МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

Факультет менеджменту

Кафедра економічної кібернетики, комп'ютерних наук
та інформаційних технологій

Кваліфікаційна наукова
праця на правах рукопису

ЄМЕЦЬ Анатолій Олегович

УДК 004.896:004.451.1:794(043.2)

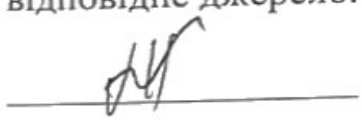
КВАЛІФІКАЦІЙНА РОБОТА

ДОСЛІДЖЕННЯ ТА ІМПЛЕМЕНТАЦІЯ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ ІГРОВОГО ПРОЦЕСУ В 2D СТРАТЕГІЯХ

Спеціальність 122 «Комп'ютерні науки»
Галузь знань – 12 «Інформаційні технології»

Подається на здобуття освітнього ступеня бакалавра

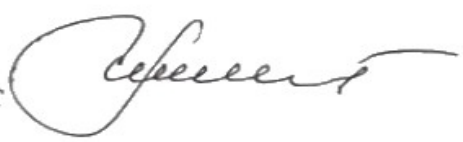
Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.



Ємець А. О.

Науковий керівник:
Пархоменко Олександр Юрійович,
кандидат фізико-математичних наук, доцент

Завідувач кафедри:
Тищенко Світлана Іванівна,
кандидат педагогічних наук, доцент



Миколаїв–2025

АНОТАЦІЯ

Ємець А. О. Використання алгоритмів штучного інтелекту для ігор жанру 2D-стратегій. – Кваліфікаційна наукова праця на правах рукопису.

Робота на здобуття освітнього ступеня бакалавра за спеціальністю 122 «Комп'ютерні науки». – Миколаївський національний аграрний університет, Миколаїв, 2025.

Об'єктом дослідження є алгоритми штучного інтелекту, що використовуються в 2D-стратегічних іграх для оптимізації ігрового процесу. У кваліфікаційній роботі досліджено алгоритми штучного інтелекту для оптимізації ігрового процесу в 2D-стратегічних іграх. Проведено аналіз традиційних алгоритмів пошуку та прийняття рішень, таких як A*, мінімакс із альфа-бета відсіканням, а також сучасних підходів машинного навчання, включаючи Q-learning.

Предмет дослідження – алгоритми штучного інтелекту для оптимізації ігрового процесу в реальному часі.

Метою роботи є дослідження та впровадження методів покращення ігрового процесу в 2D стратегічних іграх, забезпечивши ефективну поведінку ігрових агентів.

Реалізація алгоритмів здійснена на ігровому рушії Unity. Запропоновані методи дозволяють ігровим агентам ефективно шукати оптимальні маршрути, ухвалювати стратегічні рішення та адаптуватися до змінних умов гри. У процесі тестування було підтверджено ефективність використаних алгоритмів, зокрема зниження обчислювальних витрат за рахунок оптимізації пошукових процесів та покращення якості стратегічного моделювання NPC.

Робота складається з фахового розділу та спеціальної частини. Пояснювальна записка містить вступ, три розділи, висновки та додатки.

У першому розділі розглядається теоретичний аспект алгоритмів пошуку шляху, їх класифікація та порівняння. У другому розділі проведено аналіз існуючих алгоритмів штучного інтелекту та обґрунтовано вибір оптимальних методів.

У третьому розділі описано процес реалізації алгоритмів у розробленій грі, їх тестування, оптимізацію та оцінку ефективності.

Кваліфікаційна робота викладена на 52 сторінках, містить 3 таблиці, 11 рисунків, список використаних джерел включає 30 найменувань.

Ключові слова: 2D-стратегія, штучний інтелект, A*, мінімакс, Q-learning, алгоритми прийняття рішень, Unity, NPC, оптимізація, стратегічний AI.

ANNOTATION

Yemets A. O. Using artificial intelligence algorithms for 2D strategy games. – Qualification scientific work in the form of a manuscript.

Work for the degree of bachelor in specialty 122 “Computer Science”. – Mykolaiv National Agrarian University, Mykolaiv, 2025.

The object of the study is artificial intelligence algorithms used in 2D strategy games to optimize the gameplay. The qualification work investigates artificial intelligence algorithms for optimizing the gameplay in 2D strategy games. An analysis of traditional search and decision-making algorithms, such as A*, minimax with alpha-beta cutoff, as well as modern machine learning approaches, including Q-learning, was conducted.

The subject of the study is methods and algorithms for making decisions for NPCs taking into account the optimization of the gameplay.

The aim of the work is to study and implement AI methods to increase the autonomy and adaptability of NPCs in strategy games.

The implementation of the algorithms was carried out on the Unity game engine. The proposed methods allow game agents to effectively search for optimal routes, make strategic decisions, and adapt to changing game conditions. The testing process confirmed the effectiveness of the algorithms used, in particular, reducing computational costs by optimizing search processes and improving the quality of NPC strategic modeling.

The work consists of a technical section and a special part. The explanatory note contains an introduction, three sections, conclusions, and appendices.

The first section considers the theoretical aspect of path finding algorithms, their classification, and comparison. The second section analyzes existing artificial intelligence algorithms and justifies the choice of optimal methods.

The third section describes the process of implementing algorithms in the developed game, their testing, optimization, and effectiveness assessment.

The qualification work is presented on 52 pages, contains 3 tables, 11 figures, the list of sources used includes 30 names.

Keywords: 2D-strategy, artificial intelligence, A*, minimax, Q-learning, decision-making algorithms, Unity, NPC, optimization, strategic AI.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1 ОСНОВИ ЗАСТОСУВАННЯ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ В ІГРОВИХ 2D СТРАТЕГІЯХ 8
1.1. Огляд основних типів ігор-стратегій та ролі штучного інтелект 8
1.2. Алгоритми прийняття рішень для NPC (неігрових персонажів) в стратегічних іграх 10
1.3. Методи планування та прийняття рішень: пошук шляху, мінімакс 15
1.4. Основи машинного навчання для навчання ігрових агентів 17
РОЗДІЛ 2 АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІГОР ТА ВИБІР ОПТИМАЛЬНИХ 20
2.1. Аналіз алгоритмів пошуку та прийняття рішень у 2D стратегічних іграх 20
2.2. Порівняння класичних методів штучного інтелекту та підходів машинного навчання 22
2.3. Вибір алгоритмів для реалізації в грі: переваги та недоліки 24
2.4. Постановка завдання для ігрового процесу та вибору стратегії поведінки NPC 26
РОЗДІЛ 3 РЕАЛІЗАЦІЯ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ В 2D СТРАТЕГІЧНІЙ ГРІ 31
3.1. Проектування архітектури ігрового процесу та інтеграція AI-алгоритмів 31
3.2. Реалізація алгоритму пошуку шляху (A*) для ігрових агентів 34
3.3. Імплементація алгоритмів прийняття рішень для NPC: мінімакс та Qlearning 37
3.4. Тестування поведінки ігрових агентів у реальному часі: симуляції та аналіз ефективності 41
3.5 Оптимізація алгоритмів для покращення продуктивності та ігрового досвіду 43
ВИСНОВКИ 48
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
ДОДАТОК А Спрайти юнітів 55

ВСТУП

Штучний інтелект (ШІ) значно вплинув на розвиток комп'ютерних наук, зокрема на створення відеоігор. У сучасних 2D-стратегіях ефективне керування неігровими персонажами (NPC) та оптимізація ігрового процесу є критичними для забезпечення захопливого та реалістичного геймплею. Використання методів штучного інтелекту дозволяє створювати адаптивні ігрові світи, де NPC можуть навчатися, ухвалювати рішення на основі аналізу ситуації та ефективно взаємодіяти з гравцем.

Актуальність дослідження зумовлена необхідністю розробки оптимальних алгоритмів для 2D стратегічних ігор, що забезпечують швидке та коректне ухвалення рішень NPC у реальному часі. Існуючі алгоритми часто мають обмежену адаптивність, що може призводити до передбачуваної поведінки персонажів і зниження динамічності гри. Крім того, висока обчислювальна складність пошукових алгоритмів, таких як A*, може спричиняти затримки у великомасштабних сценаріях, а класичні алгоритми ухвалення рішень, зокрема мінімакс, не завжди враховують непередбачувані зміни у грі. Отже, актуальним є дослідження методів оптимізації ШІ та інтеграція сучасних підходів машинного навчання для підвищення продуктивності та адаптивності NPC.

Об'єктом дослідження є штучний інтелект у стратегічних відеоіграх, а предметом – алгоритми ухвалення рішень для NPC з урахуванням оптимізації ігрового процесу. У роботі розглядаються та реалізуються пошукові алгоритми, такі як A*, алгоритми ухвалення рішень, зокрема мінімакс із альфа-бета відсіканням, а також методи машинного навчання, включно з Q-learning, для вдосконалення поведінки NPC.

Метою дослідження є розробка та імплементація алгоритмів штучного інтелекту, які покращують якість взаємодії NPC із середовищем та сприяють створенню більш реалістичного ігрового досвіду. Зокрема, робота передбачає порівняння традиційних алгоритмів пошуку та ухвалення рішень із

сучасними методами машинного навчання, що дозволить визначити оптимальні підходи для стратегічних 2D-ігор.

Для досягнення поставленої мети вирішуються такі завдання:

- аналіз основних типів стратегічних 2D-ігор та особливостей застосування ШІ;
- дослідження алгоритмів ухвалення рішень та їх впливу на поведінку NPC;
- вибір та імплементація ефективних алгоритмів для пошуку шляху та ухвалення рішень у реальному часі;
- тестування та оптимізація алгоритмів для підвищення продуктивності та реалістичності ігрового процесу.

Методи дослідження включають алгоритми пошуку (A*, мінімакс, альфа-бета відсікання); машинне навчання для поведінки ігрових агентів (Q-learning, нейронні мережі). Практична реалізація алгоритмів здійснюється в ігровому рушії Unity, що дозволяє оцінити їхню ефективність у реальних ігрових сценаріях.

Практична значущість роботи полягає у можливості впровадження розроблених алгоритмів у сучасні відеоігри, що сприятиме покращенню взаємодії NPC із гравцями та підвищенню рівня занурення в ігровий процес. Крім того, результати дослідження можуть бути використані для подальших розробок у сфері штучного інтелекту та комп’ютерних ігор.

Основні положення роботи апробовані на 37-мій студентській науково-теоретичній конференції «Участь молоді у розбудові агропромислового комплексу країни», м. Миколаїв, 18-19 березня 2025 р. Тема доповіді: Роль штучного інтелекту в оптимізації процесів управління агропромисловим комплексом через моделювання у 2D стратегічних іграх.

Структура та обсяг роботи. Дипломна робота складається з анотації на 2 сторінках, вступу, трьох розділів, висновку, списку використаних джерел з 30 найменувань. Основна частина роботи становить 52 с., серед яких 11 рис., 3 табл.

РОЗДІЛ 1
ОСНОВИ ЗАСТОСУВАННЯ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ В ІГРОВИХ 2D СТРАТЕГІЯХ

1.1. Огляд основних типів ігор-стратегій та ролі штучного інтелект

Ігри відіграли ключову роль у стрімкому розвитку штучного інтелекту (ШІ). Виступаючи як контрольовані, економічно ефективні та реалістичні симулятори, вони створюють віртуальні світи, які слугують середовищем для вивчення ШІ. У таких світах можна моделювати широкий спектр завдань із реального життя, зокрема прийняття рішень у реальному часі , планування, розробку та освітні процеси . Завдяки цьому ігри стали природним випробувальним полігоном для перевірки здатності, універсальності, надійності та безпеки ШІ.

Ігрові платформи стали ідеальним середовищем для дослідників, де можна вивчати, аналізувати, оцінювати та експериментувати з різними ідеями у контрольованих та безпечних умовах. Вони також дозволяють проводити експерименти за принципом "що-якщо", які можуть бути небезпечними або складними для реалізації в реальному світі. Крім того, ігри надихають дослідників на постановку нових питань, стимулюючи подальший розвиток галузі.

Ігрові стратегії займають важливе місце серед жанрів комп’ютерних ігор, оскільки вони сприяють розвитку аналітичного мислення, навичок планування та прийняття рішень. Такі ігри, як "Civilization" чи "StarCraft", допомагають удосконалювати здатність прогнозувати наслідки дій, розробляти довгострокові стратегії та приймати оперативні рішення в умовах обмежених ресурсів. Цей жанр охоплює різноманітні типи ігор, від класичних покрокових стратегій до реалістичних симуляцій у реальному часі. Залежно від ігрової механіки та цільової аудиторії, такі ігри можуть мати різний рівень складності й інтерактивності, але однією з ключових

характеристик є необхідність розробки ефективної штучної поведінки для управління ігровими агентами.

Штучний інтелект виконує важливу роль у створенні цікавого та реалістичного ігрового процесу. Наприклад, у грі "Total War" AI відповідає за управління бойовими одиницями та тактичними маневрами, а в "Age of Empires" забезпечує динамічне будівництво, збір ресурсів та атаки на базу супротивника, створюючи виклик для гравця. У стратегіях основне завдання AI полягає у забезпеченні динамічної поведінки NPC (неігрових персонажів), які можуть адаптуватися до дій гравця, приймати стратегічні рішення та виконувати задані цілі. У реалістичних симуляціях штучний інтелект використовується для управління великими групами віртуальних персонажів, які повинні координувати свої дії для досягнення спільних результатів. У покрокових стратегіях AI забезпечує оптимізацію ходу NPC, базуючись на прогнозах майбутніх дій гравця.

Типи стратегічних ігор можна умовно розділити на кілька категорій: покрокові стратегії, стратегії в реальному часі, економічні симуляції та гібридні жанри. Гібридні стратегії, такі як "Total War", поєднують елементи покрокового управління глобальними процесами із реалістичними битвами в реальному часі, створюючи унікальний досвід для гравців. У покрокових стратегіях, таких як „Civilization“, гравці послідовно виконують свої дії, що дозволяє AI використовувати алгоритми прогнозування та аналізу ситуацій для прийняття оптимальних рішень. Стратегії в реальному часі, наприклад, „StarCraft“, вимагають швидких рішень і ефективного управління ресурсами, де AI застосовує методи евристичного пошуку та адаптивного навчання. Економічні симуляції, як-от „SimCity“, фокусуються на управлінні економічними процесами, і AI моделює складні системи, зважаючи на безліч взаємопов’язаних факторів.

У сучасних іграх роль AI значно зросла завдяки прогресу в галузі машинного навчання та обчислювальної потужності. Зокрема, використання глибоких нейронних мереж дозволило значно поліпшити поведінку NPC,

тоді як методи підкріплювального навчання, такі як Deep Q-Learning, сприяють створенню адаптивних агентів, здатних навчатися оптимальним стратегіям на основі досвіду. Також технології природної мови, як GPT, інтегруються для створення більш реалістичних сценаріїв взаємодії. Це дозволяє створювати більш реалістичних ігрових агентів, здатних до самонавчання, адаптації та прийняття нестандартних рішень. Розробка AI у 2D стратегіях зосереджена на оптимізації ігрового процесу, щоб гравці відчували виклик, а гра залишалася цікавою та збалансованою.

Подальший розвиток стратегічних ігор та їх складності прямо залежить від удосконалення алгоритмів штучного інтелекту. Зокрема, можна очікувати активного використання таких технологій, як нейронні мережі для прогнозування дій гравців, навчання із використанням ігрових симуляторів для створення автономних агентів та інтеграція генеративних моделей для розробки динамічних сценаріїв та середовищ. Такі підходи дозволять зробити AI ще більш адаптивним та креативним, що суттєво збагатить ігровий досвід. Вивчення різних типів AI для цього жанру дозволяє зрозуміти, які підходи є найбільш ефективними для створення захоплюючого ігрового досвіду.

1.2. Алгоритми прийняття рішень для NPC (неігрових персонажів) в стратегічних іграх

Неігрові персонажі (англ. Non-Player Charactes) скорочено NPC – це персонажі, з яким гравці можуть взаємодіяти, але якими не можуть грати.

Неігрові персонажі (NPC) у стратегічних іграх виконують роль активних елементів, які додають динаміки, складності та реалізму до ігрового процесу. Їх поведінка визначається алгоритмами прийняття рішень, які забезпечують здатність реагувати на дії гравця, адаптуватися до змін у середовищі та досягати поставлених цілей. У цьому розділі розглянуто ключові алгоритми прийняття рішень, які широко використовуються для моделювання поведінки NPC у стратегічних іграх [1].

Класичні підходи до прийняття рішень базуються на правилах, заздалегідь визначених розробниками. Окрім гри StarCraft, ці підходи ефективно використовуються в Age of Empires, де NPC слідують чітко прописаним сценаріям для збирання ресурсів і будівництва структур. Це забезпечує передбачуваність і стабільність ігрового процесу, що є важливим для класичних стратегій. У грі StarCraft класичні алгоритми використовуються для визначення основних сценаріїв поведінки NPC, таких як збирання ресурсів, атака чи оборона бази. Такі сценарії легко масштабуються і забезпечують стабільність, що важливо для підтримки передбачуваного ігрового процесу. Дерева рішень є одним із популярних методів, що дозволяють NPC обирати дії, ґрунтуючись на наборі умов.

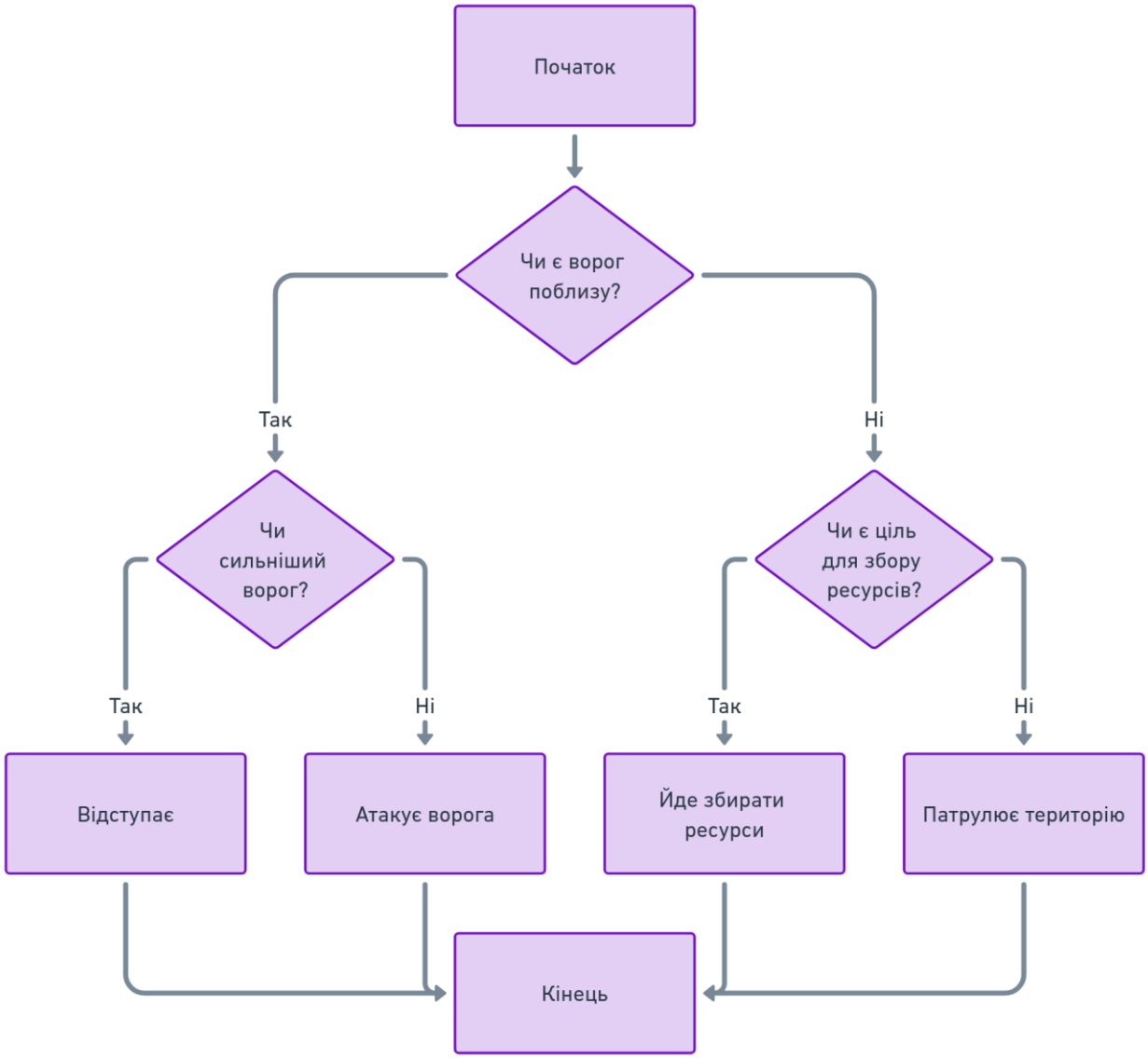


Рисунок 1.1. – Алгоритм прийняття рішень

Такі алгоритми прості у реалізації та надають високу передбачуваність поведінки, що важливо для підтримки стабільності ігрового процесу. Однак вони мають обмежену здатність до адаптації у складних сценаріях.

Іншим підходом є використання стано-орієнтованих машин (finite state machines, FSM), які дозволяють NPC змінювати свої дії залежно від поточного стану.

Кінцевий автомат, кінцевий автоматон, або машина станів (FSM, FSA, finite automaton, state machine) - це математична модель обчислень.

Це абстрактна машина, яка може перебувати в одному зі скінченної кількості станів у будь-який момент часу. Кінцевий автомат може переходити з одного стану в інший у відповідь на деякі вхідні дані; перехід з одного стану в інший називається переходом.

Кінцевий автомат визначається списком його станів, початковим станом і вхідними даними, які запускають кожен перехід.

FSM можуть бути ефективно інтегровані з іншими алгоритмами, такими як машинне навчання, для вирішення складніших сценаріїв. Наприклад, у грі Total War поєднання FSM із алгоритмами машинного навчання дозволяє NPC адаптувати свою поведінку залежно від стратегій гравця. Це підхід дає змогу забезпечити базову логіку для типових дій NPC, таких як оборона або атака, одночасно вдосконалюючи більш складні аспекти їхньої поведінки, як-от аналіз дій суперників чи адаптація до змін у реальному часі. Нейронні мережі можуть використовуватися для визначення найімовірнішого переходу між станами або для адаптації логіки FSM в реальному часі, враховуючи дії гравця чи змінні середовища. Це дозволяє поєднати структурованість FSM з адаптивністю сучасних методів машинного навчання, створюючи більш реалістичну ігрову поведінку. NPC може переходити між станами “атака”, “захист” та “патрулювання” залежно від ситуації на полі бою. FSM забезпечують більшу гнучкість порівняно з деревами рішень, але їх складність швидко зростає зі збільшенням кількості станів та переходів.

У складніших сценаріях застосовуються алгоритми пошуку, такі як A* або Dijkstra, які дозволяють NPC знаходити оптимальні шляхи до цілей на ігровій карті. A* є одним із найбільш поширених алгоритмів завдяки своїй ефективності та здатності враховувати евристичні оцінки, що дозволяє зменшити обчислювальну складність. Однак такі алгоритми фокусуються переважно на навігації і не враховують складніші аспекти прийняття рішень, такі як аналіз дій гравця чи інших NPC [2].

Машинне навчання та підкріплювальне навчання стають все більш популярними для створення адаптивних NPC, компенсуючи недоліки класичних методів, особливо у складних сценаріях, таких як адаптація до непередбачуваних змін середовища чи швидка реакція на дії гравця. Наприклад, у складних сценаріях, де NPC повинні адаптуватися до непередбачуваних змін у середовищі або приймати рішення на основі великих обсягів даних, традиційні підходи можуть бути недостатньо ефективними. Машинне навчання дозволяє NPC вчитися на основі попереднього досвіду, що значно підвищує їхню гнучкість і здатність адаптуватися до нових викликів. Алгоритми Q-learning та Deep Q-learning дозволяють NPC навчатися оптимальним стратегіям, отримуючи винагороду за успішні дії. Наприклад, NPC може навчитися обирати кращі позиції для атаки чи ухилятися від загроз. Ці методи забезпечують високу гнучкість і здатність до адаптації, але потребують значних обчислювальних ресурсів і ретельного налаштування процесу навчання.

Ще одним підходом є використання нейронних мереж для моделювання складної поведінки NPC. Наприклад, у багатокористувацьких стратегіях, таких як Dota 2, нейронні мережі використовуються для навчання ботів розуміти складні ігрові сценарії, аналізувати дії суперників і адаптувати свої стратегії. Крім рекурентних нейронних мереж (RNN), також застосовуються згорткові нейронні мережі (CNN) для обробки візуальної інформації, яка може включати стан ігрового поля чи карти. Така різноманітність підходів дозволяє NPC краще адаптуватися до змін і

забезпечувати більш природну взаємодію в ігровому середовищі. Наприклад, у грі, де NPC має взаємодіяти з динамічними елементами середовища та адаптуватися до дій гравця, нейронні мережі можуть компенсувати обмеження класичних методів. Вони здатні обробляти великий обсяг даних і виявляти приховані закономірності, що дозволяє NPC краще розуміти складні сценарії. Особливо це ефективно у багатокористувацьких іграх, де дії кожного гравця створюють непередбачуване середовище, яке потребує адаптивності та інтелектуальної реакції. Рекурентні нейронні мережі (RNN) дозволяють NPC враховувати історію подій, приймаючи рішення. Це особливо корисно у випадках, коли поведінка NPC повинна враховувати довгострокові наслідки своїх дій. Проте, як і у випадку з підкріплювальним навчанням, ці методи вимагають значних ресурсів для навчання.

Комбіновані підходи, які поєднують класичні методи та машинне навчання, також отримують все більше поширення. У грі Total War, класичні алгоритми, такі як FSM, відповідають за базові аспекти поведінки NPC, такі як перемикання між режимами "атака" та "оборона". Водночас елементи машинного навчання використовуються для вдосконалення складніших завдань, як-от аналіз і адаптація до стратегій гравця, що дозволяє NPC діяти розумніше в умовах, які виходять за межі початкових сценаріїв. FSM можуть використовуватися для загального керування поведінкою NPC, тоді як машинне навчання застосовується для оптимізації окремих дій, таких як ухилення від загроз або аналіз дій гравця.

Таким чином, вибір алгоритмів для прийняття рішень NPC залежить від складності гри, доступних ресурсів і бажаного рівня адаптивності NPC. Для стратегічних ігор, наприклад, жанру RTS або RPG, рекомендовано комбінувати методи, щоб враховувати специфіку жанру. Так, класичні алгоритми забезпечують стабільність і передбачуваність, тоді як машинне навчання дає змогу моделювати складні сценарії, адаптуючись до дій гравця. Наприклад, для багатокористувацьких ігор можна використовувати підхід, який поєднує FSM для основної логіки поведінки та нейронні мережі для

аналізу дій суперників, що забезпечить баланс між ефективністю та адаптивністю. Класичні методи забезпечують простоту та стабільність, тоді як машинне навчання надає більше можливостей для створення інтелектуальних та адаптивних агентів. Інтеграція цих підходів дозволяє створювати багатогранні ігрові системи, що відповідають сучасним вимогам індустрії.

1.3. Методи планування та прийняття рішень: пошук шляху, мінімакс

Методи планування та прийняття рішень є ключовими елементами для створення ефективних алгоритмів поведінки NPC у стратегічних іграх. Вони забезпечують фундамент для адаптивності й ефективності NPC, дозволяючи їм взаємодіяти з ігровим середовищем і приймати оптимальні рішення у складних сценаріях. У цьому розділі розглянемо кілька популярних підходів, таких як пошук шляху, мінімакс та його модифікації.

Пошук шляху є основою для забезпечення навігації NPC в ігровому середовищі, оскільки дозволяє створювати ефективні маршрути, враховуючи складність ландшафту, перешкоди та цілі поведінки персонажів. Це підґрунтя для багатьох ігрових механік, адже правильна навігація є критичною для природної взаємодії NPC з оточенням. Це забезпечує плавний і природний ігровий досвід, що робить цей підхід невід’ємним елементом багатьох жанрів відеоігор.

Одним із найпоширеніших алгоритмів у цій галузі є A*. Цей алгоритм використовує евристичні оцінки для визначення найоптимальнішого шляху між двома точками на карті. Завдяки своїй ефективності та простоті він став стандартом у багатьох іграх, зокрема у таких жанрах, як RTS та RPG. Наприклад, у грі Skyrim алгоритм A* використовується для навігації NPC складними ландшафтами, враховуючи безпечні зони та перешкоди.

Складні ландшафти, такі як гори, річки та густі ліси, створюють значні виклики для прокладання шляху. NPC адаптують свої маршрути, уникаючи

фізичних перешкод або зон із високими обчислювальними витратами. Наприклад, у грі The Witcher 3 складні географічні особливості вимагають врахування не лише фізичних бар'єрів, а й стратегічних аспектів, таких як засідки чи небезпечні території.

Для великих ігрових карт часто застосовують ієрархічні методи, які розбивають карту на менші регіони. Це дозволяє скоротити обчислювальні витрати, зберігаючи при цьому точність. Ієрархічний пошук шляху, як у грі Civilization, є прикладом оптимізації для управління великою кількістю NPC, демонструючи універсальність цього підходу.

Алгоритм мінімакс є класичним підходом до прийняття рішень у стратегічних іграх. Він дозволяє NPC прогнозувати дії суперника та будувати оптимальну стратегію. Мінімакс особливо корисний у іграх з нульовою сумою, таких як шахи чи шашки.

Модифікація алгоритму, відома як альфа-бета відсікання, дозволяє значно зменшити кількість вузлів дерева, які необхідно обробити. Це забезпечує швидшу роботу без втрати якості рішень. Наприклад, у грі StarCraft II використовується мінімакс із альфа-бета відсіканням для моделювання тактичних рішень NPC у реальному часі. Подібний підхід також застосовується в грі Age of Empires для управління бойовими юнітами під час складних сценаріїв битв. Мінімаксний алгоритм знаходить застосування у широкому спектрі ігор, де комп'ютер повинен приймати рішення, оптимальні йому, з невизначених дій гравця. Це можуть бути як класичні ігри, наприклад, шашки або аркади, так і прикладні випадки: вибір найкращої угоди, визначення поведінкової моделі для успішного проходження відбору або дії в ситуаціях, які неможливо передбачити заздалегідь.

Для складних ігор із великою кількістю змінних також застосовуються варіації мінімаксу, такі як MCTS (Monte Carlo Tree Search). Цей метод дозволяє NPC проводити випадкові симуляції, щоб визначити найкращі дії у невизначених умовах. У' у грі Total War MCTS використовується для оцінки

стратегій у битвах, зокрема у ситуаціях із множинними перемінними. Це дозволяє NPC аналізувати різні сценарії та приймати оптимальні рішення, враховуючи багатофакторність і складність умов.

У сучасних іграх часто комбінують методи планування шляху та прийняття рішень для досягнення максимального результату. Наприклад, у грі Dota 2 алгоритми пошуку шляху поєднуються з техніками прийняття рішень на основі машинного навчання, що дозволяє NPC адаптувати свої дії залежно від ситуації. Подібна інтеграція забезпечує універсальність і ефективність: від управління героями на полі бою до вибору оптимальної стратегії для команди.

Ще одним прикладом може слугувати гра XCOM, де алгоритми планування шляху враховують складність укриттів і ризик зустрічі з противником, поєднуючись із моделями передбачення поведінки ворога. Це значно підвищує динамічність ігрового процесу, дозволяючи NPC ефективно пересуватися і приймати стратегічно обґрунтовані рішення. Успішність роботи алгоритму штучного інтелекту Мінімакс оцінюватиметься на основі трьох параметрів: часової складності, просторової складності та кількості перемог над гравцем високого рівня.

У сучасному контексті, поєднання традиційних алгоритмів із машинним навчанням відкриває нові горизонти в розробці ігор. Наприклад, у грі Horizon Zero Dawn планування шляху NPC враховує поведінкові моделі та взаємодію з динамічними змінами у світі гри [3].

Планування шляху та алгоритми прийняття рішень є основою для створення інтелектуальних NPC. Завдяки ефективним алгоритмам, таким як A*, Dijkstra, мінімакс та MCTS, можливо забезпечити високу якість ігрового досвіду. Поєднання цих підходів із сучасними технологіями, наприклад, машинним навчанням, дозволяє створювати адаптивні, гнучкі та потужні системи, що відповідають потребам сучасних стратегічних ігор.

1.4. Основи машинного навчання для навчання ігрових агентів

Машинне навчання є фундаментальною складовою сучасних методів створення інтелектуальних ігрових агентів. Воно дозволяє системам адаптуватися до змін у середовищі, навчатися на основі досвіду та приймати оптимальні рішення. Завдяки цьому забезпечується створення реалістичної та інтерактивної поведінки персонажів, що значно збагачує ігровий процес.

Машинне навчання (ML) у розробці ігор означає застосування методів і алгоритмів ML для покращення різних аспектів розробки ігор, ігрової механіки та досвіду гравців. Алгоритми машинного навчання можна використовувати для динамічного створення ігрового контенту, включаючи рівні, карти, квести, предмети та персонажів. Моделі ML можуть вивчати наявний ігровий вміст, щоб створювати новий, унікальний контент, який відповідає дизайну гри та вподобанням гравця [4].

Машинне навчання включає кілька ключових методів: підкріплювальне, супервізоване та гібридне навчання.

Підкріплювальне навчання Підкріплювальне навчання дозволяє агентам досліджувати середовище та визначати оптимальні стратегії через систему винагород. Цей метод забезпечує ефективність у ситуаціях, коли точні моделі середовища відсутні. Наприклад, у грі AlphaGo цей підхід використано для досягнення високого рівня гри завдяки поєднанню глибоких нейронних мереж і алгоритмів планування ходів. Такий метод особливо корисний у стратегічних іграх із динамічними умовами.

Супервізоване навчання дозволяє навчати ігрових агентів на основі мічених даних. Це часто використовується для створення реалістичних NPC (некерованих персонажів), які імітують дії гравців чи реагують на певні події. Наприклад, у рольових іграх цей метод допомагає створювати NPC, які реалістично реагують на взаємодії, роблячи ігровий досвід глибшим [5].

Гібридні підходи поєднують елементи підкріплювального та супервізованого навчання, забезпечуючи ефективне вирішення складних завдань. У грі Dota 2, наприклад, такі методи допомагають агентам

одночасно аналізувати тактики супротивників і адаптувати власні стратегії в реальному часі.

Машинне навчання демонструє найбільшу ефективність у таких аспектах:

Навігація складними ландшафтами. У стратегіях з відкритим світом складні ландшафти, такі як гори чи густі ліси, створюють унікальні виклики для агентів. Підкріплювальне навчання дозволяє їм оптимізувати маршрути, враховуючи динамічні зміни умов середовища.

Реакція на поведінку гравця. Супервізоване навчання сприяє створенню NPC, які адекватно реагують на дії гравця. Завдяки цьому ігрові персонажі демонструють осмислену поведінку, що підвищує рівень занурення [6].

Командна взаємодія. У багатокористувацьких іграх гібридні підходи дозволяють агентам координувати дії з іншими персонажами. Наприклад, розробка спільних стратегій у реальному часі сприяє досягненню цілей у командних іграх [7].

Машинне навчання відкриває нові горизонти в розробці ігрових агентів. Використання підкріплювального, супервізованого та гібридного навчання дозволяє створювати системи, які здатні до адаптації, навчання та високоточного прийняття рішень. У підсумку це збагачує ігровий процес, надаючи гравцям унікальний та інтерактивний досвід.

РОЗДІЛ 2
АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ІГОР ТА ВИБІР ОПТИМАЛЬНИХ

2.1. Аналіз алгоритмів пошуку та прийняття рішень у 2D стратегічних іграх

Алгоритм – це скінчена послідовність указівок на виконання дій, спрямованих на розв’язування задачі. Алгоритми пошуку та прийняття рішень є важливим компонентом штучного інтелекту в стратегічних іграх. Їх основне завдання - знаходити ефективні шляхи вирішення задач у динамічному середовищі, де необхідно швидко реагувати на зміни та передбачати дії суперника [8].

Алгоритм A*. A* є одним із найпоширеніших алгоритмів пошуку шляхів у відеоіграх завдяки його ефективності та універсальності. Він комбінує переваги жадібного пошуку та методу найкоротшого шляху, використовуючи евристичні оцінки для пріоритизації вузлів. A* дозволяє швидко знаходити оптимальні маршрути, навіть у складних картах із численними перешкодами. A* забезпечує ефективне переміщення юнітів через складні території, враховуючи особливості ландшафту, наявність ворогів і стратегічні цілі [9].

Алгоритм Мінімакс. Якщо гравець розуміє, що дії інших можуть негативно вплинути на його позицію, йому потрібна стратегія, що зменшує цей вплив та мінімізує втрати. Для цього використовується правило мінімакс. Мінімакс є базовим алгоритмом для ігор із двома сторонами, таких як шахи, шашки чи стратегічні бої. Його суть полягає у побудові дерева можливих ходів та виборі рішення, що максимізує виграш гравця і мінімізує виграш суперника. Мінімакс дозволяє аналізувати можливі сценарії та приймати оптимальні рішення, але його ефективність залежить від глибини аналізу дерева [10].

Альфа-бета. Альфа-бета відсікання (англ. Alpha-beta pruning) – це алгоритм пошуку, що прагне скоротити кількість вузлів, що оцінюються в дереві пошуку алгоритмом мінімакс. Основна ідея полягає в наступному: якщо на один із ваших ходів опонент має несприятливу для вас відповідь, то безглуздо аналізувати інші його можливі відповіді на цей ваш хід, тому що навіть якщо серед них і знайдуться сприятливіші для вас, противник їх не вибере. Альфа-бета відсікання є оптимізацією, так як результати роботи алгоритму, що оптимізується, не змінюються. Цей алгоритм є оптимізованою версією мінімаксу. Альфа-бета дозволяє відкидати ті гілки дерева рішень, які не впливають на кінцевий результат. Завдяки цьому значно знижуються обчислювальні витрати, що робить алгоритм придатним для ігор із високою складністю [11].

Гібридні підходи. У сучасних іграх дедалі частіше використовуються гібридні системи, які поєднують переваги кількох алгоритмів. Наприклад: A* із машинним навчанням для адаптивного пошуку шляхів; інімакс із нейронними мережами для прогнозування дій суперника; Альфа-бета відсікання з евристичними методами для зменшення обчислювальних витрат. Гібридні алгоритми дозволяють підвищити адаптивність системи та забезпечити більш реалістичну поведінку ігрових агентів.

Алгоритм Дейкстри. Алгоритм для знаходження найкоротших шляхів від початкової вершини до всіх інших вершин у графі з невід’ємними вагами ребер. Алгоритм використовує жадібний підхід, обираючи на кожному кроці вершину з найменшою відомою відстанню від початкової вершини і оновлюючи відстані до сусідніх вершин.

Алгоритми пошуку та прийняття рішень є фундаментальними для створення штучного інтелекту в 2D стратегічних іграх. Використання методів, таких як A*, мінімакс, альфа-бета відсікання та їх гібридів, дозволяє досягати балансу між ефективністю та адаптивністю. Правильний вибір алгоритму залежить від жанру гри, її механік і технічних вимог, що забезпечує якісний ігровий процес [12].

Таблиця 2.1 Порівняння алгоритмів

Алгоритм	Сфера застосування	Переваги	Недоліки
A*	Навігація юнітів у динамічному середовищі	Швидкий пошук, адаптивність	Залежність від евристики
Мінімакс	Прийняття рішень у двосторонніх іграх	Стратегічне планування, прогнозування	Висока обчислювальна складність
Альфа-бета відсікання	Оптимізація дерева рішень	Скорочення витрат, швидка робота	Залежність від якості евристики
Гібридні алгоритми	Інтегровані стратегії	Комбінування переваг кількох підходів	Складність реалізації

Джерело: складено автором

2.2. Порівняння класичних методів штучного інтелекту та підходів машинного навчання

Штучний інтелект (ШІ) є ключовою технологією для створення ігор, де поведінка агентів повинна бути гнучкою та адаптивною. Порівняння класичних методів ШІ та підходів машинного навчання дозволяє визначити їх переваги та недоліки в контексті стратегічних 2D ігор.

Логіка, заснована на правилах: класичні методи ШІ, такі як системи правил, базуються на чітких умовах та діях. Вони легко реалізуються, мають низькі обчислювальні витрати, але демонструють обмежену адаптивність, оскільки їх поведінка строго фіксована.

Алгоритми пошуку: такі алгоритми, як A*, забезпечують ефективне планування шляху, що робить їх ідеальними для навігації у великих просторах. Вони передбачувані та контрольовані, але можуть мати проблеми масштабованості на великих картах.

Дерева рішень: ці методи використовуються для прийняття простих рішень. Їх головними перевагами є прозорість та легкість налаштування, але вони мають недоліки у вигляді експоненційного зростання складності зі збільшенням кількості умов.

Моделі з підкріпленням: підходи машинного навчання, такі як Q-learning, дозволяють агентам навчатися через взаємодію із середовищем. Вони здатні адаптуватися та самонавчатися, але мають високі обчислювальні вимоги.

Нейронні мережі: ці моделі дозволяють моделювати складну поведінку завдяки роботі з великими обсягами даних. Їх головними перевагами є гнучкість і здатність до генералізації, але вони вимагають значної кількості даних для навчання.

Евристичні методи: комбінація класичних алгоритмів із методами оптимізації дозволяє досягти більшої ефективності. Їх часто застосовують для адаптивного налаштування параметрів.

Гібридні методи: поєднання класичних підходів із машинним навчанням дозволяє використовувати сильні сторони обох методів. Наприклад, алгоритм A* може забезпечувати первинне планування, а Q-learning або нейронна мережа - адаптацію до змін у середовищі. Це забезпечує ефективну поведінку агентів навіть у складних і динамічних умовах.

Таблиця 2.2 Порівняння класичних методів і машинного навчання

Характеристика	**Класичні методи**	**Машинне навчання**
Простота реалізації	Висока	Середня
Адаптивність	Обмежена	Висока
Масштабованість	Складна	Гнучка
Обчислювальні витрати	Низькі	Високі
Необхідність даних	Мінімальна	Значна
Сфери застосування	Чітко визначені задачі	Складні адаптивні задачі

Джерело: складено автором

Класичні методи штучного інтелекту залишаються ефективними для задач із чітко визначеними умовами та обмеженими сценаріями. Водночас, підходи машинного навчання дозволяють створювати більш адаптивні та інтелектуальні системи, які краще відповідають вимогам сучасних ігор. Використання гібридних методів дозволяє об’єднати сильні сторони обох підходів, забезпечуючи ефективність і гнучкість у вирішенні складних задач.

2.3. Вибір алгоритмів для реалізації в грі: переваги та недоліки

Розробка штучного інтелекту для стратегічних 2D-ігор потребує вибору оптимальних алгоритмів, які забезпечать ефективність, швидкодію та адаптивність. У цьому розділі аналізуються найбільш підходящі алгоритми для реалізації в грі, їх переваги та недоліки.

Вибір алгоритму залежить від особливостей ігрового середовища. У стратегічних 2D-іграх ключовими завданнями є навігація агентів, прийняття рішень та адаптація до змінних умов. Тому основними алгоритмами для реалізації є алгоритм A* для пошуку шляху, мінімакс з альфа-бета відсіканням для тактичного прийняття рішень, Q-learning для навчання агентів та гібридні підходи для складніших сценаріїв.

Алгоритм A* широко застосовується у 2D-іграх для навігації персонажів. Його головна перевага - знаходження оптимального шляху в умовах статичних ігрових карт. Цей алгоритм ефективно працює у визначених середовищах, однак має високу обчислювальну складність на великих картах із динамічними перешкодами.

Мінімакс із альфа-бета відсіканням використовується у сценаріях, де необхідно прогнозувати дії противника та приймати оптимальні рішення. Він ефективний у двохсторонніх конфліктах, де агенти мають протилежні цілі. Однак цей алгоритм вимагає значних обчислювальних ресурсів при збільшенні глибини пошуку [13].

Q-learning - метод навчання з підкріпленням, що дозволяє агентам самостійно приймати рішення на основі накопиченого досвіду. Він забезпечує адаптивність, дозволяючи персонажам змінювати свою стратегію у відповідь на дії гравця. Проте цей підхід потребує великої кількості навчальних епізодів та обчислювальних ресурсів для навчання ефективних стратегій.

У найпростішій формі Q-навчання Q-функція реалізована як таблиця станів і дій (там зберігаються Q-значення для кожного s, пара), використовується алгоритм ітерації значення оновлювати значення, оскільки агент безпосередньо накопичує знання [14].

Гібридні методи поєднують класичні алгоритми з навчанням, що дає змогу досягти балансу між точністю, швидкодією та адаптивністю. Наприклад, поєднання A* для навігації та Q-learning для стратегічного планування дозволяє створити ефективну модель поведінки агентів, здатних як до оптимального пошуку шляху, так і до адаптації під час гри [15]

Таблиця 2.3 Переваги та недоліки алгоритмів

Алгоритм	**Переваги**	**Недоліки**
A*	Знаходження найкоротшого шляху, ефективність на статичних картах	Високі обчислювальні витрати у великих динамічних середовищах
Мінімакс	Оптимальне рішення для стратегічних ігор, прогнозування дій противника	Складність реалізації, експоненційне зростання витрат
Q-learning	Адаптивність, можливість самонавчання агентів	Вимоги до ресурсів, потреба у великій кількості навчальних епізодів
Гібридні методи	Баланс між ефективністю і продуктивністю	Складність інтеграції та налаштування

Джерело: складено автором

Оптимальний вибір алгоритмів залежить від потреб гри. Алгоритм A* підходить для навігації, мінімакс корисний у стратегічних сценаріях, а Q-learning дозволяє створити адаптивних агентів. Використання гібридних методів забезпечує найкращі результати, оскільки дає можливість поєднати швидкодію класичних алгоритмів з адаптивністю методів навчання. Для ефективної реалізації штучного інтелекту в 2D-іграх необхідно обирати алгоритми залежно від ігрових умов та обчислювальних можливостей системи [16].

2.4. Постановка завдання для ігрового процесу та вибору стратегії поведінки NPC

Одним із ключових аспектів розробки стратегічних 2D-ігор є постановка завдань та визначення стратегій поведінки неігрових персонажів (NPC). Чітко сформульовані завдання забезпечують ефективність взаємодії гравця з ігровим середовищем та сприяють високому рівню реалізму ігрового досвіду. Від цього залежить не тільки задоволення користувача, але й успішність реалізації алгоритмів штучного інтелекту, що визначають поведінку NPC.

Основна мета розробки стратегій поведінки NPC полягає у створенні персонажів, які демонструють логічну та реалістичну реакцію на події, здатні адаптуватися до мінливих умов гри. Такими умовами можуть бути зміни у ресурсах, несподівана поява нових супротивників, зміни в поведінці гравця або інших NPC, модифікації ігрового ландшафту чи вплив погодних явищ. Саме завдяки гнучкості поведінки персонажів забезпечується постійна залученість гравця та цікавість ігрового процесу.

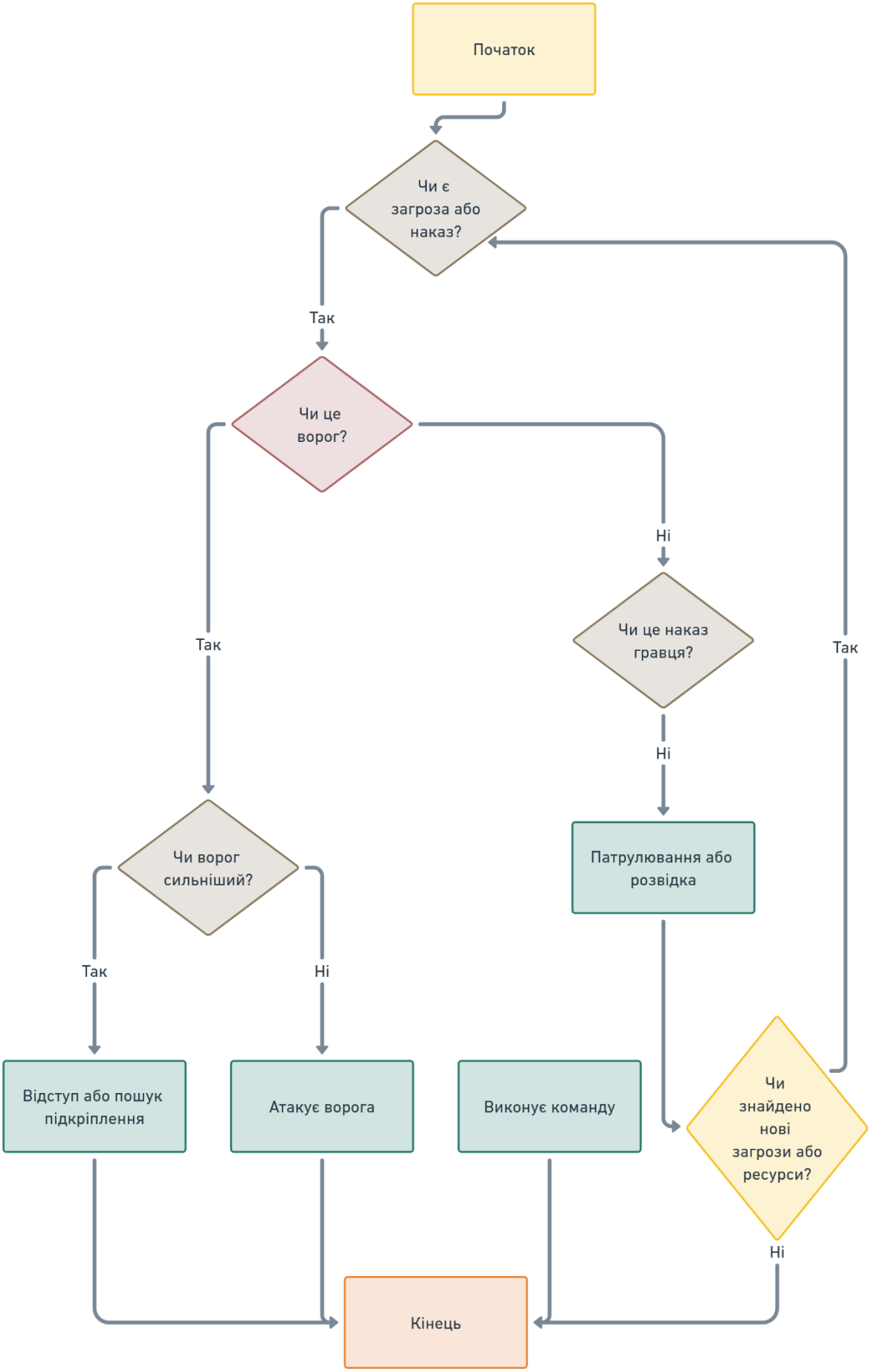


Рисунок 2.1. – Взаємодія ігрових агентів

Типовими завданнями NPC у стратегічних 2D-іграх виступають збір та розподіл ресурсів, побудова та захист стратегічних об’єктів, управління військовими силами, організація розвідки та взаємодія з іншими ігровими агентами, як у мирних, так і в конфліктних ситуаціях. Важливою є також роль NPC у економічному плануванні, визначенні оптимальних шляхів руху, швидкій реакції на тактичні можливості або загрози, що виникають у ігровому середовищі .

При виборі алгоритмів для реалізації поведінки NPC необхідно керуватись кількома основними критеріями. Перш за все, це гнучкість та ефективність алгоритму, що дає змогу персонажам діяти реалістично і непередбачувано, забезпечуючи природність їхньої поведінки. Другим важливим аспектом є складність впровадження обраних алгоритмів і необхідні для їхньої роботи обчислювальні ресурси. Нарешті, слід враховувати можливість масштабування алгоритмів, що забезпечує їхню ефективність у різних ігрових ситуаціях та умовах різної складності.

Для досягнення оптимальних результатів рекомендується використовувати комбіновані методи, що включають як класичні алгоритми штучного інтелекту (наприклад, пошукові алгоритми A* та алгоритми прийняття рішень мінімакс з альфа-бета відсіканням), так і сучасні методики машинного навчання (Q-learning, штучні нейронні мережі). Такий підхід дозволяє досягти стабільності у стандартних ігрових сценаріях і одночасно забезпечити високу адаптивність NPC до нових, нестандартних ситуацій, що виникають під час гри.

Для побудови адаптивної поведінки неігрових персонажів (NPC) у стратегічній 2D-грі задача ухвалення рішень моделюється як задача оптимізації в динамічному середовищі. Поведінку NPC можна описати формально за допомогою підходу, який використовується в навчанні з підкріпленням (reinforcement learning), де агент максимізує очікувану винагороду на основі взаємодії з середовищем.

Позначимо:

S – множина всіх можливих ігрових станів;

A – множина дій, доступних NPC (переміщення, атака, відступ, збір ресурсу);

$R(s,a)$ – функція винагороди, що оцінює доцільність дії a у стані s;

$\pi(s)$ – стратегія поведінки, яка відображає оптимальну дію у стані s;

$Q(s,a)$ – функція якості (цінності) дії a у стані s, що оновлюється за формулою:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right),$$

де α – коефіцієнт навчання, γ – коефіцієнт дисконту, s' – новий стан.

Метою NPC є побудова такої стратегії π^*, що максимізує сумарну винагороду:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$$

У випадку використання класичного підходу, як-от алгоритму мінімакс, поведінка NPC моделюється через дерево можливих ходів, де агент намагається мінімізувати виграш супротивника та максимізувати власний. Розмірність дерева залежить від кількості можливих дій у кожному стані та глибини прогнозу.

Також поведінка NPC може бути подана у вигляді скінченного автомата (FSM), де кожен стан відповідає певному режиму: «Патрулювання», «Атака», «Відступ», «Захист». Переходи між станами залежать від умов середовища, наприклад:

якщо ворог виявлений поблизу → перехід у стан «Атака»;

якщо рівень здоров’я низький → «Відступ»;

якщо ворожих юнітів немає – повернення до «Патрулювання».

Такий формалізований підхід дозволяє інтегрувати класичні алгоритми пошуку (A*), стратегічного аналізу (мінімакс) та машинного навчання (Q-

learning) у єдину систему ухвалення рішень, що забезпечує гнучку і адаптивну поведінку NPC у динамічному середовищі гри.

Отже, ретельний вибір та інтеграція алгоритмів штучного інтелекту забезпечують створення адаптивних, реалістичних та ефективних неігрових персонажів. Це, у свою чергу, суттєво підвищує якість ігрового процесу, забезпечує високий рівень залучення гравця та загалом сприяє успіху ігрового проекту.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ В 2D СТРАТЕГІЧНІЙ ГРІ

3.1. Проектування архітектури ігрового процесу та інтеграція AI-алгоритмів

Розробка покрокової 2D-стратегії потребує чіткої архітектури, що забезпечує коректну взаємодію між ігровими об’єктами, керування юнітами та впровадження штучного інтелекту для NPC. Основними складовими ігрового процесу є:

- ігрове поле – розділене на тайли, що визначають можливі шляхи переміщення юнітів;
- юніти гравця та NPC – персонажі, що виконують різні дії: переміщення, атака, взаємодія;
- перешкоди – об’єкти, що блокують рух (стіни, барикади).

Для розробки програмної частини було використано середовище Visual Studio Code, а всі скрипти написані мовою програмування C#. Таке поєднання забезпечило зручну інтеграцію з ігровим рушієм Unity та ефективне налагодження коду[18].

Нижче наведено інтерфейс середовища розробки зі скриптом керування підказками для AI (рис. 3.1).

В рамках розробки стратегічної 2D-гри на рушії Unity (рис. 3.2) було реалізовано алгоритми прийняття рішень для NPC із використанням підходів мінімакс та Q-learning [17]. Метою впровадження цих алгоритмів є забезпечення оптимальної поведінки неігрових персонажів у межах тактичної стратегії, що дозволяє їм аналізувати ігрову ситуацію та приймати рішення на основі розрахунку вигідних ходів.

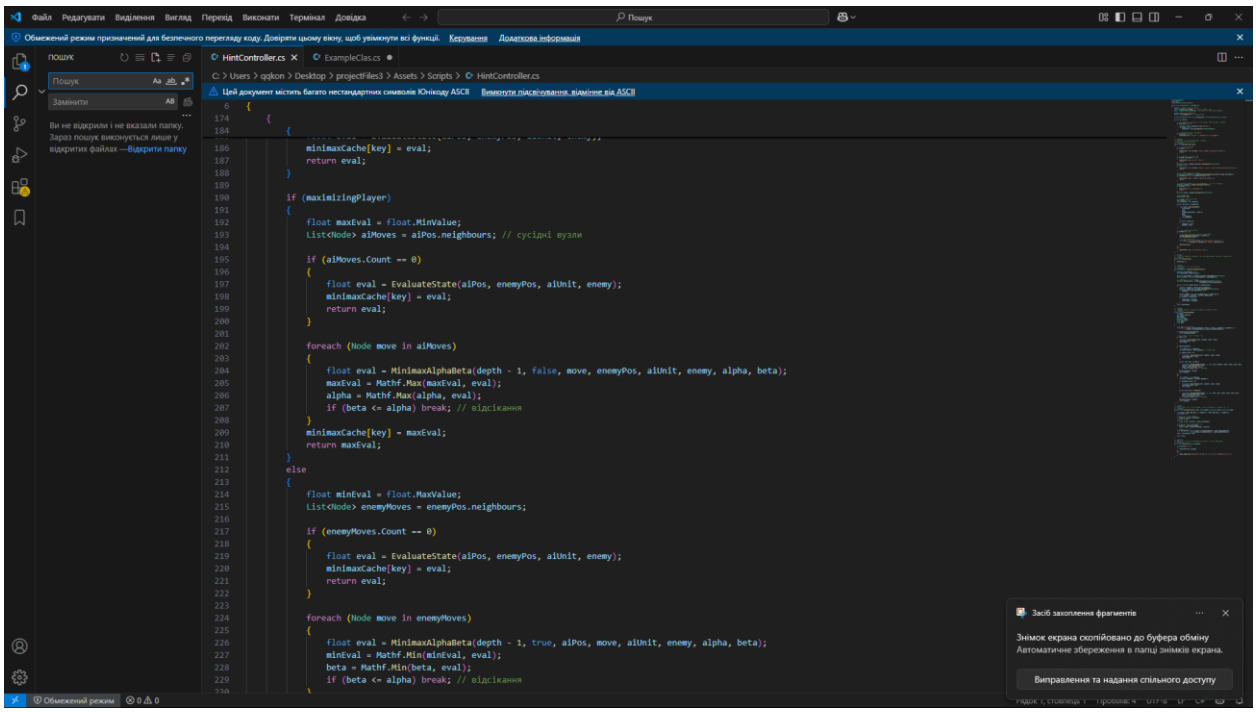


Рисунок 3.1 – Інтерфейс розробки у Visual Studio Code з відкритим скриптом HintController.cs

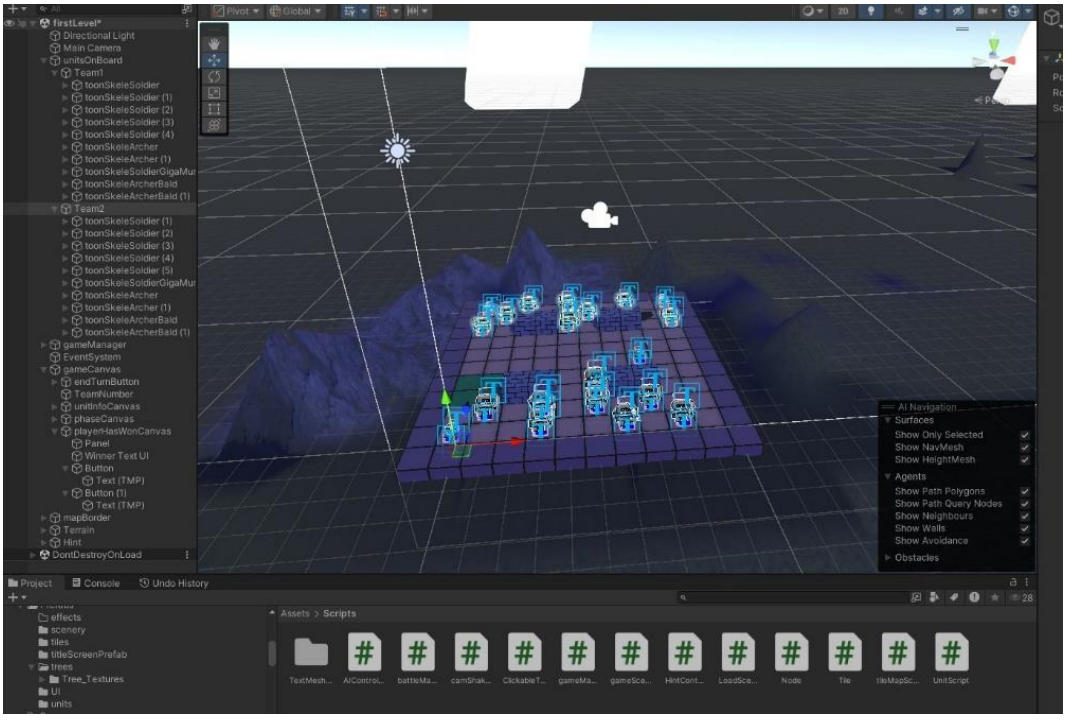


Рисунок 3.2 – Середовище розробки "Unity”

Створення власної системи стратегічної гри було виконано самостійно, логіка гри, алгоритми штучного інтелекту, система управління юнітами та

інші компоненти були розроблені заново або значно модифіковані відповідно до цілей дослідження.

У процесі реалізації ігрового прототипу особливу увагу було приділено трьом ключовим аспектам: впровадженню алгоритмів пошуку шляху, модулю ухвалення рішень неігрових персонажів (NPC), а також загальній оптимізації продуктивності гри. Таке технічне рішення дозволило досягти високого рівня адаптивності та реалістичності поведінки ігрових агентів у змінному середовищі.

Функціональну архітектуру проєкту побудовано за модульним принципом, що передбачає розподіл відповідальностей між окремими скриптами: головний керуючий модуль гри, модуль штучного інтелекту, модуль бойової взаємодії.

Головний керуючий модуль гри (gameManagerScript) – відповідає за управління основними фазами ігрового процесу, ініціалізацію рівня, активацію юнітів та обробку ігрових подій. Його функціональність охоплює логіку ходу гри, керування черговістю дій і контроль завершення рівнів.

Модуль штучного інтелекту (AIController) – реалізує систему ухвалення рішень для NPC, інтегруючи декілька алгоритмів. Зокрема, для навігації використовується пошуковий алгоритм A*, а для стратегічного вибору дій – алгоритм Minimax з можливістю подальшої модифікації через евристичні правила. Це дозволяє NPC не лише орієнтуватися на карті, а й ухвалювати тактично обґрунтовані рішення залежно від поточної ситуації.

Модуль бойової взаємодії (battleManagerScript) – забезпечує обробку бойових сценаріїв, зокрема механіку атак, зміну показників здоров’я та моделювання результатів зіткнень між юнітами. Реалізовані логічні умови дозволяють NPC оцінювати бойову доцільність своїх дій з урахуванням стану здоров’я, кількості ворогів та позиційних переваг.

Ігровий процес організовано на сітковому полі (tile-based map), де всі юніти розміщені у межах дискретних клітин (рис. 3.3). При виборі юніта відображаються допустимі шляхи для переміщення шляхом підсвічування

клітин, що забезпечує наочність для гравця. У верхній частині інтерфейсу додатково виводиться поточна інформація про обраного персонажа, включно з параметрами здоров’я, позиції та доступними діями.

Застосована модульна структура, поєднана з ефективними алгоритмами штучного інтелекту, дозволяє забезпечити динамічну, логічно обґрунтовану поведінку NPC, адаптацію до змін ігрового середовища, а також плавність і передбачуваність ігрового процесу.

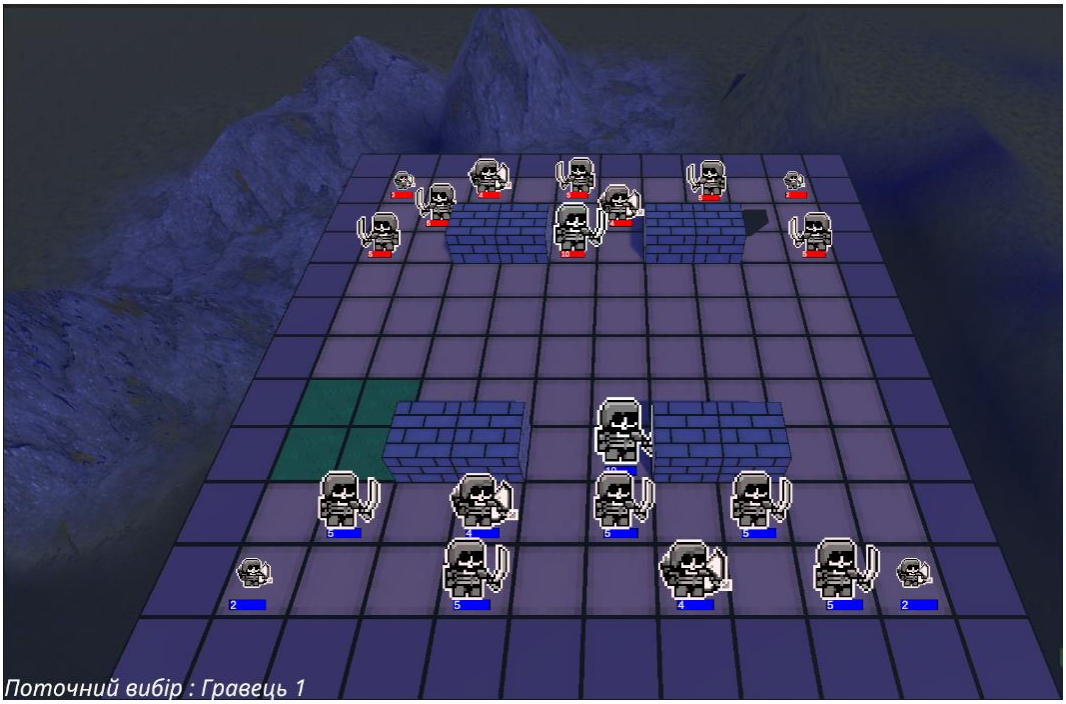


Рисунок 3.3 – Ігрове поле зі структурами та юнітами в покроковій стратегії

Ця структура дозволяє реалізувати динамічний та інтерактивний ігровий процес, де NPC можуть аналізувати стан карти, ухвалювати оптимальні рішення та адаптувати стратегію в залежності від дій гравця.

3.2. Реалізація алгоритму пошуку шляху (A*) для ігрових агентів

Загальний алгоритм переміщення юнітів у грі наведено на рис. 3.4. Якщо NPC отримує команду руху, спочатку перевіряється наявність цілі. Якщо вона існує, алгоритм аналізує можливі перешкоди та обирає оптимальний шлях. Якщо перешкод немає, персонаж рухається

безпосередньо до цілі. Якщо ж зустрічається перепона, система обчислює обхідний маршрут [19]. Коли NPC досягає цілі, він або завершує рух, або отримує нову команду.

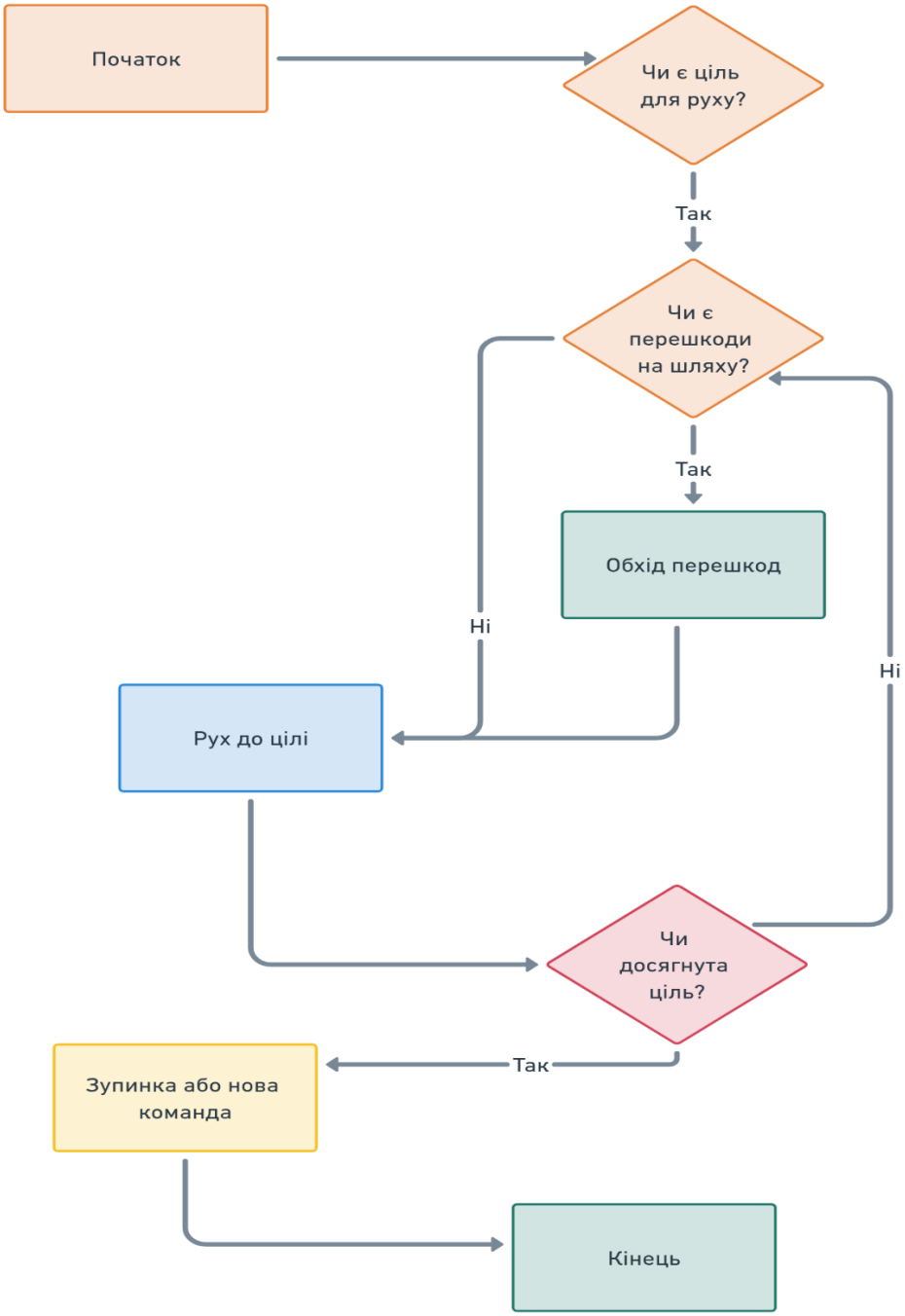


Рисунок 3.4 – Діаграма руху ігрового агента

У рамках розробки ігрового процесу було впроваджено один з найбільш ефективних та поширених алгоритмів пошуку шляху – A*, який

забезпечує обчислення оптимального маршруту переміщення ігрових агентів у дискретному середовищі (tile-based map).

Реалізація алгоритму здійснена у скрипті tileMapScript.cs і активується кожного разу, коли юніт (NPC або гравець) отримує команду на переміщення. Алгоритм функціонує на основі евристичної оцінки вартості маршруту між поточним і цільовим положенням та дозволяє мінімізувати кількість кроків з урахуванням перешкод, доступних напрямків і ландшафту.

Покроково алгоритм працює наступним чином: ініціалізація, оцінка вартості вузлів, вибір оптимального вузла, перевірка досягнення цілі, передача маршруту юніту.

Ініціалізація – задаються початкова та кінцева клітинки на ігровій сітці. Поточна позиція юніта додається до відкритого списку (open list), що містить вузли, які ще не опрацьовані.

Оцінка вартості вузлів – для кожного вузла обчислюється загальна вартість

$$f(n) = g(n) + h(n)$$

де:

$g(n)$ – вартість шляху від старту до вузла n;

$h(n)$ – евристична оцінка відстані до цілі (найчастіше евклідова або манхеттенська метрика).

Вибір оптимального вузла – з відкритого списку вибирається вузол із мінімальним значенням $f(n)$, який переноситься до закритого списку (closed list).

Перевірка досягнення цілі – якщо поточний вузол відповідає цільовому положенню, відбувається зворотне відтворення повного маршруту шляхом проходження через батьківські вузли.

Передача маршруту юніту – сформований шлях передається відповідному агенту, після чого він починає переміщення за заданою послідовністю клітинок.

Графічна візуалізація процесу представлена у вигляді підсвічених тайлів, які демонструють потенційні шляхи переміщення, а також маркера-індикатора (жовта стрілка або лінія), що ілюструє оптимальний обраний маршрут. Такий підхід значно підвищує інтуїтивність взаємодії гравця з інтерфейсом гри (рис. 3.5).

Використання алгоритму A* дозволяє забезпечити ефективну навігацію в обмежених умовах карти, уникати зіткнень із перешкодами та динамічно реагувати на зміни середовища. Крім того, його гнучкість дозволяє масштабування для використання в складніших сценаріях, зокрема з урахуванням ваг клітин, зони ризику або динамічних цілей.

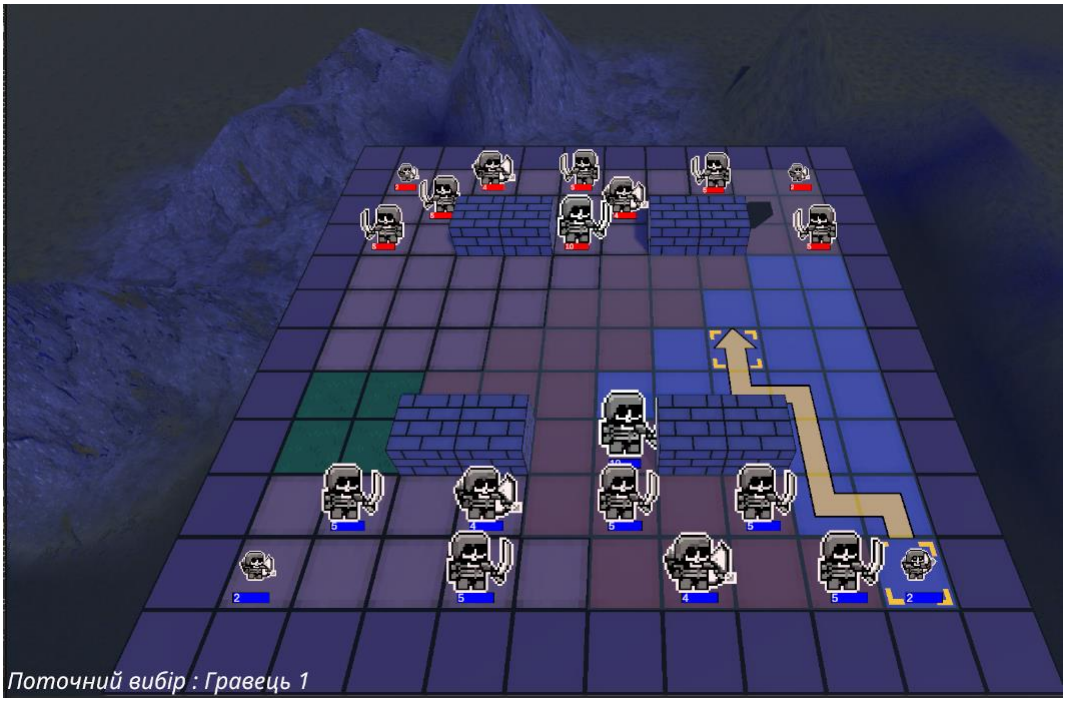


Рисунок 3.5 – Побудова оптимального маршруту переміщення юніта за алгоритмом A*

3.3. Імплементація алгоритмів прийняття рішень для NPC: мінімакс та Qlearning

Алгоритм мінімакс було інтегровано для визначення оптимальних ходів NPC у тактичних ситуаціях. Його основна концепція полягає в побудові дерева можливих ходів, у якому NPC оцінює свої варіанти дій та

передбачає можливі відповіді суперника. Цей підхід ефективний у покрокових стратегічних іграх, де NPC має обмежену кількість доступних дій на кожному кроці (рис. 3.6).

Рисунок 3.6 – Використання алгоритму мінімакс

Кнопка "Підказка" виконує функцію аналізу ігрової ситуації та надає гравцеві рекомендації щодо найкращого можливого ходу. Вона використовує алгоритм ухвалення рішень, що базується на оцінці доступних варіантів та виборі оптимального. При натисканні кнопки відбувається аналіз ігрового стану, визначення найбільш вигідного ходу і візуалізація рекомендації на екрані.

Оцінка стану включає перевірку всіх доступних одиниць, можливих дій, розташування ворогів, наявності бар’єрів та доступних клітинок для переміщення. Якщо використовується алгоритм мінімакс, система прораховує всі можливі ходи, оцінюючи їх вигідність з точки зору гравця та

потенційної реакції противника. Якщо ж застосовується Q-learning або інший метод машинного навчання, система звертається до попередньо зібраних даних, щоб знайти найбільш вигідний хід.

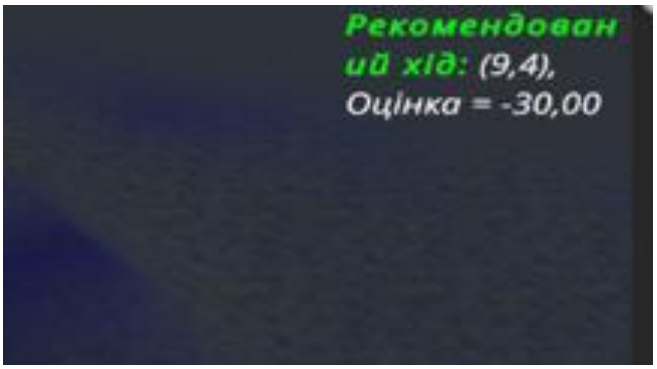


Рисунок 3.7 – оцінка запропонованого ходу

Результат аналізу виводиться на екран у вигляді рекомендованої позиції та підсвіченого шляху руху (рис. 3.7). Гравець бачить оцінку вибраного ходу, що може містити числовий показник, наприклад "-30.00", що вказує на можливі ризики чи вигоди. Для відображення підказки використовується підсвічування клітинок на полі та текстові підказки у верхньому правому куті екрана.

Реалізація функціоналу кнопки здійснюється у скриптах, серед яких AIController.cs або HintController.cs. Алгоритм проходить через усі доступні ходи, порівнює їх ефективність і вибирає найкращий варіант. Вигідність визначається на основі позиційної оцінки, стратегічних переваг і ризиків.

Інтеграція цієї функції робить ігровий процес динамічнішим і дозволяє гравцям ухвалювати більш зважені рішення, спираючись на рекомендації AI. Це підвищує рівень занурення у гру та робить її механіку глибшою та продуманішою.

Фрагмент коду, відповідальний за кнопку "Підказка":

```
public class HintController : MonoBehaviour
{
    public void ShowHint()
```

```
        {
            Move bestMove = GetBestMove();

            if (bestMove != null)
            {
                                Debug.Log($"Рекомендований   хід:
{bestMove.targetPosition}");
                ShowHintOnBoard(bestMove);
            }
        }
        private Move GetBestMove()
        {
            Move bestMove = null;
            float bestScore = float.MinValue;
                        foreach    (Move    possibleMove    in
GetAllPossibleMoves())
            {
                float score = EvaluateMove(possibleMove);
                if (score > bestScore)
                {
                    bestScore = score;
                    bestMove = possibleMove;
                }
            }
            return bestMove;
        }
        private float EvaluateMove(Move move)
        {
             // Оцінка вигідності ходу (на основі HP, позиції,
ризику атаки)
            return move.advantageScore - move.riskFactor;
        }
    }
```

3.4. Тестування поведінки ігрових агентів у реальному часі: симуляції та аналіз ефективності

Тестування алгоритмів штучного інтелекту є важливим етапом розробки стратегічної 2D-гри. Воно дозволяє оцінити, наскільки ефективно NPC приймають рішення, адаптуються до змін у грі та взаємодіють з іншими агентами. Основний підхід до тестування включав симуляцію різних сценаріїв та аналіз поведінки NPC у відповідь на події, що відбуваються в ігровому середовищі.

На основі розробленої механіки прийняття рішень NPC проводилися серії ігрових симуляцій, що дозволяли перевірити коректність роботи алгоритмів пошуку шляху, атаки, захисту та збору ресурсів. В ході тестування моделювалися ситуації з різними рівнями складності, де NPC змушені були діяти автономно, використовуючи мінімакс-стратегію та підхід Q-learning для прийняття оптимальних рішень.

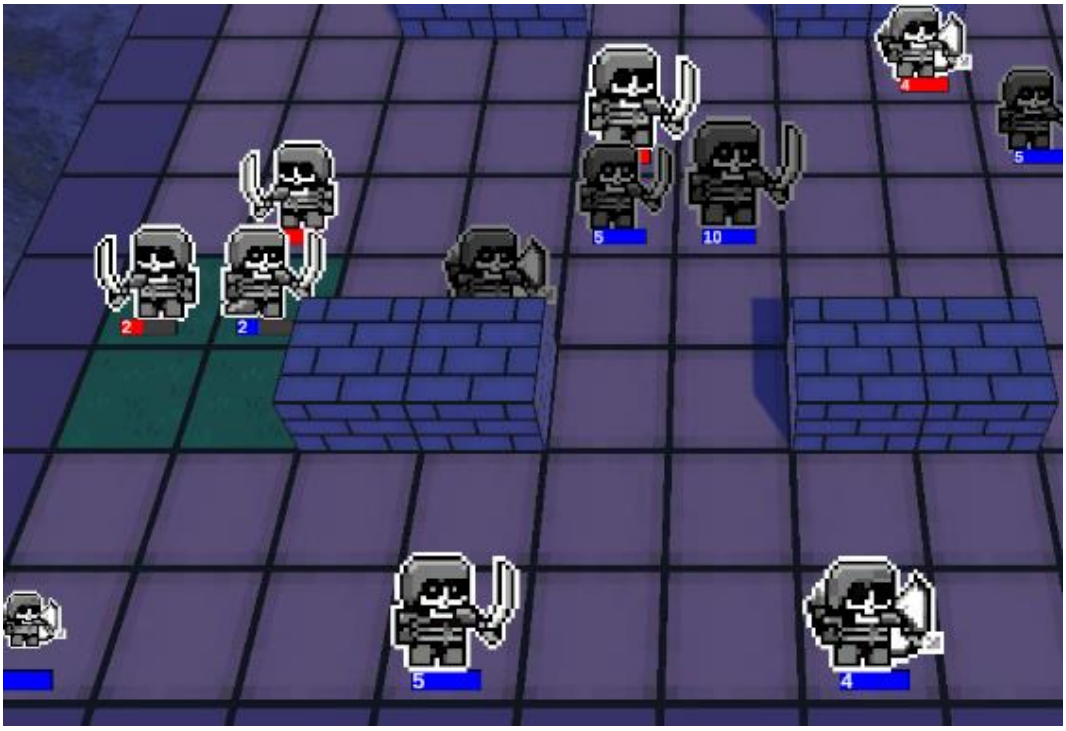


Рисунок 3.8 – Тестування поведінки NPC

На зображенні представлений один із сценаріїв тестування (рис. 3.8). Видно, як NPC з обох команд знаходяться у стратегічних точках, приймаючи рішення про атаки, переміщення та взаємодію між собою. Червоні індикатори здоров’я демонструють отриману шкоду під час бою, а сині показують стан союзних агентів. Завдяки алгоритму A* персонажі знаходять оптимальний шлях до цілі, уникаючи перешкод та ефективно використовуючи доступний простір.

Результати тестування показали, що використання комбінованого підходу, який включає класичні алгоритми та методи машинного навчання, дозволяє створювати NPC, які демонструють гнучку та адаптивну поведінку. Вони не лише виконують заздалегідь визначені дії, але й можуть змінювати свою стратегію в реальному часі відповідно до поточної ситуації на полі бою.

Загальний аналіз ефективності виявив, що впроваджені алгоритми дозволяють досягти збалансованої поведінки NPC, де вони не діють хаотично, а приймають зважені рішення. Це позитивно впливає на загальну динаміку гри та робить її цікавою як для новачків, так і для досвідчених гравців.

Рисунок 3.9. – Екран завершення гри після поразки гравця

Ще одним важливим аспектом тестування було перевірити, наскільки ефективно NPC можуть досягати перемоги у змагальному середовищі. На зображенні представлений екран завершення гри, де противник досяг перемоги над гравцем (рис. 3.9). Це демонструє, що алгоритми NPC можуть виконувати складні тактичні маневри, ухвалювати рішення щодо атаки, оборони та пересування відповідно до загальної ситуації на полі бою.

Впровадження адаптивних алгоритмів дало змогу досягти реалістичної поведінки супротивників, що підвищує рівень занурення у гру та робить її більш складною та цікавою для гравців.

3.5 Оптимізація алгоритмів для покращення продуктивності та ігрового досвіду

Процес оптимізації алгоритмів у грі був спрямований на покращення продуктивності, зменшення навантаження на систему та забезпечення плавності геймплею. Оптимізація проводилася у ключових аспектах: вибір рішень NPC, завантаження сцен, обчислення шляхів та рух юнітів.

Оптимізація вибору ходів NPC. Для зменшення затримок під час аналізу ігрових ходів у HintController.cs було застосовано кешування результатів Minimax. Це дозволило уникнути повторних розрахунків та зменшити час ухвалення рішень.

Фрагмент коду, що реалізує кешування:

```
    private Dictionary<GameState, float> cachedMoves = new Dictionary<GameState, float>();

    private float Minimax(GameState state, int depth, bool isMaximizing)
    {
        if (cachedMoves.ContainsKey(state))
            return cachedMoves[state];
        if (depth == 0 || state.IsGameOver())
            return EvaluateState(state);
```

```
            float bestValue = isMaximizing ? float.MinValue :
float.MaxValue;
            foreach (var move in state.GetPossibleMoves())
            {
                GameState newState = state.ApplyMove(move);
                  float moveValue = Minimax(newState, depth - 1,
!isMaximizing);

                  bestValue = isMaximizing ? Mathf.Max(bestValue,
moveValue) : Mathf.Min(bestValue, moveValue);
            }
            cachedMoves[state] = bestValue;
            return bestValue;
        }
```

Така оптимізація значно зменшила затримку під час аналізу стратегічних ходів у грі.

Оптимізація завантаження сцен. Замість стандартного завантаження сцен, яке могло спричиняти зависання гри, було застосовано асинхронне завантаження у LoadScene.cs. Це дозволило зменшити час очікування для гравця.

```
public void LoadNewScene(string sceneName)
{
   StartCoroutine(LoadSceneAsync(sceneName));
}
private IEnumerator LoadSceneAsync(string sceneName)
{
  AsyncOperation operation = SceneManager.LoadSceneAsync(sceneName);
   while (!operation.isDone)
   {
      yield return null;
   }
}
```

Цей метод дозволяє завантажувати нові рівні у фоновому режимі, не блокуючи основний потік гри.

Оптимізація обчислення шляхів (A)*. У tileMapScript.cs було оптимізовано алгоритм A*, який раніше обробляв занадто велику кількість вузлів. Було впроваджено обмеження області пошуку та ефективніше оновлення списку вузлів.

```
    private List<Node> FindPath(Vector2 start, Vector2 target)
    {
        List<Node> openList = new List<Node>();
        HashSet<Node> closedList = new HashSet<Node>();
        Node startNode = GetNodeFromPosition(start);
        Node targetNode = GetNodeFromPosition(target);
        openList.Add(startNode);
        while (openList.Count > 0)
        {
                Node currentNode = openList.OrderBy(n => 
n.F).First();
            if (currentNode == targetNode)
                return RetracePath(startNode, targetNode);
            openList.Remove(currentNode);
            closedList.Add(currentNode);
                        foreach (Node neighbor in 
GetNeighbors(currentNode))
            {
                            if (!neighbor.IsWalkable || 
closedList.Contains(neighbor))
                    continue;
                    float newMovementCost = currentNode.G + 
GetDistance(currentNode, neighbor);
                        if (newMovementCost < neighbor.G || 
!openList.Contains(neighbor))
                {
                    neighbor.G = newMovementCost;
                            neighbor.H = GetDistance(neighbor, 
targetNode);
                    neighbor.Parent = currentNode;
```

```
                    if (!openList.Contains(neighbor))
                        openList.Add(neighbor);
                }
            }
        }
        return new List<Node>();
    }
```

Ця зміна зменшила кількість непотрібних перевірок та значно прискорила розрахунок шляху.

Оптимізація руху юнітів. Щоб забезпечити більш реалістичний рух юнітів без різких змін напряму, було реалізовано метод MoveNextTile() у UnitScript.cs, що використовує Lerp для згладженого руху.

```
    private IEnumerator MoveNextTile(Vector3 targetPosition)
    {
            while (Vector3.Distance(transform.position, targetPosition) > 0.1f)
        {
                                    transform.position = Vector3.Lerp(transform.position, targetPosition, moveSpeed * Time.deltaTime);
            yield return null;
        }
        transform.position = targetPosition;
    }
```

Така реалізація забезпечила плавний рух юнітів та позбулася "ривків" під час зміни позицій.

Результати оптимізації. Після впровадження зазначених змін тестування показало значне покращення продуктивності: Час обчислення оптимального ходу за алгоритмом Minimax скоротився у 2-3 рази завдяки кешуванню. Завантаження рівнів відбувається без зависань, що покращило плавність гри. Алгоритм A* обробляє на 30% менше вузлів, що

пришвидшило генерацію шляху. Рух юнітів став більш реалістичним, без різких змін напряму та зависань у перешкодах.

Таким чином, оптимізація AI-алгоритмів дозволила значно покращити якість гри, забезпечивши плавний і продуктивний геймплей.

ВИСНОВКИ

У ході виконання дипломної роботи було здійснено комплексний аналіз і реалізацію алгоритмів штучного інтелекту для покращення ігрового процесу в 2D-стратегічних іграх. Дослідження охопило як класичні алгоритми пошуку та прийняття рішень, так і сучасні підходи машинного навчання, що дозволило оцінити їхню ефективність у реальних ігрових сценаріях.

Результати дослідження підтвердили ефективність використання алгоритму A* для швидкого та оптимального пошуку шляху NPC у динамічному середовищі. Він дозволяє персонажам оперативно адаптувати свій рух до змін у ландшафті, уникаючи перешкод та знаходячи найоптимальніші маршрути. Алгоритми прийняття рішень, такі як мінімакс із альфа-бета відсіканням, забезпечують NPC можливість оцінювати наслідки своїх дій, що покращує їхню стратегічну поведінку. Це особливо важливо у сценаріях, які вимагають довгострокового планування для ефективного управління ресурсами чи військовими силами. Водночас методи підкріплювального навчання, зокрема Q-learning, продемонстрували високий потенціал у створенні адаптивних агентів, здатних навчатися на основі досвіду, що значно покращує їхню реакцію на динамічні зміни в грі.

Реалізація обраних алгоритмів у розробленому ігровому проєкті дозволила досягти високого рівня автономності NPC та їхньої здатності до адаптації у мінливих умовах гри. Впровадження оптимізаційних підходів, таких як покращення евристичних функцій у A*, зменшило обчислювальні витрати без втрати якості результатів. Це особливо важливо для ігор із великими картами та численними NPC, де значне навантаження на систему може впливати на загальну продуктивність гри. Крім того, оптимізація мінімакс-алгоритму шляхом використання альфа-бета відсікання дозволила значно скоротити кількість перевірюваних варіантів розвитку подій, що суттєво прискорило ухвалення рішень NPC у стратегічних сценаріях.

Значущість отриманих результатів полягає у можливості їхнього застосування для покращення ігрового процесу в стратегічних 2D-іграх, а також у розширенні методів реалізації штучного інтелекту у відеоіграх загалом. Запропоновані рішення дозволяють створювати більш динамічних NPC, які не лише реагують на зміни в ігровому середовищі, але й прогнозують розвиток подій та ухвалюють рішення на основі накопиченого досвіду. Отримані висновки та розроблені алгоритмічні підходи можуть бути використані для подальших досліджень у сфері адаптивного геймплейного AI, а також інтеграції більш складних моделей машинного навчання, зокрема глибоких нейронних мереж для прогнозування поведінки NPC.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Голубенко В. А. Аналіз використання алгоритмів штучного інтелекту в іграх. *Технічна інженерія*. 2023. № 2(92). С. 109-112. URL: https://doi.org/10.26642/ten-2023-2(92)-109-112
2. Кривенко О. В., Загірний М. М. Дослідження та моделювання ігрового процесу на базі методів машинного навчання. *Наука та виробництво*. 2020. № 22. URL: https://doi.org/10.31498/2522-9990222020197581.
3. Мокін В.Б., Дратований М.В. Наука про дані: машинне навчання та інтелектуальний аналіз дани. *Наука про дані: машинне навчання та інтелектуальний аналіз даних - Електронний навчальний посібник*. 2024. С. 263.
   URL: https://pdf.lib.vntu.edu.ua/books/2024/Mokin_2024_263.pdf.
4. Штучний інтелект та машинне навчання: що спільного, у чому різниця між ними і чому це важливо?. *blog.colobridge.net*. URL: https://blog.colobridge.net/uk/2024/05/artificial-intelligence-and-machine-learning-ua/.
5. *Apache HTTP Server Test Page powered by Linux*. URL: https://personales.upv.es/thinkmind/dl/conferences/simul/simul_2022/simul_2022_1_30_50017.pdf.
6. Apperson L. Beginning Game Development: NPC Pathfinding. *Medium*. URL: https://medium.com/@lemapp09/beginning-game-development-npc-pathfinding-5cbf721c0039.
7. Artificial intelligence for video game visualization, advancements, benefits and challenges / Y. Wu et al. *Mathematical Biosciences and Engineering*. 2023. Vol. 20, no. 8. P. 15345-15373. URL: https://doi.org/10.3934/mbe.2023686.
8. Basics of Artificial Intelligence in Game Development | Melior Games. *Melior Games*. URL: https://meliorgames.com/best-practices/basics-of-artificial-intelligence-in-game-development/.

9. Brennan A. Minimax algorithm and alpha-beta pruning. *Medium*. URL: https://medium.com/@aaronbrennan.brennan/minimax-algorithm-and-alpha-beta-pruning-646beb01566c.
10. GeeksforGeeks. Explain the role of minimax algorithm in adversarial search for optimal decision-making? - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/minimax-algorithm-in-adversarial-search-for-optimal-decision-making/.
11. GeeksforGeeks. Minimax Algorithm in Game Theory | Set 1 (Introduction) - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/.
12. Karaca Y., Derias D., Sarsar G. AI-Powered Procedural Content Generation: Enhancing NPC Behaviour for an Immersive Gaming Experience. *SSRN Electronic Journal*. 2024. URL: https://doi.org/10.2139/ssrn.4663382.
13. Pal S. Optimizing Decision-making with the Minimax AI algorithm. *Medium*. URL: https://medium.com/swlh/optimizing-decision-making-with-the-minimax-ai-algorithm-69cce500c6d6.
14. Simplilearn. Q-Learning Explained: Learn Reinforcement Learning Basics. *Simplilearn.com*. URL: https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning.
15. Team E. I. S. What is Hybrid AI? An Approach for Data Discovery. *Information Architecture Consulting | Earley Information Science*. URL: https://www.earley.com/insights/what-is-hybrid-ai-approach-to-data-discovery#:~:text=Hybrid%20AI%20is%20a%20method,achievable%20by%20either%20one%20alone.
16. Uludağlı M. Ç., Oğuz K. Non-player character decision-making in computer games. *Artificial Intelligence Review*. 2023. URL: https://doi.org/10.1007/s10462-023-10491-7

17. Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. *Unity*. URL: https://unity.com.
18. Visual Studio: IDE and Code Editor for Software Developers and Teams. *Visual Studio*. URL: https://visualstudio.microsoft.com/.
19. Xue L. Application of Artificial Intelligence in Digital Games Based on Mathematical Statistics. *Mobile Information Systems*. 2022. Vol. 2022. P. 1-11. URL: https://doi.org/10.1155/2022/7145588.
20. K. Karpouzis and G. Tsatiris. AI in (and for) Games. Springer International Publishing, 01 2022. https://doi.org/10.1007/978-3-030-76794-5_3.
21. R. Santamaria and Contributors. Raylib, 2023. https://www.raylib.com/. Last accessed on: 30/06/2024.
22. X. Cui and H. Shi. A*-based Pathfnding in Modern Computer Games. International Journal of Computer Science and Network Security (IJCSNS), 11(1):125–130, 2011. https://api.semanticscholar.org/CorpusID:6458879. A. Primanita, R. Efendi, and W. Hidayat. Comparison of A* and Iterative Deepening A* algorithms for non-player character in Role Playing Game. In 2017 International Conference on Electrical Engineering and Computer Science (ICECOS), pages 202–205, 2017. https://doi.org/10.1109/ICECOS.2017.8167134.
23. Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. https: //unity.com/.
24.] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling, volume 5, pages 262–271, 2005. https://dl.acm.org/doi/10.5555/3037062.303 7096.
25. N.A. Mas'udi, E.M.A Jonemaro, M.A. Akbar, and T. Afrianto. Development of Non-Player Character for 3D Kart Racing Game Using Decision Tree. Fountain of Informatics Journal, 6(2):51–60, 2021. https://doi.org/10.21111/fij.v6i2.46 78.

26. R. Dworzanski and H. Hlavacs. Shaping AI Behavior: A Q-Learning Driven Approach to Automatic Behavior Tree Creation. In 2024 IEEE International Conference on Artifcial Intelligence and eXtended and Virtual Reality (AIxVR), pages 241–245. IEEE Computer Society, 01 2024. https://doi.ieeecomputersociety.org/10.1 109/AIxVR59861.2024.00039.
27. R. Rosalina, A. Sengkey, G. Sahuri, and R. Mandala. Generating intelligent agent behaviors in multi-agent game AI using deep reinforcement learning algorithm. International Journal of Advances in Applied Sciences, 12:396, 12 2023. http: //dx.doi.org/10.11591/ijaas.v12.i4.pp396-404.
28. Godot Engine - Free and open source 2D and 3D game engine. https://godoteng ine.org/.
29. R. Dworzanski and H. Hlavacs. Shaping AI Behavior: A Q-Learning Driven Approach to Automatic Behavior Tree Creation. In 2024 IEEE International Conference on Artifcial Intelligence and eXtended and Virtual Reality (AIxVR), pages 241–245. IEEE Computer Society, 01 2024. https://doi.ieeecomputersociety.org/10.1 109/AIxVR59861.2024.00039.
30. R. Rosalina, A. Sengkey, G. Sahuri, and R. Mandala. Generating intelligent agent behaviors in multi-agent game AI using deep reinforcement learning algorithm. International Journal of Advances in Applied Sciences, 12:396, 12 2023. http: //dx.doi.org/10.11591/ijaas.v12.i4.pp396-404.

ДОДАТКИ

ДОДАТОК А

Спрайти юнітів

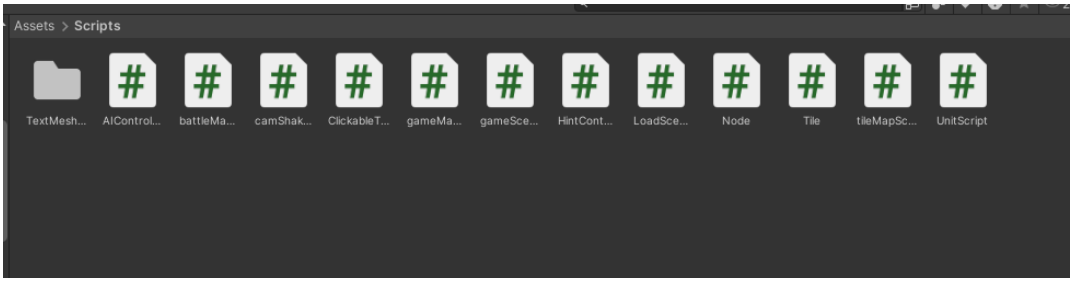


Рисунок 1 - Всі скрипти які присутні у проекті

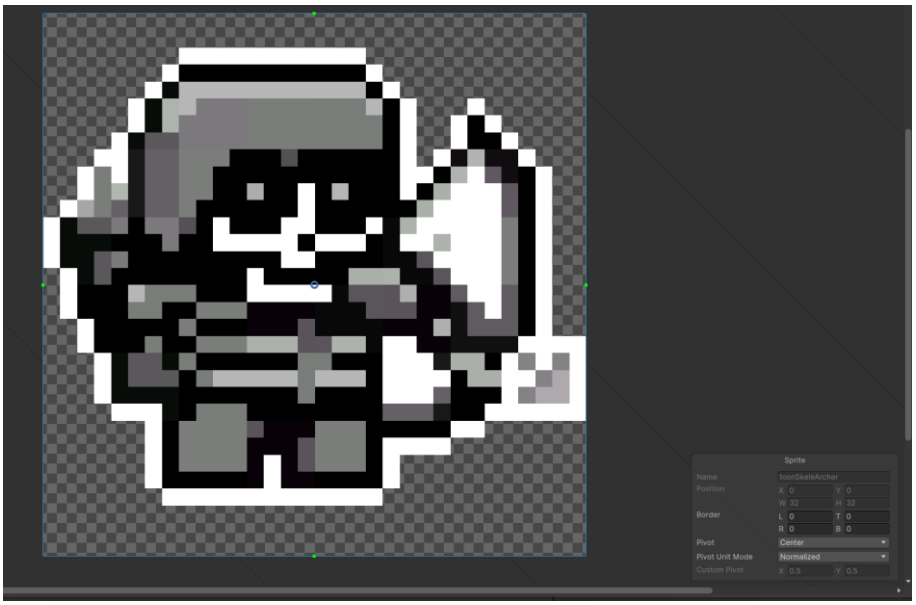


Рисунок 2 - Спрайт солдата лучника

Рисунок 3 - Спрайт солдата мечника

ДОДАТОК Б

Налаштування юніта

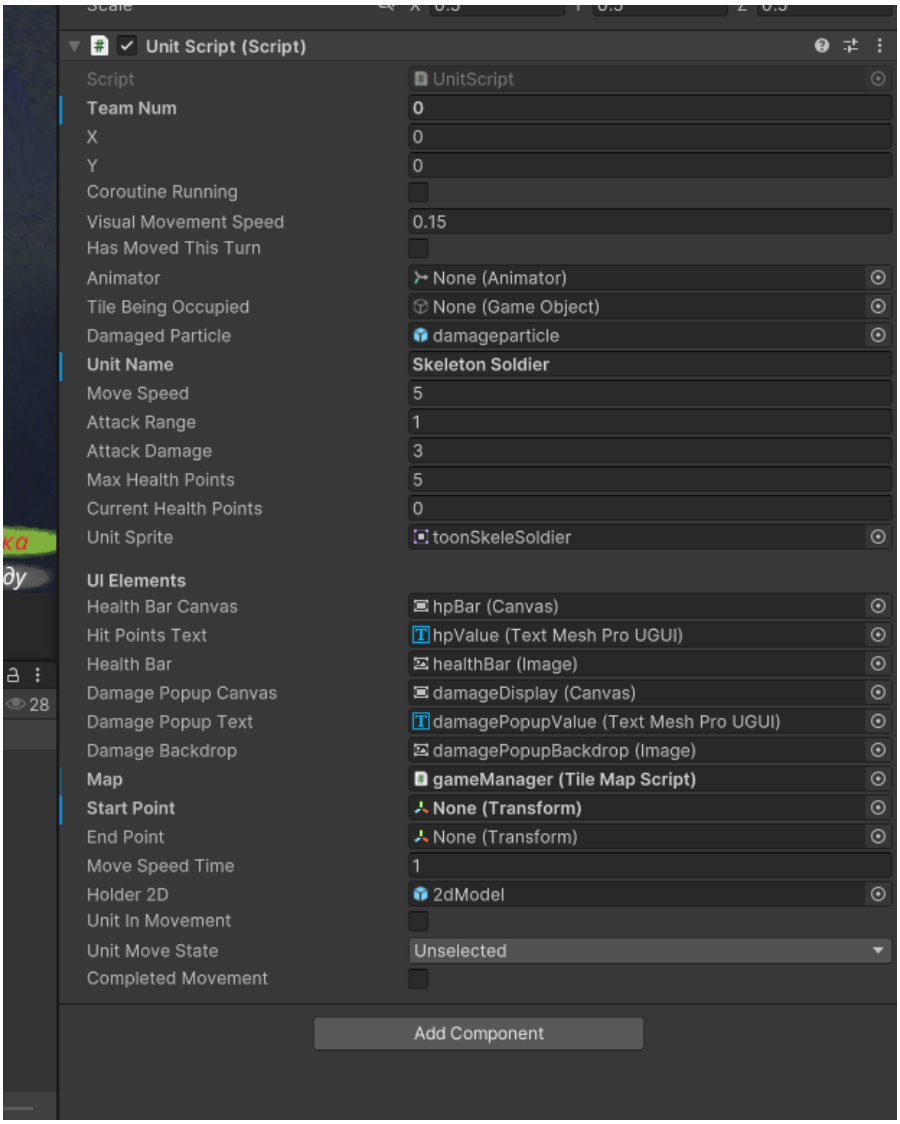


Рисунок 4. Налаштування юніта