НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.89

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету	Завідувач кафедри
інформаційних технологій	комп'ютерних наук
Глазунова О.Г., д.п.н., професор	Голуб Б.Л., к.т.н., доцент
_____	_____
____ _____ 2023 р.	**____ _____ 2023р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему «Програмна реалізація генетичного алгоритму у розробці штучного інтелекту для ігор»

Спеціальність 121 Інженерія програмного забезпечення

(код і назва)

Освітня програма «Програмне забезпечення інформаційних систем»

(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

доцент, кандидат фізико-математичних наук _____ Кириченко В.В
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

к.т.н, доцент _____ Лендєл Тарас Іванович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Довгорукий Андрій Андрійович
(підпис) (ПІБ студента)

КИЇВ-2023

РЕФЕРАТ

Магістерська кваліфікаційна робота на тему "Програмна реалізація генетичного алгоритму у розробці штучного інтелекту для ігор" розглядає застосування генетичних алгоритмів для оптимізації ігрового процесу в середовищі Unity. Робота містить 50 сторінок, включаючи 16 ілюстрацій, 0 таблиць, 0 додатків та 11 використаних літературних джерел.

Об’єктом дослідження є процес автоматичної генерації ігрових карт та механізм балансування сили ворожих NPC (неігрових персонажів), які атакують гравця. Дослідження фокусується на покращенні ігрового досвіду через адаптацію ігрового середовища та поведінки ворогів до стилю гри користувача.

Використані методи дослідження охоплюють програмування в Unity, тестування генетичних алгоритмів, а також аналітичне оцінювання отриманих результатів. При виконанні роботи було застосовано ітеративний підхід з регулярною верифікацією результатів генерації та балансування.

Метою роботи є розробка ефективного методу створення ігрових карт та балансування ворогів в реальному часі, що дозволяє створювати унікальний ігровий досвід для кожного користувача.

Отримані результати показали, що використання генетичних алгоритмів може значно покращити процес створення ігрових карт та балансування ворогів, забезпечуючи високий рівень адаптивності до дій гравця. Наукова новизна полягає у розробці генетичного алгоритму, адаптованого для динамічного ігрового середовища Unity, який може бути використаний для інших типів ігор.

Рекомендації щодо впровадження результатів роботи включають інтеграцію розробленого генетичного алгоритму у комерційні ігрові проекти, а також його подальше використання для навчання і поліпшення штучного інтелекту в іграх.

Прикладна значимість роботи полягає в здатності алгоритму до швидкої адаптації, що робить його ідеальним для ігрової індустрії, де потрібна висока реактивність ігрових елементів та персоналізація ігрового процесу.

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) **Інформаційних технологій**

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Голуб Б. Л
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"____" _____2023 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Довгорукий Андрій Андрійович
(прізвище, ім'я, по батькові)

Спеціальність 121 Інженерія Програмного Забезпечення
(код і назва)

Освітня програма «Програмне забезпечення інформаційних систем»
(назва)

Орієнтація освітньої програми освітньо-профейсійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи «Програмна реалізація генетичного алгоритму » у розробці штучного інтелекту для ігор
затверджена наказом ректора НУБіП України від "30" грудня 2022р. №1939 – «С»

Термін подання завершеної роботи на кафедру «03» листопада 2023 р.
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи _____

Перелік питань, що підлягають дослідженню:
1. _____
2. _____
3. _____

Перелік графічного матеріалу (за потреби) _____

Дата видачі завдання "_____"_____20___ р.

Керівник магістерської кваліфікаційної роботи _____ _____
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Довгорукий А. А
(підпис) (прізвище та ініціали студента)

ЗМІСТ

/C:\\Users\\SOLITARIA\\Downloads\\Telegram%20Desktop\\Хміль_Кім21%20(1

E

в

HYPERLINK

file:///C:\\Users\\SOLITARIA\\Downloads\\Telegram%20Desktop\\Хміль_Кім21

HYPERLINK

"file:///C:\\Users\\SOLITARIA\\Downloads\\Telegram%20Desktop\\Хміль_Кім21

C:\\Users\\SOLITARIA\\Downloads\\Telegram%20Desktop\\Хміль_Кім21%20(1

HYPERLINK

"file:///C:\\Users\\SOLITARIA\\Downloads\\Telegram%20Desktop\\Хміль_Кім21

C:\\Users\\SOLITARIA\\Downloads\\Telegram%20Desktop\\Хміль_Кім21%20(1

///C:\\Users\\SOLITARIA\\Downloads\\Telegram%20Desktop\\Хміль_Кім21%

C:\\Users\\SOLITARIA\\Downloads\\Telegram%20Desktop\\Хміль_Кім21%20(1

ВСТУП

У світі, де технологічний прогрес відбувається з неймовірною швидкістю, індустрія комп'ютерних ігор виступає не просто як сегмент розваг, а як платформа для інновацій та випробування новітніх технологій. Штучний інтелект (ШІ) відіграє ключову роль у формуванні нових трендів в розробці ігор, виходячи за рамки традиційних алгоритмів до створення комплексних імітативних систем, здатних до адаптації та навчання. Зокрема, генетичні алгоритми як одна з областей ШІ відкривають нові можливості для оптимізації та автоматизації ігрових процесів. Такий підхід актуальний у контексті підвищення динамічності та реалістичності ігрового середовища, що є ключовою вимогою сучасної аудиторії геймерів.

Актуальність цієї магістерської роботи обумовлена необхідністю подальшого розвитку методів ШІ у геймдеві. Застосування генетичних алгоритмів для генерації ігрових карт і балансування ворогів в Unity спрямоване на створення більш залучаючих і різноманітних ігрових досвідів, що адаптуються до поведінки і переваг користувачів.

Предмет дослідження – є процес використання генетичного алгоритму як форми штучного інтелекту для балансування гри, а об’єкт - програмна реалізація цих алгоритмів у середовищі Unity для генерації динамічних ігрових світів та балансування поведінки NPC (Non-Player Characters).

Мета дослідження полягає у розробці та імплементації алгоритмів, заснованих на принципах генетичного алгоритму для створення адаптивних ігрових середовищ, які здатні відтворювати багатий та змінюваний контент залежно від дій гравців.

У рамках досягнення визначеної мети перед дослідженням стоять такі завдання:

1) Проаналізувати існуючі підходи до генетичних алгоритмів в ігровій індустрії.

2) Розробити власний метод генерації ігрових карт, оптимізований під потреби реального ігрового проекту.
3) Створити систему балансування NPC, яка адаптується під рівень та стиль гри користувача.

Методологічну базу дослідження складають системний підхід та методи теоретичного аналізу, моделювання, програмування та експерименту.

Наукова новизна роботи виявлена у розробці і впровадженні оригінальних алгоритмів генерації карт та адаптивного балансування NPC.

Апробація результатів дослідження відбувалася на етапі розробки ігрових проектів із застосуванням Unity, де було досягнуто позитивного відгуку від користувачів, що свідчить про практичну цінність розроблених методик.

Структурно магістерська робота поділена на вступ, 4 розділи, що містять теоретичні обґрунтування, опис методик розробки та результати експериментальних досліджень, загальні висновки та рекомендації, список використаної літератури та додатки, що містять допоміжний матеріал та програмний код розроблених алгоритмів.

РОЗДІЛ 1 ВВЕДЕННЯ В ГЕНЕТИЧНІ АЛГОРИТМИ

1.1 Поняття генетичного алгоритму

Генетичний алгоритм (ГА) являє собою пошуковий евристичний інструмент, який імітує процеси природної еволюції, які відбуваються в біології. Основним принципом генетичного алгоритму є використання генетичних принципів для структурування можливих рішень задачі та їх подальшого удосконалення через механізми, подібні до природного добору, мутацій та схрещування, з метою віднайдення найбільш відповідного рішення. Ці алгоритми виявляються особливо корисними при розв'язуванні задач, які представляють велику обчислювальну складність і не піддаються легкому аналітичному або евристичному розв'язку. Такі проблеми часто характеризуються великою кількістю змінних та взаємозв'язків, які ускладнюють пошук оптимального рішення.

В роботі генетичного алгоритму можна виділити шість основних етапів: ініціалізація популяції, оцінка пристосованості особин, вибір батьків (селекція), рекомбінація (схрещування), мутація та відбір для виживання. Ініціація популяції полягає у створенні вихідного набору потенційних рішень, а оцінка пристосованості — у визначенні, наскільки добре кожне рішення вирішує поставлену задачу. Потім, в рамках циклічного процесу, що продовжується до виконання умов зупинки (наприклад, знаходження достатньо хорошого рішення, вичерпання ліміту поколінь чи відсутності підвищення пристосованості протягом заданої кількості поколінь), відбувається вибір батьків, їх рекомбінація для створення потомства, потенційна мутація потомства, оцінка нового покоління та визначення, які особини залишаться у популяції, а які будуть відсіяні. Рис. 1.1 ілюструє робочий процес генетичного алгоритму.

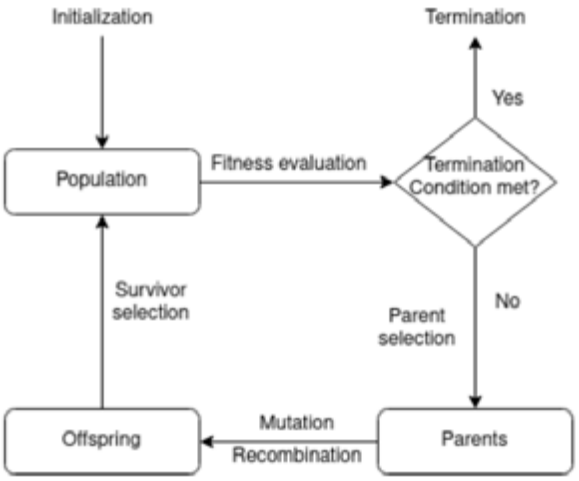

Рис. 1.1

Термінологія, яка використовується в генетичних алгоритмах, інтенсивно запозичує поняття з біології. Так, кандидат на рішення задачі називається фенотипом, його репрезентація – індивідом або хромосомою, що складається з набору генів. Ген може представляти собою число, біт або будь-яку іншу форму представлення окремої частини особини. Кожен індивід має бути перетворений з своєї генетичної форми в конкретне рішення задачі – фенотип. Варто зазначити, що різні індивіди можуть відповідати одному фенотипу, а деякі фенотипи можуть представляти недійсні рішення, особливо якщо порушуються обмеження задачі.

1.2 Ініціалізація популяції у генетичних алгоритмах

На початку використання генетичного алгоритму (ГА) необхідно сформувати початкову популяцію. Популяція представляє собою сукупність всіх особин, які оцінюються в рамках ГА. Розмір популяції є критичним параметром: велика популяція вимагає більш тривалого часу для оцінки кожної особини, а маленька популяція може страждати від недостатньої різноманітності. Під час ініціалізації популяції слід враховувати два важливих аспекти: різноманітність популяції та спрямування на потенційно ефективні рішення.

Різноманітність популяції означає ступінь варіативності між окремими особинами. Чим більш різноманітною є популяція, тим більша ймовірність того, що алгоритм не збігатиметься до локального максимуму на ранніх етапах і не застрягне у ньому. Водночас, існує практика формування популяції з певним ухилом на користь якісних рішень. Такий підхід зменшує різноманітність популяції, і не завжди він є придатним для всіх видів задач. Проте, якщо це можливо, то створення початкової популяції, яка вже "цілить" у перспективні області простору пошуку, може значно зменшити обсяг роботи, необхідний для виявлення хороших рішень.

Так, у контексті генетичного алгоритму, ініціалізація популяції стає процесом досить нюансованим. З одного боку, бажано забезпечити максимальну різноманітність, щоб алгоритм міг досліджувати широкий простір рішень. З іншого — цінним є напрямок пошуку до областей, де вже імовірно існують оптимальні рішення. Це можна досягти через застосування стратегій, як-от контрольоване введення до популяції високо адаптованих особин чи використання попередньої інформації про проблему для формування первинної популяції. Також ефективним є балансування між експлуатацією і дослідженням, тобто між використанням уже знайдених хороших рішень для формування нових особин і генерацією варіантів, що суттєво відрізняються від існуючих, що забезпечує розширення простору пошуку.

1.3 Оцінка придатності особин у генетичних алгоритмах

Поняття "придатність" (fitness) є ключовим у генетичних алгоритмах і вказує на якість рішення проблеми, яке забезпечує дана особина. Для його визначення застосовують функцію придатності, яка, як правило, враховує не тільки вартість рішення, але й ступінь його недоцільності чи неможливості.

Оцінка придатності визначається відповідно до об'єктивної функції задачі, і рішення, які краще справляються з поставленим завданням, зазвичай отримують вищу оцінку придатності. Рішення, що є допустимими та

відповідають обмеженням задачі, зазвичай мають вищу придатність, ніж ті, які цього не роблять.

Функція придатності для простих задач може бути зведена до прямої відповідності між бажаним результатом у задачі та значенням придатності. Однак, іноді задача більш складна, і для досягнення високої продуктивності потрібні більш складні функції придатності.

Для ефективного використання оцінок придатності необхідно, щоб вони існували на неперервній шкалі, де незначне покращення рішення призводить до відповідного збільшення його оцінки. Таким чином, особина може мати майже ідентичних нащадків, які роблять маленькі кроки до покращення, тим самим стимулюючи продовження еволюційного процесу покращення.

Зазвичай оцінка придатності особини вимагає дослідження та застосування представленого кандидата рішення до базової проблеми, що може бути ресурсомісткою задачею залежно від того, наскільки складною є оцінка базової проблеми.

Розглядаючи детальніше, оцінка придатності є процесом, який може значно відрізнятися в залежності від конкретного застосування. У деяких випадках можливо спростити функцію придатності до лінійної форми, де оцінка придатності напряму відображає доцільність особини.

Для популяції розміром N, присвоюючи найкращій особині ранг 1, а найгіршій N - 1, і представляючи ранг через r та відносну пристосованість через f = f(r), ці відображення можна записати наступним чином:

$$f(r) = s - (s - 1) \cdot \frac{2r}{N - 1}$$

де s, $1 < s \leq 2$, є бажаною відносною пристосованістю для найкращої особини. Верхня межа для s виникає тому, що пристосованість повинна бути невід'ємною для всіх особин, зберігаючи при цьому $\sum_{i=0}^{N-1} f(i) = N$.

У більш складних ситуаціях функція придатності може включати різноманітні параметри, такі як штрафи за порушення обмежень чи додаткові

бали за досягнення певних критеріїв, що збільшує складність її обчислення, але дозволяє точніше оцінити якість рішень.

1.4 Селекція (вибір батьківських особин)

Селекція або реплікація — це методика відбору особин для участі в створенні потомства, залежно від їхньої пристосованості. Однак, оскільки кількість потомства має бути цілим числом, фактична кількість отриманих нащадків часто відрізняється від запланованої, що призводить до так званої помилки селекції.

Для того, щоб уникнути систематичних помилок селекції, у генетичних алгоритмах (ГА) селекцію зазвичай здійснюють стохастично. Очікуване число потомства кожної особини має відповідати її пристосованості, тоді як стохастичні помилки селекції повинні бути мінімізовані. Бейкер у 1987 році запропонував для досягнення цих двох цілей алгоритм вибірки, який назвав Стохастичним Універсальним Відбором (СУВ). Цей метод був висунутий як альтернатива Вибірці Рулетки Голдберга (Рулетка Голдберга, 1989 р.), яка дозволяла виникати значним помилкам селекції (рис. 1.2).

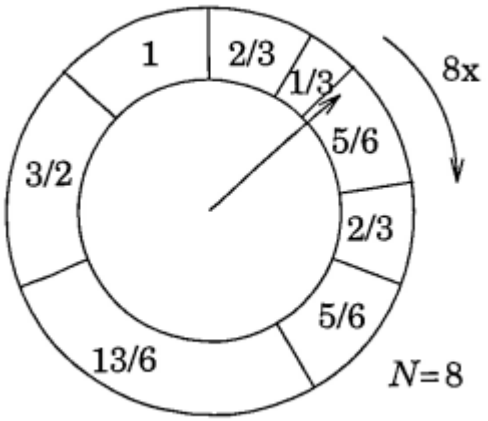

Рис. 1.2 Вибір колеса рулетки

Селекція за допомогою СУВ може бути уявлена як результат обертання рулетки зі слотами, ширина яких пропорційна пристосованості особин у

популяції, але з кількома однаково розміщеними вказівниками (рис. 1.3). Після зупинки рулетки число вказівників, що вказують на кожен сектор, має відповідати найближчому більшому або меншому числу потрібних нащадків, забезпечуючи таким чином мінімальні відхилення від бажаного розподілу. Отримані реплікати необхідно перемішати перед тим, як алгоритм перейде до рекомбінації та мутації.

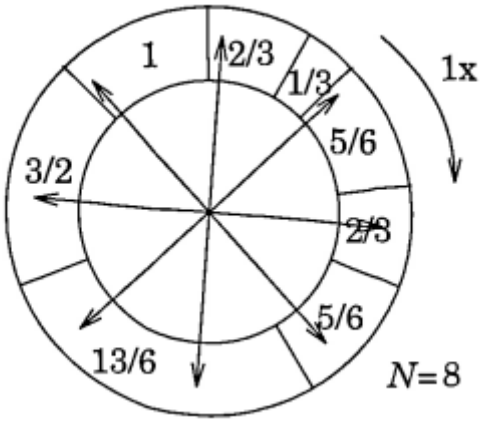

Рис. 1.3 Стохастичний Універсальний Відбір

У крайньому випадку, коли треба вибрати лише одну особину, СУВ є фактично еквівалентним Рулетці Голдберга. При виборі декількох особин перевага СУВ перед Рулеткою Голдберга зростає з кількістю генерованих нащадків. Зазвичай немає причин використовувати Рулетку Голдберга в таких умовах.

Доволі іншим підходом до селекції є турнірний відбір. У його найпростішій формі пари особин випадково витягують з популяції, і особини кожної пари порівнюють між собою. Найкраща особина кожної пари оголошується переможцем турніру та вибирається.

Турнірний відбір особливо зручний, коли порівняти особин легше, ніж визначити їхній абсолютний показник, що часто відбувається у генетичному програмуванні. Однак, як і у випадку Рулетки Голдберга, турнірний відбір з випадковими турнірами може призводити до великих помилок селекції. При виборі декількох особин має бути можливість ранжувати популяцію з будь-якою

кількістю попарних порівнянь або турнірів, і використовувати СУВ для вибору потомства згідно з цим ранжуванням. Чим більше особин порівнюється, тим точніший буде процес селекції.

Турнірний відбір схожий на ранжування тим, що він не залежить від масштабної інформації. Змінюючи розмір і кількість турнірів, можна контролювати ефективну відносну пристосованість найкращої особини, хоча і менш гнучко, ніж за допомогою ранжування. Зменшення ймовірності перемоги найкращої особини в турнірі дозволяє більш плавно контролювати очікувану відносну пристосованість найкращої особини в популяції. Вирішення турнірів з ймовірністю, яка залежить від відносної пристосованості двох конкурентів, призводить до зниження диференціації пристосованості по мірі зближення популяції.

Ранжування — це інший метод селекції, який використовується у генетичних алгоритмах для забезпечення диференціації між особинами без прямого відношення до їх абсолютної пристосованості. Особини сортуються за їхньою пристосованістю, і кожній присвоюється ранг. Це зменшує вплив екстремальних значень пристосованості, які можуть виникнути в популяціях з великим розмаїттям. Відбір, заснований на ранжуванні, визначає ймовірність вибору особини для репродукції на основі її рангу, замість абсолютного значення пристосованості, що допомагає підтримувати різноманіття у популяції та уникати передчасної конвергенції на локальні оптимуми.

1.5 Схрещування (рекомбінація)

У генетичних алгоритмах (ГА) застосовуються два типи операторів варіацій: рекомбінація та мутація. Вони дозволяють індивідам змінюватися та вдосконалюватися. Рекомбінація є аналогом розмноження у ГА. Два або більше індивідів вибираються алгоритмом вибору батьків та породжують потомство.

Створення потомства залежить від алгоритму рекомбінації. Існує кілька загальних операцій рекомбінації, які використовуються багатьма ГА у

різноманітних ситуаціях. Вони служать в якості хороших орієнтирів для порівняння з більш специфічними для проблем операторами рекомбінації.

Загальні оператори включають одноточковий кросовер (рис.1.4), багатоточковий кросовер та уніформний кросовер. Ці оператори працюють, беручи перший "шматок" представлення першого батька, потім другий "шматок" представлення другого батька і комбінуючи ці шматки у першому потомстві. Той самий процес повторюється, але для другого "шматка" першого батька та першого "шматка" другого батька. Це створює мікс генів від кожного з батьків у потомстві. "Шматок" є частиною представлення, приклад "нарізки" можна побачити на рис. 1.5.

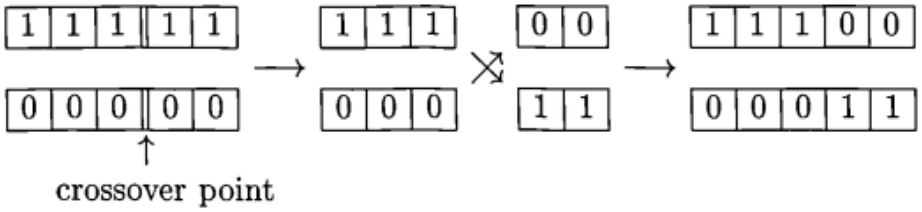

Рис.1.4 Одноточковий кросовер

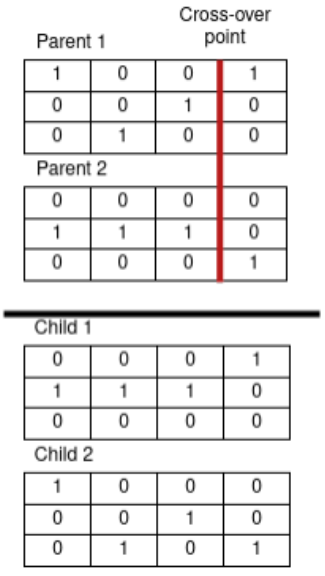

Рис.1.5 Приклад міксу генів від кожного батька у потомстві

Існують важливі аспекти якісного алгоритму рекомбінації, такі як успадковуваність, поважна успадковуваність та валідність. Успадковуваність

рекомбінації означає, що потомство успадковує гени від своїх батьків, сильна успадковуваність означає, що оператор виробить копію батьків, якщо батьки є ідентичними. Оператор рекомбінації вважається поважним, якщо спільні рішення батьків зберігаються. Оператор рекомбінації є асортуючим, якщо відстань між батьками та потомством менша, ніж відстань між самими батьками, де відстань визначається як схожість. Бажано, щоб оператори рекомбінації створювали валідні рішення з валідних батьків, проте для обмежених проблем це може бути важко, а інколи і неможливо гарантувати.

1.6 Мутація

Іншим типом оператора варіації є мутація, яка є унарною операцією. Це означає, що для вхідних даних потрібен лише один індивід. Мутації, як правило, застосовуються статистично з певною частотою мутацій та вносять невеликі зміни в індивіда. Ці зміни також зазвичай стохастичні і призначені для створення різноманітності в популяції, не створюючи такої великої різниці, як при рекомбінації. Загалом існують деякі узагальнені мутаційні оператори, які можуть бути використані майже для всіх проблем генетичного алгоритму. Зазвичай вони передбачають обмін значеннями генів у індивіда (рис. 1.6).

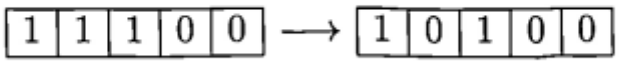

Рис. 1.6 Приклад бітової мутації

1.7 Відбір на виживання

Відбір на виживання – це операція, що відповідає за видалення зайвих індивідів після рекомбінації та мутації. Зазвичай відбір на виживання прагне стабілізувати популяцію так, щоб вона не зменшувалась і не збільшувалась з покоління в покоління. Немає єдиної думки щодо найкращого типу оператора

відбору на виживання, але в одному дослідженні аргументується, що кращі результати для задачі призначення досягаються шляхом видалення найменш пристосованих індивідів.

В іншому джерелі стверджується, що замість використання одновимірного значення пристосованості краще застосовувати двовимірний підхід із значеннями пристосованості та непристосованості, оскільки відбір на виживання показує кращі результати, якщо використовується лише значення непристосованості. Існує думка, що часто краще не видаляти найгірших індивідів, оскільки це зменшує різноманіття, а натомість слід використовувати статистичний підхід.

1.8 Локальний пошук

Локальний пошук — це операція, яка має на меті знаходження локальних оптимумів шляхом розгляду найкращого рішення, аналізу всіх його сусідів та вибору найкращого з них, продовжуючи процес до досягнення точки, де жоден сусід не є кращим. Проведення всебічного пошуку всього сусідства може бути затратним по часу, тому алгоритми локального пошуку іноді модифікують у способі вибору сусіда, до якого слід перейти, якщо знаходять поліпшення.

Існують три варіанти локального пошуку. Пошук кращого поліпшення, при якому оцінюються всі сусіди, і той, що дає найбільше поліпшення, вибирається для наступного циклу оцінювання. Пошук першого поліпшення, де одразу вибирається сусід, що покращує рішення для наступного циклу без продовження дослідження цього сусідства, якщо поліпшення не знайдене, виконується повне дослідження сусідства. Випадковий вибір, де серед тих, що покращили рішення, випадковим чином вибирається сусід для наступного циклу. На практиці стратегія пошуку першого поліпшення призводить до рішень тієї ж якості, що і стратегія пошуку кращого поліпшення, використовуючи при цьому менше обчислювального часу.

Проблемою, яка може виникнути під час виконання локального пошуку, є швидка конвергенція до локального оптимуму. Цей ефект пом'якшується, якщо використовується стратегія пошуку першого поліпшення замість пошуку кращого поліпшення, оскільки вона має повільніший темп конвергенції. Ця рання конвергенція збільшує чутливість до початкової популяції. Локальний пошук є корисним оператором, коли у просторі пошуку не так багато локальних оптимумів або якщо якість різних локальних оптимумів є більш-менш схожою.

1.9 Параметри генетичного алгоритму

Генетичний алгоритм складається з набору значень, які окреслюють його функціонування. Серед них - швидкість мутації, розмір популяції, максимальна кількість поколінь, які слід оцінити, алгоритм рекомбінації тощо. Ці значення називають параметрами. Вони поділяються на дві категорії: параметри з чисельними доменами, як от розмір популяції, швидкість мутації, і параметри з неупорядкованими доменами, які називають символічними, що включає вибір алгоритмів для рекомбінації та мутації. Зазвичай різні значення символічних параметрів описують різні випадки застосування генетичного алгоритму, тоді як чисельні значення визначають роботу конкретного випадку алгоритму. Тип обраних параметрів може значно вплинути на ефективність алгоритму у пошуку рішення та на швидкість цього процесу. Кращі параметри залежать від конкретної задачі. Тому визначення правильних параметрів є істотною частиною вирішення проблеми за допомогою генетичного алгоритму.

Через труднощі у виборі оптимальних параметрів було розроблено декілька методів. Простий метод - це спробувати різні варіанти на практиці і подивитися, що працює краще. Проте такий підхід забирає багато часу, особливо при великій кількості параметрів. Інший метод полягає у використанні генетичних алгоритмів для вирішення самої оптимізаційної задачі вибору параметрів. З цього виникла ідея мета-генетичного алгоритму, який оптимізує

рівень придатності і кількість поколінь основного генетичного алгоритму, змінюючи його параметри.

Проте, використання мета-генетичного алгоритму вимагає додаткового часу для оцінювання, адже необхідний повний запуск генетичного алгоритму, щоб оцінити придатність кожної особини. Існує також проблема визначення параметрів для самого мета-генетичного алгоритму.

Вплив параметрів генетичного алгоритму може варіюватися протягом його виконання. Наприклад, на початку процесу може бути ефективним високий рівень мутації, а до кінця - низький. Тому неможливо закріпити параметр за статичним значенням, та вводиться поняття динамічного параметру. Динамічний параметр - це такий, що може змінюватися під час одного циклу виконання генетичного алгоритму. Він представлений функцією, а не фіксованим значенням. Використання динамічних параметрів забезпечує кращі результати, але водночас виникає завдання знайти оптимальні функції для цих параметрів, подібне до вибору статичних параметрів.

Існує три способи контролю динамічних параметрів. Детермінований контроль передбачає зміну параметрів за попередньо визначеним алгоритмом без зворотного зв'язку з пошуковим процесом, за винятком виміру часу. Адаптивний контроль означає зміну параметрів на основі зворотного зв'язку ззовні від алгоритму ГА. Самоадаптивний контроль відбувається, коли ГА дозволяє особинам самостійно визначати, які параметри до них застосовуються.

Це означає, що значення параметрів підлягають оптимізації через природний відбір у алгоритмі. Один із прикладів - кожна особина описує свій власний рівень мутації як ген. Це дозволяє різним особинам створювати потомство з різними швидкостями мутації, гарантуючи, що рівень мутації змінюється тоді, коли це вигідно. Однак недоліком є збільшення складності представлення та обсягу простору пошуку через необхідність опису швидкості мутації.

Одна з істотних проблем, яка виникає при оцінці простору параметрів, будь то динамічні чи статичні, полягає в тому, що для кожної оцінки потрібно

проводити повний цикл ГА, щоб визначити ефективність цих параметрів. Якщо основна задача для ГА є ресурсоємною, оцінка ГА швидко стає дуже вимогливою до ресурсів задачею. Щоб полегшити ресурсомістке оцінювання реальної проблеми, іноді можна знайти замісний об'єкт, який приблизно відтворює поведінку основної проблеми, але є менш вимогливим до ресурсів. Схожість між замісним об'єктом та реальною проблемою має бути лише у тому, які параметри ГА ефективні для її вирішення, а не в самій суті проблеми. Якщо вдається знайти такий замісний об'єкт, оцінювання того, які параметри ефективні для рішення проблеми, стає набагато менш ресурсоємним, і ці знання можна використовувати для реального алгоритму.

РОЗДІЛ 2 ІСТОРІЯ ТА ЗАСТОСУВАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ У ІГРАХ

2.1 Історія розвитку

Початкові етапи

Зародження ідеї генетичних алгоритмів відноситься до 1970-х років, коли Джон Голланд представив концепцію алгоритмів, що імітують природний відбір та генетичне розмаїття в процесах пошуку та оптимізації. Проте, до ігрової індустрії ці ідеї прийшли значно пізніше, коли з'явилася потреба в більш складному та непередбачуваному штучному інтелекті.

1980-1990-ті роки

Впродовж 1980-х та на початку 1990-х, генетичні алгоритми почали застосовуватись у простих іграх, зокрема для оптимізації поведінки неперсонажів та покращення ігрових стратегій. Це були перші кроки до створення адаптивного штучного інтелекту, який міг би самовдосконалюватись у відповідь на дії гравця.

Розвиток у 2000-ті

На початку 2000-х з розвитком технологій та збільшенням обчислювальних потужностей генетичні алгоритми стали використовуватися для розв'язання більш складних задач у іграх. Вони використовувалися для автоматичного генерування рівнів, балансування геймплея та навіть створення унікальних ігрових сутностей.

2.2 Перші ігри написані з використання генетичного алгоритму

Одним з відомих прикладів застосування еволюційних алгоритмів в іграх є "Creatures", випущена в 1996 році компанією Cyberlife (тепер Creatures Labs). У

цій грі гравці виховують створінь званих "Norns", штучних життєвих форм, які мають власну унікальну ДНК, яка еволюціонує з покоління в покоління (рис. 2.1). Ці створіння вчились і адаптувались до світу навколо них, використовуючи систему, що містить елементи генетичних алгоритмів.

Рис. 2.1 Гра "Creatures"

Ще одним раннім прикладом використання генетичних алгоритмів в іграх є програма "Galapagos: Mendel's Escape" від Anark (рис. 2.2). У цій грі, випущеній у 1997 році, генетичні алгоритми використовуються для моделювання адаптивної поведінки головного персонажа - створіння на ім'я "Mendel". Можливість Mendel адаптуватися і вчитися в процесі гри була центральним елементом ігрового процесу.

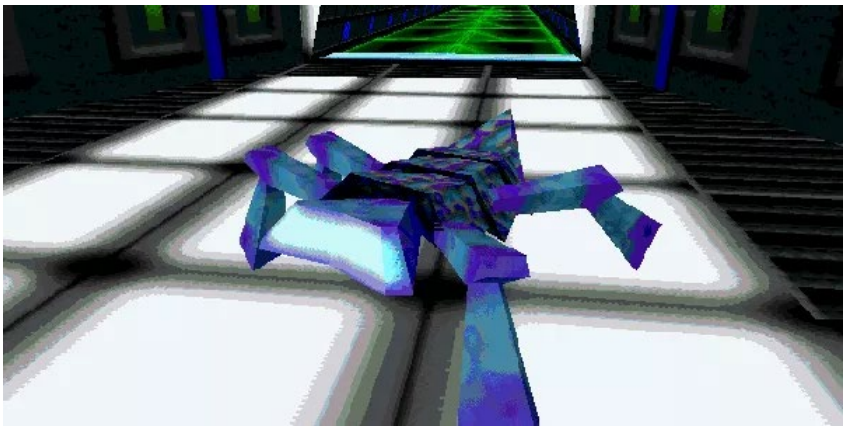

Рис. 2.2 Гра «Galapagos: Mendel's Escape»

Ці ігри і багато інших експериментальних проектів засвідчили потенціал генетичних алгоритмів у розробці ігор, дозволяючи створювати більш динамічні та адаптивні ігрові середовища. Водночас, важливо зазначити, що генетичні алгоритми зазвичай застосовувались у комбінації з іншими методами штучного інтелекту для досягнення більшої ефективності та глибини поведінки в іграх.

2.3 Еволюційне моделювання у генетичних алгоритмах ігор

Еволюційне моделювання є одним із захоплюючих застосувань генетичних алгоритмів у сфері комп'ютерних ігор. Воно імітує біологічний процес природного добору, дозволяючи віртуальним організмам або агентам "еволюціонувати" з часом, щоб адаптуватися до змінюваного середовища, яке часто визначається діями та стратегіями гравців.

Основні принципи

Генетичні алгоритми використовуються для моделювання процесів еволюції. Ігрові агенти мають набори "генів" - параметри, що визначають їхню поведінку, здібності, зовнішній вигляд, тощо. Протягом ігрового процесу, ці агенти піддаються таким процесам (рис. 2.3):

1. ***Ініціалізація популяції:*** Алгоритм починається з популяції потенційних рішень проблеми. У контексті гри ці рішення можуть бути різними стратегіями або наборами параметрів, які контролюють поведінку ігрового об’єкта.
2. ***Оцінка придатності:*** кожен член популяції оцінюється на основі функції придатності. В іграх це може означати, наскільки добре працює стратегія в ігровому середовищі, наприклад, скільки виграшів конкретна стратегія накопичує протягом кількох раундів.
3. ***Відбір:*** на основі оцінки придатності алгоритм вибирає найбільш підготовлених людей, які стануть батьками наступного покоління. Це часто передбачає процес, схожий на природний відбір, коли найбільш підготовлені особини мають більшу ймовірність розмноження.

4. ***Схрещування:*** процес схрещування або спаровування поєднує в собі особливості двох батьківських рішень для створення нащадків для наступного покоління. В іграх це може означати змішування двох стратегій, щоб сподіватися поєднати сильні сторони обох.
5. ***Мутація:*** щоб зберегти генетичне різноманіття та потенційно запровадити нові стратегії, невеликі випадкові зміни вводяться до деяких нащадків. Для ігор це може передбачати дещо зміну стратегії або поведінки гравця для дослідження нових потенційних рішень.
6. ***Нове покоління:*** нащадки формують нове покоління рішень, і процес повторюється з кроку 2. Протягом наступних поколінь популяція еволюціонує, виробляючи все більш придатні стратегії.
7. ***Припинення:*** Алгоритм припиняє роботу, коли знайдено задовільне рішення або після того, як мине фіксоване число поколінь або час.

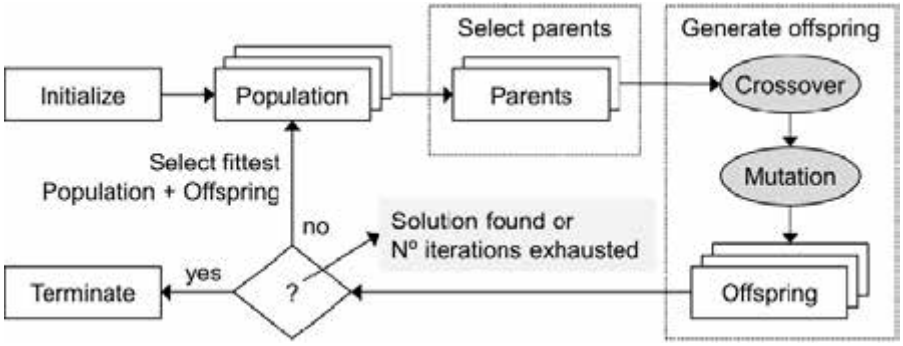

Рис. 2.3 Схема роботи еволюційного моделювання

Застосування в іграх

У комп'ютерних іграх, еволюційне моделювання може бути використане для різних цілей:

Розвиток штучного інтелекту протягом гри: Ігрові персонажі або сутності можуть адаптуватися до стратегій гравця, змінюючи свою поведінку або зовнішність, що робить гру більш динамічною та непередбачуваною.

Генерація унікальних противників: Замість стандартних ворогів, кожен гравець може стикатися з унікально еволюційно розвинутими противниками, що забезпечують унікальний ігровий досвід.

Балансування гри: Еволюційне моделювання може автоматично адаптувати складність гри до рівня навичок гравця, зберігаючи постійний виклик.

Еволюційний геймплей: Ігри, які включають елементи виживання або стратегії, можуть використовувати еволюційне моделювання для створення реалістичних сценаріїв, де агенти мають розвиватися, щоб вижити в змінних умовах.

Приклади

Один із класичних прикладів використання еволюційного моделювання у грі - "Galactic Arms Race" (GAR), де збройні системи на кораблях еволюціонують відповідно до переваг гравців. Інший приклад - це "Niche - a genetics survival game", яка дозволяє гравцям створювати та управляти власними видами тварин, ведучи їх через процеси еволюції.

2.4 Балансування гри за допомогою генетичних алгоритмів

Балансування гри є ключовим аспектом геймдизайну, який забезпечує справедливість, конкурентоспроможність та задоволення від ігрового процесу. Генетичні алгоритми (ГА) пропонують ефективний підхід до автоматизації процесу балансування, особливо в складних іграх, де безліч змінних елементів можуть впливати на ігровий досвід.

Принципи балансування генетичними алгоритмами

- ***Пошук оптимальних параметрів:*** ГА можуть автоматично тестувати різні комбінації ігрових елементів, таких як сила персонажів, вартість ресурсів чи ефективність зброї, щоб знайти найбільш збалансовані налаштування.

- ***Адаптація до стилів гри:*** Алгоритми можуть аналізувати дані про те, як гравці використовують різні ігрові елементи, та вносити корективи, щоб усі стилі гри були життєздатними.
- ***Динамічне балансування:*** В реальному часі ГА можуть відстежувати ефективність гравців та вносити зміни у гру, щоб підтримувати виклик та інтерес.

Процес балансування

Визначення цілей балансу: Розробники визначають, що означає "баланс" у контексті їхньої гри, наприклад, час проходження рівня або успішність різних стратегій.

Кодування геному: Ігрові параметри кодуються як "гени" в генетичному алгоритмі, що дозволяє їх мутувати та рекомбінувати.

Формування популяції: Створюється початкова популяція можливих ігрових налаштувань.

Селекція: Вибираються найбільш обіцяючі налаштування з популяції на основі заданих критеріїв балансу.

Схрещування та мутація: Найкращі налаштування комбінуються та модифікуються для створення нового покоління налаштувань.

Оцінка: Нове покоління тестується на баланс, як правило, через автоматизовані матчі або симуляції.

Ітерація: Процес повторюється до тих пір, поки не буде досягнуто оптимального балансу.

Приклади застосування

Multiplayer Online Battle Arenas (MOBAs): ГА можуть балансувати сили і здібності героїв, враховуючи велику кількість змінних та комбінацій.

Карточні ігри: Алгоритми можуть аналізувати та балансувати карти, забезпечуючи рівні умови для різних колод та стратегій.

Стратегічні ігри: ГА використовуються для балансування ресурсів, юнітів та технологій, щоб жодна стратегія не була переважно домінантною.

2.4 Процедурна генерація контенту за допомогою генетичних алгоритмів

Процедурна генерація контенту (PGC) у відеоіграх – це метод створення даних алгоритмічно, а не вручну. Цей підхід дозволяє створювати великі та різноманітні ігрові світи, які можуть забезпечити унікальний ігровий досвід для кожного гравця. Генетичні алгоритми (ГА) є одним із методів PGC, дозволяючи розвивати контент, який адаптується та еволюціонує з часом.

Концепції процедурної генерації з ГА

- ***Створення карт:*** ГА можуть генерувати унікальні ландшафти та рівні, адаптуючи їх до складності та стилю гри, який вимагається.
- ***Генерація NPC (негравцевих персонажів):*** ГА можуть використовуватися для визначення зовнішності, здібностей та поведінки NPC, забезпечуючи більшу різноманітність ігрових взаємодій.
- ***Розвиток складності:*** ГА можуть застосовуватись для налаштування складності гри в залежності від рівня навичок гравця.

Процес процедурної генерації з ГА.

- ***Визначення критеріїв:*** Визначаються параметри, за якими буде оцінюватися згенерований контент, такі як різноманітність, складність або грайливість.
- ***Ініціалізація:*** Створюється початкова популяція потенційних контентних елементів, таких як ландшафти, персонажі чи завдання.
- ***Оцінка:*** Кожен елемент контенту оцінюється на основі заданих критеріїв.
- ***Селекція:*** Вибираються найкращі варіанти, які відповідають критеріям найбільше.
- ***Схрещування та мутація:*** Відібрані елементи комбінуються або модифікуються для створення нових варіантів контенту.

- ***Ітерація:*** Процес повторюється, створюючи нові покоління контенту до тих пір, поки не буде досягнуто задовільного результату.

Приклади застосування

"No Man's Sky": Використовує PGC для створення величезної кількості унікальних планет і екосистем.

"Diablo": Генерує випадкові рівні з різноманітними ворогами та предметами, забезпечуючи новий ігровий досвід при кожному проходженні.

"Minecraft": Створює величезні випадково генеровані світи, де кожна гра відрізняється від попередньої.

2.5 Штучні екосистеми за допомогою генетичних алгоритмів

Штучні екосистеми в іграх – це симуляції живих систем, де різні сутності взаємодіють між собою та зі своїм середовищем. Вони можуть включати флору, фауну, погодні умови, природні ресурси тощо. Генетичні алгоритми (ГА) дозволяють цим екосистемам розвиватися і змінюватися у відповідь на дії гравця та інші події у ігровому світі.

Основні принципи штучних екосистем з ГА

- ***Взаємодія і залежність:*** Елементи екосистеми взаємодіють у складних і динамічних способах, формуючи харчові ланцюги та екологічні ніші.
- ***Адаптація та еволюція:*** Сутності адаптуються до змін у середовищі, з їхнім біологічним різноманіттям та поведінкою, яке формується за допомогою мутацій та природного відбору, що симулюються ГА.
- ***Динамічний баланс:*** Екосистема постійно знаходиться в русі, із змінами, що відбуваються в результаті різних факторів, включаючи зміну клімату, хвороби, хижацтво, інвазивні види та інші екологічні збурення.

Процес розвитку штучних екосистем з ГА

Ініціалізація: Створюється початкове середовище з базовим набором сутностей і правил взаємодії.

Симуляція: Сутності живуть, розмножуються, харчуються та взаємодіють один з одним та зі своїм середовищем.

Мутація та відбір: Варіації в характеристиках сутностей з'являються внаслідок мутацій; ті, що краще пристосовані до середовища, мають більше шансів на виживання.

Оцінка та адаптація: Екосистема оцінюється на основі заданих критеріїв (наприклад, стабільність, біорізноманіття), і зміни вносяться для досягнення бажаних характеристик.

Еволюція: Через низку поколінь сутності еволюціонують, розвиваючи нові характеристики та поведінку.

Балансування: Екосистема балансується для запобігання занадто швидких або екстремальних змін, що можуть призвести до її занепаду.

Приклади застосування

"SimLife": Гра, де гравці можуть створювати та керувати власними екосистемами, спостерігаючи за еволюцією та адаптацією організмів.

"Spore": Дозволяє гравцям проектувати створінь, які потім адаптуються та еволюціонують у відповідь на середовище та дії інших створінь.

"Eco": Ігровий світ, де дії гравців безпосередньо впливають на екосистему, яка може змінюватися і розвиватися відповідно до цих дій.

2.6 Навчання штучного інтелекту за допомогою генетичних алгоритмів

Генетичні алгоритми (ГА) можуть бути використані для навчання штучного інтелекту (ШІ), зокрема в іграх, де потрібно швидко адаптуватися до

дій гравця і виконувати складні стратегії. Використання ГА в навчанні ШІ полягає в оптимізації параметрів алгоритму або нейронної мережі, щоб досягти бажаного поведінки.

Ключові концепції

Популяція: Набір потенційних рішень (наприклад, набори параметрів нейронної мережі), які мають бути оптимізовані.

Геном: Представлення індивідуальних характеристик ШІ в популяції, які будуть мутувати та спрямовуватися.

Фітнес-функція: Оцінка, яка вимірює ефективність ШІ в ігровому контексті, зазвичай на основі його успіху або продуктивності.

Селекція: Відбір найкращих індивідів для репродукції та передачі їхніх генів наступному поколінню.

Кросовер: Комбінування генів двох індивідів для створення нащадків.

Мутація: Випадкові зміни у геномі, які сприяють різноманітності і дослідженню нових стратегій.

Процес навчання ШІ з ГА

Ініціалізація: Створення початкової популяції ШІ з випадково сгенерованими параметрами.

Оцінка фітнесу: Кожен ШІ оцінюється за його ефективністю в ігрових завданнях.

Селекція: Найкращі ШІ обираються для розмноження на основі їхнього фітнесу.

Репродукція: Кросовер і мутації застосовуються для створення нових ШІ.

Ітерація: Процес повторюється з новим поколінням ШІ.

Конвергенція: Процес продовжується до тих пір, поки не буде досягнуто задовільної ефективності або фіксованої кількості ітерацій.

Приклади застосування

Навчання ігрових ботів: ГА можуть оптимізувати стратегії ботів у стратегічних іграх, щоб вони могли краще конкурувати з людьми або адаптуватися до їхньої гри.

Персоналізація досвіду гри: ШІ може бути навчений забезпечувати індивідуальний досвід для кожного гравця, змінюючи складність або стиль гри.

Оптимізація нейронних мереж: ГА можуть використовуватися для налаштування ваг і гіперпараметрів у нейронних мережах, які керують ШІ в іграх.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ГЕНЕТИЧНОГО АЛГОРИТМУ У ІГРОВОМУ РУШІЇ UNITY

3.1 Unity як ігровий рушій для розробки ігор

Unity – це передовий ігровий рушій, який надає розробникам потужний набір інструментів для створення ігор та інтерактивних досвідів. Він підтримує ряд платформ, включаючи PC, консолі, мобільні пристрої та веб, забезпечуючи високий ступінь портативності для розробленого контенту.

Підтримка Мов Програмування

Unity підтримує C#, який є однією з найбільш популярних мов програмування у світі розробки ігор. C# має багатий набір функцій, що дозволяє легко імплементувати складні алгоритми, включаючи генетичні алгоритми.

Графічний та Фізичний Двигуни

Unity включає потужний графічний двигун, який може відтворювати високоякісні візуальні ефекти, та фізичний двигун, який дозволяє реалізовувати реалістичні моделі руху та колізії. Ці можливості є важливими для створення ігрових світів, де можуть застосовуватися генетичні алгоритми, наприклад, для моделювання поведінки істот або для процедурної генерації терену.

Інтегровані Системи Розробки

Unity надає інтегровані інструменти для роботи з анімацією, AI, вводом/виводом, мережею та багатьма іншими компонентами, які є необхідними для розробки ігор. Використання генетичних алгоритмів з цими компонентами може підвищити рівень динамічності та адаптивності ігрового процесу.

Підтримка Генетичних Алгоритмів

Unity ідеально підходить для реалізації генетичних алгоритмів завдяки своїй гнучкості та потужності. Генетичні алгоритми можуть бути інтегровані безпосередньо у ігровий цикл для управління поведінкою NPC, процедурної генерації контенту, адаптивного балансування гри та інших аспектів ігрового дизайну.

Масштабованість та Оптимізація

Unity пропонує інструменти для оптимізації та масштабування проектів, що дозволяє розробникам створювати ігри різної складності, від малих інді-ігор до великих AAA-проектів. Це особливо корисно при впровадженні генетичних алгоритмів, які можуть бути ресурсоємними при роботі з великою кількістю агентів або складними сценаріями.

Спільнота та Підтримка

Unity має велику та активну спільноту розробників, з якою можна обмінюватися знаннями, отримувати допомогу та співпрацювати. Існує багато ресурсів, туторіалів та готових рішень для реалізації генетичних алгоритмів, що спрощує їх використання в ігрових проектах.

3.2 Реалізація генетичного алгоритму в балансуванні ворогів.

Реалізований генетичний алгоритм використовує геном, який складається з таких значень:

Здоров’я, шкода, швидкість. Популяція ворогів створюється за рандомними значеннями геному (рис. 3.1).

Значення максимуму і мінімуму задається користувачем.

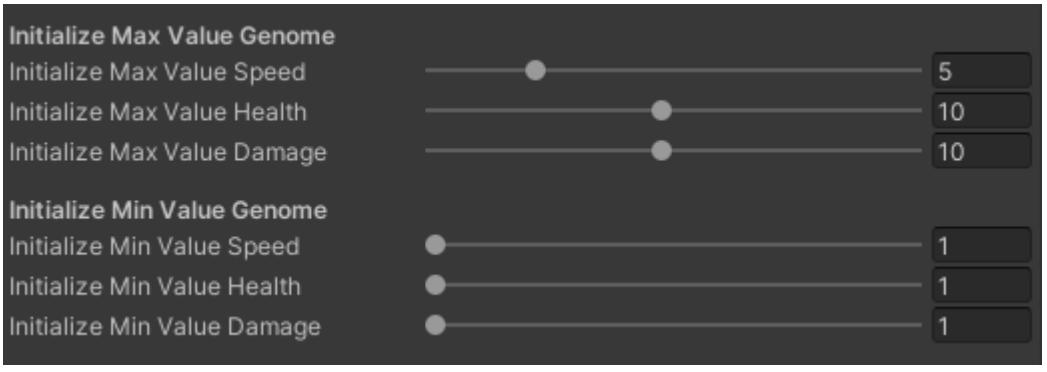

Рис. 3.1 Налаштування значень геному в Unity

Після ініціалізації вороги з’являються на карті, рухаються та нападають на гравця, який автоматично рухається по згенерованій стежці (рис. 3.2).

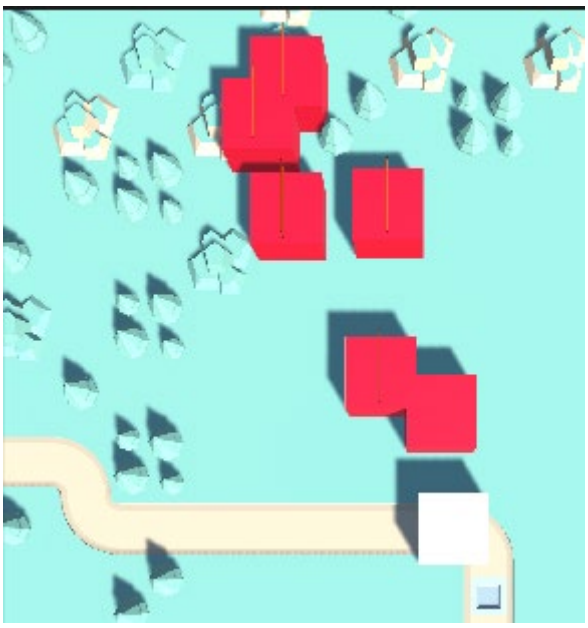

Рис. 3.2 Рух ворогів до гравця

Під час раунду працює дві системи, які відслідковують дані про раунд.

Перша система ***AnalyticsPlayer.*** Вона відслідковує, час раунду, кількість натисків за секунду. В кінці раунду вона підраховує результативність гравця за такими параметрами (рис 3.3-3.4).

```
private float CalculationResultRound(float roundTime, float percentClickToEnemy, int damageForRound)
{
    float result = 0;

    result += (_maxDurationRound - roundTime) * _coefficientDurationRound;

    result += 1f - (1f - percentClickToEnemy) * _coefficientAccuracy;

    result += (_maxTakeDamage - damageForRound) * _coefficientTakingDamage;

    return result;
}
```

Рис. 3.3 Функція розрахунку результатів раунду

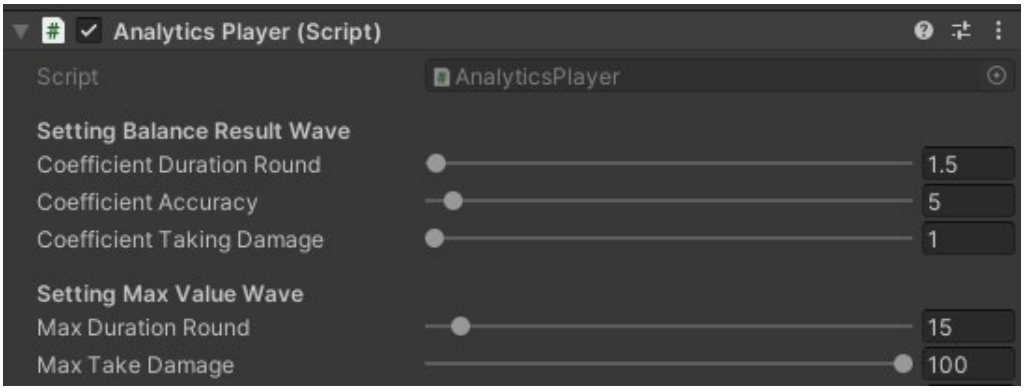

Рис. 3.4 Параметри розрахунку результативності гравця

Значення, яке отримане з цієї функції записується в массив, для того, щоб порівняти це значення з попереднім значенням, якщо значення виросло, то це дозволяє ворогам мутувати, так, як гравець призвичаївся до нових ворогів.

Друга система це ***EnemyPopulation***, яка приймає дані про знищених ворогів гравцем. Дані включають в себе такі значення (рис. 3.5)

```
public class EnemyLiveData
{
    public float LiveTime;
    public float DamageToTarget;
    public float DistanceToTarget;
    public float CountAttack;
}
```

Рис. 3.5 Дані про знищених ворогів

LiveTime – тривалість життя ворога

DamageToTarget – шкода нанесена гравцю

DistanceToTarget – яка відстань до гравця в момент знищення ворога

CountAttack – кількість атак, яку наніс ворог гравцю.

Після знищення популяції запускається цикл еволюції.

1. Визначається значення пристосованості кожного гена, за допомогою фітнес функції за такими формулами (рис. 3.6).

```
public float CalculateFitnessForEnemy(EnemyData enemyData)
{
    float fitness = 0.0f;

    float valueDistanceBeforePlayer = FitnessDistanceBeforePlayer(enemyData.LiveData.DistanceToTarget);

    fitness += valueDistanceBeforePlayer * DistanceToPlayerWeight;

    if(enemyData.LiveData.CountAttack == 1)
    {
        fitness += AttackToPlayerWeight;
    }
    else if(enemyData.LiveData.CountAttack > 1)
    {
        fitness -= enemyData.LiveData.CountAttack * AttackToPlayerWeight;
    }

    if (enemyData.LiveData.LiveTime > MaxLiveTime)
    {
        fitness -= enemyData.LiveData.LiveTime - MaxLiveTime * OvercomingLiveTimeWeight;
    }

    if (enemyData.Genome.speed > MaxValueSpeed)
    {
        fitness -= enemyData.Genome.speed - MaxValueSpeed * OvercomingValueWeight;
    }

    if (enemyData.Genome.health > MaxValueHealth)
    {
        fitness -= enemyData.Genome.health - MaxValueHealth * OvercomingValueWeight;
    }

    return fitness;
}
```

Рис. 3.6 Розрахунок фітнесу ворога

Селекційний метод я обрав елітарний. Цей метод передбачає, що одна або декілька найкращих особин покоління гарантовано передають свої гени до наступного покоління. Я обрав саме цей метод, тому що вибірка моєї популяції становить 6 геномів. І інші методи селекції при такій малій кількості генів призводили до передчасної конвергенції та втрати генетичного різноманіття, а це в свою чергу не підходить під поставлену задачу.

Стратегією для схрещування, я обрав асортативне схрещування. Такий підхід до схрещування зосереджується на поширенні генів найкращого індивіда по популяції, сподіваючись, що це прискорить прогрес до оптимуму. Таке схрещування може бути ефективним у певних ситуаціях, наприклад, коли оптимальне рішення вимагає наявності певних ключових генів, які вже присутні у найкращому індивіді.

Також, я використовую функцію мутації, яка урізноманітнює мою популяціє і рятує її від втрати генетичного різноманіття. Для контролю мутації,

я використовую різницю прогресу гравця з поперденім раундом. Це дає змогу навикам гравця контролювати прогрес популяції (рис. 3.7).

Рис. 3.7 Розрахунок мутації

Після цих кроків нова популяції створюється і це повторюється циклічно.

3.3 Реалізація генетичного алгоритму в створенні карти

Використання генетичних алгоритмів у створенні процедурного контенту, і зокрема в генерації карт, є однією з найбільш захоплюючих областей дослідження в галузі розробки ігор та симуляційного моделювання.

Процедурна генерація контенту (PCG) використовується для створення великої кількості контенту за допомогою меншої кількості вхідних даних, зазвичай з використанням алгоритмічних методів. У контексті генерації карт, це може означати створення унікальних ландшафтів, рівнів чи інших типів ігрових арен, які є цікавими, збалансованими та придатними для гри без необхідності ручного проектування.

Використання генетичних алгоритмів у цьому процесі приносить кілька переваг. Зокрема, це дозволяє розробникам створювати карти, які не тільки відрізняються візуально, але й мають різні ігрові характеристики. Генетичні алгоритми можуть оптимізувати карту для певного балансу між ворогами та ресурсами, певного рівня складності, або навіть для адаптації до стилю гри конкретного гравця.

Основні компоненти генетичного алгоритму включають популяцію кандидатів (у випадку генерації карт, це будуть різні варіанти дизайну карти), функцію пристосування, яка оцінює кожного кандидата, оператори селекції для вибору кандидатів для розмноження, а також генетичні оператори, такі як хрестовина та мутація, для генерації нащадків з поточного покоління кандидатів.

Розглядаючи цю тему, ми будемо досліджувати, як генетичні алгоритми можуть бути застосовані для автоматизації процесу створення карт, забезпечуючи різноманітність та інноваційність у ігровому дизайні.

Для генерації карти я використовував деякі налаштовувані параметри (рис 3.8).

Рис. 3.8 Параметри для генерації карти

Ініціалізація популяції:

Створення початкової популяції: Вибирається первісний набір випадково сгенерованих карт, кожна з яких представляє можливе рішення задачі (у вашому випадку — карту для гри). Розмір популяції залежить від задачі та може варіюватись залежно від обмежень обчислювальної потужності та інших чинників.

Оцінка пристосованості:

Функція **CalculateFitness** визначає "пристосованість" кожної карти на основі різних факторів, які сприяють якості карти для гри (рис. 3.9). Ось детальний розбір того, що робить кожен рядок коду:

int numberOfObstacles = mapData.obstacleArray.Where(isObstacle => isObstacle).Count();:

Цей рядок підраховує кількість перешкод на карті. **mapData.obstacleArray** – масив або список, який містить булеві значення (**true** або **false**), які вказують, чи є певний елемент перешкодою (**true** означає наявність перешкоди).

Int score = mapData.path.Count * _fitnessPathWeight + (int)(numberOfObstacles * _fitnessObsacleWeight);

Тут обчислюється частина оцінки пристосованості, що враховує два фактори: довжину шляху (**mapData.path.Count**) і кількість перешкод (**numberOfObstacles**). Кожен фактор множиться на відповідну вагу (**_fitnessPathWeight**, **_fitnessObsacleWeight**), яка визначає значення фактору у загальній оцінці.

Int cornersCount = mapData.cornersList.Count

Розраховується кількість поворотів на карті, збережених у **mapData.cornersList**.

Ця частина коду оцінює кількість поворотів на карті (**cornersCount**) і коригує загальну оцінку (**score**) на основі заданих порогів (**_fitnessCornerMin**, **_fitnessCornerMax**):

Якщо кількість поворотів знаходиться в межах заданого діапазону, до загальної оцінки додається бал за кожен поворот, помножений на вагу повороту (**_fitnessCornerWeight**).

Якщо кількість поворотів перевищує максимально допустиму (**_fitnessCornerMax**), з оцінки віднімаються бали за кожен зайвий поворот.

Якщо кількість поворотів менше мінімальної (**_fitnessCornerMin**), з оцінки віднімаються бали за відсутність необхідної кількості поворотів.

Score -= mapData.cornersNearEachOther * _fitnessNearCornerWeight:

Загальна оцінка зменшується, якщо на карті є повороти, розташовані близько один до одного. За кожну пару таких поворотів з оцінки віднімається певна кількість балів, помножених на вагу близькості поворотів (**_fitnessNearCornerWeight**).

Return score

```
private int CalculateFitness(MapData mapData)
{
    int numberOfObstacles = mapData.obstacleArray.Where(isObstacle => isObstacle).Count();
    int score = mapData.path.Count * _fitnessPathWeight + (int)(numberOfObstacles * _fitnessObsacleWeight);
    int cornersCount = mapData.cornersList.Count;
    if(cornersCount >= _fitnessCornerMin && cornersCount <= _fitnessCornerMax)
    {
        score += cornersCount * _fitnessCornerWeight;
    }
    else if(cornersCount > _fitnessCornerMax)
    {
        score -= _fitnessCornerWeight * (cornersCount - _fitnessCornerMax);
    }
    else if(cornersCount < _fitnessCornerMin)
    {
        score -= _fitnessCornerWeight * _fitnessCornerMin;
    }
    score -= mapData.cornersNearEachOther * _fitnessNearCornerWeight;

    return score;
}
```

Рис. 3.9 Обрахування пристосованості карти

Фінальна оцінка пристосованості повертається як результат роботи функції.

Сумуючи, функція **CalculateFitness**s стимулює пошук карт з оптимальною довжиною шляху, відповідною кількістю перешкод і поворотів, а також карає за надто близьке розташування поворотів, щоб створити більш різноманітні та цікаві карти для гри.

Розрахунок пристосованості: Кожна карта оцінюється на основі певної функції пристосованості (fitness function), яка вимірює, наскільки добре карта відповідає бажаним критеріям. Це може бути, наприклад, мінімізація часу, необхідного для проходження карти, або оптимізація розміщення перешкод.

Селекція:

Вибір особин для розмноження: Особини з високою пристосованістю вибираються для участі у створенні наступного покоління. Існує кілька методів селекції, включаючи турнірний вибір, рулетку та інші.

Кросінговер (схрещування): Дві обрані особини комбінуються для створення нових «дітей». Це може бути виконано шляхом обміну ділянками їх «хромосом», в моєму випадку – частинами карти (рис. 3.10).

```
private void CrossOverParrents(CandidateMap parent1, CandidateMap parent2, out CandidateMap child1, out CandidateMap child2)
{
    child1 = parent1.DeepClone();
    child2 = parent2.DeepClone();

    if(Random.value < _crossoverRatePercent)
    {
        int numBits = parent1.ObstaclesArray.Length;

        int crossOverIndex = Random.Range(0, numBits);

        for(int i = crossOverIndex; i < numBits; i++)
        {
            child1.PlaceObstacle(i, parent2.IsObstacleAt(i));
            child2.PlaceObstacle(i, parent1.IsObstacleAt(i));
        }
    }
}
```

Рис. 3.10 Процес схрещування

Мутація: Новостворені карти піддаються випадковим змінам, щоб збільшити генетичне різноманіття. Мутація може включати зміни, такі як додавання, видалення або переміщення перешкод на карті.

Нове покоління:

Формування нового покоління: Нове покоління формується з «дітей», що вийшли з процесу кросінговеру та мутації, а також, можливо, з деяких особин з попереднього покоління (елітизм).

Повторення процесу:

Повторення процесу: Кроки від 2 до 6 повторюються протягом багатьох поколінь. З кожним новим поколінням популяція повинна еволюціонувати до кращих рішень проблеми.

Завершення алгоритму: Процес триває до досягнення певного критерію зупинки, наприклад, до тих пір, поки не буде знайдено достатньо задовільне рішення, не буде досягнуто певної кількості поколінь або не пройде визначений час обчислень.

Кожен з цих кроків виконується в коді за допомогою різних класів і функцій, кожен з яких відповідає за свою частину процесу еволюції.

РОЗДІЛ 4 АНАЛІЗ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

4.1. Ігрове тестування генетичного алгоритму

Ігрове тестування генетичного алгоритму є критичним етапом в процесі розробки, оскільки воно дозволяє зрозуміти і оцінити ефективність та адекватність адаптивних механізмів штучного інтелекту, втілених у ворожій популяції. Під час цього етапу інтенсивно тестувалися кожна ітерація ігрових ворогів, використовуючи серії спеціальних ігрових сесій, де кінцева мета — знищення всієї групи, що складається з шести ворогів, які були сгенеровані алгоритмом. Це тестування допомогло ідентифікувати як сильні, так і слабкі сторони генетичної регуляції поведінки ворогів, а також визначити, як швидко вони адаптуються до стратегій гравців і чи забезпечує система відповідний рівень складності та різноманіття ігрового процесу.

4.2 Процес підбору оптимальних значень

Одноточкове схрещування + турнірний відбір:

Схема показує помірне коливання у всіх характеристиках з незначним трендом на зростання. Турнірний відбір, який полягає в виборі найкращих індивідів для репродукції через "змагання", здається, підтримує конкуренцію між різними генетичними варіантами, проте одноточкове схрещування може призвести до меншої різноманітності в нових генах (рис. 4.1).

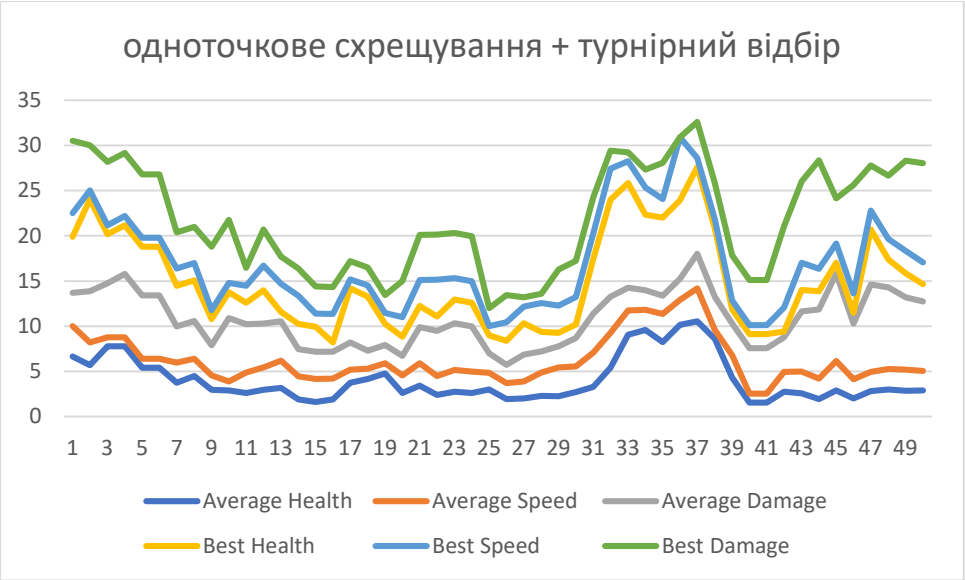

Рис. 4.1 Результати поєднання одноточкового схрещування та турнірного вибору

Тут ми бачимо більш виражену волатильність у всіх трьох характеристиках. Рулетковий відбір, який вибирає індивідів залежно від їх "пристосованості" (пропорційно до їх фітнесу) і це може призвести до того, що, індивіди з незначно кращим фітнесом можуть отримати непропорційно велику кількість потомства. Це може створити "сплески" у показниках пристосованості, як це видно на графіку (рис. 4.2).

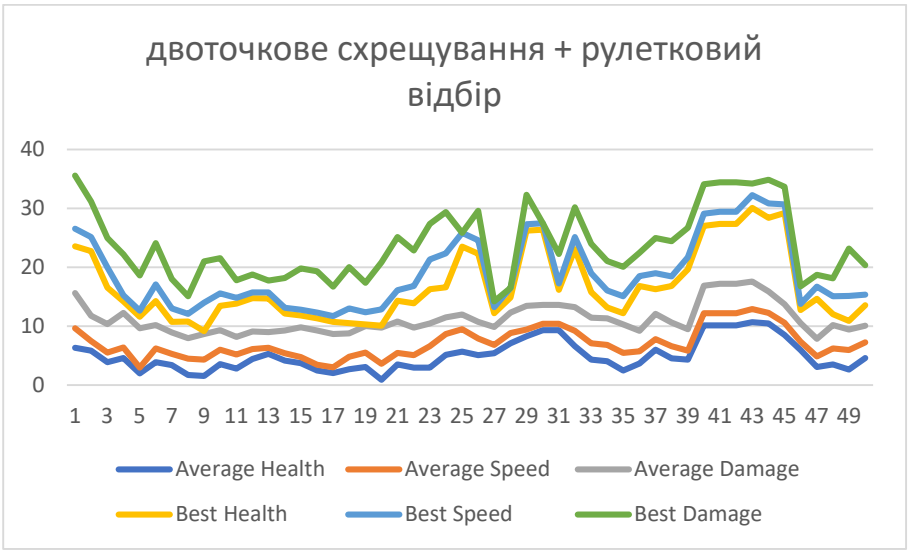

Рис. 4.2 Результати поєднання двоточкового схрещування та рулеткового відбору

Відсутність механізму, подібного до елітарного відбору, означає, що найсильніші індивіди можуть бути випадково втрачені з покоління в покоління, що знижує загальний рівень пристосованості популяції.

Двоточкове схрещування збільшує варіабельність у потомстві, що може бути корисно для експлорації, але також може руйнувати вже існуючі адаптації, створюючи потомство з нижчим фітнесом.

Мною була вибрана зв’язка асортативного схрещування та елітарного відбору (рис. 4.3), тому що:

1. ***Баланс різноманітності і оптимізації:*** Асортативне схрещування, яке віддає перевагу паруванню особин зі схожими характеристиками, сприяє підтриманню та посиленню корисних рис, які вже існують у популяції. Це зменшує ризик втрати досягнень внаслідок схрещування з менш пристосованими особинами.
2. ***Консервація кращих рішень:*** Елітарний відбір гарантує, що найкращі індивіди популяції будуть збережені для наступного покоління, забезпечуючи, що досягнення не будуть втрачені і можуть бути ще поліпшені.
3. ***Стабільність протягом еволюції:*** Графік показує відносну стабільність у показниках "Average Health", "Average Speed" та "Average Damage", що свідчить про здатність алгоритму підтримувати здобутки без великих коливань, що є важливим для багатьох застосувань, де різкі зміни між поколіннями можуть бути небажані.
4. ***Адаптація до конкретної задачі:*** Якщо ваша задача вимагає пошуку стабільних і надійних рішень, а не постійного пошуку нових нерозвіданих областей простору рішень, то методи, що використовуються на цьому графіку, можуть бути найбі

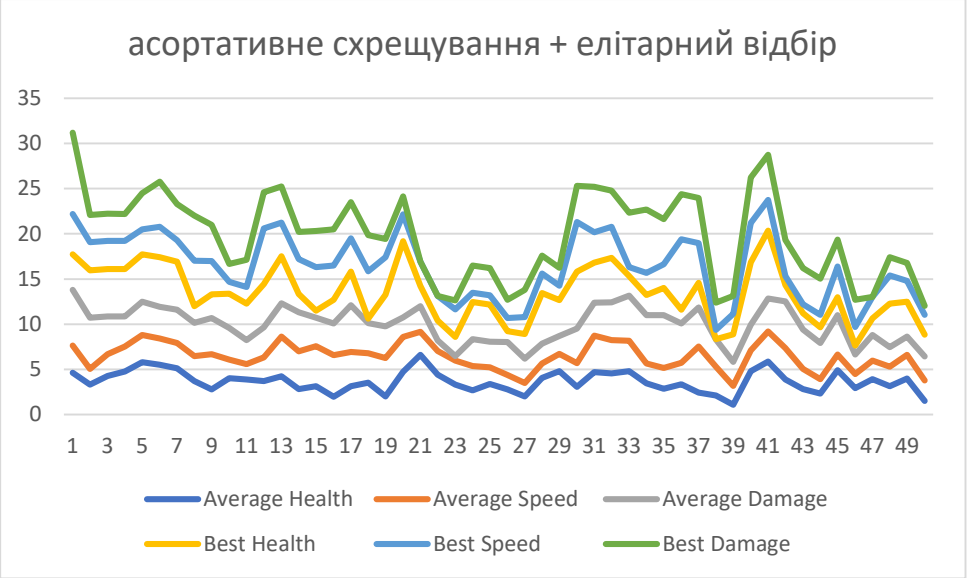

Рис. 4.3 Результати поєднання асортативного схрещування та елітарного відбору

4.3 Результати дослідження

Пристосування гравця: Графіки показують підвищення пристосування гравця протягом часу з деякими коливаннями. Це може бути ознакою того, що алгоритм періодично пробує нові стратегії, які іноді виявляються менш успішними, але є необхідними для експлорації простору рішень. Збільшення пристосування з покоління на покоління свідчить про те, що гравець в цілому стає більш пристосованим до середовища, можливо, стаючи краще в уникненні шкоди або вибираючи більш ефективні стратегії бою.

Ці динаміки і коливання є характерними для процесу еволюції, де алгоритм постійно пробує різні комбінації генів (характеристик) для виявлення найсильніших індивідів. З кожним поколінням, кращі риси зберігаються і посилюються, тоді як менш пристосовані відкидаються. Цей процес не завжди лінійний і може включати тимчасові зворотні рухи, оскільки система досліджує різноманітність можливих стратегій і адаптацій (рис. 4.4).

На другому графіку показано два параметри: "Best Fitness" (найкраща фітнес-характеристика) та "Average Fitness" (середня фітнес-характеристика) для ворогів (рис. 4.5).

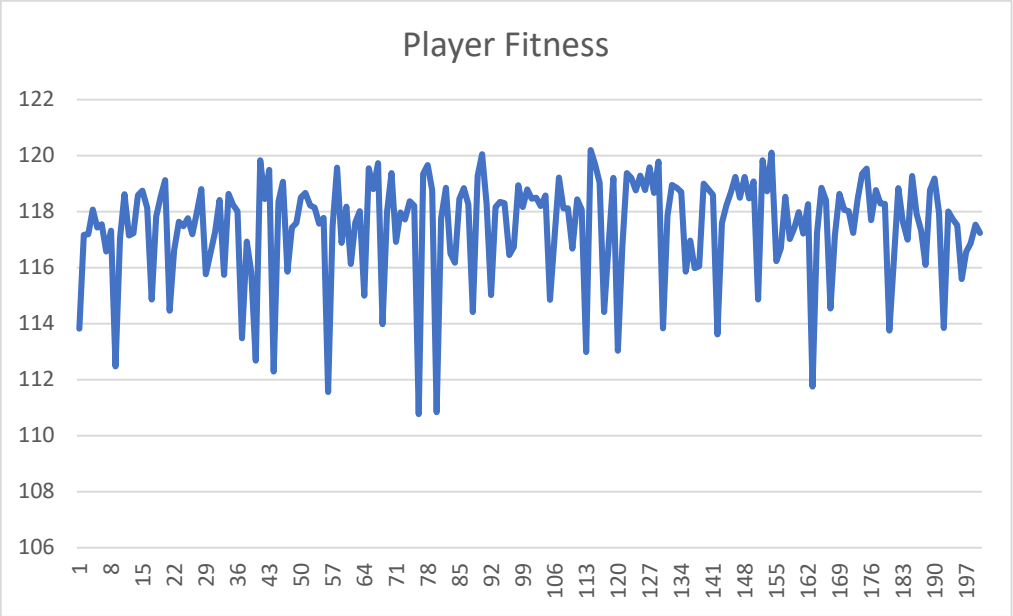

Рис. 4.4 Пристосування гравця

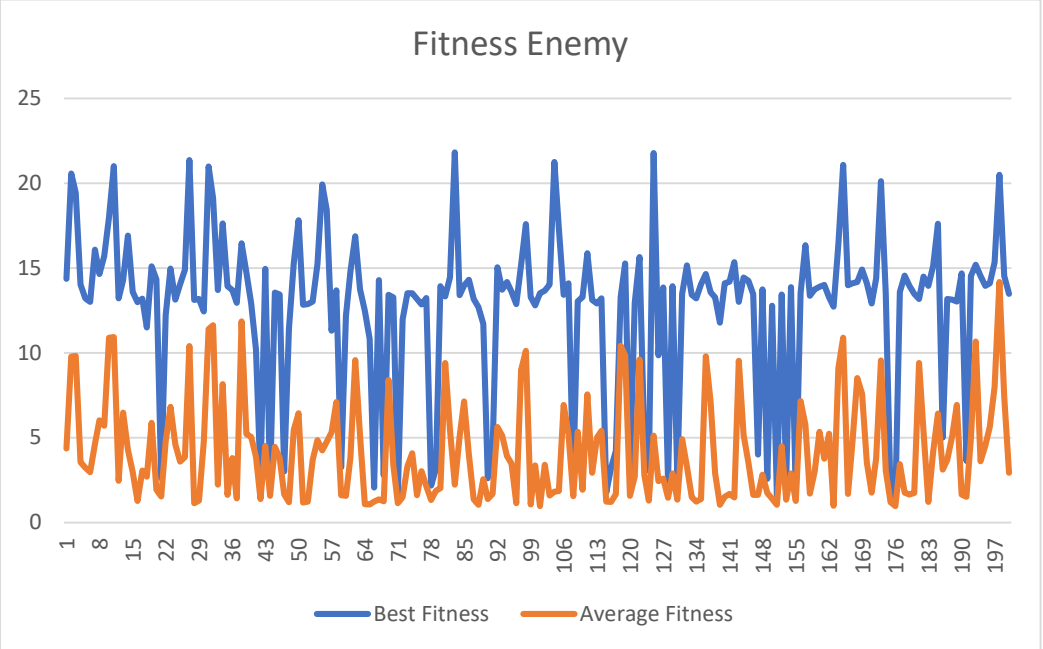

Рис. 4.5 Пристосування ворогів

Best vs. Average: Можна помітити, що лінія найкращої фітнес-характеристики (синя) зазвичай вища за середню (оранжева), що очікувано, оскільки це показує, що хоча б деякі одиниці досягають кращих характеристик, ніж середня по популяції.

Коливання: Середня фітнес-характеристика має тенденцію до збільшення та зменшення, але без чіткої тенденції до постійного зростання, що може означати, що популяція ворогів змінюється, але не обов'язково еволюціонує до "кращого" стану.

Параметри ворога (рис. 4.6):

- Здоров'я (Health)

Середнє здоров'я та найкраще здоров'я: Стабільне зростання середнього і найкращого здоров'я вказує на те, що алгоритм віддає перевагу індивідам з вищим рівнем здоров'я. Це логічно для середовища, де виживання є ключовим показником успіху. Вороги з більшим здоров'ям можуть витримувати більше атак, тому такі характеристики продовжують розвиватися.

- Швидкість (Speed)

Середня швидкість та найкраща швидкість: Початкове зростання може вказувати на те, що швидкість є важливим атрибутом для виживання. Однак стабілізація свідчить про досягнення плато, коли додаткове збільшення швидкості вже не приносить суттєвого виграшу в приспособленості. Можливо, що існує компроміс між швидкістю та іншими атрибутами (наприклад, більша швидкість може погіршувати точність).

- Шкода (Damage)

Середня шкода та найкраща шкода: Значення шкоди, яку завдають вороги, варіюється більше, ніж інші атрибути, і не показує чіткого тренду зростання. Це може бути ознакою того, що в контексті цієї симуляції, здатність наносити шкоду має менший вплив на виживання, ніж здоров'я та швидкість, або що існує певний баланс між шкодою та іншими характеристиками, який алгоритм намагається знайти.

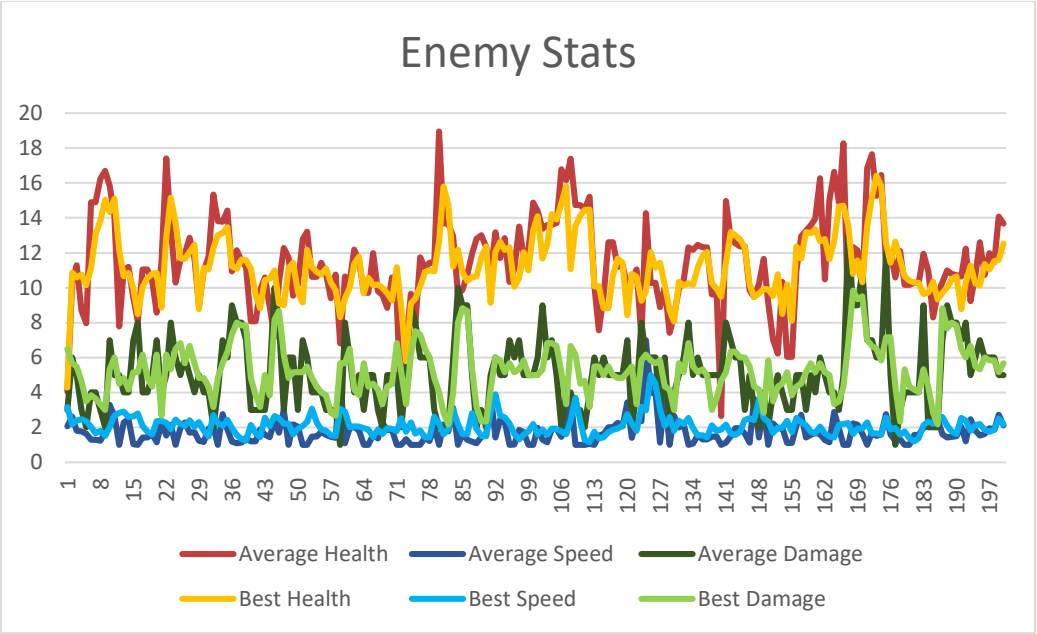

Рис 4.6 Параметри ворога

У генетичних алгоритмах індивіди пристосовуються до середовища, в якому вони конкурують за ресурси або, у цьому випадку, за виживання і домінування. Якщо ми розглянемо взаємодію між гравцем і ворогами як взаємозв'язок хижака та здобичі, то можемо побачити, що успіх однієї групи може впливати на успіх іншої групи.

Коли фітнес гравця падає, це може свідчити про те, що вороги еволюціонували, щоб краще протистояти гравцеві. Це може бути у вигляді підвищення здоров'я, швидкості, або здатності наносити шкоду. Як результат, вороги стають ефективнішими у своїй взаємодії з гравцем, що призводить до зниження фітнесу гравця.

Навпаки, коли фітнес гравця зростає, це може означати, що гравець набув стратегій або здібностей, які дозволяють йому ефективніше справлятися з ворогами. Це може викликати зниження показників здоров'я, швидкості, та шкоди серед ворогів, оскільки менш пристосовані вороги елімінуються.

Ця динаміка подібна до того, що називають еволюційними гонками озброєнь, де два види або дві групи постійно адаптуються один до одного з метою підвищення власних шансів на успіх. Це може створити циклічний патерн, де покращення в однієї сторони стимулюють адаптації в іншої.

ВИСНОВКИ

У ширшому контексті розробки ігор інтеграція генетичних алгоритмів (ГА) у системи штучного інтелекту (ШІ) є інноваційним підходом до покращення ігрового досвіду завдяки адаптивним викликам і середовищам. Це дослідження мало на меті вивчити практичні наслідки вбудовування ГА у двох ключових аспектах: процедурна генерація ігрових карт та динамічне балансування атрибутів ворога.

Процедурна генерація карт за допомогою ГА виявилася потужним інструментом для створення різноманітних ігрових ландшафтів, кожен з яких має свої унікальні завдання та характеристики. Ця методологія підтримує створення складних дизайнів, які можуть адаптуватися до взаємодії гравців, потенційно збільшуючи цінність гри для повторного проходження та підтримуючи інтерес гравців з часом. Однак складна природа цих алгоритмів також означає, що розробники повинні ретельно враховувати необхідні обчислювальні ресурси і потенціал для створення карт, які не завжди можуть відповідати запланованому ігровому досвіду.

Використання ГА для балансування противника додає динамічний компонент до складності гри, дозволяючи грі адаптуватися до рівня навичок гравця в реальному часі. Така адаптивна складність може покращити ігрову взаємодію, надаючи індивідуальний досвід для різних типів гравців, гарантуючи, що гра залишається складною, але не нездоланною. Однак непередбачувана природа ГА може також призвести до нерівномірності кривих складності, і розробники повинні впроваджувати запобіжні заходи, щоб запобігти розчаруванню або демотивації серед гравців.

Вивчаючи накладні витрати, пов'язані з впровадженням ГА, дуже важливо зважити переваги та витрати на розробку. Хоча ГА пропонують складні засоби розробки ігор, час і ресурси, необхідні для впровадження та налаштування цих алгоритмів, повинні бути виправдані значним покращенням ігрового досвіду.

Крім того, інтеграція ГА з низкою стратегій ШІ може призвести до створення більш досконалої системи ШІ противника, здатної розвиватися і

адаптуватися до стратегій гравця. Такий підхід може призвести до створення ШІ, який не лише кидає виклик гравцеві, але й еволюціонує разом з ним, сприяючи більш захопливому та індивідуальному ігровому досвіду.

Отже, ГА відкривають нові можливості в розробці ігор, пропонуючи потенціал для створення більш захопливих, складних і персоналізованих ігор. Однак їхнє впровадження не позбавлене викликів і вимагає ретельного балансу між інноваціями та практичністю. Оскільки ігрова індустрія продовжує розвиватися, роль ГА в розробці ШІ, ймовірно, зростатиме, пропонуючи нові способи захопити гравців і кинути їм виклик. Розробники ігор повинні використовувати цю технологію ефективно та раціонально, гарантуючи, що переваги ігрового досвіду переважать складнощі впровадження.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Holland, J. H. (1992). "Adaptation in Natural and Artificial Systems". University of Michigan Press.

2. Goldberg, D. E. (1989). "Genetic Algorithms in Search, Optimization, and Machine Learning". Addison-Wesley.

3. Mitchell, M. (1998). "An Introduction to Genetic Algorithms". MIT Press.

4. Eiben, A. E., & Smith, J. E. (2015). "Introduction to Evolutionary Computing". Springer.

5. Fogel, D. B. (2006). "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence". IEEE Press.

6. Sivanandam, S. N., & Deepa, S. N. (2007). "Introduction to Genetic Algorithms". Springer.

7. Yannakakis, G. N., & Togelius, J. (2018). "Artificial Intelligence and Games". Springer.

8. Buckland, M. (2004). "Programming Game AI by Example". Wordware Publishing, Inc.

9. Millington, I., & Funge, J. (2009). "Artificial Intelligence for Games". CRC Press.

10. Rabin, S. (Ed.). (2013). "Game AI Pro: Collected Wisdom of Game AI Professionals". CRC Press.

11. Sweetser, P. (Ed.). (2008). "Emergence in Games". Charles River Media.