Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка стратегічної комп'ютерної гри з використанням pathfinding-алгоритму

Текстова частина до кваліфікаційної роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник кваліфікаційної роботи
с.в. _Борозенний С.О.__
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2024 р.

Виконав студент _____
___Боровік Н. І.._____
(прізвище та ініціали)
“____” _____ 2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

доцент, к.ф-м.н.

_____О. П. Жежерун (підпис)

„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Боровіку Нікіті Івановичу факультету інформатики 4-го курсу

ТЕМА Розробка стратегічної комп'ютерної гри з використанням pathfinding-алгоритму.

Зміст ТЧ до кваліфікаційної роботи:

Індивідуальне завдання

Анотація

Вступ

1 Жанр Strategy, піджанр Tower Defense та використання pathfinding-алгоритмів у стратегічних іграх.

2 Аналіз алгоритмів знаходження шляху та опис обраного алгоритму.

3 Розробка стратегічної гри.

Висновки

Список літератури

Додатки (за необхідністю)

Дата видачі „___” _____ 2024 р. Борозенний О.С. _____ (підпис)

Завдання отримав _____ (підпис)

Тема: Розробка стратегічної комп'ютерної гри з використанням pathfinding-алгоритму.

Календарний план виконання роботи:

№ п/п	Назва етапу кваліфікаційного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	02.10.2023	
2.	Огляд та аналіз матеріалів за темою роботи, дослідження існуючих алгоритмів.	02.11.2023	
3.	Проектування комп’ютерної гри.	19.12.2023	
4.	Розробка комп’ютерної гри.	05.04.2024	
5.	Написання текстової частини роботи.	30.04.2024	
6.	Створення доповіді.	09.05.2024	
7.	Захист курсової роботи.	27.05.2024	

Боровік Н. І. _____

Борозенний О. С. _____

“_____”_____

Зміст

Анотація

Кваліфікаційна робота присвячена розробці стратегічної комп’ютерної гри, що використовує pathfinding-алгоритм для ігрових сутностей. У процесі розробки комп’ютерної гри було досліджено жанр стратегія і виокремлено особливості його піджанру tower defense, а також досліджено способи знаходження шляхів на ігровому світі сутностями гри в заданому жанрі.

Ключові слова: комп’ютерна гра, Unity, pathfinding, стратегічна гра, tower defense.

Вступ

Ігрова індустрія є однією з провідних ґалузей у сфері інформаційних технологій. Важливою складовою багатьох ігор є впровадження pathfinding-алгоритмів для ігрових сутностей, що дозволяє їм природньо орієнтуватися в ігровому світі. Особливо широкого використання такі алгоритми набули у стратегічних іграх, популярних також і через можливість гравцю використовувати усі ресурси свого розуму для вирішення складних задач в середовищі з великою кількістю сутностей, що природньо в ньому орієнтуються та створюють відчуття масштабності подій.

Метою кваліфікаційної роботи є аналіз способів знаходження ігровими сутностями шляхів у ігровому світі, та розробка стратегічної гри з впровадженням у неї pathfinding-алгоритму.

Текстова частина кваліфікаційної роботи містить в собі 3 основні розділи:

У першому розділі висвітлюються особливості жанру стратегія, обраного для розробки піджанру, а також описується місце pathfinding-алгоритмів в стратегічних іграх.

У другому розділі аналізуються алгоритми знаходження шляху та описується обраний для кваліфікаційної роботи pathfinding-алгоритм.

У третьому розділі презентовано засоби розробки та більш детальний огляд коду впровадженого алгоритму з його класами та методами.

Основна частина

Розділ 1: Жанр Strategy, піджанр Tower Defense та використання pathfinding-алгоритмів у стратегічних іграх

1.1 Що таке стратегічна гра

Стратегічна гра[1] – одна з найстаріших видів ігор, відомих людству. Для стратегічної гри запорукою перемоги виступає мислення та планування, а не тільки прямі миттєві дії. Однією з найдавніших стратегічних ігор можемо вважати китайські шашки, де три гравці змагалися, намагаючись перемістити свої кульки з однієї половини дошки на іншу, використовуючи простий набір правил, де можна було переміщувати свою кульку на одне місце за раз, перестрибуючи, за необхідності, одну кульку, але не маючи можливості перестрибнути дві чи більше. Навіть такий простий набір правил передбачав складну тактику гри та необхідність обмірковувати свої дії для досягнення перемоги.

Першими комп’ютерними стратегічними іграми були симуляції реальних ігор, таких як шахи і хрестики-нулики, у яких одним з опонентів зазвичай виступав комп’ютер [1].

Сучасні стратегічні відеоігри поділяють на два головні підвиди: “покрокові стратегії” (англ. Turn-Based Strategy) та “стратегії в реальному часі” (англ. Real Time Strategy)[2]. Хоча існує багато піджанрів комп’ютерних стратегічних ігор, проте усіх їх так чи інакше можна віднести до однієї з двох наведених груп.

Покрокові стратегії[3] – вид стратегічних ігор, що поділяють ігровий процес на “кроки”, що дозволяють кожному з гравців зробити лише певну обмежену послідовність дій. Існують як підходи, коли хід кожного гравця

має свою черговість і гравці ходять один за одним, так і такі, коли усі гравці мають виконати свої дії, щоб почати новий “крок” гри. Такий підхід відомий людству ще з часів настільних ігор, де гравці зазвичай ходять один за одним. Однією з перших успішних покрокових стратегічних відеоігор вважають Eastern Front (1941), розробленою для відео-консолі Atari у 1981 році [2].

Рис. 1 Eastern Front (1941)

Стратегії в реальному часі[4] – вид стратегічних ігор, процес гри в яких триває безперервно, тобто гравці можуть виконувати свої дії одночасно, не чекаючи дій від опонентів. Цей вид стратегічних ігор набув своєї популярності через більший динамізм подій. Однією з перших стратегій в реальному часі була Herzog Zwei, випущена у 1989 році. Визначною ж для жанру стала Dune II, що побачила світ у 1992 році[5].

Рис. 2 Herzog Zwei

Рис. 3 Dune 2

Жанр стратегічних ігор має багато піджанрів, що можна віднести до покрокових стратегій, стратегій у реальному часі або їх комбінації. Серед головних піджанрів слід виокремити:

- 4X (e*x*plore, e*x*pand, e*x*ploit, and e*x*terminate) – піджанр стратегічних ігор, в якому гравець контролює імперію і, як засвідчує назва, повинен досліджувати (e*x*plore) території, розширювати (e*x*pand) власні землі, використовувати (e*x*ploit) наявні ресурси та знищувати (e*x*terminate) супротивників[6].
- Artillery– піджанр стратегічних ігор (зазвичай покрокових), що включають танкові бої між гравцями а також схожі похідні ігри. Зазвичай гравці повинні прицілюватися та стріляти, намагаючись знищити свого опонента[7].
- Auto battler– піджанр стратегічних ігор, також відомий як auto chess, що містить подібні до шахів елементи, де гравці розміщують персонажів на полі бою, що представляє собою сітку, подібну до шахової, які потім борються з персонажами команди суперника без прямого втручання гравця[8].
- Multiplayer online battle arena (MOBA) – піджанр стратегічних відеоігор у реальному часі, де гравець керує одним персонажем із набором унікальних здібностей, які можна вдосконалювати протягом гри та які впливають на загальну стратегію команди[9]
- Massively multiplayer online real-time strategy games (MMORTS) – піджанр стратегічних ігор у реальному часі з постійним ігровим світом (persistent world), де гравці виступають у ролі лідера, що веде армію в бій, зберігаючи та накопичуючи ресурси, необхідні для війни[10]

- Tower Defense – піджанр стратегічних ігор, де гравець повинен будувати вежі на шляху ворогів, щоб знищити їх раніше, ніж вони тим чи іншим шляхом завдадуть йому поразки[11].

1.2 Tower Defense як піджанр стратегічних ігор

Для виконання розробки стратегічної гри і впровадження у неї pathfinding-алгоритму було обрано піджанр Tower Defense через його простоту відносно інших піджанрів стратегічних ігор та видиму можливість впровадження алгоритму знаходження шляху для ігрових сутностей, які становлять важливу частину ігор заданого піджанру.

Ігри заданого жанру фокусують гравця на розподілі ресурсів та розміщенні бойових одиниць (веж). Основною ігровою активністю виступає покупка, розміщення та модернізація захисних споруд, що автоматично атакують хвилі ворогів (крипів). Знищення ворогів дає гравцю ресурси, що обраховуються відповідно до сили знищеного кріпа. Гравець досягає перемоги після знищення певної кількості крипів або програє, якщо кріпи дійшли до краю ігрового світу чи знищили головну споруду гравця.

Можна виокремити 5 основних компонентів, що в комбінації дають представника жанру tower defense[12]:

- Місцевість - ігрова карта визначає як гравець розміщує свої вежі. Вороги можуть переміщуватися тільки по певних шляхах (підхід класичних представників жанру) або мати можливість шукати оптимальний шлях до своєї цілі. Гравцю може бути дана можливість розміщувати захисні споруди у будь-якій частині ігрової місцевості, або ж, для підвищення складності, місця розміщення башт можуть бути певним чином обмежені.
- Вежі - захист гравця від крипів. Їх розміщення може бути нічим не обмежене, або обмежене шляхами переміщення крипів та певними особливостями ігрової місцевості. Вежі можна покращувати і вони

можуть мати різні можливості та вартість. Зазвичай різні вежі треба використовувати проти різних ворогів та у певних ігрових ситуаціях, що спонукає гравця детальніше продумувати власну стратегію та акцентувати увагу на виборі певного типу вежі.

- Кріпи – представлення ворогів у жанрі tower defense, що прагнуть дійти до кінця ігрового світу або знищити основну споруду гравця. Вони зазвичай мають різні атрибути, такі як розмір, швидкість, шкода та захист. В більшості випадків у іграх заданого жанру вороги початково мають низькі ігрові параметри і стають сильнішими у процесі проходження гравцем гри, тим самим збільшуючи складність ігрового процесу. Кріпи можуть мати супротив проти веж певного типу, що змушує гравця урізноманітнювати свій арсенал та продумувати стратегію боротьби.
- Система нагород – спосіб підвищити цікавість гри та збільшити її тривалість. Вежі, розміщені гравцем, знищують крипів, що призводить до отримання гравцем нагороди у вигляді певного ресурсу, необхідного для розміщення нових захисних споруд та модернізації вже існуючих. Таким чином гравець може постійно збільшувати свої оборонні спроможності та протистояти дедалі сильнішим та численнішим ворогам.
- Один чи кілька гравців. В більшості випадків гравець лише один і він самотужки розробляє стратегію використання захисних веж проти крипів, які є керованими грою. Також є випадки, коли декілька гравців можуть співпрацювати, спільно розробляючи стратегію та розташовуючи захисні споруди, та змагатися в кількості знищених крипів.

1.3 Роль Pathfinding-алгоритмів у іграх стратегічного жанру

Алгоритми пошуку шляху в комп’ютерних іграх є предметом багатьох досліджень. Дана проблема є як однією з найпопулярніших, так і однією з найскладніших проблем ігрового штучного інтелекту[13]. Проблема пошуку шляху в комп’ютерних іграх полягає в тому, що вона повинна вирішуватися в реальному часі та з використанням обмежених ресурсів пам’яті та процесору.

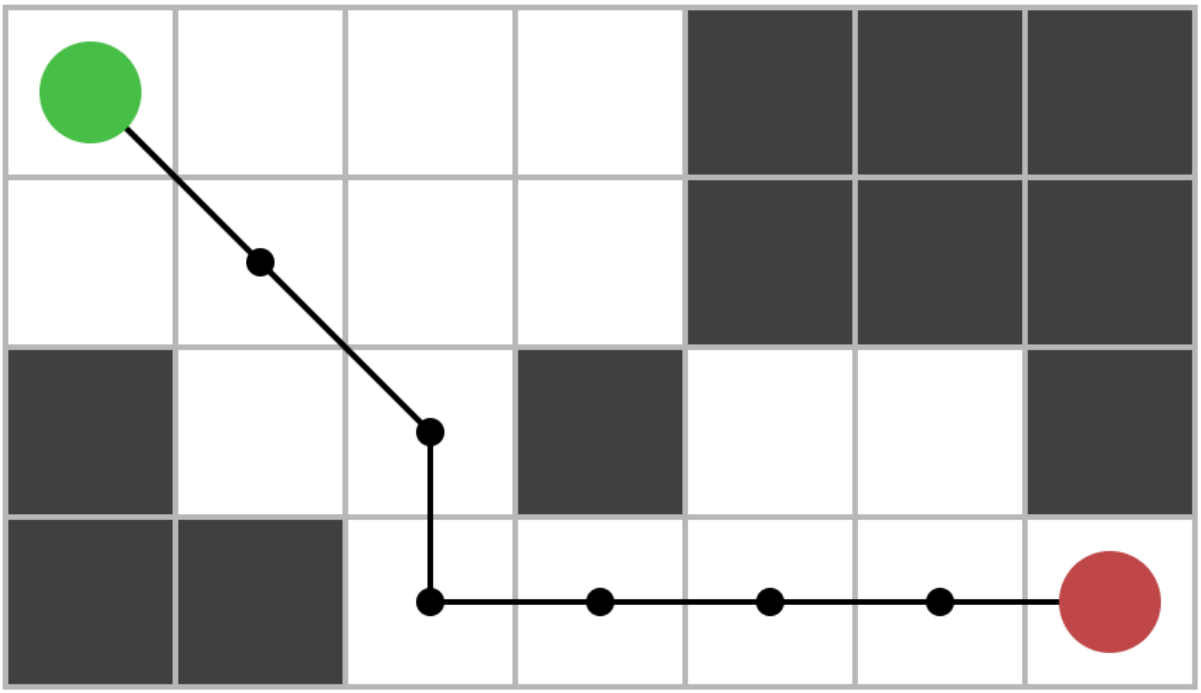


Рис. 4 Приклад побудови шляху на сітці

У стратегічних іграх зазвичай є певні ігрові та неігрові персонажі, що повинні пересуватися ігровим світом зі свого місцезнаходження у визначений гравцем або ігровою системою пункт призначення. Таким чином перед розробниками ігор (зокрема стратегічних) постає задача пересування ігрових агентів, що і є однією з найбільших проблем у розробці інтелекту ігрових персонажів, які будуть поводитися реалістично. Зазвичай найбільші труднощі виникають у стратегіях в реальному часі, зумовлені тим, що усі ігрові персонажі можуть рухатися одночасно. Найпоширенішим завданням

тут виступає пошук найвигіднішого шляху за умови уникання перешкод на ігровій мапі, що додатково ускладняється необхідністю розраховувати шлях для багатьох ігрових персонажів, велика кількість яких дуже часто є ключовою особливістю ігор в жанрі стратегія.

Рис. 5 Приклад гри з великою кількістю персонажів

Розділ 2: Аналіз алгоритмів знаходження шляху та опис обраного алгоритму

2.1 Аналіз алгоритмів знаходження шляху

Основна мета використання алгоритмів знаходження шляху в іграх полягає в тому, щоб забезпечити ігрових персонажів ефективним і оптимальним переміщенням по ігровому полю, зменшуючи час на прийняття рішень та підвищуючи реалістичність їх поведінки. Пошук шляху в іграх (зокрема і у стратегічних) можна віднести до концепції[14] знаходження оптимального шляху від початкового вузла до кінцевого на певному графі за мінімальний час.

Для представлення ігрового світу зазвичай використовують сітку, що представляє собою поєднання між собою певних вершин або точок через ребра, що формують граф. За структурою сітки можуть бути регулярними та нерегулярними. Регулярна сітка складається з однакових одиниць меншого розміру, що є двовимірними геометричними фігурами або тривимірними геометричними об’єктами. До регулярних сіток відносяться трикутні, квадратні, шестикутні та кубічні. Нерегулярні сітки не сформовані меншими за розміром одиницями однакової форми, натомість до них відносять маршрутні точки (waypoints), граф видимості (visibility graph) або навігаційну сітку (navigation mesh). Надалі в роботі розглядатимемо алгоритми знаходження шляху відносно двовимірної квадратної регулярної сітки.

Можна виокремити три основні напрямки алгоритмів знаходження шляху: неінформовані алгоритми пошуку, інформовані або евристичні алгоритми пошуку, та метаевристичні алгоритми.

Неінформовані алгоритми пошуку[15] не використовують будь-які конкретні знання та евристику про проблемну область для дослідження простору. Такі алгоритми також відомі як сліпий пошук, вони

використовують грубу силу, перевіряючи кожну частину пошукового простору наосліп.

Інформовані алгоритми пошуку[16] містять набір спеціальних даних, таких як відстань від цілі або вартість конкретного шляху, за допомогою яких вони можуть досліджувати менший обсяг простору та ефективніше знаходити шлях до цілі. Такі алгоритми пошуку використовують ідею евристики, маючи евристичну функцію, що розраховує вартість шляху від конкретного вузла до цільового, допомагаючи визначити більш оптимальні для перевірки вузли та зменшити обсяг обчислень. Через використання евристики ці алгоритми також часто називають евристичними.

Метаевристичні алгоритми пошуку[17], подібно до евристичних алгоритмів, спрямовані на пошук багатообіцяючих результатів для вирішення проблеми. Однак, алгоритм, який використовується для метаевристики, є загальним і не залежить від конкретної задачі. Таким чином метаевристика змінює напрям дизайну з проблемно-орієнтованого на проблемно-незалежний.

Серед відомих алгоритмів знаходження шляху для кожної з груп слід виокремити наступні:

Неінформовані:

- Breadth First Search (BFS) або пошук в ширину[18] – алгоритм неінформованого пошуку шляху, вперше опублікований Едвардом Муром у 1959 році. Згідно з алгоритмом, вершини графа (клітини сітки) відвідуються по черзі. Спочатку перевіряється кожна клітина за один крок від стартової, потім клітини другого кроку і так далі, поки усі клітини не будуть перевірені. Перевагою алгоритму є те, що він дає нам можливість знайти найкоротший шлях за мінімальну кількість пройдених кроків і рішення буде знайдено завжди для кожної конкретної проблеми. Недоліком цього алгоритму можна назвати його високий рівень використання пам’яті через необхідність тримати усі

вузли в дереві і перевіряти кожен вузол попереднього рівня для перевірки вузла з наступного рівня. Часова складність для алгоритму становить O(V + E), а просторова – O(V), де V – кількість вершин графу (клітин на сітці), а E – кількість ребер у графі (зв'язків між клітинами на сітці)[19]. Цей алгоритм був одним з фундаментальних алгоритмів в ігровій розробці. Одним з прикладів його використання в іграх є проходження ігрових лабіринтів.

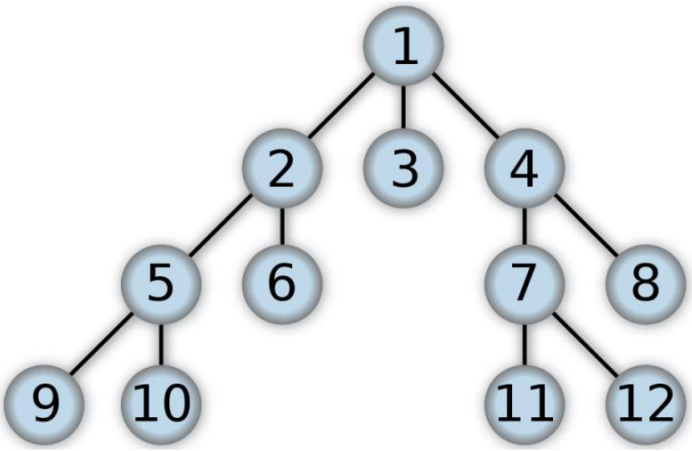


Рис. 6 Порядок обходу графу алгоритмом BFS

- Depth First Search (DFS) або пошук в глибину[20] - алгоритм неінформованого пошуку шляху, що використовує зворотне відстеження. Алгоритм початково був винайдений Шарлем П'єром Тремо як стратегія для вирішення лабіринтів[21]. Він полягає в вичерпному дослідженні усіх вузлів шляхом переходу вперед від початкового, якщо це можливо, та повернення назад, якщо встановити шлях не вдалося. Згідно з алгоритмом, обирається початковий вузол (клітина) і усі його сусіди необхідно помістити у стек. Далі зі стеку береться наступний вузол для відвідування і всі його сусіди також поміщаються у стек. Цей процес повторюється допоки стек не

спорожніє. Корисною практикою також є позначення відвіданих вузлів для запобігання створенню циклів. Головним недоліком цього алгоритму є те, що він не гарантує, що знайдений шлях буде найкоротшим. Часова складність для алгоритму становить O(V + E), а просторова – O(V), де V – кількість вершин графу (клітин на сітці), а E – кількість ребер у графі (зв'язків між клітинами на сітці) [22].

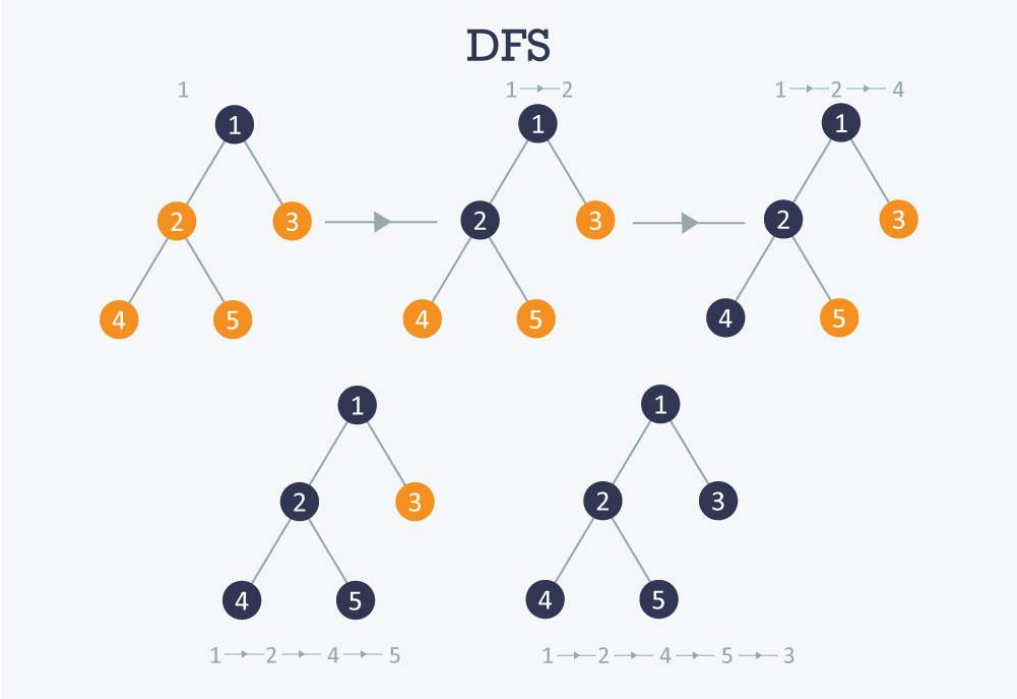
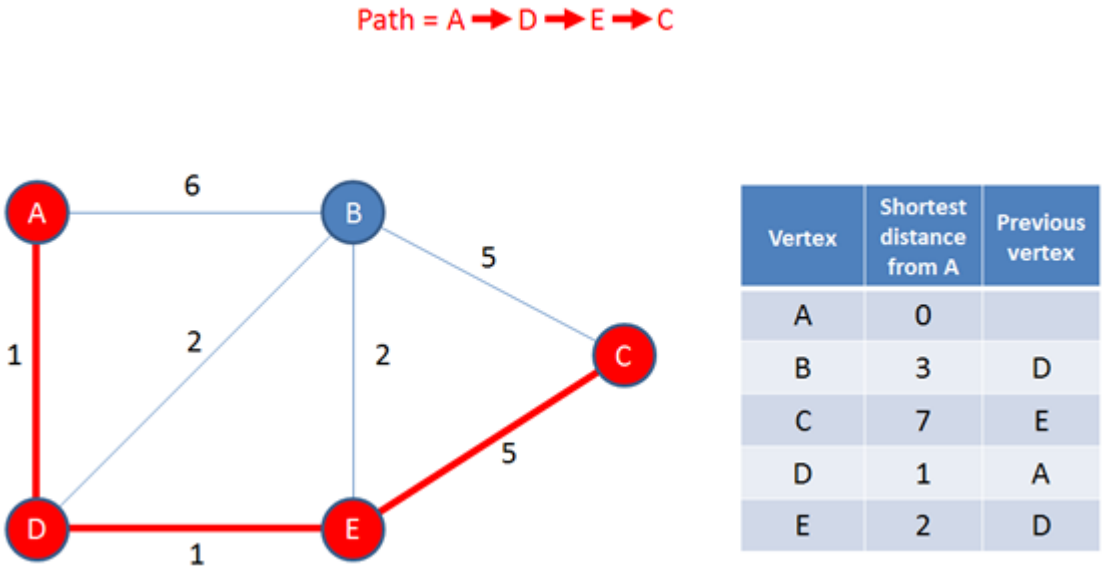


Рис. 7 Порядок обходу графу алгоритмом DFS

- Алгоритм Дейкстри[23] – алгоритм неінформованого пошуку, вперше опублікований Едсгером Дейкстрою. Алгоритм полягає у знаходженні найкоротшого шляху від початкової до всіх інших вершин шляхом вибору найближчої невідвіданої вершини та обчислення відстані для всіх невідвіданих сусідніх з нею вершин. Цей алгоритм був популярним для пошуку оптимального шляху в іграх до винайдення більш оптимального алгоритму A*[24]. У даного алгоритму є недоліки у вигляді значного використання пам'яті (потрібно обробити усі вершини) та неможливості використовувати його для ребер з

від’ємними значеннями[14]. Часова складність для алгоритму становить O($(V + E)\ log\ V$) в загальному випадку або O(V^2) в гіршому, а просторова – O(V), де V – кількість вершин графу (клітин на сітці), а E – кількість ребер у графі (зв’язків між клітинами на сітці) [25].



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Рис. 8 Ілюстрація алгоритму Дейкстри на графі

Евристичні:

- Алгоритм A*[26] – евристичний алгоритм знаходження шляху, представлений Хартом, Нільсоном і Рафаелем у 1968 році. Для пошуку шляху алгоритм досліджує найбільш багатообіцяючий вузол з відомих (вузол з найменшою вартістю, яка вираховується як значення суми вартості шляху до цього вузла від початку та евристики, що є припущенням ваги переміщення від заданого вузла до цілі). Якщо вузол є ціллю, алгоритм завершується. Інакше усі сусіди поточного вузла помічаються для подальшого дослідження. Алгоритм має декілька важливих особливостей[27]: по-перше він гарантовано знаходить шлях від початку до цілі, якщо такий існує, по-друге цей шлях є оптимальним, якщо евристика h(n) є прийнятною (завжди

менша або рівна до справжньої найменшої вартості шляху від n до цілі), по-третє A* найоптимальнішим чином використовує евристику, тобто жоден метод пошуку, який використовує ту саму евристичну функцію для пошуку оптимального шляху, не досліджує менше вузлів, ніж A*. Також важливою перевагою є наявність додаткових оптимізацій алгоритму[26], таких як HPA* або використання NavMesh. Зважаючи на простоту та ефективність цього алгоритму, він є одним з найпопулярніших алгоритмів знаходження шляху для ігрової індустрії. Часова складність для алгоритму залежить від евристичної функції і у гіршому випадку становить O(b^d), а просторова – завжди O(b^d), де b – середня кількість граней від кожного вузла, а d – кількість вершин у результуючому шляху[28].

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

Рис. 9 Робота алгоритму A на квадратній сітці*

- Алгоритм D* (Dynamic A*) - евристичний алгоритм, винайдений Ентоні Стенцом у 1994 році [29]. D* це алгоритм, що розроблявся для ситуацій, коли повна інформація про оточення може бути недоступною або вона може дуже швидко змінюватися. Прикладом сфери використання алгоритму D* може бути робот-дослідник, що

орієнтується на невідомій місцевості планети або персонаж комп'ютерної гри, якому треба швидко орієнтуватися на невідомій закритій мапі. Алгоритм працює шляхом почергового вибору вузлів зі списку відкритих вузлів (OPEN) та їх оцінки. D* проводить пошук шляху від цільового вузла до початкового. Кожен оброблений вузол має вказівник на наступний вузол на шляху до цілі та вартість шляху до неї. Коли виявляється перешкода на шляху, всі точки, які зазнали її впливу і мають бути перераховані, знову поміщаються в список відкритих, цього разу позначені як підняті (RAISE) для збільшення їх вартості. Перш ніж піднятий вузол збільшиться за вартістю, алгоритм перевіряє його сусідів і досліджує, чи є можливість знизити вартість вузла. Якщо ні, стан піднятого передається всім нащадкам вузла, тобто вузлам, які мають на нього вказівники. Ці вузли потім оцінюються, і стан RAISE передається далі, утворюючи хвилю. Існують ситуації, коли вартість вузла може бути знижена. Тоді такий вузол отримує помітку LOWERED, його вартість оновлюється а він передає цей стан свої сусідам, щоб алгоритм міг спробувати знизити їх вартість. Часову складність алгоритму D* складно оцінити через значний вплив евристичної функції та частоти зміни інформації про простір. В загальному випадку алгоритм D* показує дещо гірші часові результати, ніж A* [30], проте він може бути оптимальнішим для закритої мапи або у випадках дуже частих її змін. Просторова складність D*, як і у A*, становить O(b^d).

Метаевристичні:

- Генетичний алгоритм (ГА)[31] – метаевристичний алгоритм, особливий вид стохастичного пошукового алгоритму, що використовує біологічну еволюцію як метод розв'язання проблеми. Цей алгоритм базується на відомому принципі Дарвіна – "виживають

найкращі”. Генетичний алгоритм працює у просторі пошуку, що називається популяцією, де кожен окремий розв’язок називається хромосомою. На кожному кроці ГА обирає деяких представників з поточної популяції і використовує їх для створення нащадків для наступного покоління. Алгоритм обирає рішення для створення наступної популяції, обраховуючи їх придатність за допомогою евристичної техніки пошуку. Нові хромосоми генеруються з попередників шляхом відбору, перехрещування та мутацій. Таким чином хромосоми нового покоління “еволюціонують” щоб наблизитися до оптимального розв’язку. Цей процес повторюється допоки не буде досягнуто задовільного рішення для заданої проблеми. Хоча теоретично визначити складність генетичного алгоритму важко через його стохастичну природу та залежність від конкретної задачі, емпіричні дослідження показують, що цей алгоритм може знаходити шляхи швидше за A*, проте знайдені шляхи не завжди є найоптимальнішими, а самі результати роботи алгоритму не є постійними і можуть відрізнятися між запусками[32].

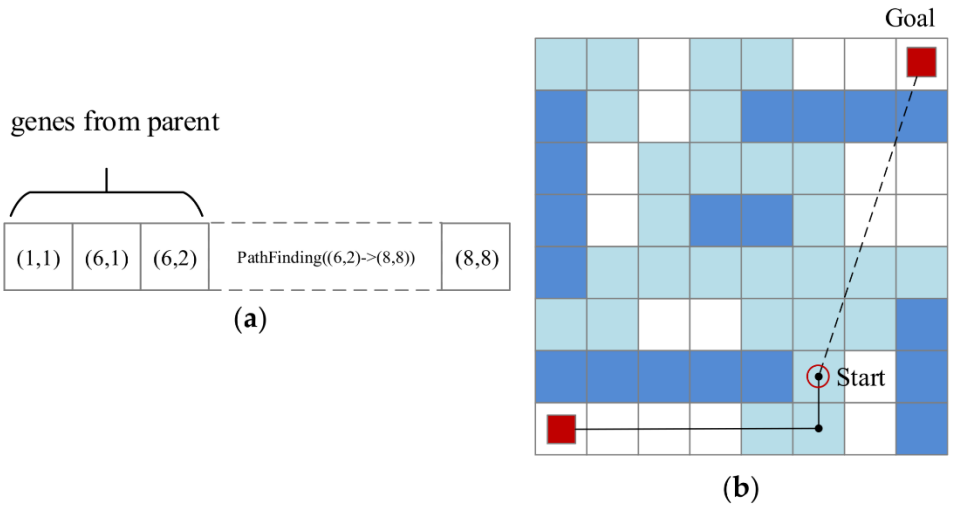


Рис. 10 Приклад процесу роботи генетичного алгоритму.

- Оптимізація мурашиної колонії (Ant Colony Optimization / ACO)[33] – не алгоритм, а ціла парадигма для розробки метаевристичних алгоритмів. Оптимізація мурашиної колонії використовує поведінку мурах у колонії як джерело натхнення для вирішення задач оптимізації. Важливою рисою, задекларованою в ACO, є поєднання апріорної інформації про структуру перспективного рішення з апостеріорною інформацією про структуру попередніх задовільних рішень. Використовуючи ACO для пошуку шляху, ми уподібнюємо агента пошуку до мурахи. Мураха починає з початкової точки і має порожній набір вузлів у своєму шляху. Пересуваючись між вузлами, агент помічає їх певним значенням, подібно до того, як мурахи позначають свої шляхи феромонами. Ці значення використовуються іншими агентами для прийняття рішення щодо вибору наступного вузла. На кожному кроці мураха обирає наступний вузол відповідно до вірогідності переходу до нього, що є балансом між евристичною інформацією про перехід (бажаність кроку) та інтенсивністю феромонів на цьому шляху. Процес повторюється до знаходження шляху до цілі, після чого наступна мураха починає свій шлях від початку. Це триває поки певна кількість мурах не побудує маршрут і не буде знайдено оптимальний шлях до цілі. Для оптимізації шляху, для кожної перешкоди на просторі використовується окрема колонія (набір агентів пошуку), що оптимізують шлях через цю перешкоду. Колонії координуються для знаходження рішення, що оптимізує подолання усіх перешкод, шляхом обміну інформацією про рішення, знайдені кожною колонією. Як і у випадку з генетичним алгоритмом, теоретичну складність ACO важко оцінити, проте, на основі емпіричних досліджень ACO виявляється швидшим за генетичний алгоритм.[34].

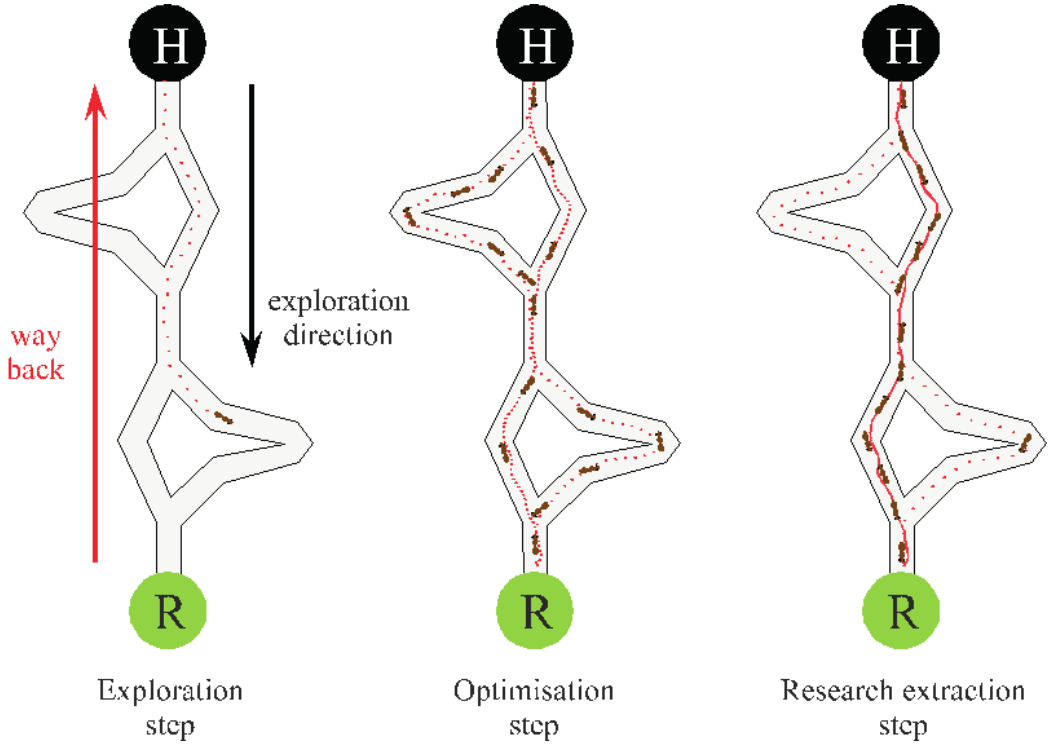


Рис. 11 Приклад стратегії ACO

2.2 Детальний огляд обраного алгоритму знаходження шляху

2.2.1 Теоретичні відомості.

Для реалізації знаходження шляху в грі у жанрі стратегія було обрано алгоритм A*. Рішення було прийняте на основі багатьох переваг, притаманних алгоритму:

- Описані в пункті 2.1 переваги A* стосовно того, що алгоритм завжди знаходить шлях (найоптимальніший для допустимої евристики) і найефективніше використовує евристику.
- Алгоритм A* є доволі простим в розумінні і реалізації, що робить процес його впровадження легшим та швидшим.
- Існують додаткові оптимізації алгоритму A*, що може стати в нагоді для вирішення задач пошуку з підвищеною складністю[26].
- Змінюючи значення параметрів евристичної функції можна додатково підлаштовувати поведінку сутностей, які оперують A* для

знаходження шляху, що може бути корисним для досягнення бажаної поведінки ігрових персонажів.

- Розповсюдженість даного алгоритму в ігровій розробці робить його надійним та перевіреним часом засобом знаходження шляху.

Алгоритм A* оперує трьома важливими параметрами: f, g, h.

f(n) – параметр, що відповідає значенню зваженої ваги вузла n. На кожному кроці роботи алгоритм обирає вузол з найменшим значенням цього параметра, щоб знайти оптимальний шлях до цілі. Саме значення параметра f це сума двох інших параметрів, тобто f(n) = g(n) + h(n)

g(n) – вартість шляху від початку маршруту до вузла n, рухаючись маршрутом, який вже був згенерований, щоб дістатися вузла n.

h(n) – припущена вартість переміщення від вузла n до цільового вузла. Цей параметр часто називають евристикою, тобто розумним припущенням. Пересуваючись сіткою в ігровому світі, ми не можемо бути впевнені, що на шляху не виникне перешкод, таких, як, стіни чи прірва. Через це h(n) є лише певною мірою вірогідності того, наскільки сильно вузол n наближає нас до цілі.

Значення h вираховується відповідно до специфіки конкретної задачі і функція його обрахування буде різною для різних проблем. Так, наприклад, для знаходження шляху між містами на мапі значенням h може бути довжина прямої між початковим містом та містом призначення.

Зазвичай, коли розглядаємо рух двовимірною квадратною сіткою, є три основні апроксимаційні евристики, щоб розрахувати значення h[35]: Манхеттенська відстань – сума абсолютних значень різниць координат x та y цільового та поточного вузла. Така евристика є максимально простою та ідеально підходить для випадків, коли рух дозволено лише в чотирьох напрямках (вертикально та горизонтально).

$$\mathrm{h}(n) = |n.x - goal.x| + |n.y - goal.y|$$

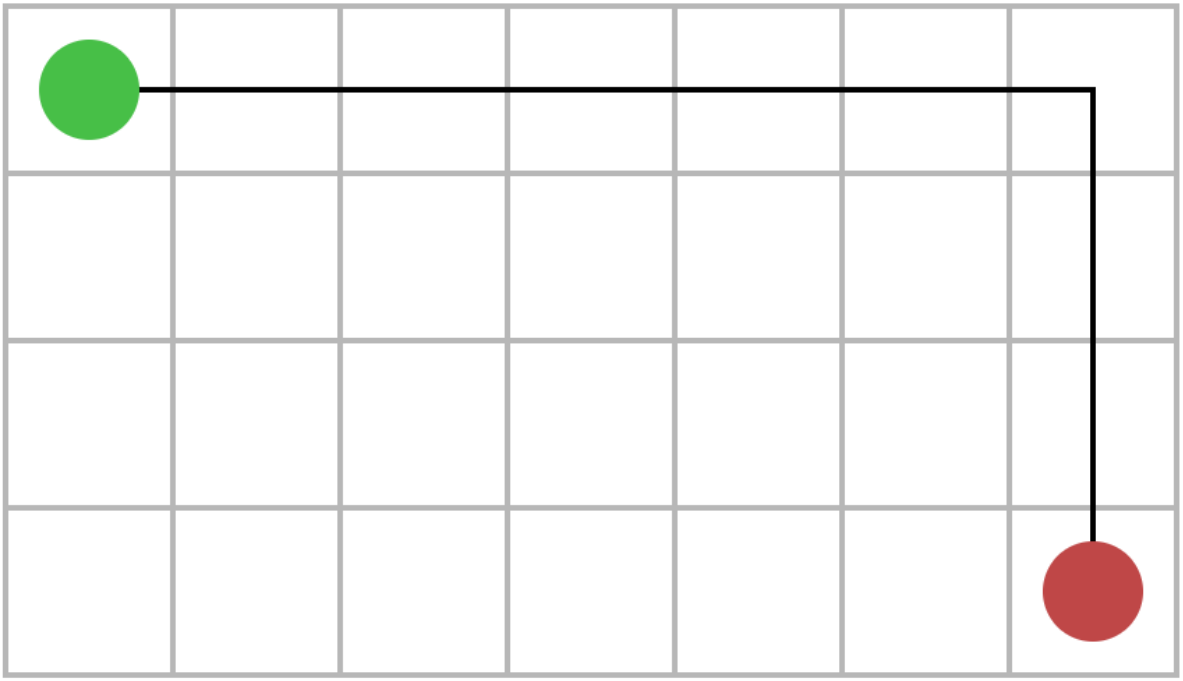


Рис. 12 Ілюстрація манхеттенської відстані між двома точками на сітці

Діагональна відстань – більш складна евристика, що найкраще підходить для випадків, коли рух на сітці дозволено у восьми напрямках (вертикально, горизонтально та діагонально).

$$dx = |n.x - goal.x|$$

$$dy = |n.y - goal.y|$$

$$h(n) = c_h * |dx - dy| + c_d * \min(dx, dy)$$

c_h – вартість горизонтального переміщення

c_d – вартість діагонального переміщення

рис. 13 Ілюстрація діагональної відстані між двома точками

Евклідова відстань – евристика, що застосовується, коли рух дозволено у будь-якому напрямку і під будь-яким кутом.

$$h(n) = \sqrt{(n.x - goal.x)^2 + (n.y - goal.y)^2}$$

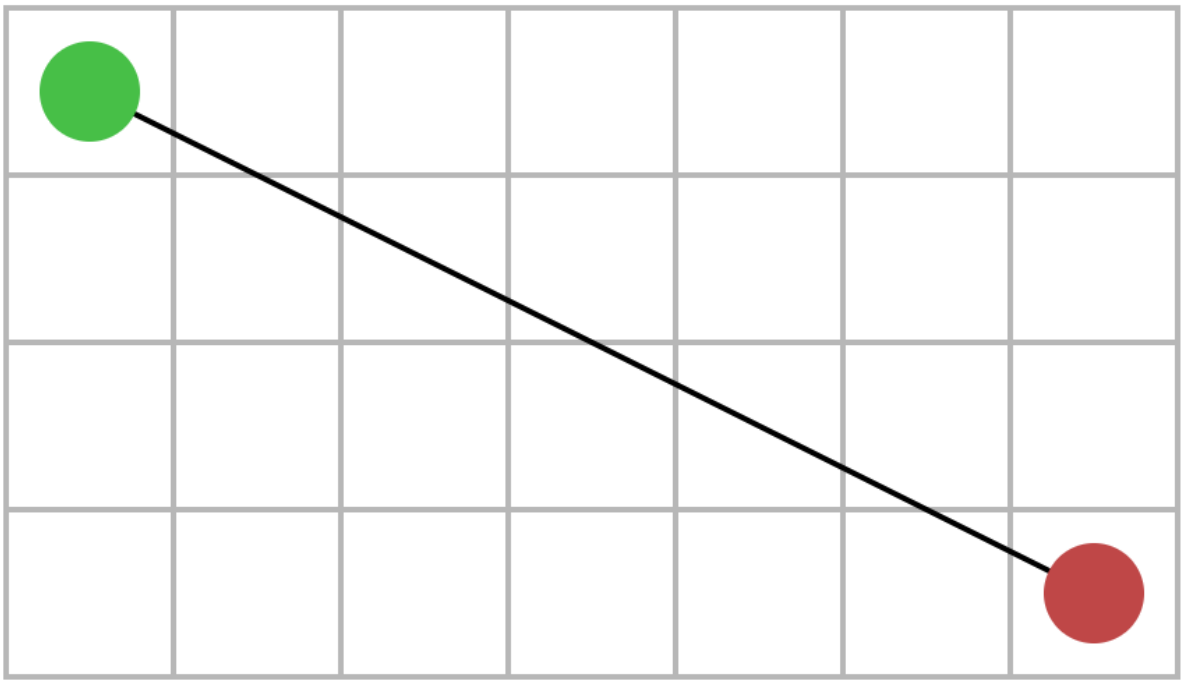


рис. 14 Ілюстрація Евклідової відстані між двома точками

2.2.2 Принцип роботи алгоритму A*.

Під час роботи, алгоритм оперує наступними компонентами:

1. Вузол – представлення клітини на сітці. Послідовність вузлів створює шлях. Кожен вузол має параметри f, g, h, вказівник на вузла-батька, тобто попередній вузол зі шляху, та координати (x та y для двовимірної сітки). Вузли також можуть бути позначеними як недоступні для переміщення, тобто представляти перешкоди на шляху.
2. Сітка – область пошуку, що представляє собою набір вузлів.
3. Відкритий список – сховище вузлів, які треба опрацювати. Для його реалізації гарною практикою є вибір черги з пріоритетом як структури даних.
4. Закритий список – сховище вузлів, які вже були опрацьовані алгоритмом. Зазвичай множина використовується як обрана для реалізації структура даних.

Процес роботи алгоритму можна розділити на декілька важливих етапів:

1. Відбувається ініціалізація відкритого та закритого списків.
2. Початковий вузол додається у відкритий список.
3. Починається ітеративний процес пошуку шляху.
    a. З відкритого списку обирається вузол з найменшим значенням f, одразу ж прибираючи його з відкритого списку та додаючи до закритого. Цей процес стає більш оптимізованим, якщо черга з пріоритетом була обрана як структура даних для реалізації відкритого списку.
    b. Для обраного вузла обирається 8 його сусідів (усі горизонтальні, вертикальні та діагональні) з сітки.
    c. Опрацьовуємо кожного з 8 сусідів.
        i. Якщо вузол-сусід є недоступним для переміщення або він є у закритому списку, алгоритм ігнорує його.

ii. Якщо вузла-сусіда немає у відкритому списку, відбувається додавання даного вузла до відкритого списку, одночасно розраховуючи для нього значення g, h, f та встановлюючи поточний вузол батьком для опрацьовуваного вузла-сусіда.

iii. Якщо він є у відкритому списку, відбувається перевірка оптимальності шляху до цього вузла, порівнюючи значення g вузла з відкритого списку та обраховане для шляху з поточного вузла до вузла-сусіда (нижче значення g показує більш оптимальний шлях). Якщо поточно обраховане значення є кращим, батьком вузла з відкритого списку стає поточний вузол і його значення g та f перераховуються.

d. Пошук шляху припиняється успішно, якщо вузол, до якого ми прагнули побудувати шлях, був доданий до закритого списку. Якщо ж не вдалось дійти до цільового вузла і відкритий список став порожнім, то побудувати шлях не вдалось.

4. Якщо вдалось дійти до цільового вузла, то можна побудувати шлях, почавши з кінцевого вузла і рухаючись по батьківським вузлам, додаючи їх у стек.

A* надає можливість керувати процесом побудування шляху через внесення змін до значень параметра f у вузлів. Додатково зменшуючи, або збільшуючи значення f, відповідно до певних умов, можна робити вузол більш або менш бажаним для відвідування. Прикладом може бути надання певним клітинкам ігрового поля додаткової ваги, щоб наказати ігровим сутностям уникати їх за можливості, але залишити можливість пересуватись ними, якщо немає альтернативних шляхів, або якщо вони є занадто довгими.

Розділ 3: Розробка стратегічної гри.

3.1 Інструменти розробки.

Підходячи до розробки гри, було обрано рушій Unity як основний інструмент для створення проєкту. Unity має декілька важливих переваг, що роблять його гарним вибором для розробки ігрових продуктів. Серед таких переваг можна виокремити широкий спектр вбудованих інструментів для розробки гри, підтримку гнучкої та потужної мови програмування C#, постійне оновлення рушія з відповідним покращенням наявних можливостей та додаванням нових, орієнтованість рушія на невеликі команди розробників, що значно спрощує процес створення гри навіть однією людиною.

Головними інструментами Unity та можливостями, що значною мірою полегшили процес розробки, можна назвати наступні:

- Префаби(prefabs)[36] – механізм Unity для створення, зберігання, налаштування та перевикористання ігрових об’єктів (GameObjects[37]) з усіма їхніми компонентами та властивостями. Завдяки їм у розробленій грі є певна кількість унікальних персонажів-ворогів, екземпляри яких можна швидко створювати під час ігрового процесу.
- Скриптові об’єкти (ScriptableObjects[38]) – контейнери даних, що є спільними для ігрових об’єктів певного типу. Завдяки ним можливо налаштовувати такі важливі складові ігор як параметри внутрішньої економіки, характеристики персонажів, параметри анімацій та інше. Ці значення легко змінювати, користуючись зручним інтефрейсом, що робить процес балансування гри більш зручним та швидким.
- Сітка(Grid)[39] – інструмент Unity, що допомагає вирівнювати ігрові об’єкти на основі вибраного макета та зручно працювати з координатами об’єктів ігрового світу, надаючи для них відповідні координати на сітці. Цей інструмент Unity був важливим для

створення двовимірної регулярної квадратної сітки, що стала простором пошуку шляху для ігрових сутностей. До того ж співставлення між клітинами сітки та вузлами, що використовує алгоритм A*, дозволило перетворити шлях, побудований алгоритмом, на послідовність точок маршруту в ігровому світі.

- Tilemap[40] – система, що зберігає та обробляє спеціальні ресурси плитки (tiles), з яких можна будувати двовимірні ігрові рівні. Tilemap в поєднанні з сіткою дав можливість створювати гарні ігрові рівні і зручно налаштовувати навігацію для персонажів гри. Плитки з tilemap розташовуються на ігровій сітці і можуть бути перевірені під час проходження по ним ігрових сутностей. Спеціальні ваги та властивості зробили проходження цих плиток більш або менш пріоритетним, або ж взагалі заборонили його для ігрових персонажів..

3.2 Огляд готового продукту

3.2.1 Загальний огляд гри

Результатом розробки стала комп’ютерна гра Royal Bastion у жанрі tower defense, що є піджанром жанру стратегії. Розроблений продукт адаптовано під операційну систему Windows і спрямовано на аудиторію фанатів таких ігор як Plants vs Zombies та Kingdom Rush.

Royal Bastion переймає основні механіки відомих представників жанру Tower Defense. Серед них варто виокремити:

- Зосередженість гравця на захисті головної споруди. Основною метою гри є захист замку від атак армії ворогів. Існує два головних підходи до організації головної мети в іграх жанру Tower Defense - захист головної споруди та стримування ворогів від досягнення певної точки в ігровому світі, Royal Bastion зорієнтовано на перший.

- Розміщення захисних споруд. Гравець може будувати вежі, стіни та шипи для захисту свого замку. Дана механіка є обов’язковою для ігор жанру tower defense.
- Отримання винагороди за знищення ворогів. Знищення ворогів (крипів) винагороджує гравця ігровою валютою, яка потрібна для будівництва нових захисних споруд. Дана механіка є широко розповсюдженою і дозволяє створити прогресію гравця у реальному часі, а також додати нову складову стратегії – планування витрат.
- Різновиди ворогів. Гра містить декілька видів ворогів з відмінними одне від одного характеристиками та зовнішнім виглядом. Дана особливість притаманна найбільш популярним іграм жанру tower defense і дозволяє підвищити складність і цікавість стратегічної складової гри, змусивши гравця адаптувати свою тактику для боротьби з різними ворогами.

Royal Bastion також має набір певних особливостей, що вирізняють його серед більшості ігор даного жанру:

- Налаштовуваність ігрового середовища. Ігрове середовище в Royal Bastion будується з клітин, що мають певні властивості: доступність для переміщення та будівництва, заблокованість для будівництва, заборонена на переміщення. Завдяки комбінуванню цих типів клітин у Royal Bastion стає можливим як створення стандартних рівнів з лабіринтоподібними маршрутами для ворогів та окремими точками для розміщення ігрових споруд, так і більш цікавих відкритих рівнів, де гравець має безліч варіантів для розміщення своєї оборони, а вороги здатні маневрувати та обходити перешкоди.
- Наявність атрибутів і супротивів. Важливою складовою гри є нанесення захисними спорудами шкоди персонажам ворогів. У Royal Bastion шкода має атрибут, що позначає її тип. Різні вороги мають супротив до певних типів шкоди, тому гравець повинен знаходити

баланс між типами шкоди, які мають його башти, щоб вправно знищувати хвилі крипів і проходити ігрові рівні.

- Розумна навігація ворогів. У Royal Bastion вороги використовують увесь доступний для переміщення простір, щоб побудувати найшвидший та найбезпечніший шлях до замку. Вороги враховують довжину шляху та шкідливі для них атрибути шкоди, знаходячи маршрут, що доведе їх до цілі якнайшвидше і з мінімальними втратами здоров'я. Така ігрова складова робить стратегію гравця важливішою, заохочуючи його будувати оборону без наскрізних дір.
- Адаптивна спеціалізація башт. Башти є головними атакуючими будівлями для гравця. На відміну від більшості ігор в жанрі tower defense, у Royal Bastion башти не мають наперед визначених ігрових характеристик. Гравець власноруч наймає бойові одиниці на башту, кожна з яких має особливі характеристики та атрибут атаки, що дозволяє динамічно визначати спеціалізацію башти та робити її корисною для конкретного стану гри.

Важливою особливістю гри також є візуальний стиль, усі компоненти якого були створені спеціально для Royal Bastion. Гра має 4 основних екрани, серед яких головне меню, екран вибору рівнів, екран туторіалу та головний ігровий екран, на якому відбувається ігровий процес. Кожен із екранів гри намагається слідувати за унікальним ігровим стилем і створювати враження від гри як від цілісного продукту.

рис. 15Екран головного меню гри

рис. 16 Екран туторіалу з гри

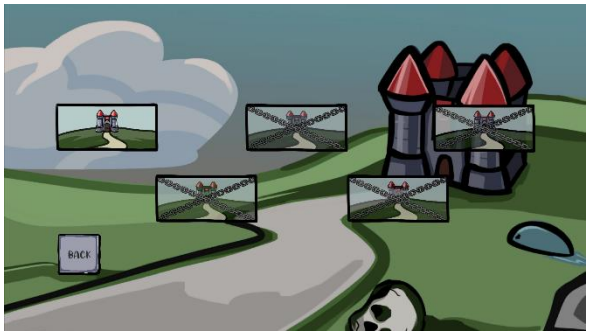


рис. 18 Екран вибору рівнів у грі

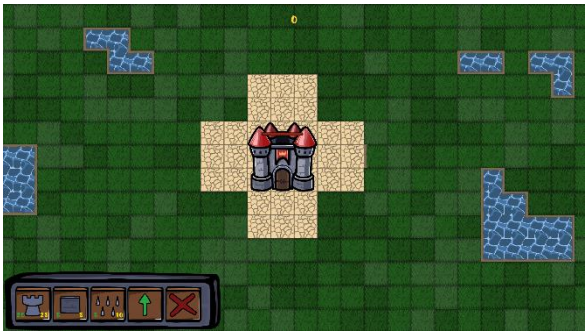


рис. 17 Фрагент одного з рівнів гри

3.2.2 Головні ігрові системи. Pathfinder

Під час розробки продукту було розділено логіку його роботи на декілька окремих ігрових систем, що є головними компонентами готової гри. Головними ігровими системами є :

- Inputs – система опрацювання сигналів від користувача, що надає йому можливість взаємодіяти з ігровим середовищем і брати участь в ігровому процесі.
- BuildingSystem – система, що відповідає за розміщення, знищення та покращення захисних споруд. Ця система працює з об’єктами таких споруд як башти, стіни та шипи, перевіряючи їх вартість та розташування, а також опрацьовуючи усі можливі у грі дії з цими об’єктами.
- MoneySystem – система, що веде підрахунок коштів гравця, які він може витрачати на створення захисних споруд.
- SpawnerSystem – ігрова система, що здатна створювати персонажів ворогів в заданих місцях ігрового світу, виконуючи їх попередню ініціалізацію зі стартовими параметрами.
- BattleWaveSystem – система, що координує перебіг гри. Вона дає команди системі створення ворогів, визначаючи кількість, тип та час створення ворогів на ігровому полі. Також ця система відповідає за закінчення гри у разі знищення хвилею ворогів замку гравця (поразка) або повного знищення усіх ворогів гравцем (перемога).

- SoundSystem – система, що керує налаштуваннями звуків, дозволяючи гравцю підлаштовувати його показники під комфортний для себе рівень гучності.
- SceneManager – система, що керує зміною поточної сцени. Завдяки цій системі гравець може переключатися між меню та ігровими рівнями.

Розумна навігація ворогів є однією з головних механік. Для реалізації даного функціоналу було реалізовано алгоритм знаходження шляху A*, який було розглянуто у пункті 2.2. Ігрові персонажі ворогів використовують результати роботи даного алгоритму для знаходження шляху до замку гравця.

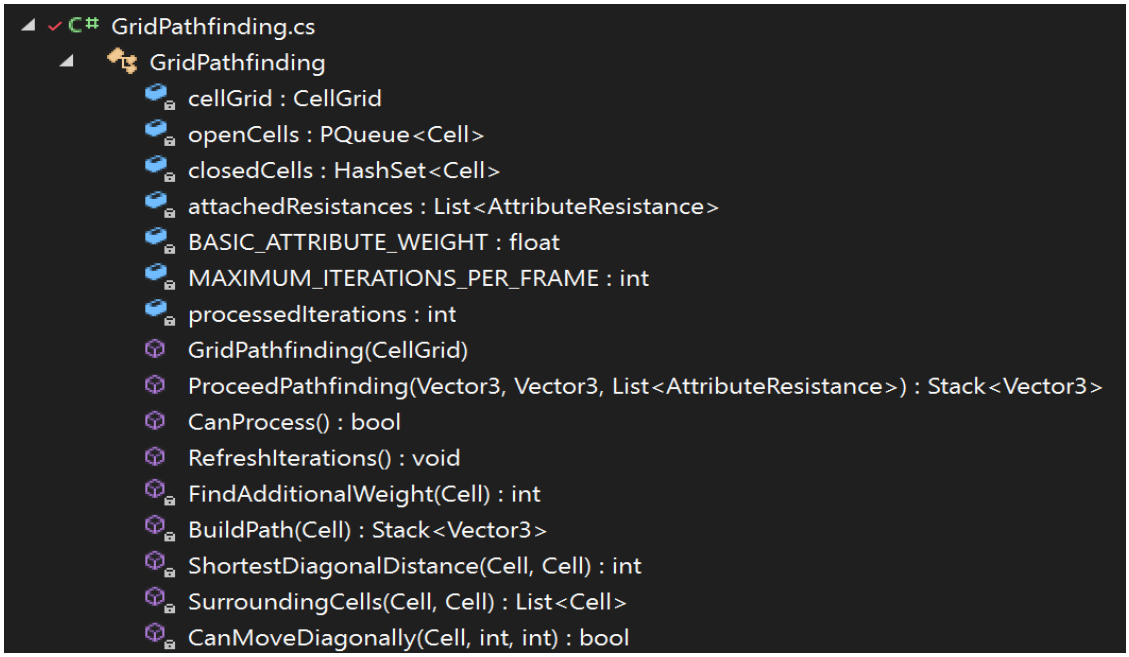


рис. 19 Структура pathfinder-класу

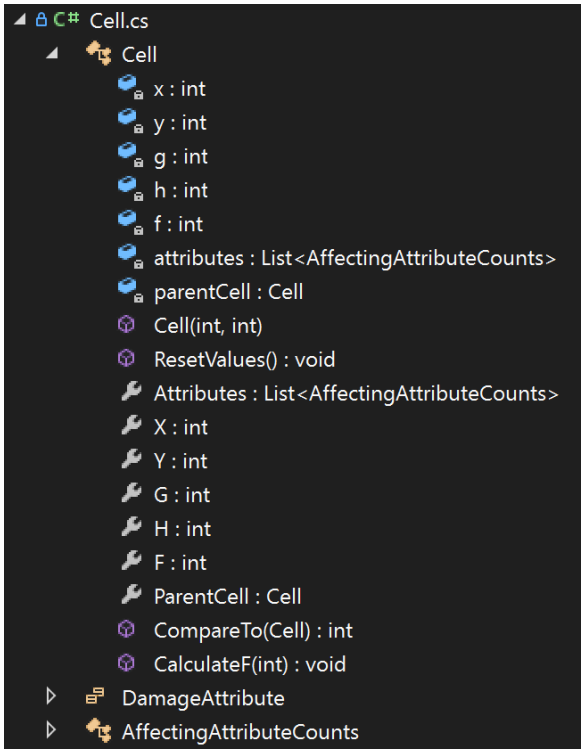


рис. 20 Структура класу, що описує клітину на сітці для пошуку шляху

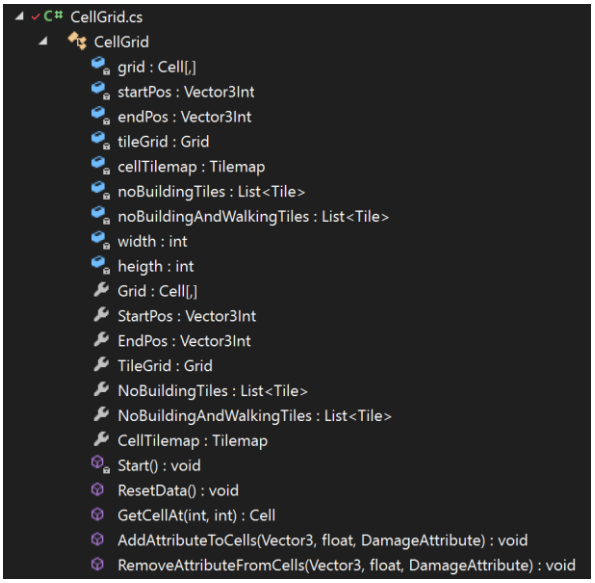


рис. 21 Структура класу, що описує сітку ігрового світу, на якій pathfinder намагається знайти шлях

GridPathdinding є класом, що здійснює знаходження шляху в ігровому світі. Основними методами цього класу є ProceedPathfinding та BuildPath. Перший з них опрацьовує клітини з сітки, доки не дійде до клітини призначення. Потім він передає її до методу BuildPath, щоб побудувати оптимальний шлях від початкової до кінцевої клітини. Також наявні методи, що інкапсулюють окремі функціональні блоки і не допускають надмірного розростання основних методів. Серед них:

- ShortestDiagonalDistance – метод для знаходження діагональної відстані між точками, необхідний для розрахування значення h.
- SurroundingCells – метод для знаходження усіх валідних клітин-сусідів для заданої.
- CanMoveDiagonally – метод, потрібний для заборони діагонального переміщення біля клітин, недоступних для руху, щоб запобігти потраплянню на їх територію.

- FindAdditionalWeigths – метод, що вираховує додаткову вагу, яку потрібно додати до значення f клітини, враховуючи усі атрибути шкоди, які діють на цю клітинку. Таким чином вороги можуть оминати найбільш небезпечні для себе місця.
- CanProcess та RefreshIterations – методи оптимізації, що обмежують кількість послідовних перерахувань шляху за один кадр ігрового процесу. Цей функціонал став додатковою оптимізацією, щоб запобігти можливим проблемам зі швидкодією у разі великої кількості агентів навігації.

Також для роботи класу-патфайндера було створено окремі класи Cell та CellGrid, що представляють собою клітину та сітку відповідно. Сітка є простором пошуку для алгоритму (аналог графа), тоді як клітина є аналогом вузла, послідовність яких і створює готовий шлях.

В результаті роботи було створено набір самостійних систем, що керують процесом гри, та структурований механізм знаходження шляху, який реалізовує алгоритм A*.

Висновки

У процесі розробки комп’ютерної гри було детально досліджено жанр стратегії з його піджанром tower defense. Також було проаналізовано використання pathfinding-алгоритмів в іграх цих жанрів та розглянуто основні типи алгоритмів знаходження шляху, досліджено їх відомих представників.

Після проведеного дослідження було обрано алгоритм знаходження шляху A* і наведено перелік його переваг для використання в ігровій розробці, які зробили його найкращим вибором для реалізації навігації ігрових сутностей у готовому проєкті. Також було використано можливості рушія Unity та мови C# і таким чином створено гру в жанрі tower defense під назвою Royal Bastion, в яку було закладено як унікальні, так і відомі ігрові механіки, розумну навігацію ігрових персонажів та цікавий авторський візуальний стиль, що доповнює автентичність ігрового продукту.

Список літератури

1. What is a Strategy Game?. *TheGameGuru.Me*. URL: https://troublmaker.wordpress.com/games-ive-played/what-is-a-strategy-game/

2. Hosch W. L. Electronic strategy game | History & Examples. *Encyclopedia Britannica*. URL: https://www.britannica.com/topic/electronic-strategy-game#ref1080973

[3] What is a Turn-based Game? | Dreams Quest. *Dreams Quest Main*. URL: https://dreams.quest/post/what-is-a-turn-based-game

[4] Hoekstra K. What Is an RTS game? A Guide To Real-Time Strategy. *History Hit*. URL: https://www.historyhit.com/gaming/what-is-an-rts-game-a-guide-to-real-time-strategy/

[5] Sato H. The impact of the RTS genre: Dune II. *Medium*. URL: https://medium.com/@vladkardash/the-impact-of-the-rts-genre-dune-ii-78a7f8aa6d21

[6] 4X games. *Wayback Machine*. URL: https://web.archive.org/web/20080618074603/http://www.mobygames.com/game-group/4x-games

[7] Scorched Parabolas: A History of the Artillery Game | Armchair Arcade. *Wayback Machine*. URL: https://web.archive.org/web/20160305113923/http://armchairarcade.com/neo/node/427

[8] Cox M. Spawn Point: What on earth is an auto battler?. *Rock Paper Shotgun*. URL: https://www.rockpapershotgun.com/what-is-an-auto-battler

[9] WePlay Holding. What are multiplayer online battle arena games?. *WePlay Holding*. URL: https://weplayholding.com/blog/what-are-multiplayer-online-battle-arena-games/

[10] What is an MMORTS? A Beginner`s Guide to War Strategy Games. *plarium.com*. URL: https://plarium.com/en/blog/mmorts-games/

[11] MasterClass. Tower Defense Game Genre: 6 Characteristics of TD Games. *MasterClass*. URL: https://www.masterclass.com/articles/tower-defense-game-video-game-guide

[12] Computational intelligence and tower defence games / P. Avery et al. *2011 IEEE Congress on Evolutionary Computation (CEC)*, New Orleans, LA, USA, 5–8 June 2011. 2011. URL: https://doi.org/10.1109/cec.2011.5949738

[13] Barnouti N. H., Al-Dabbagh S. S. M., Sahib Naser M. A. Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm. *Journal of Computer and Communications*. 2016. Vol. 04, no. 11. P. 15–25. URL: https://doi.org/10.4236/jcc.2016.411002

[14] A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games / S. R. Lawande et al. *Applied Sciences*. 2022. Vol. 12, no. 11. P. 5499. URL: https://doi.org/10.3390/app12115499

[15] Mughees Ur Rehman. What is Uninformed Search Algorithm in AI?. *Educative*. URL: https://www.educative.io/answers/what-is-uninformed-search-algorithm-in-ai

[16] Informed Search Algorithms in AI - Javatpoint. *www.javatpoint.com*. URL: https://www.javatpoint.com/ai-informed-search-algorithms

[17] Vinicius Fulber-Garcia. Heuristics vs. Meta-Heuristics vs. Probabilistic Algorithms | Baeldung on Computer Science. *Baeldung on Computer Science*. URL: https://www.baeldung.com/cs/heuristics-vs-meta-heuristics-vs-probabilistic-algorithms#:~:text=Similar%20to%20heuristics,%20metaheuristics%20aim,as%20discussed%20for%20the%20heuristics

[18] Breadth First Search or BFS for a Graph - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[19] Time and Space Complexity of Breadth First Search (BFS) - GeeksforGeeks. *GeeksforGeeks*.

URL: https://www.geeksforgeeks.org/time-and-space-complexity-of-breadth-first-search-bfs/

[20] Prateek Garg. Depth First Search. *Hackerearth*.

URL: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial

[21] Graph Algorithms. *O'Reilly Online Learning*.

URL: https://www.oreilly.com/library/view/graph-algorithms/9781492047674/ch04.html#:~:text=evaluated%20before%20Colchester.-,Depth%20First%20Search,along%20that%20path%20before%20backtracking.C%20else%20by%20backtracking.

[22] Time and Space Complexity of Depth First Search (DFS) - GeeksforGeeks.

URL: https://www.geeksforgeeks.org/time-and-space-complexity-of-depth-first-search-dfs/

[23] Dijkstra's Algorithm. *Programiz*.

URL: https://www.programiz.com/dsa/dijkstra-algorithm

[24] Minhang Zhou, Nina Gao. Research on Optimal Path based on Dijkstra Algorithms. *Advances in Computer Science Research*. 2019. Vol. 87. P. 884–892.

URL: https://www.atlantis-press.com/proceedings/icmeit-19/55917280

[25] Time and Space Complexity of Dijkstra's Algorithm - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/time-and-space-complexity-of-dijkstras-algorithm/

[26] Xiao Cui, Hao Shi. A*-based Pathfinding in Modern Computer Games. *IJCSNS International Journal of Computer Science and Network Security*. 2011. Vol. 11, no. 1. P. 125–130.

URL: https://www.researchgate.net/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games

[27] Hart P., Nilsson N., Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. 1968. Vol. 4, no. 2. P. 100–107.
URL: https://doi.org/10.1109/tssc.1968.300136
[28] Graham Cox. A* Pathfinding Algorithm | Baeldung on Computer Science. *Baeldung on Computer Science*.
URL: https://www.baeldung.com/cs/a-star-algorithm
[29] Anthony Stentz. The focussed D* algorithm for real-time replanning. *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*. 1995. Vol. 2. P. 1652–1659.
URL: https://dl.acm.org/doi/10.5555/1643031.1643113
[30] Reeves M. C. An Analysis of Path Planning Algorithms Focusing on A* and D* : Master of Science. Dayton, 2019. 48 p.
URL: http://rave.ohiolink.edu/etdc/view?acc_num=dayton1557245975528397
[31] Dr Rakesh Kumar, Mahesh Kumar. Exploring Genetic Algorithm for Shortest Path Optimization in Data Networks. *Global Journal of Computer Science and Technology*. 2010. Vol. 10. P. 8–12.
URL: https://globaljournals.org/GJCST_Volume10/2-Exploring-Genetic-Algorithm-for-Shortest-Path-Optimization-in-Data-Networks.pdf
[32] Leigh R., Louis S. J., Miles C. Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms. *2007 IEEE Symposium on Computational Intelligence and Games*, Honolulu, HI, USA, 1–5 April 2007. 2007. P. 72–79.
URL: https://doi.org/10.1109/cig.2007.368081
[33] Rishiwal V., Yadav M., Arya K. V. Finding optimal paths on terrain maps using ant colony algorithm. *International journal of computer theory and engineering*. 2010. P. 416–419.
URL: https://doi.org/10.7763/ijcte.2010.v2.178

[34] Rafiq A., Asmawaty Abdul Kadir T., Normaziah Ihsan S. Pathfinding algorithms in game development. *IOP conference series: materials science and engineering*. 2020. Vol. 769. P. 012021.
URL: https://doi.org/10.1088/1757-899x/769/1/012021
[35] A* pathfinding algorithm. *Growing with the Web*.
URL: https://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html
[36] Unity Technologies. Unity - manual: prefabs. *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
URL: https://docs.unity3d.com/Manual/Prefabs.html
[37] Unity Technologies. Unity - Scripting API: GameObject. *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
[38] Unity Technologies. Unity - Manual: ScriptableObject. *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html
[39] Unity Technologies. Unity - Manual: grid component reference. *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
URL: https://docs.unity3d.com/Manual/class-Grid.html
[40] Unity Technologies. Unity - Manual: tilemap component reference. Unity - Manual: Unity User Manual 2022.3 (LTS).
URL: https://docs.unity3d.com/Manual/class-Tilemap.html