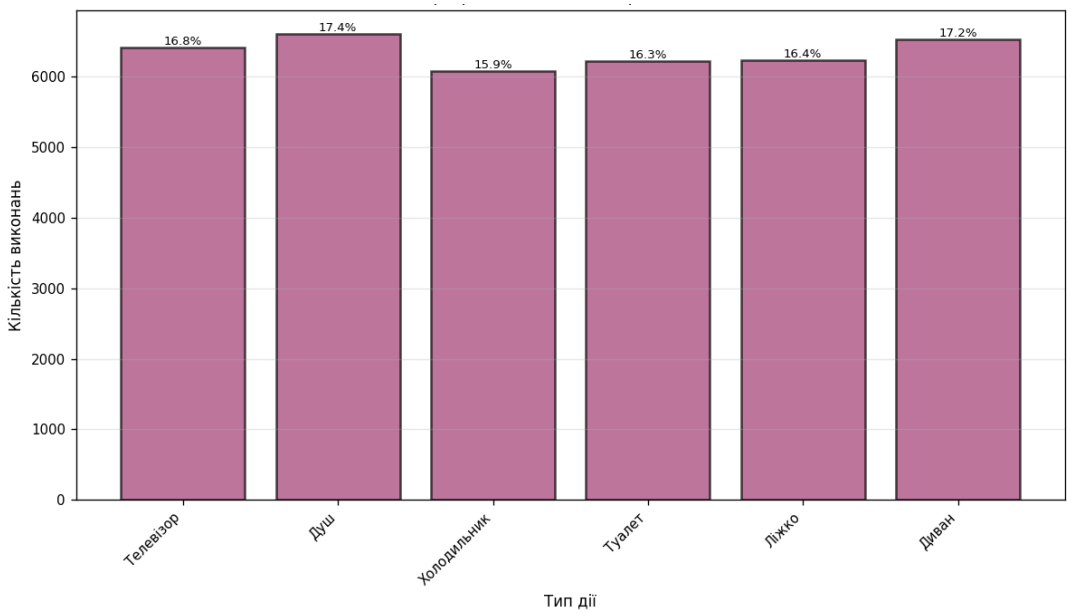

Рис. 11 **Розподіл обраних дій**

Наявність пікової продуктивності у середині тренування та подальший спад прямо пов'язані зі змінами політики дослідження. Розглянемо детальніше, як параметри ϵ вплинули на стабільність навчання. Спостережувана деградація продуктивності на пізній фазі (після покоління 150) є класичним прикладом перенавчання (overfitting) для табличного Q-Learning в умовах високорозмірного, дискретизованого простору станів.

Зниження ϵ до мінімального значення (0.15) різко скоротило дослідження (exploration), змушуючи агента сліпо експлуатувати субоптимальну політику, яка була сформована на середніх етапах. Медіанна тривалість життя на пізньому етапі впала на 51.6% порівняно з піком. NPC почали застрягати у локальних оптимумах, втрачаючи здатність адаптуватися до нових або рідкісних станів. Незважаючи на перенавчання, досягнення піку продуктивності на середньому етапі ($\epsilon \approx 0.50$) підтверджує, що архітектура системи та параметри винагород є коректними, а алгоритм успішно формує складні адаптивні стратегії, здатні балансувати шість конкуруючих потреб.

Для підтвердження коректності вибраних гіперпараметрів $(\alpha, \gamma, \epsilon)$ було проведено серію додаткових експериментів з альтернативними конфігураціями. Результати наведено на рис. 12–14.

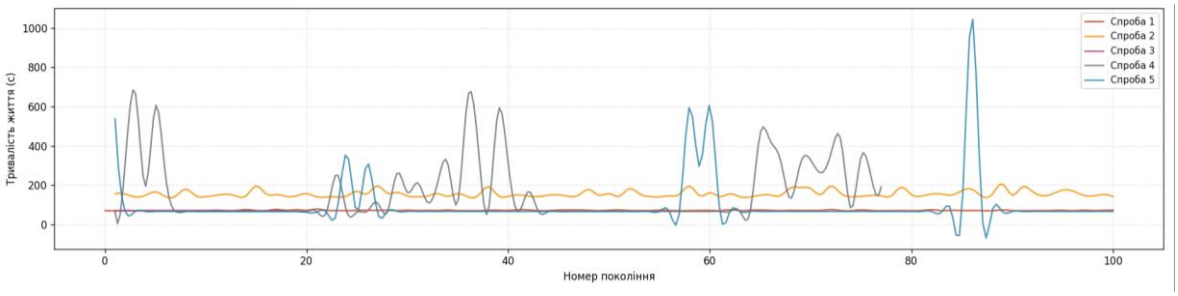

Рис. 12 **Порівняння тривалості життя у невдалих спробах**

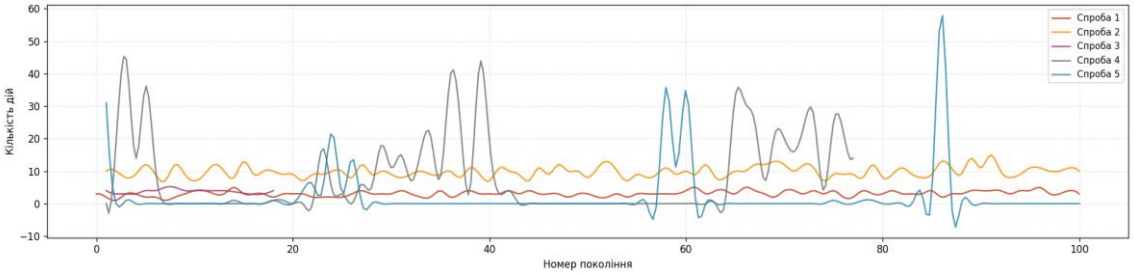

Рис. 13 **Порівняння кількості дій у невдалих спробах**

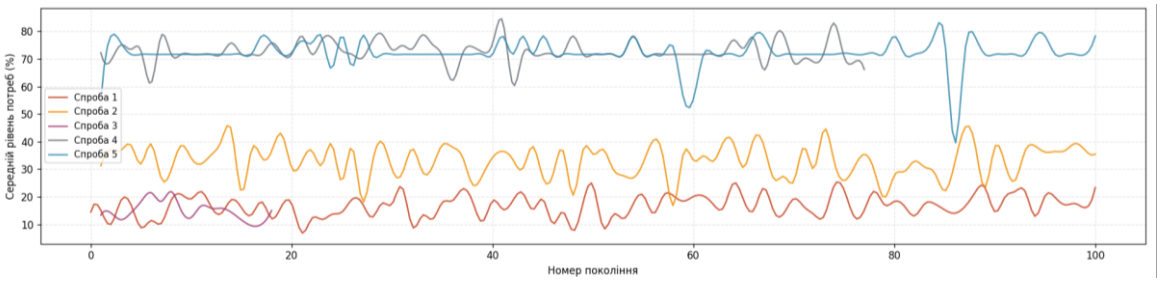

Рис. 14 **Порівняння середнього рівня потреб у невдалих спробах**

Спроби 1, 3, 4 – не продемонстрували збіжності, тривалість життя становила 70-90 с, що вказує на повну відсутність навчання та неефективність політики. Спроба 5 – показала екстремальні коливання тривалості життя, що свідчить про нестабільність навчального процесу.

Успішна конфігурація досягла середньої кількості дій 324 на піку, що у 30-50 разів перевищує результати невдалих спроб, доводячи, що обрані α =0.3, γ =0.95 та схема спадання ϵ були оптимально підібрані.

Висновки. У статті успішно розроблено та реалізовано бек-енд архітектуру адаптивної системи поведінки неігрових персонажів (NPC), що базується на алгоритмі Q-learning та інтегрована в ігровий рушій Unreal Engine 5 з використанням C++. Була створена модель Марківського процесу прийняття рішень (MDP), що включає дискретизацію шести конкуруючих потреб NPC та механізм формування Q-таблиці з відповідною системою винагород. Запропонований підхід забезпечив високу продуктивність, необхідну для прискорення симуляції у 15 × разів, що дозволило провести комплексне експериментальне дослідження та накопичити понад 38 тисяч записів про стан та дії. Аналіз навченої політики підтвердив здатність системи формувати складну превентивну стратегію балансування потреб, досягнувши на піковій фазі навчання (покоління 60–90) середньої тривалості життя персонажів у 4600 секунд, що значно перевищує базові показники виживання.

Експериментальні результати демонструють чітку збіжність алгоритму, але також виявляють його фундаментальні обмеження, зокрема нелінійну динаміку продуктивності з наступною деградацією через перенавчання (overfitting) на пізніх етапах. Цей феномен пояснюється втратою адаптивності внаслідок мінімізації коефіцієнта дослідження ϵ у дискретизованому просторі станів. Незважаючи на це, досягнуті показники ефективності та результати порівняльного аналізу підтверджують коректність архітектурних рішень та обраних параметрів навчання. Подальший розвиток системи передбачає перехід до методів глибокого Q-навчання (DQL) для усунення обмежень, пов'язаних із дискретизацією, та підвищення загальної стійкості політики.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

[1] Millington, I., & Funge, J. (2016). Artificial Intelligence for Games (3rd ed.). CRC Press.
[2] Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.
[3] Buckland, M. (2005). Programming Game AI by Example. Jones & Bartlett.
[4] Champandard, A. (2007). Behavior Trees for Next-Gen Game AI. AiGameDev.com.
[5] Orkin, J. (2006). Three States and a Plan: The AI of F.E.A.R.
[6] Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3), 279–292.
[7] Epic Games. (2023). Unreal Engine 5 Documentation.