УДК 004.89; 004.94

Н.О. СОКОЛОВА, В.В.ГНАТУШЕНКО, М.С. МІЩЕНКО, О.А. АТАМАНЧУК
Національний технічний університет «Дніпровська політехніка»

МОДЕЛЮВАННЯ ПОВЕДІНКИ НЕІГРОВИХ ПЕРСОНАЖІВ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

Комп'ютерні та відео-ігри – один з найбільших сегментів індустрії розваг, який стрімко розвивається у останні роки. Навіть пандемія пішла на користь цій галузі. Різноманіття ігор вражає, користувачі бажають отримувати якісний різноманітний контент, важливою частиною якого є неігрові персонажі.

Неігровий персонаж – це персонаж, керований програмою або майстром. Неігрові персонажі (доброзичливі або ворожі до гравця) слугують важливим засобом створення ігрової атмосфери, вони мотивують гравців робити ті чи інші дії, є основним джерелом інформації про ігровий світ та сюжет гри. Моделювання дій неігрових персонажів є важливою задачею при розробці ігор для покращення якості гри та більшого задоволення гравців. В сучасній розробці ігор для вирішення цієї задачі використовують алгоритми штучного інтелекту (ШІ), в тому числі машинне навчання, наприклад, для створення «розумних» ботів, які будуть боротися із гравцем. Ігровий ШІ не здатний на мислення чи творчість, його дії зумовлені розробниками, він підлаштовується під ситуацію та змінює поведінку залежно від контексту. Основними підходами у розробці поведінки неігрових персонажів є створення на фундаменті базових концепцій ШІ власних рішень для моделювання поведінки персонажів.

Дана робота присвячена моделюванню поведінки неігрових персонажів з використанням алгоритмів штучного інтелекту, зокрема дерева поведінки та алгоритму A пошуку. Дерево поведінки дозволяє неігровому персонажу приймати рішення в залежності від стану середовища та реагувати на його зміни. Переміщення є важливою частиною поведінки неігрових персонажів, які найчастіше некеровані гравцем, але часто логіка гри вимагає переміщення найкоротшим маршрутом задля динаміки гри. Проведена інтеграція змодельованої поведінки у вже існуючі вільні моделі ігрового рушія Unity, доведена коректна робота пошуку найкоротшого шляху за алгоритмом A*, враховуючи різні кейси оточуючого середовища та самого персонажу, а також працездатність системи навігації між сценами. Розроблені моделі поведінки можна у подальшому або інтегрувати у вже існуючі моделі, або використати при створенні нових ігор та персонажів.*

Ключові слова: розробка ігор, неігрові персонажі, моделювання поведінки, штучний інтелект, дерева поведінки, пошук найкоротшого шляху.

N.O.SOKOLOVA, V.V. HNATUSHENKO, M.S. MISHCHENKO, O.A. ATAMANCHUK
Dnipro University of Technology

MODELING THE BEHAVIOR OF NON-PLAY CHARACTERS BASED ON ARTIFICIAL INTELLIGENCE

Computer and video games are one of the largest segments of the entertainment industry, which has been developing rapidly in recent years. Even the pandemic benefited this industry. The variety of games is impressive, users want to get high-quality diverse content, an important part of which are non-game characters.

A Non-Player Character is a character controlled by a program or wizard. Non-Player Characters (friendly or hostile to the player) serve as an important means of creating a game atmosphere, they motivate players to take certain actions, and are the main source of information about the game world and the plot of the game. Modeling the actions of Non-Player Characters is an important task in game development to improve game quality and enhance player satisfaction. In modern game development, artificial intelligence algorithms are used to solve this problem, including machine learning, for example, to create «smart» bots that will fight with the player. Game AI is not capable of thinking or creativity, its actions are determined by developers, it adapts to the situation and changes behavior depending on the context. The main approaches in the development of the behavior of non-game characters are the creation of own solutions for modeling the behavior of characters based on the basic concepts of AI.

This work is devoted to modeling the behavior of Non-Player Character using artificial intelligence algorithms, in particular the Behavior Tree and the A search algorithm. A Behavior Tree allows a Non-Player Character to make decisions depending on the state of the environment and react to its changes. Movement is an important part of the behavior of Non-Player Characters, which are often not controlled by the player, but often game logic requires movement by the shortest route for the sake of game dynamics. The integration of the simulated behavior into the already existing free models of the Unity game engine was carried out, the correct work of finding the shortest path according to the A* algorithm, taking into account various cases of the environment and the character itself, as well as the functionality of the navigation system between scenes was proven. The developed behavior models can be further integrated into existing models or used in the creation of new games and characters.*

Keywords: game development, Non-Player Character, behavior modeling, artificial intelligence, Behavior Tree, shortest path search.

https://doi.org/10.32782/mathematical-modelling/2022-5-1-11

Постановка проблеми

Створення ігор – один із найбільших сегментів індустрії розваг, масштаби якого можна порівняти з кіноіндустрією. А за швидкістю зростання за останні п'ять років індустрія ігор суттєво її випереджала. У 2020 році за оцінками Google світовий ринок ігор зріс на 23,1% – це був найвищий показник за десятиліття і був оцінений у $177,8 млрд, очікувалося, що до 2024 року ринок досягне $218,7 млрд, при цьому середньорічний темп зростання становитиме 9,64% [1]. За даними британського видання Games Industry у 2022-му ринок оцінювався у $184,4 млрд [2]. За рік цей показник впав на 4,3%. Найбільше прибутку генерують мобільні ігри (50%). За ними йдуть консольні (28%), завантажувані чи фізичні ПК-ігри (21%) та браузерні ПК-ігри (1%). 94,2% продажів ігор були цифровими, лише 5,8% припало на фізичні копії. На ПК фізичні ігри майже не купують (2%), на консолях купують 28%. Найбільше продавалася FIFA 23 (Велика Британія), Call of Duty Modern Warfare 2 (США), у Японії – Splatoon 3.

В умовах пандемії ігрова індустрія, яка у 2020 році вийшла на першу позицію за рівнем затребуваності і залишається там і зараз, незважаючи на невелике падіння у 2022 році, дарує розробникам ігор шанс знайти свою нішу. Навіть в умовах війни європейські та інші компанії були не проти поширити свої робочі місця для українців [3], тому розробка алгоритмів з використанням штучного інтелекту для ігрової індустрії є актуальною. При розробці ігор два шляхи: скористатись повністю готовим ШІ або ж взяти готові концепції та на їх основі створювати власний ШІ. Більш популярним в середовищі розробників є другий підхід.

Аналіз останніх досліджень і публікацій

Штучний інтелект здатний трансформувати робочий процес у розробці ігор (геймдев, англ. games development, GameDev). Генеративний дизайн – нова технологія проєктування, яка дозволяє генерувати тривимірні моделі, предмети, людей, цілі всесвіти та переносити безпосередньо до 3D двигуна. Як інструментарій в генеративному дизайні використовують нейронні мережі. Нейронна мережа може навіть змінити спосіб створення 3D-контенту в іграх та генерувати кадри гри в режимі реального часу на основі зворотного зв’язку від користувача. Таку можливість надасть нова архітектура — варіаційні автоенкодери (VAE). Це генеративна нейромережа, яка вчиться зображати та вбудовувати об’єкти у потрібний простір. Прикладом такого застосування нейронної мережі є проєкт Enhancing Photorealism Enhancement [4], в якому графіка гри GTA V перетворюється на більш реалістичну на основі нейронного рендерингу, основою якого є навчання на цілій низці фотографій міських пейзажів.

Штучний інтелект у співпраці з художниками також здатен прискорити ідеєгенерацію при розробці концептів 3D-моделей, автоматизувати перенесення 2D-скетчів у 3D-моделі [5]. Так експерти NVIDIA презентували технологію Image2Car -інструмент доступний як розширення в AI Toy Box в NVIDIA Omniverse, який переносить модель із 2D в 3D і створює точну тривимірну модель автомобіля за фото, відтворюючи всі пропорції [6] та анонсувала інструмент Audio2Face на основі штучного інтелекту, який створює реалістичні вирази обличчя з аудіофайлу та панель Blender, яка спрямована на оптимізацію сцени.

Для створення ландшафтів найчастіше використовують автоматизований спосіб на основі алгоритму процедурної генерації, який створює шаблонний ландшафт. Для відтворення в такому шаблоні певного стилю, який схожий на Альпи чи Карпати в нагоді стає машинне навчання, яке дозволяє ШІ генерувати ландшафт за зразком побудованим художником [7].

Також ШІ незамінний при розробці режимів навчання, режимів PvE (Player vs Environment) і PvP (Player vs Player). Коли гравець розпочинає гру, наприклад World of Tanks, якщо він новачок, то потрапляє в режим навчання, де йому потрібно виконати певні практичні дії. З іншого боку, є противники та союзники, які чекають від гравця потрібних дій, виїжджають, стріляють та взаємодіють із гравцем. Вони допомагають навчити гравця базових механік. В процесі основної гри, поки досвіду у гравця небагато, він часто натрапляє на ботів-суперників, які сильніші

https://doi.org/10.32782/mathematical-modelling/2022-5-1-11

за навчальних ботів. Далі, окрім режиму PvP, багато гравців обирають режим PvE, тобто гру з середовищем, по суті гру проти ШІ. Основними підходами компанії Wargaming у розробці є Goal Oriented Action Planning, Behavior Tree, Utility System [8].

Мета дослідження

Метою дослідження є моделювання поведінки неігрових персонажів та програмна інтеграція її до ігрового рушія.

Основна частина

В якості рушія гри для тестування розробленої поведінки неігрових персонажів (англ. Non-Player Character, NPC) було обрано Unity – з одного боку, це інструмент для розробки відеоігор і застосунків, з іншого боку, рушій, на якому вони працюють. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL. Unity – кросплатформний рушій, який дозволяє переносити створені проекти на різні платформи для запуску (мобільні пристрої, різні операційні системи), також він має зручний комплексний інтерфейс. Ігрова логіка застосунку пишеться мовою C#.

Моделювання поведінки неігрових персонажів

Серед неігрових персонажів виділяють ботів – ворожих до гравця ШІ-персонажів, що наближаються за можливостями до ігрового персонажа, та мобів – ворожих до гравця «низькоінтелектуальних» ШІ-персонажів. Моби взаємодіють з гравцями у великих кількостях, гравець їх знищує заради очок досвіду, артефактів або проходження території. В будь-який конкретний момент часу гравцю протистоїть невелика кількість ботів. Боти складніші в програмуванні ніж моби та дружні до гравця неігрові персонажі, яких так і звуть NPC, відрізняючи від мобів та ботів.

Дерево поведінки – це формалізований підхід побудови поведінки NPC. Його особливість полягає в тому, що всі стани персонажа організовані у вигляді деревоподібної ієрархічної структури. Дерево поведінки містить у собі усі можливі стани, у яких може бути NPC [9, 10]. Коли у грі відбувається якась подія, ШІ перевіряє, в яких умовах знаходиться NPC, і перебирає всі стани у пошуках того, що підійде для нинішньої ситуації. Дерево поведінки відмінно підходить для того, щоб систематизувати стани NPC в іграх, в яких є безліч механік та геймплейних елементів. У ситуації, коли NPC бере участь у перестрілці, йому не потрібно буде шукати відповідну дію у гілці патрулювання. Такий підхід допомагає зробити поведінку NPC чуйною до ситуації в грі та забезпечує плавний перехід між різними станами. Замість визначення кінцевого набору станів персонажу визначається дерево вузлів з гілками з різною поведінкою. Дерева поведінки дозволяють зробити пріоритети дій простими: оскільки успішність дочірніх вузлів визначає те, як працює гілка, а отже, і дерево, поєднання складених «і» та «або» безпосередньо дає набір пріоритетних функцій, які нативно обробляють послідовності, переривання та резервні варіанти. Крім того така структуру досить легко читається та візуалізується – достатньо слідкувати за гілками за пріоритетом і на кожному вузлі перевіряти, чи успішна вона чи ні, щоб знати, чи можна продовжити цю гілку чи треба перейти до наступної. Але найбільшою перевагою дерев поведінки є те, що вони дуже модульні та масштабовані: видаляючи або додаючи гілку, фактично додаємо або видаляємо функцію для персонажа; так само досить просто скопіювати піддерево та вставити її в інше місце, щоб скопіювати та вставити частину поведінки. Деякі піддерева можуть бути навіть повністю самодостатніми автономними бітами ігрової логіки, які потім можна повторно імпортувати та повторно використовувати в дереві поведінки іншого персонажа, щоб швидко реалізувати фрагменти системи (наприклад, слідування за ціллю, патрулювання між набором точок).

Крім того, така модульність означає, що можна будувати дерево по частинах і мати стратегію поступового проектування. На відміну від кінцевих автоматів, де, за визначенням, стани

https://doi.org/10.32782/mathematical-modelling/2022-5-1-11

максимально ізольовані один від одного наскільки це можливо, дерева поведінки набагато більш гнучкі від витоку інформації між вузлами. Незважаючи на те, що кожен вузол виконується сам по собі та живе власним життям, досить часто є певний стан спільних даних, до якого можна отримати доступ із різних точок дерева, щоб мати глобальні (хоча й залежні від контексту) змінні.

Для інтеграції розробленого дерева поведінки були поєднані вже існуюча модель гра у стилі рпг та власні сцени. Як об'єкт тестування був обраний NPC-охоронець, який патрулює локацію та атакує монстрів, якщо ті знаходяться в заданому радіусі.

Реалізація дерева поведінки виконувалася в два етапи:

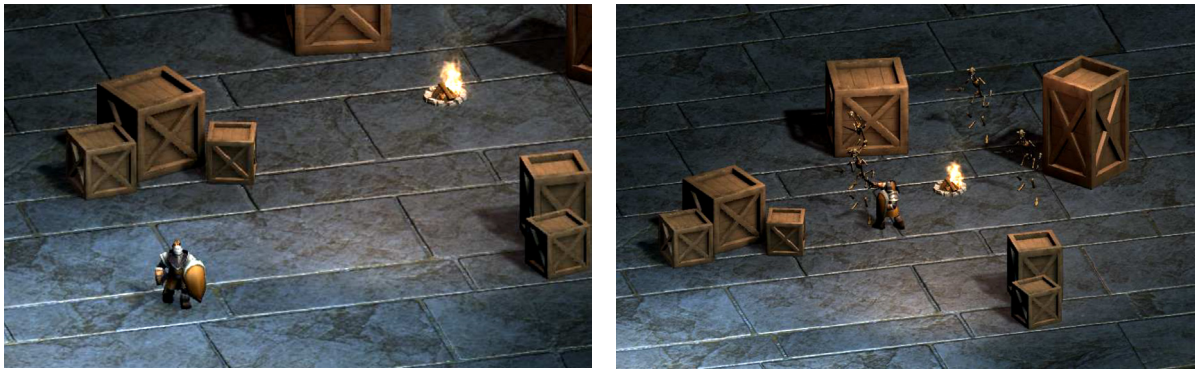
1. Визначення загальної архітектури, яка може використовуватися будь-яким деревом поведінки разом із кількома складеними вузлами.

2. Розробка конкретної структури та його вузлів дерева поведінки охоронця.

На першому етапі був створений атомарний базовий клас Node C#, який представляє окремий елемент у дереві та може мати доступ як до дочірніх, так і до батьківських елементів, має стан вузла, який використовує перелік і може зберігати, отримувати або очищати спільні дані. Другий основний об’єкт – клас Tree, який є MonoBehaviour і містить посилання на кореневий вузол, який сам рекурсивно містить усе дерево. Йому потрібно лише виконувати дві речі: після старту будувати дерево поведінки відповідно до визначеної функції установки та у відповідному методі за наявності вже побудованого дерева, оцінювати його стан постійно. Алгоритм патрулювання потребує точок шляху для використання, а також посилання на перетворення агента, який виконує цю дію. Поведінку NPC для виявлення ворогів поблизу та переміщення до визначеної цілі визначають два вузли дерева: вузол CheckEnemyInFOVRange, який повертає успіх, якщо є принаймні один ворог поблизу у полі видимості або помилку, та вузол TaskGoToTarget, який переміщує охоронця до знайденої цілі.

Коли охоронець наближається дуже близько до ворога, і цей ворог потрапляє в зону його атаки, охоронець повинен переключається в режим атаки і задавати певної шкоди ворогу. Для реалізації цих дій дерево поведінки має вузол перевірки для діапазону атаки та вузол завдання для фактичної атаки CheckEnemyInAttackRange, якщо він не знаходить ціль, він зазнає невдачі. Інакше, якщо ворог досить близько, він досягне успіху

Результати інтеграції розробленого дерева поведінки наведені на рис. 1.



а) б)

Рис. 1. Результат виконання сцени з тестування поведінки автоматичного виявлення та атакування ворогів: а) режим патрулювання; б) режим атаки на ворогів

NPC-охоронець спочатку патрулює місцевість за заданими точками, потім помічає скелетів та атакує їх. Після знищення всіх монстрів охоронець повертається в початкове місце та продовжує патрулювати.

Моделювання вибору найкращого шляху для NPC

Дії неігрового персонажу безпосередньо пов'язані з його завданням: це може бути удар у ближньому бою, зайняття більш вигідної позиції для стрілянини, допомога союзнику. Незалежно від завдання агент має дістатися місця виконання дії — для цього йому потрібна

https://doi.org/10.32782/mathematical-modelling/2022-5-1-11

інформація про навколишнє середовище. Зазвичай такі дані містяться в навігаційній сітці (NavMesh) – це особлива карта, де зазначено, де NPC можуть пересуватися. Поверх навігаційної сітки може розташовуватися система вузлів, яка застосовується у тому, щоб NPC коректно виконували сценарії поведінки.

Алгоритм A* [10] є одним із найпопулярніших алгоритмів пошуку, який часто використовується в різних галузях інформатики, зокрема і в задачах штучного інтелекту [11], завдяки своїй ефективності та автономному характеру. Він використовує структуру даних сітки для виконання певного обходу графа, щоб знайти найкоротший шлях між двома точками. Він сприймається як розширення алгоритму Едсгера Дейкстри [10]. Однак, на відміну від алгоритму Дейкстри, алгоритм A* використовує евристику (допоміжну функцію) для управління пошуком для досягнення кращої продуктивності.

Алгоритм відвідує вершини, які, швидше за все, ведуть найкоротшим шляхом до мети. Такі вершини визначаються за формулою (1):

$$F(x) = G(x) + H(x)\,, \tag{1}$$

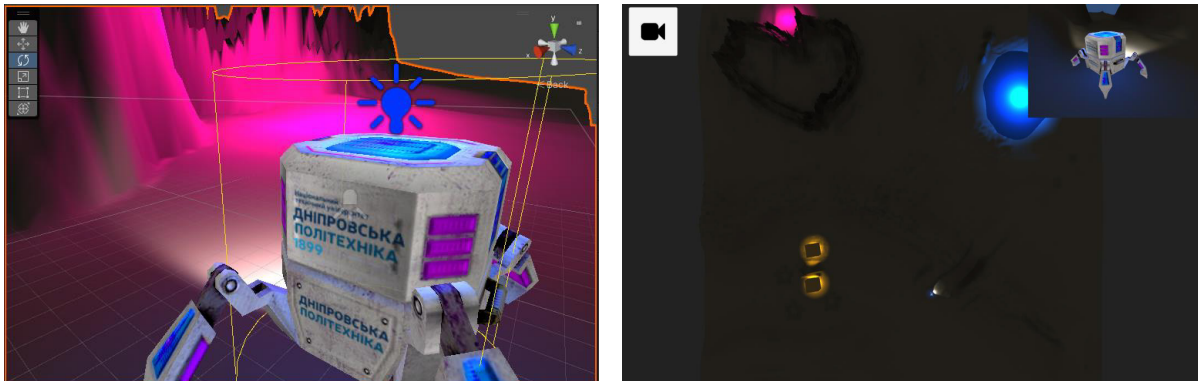
де $F(x)$ – мінімальна вартість переходу до наступної вершини, чим менше функція, то «ближче» вершина стоїть у черзі відвідування; $G(x)$ – це вартість шляху від початкової вершини і до будь-якої іншої; $H(x)$ – це евристичний показник вартості шляху від вершини x і до кінцевої вершини. Обчислюючи $F(x)$, за x береться сусідня вже відвідана вершина, тобто алгоритм обчислює вартість всього шляху у всіх сусідніх вершин і зупиняє вибір на тій, у якої мінімальна вартість.

У A* використовується сітка, де кожна клітинка є «прохідною» або «непрохідною». Щоб знайти шлях між двома точками, треба спочатку обчислити відстань між ними, тобто необхідно вказати, як будемо вимірювати відстань між двома довільними клітинками в сітці [6].

Переваги алгоритму: швидкий пошук на відкритих просторах, його універсальність. Єдиним серйозним недоліком є те, що всі вузли, що утворюють обчислений шлях, зберігаються в пам’яті, що може потребувати великої кількості пам’яті для обчислення маршруту.

Модель переміщення неігрового персонажу із застосуванням алгоритму A* була реалізована у невеликій грі, всі сцени якої мають різний інтерфейс та тестують інтелектуальну поведінку неігрового персонажу.

У першій сцені користувач може керувати марсоходом (пасфандером) курсором миші для досягнення рожевого світла (рис. 2, а). NPC автоматично слідує до курсора миші, обираючи самий ефективний маршрут згідно алгоритму A*. Для зручності гравця є камера, яка дозволяє оглядати сцену в декількох режимах: режим 1 – відображення двох камер: зверху та від третьої особи (рис. 2, б), режим 2 – відображення тільки виду зверху, режим 3 – зміна камери на вид від третьої особи. Перший режим є початковим та при зміні режимів, до нього неможливо повернутись. Коли натискається кнопка камери, чергується режим 2 та 3.



а) б)

Рис. 2. Тестування знаходження найкоротшого шляху до заданої точки: а) марсохід (пасфандер); б) режим камери першої сцени

https://doi.org/10.32782/mathematical-modelling/2022-5-1-11

У другій сцені тестується переміщення пасфандера з вибором маршруту переміщення між об'єктами для досягненні визначеної точки (одного з синіх ліхтарів) задля переключення на нову сцену (рис. 3).

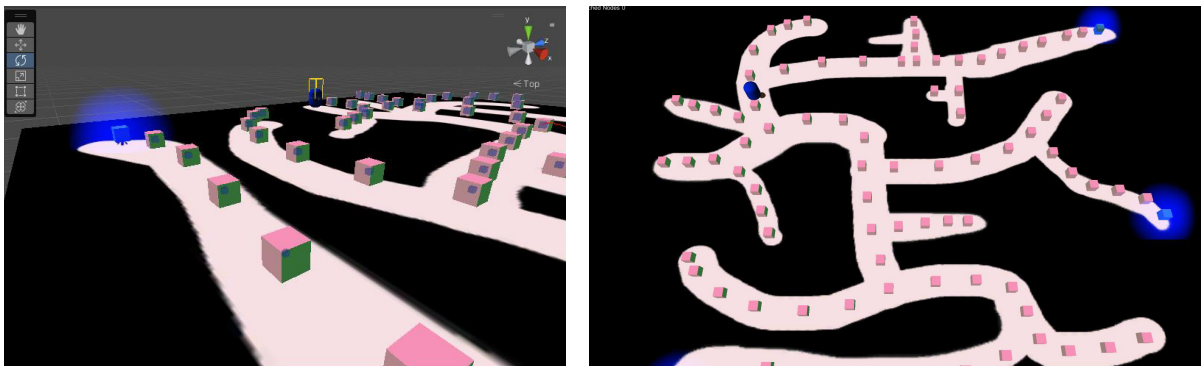


Рис. 3. Тестування знаходження найкоротшого шляху з вибором маршруту переміщення між об'єктами

Третя сцена – це кімната з перешкодами, яка візуалізує поведінку по зміні маршруту пасфандера при зіткненні з перешкодами. При досягненні голубого ліхтаря, йде переключення на нову сцену.

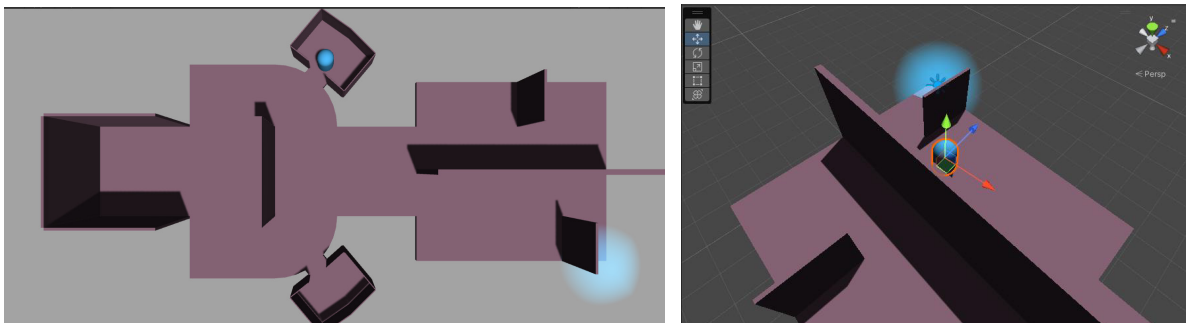


Рис. 4. Тестування поведінки по зміні маршруту при зіткненні з перешкодами

У четвертій сцені на полі з різними об'єктами можуть переміщуватись два марсоходи (рис. 5). Ця сцена наявно демонструє взаємодію двох об'єктів ігрового штучного інтелекту (при пошуку кращого маршруту) та заборону на переміщення одному з них у визначеній області. При досягненні червоного ліхтаря, йде переключення на нову сцену.

Рис. 5. Взаємодія двох об'єктів ігрового ШІ

https://doi.org/10.32782/mathematical-modelling/2022-5-1-11

П’ята сцена – це поле з різними об’єктами, які генеруються рандомним шляхом, по якому переміщується марсохід, обираючи максимально швидкий маршрут, обходячи ці об’єкти (рис. 6). Ця сцена наявно демонструє взаємодію об’єкта ігрового штучного інтелекту при пошуку кращого маршруту та великої кількості перешкод. При досягненні червоного ліхтаря, йде переключення на нову сцену.

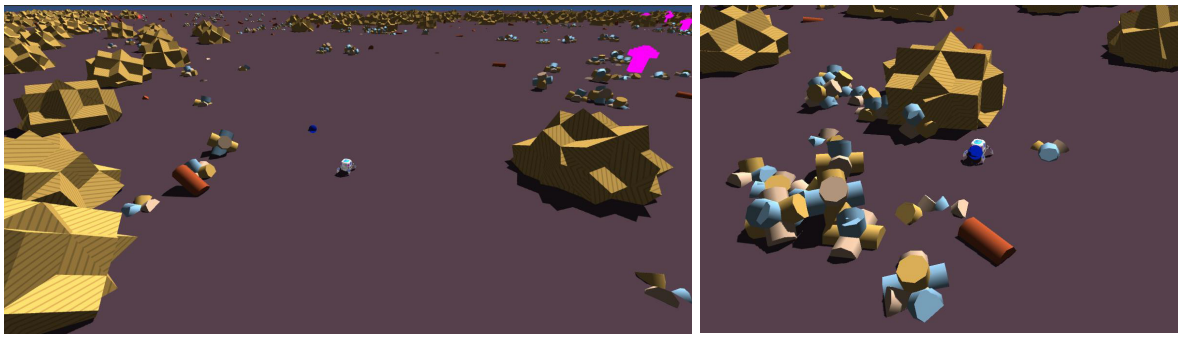


Рис. 6. Пошук максимально швидкого маршруту при великій кількості перешкод

Для демонстрації можливості використання пасфандінгу у 2D-просторі була створена шоста сцена гри – це двовимірний простір з ботом, який переміщується по ньому по раніше заданим дорогам (обираючи найкращій шлях, рис.7).

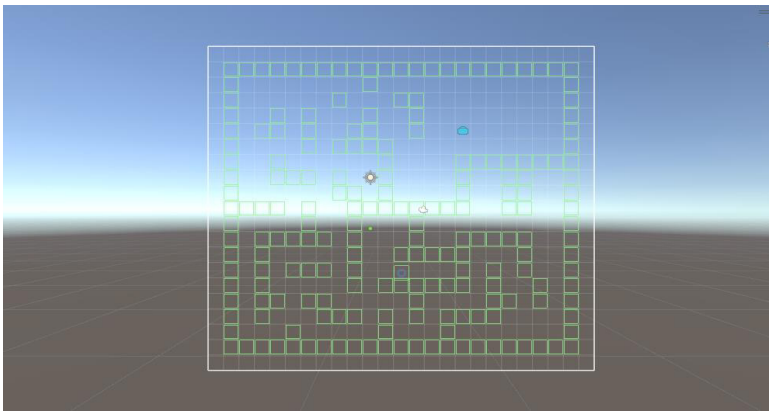


Рис. 7. Двовимірна візуалізація шляхів переміщення неігрового персонажу

Висновки

У роботі було розроблено модель поведінки неігрового персонажу, яка була протестована шляхом інтеграції до вже існуючої гри. Тестування підтвердило коректну роботу алгоритму поведінки об’єктів, їх взаємодії один з одним, відпрацювання різних кейсів оточуючого середовища та відсутність візуальних дефектів роботи алгоритму.

Розроблена міні-гра на основі існуючих моделей неігрових персонажів підтвердила ефективність застосування A* алгоритму пошуку найкоротшого шляху в різних ігрових умовах. Напрямом подальших досліджень є розширення ігрового застосунку до стабільної версії з чітким оповіданням та сюжетом, доопрацювання загального ігрового та дизайну кожного рівня для покращення взаємодії з користувачем та створення власних 3D моделей та інших ігрових ресурсів (асетів) замість безкоштовних.

Список використаної літератури

1. Google for Games. Beyond 2021: Where does gaming go next? Режим доступу:[https://games.withgoogle.com/reports/beyondreport/]

https://doi.org/10.32782/mathematical-modelling/2022-5-1-11

2. Batchelor J. GamesIndustry.biz presents… The Year In Numbers 2022. https://www.gamesindustry.biz/gamesindustrybiz-presents-the-year-in-numbers-2022.
3. Як знайти роботу в геймдеві під час війни? Історії українських спеціалістів, 2022. Режим доступу: [https://gamedev.dou.ua/articles/how-to-get-a-job-in-gamedev-during-the-war-1/?from=recent]
4. Richter S.R., AlHaija H.A., Koltun V. Enhancing Photorealism Enhancement. arXiv:2105.04619 2021. https://doi.org/10.48550/arXiv.2105.04619.
5. Максим Маковский. Як штучний інтелект може трансформувати робочий процес 3D-художників у геймдеві. 2022. Режим доступу: [https://gamedev.dou.ua/blogs/ai-for-3D-artists-in-gamedev/]
6. GANverse3D Extension. Режим доступу: [https://docs.omniverse.nvidia.com/prod-_extensions/prod_extensions/ext_ganverse3d.html]
7. Noor S., Julian T., Mark J. N. Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer. ISBN 978-3-319-42714-0. 237p.
8. Harbuzova A. Яка користь штучного інтелекту у розробці… 2021. Режим доступу: [https://gamedev.dou.ua/articles/wargaming-developer-about-creating-ai-for-games/]
9. Behaviour Trees. Режим доступу: [https://learn.unity.com/project/behaviour-trees]
10. Heineman G.T., Pollice G., Selkow S. Algorithms in a Nutshell. Second Edition. Beijing, Boston, Farnham, Sebastopol, Tokyo. O'Reilly Media, Inc. 2016. 390p.
11. Singhal A. A* Algorithm Example in AI. Gate Vidyalay. Режим доступу: [https://www.gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/]

References

1. Google for Games. Beyond 2021: Where does gaming go next? [https://games.withgoogle.com/reports/beyondreport/]
2. Batchelor J. GamesIndustry.biz presents… The Year In Numbers 2022. https://www.gamesindustry.biz/gamesindustrybiz-presents-the-year-in-numbers-2022
3. Yak znaity robotu v heimdevi pid chas viiny? Istorii ukrainskykh spetsialistiv, (2022). [https://gamedev.dou.ua/articles/how-to-get-a-job-in-gamedev-during-the-war-1/?from=recent]
4. Richter S.R., AlHaija H.A., Koltun V. Enhancing Photorealism Enhancement. arXiv:2105.04619 2021. https://doi.org/10.48550/arXiv.2105.04619
5. Maksym Makovskyi. Yak shtuchnyi intelekt mozhe transformuvaty robochyi protses 3D-khudozhnykiv u heimdevi. (2022). [https://gamedev.dou.ua/blogs/ai-for-3D-artists-in-gamedev/]
6. GANverse3D Extension. [https://docs.omniverse.nvidia.com/prod-_extensions/prod_extensions/ext_ganverse3d.html]
7. Noor, S., Julian, T., & Mark, J. N. Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer. ISBN 978-3-319-42714-0.
8. Harbuzova, A. (2021). Yaka koryst shtuchnoho intelektu u rozrobtsi…. [https://gamedev.dou.ua/articles/wargaming-developer-about-creating-ai-for-games/]
9. Behaviour Trees. [https://learn.unity.com/project/behaviour-trees]
10. Heineman, G.T., Pollice, G., & Selkow, S. (2016). Algorithms in a Nutshell. Second Edition. Beijing, Boston, Farnham, Sebastopol, Tokyo. O'Reilly Media, Inc.
11. Singhal, A. A* Algorithm Example in AI. Gate Vidyalay. [https://www. gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/]

Соколова Наталя Олегівна, к.т.н., доц. кафедри Інформаційних технологій та комп'ютерної інженерії Національного технічного університету «Дніпровська політехніка»; e-mail: n.olegowna@gmail.com ORCID: 0000-0003-2493-3553

Гнатушенко Володимир Володимирович, д.т.н., проф., завідувач кафедри Інформаційних технологій та комп'ютерної інженерії Національного технічного університету «Дніпровська політехніка»; e-mail: vvgnat@ukr.net ORCID: 0000-0003-3140-3788

Міщенко Микита Сергійович, бакалавр Національного технічного університету «Дніпровська політехніка»; e-mail: mishchenko.my.s@nmu.one

Атаманчук Олександра Анатоліївна, бакалавр Національного технічного університету «Дніпровська політехніка»; e-mail: atamanchuk.o.a@nmu.one

https://doi.org/10.32782/mathematical-modelling/2022-5-1-11