Issue 13, 2023

https://doi.org/10.23939/sisn2023.13.079

УДК 004.85

МЕТОДИ МАШИННОГО НАВЧАННЯ ДЛЯ КЕРУВАННЯ ПОВЕДІНКОЮ НЕІГРОВИХ ПЕРСОНАЖІВ У БАГАТОКОРИСТУВАЦЬКІЙ РОЛЬОВІЙ ВІДЕОГРІ

Роман Будник, Віталій Яковина

Національний університет "Львівська політехніка",
кафедра систем штучного інтелекту, Львів, Україна
E-mail: roman.budnyk.mknssh.2021@lpnu.ua, ORCID: 0009-0002-2576-531X
E-mail: vitaliy.s.yakovyna@lpnu.ua, ORCID: 0000-0003-0133-8591

Розглянуто розроблення системи, що керуватиме неігровими персонажами для багатокористувацької рольової гри (англ. Role-Playing Game, RPG). У галузі відеоігор, зокрема жанрі RPG, комерційні проєкти рідко застосовують моделі машинного навчання для реалізації поведінки персонажів. Зазвичай використовують примітивні наперед запрограмовані правила або реалізується примітивний скінченний автомат. Такі підходи не дають гравцям відчути, що вони грають із розумними істотами, оскільки різні наперед задані правила стають передбачуваними. Хороший ігровий штучний інтелект повинен створити у гравця враження, ніби він взаємодіє зі справжніми персонажами, які приймають різноманітні, часом непередбачувані, рішення. У статті розглянуто застосування різних моделей машинного навчання для досягнення цієї мети у поєднанні із традиційно використовуваними скінченними автоматами. Поставлене завдання реалізується на прикладі попередньо розробленої відеогри. Здійснено огляд досліджень у галузі ігрового штучного інтелекту, відтак описано систему, яку розробили автори, та її реалізації, що застосовують декілька відібраних і успішно навчених моделей машинного навчання. Для навчання випробувано кілька моделей. Зрештою відібрано модель дерев рішень та нейронної мережі, оскільки вони дали найкращі результати. Описано також навчання цих моделей та наведено огляд результатів. Для цього здійснено випробувальні бої між різними моделями. Результати моделі дерева рішень були трохи кращими, ніж традиційного скінченного автомата, тоді як результати нейромережі набагато кращі. Запропоновані рішення можна розвивати далі, залучаючи складніші моделі та поліпшуючи навчання, щоб створити "розумніших" персонажів, участь яких у грі якісно покращує ігровий процес. Отриману систему інтегровано у відеогру, яка потенційно може стати комерційним продуктом.

Ключові слова: машинне навчання; відеогра; ігровий штучний інтелект.

Вступ

Питання штучного інтелекту важливе у галузі розроблення відеоігор. Історично ігровий штучний інтелект розвивався практично паралельно з галуззю відеоігор загалом і вже в перших відомих відеоіграх, таких як Pac-man чи Mario, широко застосовували персонажі, що керуються примітивними моделями штучного інтелекту. Здебільшого ігровий штучний інтелект виконує одне із двох завдань: або керує неігровими персонажами (англ. Non-player character, NPC) і явищами, що оточують гравця, або ж покликаний замінити реальних гравців, з якими контактує користувач. Штучний інтелект другого типу зазвичай набагато складніший, тому що він повинен автентично імітувати поведінку справжньої людини-гравця. Оскільки відеоігри надзвичайно різноманітні за

жанрами, за правилами гри і тим, як влаштовано ігровий світ, то існує багато різних підходів і моделей для розроблення штучного інтелекту для конкретної гри.

Одним із найпростіших підходів до реалізації ігрового штучного інтелекту є побудова поведінки на основі невеликого переліку простих правил. Взагалі це важко назвати штучним інтелектом. Персонажі, якими керує така модель, залежно від заданих програмою умов (що можуть спиратися на власний стан персонажа і на стан світу, що його оточує), виконують ті чи інші дії. Ця модель підходить лише для найпростіших ігор, у яких перелік можливих дій персонажа, яким керуватиме така модель, дуже обмежений.

Підходом, який істотно розширяє і покращує ідею з правилами, є розроблення моделі ігрового штучного інтелекту на основі скінченних автоматів. У керованого персонажа є набір станів, кожен з яких задає певну поведінку для нього. Задають також набір правил, за якими здійснюється перехід між цими станами. Таку модель доволі широко використовують у значній кількості сучасних відеоігор, переважно в таких, де ігровий процес триває в реальному часі (тобто гра не є покроковою).

Підхід із деревами рішень часто застосовують у багатьох комп'ютерних версіях класичних настільних ігор (шахи, шашки, го), а також для багатьох ігор жанру покрокової стратегії. Штучний інтелект аналізує всі можливі стани гри на певну кількість ходів вперед і обчислює оцінку для кожного можливого ходу на основі того, наскільки цей хід є вигідним для нього. Після цього модель здійснює той хід, який отримав найвищу оцінку.

Останнім часом як моделі для ігрового штучного інтелекту набувають популярності також нейронні мережі. Щоб застосувати такий підхід, розробник повинен правильно сконструювати архітектуру, вибрати дані, які надходитимуть на вхідні вузли мережі, формат цих даних тощо. Крім того, таку модель потрібно правильно навчити.

Для навчання моделей ігрового штучного інтелекту, зокрема основаних на нейронних мережах, часто застосовують підхід навчання із підкріпленням (англ. Reinforcement learning). Підходи з наявними наборами даних, навчальною та тестовою вибіркою незастосовні до ігор, натомість навчання з підкріпленням добре для цього підходить, оскільки під час навчання можна симулювати безліч ігрових сценаріїв із випадковими модифікаціями тієї самої моделі. Після цього визначають, яка модель виявилась найкращою, і будують нові модифікації на її основі, які візьмуть участь у новому раунді симуляцій.

Одним із доволі популярних жанрів відеоігор є жанр рольової гри (англ. Role-Playing Game, RPG). У іграх цього жанру широко застосовують персонажі, що керуються ігровим штучним інтелектом. Найкращі підходи для такого жанру – модель скінченного автомата та нейронна мережа. Втім, якщо гра покрокова, для неї також можна застосувати модель дерев рішень.

Розвиток нових систем і підходів керування персонажами у відеоіграх жанру рольової гри актуальний, оскільки практично всі такі відеоігри застосовують лише примітивні моделі поведінки персонажів, що ґрунтуються на наперед визначених простих правилах чи на скінченному автоматі зі строго заданими правилами переходу між станами. З розвитком штучного інтелекту останніми роками з'явився потенціал залучення моделей машинного навчання у системах керування такими персонажами. Робота в цьому напрямі дасть змогу зробити персонажів ігор "живішими", а саму гру відповідно цікавішою.

Мета цієї статті – аналіз підходів, спрямованих на підвищення реалістичності поведінки ігрових персонажів рольових комп'ютерних ігор за допомогою побудови відповідних моделей ігрового штучного інтелекту на основі методів машинного навчання. Завдання полягає у розробленні системи керування персонажами гри на основі моделей машинного навчання з відповідно підібраними параметрами.

Об'єктом нашого дослідження є моделювання та управління поведінкою ігрових персонажів рольової комп'ютерної гри на основі методів штучного інтелекту, а предметом – моделі ігрового штучного інтелекту для відеоігор жанру RPG та методи ефективного навчання таких моделей

Наукова новизна роботи полягає в удосконаленні моделей та методів ігрового штучного інтелекту для керування неігровими персонажами у RPG іграх за допомогою поєднання методу скінченних автоматів з моделями машинного навчання, що дає змогу приймати рішення в реальному ігровому часі, враховуючи численні параметри, які описують конкретну ігрову ситуацію.

Практична цінність роботи у тому, що удосконалений ігровий штучний інтелект можна застосовувати в різних іграх цього жанру, а розроблену систему – інтегрувати в наявні комерційні та вільні відеоігри. Залучення такої системи для розроблення неігрових персонажів зробить їх "розумнішими", взаємодія з ними стане цікавішою для гравців та заохочуватиме їх до повторного проходження гри (англ. Replayability), тобто зросте імовірність того, що користувачі будуть довше грати у неї. Це прямо впливає на репутацію та комерційний успіх ігрового продукту.

Постановка проблеми

Вважаємо за доцільне дослідити та вибрати моделі машинного навчання, придатні для цієї предметної області, здійснити їх навчання та оптимізацію параметрів та розробити систему керування неігровими персонажами RPG гри, яка для керування персонажами буде застосовувати моделі машинного навчання, навчені за підходом навчання з підкріпленням, з метою досягнення максимальної ефективності поведінки таких персонажів та наближеності їх до людей-гравців.

Аналіз останніх досліджень та публікацій

У [1] розглянуто розроблення системи навігації та поведінки для одночасного керування групою персонажів та координування їх дій. Ця система проєктується для гри у жанрі стратегії в реальному часі (англ. Real-time strategy, RTS). Для побудови маршрутів, до яких мають рухатися персонажі, застосовують деякі ідеї з теорії поля. Різні об'єкти світу приймають за точкові заряди з різним потенціалом і для кожної точки світу обчислюють деякий заряд, який є сумою потенціалів точкових зарядів усіх об'єктів із урахуванням відстані до них. Персонажі намагаються оминати точки з негативним зарядом і дістатись до якоїсь з точок з позитивним зарядом. Основним недоліком цього підходу є його висока обчислювальна складність, оскільки в ідеалі потрібно обчислити значення заряду у всіх точках ігрової локації, яка потенційно може бути безмежною.

У статті [2] розглянуто модель поведінки ботів під час стрільби у грі жанру тривимірного шутера від першої особи (англ. First person shooter, FPS). Автори публікації зазначають, що розробити бота, який стрілятиме максимально швидко і з ідеальною точністю, доволі легко, оскільки швидкодія комп'ютера і точність заданих ним рухів персонажа вищі від людської. Натомість автори ставлять перед собою завдання розробити модель поведінки, яка вестиме стрільбу подібно до людини, для того, щоб користувач не міг відрізнити ботів, з якими він грає, від реальних людських гравців. Модель розробляється на прикладі гри Unreal Tournament 2004, на основі алгоритму SARSA, на вхід моделі подається інформація про наявні види зброї у персонажа, а також інформація про рух цілі, по якій потрібно стріляти. Модель на основі цих даних приймає рішення про те, куди стріляти. Для навчання моделі застосовують підхід Reinforcement Learning, як показник винагороди використовують кількість очок шкоди, заподіяної цілі. Недолік моделі – її прив'язаність до вибраної ігрової локації та до переліку зразків озброєння – такий ігровий штучний інтелект потрібно перенавчати під кожну нову ігрову локацію і щоразу, коли у гру додається новий вид зброї.

Публікація [3] стосується навігації персонажів, тобто їх грамотного переміщення по ігрових локаціях, під час якого вони досягають цілі у будь-якій точці ігрової локації, будуючи до неї маршрут і не застрягають ніде на своєму маршруті. Стаття для прикладу знов-таки розглядає гру Unreal Tournament 2004. У цій грі для навігації ботів на кожній ігровій локації є граф із великою кількістю вершин і боти будують свій маршрут через ігрову локацію по вершинах графа за алгоритмом A*. На деяких ігрових локаціях персонаж, керований штучним інтелектом, з тієї чи іншої причини іноді потрапляє за межі графа і застрягає. Автори статті розглядають різні способи визначення того, що бот застряг (наприклад, поблизу немає жодної вершини навігаційного графа, або бот дуже довго

перебуває біля однієї вершини, коли мав би рухатись до наступних). Розглянуто також способи вирішення таких ситуацій. Автори пропонують простежувати маршрути реальних людських гравців по ігровій локації та записувати ці маршрути в базу. Якщо бот застряг, то з бази видобувають людський маршрут, який проходить якомога ближче до позиції бота, і бот рухається цим маршрутом, щоб вибратись з проблемної частини ігрової локації. Це доволі простий і водночас ефективний підхід, який також можна було б адаптувати до гри у жанрі RPG. Основним недоліком, очевидно, є те, що для кожної ігрової локації потрібно наповнити базу маршрутами людей, тобто кожну ігрову локацію повинні "обіграти" люди.

Публікація [4] розглядає модель поведінки ігрового штучного інтелекту EISBot для гри жанру стратегії в реальному часі StarCraft. Ця модель штучного інтелекту працює на основі "поведінкової мови" (англ. A behavior language, ABL), за допомогою якої вибудовують поведінкові дерева (англ. Behavior tree, BT). Цілі та дії для досягнення цілей утворюють вузли та листки дерева відповідно. Кожному листку дерева присвоюють певний пріоритет, і модель приймає рішення для досягнення певної цілі, вибираючи листок дерева із найвищим пріоритетом. У статті зазначено, що така модель не вчиться на власних помилках, оскільки пріоритети дій є статичними, і якщо вибрана дія з найвищим пріоритетом виявляється помилковою, її пріоритет все ще залишається найвищим. Під час наступної спроби досягти тієї самої цілі така помилка буде допущена знову. Автор пропонує модифікацію наявної моделі поведінки штучного інтелекту, у якій пріоритети дій для досягнення цілей є динамічними – вони знижуються, якщо дія виявляється помилковою, та збільшуються, якщо дія дала змогу успішно досягти цілі, задля якої її здійснено. Недоліком моделі можна назвати те, що модель повинна зробити кілька помилок, перш ніж виставить пріоритети правильно. Крім того, в разі зміни стратегії, яку здійснив опонент, імовірно, доведеться перевизначити пріоритети дій, які може вибирати модель, що призводить до додаткових помилок.

У роботі [5] розглянуто підходи до навігації у іграх-платформерах на основі поведінкових дерев у поєднанні з еволюційними алгоритмами та алгоритмом пошуку маршрутів A*. Модель випробувано на грі Mario AI Benchmark – це модифікація класичної Mario з відкритим кодом, розроблена спеціально для тестування моделей штучного інтелекту. Еволюційний алгоритм у ході навчання будує поведінкове дерево, у якому закладені різні моделі поведінки, придатні для різних ситуацій, у яких може опинитися гравець. Цей підхід важко адаптувати до гри жанру RPG, оскільки алгоритми поведінки у грі-платформері істотно відрізняються від більшості інших жанрів.

Автори [6] розглядають підхід для навігації персонажів на основі концепції потенціальних полів та точкових зарядів у іграх жанру стратегії в реальному часі. Вони застосовують моделі на прикладі гри StarCraft. Розглянуто розширену версію моделі із зарядами, у якій той самий об'єкт може мати різний точковий заряд для різних персонажів, залежно від типу персонажів, їх цілей, озброєння та можливостей. Наприклад, якщо один персонаж повинен атакувати певний об'єкт, а інший – уникати його, то для першого персонажа цей об'єкт матиме додатний заряд, а для другого – від'ємний. Також у статті проаналізовано особливість ігрових локацій цієї гри – наявність великої кількості стратегічних вузьких коридорів, проблемних для навігації запропонованим підходом. Запропоновано підхід, що поєднує цей алгоритм з алгоритмом A*: алгоритм A* застосовується для навігації, коли персонажі просто рухаються до будь-якої точки, а алгоритм із потенційними полями задіюється тоді, коли поблизу є ворожі персонажі чи об'єкти, які потрібно атакувати.

Робота [7] розглядає поєднання еволюційного програмування із моделлю поведінкових дерев. Запропонований підхід реалізується на прикладі гри Mario AI Benchmark. У публікації описано поєднання алгоритму A* для навігації по ігровій локації з алгоритмом поведінкових дерев для прийняття рішень та з еволюційним програмуванням для побудови конкретних екземплярів дерев. Стаття пояснює принцип роботи еволюційного алгоритму, який на основі заданих граматичних правил конструює поведінкові дерева. Також обговорюється проблема узагальнення – рівні для гри генеруються довільно, і дерева, отримані еволюційним підходом на одних ігрових рівнях, можуть не підходити для інших рівнів. Запропоновано способи вирішення цієї проблеми: еволюція того самого

поведінкового дерева на кількох випадкових рівнях або еволюція на одному “усередненому” рівні, який має достатньо особливостей для еволюційного отримання універсального поведінкового дерева.

Способи імітації людини-гравця розглянуто в статті [8]. Модель розроблено та перевірено у грі “The Open Racing Car Simulator” (TORCS), вона використовує рекурентні нейронні мережі (Recurrent neural network, RNN). На вхід до мереж подаються різні показники, такі як швидкість автомобіля, відхилення від середини дороги, відстань до країв маршруту, яка вимірюється 19 віддалемірами, кут відхилення автомобіля від напрямку дороги тощо. Для оцінювання роботи моделі застосовано три метрики, одна з них орієнтована на якість і швидкість водіння автомобіля, а дві інші – на подібність до того, як автомобілем керувала би людина-гравець. Цей підхід важко адаптувати до гри у жанрі RPG, оскільки вхідні дані, які подаються на модель у грі жанру автомобільних перегонів, практично нерелевантні у іграх інших жанрів.

У статті [9] розглянуто ігровий штучний інтелект для гри у RPG жанрі під назвою “Desktop Dungeons”. Особливість цієї гри у тому, що рівні завжди генеруються випадково, що ставить перед моделлю ігрового штучного інтелекту вимогу до вміння узагальнювати. Автори публікації розробили “жадібний” алгоритм штучного інтелекту для керування гравцем та еволюційний алгоритм для його детального налаштування. “Жадібний” алгоритм має два впорядковані списки стратегій – стратегії для проходження гри та стратегії для вбивства ворожого персонажа. Еволюційний алгоритм покликаний, навчаючись, впорядкувати стратегії у кожному зі списків та налаштувати їх параметри так, щоб зробити “жадібний” алгоритм максимально ефективним. Одним із основних недоліків запропонованого підходу є примітивність “жадібного” алгоритму.

Застосування згорткових нейронних мереж (Convolutional neural network, CNN) розглянуто в науковій розвідці [10] на прикладі гри у жанрі шутера від першої особи ViZDoom, яка є модифікацією класичної гри Doom. Автори публікації пропонують згорткову нейронну мережу, на вхід якої подається зображення, яке бачить користувач під час гри, а на виході мережа віддає дію, яку повинен виконати персонаж. У статті наведено кілька експериментів з різними способами оцінювання ефективності моделі.

Детальне аналізування показало, що для випробування моделей ігрового штучного інтелекту найчастіше застосовують жанри RTS, FPS і платформери. Популярними алгоритмами для ігрового штучного інтелекту є алгоритм A*, поведінкові дерева, деякі види нейронних мереж.

У відеоіграх жанру FPS та автоперегонів часто використовують різні моделі нейронних мереж та підходи глибинного навчання: RNN, CNN, SARSA. Ці моделі добре поєднуються з такими жанрами, оскільки у іграх цих жанрів багато неперервних мінливих параметрів, які можна передати у таку модель – це координати різних об’єктів у грі, швидкість їх руху, відстань гравця до перешкод тощо. Всі ці величини є неперервними, їх багато і вони швидко змінюються під час гри. Також у одній з публікацій запропоновано цікавий підхід, який максимально уподібнює модель до людини – на вхід моделі подається лише зображення з монітора комп’ютера, ніякі додаткові параметри чи показники з самої гри не надаються. Навчання такої моделі надзвичайно довге і складне, оскільки передбачає і навчання моделі розпізнаванню зображення, проте на деяких примітивних сценаріях така модель все одно виявилась успішною.

Натомість у іграх жанрів RTS та платформера часто застосовують моделі з поведінковими деревами та алгоритм пошуку шляху A*. У цих жанрах використовувати модель поведінкових дерев доцільніше, зважаючи на особливості гри і те, яка інформація доступна для персонажів у таких іграх. Існує набагато більше різних ситуацій та сценаріїв, у які можуть потрапляти персонажі, і поведінкові дерева добре підходять для вибору стратегій поведінки у різноманітних сценаріях. Методи, пропоновані для жанру RTS, здебільшого можна застосувати і для жанру RPG, оскільки ці два жанри схожі в деяких аспектах. З іншого боку, розроблювані під жанр платформера моделі для жанру RPG застосувати дуже важко. У платформерах велике значення має фізичний рушій і вся гра зосереджена на тому, що гравець долає перешкоди у фізичній симуляції, тоді як у RPG іграх роль фізичного рушія під час гри набагато менша.

Формулювання цілі статті

Основні цілі статті – описати розроблення та реалізацію системи керування неігровими персонажами за допомогою моделей машинного навчання, збирання даних для навчання моделей та власне навчання вибраних моделей машинного навчання, а також проаналізувати отримані результати.

Виклад основного матеріалу

Гра, для якої розробляється система керування персонажами, є браузерною грою з підтримкою багатокористувацького ігрового процесу в реальному часі в мережі інтернет. Серверну частину гри розроблено на платформі Node.js [2], клієнтську частину – із застосуванням фреймворку Angular 2. Обидва компоненти використовують ігровий рушій Phaser 3, що надає засоби роботи з графікою та фізичний рушій, а також мову програмування TypeScript.

Користувач може керувати своїм персонажем, подорожувати ним по ігровому світу, змінювати спорядження та вступати в бій із ворожими персонажами. Цими ворожими персонажами можуть керувати як інші користувачі, так і штучний інтелект.

Усі персонажі у грі керуються сутностями, що називаються “контролерами”. Ідея контролерів полягає у тому, щоб інкапсулювати модель поведінки та прийняття рішень персонажа та відокремити її від самого персонажа, яким ця модель поведінки керує. Це робить контролери взаємозамінними – і гравець, і комп’ютер можуть керувати тими самими персонажами, достатньо просто підмінити контролер персонажа.

Усі типи контролерів реалізують інтерфейс “Controller”, який має два поля, щоб передавати повну інформацію для керування довільним персонажем. Це поле movement, яке містить вектор напрямку руху персонажа, та поле attack, що містить координати атаки персонажа.

Шкода, якої зазнають ворожі істоти від атак, залежить від значення та типу шкоди зброї, а також від того, яка броня на цих істотах. Якщо персонажі у броні, то під час удару частина шкоди від зброї втрачається, і залежно від типу шкоди і кількості броні ці втрати різняться.

На стороні сервера розроблено декілька різних контролерів, що реалізують інтерфейс Controller та керують персонажами: WalkerController (контролер для керування примітивними ворогами), ServerWrapperController (контролер, що слугує обгорткою для будь-якого іншого контролера з метою синхронізації з клієнтом), ClientController (контролер для персонажа, яким керує гравець через вебсокет) і StateMachineController (контролер для керування ворогами на основі скінченного автомата).

На рис. 1 зображено UML-діаграму архітектури розроблених контролерів.

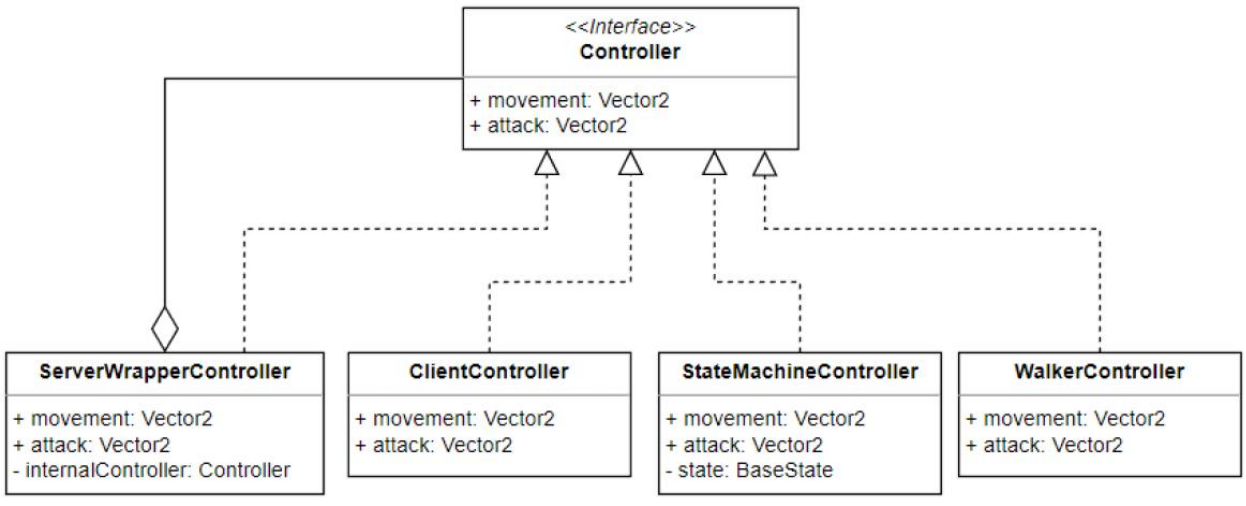

Рис. 1. UML-діаграма архітектури контролерів керування персонажами

Контролер StateMachineController реалізує складну логіку поведінки керованого комп'ютером персонажа. Цей контролер містить скінченний автомат із декількох станів. У кожен момент часу в цього контролера є активним якийсь один стан, і саме активний стан визначає алгоритми обчислення векторів movement та attack цього контролера. Також активний стан має власний алгоритм, за яким може замінити себе на інший активний стан. У сукупності логіка переходів всіх цих станів один між одним утворює систему жорстких правил переходу між станами. Всі можливі стани наслідуються від базового абстрактного класу BaseState, який має такі поля, як movement, attack, currentState, та метод transitionState, що визначає, на який стан потрібно перейти: IdleState (стан простоювання), NavigateToTargetState (стан руху до певної цілі за маршрутом, побудованим алгоритмом A*), AttackState (стан нападу на ціль) і RetreatState (стан втечі).

Для випробування персонажів, керованих штучним інтелектом, розроблено архітектуру тестових кімнат як надбудову чи розширення над наявною архітектурою сцен для звичайного багатокористувацького ігрового процесу. Рушій Phaser надає вбудований клас Scene [3], від якого наслідується клас LevelScene, що реалізує фундаментальну ігрову логіку, необхідну для ігрового процесу. Від нього відтак наслідується клас NetworkLevel.

Клас NetworkLevel реалізує будь-яку ігрову локацію із підтримкою багатокористувацького ігрового процесу. Він містить логіку, пов'язану з опрацюванням підключень нових гравців, завантаженням ігрових рівнів, створенням персонажів та синхронізацією усіх ігрових подій між клієнтами.

Клас AIPlaygroundLevel наслідує функціонал від NetworkLevel та розширяє його. В межах цього класу розроблено фунціонал, корисний саме для проведення тестових боїв між групами персонажів – це зокрема система збирання статистики та система відстеження закінчення бою. Також цей клас, для створення персонажів, які будуть тестуватись, використовує систему конфігурацій боїв. На початку бою екземпляр цього класу використовує надану йому конфігурацію бою та повністю делегує їй створення персонажів, що братимуть участь у битві. Це дає змогу розробити безліч конфігурацій, що імітуватимуть різні сценарії.

Клас AITrainingLevel є подальшим розширенням AIPlaygroundLevel, він відрізняється здебільшого тим, що сам вибирає випадкову конфігурацію зі списку вказаних. Цей клас застосовуватиметься для генерації набору даних, він збирає дані під час бою, а після закінчення бою зберігає отримані дані та припиняє роботу конкретного екземпляра гри.

Побудова системи тестових кімнат на базі багатокористувацької системи з підтримкою підключень по мережі уможливлює ігрову симуляцію тестової кімнати на боці сервера і водночас підключення до такої кімнати через ігровий клієнт, а також можливість підключити інших користувачів з метою спостереження за грою або участі в ній.

Система конфігурації боїв реалізована через абстрактний клас BaseBattlePreset та класи, що його наслідують. Базовий клас надає допоміжний функціонал для генерації персонажів та їх спорядження, а кожна його реалізація дає змогу сформувати зовсім інші умови бою. У ході випробувань персонажів у різних ситуаціях розроблено багато різних конфігурацій. Зразки розміщення персонажів у деяких з них подано на рис. 2.

Навчання моделей, що керуватимуть персонажами, повинно здійснюватись на певному наборі даних. Оскільки ця гра є власною розробкою, то для неї не існує готових наборів даних. Тому необхідно створити власну вибірку даних, на якій моделі навчатимуться. Спершу треба виділити атрибути цієї вибірки, на основі яких моделі будуть приймати рішення під час гри. Як атрибути у вибірку даних буде внесено поточний стан керованого персонажа, його попередній стан, перелік параметрів, таких як кількість очок здоров'я, броні, шкоди, швидкість руху для керованого персонажа, вибраної цілі, а також агреговані (усереднені чи максимальні) варіанти цих параметрів для всіх союзників та ворогів, які перебувають неподалік. Агреговані параметри щодо ворогів і союзників застосовують для того, щоб змусити моделі, під час прийняття рішення, також враховувати різноманітну інформацію стосовно усіх гравців на арені. Це уможливлює певну командну взаємодію,

наприклад, дає змогу персонажам діяти сміливіше, коли вони знають, що у них є перевага за кількістю над ворогом. Оскільки кількість як союзників, так і ворогів, змінна, доводиться застосовувати параметри, усереднені по всій групі сутностей, оскільки у наборі даних кількість атрибутів не може змінюватись.

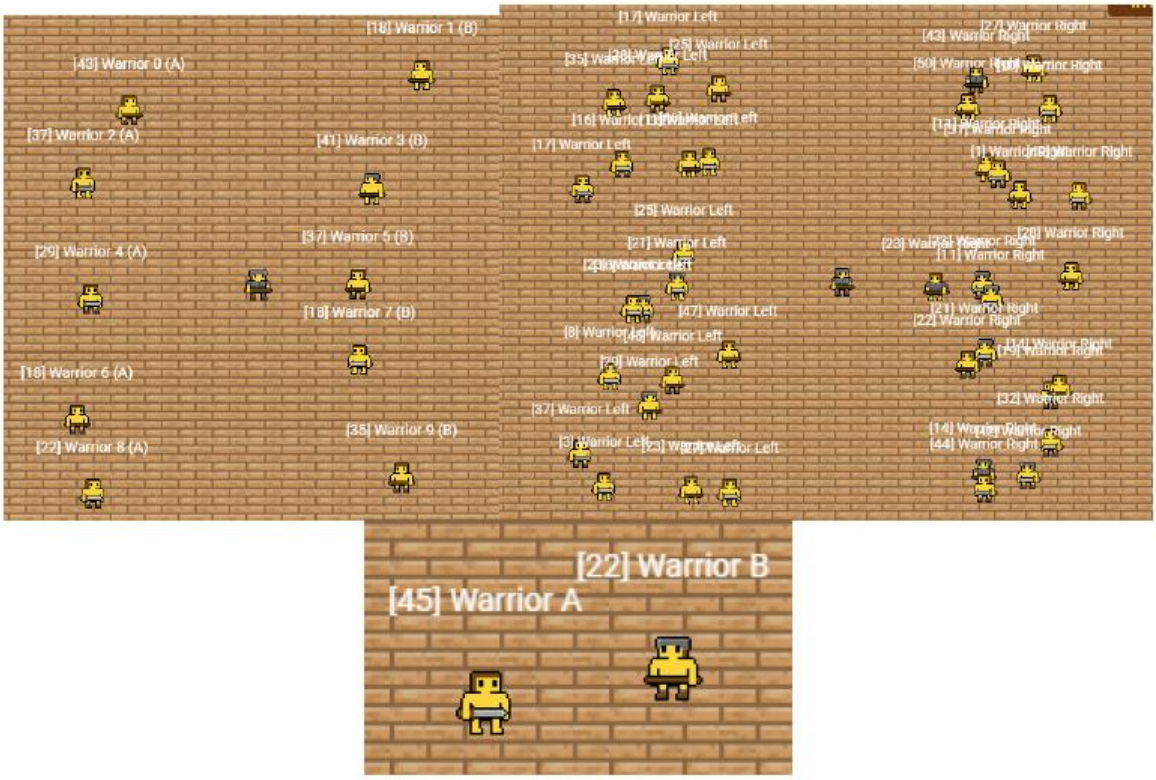

Рис. 2. Зразки розміщень і спорядження персонажів у різних конфігураціях

Усі ці атрибути разом утворюють сутність, яка називається “контекстом ситуації”. Всі моделі, що навчатимуться, вчитимуться приймати рішення про поточний стан персонажа на основі сформованого в реальному часі контексту ситуації. По суті, завдання зводиться до класифікації, оскільки на основі переліку атрибутів потрібно визначити найпридатніший стан персонажа у певний момент часу із фіксованого переліку цих станів. На рис. 3 схематично зображено повний перелік атрибутів, які входять у контекст ситуації.

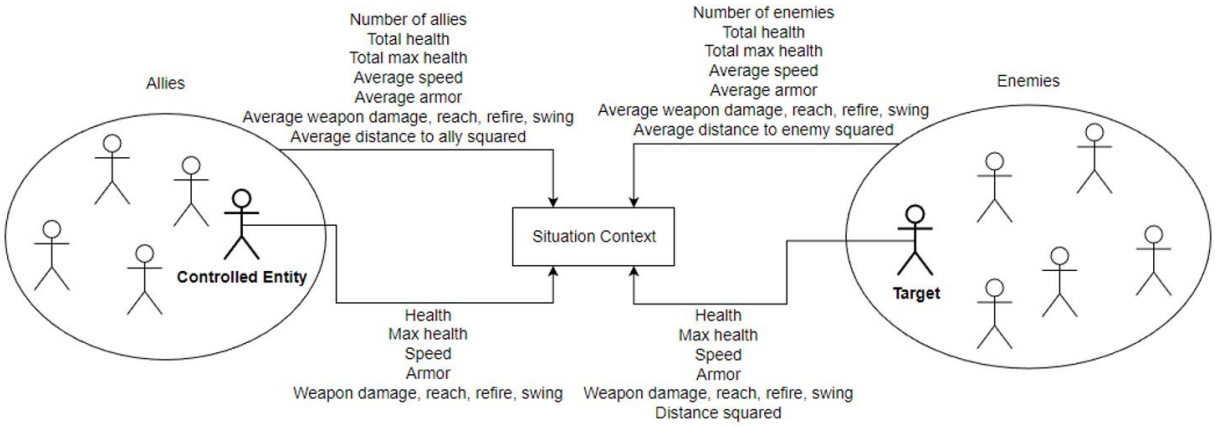

Рис. 3. Формування контексту ситуації

Визначивши атрибути набору даних, необхідно власне згенерувати самі дані. Для цього розроблено систему генерації датасету, яка запускає паралельно велику кількість екземплярів гри. У кожному екземплярі вибирають випадкову конфігурацію бою та випадкове спорядження для всіх персонажів. Самі персонажі керуються стандартним скінченним автоматом із жорсткими правилами. У ході ігрового процесу цих екземплярів, у кожному екземплярі, для кожного персонажа, під час кожного кадру гри записують поточний контекст ситуації, а також попередній і поточний стани персонажа. Коли бої у всіх екземплярах закінчились, вся записана інформація зберігається у новий CSV файл у вигляді набору даних, для подальшого навчання моделей на цьому наборі. На рис. 4 схематично зображено структуру системи генерації наборів даних, яка містить багато випадкових екземплярів гри.

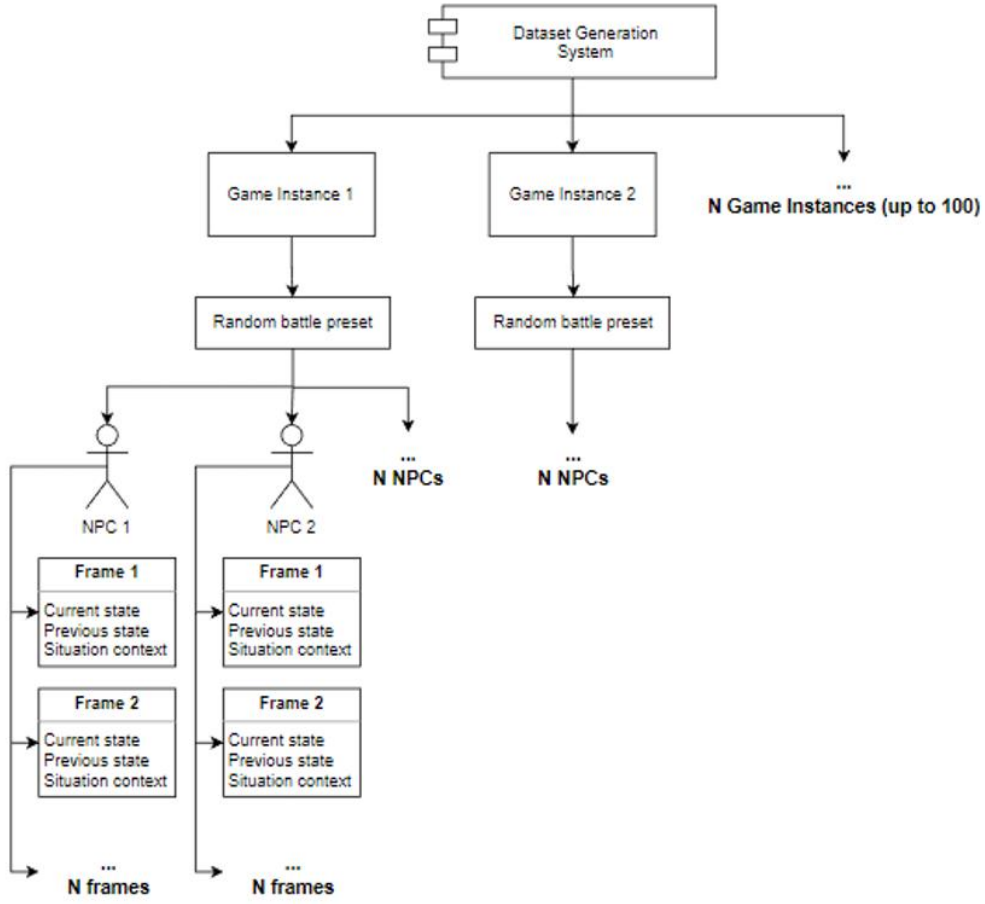

Рис. 4. Структура системи генерації датасету

Для навчання штучного інтелекту керувати персонажами на основі набору даних, згенерованого під час великої кількості ігор, вибрано кілька різних методів машинного навчання, які унаслідок своїх особливостей можуть ефективно або не дуже справлятись із навчанням на згенерованому наборі цієї предметної області. Серед вибраних методів такі, як метод дерев рішень (decision trees), K-найближчих сусідів (KNN), метод опорних векторів (SVC), метод гауссівського процесу (Gaussian process), випадкового лісу (random forest) та нейромережева модель (neural network).

Під час навчання найкращий результат серед всіх моделей показали модель дерева рішень та нейромережева модель, тому було прийнято рішення після експериментів з навчанням на наборі даних зберегти саме ці навчені моделі у файли та інтегрувати в ігрову систему.

Навчання моделі дерев рішень на згенерованому наборі даних дало найкращий результат з усіх класифікаторів, вдалося отримати точність 0,997. Таку високу точність у цьому випадку можна пояснити тим, що саму вибірку даних згенеровано на основі моделі поведінки, що керується жорсткими правилами, і ця модель змогла успішно вивчити подібні правила з одержаної вибірки.

Навчання нейромережевої моделі здійснено на сервері гри з використанням адаптації бібліотеки Tensorflow.js під платформу Node.js. Архітектура мережі складається із вхідного шару на 33 нейрони (що відповідає кількості вхідних полів) із сигмоїдною функцією активації, прихованого шару на 16 нейронів із функцією активації softmax та вихідного шару на чотири нейрони (що відповідає кількості можливих станів). У результаті роботи мережа видає масив із чотирьох чисел, кожне в інтервалі [0;1]. Перше число відповідає стану idle, друге – стану navigateTo Target, третє – attack і четверте – retreat. Найбільше число у масиві є тим станом, який модель вважає найдоцільнішим у цій ситуації. Модель навчалась на тому самому наборі даних, розділеному на групи по 100 рядків та на 50 епохах навчання. У ході навчання значення loss (функції втрат) становило близько 0.77.

Для інтеграції вибраних моделей у ігрову систему контролер State Machine Controller розширено двома новими контролерами, що наслідуються від нього: Neural Net Classifier State Machine Controller та Decision Tree Classifier State Machine Controller. Ці контролери підміняють стандартну логіку переходу між станами на основі жорстких правил, замість цих правил рішення про перехід між станами у цих контролерах приймають відповідно моделі нейронної мережі та дерева рішень. Коли у контролера виникає потреба у тому, щоб змінити стан персонажа, він збирає всі доступні йому дані та формує їх у вхідний об’єкт для моделі, після чого передає його моделі й отримує від неї стан, до якого потрібно перейти.

Після інтеграції навчених моделей у систему гри було розроблено систему збирання статистики боїв, щоб визначити, наскільки добре різні моделі показують себе в дії. Ця система паралельно запускає велику кількість екземплярів гри, у кожному з яких дві команди по п’ять персонажів отримують випадкове спорядження і вступають у бій одна з одною. Усі ігрові події, такі як початок бою, заподіяння шкоди будь-якому з персонажів, смерть персонажа чи кінець бою логуються, інформація про них агрегується і записується у CSV файл. Серед інформації, що записується у файл щодо кожної команди кожної гри, є такі дані, як назва команди (формується на основі моделі, що керує персонажами цієї команди), показник того, чи виграла ця команда гру, відсотки часу у станах атаки, навігації та відступу, кількість та відсоток вбитих ворогів, втрачених союзників, заподіяної шкоди, отриманої шкоди, а також ідентифікатор кімнати, за яким можна зіставити дві команди з того самого екземпляра гри. Відсотки заподіяної та отриманої шкоди розраховують на підставі сумарної кількості очок здоров’я відповідно ворожої та власної команди.

Для порівняння моделей зібрано статистику стосовно трьох різних комбінацій моделей у боях: модель із жорсткими правилами проти моделі дерев рішень, модель із жорсткими правилами проти нейромережевої моделі та модель дерев рішень проти нейромережевої моделі. У кожному випадку для збирання статистики було запущено 1000 екземплярів гри, що в результаті дає статистику по 2000 командах, керованих двома різними моделями.

У табл. 1 наведено статистику за 1000 боїв між моделлю на жорстких правилах та моделлю дерев рішень. Зі статистики видно, що відсоток перемог у моделі дерев рішень трохи вищий, ніж у стандартної моделі скінченного автомата із жорсткими правилами переходу станів. Також персонажі, що керуються моделлю дерев рішень, трохи рідше втікають і трохи частіше перебувають у стані навігації чи атаки. Крім цього вони в середньому знищують більше ворогів та заподіюють більше шкоди (очок) ворогам. Тому можна зробити висновок, що, порівняно зі звичайною простою моделлю, модель дерев рішень трохи покращила поведінку персонажів у бою.

Таблиця 1

Статистика боїв між моделлю на жорстких правилах та моделлю дерев рішень

Модель	Перемоги, %	Час в атаці, %	Час в навігації, %	Час у втечі, %	Середня кількість знищених ворогів, шт./%	Середня заподіяна шкода, %
Жорсткі правила	46,2	29,54	55,27	14,14	2,776 (55,52 %)	538 (86,5 %)
Дерева рішень	50,1	29,71	56,18	13,05	2,901 (58,02 %)	552 (88,45 %)

У табл. 2 подано статистику для порівняння моделі на жорстких правилах та нейромережевої моделі, зібрану під час 1000 ігор. Як бачимо, відсоток перемог у нейронної мережі істотно більший, ніж у моделі на жорстких правилах, тобто така модель перемагає набагато частіше. Цікаво, що персонажі, керовані моделлю нейромережі, жодного разу не втікали від ворога, також вони частіше атакують чи просто рухаються по кімнаті. Крім цього вони в середньому знищують помітно більше ворогів та заподіюють в середньому більше шкоди, ніж персонажі, керовані моделлю на жорстких правилах. Отже, модель нейронної мережі істотно підвищила ефективність персонажів у боях, порівняно зі звичайною моделлю на жорстких правилах. Персонажі, керовані цією моделлю, перемагають набагато частіше, заподіюють ворогам більше шкоди (очок) та загалом знищують більше ворогів.

Таблиця 2

Статистика боїв між моделлю на жорстких правилах та моделлю дерев рішень

Модель	Перемоги, %	Час в атаці, %	Час у навігації, %	Час втечі, %	Середня кількість знищених ворогів, шт./%	Середня заподіяна шкода, %
Жорсткі правила	29,4	25,5	56,62	16,44	2,334 (46,68 %)	498 (79,92 %)
Нейронна мережа	69,7	36,8	62,96	0	3,31 (66,2 %)	589 (94,34 %)

У табл. 3 за статистикою порівняно навчені моделі дерев рішень та нейронної мережі одна із одною. Модель нейронної мережі перемагає модель дерев рішень приблизно так само часто, як перемагала модель на жорстких правилах. Персонажі, керовані нейромережевою моделлю, знову жодного разу не втікали, натомість атакували і рухались по кімнаті частіше, ніж персонажі, керовані моделлю дерев рішень. Також модель нейронної мережі знищила в середньому набагато більше ворогів і заподіяла більше шкоди (очок). Отже, модель нейронної мережі показала, що проти моделі дерев рішень вона така ж ефективна, як і проти примітивної моделі на жорстких правилах. Проти моделі дерев рішень нейромережева модель перемагала навіть трохи частіше, проте, імовірно, це спричинено статистичною похибкою і за більшої кількості ігор для збирання статистики показники могли б бути іншими.

Таблиця 3

Статистика боїв між моделлю на жорстких правилах та моделлю дерев рішень

Модель	Перемоги, %	Час в атаці, %	Час в навігації, %	Час втечі, %	Середня кількість знищених ворогів, шт./%	Середня заподіяна шкода, %
Дерева рішень	28,3	23,81	59,06	15,76	2,352 (47,04 %)	501 (80,11 %)
Нейронна мережа	70,6	35,25	64,62	0	3,354 (67,08 %)	595 (95,19 %)

Висновки

У статті описано розроблення системи керування неігровими персонажами на основі моделей машинного навчання із залученням навчання з підкріпленням у багатокористувацькій RPG. У галузі розроблення відеоігор та зокрема штучного інтелекту доволі мало розробок із залученням моделей машинного навчання у потенціально комерційних ігрових продуктах, які б вийшли за межі прототипу. Ще менше таких розробок, що перебувають у відкритому доступі. Тому цей напрям вважаємо актуальним та таким, що варто розвивати.

Спроєктовано та реалізовано систему керування персонажами із класичною моделлю поведінки на основі жорстких програмованих правил, яка підтримує інтеграцію моделей машинного навчання, а також інфраструктуру для зручного розроблення, інтеграції, тестування та використання таких моделей. Така інфраструктура містить систему тестових кімнат та конфігурацій для симуляції різних сценаріїв, у які потрапляють персонажі. Описано систему керування персонажами на основі контролерів та зокрема контролер на основі жорстких правил, що є своєрідною моделлю скінченного автомата. Відібрано також деякі моделі машинного навчання, які потенційно можна інтегрувати у цей проєкт.

Виконано експерименти із навчання відібраних моделей машинного навчання на отриманому наборі даних. Вони показали, що найкраще із цим завданням справляються моделі дерев рішень та нейронних мереж. Ці моделі ми інтегрували у систему гри, розширивши архітектуру контролерів, описану вище. Також було реалізовано систему збирання статистики, за її допомогою зібрано статистику, для зручності сприйняття оформлену в табличних поданнях. Ця статистика свідчить, що обидві моделі машинного навчання забезпечили підвищення ефективності поведінки персонажів порівняно зі звичайною моделлю, програмованою на жорстких правилах. Поліпшення моделі дерев рішень було незначним, а нейромережева модель дала помітно кращий результат.

Напрям подальших досліджень передбачає експерименти з ігровими кімнатами складніших конфігурацій, використання ансамблевих методів машинного навчання та складнішої архітектури нейронної мережі, зокрема нейронної мережі із довгою короткочасною пам’яттю (LSTM). Оскільки ігровий процес відбувається в реальному часі й нейромережа приймає певні рішення на кожному кроці, такі моделі зможуть дати кращий результат порівняно з простою моделлю.

Список літератури

1. Hagelbäck, J., & Johansson, S. J. (2009). A multiagent potential field-based bot for real-time strategy games. *International Journal of Computer Games Technology*, 2009, 1–10. https://doi.org/10.1155/2009/910819.

2. Glavin, F. G., & Madden, M. G. (2015). Adaptive shooting for bots in first person shooter games using reinforcement learning. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2), 180–192. https://doi.org/10.1109/TCIAIG.2014.2363042.

3. Karpov, I. V., Schrum, J., & Miikkulainen, R. (2013). Believable bot navigation via playback of human traces. In P. Hingston (Ed.), *Believable Bots,* 151–170. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-32323-2_6.

4. Okun, J. (2016). Evaluating the improvements of starcraft gameplay in the abl agent eisbot by implementing dynamic specificities through an external component. *MATEC Web of Conferences*, 68, 18001. https://doi.org/10.1051/matecconf/20166818001.

5. Nicolau, M., Perez-Liebana, D., O'Neill, M., & Brabazon, A. (2017). Evolutionary behavior tree approaches for navigating platform games. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(3), 227–238. http://dx.doi.org/10.1109/TCIAIG.2016.2543661.

6. Hagelback, J. (2012). Potential-field based navigation in starcraft. *IEEE Conference on Computational Intelligence and Games (CIG)*, Granada, Spain, 388–393. http://dx.doi.org/10.1109/CIG.2012.6374181.

7. Perez, D., Nicolau, M., O'Neill, M., & Brabazon, A. (2011). Reactiveness and navigation in computer games: different needs, different approaches. *IEEE Conference on Computational Intelligence and Games (CIG'11)*, Seoul, Korea (South), 273–280. http://dx.doi.org/10.1109/CIG.2011.6032017.

8. Van Hoorn, N., Togelius, J., Wierstra, D., & Schmidhuber, J. (2009). Robust player imitation using multiobjective evolution. *IEEE Congress on Evolutionary Computation*, Trondheim, Norway, 652–659. http://dx.doi.org/10.1109/CEC.2009.4983007.

9. Cerny, V., & Dechterenko, F. (2015). Rogue-like games as a playground for artificial intelligence – evolutionary approach. In K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, R. Malaka (Eds.), *Entertainment Computing – ICEC 2015*, 261–271. Springer. https://doi.org/10.1007/978-3-319-24589-8_20.

10. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016). ViZDoom: a Doom-based AI research platform for visual reinforcement learning. *IEEE Conference on Computational Intelligence and Games (CIG)*, Santorini, Greece, 1–8. http://dx.doi.org/10.1109/CIG.2016.7860433

11. OpenJS Foundation (n. d.). *Node.js About.* https://nodejs.org/en/about/

12. Photon Storm Ltd. (2021). *Phaser 3 API Documentation – Class: Scene.* https://photonstorm.github.io/phaser3-docs/Phaser.Scene.html.

MACHINE LEARNING METHODS FOR CONTROL OF NON-PLAYABLE CHARACTERS BEHAVIOUR IN MULTIPLAYER RPG

Roman Budnyk, Vitaliy Yakovyna

Lviv Polytechnic National University,
department of artificial intelligence systems, Lviv, Ukraine
E-mail: roman.budnyk.mknssh.2021@lpnu.ua, ORCID: 0009-0002-2576-531X
E-mail: vitaliy.s.yakovyna@lpnu.ua, ORCID: 0000-0003-0133-8591

This article covers the problem of developing a control system for non-player characters in a multiplayer RPG. Commercial projects in the field of videogames and RPG (Role-Playing Game) projects in particular seldom use machine learning models for the implementation of character behaviour. The most common approach is to use primitive preprogrammed rules, or to implement a finite state machine. Such approaches ruin the immersion of playing with real creatures, since various predefined rules make the characters predictable. A good game AI is supposed to give the player an impression of interacting with real characters, that make various decisions, sometimes unpredictable. To achieve this goal, this article covers an approach with using various machine learning models in conjunction with a traditional finite state machine. A videogame developed earlier is used as the basis for problem solution. The article conducts an analysis of the existing works in the field of videogame

AI. Next, the implementation of control system is described. This system utilizes a couple selected and successfully trained machine learning models. A multitude of models were tested, eventually a decision tree model and a neural network were selected, since they yielded the best results. The process of development and implementation of a control system involving machine learning models is then described. The approaches of teaching such models are described, and finally the achieved results are analyzed. To gauge the results, different models were compared against eachother in test battles. The decision tree model showed results slightly better than the traditional finite state machine. Meanwhile, the neural network performed significantly better, beating other models far more often. Achieved results can be developed further, utilizing more complex models and improving training methods, which will result in even more sophisticated characters. Presence of such characters in the game will qualitatively improve the gameplay. Obtained system was integrated into the videogame, that may potentially become a commercial product.

Key words: machine learning; videogame; game artificial intelligence.