УДК 004.358
DOI https://doi.org/10.32782/2521-6643-2025-1-69.12

Ульяновська Ю. В., кандидат технічних наук, доцент,
завідувач кафедри комп'ютерних наук та інженерії
програмного забезпечення
Університету митної справи та фінансів
ORCID: 0000-0001-5945-5251

Рудянова Т. М., кандидат фізико-математичних наук, доцент,
доцент кафедри комп'ютерних наук та інженерії
програмного забезпечення
Університету митної справи та фінансів
ORCID: 0000-0002-2750-6031

Рябоволенко Е. А., викладач кафедри комп'ютерних наук
та інженерії програмного забезпечення
Університету митної справи та фінансів
ORCID: 0009-0005-7713-3287

Фесенко А. Р., магістр
Університету митної справи та фінансів

АНАЛІЗ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ ДЛЯ ПРОГРАМОВАНИХ ОБ'ЄКТІВ У ЦИФРОВИХ ІГРОВИХ ПРОСТОРАХ

Дослідження присвячене розробці методів пошуку оптимального шляху для програмованих об'єктів у віртуальному ігровому середовищі. Розглядаються алгоритмічні підходи щодо визначення найкоротшого шляху та досліджується застосування алгоритму A у рушію Unity для реалізації ефективного пошуку найкоротших відстанів між початковою та цільовою точками переміщення ігрового персонажа. Особливу увагу приділено аналізу ключових параметрів алгоритму – gCost, hCost та fCost, які визначають евристичну оцінку вартості переміщення та дозволяють забезпечити оптимальність маршрутів у контексті складної структури ігрового простору. У дослідженні також враховуються особливості віртуального середовища, такі як наявність перешкод, змінні умови навколишнього середовища та динаміка ігрового процесу, що впливають на прийняття рішень системою навігації.*

Особливу увагу в процесі програмної реалізації алгоритму A приділено механізму динамічного оновлення множини сусідніх вузлів для кожної вершини графа навігації. Такий підхід забезпечує гнучкість алгоритму в умовах змінного середовища, дозволяючи оперативно реагувати на появу нових перешкод або зміну топології ігрової сцени. Завдяки цьому вдалося підвищити адаптивність системи пошуку шляху до умов реального часу, що є критично важливим для сучасних ігрових застосунків.*

Результати дослідження підтверджують високу ефективність алгоритму A у вирішенні задач пошуку оптимального шляху, що зумовлює його актуальність для широкого спектра застосувань у сфері комп'ютерних технологій, зокрема в ігровому дизайні, віртуальних симуляціях та системах автономної навігації в робототехніці. Інтеграція алгоритму в середовище розробки Unity не лише розширює функціональні можливості цього рушія, але й створює передумови для подальшого вдосконалення інтелектуальних компонентів ігрових та симуляційних систем. Представлене дослідження підкреслює важливість впровадження адаптивних алгоритмів навігації та їхню роль у розвитку інноваційних технологій, результати можуть бути використані для підвищення адаптивності та продуктивності навігаційних систем у комп'ютерних іграх.*

Ключові слова: *A*, Unity, IDA*, SMA*, пошук оптимального шляху, графи.*

Ulianovska Yu. V., Rudianova T. M., Riabovolenko E. A., Fesenko A. R. Analysis and implementation of algorithms for finding the optimal path for programmable objects in digital game spaces

The study is devoted to developing methods for finding the optimal path for programmable objects in a virtual game environment. Algorithmic approaches to determining the shortest path are considered, and the application of the A algorithm in the Unity engine is investigated for implementing an effective search for the shortest distances between the initial and target points of movement of a game character. Particular attention is paid to the analysis of the key parameters of the algorithm – gCost, hCost, and fCost, which determine the heuristic estimate of the cost of movement and ensure the optimality of routes in the*

© Ю. В. Ульяновська, Т. М. Рудянова, Е. А. Рябоволенко, А. Р. Фесенко, 2025

context of the complex structure of the game space. The study also considers the features of the virtual environment, such as the presence of obstacles, variable environmental conditions, and the dynamics of the game process, which affect decision-making by the navigation system.

In the process of software implementation of the A algorithm, special attention is paid to the mechanism of dynamic updating of the set of neighboring nodes for each vertex of the navigation graph. This approach provides flexibility for the algorithm in a changing environment, allowing it to quickly respond to the appearance of new obstacles or changes in the topology of the game scene. This allowed us to increase the adaptability of the pathfinding system to real-time conditions, which is critically important for modern gaming applications.*

The results of the study confirm the high efficiency of the A algorithm in solving the problems of finding the optimal path, which makes it relevant for a wide range of applications in the field of computer technology, in particular in game design, virtual simulations, and autonomous navigation systems in robotics. The integration of the algorithm into the Unity development environment not only expands the functionality of this engine but also creates the prerequisites for further improvement of the intelligent components of gaming and simulation systems. The presented study emphasizes the importance of implementing adaptive navigation algorithms and their role in the development of innovative technologies. The results can be used to increase the adaptability and performance of navigation systems in computer games.*

Key words: *A*, Unity, IDA*, SMA*, optimal path finding, graphs.*

Постановка проблеми. Пошук оптимального шляху є однією з фундаментальних задач в галузі комп'ютерних наук, яка має широке практичне застосування у таких сферах людської діяльності, як штучний інтелект, робототехніка, комп'ютерні ігри, навігаційні системи, логістика, картографія та інших суміжних напрямках. Актуальність задачі пошуку шляху в програмованих об'єктах ігрового середовища обумовлюється кількома ключовими чинниками. По-перше, зростаюча складність цифрових середовищ та зростання вимог до інтелектуальної поведінки програмних об'єктів обумовлюють потребу в ефективних алгоритмах для визначення найкоротших або найменш витратних маршрутів у заданому просторі. Сучасні комп'ютерні ігри характеризуються високим ступенем складності та динамічності віртуальних просторів, що потребує використання адаптивних і ефективних алгоритмів навігації. По-друге, підвищення вимог до реалістичності поведінки ігрових агентів, зокрема в аспектах автономного пересування, ухилення від перешкод і взаємодії з динамічними об'єктами, стимулює розвиток інтелектуальних систем керування. Крім того, сфера застосування алгоритмів пошуку шляху значно розширюється поза межами індустрії відеоігор. Такі алгоритми знаходять широке застосування в системах автономної навігації мобільних роботів, у транспортній логістиці, у візуалізації віртуальних середовищ для навчальних і тренувальних симуляцій, а також у геоінформаційних системах. Це підкреслює міждисциплінарну важливість теми дослідження та обґрунтовує потребу у створенні універсальних та продуктивних інструментів пошуку оптимальних маршрутів.

Сучасні ігри ставлять перед персонажами складні задачі з навігації в віртуальних світах, де їм необхідно швидко та ефективно знаходити шлях до цілей, обминаючи перешкоди та реагуючи на зміни в середовищі. Гравці очікують, що персонажі в іграх будуть вести себе максимально реалістично, що робить алгоритми пошуку шляху незамінними інструментами для розробників. Забезпечення реалістичної та адаптивної навігації персонажів є важливим аспектом геймдизайну, що безпосередньо впливає на якість користувацького інтерфейсу. В умовах обмежених ресурсів мобільних пристроїв та інших платформ виникає потреба в оптимізації алгоритмів пошуку шляху для забезпечення плавної роботи ігрового процесу.

Аналіз останніх досліджень і публікацій. Проблематика пошуку оптимального шляху в середовищах з обмеженнями активно досліджується впродовж останніх десятиліть, зокрема у контексті штучного інтелекту, робототехніки та комп'ютерної графіки. Пошук оптимального маршруту (англ. Pathfinding) – це складний процес визначення найбільш вигідного шляху між двома місцями гри. На перший погляд може здаватися, що це доволі проста задача, але вона ускладнюється, коли треба врахувати такі фактори, як наявність перешкод, складність території або час, необхідний для пересування. Відтак з'ясовується, що визначення найбільш вигідного шляху є більш складним завданням, ніж просто прокладення лінії між двома точками [1], [2].

Здається, що існує багато алгоритмів пошуку шляху для вирішення одного й того самого завдання – визначення найкоротшого маршруту між двома точками на карті, проте, методи, якими вони досягають цієї мети, можуть значно відрізнятися. З плином часу деякі алгоритми оптимізувались та стали кращими за своїх попередників з точки зору швидкості та ефективності. Слід також враховувати, що не завжди найвища ефективність пошуку є пріоритетом. В контексті відеоігор, особливо важливим стає реалістичне моделювання поведінки персонажів, керованих комп'ютером (NPC, англ. Non-Player Character). Розробники іноді віддають перевагу поведінці NPC, яка наближена до реалістичної замість того, щоб просто дотримуватись найкоротшого маршруту. Це означає, що вони можуть обирати алгоритми пошуку шляху, які не обов'язково є найефективнішими в плані використання ресурсів, але забезпечують більш природну поведінку персонажів [2], [3]. Таким чином, у сфері розробки ігор підбір алгоритму пошуку шляху може базуватися не лише на технічних характеристиках, а й на прагненні до створення більш занурюваного та реалістичного ігрового досвіду. Багато відомих ігор розробляють унікальні версії цих алгоритмів, модифікуючи та адаптуючи

існуючі рішення під свої специфічні потреби. Проведемо аналіз найпопулярніших алгоритмів пошуку найкоротшого шляху та визначмо особливості їхнього застосування в контексті відеоігор.

Пошук у ширину (BFS) рівномірно досліджує всі напрямки. Це надзвичайно корисний алгоритм, який використовується не тільки для знаходження шляхів, а й для генерації процедурних карт, пошуку шляхів за допомогою поля потоків, карт відстаней та іншого аналізу карт [4]. Алгоритм пошуку у ширину (BFS) є одним з основних інструментів для аналізу та навігації по графам. Він починає свою роботу з визначеного вузла, який називається кореневим, і поступово просувається через граф, обробляючи кожен вузол на однаковій відстані від кореня перед переходом до наступного рівня. Цей методичний підхід дозволяє BFS досліджувати граф шар за шаром, що робить його ідеальним для задач, де потрібно забезпечити рівномірне та повне охоплення простору графа. BFS використовує структуру даних у вигляді черги для зберігання вузлів, які потрібно обробити. Коли вузол обробляється, всі його сусідні вузли, які ще не були відвідані, додаються до черги. Ця особливість робить BFS особливо корисним для знаходження найкоротших шляхів у графах, де всі ребра мають однакову вагу, оскільки він гарантовано знайде найкоротший шлях від кореневого вузла до будь-якого іншого вузла [2]. Однак, BFS має свої обмеження, зокрема, він може бути ресурсномістким у великих графах, оскільки потребує зберігання всіх вузлів на поточному рівні перед переходом до наступного. У сфері ігрової індустрії BFS може використовуватися не тільки для пошуку шляхів, але й для генерації карт, створення полів потоків, карт відстаней та виконання інших видів аналізу карт, що робить його універсальним та цінним інструментом у руках геймдевелоперів [5]. Прикладами застосування алгоритму BFS є такі ігри, як «The Legend of Zelda: Breath of the Wild», «Diablo II», «Minecraft».

Алгоритм Дейкстри, також відомий як пошук із єдиною вартістю, дозволяє нам визначати пріоритети шляхів для дослідження. Видатний учений у сфері інформаційних технологій Едсгер Дейкстра у 1959 році представив свій алгоритм, який пізніше був названий на його честь. Цей алгоритм, що відноситься до класу жадібних алгоритмів, здатен знаходити найкоротші шляхи від однієї обраної точки графу до усіх інших точок, а також може бути адаптований для знаходження найкоротшого шляху між будь-якою парою точок [5]. Замість дослідження всіх можливих шляхів однаково, він віддає перевагу шляхам з меншою вартістю. Протягом роботи алгоритм зберігає інформацію про найкоротші відстані від початкової точки до кожної з інших точок, при цьому дані оновлюються кожного разу, коли знаходиться більш короткий шлях. Алгоритм завершується, коли всі вузли були «відвідані» та найкоротші шляхи до них визначені, надаючи на виході повний набір найкоротших шляхів від стартового вузла до усіх інших точок графу. Базова версія алгоритму Дейкстри вимагає O(V2) операцій, де V – кількість вершин у графі [6]. На старті алгоритму дистанції ініціалізуються дуже великим числом, що перевищує будь-яку можливу довжину шляху в графі, а масив відміток заповнюється нулями. Дистанція до початкової вершини встановлюється як нуль, і розпочинається основний цикл алгоритму. У кожній ітерації циклу алгоритм вибирає вершину з найменшою дистанцією, що ще не була оброблена, при цьому прапорець рівний нулю. Встановлюючи прапорець в одиницю, він позначає цю вершину як оброблену та переглядає всі її сусідні вершини. Якщо знаходиться більш короткий шлях до сусідньої вершини через поточну вершину, алгоритм оновлює дистанцію до цієї сусідньої вершини. Процес триває до тих пір, поки всі вершини не будуть оброблені – прапорець кожної вершини стане рівним 1. Прикладами застосування алгоритму Дейкстри є ігри: «Pac-Man», «Diablo III», «Civilization».

Одним із найпоширеніших алгоритмів, що використовуються для вирішення таких задач, є алгоритм A*, відомий також як «A star», який поєднує ефективність методу Дейкстри з евристичною оцінкою напрямку пошуку та широко застосовується для визначення найефективнішого маршруту між двома точками у графі з позитивною вагою кожного ребра. У науковій літературі [6] описано переваги алгоритму A* при роботі у статичних та частково динамічних середовищах. У роботах [7], [8] підкреслюється застосовність A* до задач планування руху у графових моделях середовища.

Значна кількість сучасних досліджень присвячена модифікаціям базового алгоритму A*, таким як Hierarchical A*, Jump Point Search, Theta* тощо, що спрямовані на зменшення обчислювальної складності та часу пошуку. У публікаціях [9] розглядається інтеграція таких алгоритмів в ігрові рушії, включаючи Unity та Unreal Engine, а також практичні аспекти реалізації інтелектуальних агентів.

Окрему нішу займають роботи, присвячені застосуванню A* в динамічних ігрових середовищах. Зокрема, у роботах [10] розглядається підхід D* (Dynamic A*), який дозволяє адаптуватися до змін у структурі карти під час виконання.

У середовищі Unity дослідники та практики впроваджують власні системи навігаційної сітки – NavMesh [11], однак потреба в користувацькій реалізації A* з гнучким управлінням вузлами та евристикою залишається актуальною, особливо у випадках, коли стандартні інструменти не задовольняють вимогам розробника.

Таким чином, незважаючи на наявність численних рішень і оптимізацій, завдання ефективної реалізації алгоритму A* у середовищі Unity, з підтримкою оновлення структури графа в режимі реального часу, потребує подальших досліджень. Саме на цю проблематику орієнтовано представлене дослідження.

Особливість алгоритму A* полягає у використанні спеціальної допоміжної функції, так званої евристики, яка спрямовує пошук у більш перспективні напрямки та допомагає зменшити час, необхідний для

знаходження рішення [12]. Цей алгоритм класифікує вузли графу на три категорії: невідомі вузли, відомі вузли та повністю досліджені вузли. Спочатку, усі вузли, за винятком стартового, є невідомими. Відомі вузли, зі своїм потенційним шляхом, зберігаються у списку з пріоритетом, де з кожним кроком вибираються найбільш обнадійливі вузли для подальшого дослідження. Такий підхід дозволяє оптимізувати швидкість виконання алгоритму. Коли вузол повністю досліджений, його сусідні вузли додаються до списку відомих вузлів, а сам вузол переміщається до списку повністю досліджених. Якщо сусідні вузли вже досліджені або є у списку відомих, алгоритм оновлює їхній статус, якщо через поточний вузол знайдений коротший шлях. Процес пошуку продовжується, поки кінцева вершина не опиниться у списку повністю досліджених вузлів, або ж поки список відомих вузлів не спорожніє, що свідчить про відсутність розв'язку. Шлях відновлюється у зворотньому порядку від кінцевої до стартової вершини, але може бути легко перевернутий для отримання шляху у правильному напрямку [2].

Припустима евристика оцінює вартість досягнення цільового вузла таким чином, що фактична вартість не перевищує оцінену. Це означає, що вона завжди дає нижню межу вартості шляху. Якщо евристика є припустимою, але не монотонною, то можуть виникати ситуації, коли доведеться повторно досліджувати деякі вузли. Монотонна евристика, крім умови непереоцінення вартості, також повинна задовольняти нерівність трикутника. Це означає, що оцінка вартості від поточного вузла до цілі завжди менша або рівна сумі вартості переходу до сусіднього вузла та оцінки вартості від цього сусіднього вузла до цілі. Швидкість виконання алгоритму значно залежить від точності евристичної функції та ефективності реалізації структур даних для зберігання відомих та повністю досліджених вузлів. Найбільшим недоліком алгоритму A* є його вимоги до використання пам'яті, оскільки потрібно зберігати велику кількість вузлів, що може бути непридатним для задач з великою кількістю можливих станів [2]. Прикладами використання алгоритму A* в іграх є: «StarCraft», «Warcraft III», «Half-Life».

Алгоритм Беллмана-Форда – це алгоритм пошуку найкоротшого шляху в зваженому графі, де вага дуг може бути як позитивною, так і негативною. Він названий на честь своїх розробників Річарда Беллмана та Лестера Форда. Цей алгоритм став фундаментальним у теорії графів та використовується у багатьох областях: від маршрутизації в мережах до розподілу ресурсів у логістичних системах [13]. Алгоритм має ряд переваг: враховує негативні ваги, може виявити наявність циклів з негативною вагою в графі, що є важливою властивістю при аналізі графів. До недоліків можна віднести:

– високу часову складність, алгоритм має часову складність O(V*E), де V – кількість вершин, а E – кількість ребер у графі, що робить алгоритм відносно повільним, особливо для графів з великою кількістю вершин і ребер;

– алгоритм є неефективним для густих графів, у густих графах, де кількість ребер наближається до V^2, алгоритм може бути неефективним через високу часову складність;

– працює повільніше за інші алгоритми для специфічних задач.

Алгоритм Беллмана-Форда застосовують у комп'ютерних іграх для різних цілей, враховуючи його здатність працювати з графами, які містять ребра з негативною вагою, та виявляти цикли з негативною вагою [2]. В іграх з великими відкритими світами NPC можуть використовувати алгоритм Беллмана-Форда для знаходження найкоротшого шляху до персонажів, подорожей і інші.

У стратегічних іграх алгоритм може бути використаний для оптимізації розподілу ресурсів та логістичних маршрутів, особливо коли враховуються витрати та ризики переміщення через різні території. У процесі розробки гри алгоритм може допомогти розробникам в аналізі та балансуванні геймплею, виявляючи потенційно нескінченні цикли або інші аномалії, які можуть вплинути на геймплей.

Хоча основна ціль цих алгоритмів однакова – знайти найкоротший шлях між двома точками на карті, методи, за допомогою яких кожен з них досягає цієї мети, різняться. Це дає розробникам можливість вибрати алгоритм, що найбільш адаптований під конкретні потреби та особливості їхніх ігор.

У той же час, актуальним напрямом залишаються дослідження, спрямовані на підвищення продуктивності алгоритму в умовах реального часу, адаптацію до зміни топології середовища, а також використання у графічних рушіях для комп'ютерних ігор.

Мета дослідження. Метою даного дослідження є програмна реалізація алгоритму оптимізації пошуку найкоротшого шляху для підвищення продуктивності та точності навігації в комп'ютерних іграх в умовах змінної структури сцени.

Методи дослідження. У процесі дослідження використано методи алгоритмічного аналізу, моделювання та програмної реалізації з акцентом на практичну імплементацію в ігровому рушії Unity. В роботі використовувалися методи та технології розробки ігрових мовою програмування C#. Як основний підхід до побудови навігаційної системи було обрано алгоритм A*, який базується на евристичному пошуку найкоротшого шляху у графі.

Виклад основного матеріалу. Алгоритм A* є методом пошуку найкоротшого шляху в зваженому графі, що забезпечує знаходження оптимального маршруту від початкової вершини до кінцевої. Його робота базується на об'єднанні фактичної вартості вже пройденого шляху (g) та евристичної оцінки (h), яка наближено визначає вартість переміщення до цільової вершини. На кожному кроці алгоритм обирає для

подальшого розгляду вершину з мінімальним значенням сумарної оцінки (f = g + h). Параметр g відображає сукупну вартість переміщення від початкової точки до поточної, враховуючи всі пройдені клітинки. Параметр h є евристичною оцінкою вартості пересування від поточного положення до кінцевої вершини і використовується для наближеного прогнозування залишкової вартості шляху. Загальна оцінка f обчислюється як сума значень g та h, що дозволяє алгоритму ефективно спрямовувати пошук у напрямку до мети.

Алгоритм здійснює вибір напряму пошуку, орієнтуючись на мінімальне значення функції f: на кожному етапі він обирає клітинку з найменшим значенням f та переходить у неї. Процес пошуку триває доти, доки не буде досягнута цільова клітинка (рисунок 1).

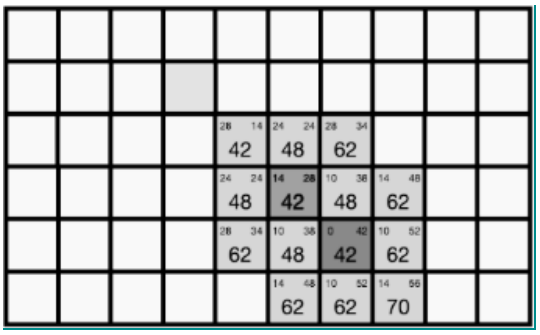

Рис. 1. Ілюстрація алгоритму пошуку шляху A*

Алгоритм A* у процесі виконання зберігає всі згенеровані вузли в оперативній пам'яті, що дозволяє ефективно використовувати наявну інформацію для побудови найкоротшого шляху. Однак така стратегія зумовлює високу пам'яттєву складність алгоритму, оскільки зі збільшенням розміру простору пошуку зростає і кількість вузлів, які необхідно зберігати. Це є одним із суттєвих недоліків A*, особливо у випадках роботи з великими графами. Для подолання цієї обмеженості було запропоновано алгоритм ітеративного заглиблення A* (IDA*), який поєднує переваги евристичного пошуку A* та економного за пам'яттю пошуку в глибину. IDA* дозволяє знаходити оптимальні маршрути між початковим і кінцевим станами в графах або деревоподібних структурах, суттєво зменшуючи витрати пам'яті. На відміну від A*, алгоритм IDA* підтримує у пам'яті лише інформацію про поточний шлях і його вартість, що робить його придатним для вирішення задач у великих просторах станів із обмеженими ресурсами пам'яті [14].

З погляду просторової складності алгоритм ітеративного заглиблення A* (IDA*) є більш ефективним порівняно з класичним алгоритмом A*. Під час виконання алгоритму A* в оперативній пам'яті зберігається уся область пошуку, що при роботі з великими просторами станів призводить до істотних витрат пам'яттєвих ресурсів. Натомість алгоритм IDA* оптимізує використання пам'яті шляхом збереження лише поточного вузла та відповідної вартості шляху, що дозволяє уникнути експоненційного зростання обсягу пам'яті при розширенні простору пошуку. Такий підхід забезпечує зменшення просторових витрат без істотного погіршення якості пошуку оптимального маршруту (рисунок 2).

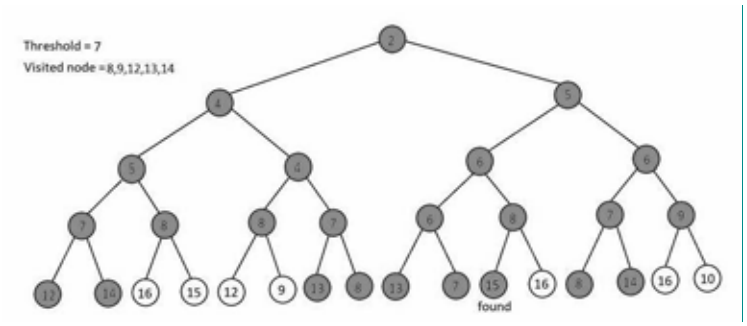

Рис. 2. Алгоритм IDA*

Алгоритм ітеративного заглиблення A* (IDA*) здійснює дослідження простору станів за допомогою пошуку в глибину з обмеженням за вартістю. На початковому етапі задається порогове значення, яке дорівнює значенню евристичної оцінки f для початкового вузла, де f(n)=g(n)+h(n). Під час пошуку алгоритм розширює лише ті вузли, для яких виконується умова f(n)≤thresholdf(n). Якщо у процесі пошуку знаходиться цільовий вузол, алгоритм завершує роботу. У випадку, коли для певного вузла значення f(n) перевищує поточне порогове значення, алгоритм не розширює цей вузол, але фіксує мінімальне з перевищених значень.

Після завершення ітерації нове порогове значення встановлюється рівним мінімальному зафіксованому f(n), яке перевищило попередній поріг. Цей ітеративний процес повторюється до тих пір, поки не буде знайдено оптимальне рішення, або доки простір пошуку не буде повністю вичерпано.

Алгоритм Weighted A* (вагований A*) являє собою модифікацію класичного алгоритму A* для задачі пошуку шляху в графах, де окремі ребра або дії мають різні вагові коефіцієнти або вартості. На відміну від базового варіанта A*, який зазвичай передбачає однакову вагу всіх ребер і спрямований на знаходження найкоротшого шляху, у вагованому варіанті кожне ребро або дія може мати індивідуальну вагу. Це дозволяє точніше моделювати, наприклад, різну вартість переміщення в залежності від напрямку або складність виконання певних дій.

Основна концепція роботи алгоритму Weighted A* зберігає структуру класичного A*, проте обчислення функції оцінки f(n) доповнюється урахуванням вагових коефіцієнтів. У загальному вигляді функція f(n) визначається за формулою:

$$f(n) = g(n) + w \cdot h(n),$$

де g(n) – фактична вартість шляху від початкового вузла до поточного, h(n) – евристична оцінка вартості шляху від поточного вузла до цільового, а w –коефіцієнт ваги, що відображає вартість переходу через відповідне ребро або виконання певної дії [15].

Алгоритм Weighted A* є ефективним засобом для розв'язання задач пошуку шляху в умовах, коли вартість переходів між вузлами варіюється, і виникає необхідність врахування цих відмінностей при прийнятті рішень щодо оптимального вибору маршруту.

Алгоритм D* (динамічний A*) є модифікацією класичного підходу до пошуку шляху, який використовує концепцію сенсорного сприйняття навколишнього середовища та здатний адаптуватися до змін, зокрема до появи динамічних перешкод, шляхом оновлення ваг ребер у реальному часі. На кожному етапі роботи D* розв'язує задачу знаходження найкоротшого шляху від поточного положення до стартового вузла, припускаючи, що вузли з невідомим статусом прохідності є доступними для пересування. Для обробки змін у вартості переходів алгоритм підтримує спеціальний список вузлів (OPEN list), який використовується для поширення оновленої інформації. Вузли, що беруть участь у процесі, класифікуються за кількома станами: NEW, OPEN, CLOSED, RAISE та LOWER. Робота алгоритму має ітеративний характер: на кожному кроці обирається вершина зі списку OPEN для оцінювання. Пошук у D* здійснюється у зворотному напрямку – від цільового вузла до початкового. Кожна розширювана вершина містить зворотний покажчик на наступну вершину маршруту до цілі та має доступ до точної вартості шляху до кінцевого вузла.

Алгоритм LPA* (Lifelong Planning A*) є розвитком класичного алгоритму A*, орієнтованим на довготривале планування шляхів у динамічних середовищах (рисунок 3). Ця інкрементна модифікація A* здатна адаптуватися до змін у структурі графа без необхідності повного перерахунку шляху. У процесі пошуку алгоритм оновлює значення функції ggg, що представляє собою відстань від стартового вузла, використовуючи дані, отримані під час попередніх обчислень, і за потреби коригує їх. Завдяки використанню накопиченої інформації LPA* суттєво зменшує кількість вершин, які потрібно обробити у порівнянні з класичним A*, оновлюючи лише ті значення ggg, що критично впливають на формування найкоротшого шляху. Це забезпечує значну економію обчислювальних ресурсів та підвищує ефективність роботи алгоритму.

Принцип роботи алгоритму LPA* ґрунтується на концепції довготривалого планування та інкрементного оновлення інформації у графовій структурі. Для реалізації модифікованої версії цього алгоритму необхідно сформувати базові елементи, зокрема поле ваг, яке відображає вартість переходу між суміжними вузлами графа. Наявність актуальної інформації про ваги дозволяє алгоритму своєчасно коригувати маршрут із мінімальними обчислювальними витратами у відповідь на зміни в середовищі.

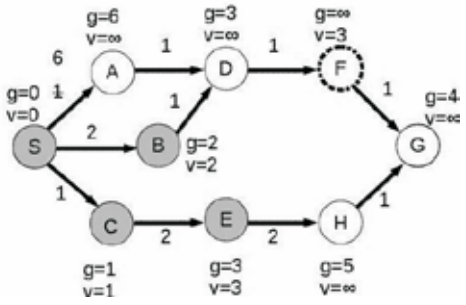

Рис. 3. Алгоритм LPA*

Алгоритм A* призначений для пошуку найкоротшого шляху у графі з ваговими коефіцієнтами ребер. На початковому етапі визначаються стартова та кінцева точки, після чого формуються два списки вузлів:

відкритий (OPEN) та закритий (CLOSED). Кожному вузлу призначаються три основні характеристики: gCost – фактична вартість пройденого шляху від стартової точки, hCost – евристична оцінка відстані від поточного вузла до цільового, та fCost, що є сумою gCost та hCost.

Алгоритм функціонує в циклі до тих пір, поки відкритий список не стане порожнім. На кожній ітерації обирається вузол із найменшим значенням fCost, який переноситься з відкритого списку до закритого. Для кожного суміжного вузла здійснюється перевірка його присутності у відкритому або закритому списку. Якщо сусідній вузол ще не оброблявся, його додають до відкритого списку з відповідним обчисленням значення fCost. Якщо вузол вже відомий, перевіряється можливість покращення шляху до нього, і за потреби відбувається оновлення відповідних характеристик. Процес пошуку завершується, коли досягнуто цільовий вузол або відкритий список спорожнів, що свідчить про відсутність шляху до мети. Блок-схема роботи алгоритму представлена на рисунку 4.

Програмна реалізація алгоритму була виконана у середовищі Unity, де в процесі роботи формується сітка, на якій визначаються стартова та кінцева точки (рисунок 5).

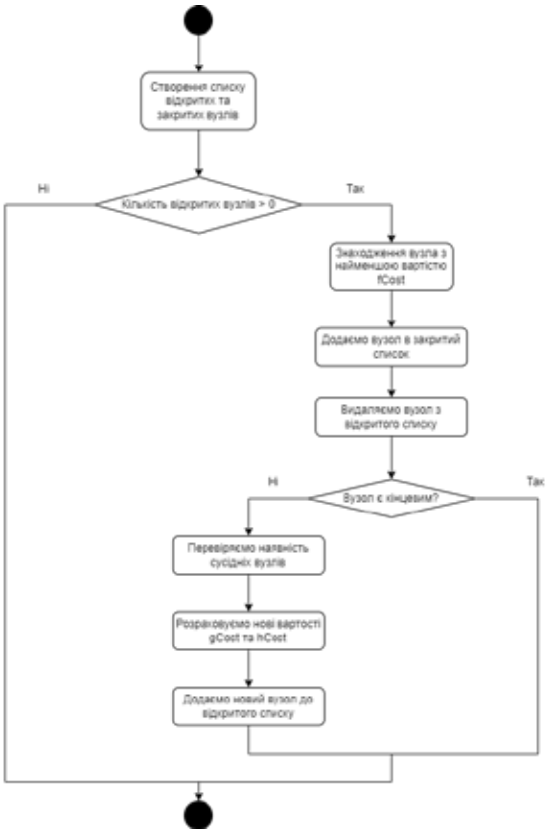

Рис. 4. Алгоритм пошуку шляху A*

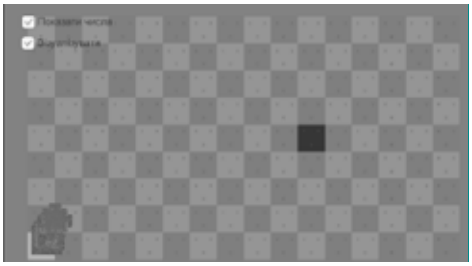

Рис. 5. Створення сітки вузлів при реалізації алгоритму Алгоритм A* на рушію Unity

Після закінчення роботи алгоритму зображується повний шлях з відповідними сусідніми вузлами, що проілюстровано на рисунку 6.

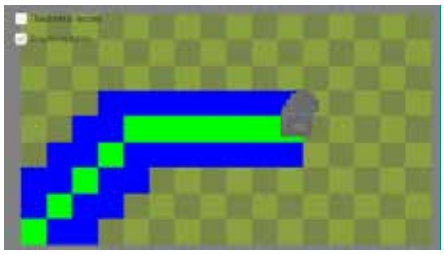

Рис. 6. Зображення повного шляху

Добавлена можливість увімкнути режим показу величини «вартостей» при розрахунку шляху (рисунок 7).

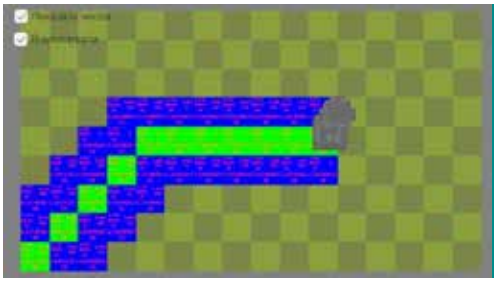

Рис. 7. Візуальне відображення чисел

При збільшенні масштабу зображення можна спостерігати всі величини, при яких обчислюється шлях (рисунок 8).

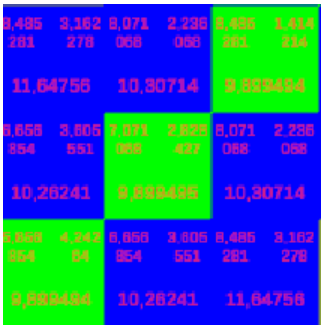

Рис. 8. Зображення всіх величин вузла

На кожному вузлі зображено перші 2 верхніх числа: gCost – відстань вузла від поточного та hCost – відстань вузла від кінцевого, а нижче зображено число fCost – сума величини gCost + hCost.

Висновки. В роботі проведений аналіз методів та алгоритмів пошуку найкоротшого шляху з визначення їхніх переваг і недоліків, а саме визначені найбільш ефективні методи для різних типів рішень, розглянути можливі модифікації методів та особливості їх практичної реалізації. Розроблена програмна реалізація алгоритму, яка заснована на модифікованому методі пошуку найкоротшого шляху, що дозволяє суттєво зменшувати час на прийняття рішень в комп'ютерній грі. Перевагою роботи є розроблення й тестування модифікованої версії алгоритму A*, реалізованого на рушію Unity з автоматичним коригуванням важелів під час динамічної зміни ігрового середовища.

Новизна дослідження полягає в імплементації алгоритму A* в рушію Unity, де було враховано параметри gCost, hCost і fCost для здійснення ефективної роботи алгоритму. Особливу увагу приділено реалізації динамічного оновлення сусідів для кожного вузла. Це дозволяє алгоритму адаптуватися до змін середовища в реальному часі, наприклад, при появі нових перешкод, зміні доступності ділянок мапи тощо. Реалізовано механізми перевірки актуальності сусідів у кожній ітерації пошуку, що підвищує надійність та гнучкість навігаційної системи. Для емпіричної перевірки ефективності розробленої моделі були створені ігрові сцени з різними сценаріями: статична карта, карта з мобільними перешкодами та динамічне оновлення структури середовища. Порівняльний аналіз проводився за кількома критеріями: кількість оброблених вузлів, час виконання пошуку, довжина знайденого маршруту.

Алгоритм демонструє високу ефективність у визначенні найкоротших шляхів, що має значення для задач, в яких критично важливою є мінімізація довжини маршруту. Така гнучкість та налаштованість робить алгоритм придатним для широкого спектру застосувань, включаючи ігрові проекти, симуляції та робототехніку. Відносна нескладність інтеграції в існуючі проекти рушію Unity є ще одним значущим внеском даної роботи, що спрощує використання та модифікацію алгоритму для розв'язання проблем інших досліджень. Розвиток штучного інтелекту, машинного навчання та інших передових технологій відкриває нові можливості для покращення алгоритмів пошуку шляху.

Список використаних джерел:

1. Daohong Liu. Research of the Path Finding Algorithm A* in Video Games. Highlights in Science, Engineering and Technology. Volume 39, 2023. 763-768 p.
2. Фесенко А.Р. Комп'ютерна реалізація аналізу пошуку шляху в програмованих об'єктах гри. URL: https://surl.li/kgipri
3. Гарт, П., Нільсен, Н., Раппапорт, Б. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics. 1968. ol. 4, No. 2. C. 100–107.
4. Introduction to the A* Algorithm. URL: https://www.redblobgames.com/pathfinding/a-star/introduction.html.
5. Dijkstra's Algorithm – Shortest Path. URL: https://surl.li/ysngqd.
6. Алгоритм Дейкстри. URL: https://www.wikiwand.com/uk/articles/Алгоритм_Дейкстри.
7. Рассел, С., Норвіг, П. Штучний інтелект: сучасний підхід. Київ: Діалектика, 2020. 1168 с.
8. LaValle, S. M. Planning Algorithms. Cambridge: Cambridge University Press, 2006. 826 p.
9. Millington, I., Funge, J. Artificial Intelligence for Games. 2nd ed.
10. Koenig, S., Likhachev, M. D* Lite // Proceedings of the AAAI Conference on Artificial Intelligence. 2002. Vol. 15, No. 1. C. 476–483.
11. Unity AI Navigation Docs, 2023.
12. A* Search: Concept, Algorithm, Implementation, Advantages, Disadvantages. URL: https://surl.li/zazxtd.
13. Krianto S. Bellman Ford algorithm – in Routing Information Protocol. Journal of Physics Conference Series, Volume 1007(1), 2018. 10 p.
14. Iterative Deepening A* Algorithm (IDA*). URL: https://www.tpointtech.com/iterative-deepening-a-algorithm.
15. D*, D* Lite & LPA*. URL: https://surl.li/psneqo

References:

1. Liu, D. (2023). Research of the Path Finding Algorithm A* in Video Games. Highlights in Science, Engineering and Technology, 39, 763–768.
2. Fesenko, A.R. (n.d.). Computer Implementation of Pathfinding Analysis in Programmed Game Objects. Retrieved from https://surl.li/kgipri
3. Hart, P., Nilsson, N., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.
4. Introduction to the A* Algorithm. (n.d.). Red Blob Games. Retrieved from https://www.redblobgames.com/pathfinding/a-star/introduction.html
5. Dijkstra's Algorithm – Shortest Path. (n.d.). SURL. Retrieved from https://surl.li/ysngqd
6. Alhorytm Deikstry [Dijkstra Algorithm]. (n.d.). Wikiwand – Wikiwand. Retrieved from https://www.wikiwand.com/uk/articles/Алгоритм_Дейкстри [in Ukrainian].
7. Russell, S., & Norvig, P. (2020). Shtuchnyi intelekt: suchasnyi pidkhid (3rd Ukrainian ed.) [Artificial Intelligence: A Modern Approach]. Kyiv: Dialektyka. [in Ukrainian].
8. LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press.
9. Millington, I., & Funge, J. (2009). Artificial Intelligence for Games (2nd ed.). Morgan Kaufmann.
10. Koenig, S., & Likhachev, M. (2002). D* Lite. Proceedings of the AAAI Conference on Artificial Intelligence – AAAI Conference on Artificial Intelligence, 15(1), 476–483.
11. Unity AI Navigation Docs. (2023). Unity Documentation – Unity Documentation. Retrieved from https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html
12. A* Search: Concept, Algorithm, Implementation, Advantages, Disadvantages. (n.d.). BrainKart – BrainKart. Retrieved from https://surl.li/fvoczu
13. Krianto, S. (2018). Bellman Ford algorithm – in Routing Information Protocol. Journal of Physics: Conference Series – Journal of Physics: Conference Series, 1007(1), 012055. https://doi.org/10.1088/1742-6596/1007/1/012055
14. Iterative Deepening A* Algorithm (IDA*). (n.d.). TpointTech – TpointTech. Retrieved from https://www.tpointtech.com/iterative-deepening-a-algorithm
15. D*, D* Lite & LPA*. (n.d.). JavaTpoint – JavaTpoint. Retrieved from https://www.javatpoint.com/iterative-deepening-a-algorithm