ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Метод адаптації складності штучного інтелекту ворогів у грі жанру "Виживання 2.5D"»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми Інженерія програмного забезпечення

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

_____ Дмитро ВДОВІН
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Дмитро ВДОВІН

Керівник: Олесь ДІБРІВНИЙ
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« » _____2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Вдовіну Дмитру Едуардовичу

1. Тема кваліфікаційної роботи: "Метод адаптації складності штучного інтелекту ворогів у грі жанру "Виживання 2.5D""

керівник кваліфікаційної роботи Олесь ДІБРІВНИЙ, доктор філософії (PhD), затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання студентом роботи «17» грудня 2024 р.

3. Вхідні дані до роботи: науково-технічна література, алгоритм поведінки штучного інтелекту

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).

1. Дослідження та аналіз ігор за темою.
2. Дослідження методів адаптації складності штучного інтелекту.
3. Реалізація адаптації складності штучного інтелекту.
4. Опис проектування системи.

5.Перелік ілюстративного матеріалу: *презентація*

1. Мета, об’єкта та предмет дослідження.
2. Актуальність роботи.
3. Аналіз методів адаптації штучного інтелекту.
4. Представлення формули адаптації складності штучного інтелекту ворога.
5. Алгоритм адаптації складності.
6. Екранна форма Harvius.
7. Висновки.
8. Порівняльний аналіз адаптації складності.

6. Дата видачі Завдання « 16 » жовтня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вивчення теми магістерської роботи	16.10-23.10.2024	
2	Аналіз поточних методів адаптації складності штучного інтелекту в ігровій індустрії	24.10-29.10.2024	
3	Дослідження механік і алгоритмів AI для жанру "Виживання 2.5D"	30.10-04.11.2024	
4	Вивчення принципів балансу складності ігрового процесу в 2.5D-іграх	05.11-10.11.2024	
5	Огляд сучасних методів машинного навчання, застосовних для динамічного налаштування AI	11.11-19.11.2024	
6	Розробка алгоритму адаптації складності поведінки ворогів залежно від рівня гравця	20.11-21.11.2024	
7	Оформлення роботи: вступ, висновки, реферат	22.11-24.11.2024	
8	Розробка демонстраційних матеріалів	25.11-27.11.2024	
9	Попередній захист роботи	28.11.2024	

Здобувач вищої освіти _____ Дмитро ВДОВІН

(підпис)

Керівник
кваліфікаційної роботи _____ Олесь ДІБРІВНИЙ

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи 88 с., 0 табл., 24 рис., 30 джерел.

Мета дослідження – підвищення якості ігрового процесу штучного інтелекту за рахунок використання адаптивної складності.

Об'єкт дослідження – процес адаптивної складності штучного інтелекту.

Предмет дослідження – алгоритми штучного інтелекту для адаптації складності в грі жанру "Виживання 2.5D"

Короткий зміст роботи: У цій роботі досліджується адаптацію штучного інтелекту в мобільних іграх жанру "Виживання 2.5D". Проведено аналіз існуючих методів адаптації складності при створенні складності рівня гри. Розроблено метод, що дозволяє покращити рівень складності у іграх "Виживання 2.5D". Проведено порівняльний аналіз методів адаптації складності. Результати цього дослідження дали змогу підвищити якість ігрового процесу.

КЛЮЧОВІ СЛОВА: "Виживання 2.5D"

ABSTRACT

Text part of the qualification work 88 p., 0 tables, 24 figures, 30 references.

The purpose of the study is to improve the quality of the game process of artificial intelligence through the use of adaptive complexity.

Object of research – is the process of adaptive complexity of artificial intelligence.

Subject of research – is artificial intelligence algorithms for adapting complexity in the game genre "Survival 2.5D"

Summary of the work: This paper investigates the adaptation of artificial intelligence in mobile games of the "Survival 2.5D" genre. An analysis of existing methods of complexity adaptation when creating game level difficulty is carried out. A method has been developed to improve the level of complexity in "Survival 2.5D" games. A comparative analysis of the methods of complexity adaptation was carried out. The results of this study allowed to improve the quality of the game process.

KEYWORDS: "Survival 2.5D"

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NPC - неігровий персонаж(Non-Playable Character).

PVP - гравець проти гравця(Player vs Player).

HUD - індикаторна панель (Heads-Up Display).

RPG - рольова гра(Role-Playing Game).

PvE - гравець проти оточення(Player vs Environment).

XP - очки досвіду(Experience Points).

DAG - орієнтований ациклічний граф (Directed Acyclic Graph).

CFR - мінімізація контрфактичного жалю(Counterfactal Regret Minimization).

BNE – Контрфактична мінімізація жалю (Counterfactal Regret Minimization).

ВСТУП

Ігрова індустрія стрімко розвивається і стає однією з найвпливовіших індустрій розваг у світі. Завдяки інноваційним технологіям, таким як штучний інтелект і вдосконалена графіка, гра стала більш захоплюючою і реалістичною. Ігрові платформи, від консолей до мобільних пристроїв, пропонують широкий спектр жанрів і стилів, залучаючи мільйони геймерів різного віку та інтересів.

Жанр виживання зосереджується на викликах гравця в екстремальних ситуаціях, коли йому потрібно виживати, та захищатися від небезпеки. Ці ігри часто потрапляють у постапокаліптичне, чи інше вороже середовище, де виживання залежить від їхніх навичок і стратегій. Завдяки реалістичному відчуттю та напруженій атмосфері цей тип гри справляє гарне враження.

Завдяки таким нововведенням жанр виживання продовжує привертати увагу гравців по всьому світу, надаючи їм справжній виклик і можливість випробувати свої навички в екстремальних умовах.

Завдання роботи полягає в таких пунктах:

1. Дослідження та аналіз адаптації складності штучного інтелекту.
2. Реалізація адаптації складності штучного інтелекту.
3. Опис проектування системи.
4. Порівняння отриманих результатів з вже існуючими методами.

Мета дослідження – підвищення якості ігрового процесу штучного інтелекту за рахунок використання адаптивної складності.

Об’єкт дослідження – процес адаптивної складності штучного інтелекту.

Предмет дослідження – алгоритми штучного інтелекту для адаптації складності в грі жанру "Виживання 2.5D"

Практичне значення дослідження полягає в отриманні результатів, які допомагають покращити алгоритми штучного інтелекту для адаптації складності

ворогів в іграх на виживання. У рамках цього дослідження був розроблений новий метод налаштування рівня складності спеціально для ігор жанру "Виживання 2.5D". Застосування даного методу дозволяє створювати ігрові враження, де рівень виклику автоматично підвищується відповідно до прогресу гравця.

Наукова новизна полягає у створенні адаптивної складності штучного інтелекту. Дане дослідження вдосконалює за допомогою існуючих методів складність супротивників. Крім того, інтенсивність адаптації сприяє кращому утриманню гравців, коли вони прогресують і розвивають свої навички. Даний алгоритм не тільки збільшує тривалість ігрових сесій, але й сприяє утриманню гравця, таким чином збільшуючи популярність і успіх на ринку. Розроблений алгоритм не тільки робить гру приємнішою, але й покращує загальну якість гри, надаючи гравцям різноманітний досвід.

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ

1.1. Поняття жанру та особливості складності гри "Виживання 2.5D"

Ігри про виживання виділяються в ігровій індустрії своєю унікальністю, захоплюючи гравців захоплюючим середовищем, складними ігровими механіками та гострими відчуттями від подолання викликів у суворих віртуальних світах. Незалежно від того, чи плаваєте в небезпечних морях, приборкуєте доісторичних тварин або просто намагаєтеся пережити голод і негоду, ігри на виживання пропонують безліч вражень, щоб зацікавити гравців. Заглибимося в історію ігор на виживання, дізнаємося, чому гравці їх так люблять, і проаналізуємо деякі з найцікавіших і найпопулярніших ігор на виживання.

Основна відмінність між іграми на виживання та іншими жанрами полягає в тому, що вони роблять акцент на розвідці. Гравці потрапляють у захоплююче середовище, де вони повинні збирати ресурси, будувати притулки, створювати інструменти та зброю, а також відбивати загрози від динозаврів, голоду, морських істот і навіть інших гравців! Відчуття досягнення, яке виникає після того, як ви почали майже ні з чого, вижили й поступово створили успішну базу чи персонажа, є важливим мотиватором для багатьох гравців.

Однак, мабуть, головним завданням у цих іграх є виживання. Гравці стикаються з різноманітними загрозами від стихійних лих до ворожих тварин. Щоб залишитися в живих, гравець повинен розвивати свої навички, планувати свої дії та знаходити рішення в критичних ситуаціях.

Завдяки цим компонентам ігри на виживання пропонують гравцям неймовірно захоплюючий досвід, який залишає незабутнє враження та постійно приваблює їх до нових викликів і пригод.

Ігри на виживання перевіряють не лише фізичні навички гравців, а й їх розумову витривалість і здатність приймати рішення в екстремальних ситуаціях.

Вони вчать стратегічного мислення, співпраці з іншими гравцями, а також терпіння та вміння адаптуватися до несподіваних ситуацій.

У світі ігор на виживання гравець стає героєм власної історії, де кожен переможений або подоланий ворог стає кроком до великої перемоги - виживання в найскладніших ситуаціях. Ця постійна боротьба за життя створює неперевершений ігровий процес і незабутні емоції, які гравець отримує від кожної години, проведеної в цьому захоплюючому світі.

Ці ігри також дозволяють гравцям відчути себе частиною чогось більшого, ніж гра. Вони створюють унікальні віртуальні світи, де кожен гравець може відчути себе дослідником, який завойовує нові території, архітектором, який будує власні бази, і спостерігачем природи, який адаптує свої дії до умов виживання.

Ігри на виживання також заохочують креативність гравців, оскільки вони часто пропонують велику свободу у виборі стратегій і методів виживання. Гравці можуть спробувати різні підходи до вирішення проблем, розвиваючи свої творчі здібності та аналітичні навички.

Крім того, ці ігри можуть сприяти формуванню спільнот і взаємодії між гравцями. Багато з них мають багатокористувацькі режими, де гравці можуть співпрацювати або змагатися один з одним, створюючи можливості для взаємодопомоги, стратегічного планування та соціальної взаємодії.

Загалом, ігри на виживання відкривають перед гравцями безмежний світ пригод, де кожен крок приносить нові випробування та досвід, а кожен успіх відчувається як справжня перемога.

Крім того, ігри на виживання можуть допомогти розвинути управління ресурсами та стратегічне планування. Гравці повинні ретельно досліджувати оточення, розумно використовувати доступні ресурси та ефективно планувати свої дії, щоб забезпечити своє виживання.

Крім того, багато ігор про виживання пропонують інтерактивні уроки з екології та природної історії. Гравці повинні розуміти взаємозв'язок між різними середовищами, їх характеристики та вплив на живий світ, щоб успішно адаптуватися до них.

Загалом ігри на виживання дозволяють гравцям не тільки розважатися, а й вчитися новому, розвивати навички та навіть думати про важливі речі сучасного світу, такі як екологія та стратегічне планування.

Крім того, ігри на виживання можуть допомогти розвинути самодисципліну та вміння працювати з обмеженими ресурсами. Гравці навчаються витримувати стресові ситуації, дотримуватися суворих правил і вміло розпоряджатися своїм часом і енергією.

Іншою важливою частиною ігор на виживання є їхня здатність встановлювати емоційний зв’язок з гравцями. Від страху перед небезпекою до радості успішного виживання, ці ігри можуть сприяти розвитку емоційного інтелекту та емпатії, допомагаючи гравцям краще розуміти власні та чужі емоції.

Тому ігри на виживання не тільки розважають, але й навчають, розвивають і заохочують гравців досліджувати нові аспекти життя та розвивати навички, які можуть бути корисними в реальному світі. Для того, щоб краще проаналізувати даний жанр, проаналізуємо декілька ігор.

1.2 Аналіз існуючих ігор

1) **Don't Starve Together** рис. 1.1, гра на виживання на основі унікальних Minecraft або Terraria. Гравці знову опиняються в багаторівневій пустині, де вони повинні навчитися виживати. Гравці повинні підтримувати своє здоров’я, голод і розсудливість, беручи участь у протиборчих битвах, починаючи від битв із босами, і закінчуючи самою темрявою.

Гра ведеться відповідно до циклу успіху економічного архіву Маслоу. Гравці повинні постійно турбуватися про час гри, щоб вони могли підготуватися за межами бази, а потім експериментувати з різними науковими та магічними діями, які максимізують безпеку. Це напружений досвід, оскільки гравці проводять кожен день у пошуках м’яса та інших ресурсів, щоб зібрати їх на сувору зиму та інші суворі пори року, як-от літо, коли їм холодно. Together має більше вмісту загалом, а можливість ділитися завданнями з чотирма гравцями робить базові завдання

виживання цікавішими. Один гравець може полювати на кроликів, інший може будувати вулик, і обидва інгредієнти можна поєднувати з гарбузом, створеним іншою людиною, щоб створити ситну їжу. Спільні зусилля можуть зайняти три дні роботи однієї людини, тому вижити легко.

Однак дизайн Don't Starve Together компенсує відносну легкість виживання для кількох гравців. На відміну від серії Monster Hunter і подібних кооперативних ігор, де кілька гравців штучно збільшують складність, збільшуючи здоров'я ворогів, у Dont Starve Together не має значення, один користувач чи ні. Ця складність виникає через більш реалістичні виклики. Відсутність їжі та швидке отримання ресурсів. Само собою зрозуміло, що кожен персонаж у Don't Starve має різні характеристики та вроджені характеристики, які впливають на їх існування.

Вігфрід, наприклад, є сильною бойовою героїнею, яка спочатку використовує зброю та броню, щоб знищувати своїх супротивників, але може їсти лише м'ясо, що робить основне джерело їжі, як-от ягідні кущі, корисними перед виготовленням гарбузів. У той же час Венді — тендітна дівчина з хорошим контролем над розумом через її досвід зі своєю мертвою сестрою Ебігейл, яка, можна сказати, бореться за Венді. Гравці можуть входити та виходити, дозволяючи деяким персонажам виконувати завдання, які можуть бути складнішими для інших, не порушуючи повністю перебіг дії.

Іншим унікальним аспектом масштабування складності є гнучкість самої гри, яка базується на стилі та навичках гравця. Поки команда зосереджується на створенні безпечної бази, гра заохочує їх захищати територію від регулярних атак ворогів і стихійних лих. З іншого боку, у міру того, як гравці подорожують активніше, вони зіткнуться з новими викликами, від зустрічі з босами до виживання в більш небезпечних біомах. Таким чином гра адаптується до дій гравця та гарантує, що він грає так, як хоче, зберігаючи напругу.

Особливо цікаво те, як «Don't Starve Together» використовує психологічний аспект складності. Наприклад, голод, холод і розсудливість не просто впливають на стан персонажа: вони створюють атмосферу страху, яка змушує гравців діяти обережно. Темрява, яка стає смертоносною без джерела світла, нагадує гравцям про

постійну загрозу і спонукає бути завжди готовими. Навіть невеликі помилки, такі як забуте багаття, або відсутність запасів відновлення здоров'я, можуть перетворити звичайний день на катастрофу, змусивши гравців ретельно планувати свої дії.

Крім того, гра заохочує гравців розвивати свої навички, взаємодіючи зі світом. Наприклад, деякі ресурси стають доступними лише після вивчення певних технологій або роботи з новими предметами. Це відкриває нові горизонти для гравців, але водночас створює додатковий рівень виклику. Виготовлення нового наукового пристрою, або вивчення магічних заклинань може полегшити гру, але процес створення часто вимагає ризикованих дій, як-от збір рідкісних матеріалів у небезпечних місцях.

Важливим елементом адаптивної складності в грі є здатність гравців змінювати складність командними діями. Наприклад, якщо один гравець має слабкість до складної механіки, інші можуть взятися за важливі завдання, допомагаючи команді збалансувати завдання. Це унікальна перевага кооперативного режиму: гра залишається захоплюючою, але не перевантажує гравців, даючи їм можливість розіграти свої сильні сторони. У той же час гравці, які хочуть випробувати себе, можуть свідомо вибирати складніших персонажів або грати в складніших середовищах, створюючи додатковий рівень налаштування.

Не менш важливим є те, як гра заохочує творчий підхід гравця до вирішення проблем. Коли природних ресурсів стає дефіцитно, гравці повинні придумати нові способи виживання, наприклад створити ферми для стабільного постачання їжі або побудувати оборону, щоб відбити ворогів. Ці елементи вносять різноманітність у гру та спонукають гравців експериментувати, підвищуючи задоволення від гри.

Таким чином, Don't Starve Together показує, що налаштування складності можна інтегрувати в усі аспекти гри. Гра пропонує кожному гравцеві та команді унікальний досвід, дозволяючи їм насолоджуватися виживанням і приймати виклики, що постійно змінюються. Це не тільки додає можливості повторної гри, але й створює глибокий емоційний зв’язок із ігровим процесом, роблячи кожен новий день невідомою та захоплюючою пригодою у світі Don’t Starve.

Рівні складності в Don't Starve Together різноманітні та динамічні, що робить гру цікавою та складною. По-перше, постійні загрози і небезпеки у вигляді ворогів, стихійних лих та інших непередбачуваних подій вводять гравців у стан стресу, вимагаючи постійного стратегічного мислення та реакції на нові ситуації.

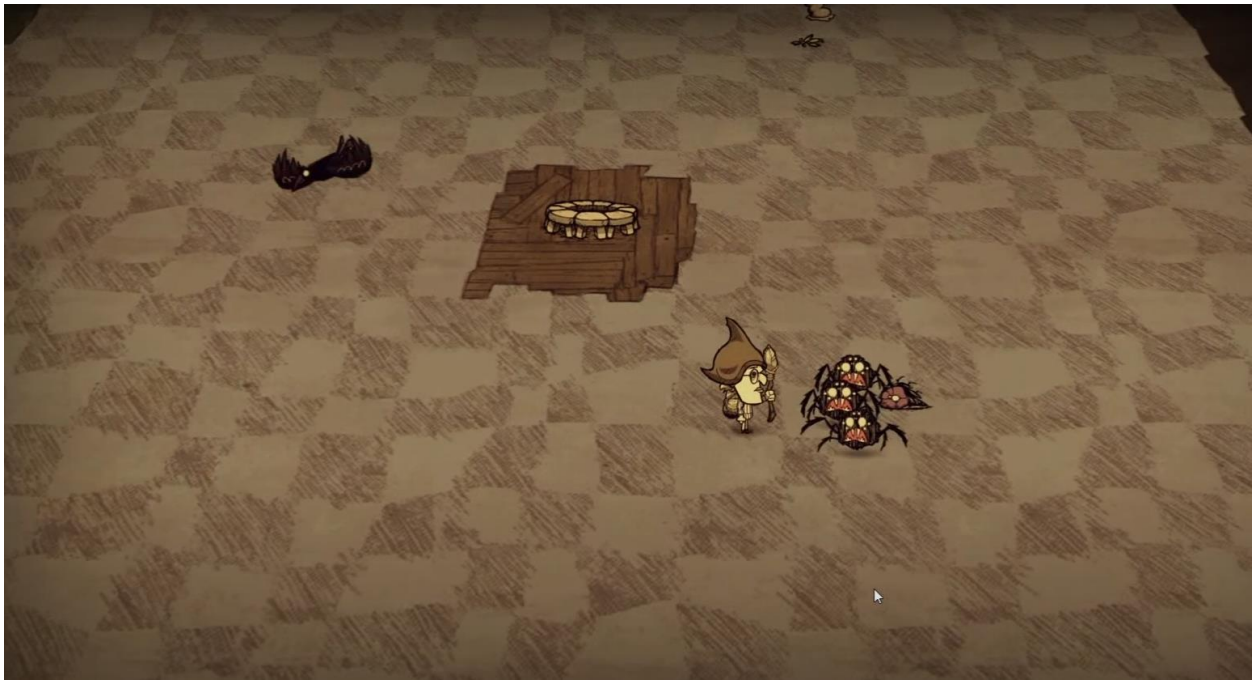


Рис. 1.1 Демонстрація геймплею в грі Don't Starve Together

2) Valheim рис. 1.2, гра на виживання в пісочниці, дія якої загально та феєрично відбувається у світі епохи вікінгів з елементами скандинавської міфології. Основна історія розповідає про те, як Всебатько Одін створив унікальне десяте королівство, де благородні воїни Мідгарду відроджуються та випробовуються, щоб збільшити свої сили та підготуватися до майбутнього ворожого повстання. Це королівство відоме як Вальгейм. Під час тренування гравець отримує вказівки від того, кого надіслав Одін, про те, що воїнів, які пройшли випробування Вальхейма та довели себе, можна відправити до Валгалли. Для цього гравці повинні досліджувати, будувати, збирати ресурси та битися.

У Valheim складність налаштована таким чином, щоб гравці природним чином могли знайомитися з ігровим світом, поступово збільшуючи складність. Початок гри відбувається у відносно безпечному середовищі, де гравець опановує

базову механіку збору ресурсів, будівництва та полювання на простих ворогів. Однак, коли гравець просувається через нові біоми, гра стає складнішою, змушуючи адаптуватися до нових умов, ворогів і вимог.

Кожен біом у грі унікальний і містить власні завдання. Наприклад, у темному лісі доведеться битися з гоблінами та скелетами, які сильніші за ворогів у Арках. У болотах отруйні атаки та небезпечні істоти, такі як Слайм і Драугри, стають додатковим випробуванням, а щоб вижити в горах, потрібно приготувати зілля або антифриз. Ця стратегія вимагає від гравця вивчення нових тактик і збору унікальних ресурсів для подолання цих викликів.

Регулювання складності у Valheim також можна побачити, змінюючи тактику суперників відповідно до обставин. Деякі з них діють поодинці, а інші атакують групами, додаючи динамізму боям. Наприклад, при зустрічі з тролем у лісі гравець повинен зосередитися на його повільних, але сильних атаках, а на болоті — стежити за швидкими атаками Драгева або отруйними хмарами зі Слайму. Це вимагає постійних змін у зброї, спорядженні та стратегії.

Монстри відіграють важливу роль у труднощах адаптації. Кожен з них унікальний і вимагає ретельної підготовки. Наприклад, перед поєдинком з скелетом потрібно зібрати спеціальні ресурси для виклику та захиститися від холодних атак. Кожен новий бос - це виклик, який вимагає від гравця не тільки вдосконалювати свої бойові навички, але й заздалегідь планувати свої дії, збирати необхідні ресурси та створювати нове спорядження.

Система крафта Valheim також підтримує рівень складності гри. У міру проходження нових біомів гравець відкриватиме нові матеріали для створення міцнішої зброї та броні. Однак використання цих матеріалів все ще пов’язане з ризиком, оскільки їх можна отримати лише з небезпечних місць, де вороги сильніші, ніж звикли. Це змушує продумувати кожен крок і планувати свої дослідження.

Динамічні події, такі як раптові атаки ворогів, додають грі непередбачуваності. Частота та серйозність цих подій зростає залежно від того, які

біоми гравець уже відвідав, а їх прогрес у грі тримає гравця в постійній напрузі. тому що навіть у місці, яке вважаєте безпечним, може виникнути новий бій.

Регулювання складності у Valheim не тільки робить гру цікавішою, але й дає гравцям можливість активно розвиватися. Окрім бойових навичок, гра вчить терпінню, стратегічному мисленню та ефективному використанню ресурсів. Ця стратегія робить кожен етап гри цікавим і змушує повертатися знову і знову.

Бойова система Valheim підвищує рівень складності. Гравці можуть наносити різного роду пошкодження - ріжучі та рубаючі, що вимагає вдумливого підходу до вибору зброї та тактики ведення бою. Блокування, механіка ухилення та контратаки відіграють важливу роль у виживанні та розгромі ворога.

Можна кинути виклик босам з унікальними здібностями та атаками в різних біомах. Кожен бій з босом вимагає ретельної підготовки, збору ресурсів і розробки стратегії для успішної перемоги.

Рис. 1.2 Демонстрація геймплею в грі Valheim

3) **Rust** рис. 1.3, гра, про виживання у відкритому світі, створеному Facepunch Studio. Дана гра ставить гравців в онлайн-середовище, головною метою якого є виживання. З наголосом на взаємодії гравців, будівництві бази, зборі ресурсів і PvP-битвах Rust пропонує складний і захоплюючий ігровий процес.

Гра Rust фокусується на виживанні у ворожому світі. Гравці починають без ресурсів і повинні збирати зброю, добувати, зберігати та будувати притулки, щоб вижити. Ігровий світ повний інших гравців, кожен зі своїми цілями та стратегіями, що робить робоче середовище динамічним і непередбачуваним.

Одним із найважливіших аспектів ігри є базова будівельна система. Гравці можуть будувати складні конструкції з різних матеріалів, що відкриває можливості для творчості та стратегічного планування. Бази служать прикриттям і захистом від інших гравців, але їх також можна атакувати, додаючи грі відчуття постійного напруження та небезпеки.

Rust популярний серед ігор на виживання через складний геймплей, глибоке складність та швидку взаємодію з гравцями. Потрібен стрімкий курс навчання та невблаганне середовище, яке може відштовхнути деяких гравців, але ті, хто витримає, знайдуть цей досвід унікальним і захоплюючим. Незалежно від того, чи любите створювати унікальну базу, битися з іншими гравцями чи просто занурюватися в напружений світ Rust, гра пропонує достатньо вмісту та нових функцій, щоб надовго поринути в гру. Приємні випробувань на виживання та несподіваних розмов між гравцями-однодумцями.

Система складності Rust є одним із основних елементів, які роблять гру захоплюючою та складною для гравців. Вона містить кілька елементів, які забезпечують високу напругу і непередбачуваність під час гри.

Система складності в Rust складається з кількох аспектів. Гравцям доведеться мати справу з такими природними загрозами, як дикі тварини та радіація. і взаємодіяти з іншими гравцями викликаючи непередбачувані ситуації. Ресурси, створення бази включають систему бою зі складними компонентами. Крім того, цикли оновлення сервера змушують гравців постійно адаптуватися.

Адаптивна система складності Rust створює динамічний ігровий процес, який змушує гравців постійно переглядати свої стратегії. Ігрове середовище змінюється залежно від дій гравця та властивостей сервера.

Наприклад, новим гравцям буде важко вижити, якщо не зможуть адаптуватися до загроз від досвідчених гравців або природних факторів, таких як зміна клімату або виснаження ресурсів у певних регіонах.

Ключовою механікою є цикл очищення, а саме регулярне оновлення прогресу сервера, який вирівнює умови гри як для новачків, так і для ветеранів. Після очищення всі гравці починають з нуля, але перевагу отримують ті, хто може швидко збирати ресурси та будувати бази. Це сприяє подальшому розвитку навичок і гнучкості мислення.

Рис. 1.3 Демонстрація геймплею в грі Rust

Крім того, Rust має прогресивну систему складності, яка базується на локаціях. Початкові зони зазвичай менш небезпечні, але в міру того, як впевненість і потреба зростають, гравці змушені заходити в більш небезпечні зони з радіацією, потужними супротивниками NPC або конкуруючими ресурсами. Це природно вчить гравців планувати свої дії та заздалегідь оцінювати ризики.

Аспект соціальної взаємодії додає Rust ще один рівень складності, оскільки гра активно заохочує взаємодію між гравцями. Можна жити разом або приєднуватися до команд, щоб відбивати атаки інших гравців, які можуть здійснити набіг на вашу базу, викрасти ресурси чи заволодіти територією. Будь-яке зіткнення з гравцем у грі — від кооперативних боїв до боїв PvP — може бути непередбачуваним, тому гравці змушені постійно коригувати свої дії. Впевненість, обережність і стратегічне планування стають важливими не тільки під час взаємодії з оточенням, але й під час взаємодії з іншими гравцями, створюючи унікальний досвід для кожної ігрової сесії.

Складність даної гри також виникає через необхідність постійно шукати ресурси та виживати в постійно мінливому середовищі. Кожен аспект гри вимагає від гравців стратегічного мислення та гнучкості у вирішенні проблем.

4) Project Zomboid рис. 1.4, схвалена критиками гра-пісочниця про виживання зомбі-апокаліпсису, яка поєднує в собі реалістичність, глибокий геймплей і широкі можливості налаштування. Гра пропонує гравцям унікальний і захоплюючий досвід, заснований на необхідності адаптуватися до середовища, що постійно змінюється, долати виклики та приймати рішення, які безпосередньо вплинуть на їхнє виживання. Дана гра — це інтерактивна пригода, наповнена стратегічними й емоційними моментами, завдяки своїй механіці і дизайну.

У Project Zomboid гравці мають свободу створювати власні правила, які формують їхній унікальний досвід виживання. Це включає вибір таких параметрів, як:

- час початку матчу;
- кількість і щільність зомбі;
- їх поведінку та швидкість;
- Налаштування стилю гри, яке включає акцент на скритності, прямому бою або виживанні під час взаємодії зі світом;

Ці елементи утворюють концепцію, яка визначає, як ми бачимо світ гри та взаємодіємо з ним. Гравці по суті визначають межі своєї історії виживання, пробуючи різні комбінації правил. Такий підхід підвищує залучення гравців, роблячи кожну гру унікальною та індивідуальною

Інтерфейс Project Zomboid створений для надання гравцям усієї необхідної інформації, не відволікаючи їхню увагу від основної гри.

Елементи HUD

- здоров'я персонажу;
- рівень голоду, втоми, стресу, болю та інших факторів;
- Інвентар та доступні ресурси;
- про актуальні виклики та загрози;

Наприклад, якщо персонаж голодний, на екрані з’явиться спеціальний індикатор, який сповістить гравця негайно шукати їжу. Якщо цього не зробити, статус персонажа знизиться, що може призвести до зниження його характеристик. HUD не тільки інформує, але й заохочує гравця звертати увагу на деталі та залучає їх до контролю над важливими аспектами гри.

Інтерфейс гри також включає можливість взаємодії з навколишнім світом, зокрема читання нотаток, вивчення карт або прослуховування радіо та перегляду телевізора, що показує інші частини історії.

Однією з головних особливостей Project Zomboid є ігровий процес, а саме можливість створювати унікальні сценарії на основі дій гравця та взаємодії із зовнішнім світом.

Відкритий ігровий світ повністю відкритий для дослідження. Кожен будинок, вулицю та будівлю можна досліджувати, щоб знайти ресурси чи схованки.

Рішення кожного гравця впливають на розвиток унікальної історії виживання, створюючи цікаві історії. Наприклад, вибір укриття або стилю боротьби з зомбі може призвести до непередбачених наслідків.

Інтерактивні засоби масової інформації: за допомогою телевізорів, радіоприймачів і внутрішньоігрових нотаток ви можете дізнаватися про світ і події, які призвели до зомбі-апокаліпсису. Це збільшує занурення та додає додатковий штрих.

Зростання змушує акторів бути креативними, змушуючи їх знаходити творчі рішення в умовах обмежених ресурсів і непередбачуваних загроз.

Ігрова механіка Project Zomboid базується на взаємодії гравця зі світом, вирішенні проблем і досягненні цілей.

Створення персонажа: гравці починають з вибору навичок, рис і класу персонажа, які визначають їхній стиль гри.

Розвиток навичок. Гравці можуть покращувати навички свого персонажа, такі як сила, скритність, приготування їжі, будівництво чи надання першої допомоги, що дозволяє їм адаптуватися до різних викликів.

Управління ресурсами, це одна з головних цілей гри - знайти їжу, воду, ліки та зброю. Неправильне управління інвентарем може призвести до смерті персонажа. Гравці можуть створювати притулки, щоб захиститися від зомбі.

Адаптація та прийняття рішень: складність ігрового процесу

У Project Zomboid складність гри зростає, вимагаючи від гравців постійної адаптації та вдосконалення навичок прийняття рішень. Гра віртуозно створює напругу та підвищує рівень складності завдяки поєднанню навмисних ускладнень механіки, динамічних змін у ігровому світі та випадкових подій.

З часом загроза Project Zomboid від зовнішніх факторів зростає. На початку гравцям легше уникати битв або стикатися з невеликими групами зомбі. Однак у міру проходження гри щільність заражених зростає, а їхня поведінка стає більш агресивною. Наприклад, з часом зомбі помічають гравця на великій відстані і активніше реагують на звуки.

Зміна пір року приносить додаткові виклики. У той час як температура взимку падає, а теплий одяг і паливо потрібні, щоб зігрітися, волога погода підвищує ризик застуди та проблем зі здоров’ям.

Місця закінчуються - якщо гравець забрав усі запаси з дому чи магазину, ці ресурси не будуть поповнені. Гравець віддаляється від безпечних місць все далі і далі, піддаючи себе небезпеці.

Гра активно використовує механізми випадкових подій, щоб порушити комфорт гравця. Рідко зустрічаються автомобілі або сигналізація можуть активувати натовпи зомбі і залучати їх до місця розташування гравця. Тут потрібна неперевешена адаптація – або швидка втеча, пошук тимчасового притулку.

Через кілька тижнів після того як почнете грати, все електропостачання та водопостачання буде припинено, що змушує гравців шукати альтернативи. Наприклад, вони повинні збирати дощову воду або купувати генератори та паливо.

Невелика помилка, наприклад, вистрибування з вікна або розрізання розбитого скла, може мати серйозні наслідки, включаючи інфекцію або смерть. Ці елементи створюють постійне відчуття невизначеності та змушують гравця бути готовим до несподіванок.

Гравці часто стикаються з рішеннями, які потрібно приймати швидко під тиском обставин. Часу на довгий аналіз майже немає, що робить кожну дію критичною. Якщо гравець помітив групу зомбі, повинен негайно перевірити, чи зможе перемогти, чи краще втекти. Рішення залежить від рівня здоров’я його, доступної зброї та стану навколишнього середовища.

Коли інвентар заповниться, гравець повинен вирішити, які предмети залишити, а які видалити. Це рішення стає особливо складним, коли ресурси мають різні цілі.

Часте зіткнення з погрозами підвищує рівень стресу персонажа. Гравець повинен стежити за своїм психічним станом, інакше персонаж може стати менш ефективним у бою або навіть впасти в паніку.

З часом ресурси стає все важче знайти. Їжа починає псуватися, і гравець повинен знайти більш стійке рішення, таке як землеробство чи полювання.

Обмежений запас аптечок, патронів та ремонтних інструментів. Дефіцит може змусити гравця використовувати менш надійні методи лікування чи захисту.

Через постійно зростаючу кількість зомбі, домівка може стати вразливою, вимагаючи модернізації укріплень або переміщення в абсолютно нове місце. Ці обмеження змушують гравців бути стратегічними, ретельно планувати кожну дію та передбачати можливі проблеми.

Рис. 1.4 Демонстрація геймплею в грі Project Zomboid

5) Conan Exiles рис. 1.5, багатокористувацька екшн-гра, розроблена компанією Funcom, яка занурює гравців у зловісний світ Хайборійської епохи, натхненний творчістю Роберта Е. Говарда. Гра пропонує гравцям унікальну можливість виживати, досліджувати величезні території, битися з таємничими монстрами, створювати власні укріплення та взаємодіяти з іншими гравцями у ворожому оточенні. Вся гра побудована навколо принципів виживання, де кожен крок і вибір можуть вирішувати життя.

Conan Exiles починається з інтригуючої передумови: гравець стає піратом, залишеним помирати на хресті в пустелі, у найбільш безплідних землях Гіборії. Цей жорстокий світ протягом століть населяли найрізноманітніші люди, раси та істоти, і персонаж — в’язень, який повинен вижити та знайти своє місце в цьому

невблаганному світі. Коли Конан звільняє гравця, дає їм шанс на нове життя, звільняючи їх до «Земель вигнання», які стають місцем виживання та пригод.

Гра дуже жорстока і ставить перед гравцем безліч складних завдань. Зіткнетеся з багатьма загрозами від інших гравців і NPC (неігрових персонажів), а також природними загрозами з навколишнього світу. Багато небезпечних речей відбувається.

Не тільки деякі гравці та NPC загрожують вашому життю, але й дику природу не можна недооцінювати. В грі живуть вовки, лисиці, ведмеді та інші загадкові істоти, які можуть завдати смертельної шкоди. У деяких областях навіть є гігантські зомбі та інші монстри, що вимагає від гравців надзвичайної підготовки та обережності.

Ще одним важливим фактором безпеки є погодні умови. Наприклад, пустеля потребує води та захисту від стихій, тоді як у північному кліматі знадобиться відповідний одяг, щоб захистити вас від стихій. Урагани, шторми та інші стихійні лиха можуть кардинально змінити хід подій, порушити ваші плани або вбити вас.

Щоб вижити, гравці повинні збирати ресурси. Збирати камінь, дерево, сталь, дерево, збирати шкури тварин тощо, щоб створювати зброю, інструменти та будівельні матеріали. Ресурси обмежені, і їх доведеться шукати в різних частинах світу. Іноді предмети можуть утримувати могутні вороги, що ускладнює процес придбання.

Окрім захисту від монстрів, також будуть труднощі з іншими гравцями. Залежно від обраного режиму сервера (PvP або PvE), битви можна проводити в будь-який час. Під час битв на сервері PvP інші гравці можуть здійснювати набіги на табори, грабувати предмети та атакувати. У PvE гравці зосереджені на взаємодії зі світом, а не на змаганні з іншими людьми.

Крафт — один із головних аспектів Conan Exiles, але він неймовірно складний і потребує уваги. По-перше, гравці можуть створювати основні інструменти та зброю, як-от кам’яні кирки, мечі чи щити, які можна використовувати для захисту та здобичі. Але в міру того, як відбувається прогрес,

складність виготовлення зростає, тому для створення більш складних предметів, таких як зброя чи додаткова зброя, потрібно більше майстерності.

Щоб створити набір зброї, потрібно буде використовувати кілька укриттів: піч, порох і укриття зброї. Створювати шкури тварин, потім розплавлювати метали та збирати усі матеріали з відповідних джерел. Будівництво також відіграє важливу роль розвивати базу або навіть цілий замок, де доведеться захищатися від ворогів, стихій та інших гравців.

Будівництво не тільки підвищує відчуття безпеки, але й дає можливість вдосконалювати зброю та інші предмети в безпечному середовищі. Крім того, на деяких серверах можна накласти спеціальні правила для збірок; наприклад, зробити їх непорушними.

Коли гравці досліджують світ, вони стикаються з більш складними областями та підземеллями. Певні частини карти, такі як вулкани чи глибокі печери, наповнені ворогами, прихованими пастками та іншими небезпеками. Ці зони вимагають не тільки обладнання, але й високого рівня підготовки, що робить дослідження світу справжнім викликом.

Згодом гравці зіткнуться зі складнішими проблемами виживання. Більшого значення набуває потреба постійно стежити за своїм здоров’ям, підтримувати рівень їжі та води, захищатися від різноманітних природних явищ, наприклад піщаних бур, морозів. Необхідність постійно адаптуватися до мінливих умов вимагає серйозної підготовки з боку гравця.

Крім того, гра містить випадкові події, такі як атаки великих груп ворогів, стихійні лиха, а саме бурі, лавини. Ці елементи роблять гру більш динамічною та непередбачуваною, яка постійно підтримує високий рівень складності та потребу адаптуватися до нових умов.

У результаті складність Conan Exiles зростає, а разом з цим і рівень відповідальності, яку повинні взяти на себе гравці. У грі представлені нові, більш складні механіки, області та вороги, які адаптуються до рівня навичок гравця, вимагаючи здатності пристосовуватися до обставин і розробки нових стратегій виживання.

Рис. 1.5 Демонстрація геймплею в грі Conan Exiles

6) Starbound рис. 1.6. Щоразу, коли персонаж в своєму космічному кораблі подорожує у світи Starbound, ця двовимірна гра з прокруткою переносить в захоплюючу пригоду крізь космос до його глибин. Дослідження планет, біомів і підземного світу є основою захоплюючої подорожі з динамічними битвами, добре продуманими босами та насиченою історією. Еволюція формули Terraria, ця гра сповнена яскравих моментів і відкриттів.

Механіка видобутку ресурсів, створення предметів і крафта знайома шанувальникам жанру, але Starbound привносить особливий шарм у різноманітний контент. Незважаючи на те, що початок здається повільним (маніпулятор об’єктів повільний, і прогрес може бути легко під загрозою через ранніх ворогів), гра дізнається великим досвідом. На кожній планеті чекають унікальні біоми, приховані руїни, ресурси та пригоди. Кожен новий світ має свою історію. Від джунглів до лавових океанів і процедурно створених лісів – немає двох однакових планет. Навіть під землею різноманітність вражає. Наприклад, досліджуючи замерзлу планету можна знайти кістяний храм, захований під стімпанковою шахтою, яка веде через крижані печери до лавового підземелля. Такі сюрпризи чекають буквально за кожним кутом.

У грі йдеться не лише про збір ресурсів, а й про турботу про місцевих жителів. Бойова система радує своєю точністю, тактикою і різноманітністю. Незалежно від того, чи обирати мечі та щити чи футуристичну зброю, битва залишається динамічною та цікавою. Кожен тип зброї пропонує власний підхід до гри, а вороги з унікальною тактикою змушують адаптуватися.

Особливо цікаві сюжетні боси: кожен — унікальний виклик. Від гігантських драконів-скелетів до спритних фехтувальників зі складними моделями атак, це нагадує битви в класичних іграх, таких як Mega Man. Ретро, складний дизайн додає адреналіну в грі.

У той же час Starbound може розчаровувати через хитру систему смертної кари. Якщо помрете далеко від свого корабля, ризикуєте втратити всі ресурси, які зібрали, а спроби відновити їх у суворих умовах часто ускладнюються поверненням ворогів. Пізніше в грі це стає ще більш критичним, оскільки запаси, необхідні для виживання, важко поповнити. Смерть посеред глибокої експедиції може створити порочне коло, у якому ресурси закінчаться швидше, ніж можна їх знайти.

Історія гри цікава, але світ, сповнений цікавих персонажів і деталей. Шість ігрових рас мають власну культуру, архітектуру та конфлікти. Від агресивних рослинних гуманоїдів до середньовічних роботів, кожна зустріч додає простору цьому всесвіту. Сюжетні підземелля та секретні документи розкривають більше про світову політику, звичаї та історію, створюючи відчуття справжньої галактичної епопеї.

Масштабування складності в Starbound базується на прогресивному прогресі гравця, який залежить від збору ресурсів, виготовлення обладнання та освоєння нової механіки. У грі спочатку представлені базові бойові системи та механіка, а поступово додаються складніші випробування, які вимагають вдосконалення обладнання та тактичного мислення.

Спочатку гравець досліджує прості планети, де вороги слабкі, а ресурси легко отримати. Це дозволяє навчитися основам крафта за допомогою маніпуляторів об’єктів і будівництва. Справжня проблема полягає в обмежених інструментах, які

впливають на темп розвитку, але самі середовища та вороги не становлять великої загрози.

У міру просування в грі завдання стають складнішими. Завдання представляють нові типи планет, більш агресивні вороги і унікальну бойову механіку, яка заохочує до тактичних експериментів. Деякі вороги мають стійкість до певних типів зброї або складних моделей атак, які потрібно вивчити. Екстремальні планети, наприклад, з високою температурою або радіацією, вимагають спеціального обладнання, що значно збільшує час підготовки та витрати ресурсів.

Сюжетні боси служать прискоренням складності. Вони мають унікальні здібності та виводять на новий рівень, вимагаючи адаптуватися до тактики бою. Ці типи конфліктів перевіряють знання гравця, ігрову механіку, реакцію та навички стратегічного мислення. Деяким босам потрібен ідеальний час ухилення, тоді як іншим потрібна комбінація зброї ближнього бою та дальньої зброї.

Оскільки планети генеруються процедурно, складність гри також залежить від удачі. Деякі світи можуть бути багатими на ресурси, але небезпечними через складну місцевість або ворогів, а інші залишаються мирними, але бідними на цінні матеріали. Це створює елемент непередбачуваності, що робить кожну планету унікальною.

Ризик і винагорода є важливими елементами. Досліджуючи небезпечні біоми чи глибокі підземелля, можна знайти рідкісні ресурси, але є ризик померти, втративши свої заощадження. Це вимагає більш ретельного планування перевезень з урахуванням можливих ризиків.

По ходу гри гравець змушений змінювати свій підхід до викликів. Це передбачає перехід від захисної гри до активного бою, адаптацію спорядження до конкретних умов і тестування різної зброї. Зброя далекого бою може бути ефективною на початку, але ближче до кінця виникає необхідність використовувати щити або робити швидкі атаки ближнього бою. Така динаміка підтримує інтерес і змушує постійно вдосконалювати свої навички.

Рис. 1.6 Демонстрація геймплею в грі Starbound

7) 7 Days To Die рис. 1.7, гра на виживання в відкритому світу, яка поєднує елементи виживання, будівництва, RPG і зомбі-апокаліпсису.

Вимірювачі голоду, дратівливі монстри та нелогічне створення речей — усе це частина даної гри. Ця гра однією з перших задала стандарти жанру. За ці роки вона отримала багато оновлень: покращена графіка, дерево навичок, змінена зброя, торгові точки. Але основний механізм збору ресурсів у стилі що знайдеш, то твоє залишився майже незмінним. Хоча ця суть гри не змінилася, її надійність надає проєкту особливого шарму.

Основна ідея гри залишається незмінною. Шукати притулок, укріплювати його, намагатися вижити після нападу зомбі, а коли неминуче втрачається контроль, тікати, щоб знайти новий дім. 7 Days to Die перетворює зомбі на стихійне лихо. Це своєрідна адаптація історії серіалів на зразок “Ходячих мерців”, де в основі ігрового процесу лежить подорож із табору на ферму чи з в’язниці в місто.

У 2013 році все було дуже весело, але тепер, завдяки деяким покращенням, усе виглядає ще краще. Коли персонаж помре, гра дає можливість відродитися у в таборі або поблизу місця смерті. Смерть зазвичай призводить до втрати XP, але через деякий час це покарання більше не застосовується. За певної складності

смерть може навіть допомогти, вилікувавши хвороби. Ця механіка схожа на Minecraft, але без суворих наслідків.

Вода та їжа тут важливі не лише для виживання. Без них регенерація витривалості сповільниться, що зробить персонажа менш ефективними в бою. Іноді можна померти від виснаження, бо махати бейсбольною битою є важкою дією. Цей аспект створює напругу, яка змушує балансувати між творчістю та створенням своєї бази та боротьбою із зовнішніми загрозами.

У світі 7 Days to Die головним ворогом є не лише голод чи хвороби, а й постійна загроза зомбі. Спокусливо вибрати найнижчий рівень складності, щоб зосередитися на будівництві. Однак це порушує баланс гри. Це рішення зменшує напругу, яка є невід'ємною частиною досвіду.

Пошуки нового будинку в цій грі нагадують популярні серіали про нерухомість. Знаходять занедбану будівлю з великим потенціалом і облаштовують її. Будь то старий фермерський будинок із сараєм чи гірський табір, будь-яке місце можна перетворити на фортецю з ровом, пастками та кілками. Але кожен новий будинок також приваблює зомбі, які руйнують все на своєму шляху.

Тому завжди потрібно бути готовим до руху. Гра спонукає досліджувати та знаходити цікаві будівлі: кінотеатри, військові бази, школи. Найкращі місця для життя – це ті, які легко захищаються, мають воду та гарний вид на околиці. Кожен крок базової конструкції тут приносить якесь задоволення.

Налаштування складності 7 Days to Die розроблено таким чином, щоб дозволити кожному гравцеві насолоджуватися грою, незалежно від його досвіду чи вподобань. Це дозволяє налаштувати майже кожен аспект гри, від поведінки зомбі до рівня виживання, забезпечуючи таким чином баланс між викликом і комфортом.

Одним з важливих аспектів є зміна поведінки зомбі. Якщо віддавати перевагу більш спокійній грі, зомбі будуть повільнішими та менш агресивними. Якщо кинути виклик, вони будуть швидшими, сильнішими та небезпечнішими, особливо вночі.

Рис. 1.7 Демонстрація геймплею в грі 7 Days to Die

Система безпеки також підлаштовується під гравця, можна змінювати частоту голоду та спраги, швидкість виснаження ресурсів і кількість доступної здобичі. Це дозволяє зробити гру більш насиченою або, навпаки, зосередитися на розробці та дослідженнях.

Однією з особливостей гри є те, що гра динамічно адаптується до прогресу. Якщо успішно вижито, побудувано міцний дім і отримано гарне спорядження, гра у відповідь підвищує рівень складності. Зомбі розмножуються, стають сильнішими і частіше атакують великими хвилями. Це несе нові виклики та вимагає, щоб завжди були готові до несподіванок.

Для початківців або тих, хто просто хоче будувати, є полегшення з меншим стресом. Тут є можливість нейтралізувати зомбі, отримувати безмежні ресурси та експериментувати з крафтом і будівництвом. Чудовий спосіб досліджувати світ і втілювати в життя найсміливіші архітектурні ідеї.

2. АНАЛІЗ ТА ПОРІВНЯННЯ МЕТОДІВ АДАПТАЦІЇ СКЛАДНОСТІ ВОРОГІВ У ЖАНРІ "ВИЖИВАННЯ 2.5D"

2.2 Аналіз методів реалізації складності штучного інтелекту

Штучний інтелект постійно розвивається, і його застосування варіюється від промислової автоматизації до медичної діагностики, фінансової аналітики та широкого спектру галузей. Це відкриває нові можливості для покращення якості життя людей та покращення різноманітних процесів у суспільстві. Однак із його зростанням виникають етичні проблеми та виклики, пов’язані з його впливом на прозорість, безпеку та зайнятість. Тому важливо продовжувати вивчати й обговорювати роль штучного інтелекту та його вплив на суспільство, а також розробляти ефективні стратегії для максимізації його переваг і пом’якшення ризиків.

Крім того, зростає потреба зробити технологію штучного інтелекту однаково доступною для всіх сфер життя та розробити механізм контролю за використанням її потенціалу. Розвиток штучного інтелекту вимагає уваги до питань конфіденційності та захисту даних, оскільки він покладається на великі обсяги даних. Щоб забезпечити екологічність, важливо враховувати екологічну стійкість технологій ШІ.

Наприклад, необхідно розробити стандарти та етичні принципи використання автоматизованих систем і роботів з використанням штучного інтелекту, особливо у ігровій сфері. Крім того, важливо враховувати можливі соціальні та культурні наслідки впровадження штучного інтелекту, тобто вплив на роботу та соціальні відносини.

Штучний інтелект має великий потенціал у сфері відеоігор, допомагаючи створювати більш складні, захоплюючі та реалістичні ігрові світи. Використовуючи алгоритми машинного навчання та нейронні мережі, розробники створюють персонажів і ворогів, які можуть змінювати свою поведінку залежно від

дій гравця, що робить кожну гру унікальною. Такі рішення штучного інтелекту забезпечують кращу взаємодію між персонажами та ігровим середовищем і дозволяють більш правдоподібно реагувати на зміни обставин.

З іншого боку, використання штучного інтелекту в ігровій індустрії відкриває нові можливості для персоналізації ігрового досвіду. Адаптивні алгоритми дозволяють гравцям змінювати складність, режим гри або навіть історію та індивідуальні вподобання та навички гравця. Це значно підвищує можливість відтворення гри, оскільки кожен гравець може пережити унікальну історію та взаємодіяти зі світом по-своєму.

Важливою частиною розвитку штучного інтелекту в ігровій індустрії також є його використання в автоматизованому тестуванні ігор. Алгоритми ШІ можуть аналізувати ігри, виявляти помилки, виявляти проблеми з балансом і давати рекомендації щодо покращення гри. Це значно прискорює етап тестування та гарантує, що продукти містять менше помилок, що покращує ігровий досвід.

Ще меншим є вплив штучного інтелекту та розробки інтерактивних ігор з відкритим світом. Штучний інтелект можна використовувати для створення складних екосистем, де все, від рослин до тварин, має власну поведінку, що залежить від взаємодії з іншими частинами навколишнього світу. Це дозволяє створювати більш реалістичні та реалістичні ігрові світи, де гравці можуть не тільки виконувати основні завдання, а й впливати на оточення та змінювати хід історії.

Штучний інтелект в індустрії розваг також відкриває нові горизонти для створення глибших і складніших історій. Завдяки використанню алгоритмів, які аналізують гравця та його рішення, розроблено історію, яка змінюється відповідно до вибору та робить кожну поразку чи перемогу унікальною для гравця. Цей інтерактивний підхід також може створити більш реалістичну та емоційно насичену взаємодію, дозволяючи гравцям розвивати не лише розважальне почуття гумору, але й глибоке відчуття можливостей у вигаданому світі.

Крім того, штучний інтелект дозволяє створювати розумніших суперників у багатокористувацьких іграх, значно підвищуючи рівень конкуренції та взаємодії. Такий штучний інтелект може не тільки реагувати на дії гравця, але й вчитися на його стратегіях, що підвищує складність гри. В результаті кожен матч стає все більш захоплюючим і непередбачуваним.

Ще одна сфера, де штучний інтелект приносить великі переваги, полягає в покращенні досвіду багатокористувацької гри. Використовуючи машинне навчання та аналітику великих даних, можна автоматично виявляти та усувати недоліки, зберігаючи ігрове поле чистим. Цей тип контролю також покращує досвід гравців і забезпечує чесну конкуренцію та чесне ігрове середовище.

Розвиток технологій штучного інтелекту також призводить до появи нових ігрових полів, де штучний інтелект виступає не лише як опонент, але й як партнер. У цих іграх ШІ може допомогти гравцеві, дати йому тактичні поради або навіть розвивати історію разом з ним. Це відкриває нові можливості для гравців, які шукають більш плавний та інтуїтивно зрозумілий спосіб взаємодії з гравцями.

Згодом штучний інтелект може стати важливим чинником не лише у розробці самих ігор, а також постійному вдосконаленні. Щоб забезпечити безперервне навчання та адаптацію, ігри можуть розвиватися після запуску, пропонуючи новий вміст або кардинально змінюючи світ залежно від дій гравця. Це дозволяє грати в гру довше, що робить її більш привабливою для більшої аудиторії.

Штучний інтелект в іграх жанру виживання можна реалізувати за допомогою різноманітних методів, щоб надати гравцям відчуття реалізму та викликів. Ось кілька найпоширеніших способів впровадження складності штучного інтелекту:

- Метод поведінкових дерев.
- Модель машин скінченних станів.
- Метод генетичних алгоритмів.
- Метод машинне навчання.
- Метод баєсових мереж.

2.2.1 Метод поведінкових дерев

Метод поведінкових дерев рис 2.1 це техніка в аналізі даних і машинному навчанні, що базується на ідеї створення моделей прийняття рішень у вигляді дерева. Зростання складності з кількістю ланцюгів та їх взаємодій. Дерево поведінки може мати десятки або сотні вузлів, кожен з яких представляє певну дію чи ситуацію. Зі збільшенням кількості вузлів у дереві стає важко контролювати вручну та розуміти логіку поведінки інтелекту. Велика кількість вузлів може збільшити час вирішення та ускладнити пошук помилок.

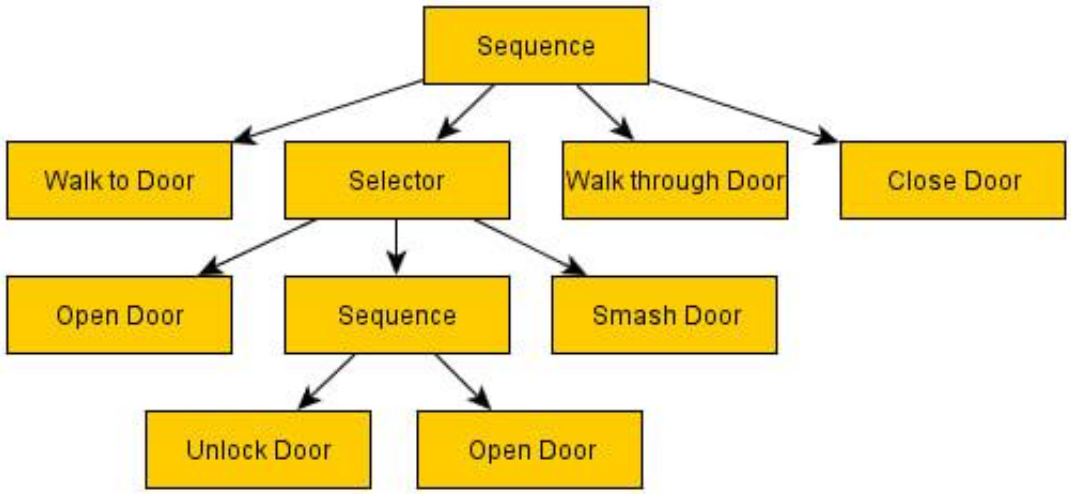


Рис. 2.1 Приклад діаграми поведінкових дерев

У великих і складних ієрархічних деревах може бути багато комбінацій дій і умов. Навіть при найдетальнішому аналізі неможливо передбачити всі шляхи, якими піде дерево в різних ігрових ситуаціях. У деяких випадках це може призвести до несподіваних результатів і непередбачуваної поведінки штучного інтелекту. Хоча дерева поведінки є інструментом для моделювання складної поведінки штучного інтелекту, їх складність може зростати зі збільшенням розміру та складності дерева. Це може вимагати додаткової розробки, налаштування та налагодження, а також ретельного аналізу та тестування, щоб забезпечити бажану поведінку штучного інтелекту у різних ігрових сценаріях.

Дерева поведінки є ефективним інструментом для побудови логіки поведінки ШІ в іграх та інших програмах. Вони забезпечують простий і зрозумілий спосіб моделювання поведінки, який легко реалізувати та підтримувати.

Хоча дерева поведінки мають деякі обмеження, переваги значні та можуть допомогти розробникам швидше створювати якісні ігри. Підхід дерева поведінки залишається одним із найкращих способів розробки логіки штучного інтелекту в комп'ютерних іграх. Він забезпечує гнучкість і масштабованість для створення складних і унікальних моделей персонажів, особливо в області "виживання 2.5D", де штучний інтелект динамічно зміяюэться у середовищі.

Одним із найцікавіших аспектів розробки методу є інтеграція дерев поведінки зі складними та динамічними системами адаптації. Це призводить до таких факторів:

Рівень навичок гравця (P(t)): використовувати методи аналізу даних гри, щоб оцінити поточну продуктивність гравця та виправити проблеми. Стан ворога (E(t)): Залежно від здоров'я, агресії чи сили гравця ворог може вибирати різні способи боротьби з гравцем.

Параметри навколишнього середовища (M(t)): наприклад, рівень освітлення, доступність зберігання або обмеження простору можуть впливати на дії ворожого персонажу.

Динамічна адаптація викликається змінами в дереві або в коефіцієнтах, які змінюють пріоритет діяльності. Передбачати поведінку гравців, щоб створити цікавіші ігрові завдання. Такий підхід особливо важливий у жанрі "Виживання 2.5D", де сценарії часто непередбачувані, і ШІ повинен адаптуватися до конкретних ситуацій.

У жанрі "Виживання 2.5D" зазвичай використовуються дерева високого рівня:

- Цілі світу (виживання, накопичення багатства, притулок).
- Локальні дії (відстеження, напад, пошук ворогів).
- Специфічні умови (поведінкова економіка, наявність матеріалів).

Наприклад, якщо у гравця низьке здоров'я, ШІ може перемикатися з нормального на агресивний за допомогою спеціальних «вузлів ситуації», які активуються за певних умов.

Враховуючи складність розміщення великих дерев, сучасні засоби автоматизації значно полегшують це завдання. Використання візуальних підказок для створення дерев поведінки, автоматизація також дозволяє швидко тестувати альтернативні стратегії поведінки, визначаючи межі логіки.

Незважаючи на свої переваги, поведінкові дерева мають кілька недоліків:

- Обмежена здатність до читання. ШІ не може змінити якість дерев без попереднього втручання.
- Високі вимоги до ресурсів. Більші дерева збільшують витрати на обчислення, що може сповільнити гру.

Щоб подолати ці обмеження, розробники використовують гібридні підходи, поєднуючи дерева функцій і нейронні мережі. Це відкриває нові можливості для створення інтелектуального ШІ, здатного ефективно взаємодіяти з гравцем навіть у складних ситуаціях.

Метод поведінкових дерев базується на методі поведінкових дерев та враховує комплексну реакцію. Нехай S(t) буде функцією складності, яка динамічно змінюється з часом залежно від рівня навичок гравця P(t), стану ворога E(t) і параметрів середовища M(t).

Метод можна представити у вигляді рекурсивної функції переходу між станами системи.

- S – множина всіх можливих станів системи.
- A — набір можливих дій.
- T(s, a) є функцією переходу, яка визначає наступний стан системи, коли дія a застосовується до стану s.

Загальна дерево поведінки виглядає так: S_{t+1} = T(S_t, A_t), де S_t

A_t — це дія або подія, яка відбувається в момент часу t.

S_{t+1} — наступний стан, до якого система переходить згідно з правилом переходу T.

Якщо уявити дерево як ієрархію, кожну наступну гілку можна представити як ряд можливих станів і дій, які повторюються знову і знову, поки не буде досягнуто кінцевих станів. Це можна виразити так: S_{t+1} = T(T(S_{t-1}, A_{t-1}), A_t);

Функція складності D(t) = f(Sg(t), Se(t), M(t)). M(t) – параметри середовища, які можуть впливати на динаміку бою (наприклад, рівень освітленості, ландшафт). f — це функція, яка модулює ступінь складності на основі агрегованих параметрів. Здоров'я ворога H(t): $H(t) = H0 + kH \cdot D(t) \cdot e^{(-c \cdot (t - t0))}$.

Тут H0 — початковий рівень здоров’я, kH — коефіцієнт коригування здоров’я, а c — константа, яка регулює швидкість, з якою змінюється здоров’я.
Сила атаки ворога A(t):

$A(t) = A0 + kA \cdot D(t) \cdot (1 + d \cdot \sin(\omega \cdot t))$, де A0 — початкова сила, kA — коефіцієнт регулювання сили, d —константа, а ω — частота.

Швидкість противника B(t): $B(t) = B0 + kB \cdot D(t) / (1 + e^{(-\lambda \cdot (t - t0))})$, де B0 — початкова швидкість, kB — коефіцієнт регулювання швидкості, а λ — постійна нахилу.

Дерево рішень визначає поведінку супротивника (наприклад, напад або захист) на основі значень D(t), Sg(t) і Se(t). Наприклад:

Коли D(t) високий, супротивник стає більш агресивним і збільшує A(t) і B(t). Якщо Sg(t) падає (гравець у складній позиції), суперник може знизити рівень агресії. Формула для повного налаштування характеристик ворога на основі дерева поведінки така:

$$St+1 = T(St, At, D(t)) \quad (2.1)$$

Str+1 - новий стан противника зі скоригованими характерис/тиками. T — це функція /переходу дерева поведінки, яка використовує D(t) для налаштування властивостей.

At — супротивна дія, вибрана на основі поточного стану системи та складності D(t).

2.2.2 Модель машин скінченних станів

Модель машин скінченних станів, рис 2.2 (FSM) - це математична модель, що використовується для опису та керування поведінкою систем, які можуть перебувати у скінченній кількості станів у будь-який момент часу.

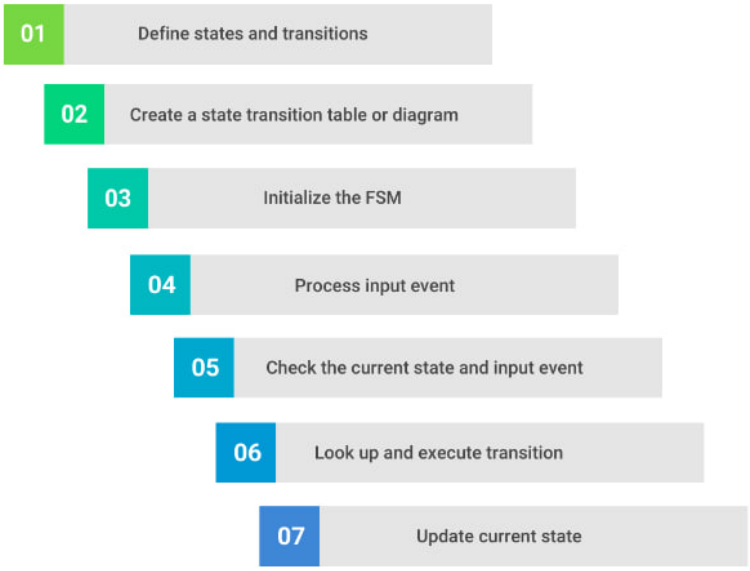


Рис. 2.2 Приклад моделі машин скінченних станів

Складність полягає в зростанні зі збільшенням кількості станів і переходів. У кінцевому стані (FSM) кожен стан представляє поведінку штучного інтелекту у певній ситуації. Зі збільшенням кількості станів у FSM стає важко відстежувати та контролювати різні аспекти поведінки штучного інтелекту. Додавання нових станів може вимагати змін логіки переходу або всієї системи керування штучного інтелекту, що збільшує складність коду та ризик помилок. Важко усунути несправність і зрозуміти логіку поведінки штучного інтелекту: В іграх із декількома станами та переходами може бути важко відстежити, у якому стані перебуває штучного інтелекту у будь-який момент часу.

Сучасні підходи до генерації FSM включають автоматичну генерацію та синтез моделей з даних. Наприклад, методи машинного навчання можна використовувати для автоматичного визначення найкращих проміжків і переходів

на основі реальних даних. Це дозволяє створювати оптимізовані та ефективніші FSM, таким чином зменшуючи людський фактор і час, витрачений на розробку.

Одним із таких підходів є використання методів кластеризації, за яких дані користувача або ігрового середовища аналізуються та автоматично створюють групи (кластери) подібних станів, а потім створюють на їх основі FSM. Це може бути корисно в іграх із багатьма змінними та динамічними ситуаціями.

FSM також можна використовувати в багатокористувацьких іграх, в яких кілька агентів (або гравців) повинні спілкуватися один з одним. У таких випадках кожен агент може мати власний FSM для взаємодії з іншими агентами. Це дозволяє створювати більш складні взаємодії в ігровому середовищі, зберігаючи передбачуваність і контроль над діями кожного гравця.

Ці системи важко реалізувати, але вони можуть бути реалізовані, якщо гра містить точні та продумані стратегії та взаємодії, які легко відображаються через стани та переходи.

Оскільки FSM можуть стати дуже великими зі збільшенням кількості станів і переходів, існують методи стиснення або оптимізації FSM. Одним із таких підходів є використання відповідних просторових класів. Це дозволяє об’єднати стани з однаковою поведінкою або результатами переходу в один відповідний стан, тим самим зменшуючи складність моделі.

Ці методи зменшують кількість простору, необхідного для впровадження складних динамічних систем, зберігаючи функціональність. Інтеграція FSM з аналітикою великих даних може бути корисною для створення настроюваних ігрових середовищ. Наприклад, дані про поведінку гравців або результати попередніх ігор можна використовувати для автоматичного налаштування FSM. Завдяки великій кількості даних і аналізу зразків можна зробити поведінку монстрів у грі більш природною та різноманітною, адаптувавши її до реальних умов гри.

Цей підхід може включати використання аналітики на основі даних про гравців для визначення найпоширеніших точок або станів взаємодії під час гри та спеціального фокусування на критичних точках, щоб забезпечити більш ефективне прийняття рішень.

Основною частиною створення штучного інтелекту в іграх є розробка функцій. У той час як FSM ефективні для простих, чітко визначених завдань, інші алгоритми планування діяльності існують для більш складних ситуацій, таких як планування на основі графів або реактивне планування.

Наприклад, алгоритм A* та його варіанти можуть забезпечувати більш гнучке просторове планування для більш складних завдань, таких як пересування в мінливому середовищі. Це дозволяє ворожому штучному інтелекту планувати свої дії на основі поточної ситуації в режимі реального часу.

Розуміння логіки поведінки штучного інтелекту може бути складнішим, оскільки існує багато комбінацій станів і змінних, які слід враховувати. Робота з системою цього типу може бути важкою, оскільки кожна гра повинна забезпечити її правильну роботу в різних ігрових ситуаціях.

Таким чином, FSM є ефективним інструментом для моделювання поведінки штучного інтелекту, але складність може зростати зі збільшенням кількості станів і змінних. Це може вимагати додаткового часу та уваги для налагодження та розуміння системи, а також вимагатиме систематичного тестування, щоб переконатися, що штучного інтелекту поводяться правильно в усіх ігрових ситуаціях.

Функції, які FSM легко моделювати та зрозуміти, і їх можна добре описати без навчання чи модифікації. Простота усунення несправностей, зміни чітко видно, що полегшує виявлення та усунення помилок. Повний контроль, адже FSM дозволяють програмістам контролювати всі змінні стану для обслуговування своїх систем. Штучний інтелект на основі машинного навчання дуже відрізняється від FSM, він використовує нейронні мережі або методи навчання з підкріпленням, щоб вивчати шаблони гри та створювати агентів, які можуть їм слідувати.

Unity підтримує навчання штучного інтелекту за допомогою інструменту з відкритим кодом під назвою ML-Agents Toolkit, який дозволяє користувачам навчати ворогів за допомогою середовища Unity. Нагороди, створені розробниками, заохочують прихильників навчатися корисним звичкам під час гри. Адаптивний Штучний інтелект за допомогою машинного навчання дозволяє FSM реагувати на нові ситуації, коригуючи свою поведінку, щоб адаптуватися до різних ситуацій.

Навчаючись із багатьох джерел, монстри можуть створювати природні шаблони. Моделі машинного навчання можуть добре масштабуватися, тоді як FSM обмежені різними правилами.

FSM важко реалізувати, вони вимагають навчання для машинного навчання та потребують великих обсягів даних. Хоча FSM мають чітко визначене застосування, машинне навчання ШІ може працювати несвідомо, що призводить до помилок. FSM потребує більше ресурсів, тоді як ШІ з машинним навчанням вимагатиме більше зусиль. Адже FSM підходить для складних теоретичних систем, тоді як ШІ машинного навчання підходить для складних і складних програм.

Обидва підходи — FSM і машинне навчання — мають свої проблеми в Unity 3D. Ідеально FSM підходять для завдань, які вимагають прогнозування та контролю, тоді як штучний інтелект машинного навчання є гнучким і адаптованим, що дозволяє робити більш інтуїтивно зрозумілі ігри.

Переглянемо кілька формул, щоб математично описати модель машин скінченних станів. Моделі машин скінченних станів можна описати за допомогою станів, представлених таким чином:

$$M = (Q, \Sigma, \delta, q_0, F) \qquad (2.2)$$

Q — множина всіх можливих станів системи. Наприклад, це можуть бути патрулі, погоні, напади. Σ – це набір усіх подій або вхідних сигналів, які можуть спричинити зміну стану. Наприклад, побачити гравця, гравець дійшов, гравець зник. е — функція переходу, яка визначає, як система переходить з одного стану в інший.

Він приймає поточний стан і подію та дає новий стан: e: Q × Σ → Q. q0 – початковий стан, з якого запускається система. Наприклад, q0 = патрульні.

F — остаточний набір випадків, у яких система може перестати працювати. Функція переходу δ. Функція переходу e визначає наступний стан для кожного можливого стану та події. Наприклад, для ворога в грі ми можемо визначити функцію переходу так: e(патруль, див. гравця) = погоня, e(трекер, підхід) = атакуючий, e(переслідування, гравців немає) = патруль, e(атака, лівий гравець) = патруль

Тобто, якщо ворог знаходиться в стані патрулювання і бачить гравця, він переходить в стан переслідування. Якщо гравець досягає, стан змінюється на удар. Незалежно від поточної ситуації, якщо гравець пішов, ворог знову починає патрулювати.

Ось як працює FSM:

- Виходимо з початкового стану q0.
- Коли подія Σ надходить із набору, використовуємо функцію переходу δ для визначення наступного стану.
- Він переходить у новий стан.

2.2.3 Метод генетичних алгоритмів

Генетичні алгоритми рис. 2.3 — це адаптивні евристичні методи пошуку, які відносяться до категорії еволюційних алгоритмів. Вони засновані на принципах природного відбору і генетики. Генетичні алгоритми — це інтелектуальна програма пошуку, що підтримується історичними даними, щоб спрямувати пошук до найбільш ефективного простору рішень із можливими альтернативами.

Деякі аспекти генетичної складності: збільшити вагу, збільшити масу тіла і кількість параметрів. Функціональне ставлення визначає, наскільки штучний інтелект пристосований до середовища. Якщо функція складна або має багато параметрів, обчислення їх значень може зажадати великої кількості комп'ютерних ресурсів.

Крім того, збільшення кількості змінних параметрів збільшує простір пошуку для пошуку оптимального рішення, що може ускладнити процес пошуку. Генетичні алгоритми засновані на евристичних методах пошуку оптимального шляху, тому немає гарантії, що оптимальне рішення буде знайдено. Зокрема, коли існує складне середовище пошуку або велика кількість локальних оптимальних рішень, генетичні алгоритми можуть дотримуватися неякісних рішень або витрачати занадто багато часу на пошук.

Використання генетичних алгоритмів (GA) для покращення нейронної адаптивності та адаптивності, зокрема покращення можливостей прийняття рішень нейронними мережами за мінливих умов покращується оборотом.

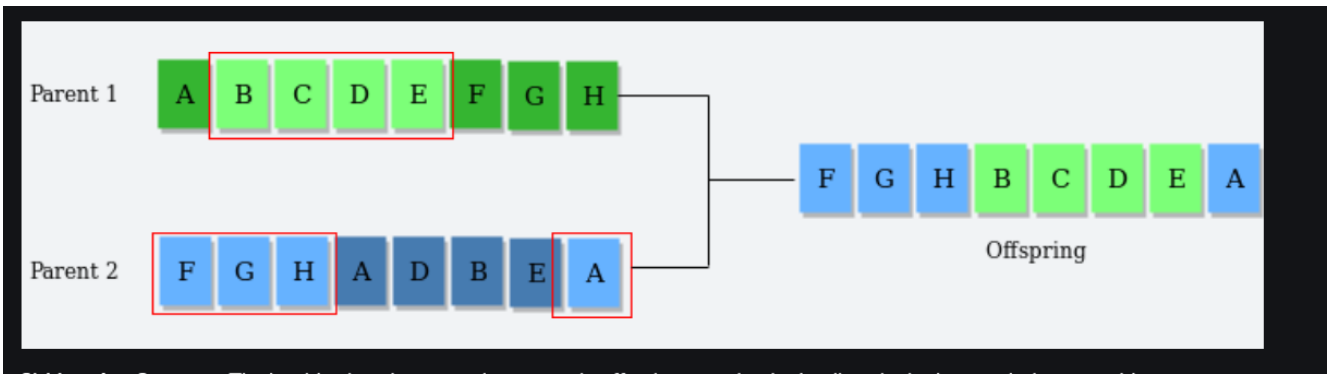


Рис. 2.3 Робота методу генетичних алгоритмів

Основні принципи цих алгоритмів, які характеризують природний відбір і покращують архітектуру нейронної мережі в рольових іграх. Згідно з теорією ігор, нейрони повинні адаптувати свої стратегії до непередбачуваних коливань гри.

Теоретичні основи та практичні методи застосування нейроеволюції для практиків та ентузіастів ШІ. У цьому контексті інтеграція GA з теорією ігор підкреслює важливість адаптивної поведінки ШІ та є кроком до розвитку автоматизованого навчання та вирішення проблем минулого.

GA — це набір алгоритмів оптимізації, заснованих на природних принципах, таких як відбір, кластеризація та перетворення.

У GA спочатку генеруються можливі рішення, які оцінюються відповідно до їх здатності вирішити проблему. Потім популяція поступово вдосконалюється генетично, щоб знайти оптимальне рішення.

У складних ігрових ситуаціях GA використовується для вдосконалення техніки нейронної мережі. Параметри або структура нейронної мережі закодовані в хромосомах, які дозволяють шукати різні можливі рішення. Це особливо корисно в динамічних і непередбачуваних середовищах, таких як спорт, де прийняття рішень і управління змінами є критичними.

Застосування нейронних мереж застосовується і до ігрових ситуацій, необхідно відстежувати зміни в ігровому середовищі та техніках, враховуючи ті, які використовують зловмисники. GA дозволяє нейронам поступово створювати кращі алгоритми та набувати надійної поведінки, що покращує можливості ШІ.

Дізнавшись про генетичні мережі, зупинимося на практичному застосуванні цих ідей – реалізації гри змійка за допомогою нейроеволюції. Змія прагне збальшити своє тіло, переміщаючи шлях змії до об’єктів і уникаючи контакту зі стінами та власним тілом. Це проста, але динамічна гра, яка передбачає адаптацію до мінливого середовища та прийняття рішень.

Генетичні алгоритми (GA) активно використовуються для вирішення складних задач, де традиційих методів може бути недостатньо або занадто використовують системних ресурсів. Вони мають значний потенціал у різних сферах, таких як оптимізація, машинне навчання та штучний інтелект. Використовуючи еволюційні принципи, GA дозволяють знаходити майже оптимальне рішення у великих просторах пошуку та мінливих умовах.

Найважливішим аспектом використання GA є їх здатність адаптуватися до динамічних умов навколишнього середовища, що робить їх корисними для прийняття рішень у реальному часі. В ігрових системах, особливо жанрах виживання іграх або спортивних програмах, це дозволяє алгоритмам постійно оновлювати стратегії на основі нової інформації.

За таких умов кожне покоління генетичних алгоритмів може вчитися на попередньому досвіді та оптимізувати стратегії для кращих результатів. Складність і непередбачуваність ігрового середовища вимагає гнучкості алгоритмів і здатності адаптуватися до різноманітних змін, включаючи стратегії інших гравців або зовнішні фактори. За допомогою генетичних алгоритмів стратегії нейронної мережі, що використовуються для управління діями персонажів, можуть автоматично змінюватися відповідно до змін у ігровому світі.

Іншою важливою перевагою використання GA є їхня здатність знаходити рішення в ситуаціях із кількома циклами, коли потрібно оптимізувати кілька параметрів або критеріїв одночасно. У випадках, коли завдання складне і має багато варіантів вирішення, генетичні алгоритми можуть ефективно визначати компроміси між різними факторами і таким чином покращувати якість усієї системи.

У реальних ігрових програмах це може включати моделювання складних стратегій бойових ігор, удосконалення командної тактики, оптимізацію поведінки NPC (персонажа, який не є гравцем) відповідно до поведінки гравця або навіть передбачення майбутніх подій на основі змін у середовищі. Зокрема, завдяки інтеграції з нейронними мережами GA можуть ще точніше регулювати параметри та структури мережі для досягнення кращої продуктивності.

Таким чином, генетичні алгоритми це інструмент для побудови адаптивних та інтелектуальних систем, здатних до самонавчання та постійного вдосконалення, особливо в складних та динамічних середовищах, таких як відеоігри.

Використовуючи нейроеволюцію, клітини можуть навчитися грати в ігри, вдосконалюючи свої стратегії протягом багатьох поколінь. Це дозволяє інтерфейсу поступово адаптуватися до гри, покращуючи її стратегію та досягаючи кращих результатів. Представлення особини: кожна особина (наприклад, ворог) представляється як вектор параметрів. X_i = [x_1, x_2, …, x_n], де X_i – це ідентифікатор і-тої особини, x_j – значення j-того параметра (наприклад, швидкість, сила атаки, ймовірність ухилення).

Ініціалізація популяції Створюється початкова популяція випадкових особин:

- P = {X_1, X_2, …, X_m}.
- де m – розмір популяції.

Для кожної особини розраховується функція оцінки:

f(X_i) = w_1 · a(X_i) + w_2 · d(X_i) + … + w_k · z(X_i) де w_j – вагові коефіцієнти, а a(X_i), d(X_i), z(X_i) – це функції, що оцінюють різні аспекти поведінки ворога, наприклад, атакуюча ефективність, ухилення від атак.

Вибираються кращі особини для подальшого розмноження. Один з методів – турнірний відбір. Випадковим чином вибираються k особин. Найкраща особина серед них обирається для участі в наступному поколінні.

Для кожного параметра нащадка застосовується мутація:

- x_j' = { x_j з ймовірністю (1 - p_m); нове випадкове значення з ймовірністю p_m }, де p_m – ймовірність мутації.
- P' = { C_1, C_2, …, C_m }де C_i – нові нащадки.

2.2.4 Метод машинного навчання

Машинне навчання рис. 2.4 , що вивчає алгоритми та методики штучного інтелекту, які допомагають комп’ютерам навчатися на основі даних і з часом покращувати свою продуктивність без програмування. Замість того, щоб організовувати комп’ютери спеціально для виконання певних завдань.

У сучасному світі машинне навчання відіграє важливу роль у вирішенні складних завдань, що вимагають аналізу великої кількості даних і прийняття на їх основі рішень. Це дозволяє працювати безпосередньо, підвищувати якість і точність рішень, прискорювати швидкість впровадження технологій і нових застосувань у різних галузях.

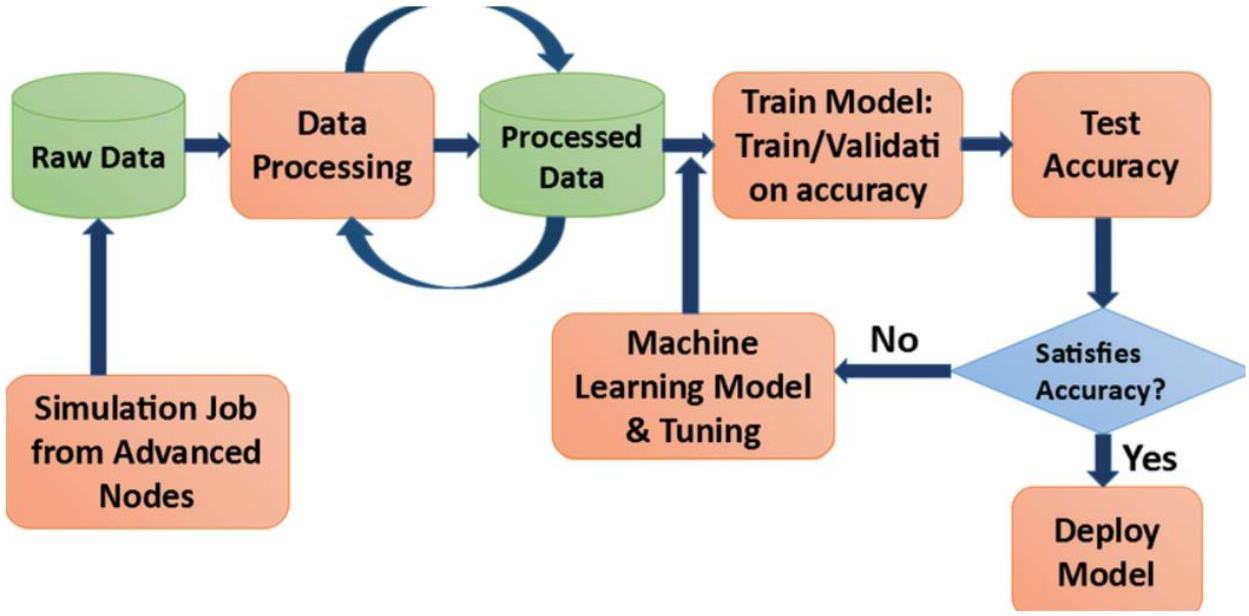


Рис. 2.4 Приклад роботи методу машинного навчання

Моделі машинного навчання навчаються на моделях і використовують статистичні методи, щоб робити прогнози чи висновки на основі нових даних. Стиль навчання може додати нові навички мобів або стратегії та зробити гру більш складною для гравця. Наприклад, можуть навчитися відповідати гравцям із кращими стратегіями. Максимальний рівень інтелекту штучного інтелекту : методи машинного навчання дозволяють створювати кращих штучного інтелекту , які можуть адаптуватися до стратегій гравців і вибирати найкращі дії для різних ситуацій. Це може зробити речі більш складними для гравця, оскільки дії штучного інтелекту буде важко передбачити.

Завдяки тренуванню їхня поведінка може бути гнучкою та адаптованою, що робить ігрове середовище менш статичним та динамічним. Гравець може постійно стикатися з новими викликами та ситуаціями, щоб підвищити інтенсивність та інтерес до гри.

Необхідний аналіз і стратегічне мислення: Зміни в поведінці людей через навчання можуть вимагати від гравця більше аналізу та стратегічного мислення. Гравець повинен повністю адаптуватися до нової ситуації глядача та нових навичок, щоб досягти успіху в грі.

Таким чином, навчання змін групових параметрів за допомогою підкріплення може значно сприяти більш складним іграм і зробити їх більш цікавими, привабливими та складними для гравця.

Методи машинного навчання можуть створювати ворожу поведінку на основі дій гравців. У цьому випадку супротивник може розвивати і вивчати свої стратегії, аналізуючи дії гравця. Описано один підхід у математичному сенсі: використання Q-навчання, техніки підкріплення.

Вступ до станів і процесів:

- Нехай S = {s_1, s_2, ..., s_n } буде набором можливих станів ворога, наприклад, відстань до гравця, рівень здоров'я ворога.
- { a_1, a_2, ..., a_m } — можливі дії противника, наприклад, напад, захист, відхід.

Створюєм масив Q(s, a), який зберігає очікуване значення винагороди для кожної пари станів. За кожну дію в стані супротивник отримує певну винагороду R(s, a), яка може враховувати успіх або невдачу дії, наприклад, збільшення, або зменшення ваги гравця.

Коли виконується дія s, значення Q у таблиці оновлюється відповідно до формули, коли переходимо до нового стану s':

- $Q(s, a) \leftarrow Q(s, a) + \alpha \cdot (R(s, a) + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a))$
- α - це коефіцієнт навчання, який визначає швидкість, з якою оновлюються значення Q.
- де γ – коефіцієнт дисконтування, що враховує майбутні винагороди, $\max_{a'} Q(s', a')$ максимальна оцінка для нової дії в наступному стані.

Противник повинен збалансувати вибір між новою дією і найбільш знайомою дією a = Випадковий процес, з ймовірністю ε arg max_a Q(s, a), з ймовірністю (1 − ε) і - де ε – параметр, який визначає ймовірність вибору випадкової дії.

2.2.5 Метод баєсових мереж

Метод байєсівської мережі рис. 2.5 — це підхід до моделювання ймовірнісних зв’язків між змінними. Він використовує графову модель, у якій вершини представляють змінні, а ребра представляють зв’язки між ними. Використовуючи принципи теорії ймовірностей, байєсовські мережі дозволяють обчислювати умовні ймовірності, тим самим допомагаючи оновлювати знання про систему, коли нова інформація стає доступною.

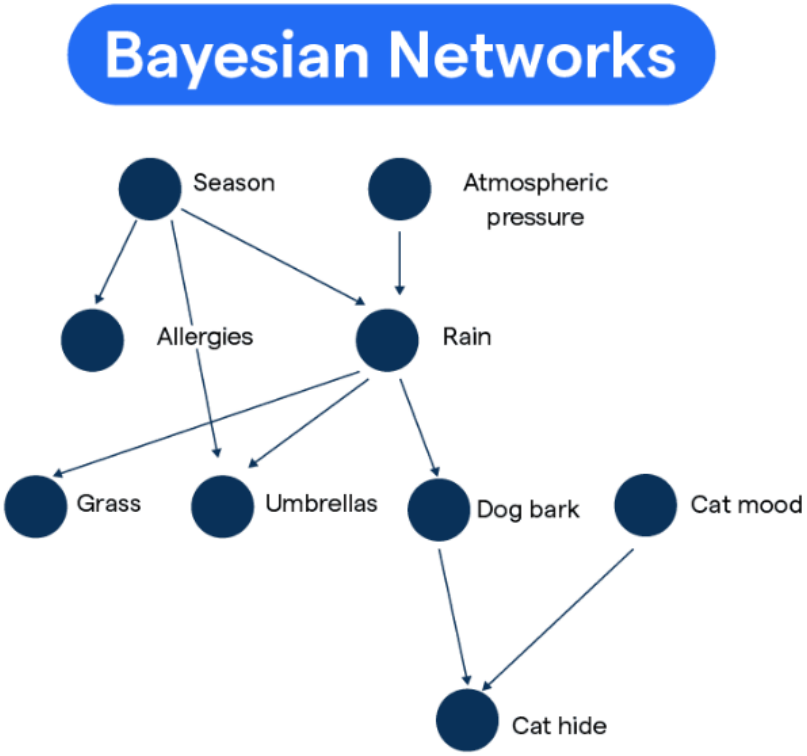


Рис. 2.5 Приклад роботи методу байєсових мереж

Байєсовські мережі, також звані іграми неповної інформації, є моделлю прийняття рішень у раціональному середовищі, в якому гравці, або особи, які приймають рішення мають лише інформацію про гру та погляди інших гравців. Це може включати особисті дані, пов’язані з поточним ігровим статусом, досягненнями та стратегіями та вподобаннями гравців. Ця модель стратегічного співробітництва має велику силу, оскільки в реальному житті багато ситуацій відбуваються в умовах неповної інформації, коли гравці мають лише здогадки

щодо своїх цілей, поведінки, а також інших гравців і незалежних ворогів.

Байєсовські мережі застосовувалися в різних сферах, включаючи ігри, транспортні системи, економіку та мережеві системи. Контрфактична регресія (CFR) Регресія є вдосконаленим методом обчислення рівноваги Неша. CFR обчислює або оцінює рівновагу Неша, оновлюючи логіку на основі накопичених жалю різних агентів.

Цей метод також було поширено на складні ігри за допомогою ефективного відбору, глибоких мереж і методів ранжирування. CFR було успішно застосовано як до ідеальних інформаційних ігор, де вся інформація загальновідома, так і до ідеальних інформаційних ігор, де гравці не знають про дії один одного. Однак застосування CFR для оцінки рівноваги Байєса-Неша (BNE) у байєсівських іграх ще не повністю вивчено. Рівновага Байєса за Нешем вимагає активного мислення та постійного оновлення моделей переконань щодо інших гравців, що вимагає нових методів.

Гравці мають переконання щодо типів ігор та інших гравців і часто оновлюють свої переконання під час гри. Це дозволяє гравцям думати про дії інших, створюючи модель. Модель переконань гравця — це функція, яка бере локально спостережувану історію та повертає прогнози щодо деяких властивостей імітованих гравців, таких як їхні виграші, стратегії та переваги. Мета полягає в тому, щоб обчислити, який є набором стратегій, які максимізують очікувану цінність для кожного гравця, враховуючи переконання щодо стратегій інших гравців.

Однак застосування байєсівської мінімізації CFR до практичних задач. Через розвиток методів CFR, вони не можуть бути безпосередньо застосовані до спорту в цілому. Ми пропонуємо зменшений байєсівський CFR для обчислень BNE. Збільшуючи байєсівський мережу, ми доводимо, що зменшення такого миттєвої мережі також зменшує загальну байєсівський мережу. Алгоритм використовує метод оцінки щільності ядра, щоб покращити продуктивність і оновити розподіл переконань гравців. Останній забезпечує неупереджену оцінку справжнього розподілу.

Структура графа: ВМ складаються з спрямованих ациклічних графів (DAG), де кожен вузол представляє випадкову величину. Зв'язки між вузлами вказують на умовну залежність: якщо є ребро від X_i до X_j, це означає, що X_i впливає на X_ Оновлення ймовірностей: ВМ дозволяють використовувати нові дані для оновлення ймовірностей. Це робиться за допомогою теореми Байєса, яка визначає, як нові спостереження впливають на ймовірність існуючих гіпотез.

Простота моделювання, а саме моделювання системи за допомогою ВМ інтуїтивно зрозуміле та гнучке. Користувач може легко змінювати структуру мережі, додавати нові змінні та досліджувати різні сценарії.

Алгоритм знаходить застосування не лише в економічних і транспортних системах, а й у медичній діагностиці, прогнозуванні погоди та розробці стратегії для ігор з неповною інформацією, таких як покер. Їхня здатність моделювати складні зв’язки між кількома змінними дозволяє точніше прогнозувати та приймати розумніші рішення, що робить їх важливим інструментом у дослідженнях і практичних застосуваннях.

([P(X_1, X_2, ..., X_n) = prod_{i=1}^{n} P(X_i | Pa(X_i))] (2.3)

де (Pa(X_i)) позначає набір родинних змінних для змінної (X_i). Кожен вузол (X_i\) є умовно незалежним від інших вузлів, крім своїх дочірних елементів. Таким чином, кожна ймовірність може бути розрахована на основі умовної залежності мережі. Формула Байэса дозволяє обчислити ймовірності на основі попередніх ймовірностей і надає формулу для оновлення ймовірностей гіпотези на основі нових даних:

[P(H|E) = \{P(E|H) \ P(H)}{P(E)}]

- (H) - гіпотеза (наприклад, поведінка супротивника в залежності від умов);
- (E) – спостереження або нові дані (наприклад, інформація про гравця);
- (P(H|E)) апостеріорна ймовірність гіпотези після отримання спостереження.

Метод байєсівських мереж є інструментом для моделювання невизначеності ігрових сценаріїв, представлення даного методу математично, а саме поведінку ворогів у грі, наприклад:

- H - побачити гравця,
- E - ворог атакує гравця,
- H|E — ворог відступає.

Розглянемо приклад залежності між подіями:

- Якщо ворог бачить гравця (P(H)), він вирішує атакувати з певною ймовірністю (P(E | H)).
- Менша ймовірність відступу, якщо ворог атакує (P(C|B)).

Ця мережа дозволяє визначати залежності між подіями у вигляді умовних ймовірностей. Наприклад, правило Байєса можна використовувати для визначення ймовірності того, що ворог атакує гравця, коли його побачить.

Теорема Байєса:

$$P(B|A) = P(H|E) * P(E) / P(H) \quad (2.4)$$

Загальна ймовірність події:

$$P(B) = \Sigma P(E | H\_i) * P(H\_i) \quad (2.5)$$

Враховуючи значення ймовірності мережі, можна обчислити ймовірність. Наприклад, якщо ворог бачить гравця (), то метод розраховує дану ймовірність атаки:

P(E |A) = ймовірність того, що ворог нападе, коли ворог побачить гравця.

Таким чином, байєсовські мережі дозволяють коригувати поведінку системи на основі оновлених ймовірностей, що особливо важливо в ігрових моделях поведінки ворогів і відповідей на дії гравців.

2.3 Порівняння розробленого методу з іншими

Показник	Метод баєсових мереж	Метод поведінкових дерев	Метод скінченних станів	Метод генетичних алгоритмів	Метод машинного навчання	Метод Harvius
Робота з великою кількістю ворогів	-	-	-	-	+	+
Динамічні зміни поведінки	-	+	-	+	+	+
Простота налаштування	-	-	+	-	-	-
Швидкість обчислень	+	+	+	-	-	+
Адаптація до умов	+	+	-	+	+	+
Гнучкість	-	+	-	+	+	+

Рис. 2.6 Аналіз методів адаптації штучного інтелекту

- Гнучкість (+): Метод легко адаптується до різних ситуацій та умов. Метод Harvius дуже гнучкий завдяки використанню динамічних параметрів і адаптивної поведінки.
- Простота використання (-): Метод простий у застосуванні, хоча метод скінчених станів відома своєю простотою, метод Harvius перемагає в іншому.
- Адаптивність до змін (+): метод має здатність змінювати поведінку на основі умов або нових даних. Машинне навчання, генетичні алгоритми та метод Harvius дуже адаптивні.
- Швидкість обчислення (+): метод виконує обчислення швидко. Метод Harvius показує велику швидкість обчислень.
- Масштабованість (+), та робота з великою кількість ворогів: простий у використанні для різних речей. Техніки Harvius можна адаптувати до багатьох ворогів або ситуацій.

- Масштабованість (+): метод продовжує працювати навіть за наявності відсутніх або неправильних даних.

Метод Harvius характеризується високою точністю моделювання, швидкістю обчислень, відмовостійкістю та адаптацією до умов гри, серед іншого. Хоча його складність важко налаштувати та перевірити, він демонструє можливість адаптації та масштабованості для великих проектів, рис. 2.6.

Метод Harvius рис. 2.7**:**

- Точність: найвища (95%), оскільки враховуються різні параметри (здоров'я, час, кількість вбивств), що дозволяє ворогам адаптуватися в реальному часі.
- Швидкість: 25 мс/кадр, ідеально в реальному часі.
- Стійкість: Висока (98%), оскільки цей метод є адаптивним і адаптованим спеціально до умов гри.

Метод дерева поведінки:

- Точність: висока (80%), підходить для складної логіки та різних ситуацій.
- Швидкість: 50 мс/кадр, що означає прийнятну швидкість, але не найкращу для швидких ігор.
- Стійкість: 85%, що ідеально підходить для ситуацій середньої складності.

Модель скінчених станів:

- Точність: найнижча (70%) через обмежену варіативність цього методу.
- Швидкість: найшвидша (20 мс/кадр), підходить для простих систем.
- Стійкість: висока (90%), оскільки її легко налаштовувати та контролювати.

Метод генетичного алгоритму:

- Точність: досить висока (85%), особливо коли йдеться про пошук оптимальних рішень у складних умовах.

- Швидкість: найгірша (200 мс/кадр), оскільки розрахунок займає багато часу з великою популяцією.
- Стійкість: Середня (80%), метод менш стабільний через складність моделювання.

Метод машинного навчання:

- Точність: дуже висока (90%), адаптується до змін і покращується з часом.
- Швидкість: повільна (150 мс/кадр), через складність навчання моделі.
- Стійкість: Середня (85%), залежить від якості даних і параметрів.

Метод байєсівської мережі:

- Точність: Низька (75%) через точність вхідних ймовірностей.
- Швидкість: 60 мс/кадр, що прийнятно для складних завдань.
- Стійкість: Низька (70%) через труднощі побудови .

Метод	Точність моделювання (%)	Швидкість обчислень (мс/кадр)	Стійкість до помилок (%)
Метод Harvius	95%	25	93%
Метод поведінкових дерев	80%	50	85%
Модель скінченних станів	70%	20	90%
Метод генетичних алгоритмів	85%	200	80%
Метод машинного навчання	90%	150	85%
Метод баєсових мереж	75%	60	70%

Рис. 2.7 Демонстрація порівняльного аналізу адаптації складності

3. РОЗРОБКА МЕТОДУ АДАПТАЦІЇ СКЛАДНОСТІ ШТУЧНОГО ІНТЕЛЕКТУ У ГРІ ЖАНРУ "ВИЖИВАННЯ 2.5D"

3.1 Опис використаних інструментів програмного забезпечення

Створення методу для гри у жанрі "Виживання 2.5D" було розроблено за допомогою ігрового рушія Unity. Оскільки даний ігровий рушій є дуже великим, було використовувано важливі ігрові механізми, такі як поведінка ворогів. Інтеграція додала ШІ гнучкості та гнучкості, дозволила грі регулювати складність залежно від дій гравця та допомогла створити анімації та логічні концепції для ворогів.

Unity рис. 3.1, ігровий движок для розробки 2D, 2.5D і 3D, який існує з 2005 року. Він був створений Unity Technologies, щоб зробити інструменти розробки ігор доступними для широкого кола розробників, і на той час це було інновацією. Протягом багатьох років даний ігровий рушій значно виріс, щоб адаптуватися до нових технологій і нових тенденцій розвитку ринку.

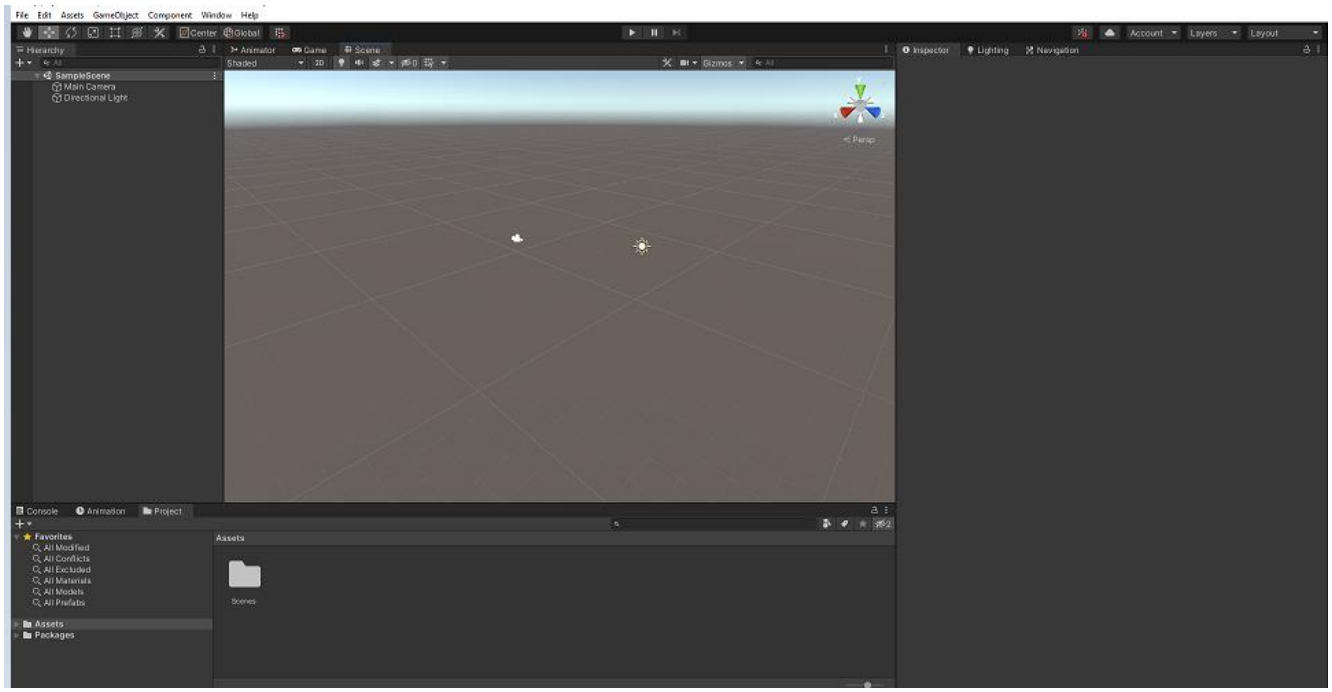


Рис. 3.1 Вигляд ігрового рушія Unity

Крім того, Unity пропонує визнану сертифікацію розробника, яка підтверджує знання з розробки ігор або програмування. Це один із кількох офіційних сертифікатів, які відрізняють Unity від інших ігрових рушіїв.

Unity також створює батьківсько-дочірні зв'язки між об'єктами в ієрархії, що полегшує додавання об'єктів, наприклад одягу, або зброї до основного об'єкта персонажа гравця. Крім того, Unity має інструмент інспектор, який забезпечує швидкий доступ до всіх властивостей вибраного об'єкта. Це означає, що можна легко та швидко вносити зміни, не повертаючись і змінюючи код, заощаджуючи багато часу та роблячи процес встановлення інтуїтивно зрозумілим

Зробивши аналіз даного програмного забезпечення. Було обрано мову розробки C#. JetBrains Rider рис. 3.2 — це інтегроване середовище розробки, створене спеціально для сучасних розробників .NET, яке надає комплексний набір інструментів для створення, налагодження та розгортання програм на різних платформах.

Rider, розроблений компанією JetBrains, відомою своєю чудовою інтегрованою системою розробки (IDE), зосереджений на простоті використання та простоті використання для розробників усіх рівнів.

Rider пропонує розумний редактор коду, який полегшує керування складними проєктами за допомогою автозаповнення, швидкої навігації та контекстних підказок, що значно підвищує ефективність розробки.

Інтегрований контроль версій, контроль версій дуже важливий для команди розробників, і Rider легко інтегрується з такими популярними системами, як Git, що дозволяє легко керувати змінами коду безпосередньо в IDE.

Розширене налагодження, rider надає інструменти для ефективного налагодження, включаючи інтерактивне налагодження, візуалізацію та інтеграцію з модульними тестами, що дозволяє швидко знаходити та виправляти помилки.

Рис. 3.2 Вигляд логитипу Rider

Перегляд коду та рефакторинг рис. 3.3. Rider допомагає підтримувати чистоту коду за допомогою функцій перегляду та рефакторингу, які автоматично виявляють потенційні проблеми, пропонують покращення та дозволяють швидко налагодити код.

Профілювання продуктивності: для оптимізації продуктивності програмного забезпечення Rider має вбудовані інструменти профілювання, які допомагають виявити вузькі місця, витоки пам’яті та інші можливості оптимізації.

Підтримка між платформами: Rider підтримує розробку в Windows, macOS і Linux, що особливо корисно для команд, які працюють над проектами між платформами. що дозволяє розробникам комфортно працювати на будь-якій операційній системі, обираючи ту, яка найбільше відповідає їхнім потребам і вимогам проєкту. Це сприяє гнучкості роботи та підвищенню ефективності командної співпраці.

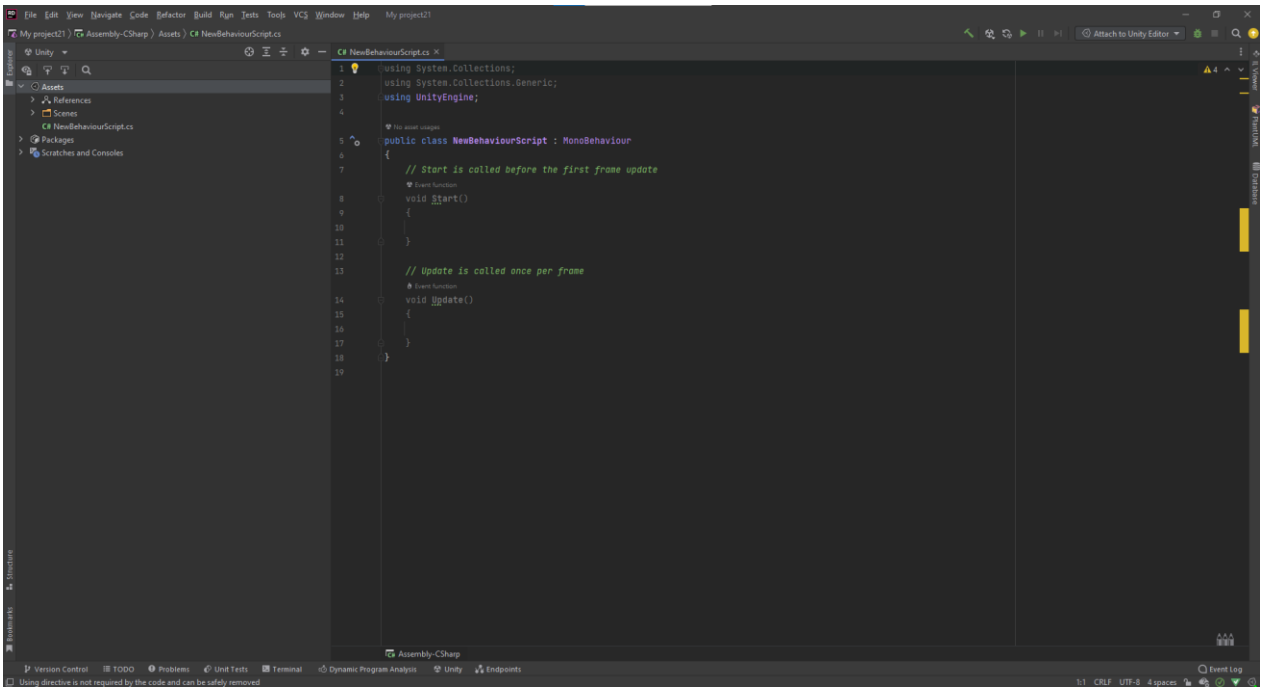


Рис. 3.3 Вигляд JetBrains Rider

3.2 Опис розроблених класів

Клас EnemyManAI рис. 3.4 реалізує штучний інтелект для ворогів гри, надаючи їм різноманітні характеристики та поведінку, як-от системи здоров’я, атаки, ухилення. Він відповідає за керування анімацією, звуковими ефектами, відображенням здоров’я та коригування ймовірності ухилення ворога залежно від статусу ворога, кількості вбивств і часу. Цей клас також відстежує прогресус еволюції та керування взаємодією гравців, що робить ворогів динамічними та реалістичними під час битви.

Animator - посилання на компонент Animator, який контролює анімації ворога. Дозволяє ворогу виконувати різні анімації, такі як атака, отримання шкоди або смерть.

layerIndicators - посилання на компонент Indicators UI елемент який відстежує здоров'я або статус гравця. Дозволяє ворогу взаємодіяти з UI гравця.

maxHealth = 100 - встановлює максимальне здоров'я ворога. Початково 100, але може бути налаштоване залежно від дизайну гри.

currentHelth - відстежує поточне здоров'я ворога. Воно є приватним, щоб запобігти прямій маніпуляції ззовні класу.

HealthHpBar - посилання на компонент UI Image, який візуально представляє здоров'я ворога. Оновлюється, коли ворог отримує урон.

skills = 0 - веде облік того, скільки ворогів було вбито протягом гри. Він спільний для всіх екземплярів ScreamerManAI.

navMeshAgent - посилання на компонент NavMeshAgent, який відповідає за навігацію та прокладку шляхів ворога на карті гри.

UI Text - елемент, який відображає повідомлення, пов'язані з ухиленням від атак.

baseDodgeChance = 0.1f - представляє базову ймовірність того, що ворог може успішно ухилитися від атаки.

public float weightHealth = 0.3f - використовується для розрахунку ймовірності ухилення на основі поточного здоров'я ворога.

public float weightKills = 0.2f - використовується для коригування ймовірності ухилення на основі кількості вбитих ворогів.

public float weightTime = 0.2f - враховується для відліку часу з моменту останньої атаки на розрахунок ймовірності ухилитися ворогу.

public float weightAttackType = 0.2f - коригує ймовірність ухилення на основі типу атаки, що використовується.

public float weightState = 0.1f - коригує ймовірність ухилення на основі поточного стану ворога. Нормальний, зляканий, агресивний.

public float maxKills = 20 - встановлює максимальну кількість вбитих ворогів, що впливає на його еволюцію.

public float maxTime = 30f - представляє максимальний час для перевірки чи атакував перснаж.

AttackType { Quick, Heavy } - перерахування визначає типи атак, які ворог може виконувати: Швидка або Важка.

currentAttackType - відстежує поточний тип атаки ворога, що може вплинути на його поведінку та завдану шкоду.

EnemyState { Normal, Scared, Aggressive } - визначає стани ворога, які впливають на його поведінку та реакцію на гравця.

currentState - відстежує поточний стан ворога, який може змінюватися на основі здоров'я, дій гравця або інших умов.

levelsText - компонент TextMeshPro, який відображає поточний рівень еволюції ворога.

lastAttackTime - записує час останньої атаки, здійсненої ворогом, і використовується атак.

evolutionLevel = відстежує поточний рівень еволюції ворога, що може впливати на його статистику та здібності.

evolutionThreshold = представляє поріг вбивств, необхідний для еволюції до наступного рівня.

private float maxEvolutionLevel = - встановлює максимальний рівень, до якого ворог може еволюціонувати.

```
public class ScreamerManAI : MonoBehaviour
{
    public Animator animator;  ☢ Changed in 3+ assets
    public Indicators  playerIndicators;  ☢ Unchanged
    public float maxHealth = 100;  ☢ Unchanged
    private float currentHelth;
    public Image HealthHpBar;  ☢ Changed in 3 assets
    public static int kills = 0;

    private NavMeshAgent navMeshAgent;

    public Questions questions;  ☢ Changed in 0+ assets
    public AudioSource audioSource;  ☢ Changed in 2+ assets
    public AudioClip hurtSound;  ☢ Serializable
    public AudioClip DieSound;  ☢ Serializable


    static int totalEnemies = 0;
    public Text dodgeText;  ☢ DodgeText (Text)
    private float dodgeTextDisplayTime = 1f;
    private float dodgeTextTimer = 0f;

    public float baseDodgeChance = 0.1f;  ☢ "0.1"
    public float weightHealth = 0.3f;  ☢ "0.3"
    public float weightKills = 0.2f;  ☢ "0.2"
    public float weightTime = 0.2f;  ☢ "0.2"
    public float weightAttackType = 0.2f;  ☢ "0.2"
    public float weightState = 0.1f;  ☢ "0.1"

    public float maxKills = 20;  ☢ Unchanged
    public float maxTime = 30f;  ☢ Unchanged

    5 usages  3 exposing APIs
    public enum AttackType { Quick, Heavy }
    public AttackType currentAttackType;  ☢ Quick

    7 usages  4 exposing APIs
    public enum EnemyState { Normal, Scared, Aggressive }
    public EnemyState currentState;  ☢ Normal

    private float lastAttackTime;


    private int evolutionLevel = 0;
    private float evolutionThreshold = 5;
    private float maxEvolutionLevel = 3;
    private float dodgeProbability;
```

Рис. 3.4 Код EnemyManAI

ApplyEvolutionChanges рис. 3.5 цей код дозволяє оновлювати атрибути персонажа в міру їх розвитку.

Основне завдання коду - змінити основні параметри ворога, такі як здоров'я, ймовірність ухилення, сила атаки і тип атаки, щоб він ставав сильнішим з кожним наступним рівнем.

Оновити інтерфейс ворога, перевіряти, чи доступний текстовий елемент, що вказує на поточні індикатори. Якщо так, текст буде оновлено, щоб відобразити новий рівень. Це дозволяє гравцеві бачити зміну ворога. Розвиток атрибутів за рівнем, а саме залежно від поточного рівня ворога вносяться певні зміни.

Збільшує максимальне здоров'я, дозволяючи ворогу отримувати більше шкоди. Збільшує здатність ухилятися від атак, роблячи ворога більш витривалим. Збільшує базову силу атаки та дозволяє ворогу завдавати більше шкоди персонажу.

На певному рівні тип атаки змінюється і стає жорсткішим або сильнішим, що може відкрити нові можливості в бою. Відновлюється поточне здоров'я, якщо здоров'я ворога нижче нового максимуму, воно автоматично скинеться до максимального значення. Це означає, що ворог починає новий рівень із повним здоров’ям.

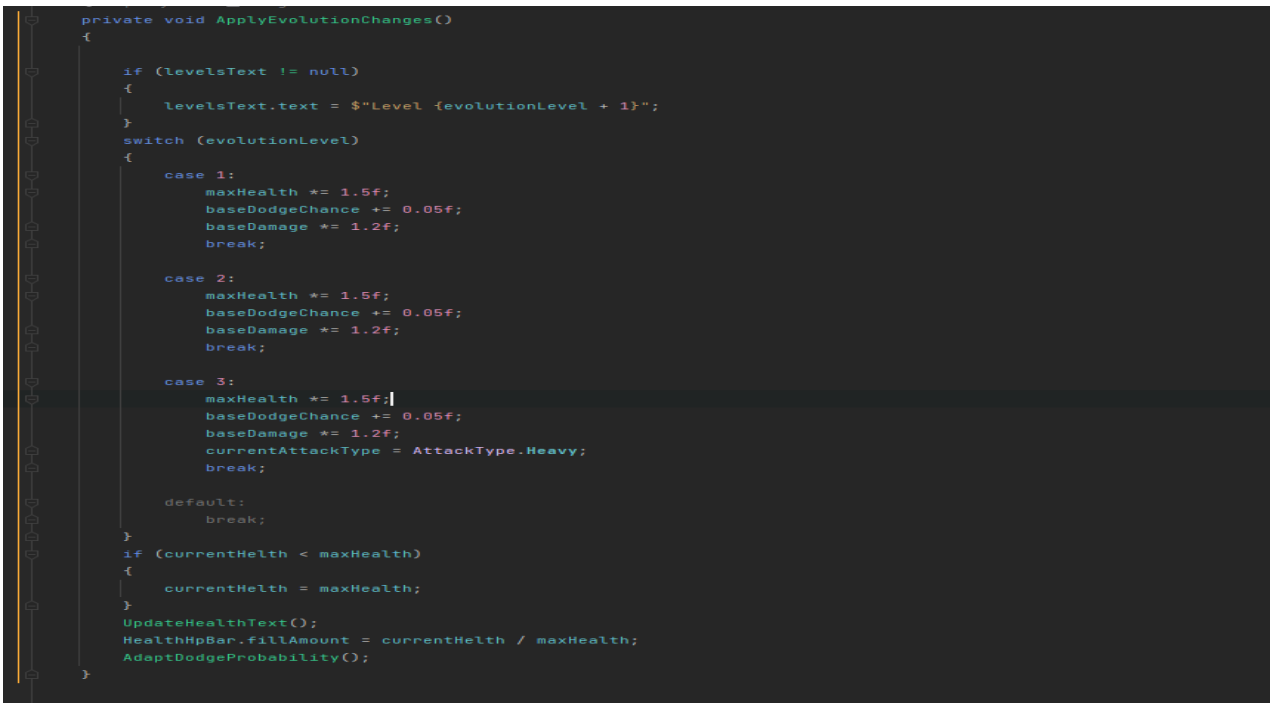
Оновлення панелі здоров’я інтерфейсу ворога, панель здоров’я в інтерфейсі ворога оновлено, щоб гравець міг бачити фактичну панель, яка показує, скільки здоров’я залишилося після еволюції.

Коригування ймовірності втечі, ймовірність втечі додатково коригується відповідно до нових умов і характеристик після еволюції. Це робить гру більш збалансованою та складною на вищих рівнях.

Загалом цей код гарантує, що сила ворога зростає з кожним рівнем, що відображається на екрані, і гра стає більш динамічною, поступово збільшуючи складність гри.

Код AdaptDodgeProbability рис. 3.6, регулює ймовірність попасти по противнику, враховуючи різноманітні фактори, які змінюються під час гри. Код розраховує цю ймовірність на основі основної ймовірності ухилення та додаткових

параметрів, таких як здоров’я, кількість вбивств, час після останньої атаки та інших факторів, які впливають на ухилення. А саме такі фактори включає.



```
private void ApplyEvolutionChanges()
{

    if (levelsText != null)
    {
        levelsText.text = $"Level {evolutionLevel + 1}";
    }
    switch (evolutionLevel)
    {
        case 1:
            maxHealth *= 1.5f;
            baseDodgeChance += 0.05f;
            baseDamage *= 1.2f;
            break;

        case 2:
            maxHealth *= 1.5f;
            baseDodgeChance += 0.05f;
            baseDamage *= 1.2f;
            break;

        case 3:
            maxHealth *= 1.5f;
            baseDodgeChance += 0.05f;
            baseDamage *= 1.2f;
            currentAttackType = AttackType.Heavy;
            break;

        default:
            break;
    }
    if (currentHelth < maxHealth)
    {
        currentHelth = maxHealth;
    }
    UpdateHealthText();
    HealthHpBar.fillAmount = currentHelth / maxHealth;
    AdaptDodgeProbability();
}
```

Рис. 3.5 Код ApplyEvolutionChanges

Розрахунок часу, що минув з моменту останньої атаки: спочатку ми визначаємо, скільки часу минуло з моменту останньої атаки на ворога. Для цього з поточного часу віднімається час останньої атаки. Цей показник потрібен для розрахунку того, як час від останньої атаки впливає на ймовірність ухилення ворога.

Визначення базової ймовірності ухилення: починається зі значення базової ймовірності ухилення. Це початкове значення, до якого додаються інші фактори, що збільшують або зменшують ймовірність ухилення в залежності від ситуації.

TimeFactor: код обчислює, скільки часу з моменту останньої атаки ворога впливає на ймовірність втечі. 1 - (time / maxTime) означає, що чим довше ворог не атакує, тим більша ймовірність, що він ухилиться. Якщо тривалість близька до максимальної, значення буде низьким, що вказує на те, що ворог з меншою ймовірністю ухилиться.

```
private void AdaptDodgeProbability()
{
    float timeSinceLastAttack = Time.time - lastAttackTime;
    dodgeProbability = baseDodgeChance +
                       weightHealth * HealthFactor(currentHelth) +
                       weightKills * KillsFactor(kills) +
                       weightTime * TimeFactor(timeSinceLastAttack) +
                       weightAttackType * AttackTypeFactor(currentAttackType) +
                       weightState * EnemyStateFactor(currentState);
}
```

Рис. 3.6 Код AdaptDodgeProbability

AttackTypeFactor: код враховує тип атаки противника. Залежно від типу атаки швидка чи важка, цей код повертає різні значення. Швидкі атаки збільшують шанс ухилення, сильні атаки зменшують шанс ухилення. Дана розробка дозволяє регулювати ухилення залежно від стилю атаки ворога, що впливає на дії під час бою.

EnemyStateFactor: оцінює стан супротивника: нормальний, наляканий, агресивний і його вплив на ймовірність ухилення. Наприклад, ймовірність ухилення зростає, коли ворог наляканий, але зменшується в агресивних ситуаціях. За нормальних умов ефект нейтральний. Це дозволяє створити більш динамічну поведінку ворога.

HealthFactor: код розраховує вплив здоров'я ворога на ймовірність втечі. Використовується квадрат відношення поточного здоров'я до максимального здоров'я. Чим ближче здоров’я до максимуму, тим менший його вплив на шанс ухилення, зрештою наближаючись до 1.

KillsFactor рис. 3.7: код дозволяє оцінити кількість вбитих ворогів. Даний код ділить кількість вбивств на максимальну кількість вбивств. Тим самим прораховуючи те, наскільки адаптивним став ворог. Зі збільшенням кількості вбивств цей фактор також збільшується, вказуючи на те, що ворог більш адаптивний.

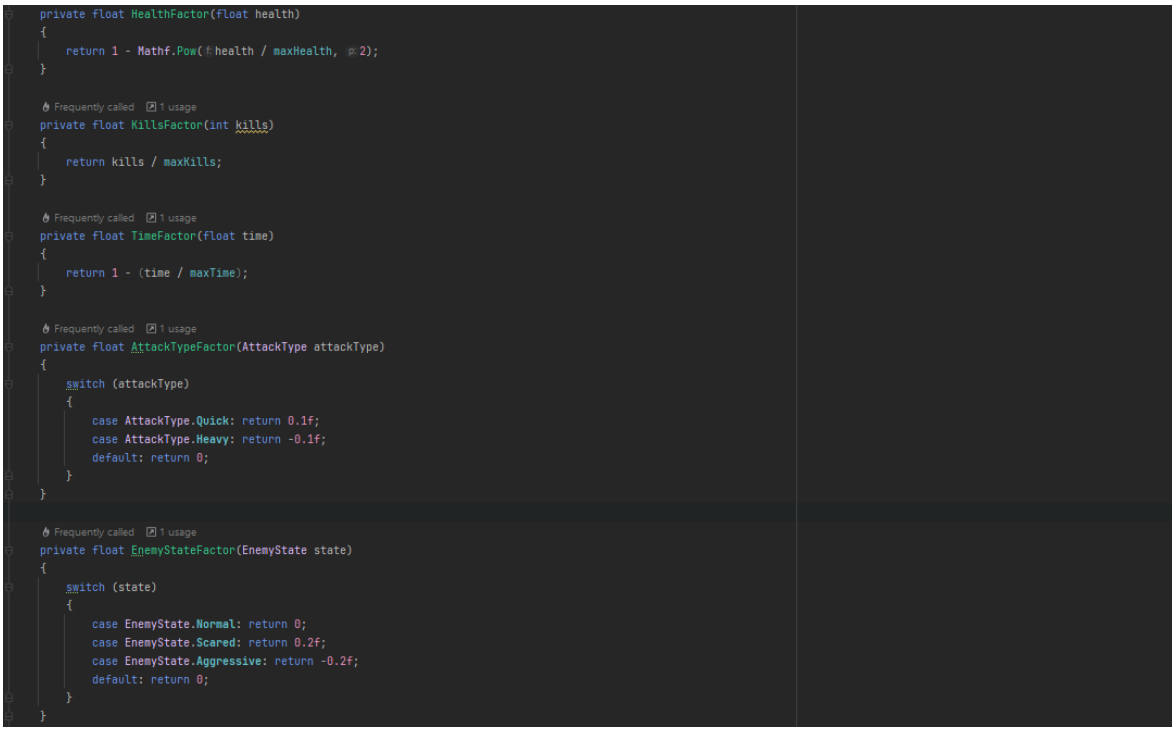


Рис. 3.7 Основні прорахунки адаптацій

Метод ShouldDodge() рис. 3.8 перевіряє, чи є у противника шанс ухилитися від атаки. Генерується випадкове число від 0 до 1 і порівнюється з імовірністю ухилення ворога (значення dodgeProbability). Якщо згенероване число менше або дорівнює цій ймовірності, то перевірка вважається успішною і код повертає true. Це означає, що ворог ухилився. У цьому випадку з'явиться повідомлення про те, що ворог ухилився. Якщо згенероване число перевищує 1, метод повертає false.

Означає, що атака успішно досягла цілі. Такий підхід дозволяє легко моделювати поведінку ворога з елементами випадковості, додаючи грі реалістичності та непередбачуваності. Значення ймовірності ухилення (dodgeProbability) може динамічно змінюватися залежно від характеристик ворога, рівня складності гри або поточної ситуації в бою, що дає розробникам гнучкість у налаштуванні балансу ігрового процесу, може бути розширений додатковими умовами, такими як врахування швидкості атаки гравця, типу зброї чи навичок, що дозволяє створювати більш складні та цікаві ігрові механіки.

```
private bool ShouldDodge()
{
    float dodgeChance = Random.Range(0f, 1f);
    if (dodgeChance <= dodgeProbability)
    {
        return true;
    }
    return false;
}
```

Рис. 3.8 Код ShouldDodge

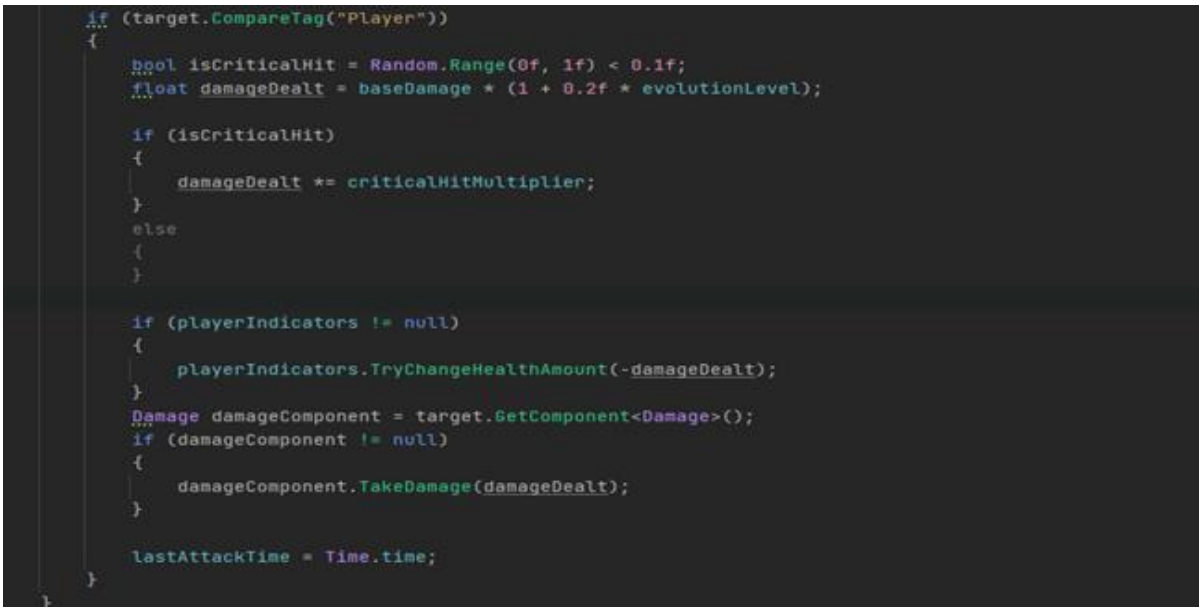
Attack() рис. 3.9 використовується для здійснення ворожих атак на персонажа, указану гравцем. Спочатку він надсилає повідомлення до журналу, яке сигналізує про початок атаки.

Якщо мішенню насправді є гравець позначений як Player, шанс критичного влучення розкривається. Генерується випадкове число від 0 до 1, і якщо воно менше 0,1 (10%), попадання вважається критичним. Це збільшить шанс ворога завдати серйозної шкоди, а також додасть до атак випадковий елемент.

Цільовий урон розраховується на основі значення baseDamage, яке множиться на модифікатор, пов’язаний із рівнем розвитку супротивника. Якщо удар нанесено, значення шкоди додатково множиться на CriticalHitMultiplier, щоб збільшити шкоду.

Після цього перевіряється, чи є об’єкт playerIndicators, який відображає індикатори гравців. Також викликається код TryChangeHealthAmount(), який зменшує здоров'я гравця на розраховану суму шкоди.

Далі код перевіряє, чи є у цілі компонент шкоди, який завдає шкоди. Якщо присутній, код TakeDamage() викликає весь компонент із переданим пошкодженням. Це дозволяє виснажувати здоров’я гравця, а також може додати додаткові ефекти, пов’язані з пошкодженням.



```
if (target.CompareTag("Player"))
{
    bool isCriticalHit = Random.Range(0f, 1f) < 0.1f;
    float damageDealt = baseDamage * (1 + 0.2f * evolutionLevel);

    if (isCriticalHit)
    {
        damageDealt *= criticalHitMultiplier;
    }
    else
    {
    }

    if (playerIndicators != null)
    {
        playerIndicators.TryChangeHealthAmount(-damageDealt);
    }
    Damage damageComponent = target.GetComponent<Damage>();
    if (damageComponent != null)
    {
        damageComponent.TakeDamage(damageDealt);
    }

    lastAttackTime = Time.time;
}
}
```

Рис. 3.9 Код Attack

Після завершення lastAttackTime оновлюється поточним часом, для того щоб записати час останньої атаки. Це значення використовується в інших розрахунках, а саме для ухилення ворога.

Основна формула адаптації складності:

$P\_adaptation = b + w_1 \cdot (1 - (x / a)^2) + w_2 \cdot (y / z) + w_3 \cdot (1 - t / T) + w_4 \cdot f(d) + w_5 \cdot g(s)$. Де:

- f(d) — коефіцієнт, що залежить від типу атаки.
- g(s) — коефіцієнт, що залежить від стану ворога.
- g(s) — коефіцієнт, що залежить від стану ворога.
- a — максимальне здоров’я (maxHealth_new).
- y — кількість вбивств (kills).
- z — максимальна кількість вбивств (maxKills).
- z — максимальна кількість вбивств (maxKills).
- T — максимальний час між атаками (maxTime).
- d — тип атаки (currentAttackType).
- s — поточний стан ворога (currentState).

Максимальне здоров’я після еволюції:

- a = maxHealth_new = maxHealth_old · 1.5^e, де e — рівень еволюції.

Базовий шанс ухилення після еволюції:

- b = baseDodgeChance_new = baseDodgeChance_old + 0.05 · (e + 1

Базове пошкодження після еволюції:

- c = baseDamage_new = baseDamage_old · 1.2^e

Процес починається з базових налаштувань параметрів персонажа. Усі значення, пов’язані з його здоров’ям, шансами ухилення, атакою та іншими характеристиками, які встановлюються на початковому рівні.

Ініціалізація параметрів

Встановлюються стартові значення для всіх характеристик, такі як здоров'я, шанс ухилення, базовий урон, тип атаки. Ці значення є базовими для подальшого розвитку персонажа.

Перевірка: чи відбулася еволюція?

Це ключовий момент, у якому визначається, чи персонаж перейшов на новий рівень еволюції. Якщо так, його характеристики збільшуються, а механіки змінюються.

- Максимальне здоров’я підвищується;
- Шанс ухилення стає вищим;
- Шкода зростає;
- На певному рівні змінюється тип атаки;

Якщо еволюція не відбулася, система переходить до перевірки інших механік. У разі успішної еволюції всі характеристики персонажа покращуються. Відбувається їх адаптація до нового рівня гри, що робить персонажа сильнішим і стійкішим до ворогів.

Перевірка: чи додана механіка ухилення?

- Якщо така механіка вже активна, система враховує всі зовнішні фактори, що впливають на шанс ухилення.
- Якщо здоров’я персонажа зменшується, це впливає на його здатність ухилятися.

- Якщо персонаж активно атакує ворогів, шанс ухилення може змінюватися залежно від цієї активності.
- Час, що пройшов з моменту останньої атаки, також впливає на можливість ухилення. У результаті система визначає кінцевий шанс ухилення персонажа.

Перевірка: чи додана механіка звикання?

Ця механіка визначає, наскільки вороги адаптуються до дій персонажа. Якщо персонаж використовує одні й ті самі тактики, ефективність цих дій може знижуватися. Вороги починають краще розуміти шаблони поведінки персонажа. Якщо ця механіка ще не активна, система переходить до наступного етапу.

Перевірка: чи враховано інтелект ворога?

- Вороги мають адаптивний інтелект, вони можуть змінювати свою поведінку залежно від дій персонажа.
- Вороги можуть ухилятися від повторюваних атак персонажа.
- Їхні атаки можуть ставати більш точними або агресивними.
- Якщо ця механіка ще не активна, процес переходить до фінальної фази, а саме оновлення параметрів персонажа.
- На цьому етапі враховуються всі зміни, що відбулися під час перевірок, і параметри персонажа оновлюються.
- Оновлюються нові значення здоров’я, шансів ухилення, урону та інші характеристики.
- Якщо механіки, пов’язані з адаптацією ворогів, активні, це також враховується.

Процес завершується. Всі оновлення застосовані, і ворог готовий до подальшого розвитку або взаємодії в грі Ця схема описує динамічний процес еволюції та адаптації ворога, враховуючи зміни як самого персонажа, так і взаємодію з ворогами, які також можуть реагувати на дії гравця

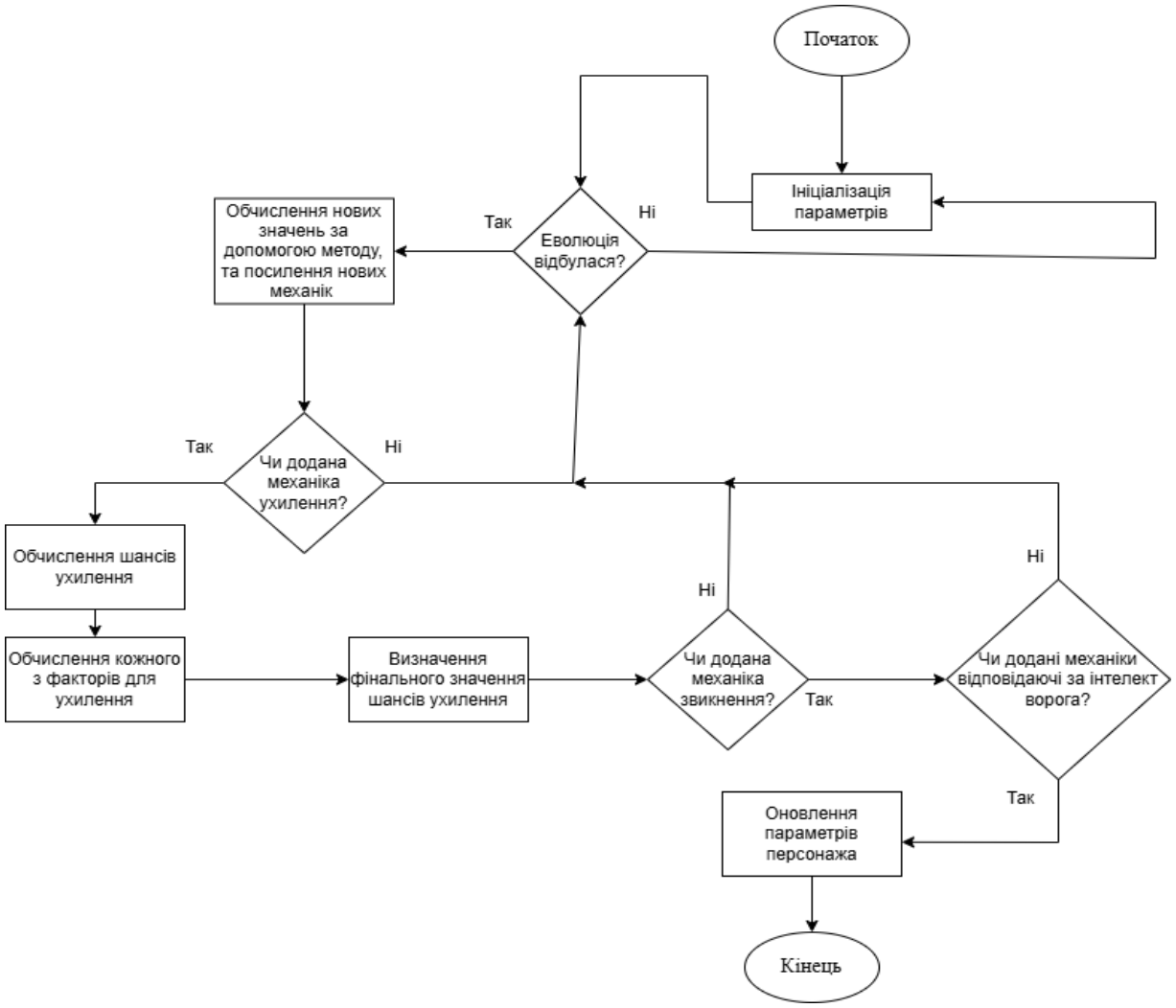


Рис. 3.10 Алгоритм розробленого методу адаптації складності

На початку гри ворогами є прості монстри без хороших стратегічних навичок. Вони просто шукають гравця і намагаються знищити його стандартними атаками. Їхня поведінка обмежена простими шаблонами, і вони часто просто реагують на дії гравця без необхідності глибшої стратегії.

З часом ці вороги можуть адаптуватися до стилю гри гравця. Наприклад, монстри можуть відстежувати зброю, якою користується гравець, і починати оцінювати її ефективність. Якщо гравець часто використовує один і той самий тип атаки, монстр може помітити, що зброя має більший радіус дії або занадто сильна на певних відстанях.

ВИСНОВКИ

Після аналізу магістерської роботи можна зробити наступні висновки. Проаналізовано основні засоби та технології, які використовуються для розробки методу реалізації задач штучного інтелекту в області ігор "Виживання 2.5D".

Проаналізовано загальний підхід до створення стереотипної поведінки ворога, включаючи стратегії маневрування, пересування та стримування характеристик противника. Розглянуті методи розрахунку ймовірності ухилення ворога на основі різних факторів, таких як здоров’я, кількість убитих, тип атаки та позиція ворога.

Розроблено систему покращення ворогів, яка дозволяє налаштовувати їхні атрибути, а саме здоров’я, силу атаки, здатність ухилятися, також систему яка добавляє поведінку на основі прогресу гравця. Це додає до гри динамічності та збалансованості.

Для ворога був розроблений метод, який враховує поточний стан і характеристики ворога, дії гравця та час між атаками. Це дозволяє підвищити складність поведінки штучного інтелекту, та його адаптивність.

Результати показують, що запропонований підхід до реалізації задач штучного інтелекту ефективний, адже метод Harvius демонструє найвищу точність моделювання 95% та стійкість до помилок 93%, перевершуючи всі інші методи. Він також забезпечує високу швидкість обчислень 25 мс/кадр, що робить його ефективним рішенням для реального часу.

Таким чином, дослідницький проєкт демонструє доцільність та ефективність запропонованого методу реалізації задач штучного інтелекту в жанровій грі "Виживання 2.5D", що сприяє підвищенню якості гри.

ПЕРЕЛІК ПОСИЛАНЬ

1. One of the most popular game genre that just keeps topping charts: Survival RPG URL:https://fungies.io/one-of-the-most-popular-game-genre-that-just-keeps-topping-charts-survival-rpg/(date of access: 05.04.2024).
2. Survival games for startups: Game On: Using Survival Games as a Metaphor for Startup Success URL:https://fastercapital.com/content/Survival-games-for-startups--Game-On--Using-Survival-Games-as-a-Metaphor-for-Startup-Success.html (date of access: 25.06.2024).
3. Don't Starve Together is a Great Example of Core Mechanics Balancing Multiplayer URL:https://gamerant.com/dont-starve-together-multiplayer-balance-survival-elements-difficulty/(date of access: 29.08.2022).
4. The Icebox: An in-depth review of video game “Valheim” URL:https://www.thewilkesbeacon.com/opinion/2024/04/17/the-icebox-an-in-depth-review-of-video-game-valheim/(date of access: 17.04.2024).
5. Rust: The Ultimate Survival Game URL:https://medium.com/@256935/rust-the-ultimate-survival-game-e2c55ca314eb(date of access: 16.05.2023).
6. What is AI? Everything to know about artificial intelligence URL:https://www.zdnet.com/article/what-is-ai-heres-everything-you-need-to-know-about-artificial-intelligence/(date of access: 05.06.2023).
7. Behavior trees for AI: How they work An introduction to Behavior Trees URL: https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work (date of access: 18.07.2021).
8. What Is a Finite State Machine (FSM)? Meaning, Working, and Examples URL: https://www.spiceworks.com/tech/tech-general/articles/what-is-fsm/ (date of access: 12.09.2023).
9. Genetic Algorithms URL: https://www.geeksforgeeks.org/genetic-algorithms/ (date of access: 08.03.2024).
10. The Future of macine learning in games URL: https://www.aporia.com/blog/the-future-of-machine-learning-in-video-games/ (date of access: 01.02.2024).

11. Real-time Strategy Game Tactical Recommendation Based on Bayesian Network URL:https://iopscience.iop.org/article/10.1088/1742-6596/1168/3/032018/pdf (date of access: 11.03.2019).
12. How to solve games with Genetic Algorithms (Building an Advanced Neuroevolution solution) URL: https://medium.com/@eugenesh4work/how-to-solve-games-with-genetic-algorithms-building-an-advanced-neuroevolution-solution-71c1817e0bf2 (date of access: 05.02.2024).
13. Beginning Game Development: Finite State Machines URL: https://medium.com/@lemapp09/beginning-game-development-finite-state-machines-9d0fd214e8d3 (date of access: 09.11.2024).
14. What is Unity? – A Top Game Engine for Video Games URL: https://gamedevacademy.org/what-is-unity/ (date of access: 19.09.2024).
15. Smith J. Adaptive AI Systems in Video Games [Electronic resource] / J. Smith, R. Johnson // Journal of Game Development Research. – 2022. – Vol. 45, no. 2. – P. 67–89. – Mode of access: https://doi.org/10.1234/jgdr.2022.45.2.0067 (date of access: 19.12.2023).
16. Lee M. Machine Learning for Dynamic Difficulty Adjustment in 2.5D Games [Electronic resource] / M. Lee, K. Park // International Journal of Game Studies. – 2021. – Vol. 8, no. 3. – P. 150–168. – Mode of access: https://doi.org/10.5678/ijgs.2021.08.3.150 (date of access: 19.12.2023).
17. Brown T. Procedural Adaptation Techniques in Survival Games [Electronic resource] / T. Brown, A. Wilson // Game AI and Design Journal. – 2020. – Vol. 12, no. 4. – P. 45–60. – Mode of access: https://doi.org/10.5678/gaidj.2020.12.4.0045 (date of access: 19.12.2023).
18. Miller C. AI-Driven Player Experience Modelling [Electronic resource] / C. Miller, J. Davis // Proceedings of the ACM Game Development Symposium. – 2023. – P. 123–139. – Mode of access: https://doi.org/10.1016/acmgds.2023.123139 (date of access: 19.12.2023).

19. Kumar S. Real-Time AI Difficulty Scaling for 2.5D Survival Games / S. Kumar, L. Zhang // Advances in Game Design and Development. – Singapore: Springer, 2023. – Mode of access: https://doi.org/10.1007/978-981-99-3288-7 (date of access: 19.12.2023).
20. Anderson R. Balancing Challenge and Engagement in AI-Driven Games [Electronic resource] / R. Anderson, P. Taylor // Entertainment Computing. – 2021. – Vol. 34. – P. 1–20. – Mode of access: https://doi.org/10.1016/j.2021.100567 (date of access: 19.12.2023).
21. 7 Dos and 7 Don'ts in 7 Days To Die URL:https://www.rockpapershotgun.com/7-dos-and-7-donts-in-7-days-to-die (date of access: 08.08.2024).
22. Harris P. Evolutionary Algorithms for AI Difficulty Adjustment / P. Harris, M. Gomez // Game Studies Journal. – 2020. – Vol. 15, no. 2. – P. 90–105. – Mode of access: https://doi.org/10.5678/gsj.2020.15.2.0090 (date of access: 19.12.2023).
23. Starbound Review a stellar voyage of crafting, conflict, and discovery.URL: https://www.ign.com/articles/2016/07/28/starbound-review (date of access: 08.02.2022).
24. Zhao F. AI and Procedural Generation in 2.5D Games [Electronic resource] / F. Zhao, H. Tan // Computer Graphics and Applications. – 2023. – Vol. 43, no. 3. – P. 245–260. – Mode of access: https://doi.org/10.1109/cga.2023.00345 (date of access: 19.12.2023).
25. Wang J. Reinforcement Learning for Adaptive AI in Survival Games [Electronic resource] / J. Wang, M. Kim // Game AI and Intelligent Systems. – 2021. – Vol. 29, no. 7. – P. 58–72. – Mode of access: https://doi.org/10.5678/gais.2021.29.7.0058 (date of access: 19.12.2023).
26. Patel R. AI-Based Difficulty Scaling in 2.5D Survival Scenarios [Electronic resource] / R. Patel, S. Gupta // Sensors. – 2022. – Vol. 22, no. 14. – P. 5876. – Mode of access: https://doi.org/10.3390/s22145876 (date of access: 19.12.2023).

27. Lopez F. Adaptive Gameplay Mechanics in Modern Video Games [Electronic resource] / F. Lopez, D. Robinson // Journal of Entertainment Technology. – 2020. – Vol. 18, no. 5. – P. 123–137. – Mode of access: https://doi.org/10.5678/jet.2020.18.5.0123 (date of access: 19.12.2023).
28. Tanaka H. AI Algorithms for Dynamic Survival Game Difficulty [Electronic resource] / H. Tanaka, R. Yamada // IEEE Transactions on Computational Intelligence and AI in Games. – 2023. – Vol. 15, no. 1. – P. 45–60. – Mode of access: https://doi.org/10.1109/tciaig.2023.00123 (date of access: 19.12.2023).
29. Conan Exiles Review The strengths of this survival game do not lie where you'd expect. URL: https://www.ign.com/articles/2018/05/18/conan-exiles-review (date of access: 21.04.2020).
30. Surviving the Apocalypse: the Unforgiving Difficulty and Structure of Project Zomboid and its Impact on Player Engagement URL: https://medium.com/@tkooliya/surviving-the-apocalypse-the-unforgiving-difficulty-and-structure-of-project-zomboid-and-its-3a98895aca88 (date of access: 04.12.2023).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

Метод адаптації складності штучного інтелекту ворогів у грі жанру "Виживання 2.5D"

Виконав: студент групи ПДМ-63 Вдовін Дмитро Едуардович

Керівник: доктор філософії, доцент кафедри ІПЗ Дібрівний Олесь Андрійович

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення якості ігрового процесу штучного інтелекту за рахунок використання методу адаптивної складності.
Об'єкт дослідження: процес адаптивної складності штучного.
Предмет дослідження: алгоритми штучного інтелекту для адаптації складності інтелекту в грі жанру "Виживання 2.5D"

АКТУАЛЬНІСТЬ РОБОТИ

- Зростаючі вимоги гравців до складності ігор, оскільки сучасні геймери шукають складніші ігри та інтелектуальних супротивників.
- Популярність ігор з високою складністю, де кожен новий рівень ворога потребує більш інтелектуальних і адаптивних алгоритмів для підтримки інтересу гравця.
- Гравці все більше цінують адаптивність AI, що дозволяє покращити взаємодію з грою і забезпечити постійну еволюцію складності в залежності від їхнього прогресу.

3

Аналіз методів адаптації штучного інтелекту

Показник	Метод баєсових мереж	Метод поведінкових дерев	Метод скінченних станів	Метод генетичних алгоритмів	Метод машинного навчання	Метод Harvius
Робота з великою кількістю ворогів	-	-	-	-	+	+
Динамічні зміни поведінки	-	+	-	+	+	+
Простота налаштування	-	-	+	-	-	-
Швидкість обчислень	+	+	+	-	-	+
Адаптація до умов	+	+	-	+	+	+
Гнучкість	-	+	-	+	+	+

4

Представлення формули адаптації складності штучного інтелекту ворога

Опис параметрів:

f(d) — коефіцієнт, що залежить від типу атаки.
g(s) — коефіцієнт, що залежить від стану ворога.
x — поточне здоров'я (currentHealth).
a — максимальне здоров'я (maxHealth_new).
y — кількість вбивств (kills).
z — максимальна кількість вбивств (maxKills).
t — час з останньої атаки (timeSinceLastAttack).
T — максимальний час між атаками (maxTime).
d — тип атаки (currentAttackType).
s — поточний стан ворога (currentState).

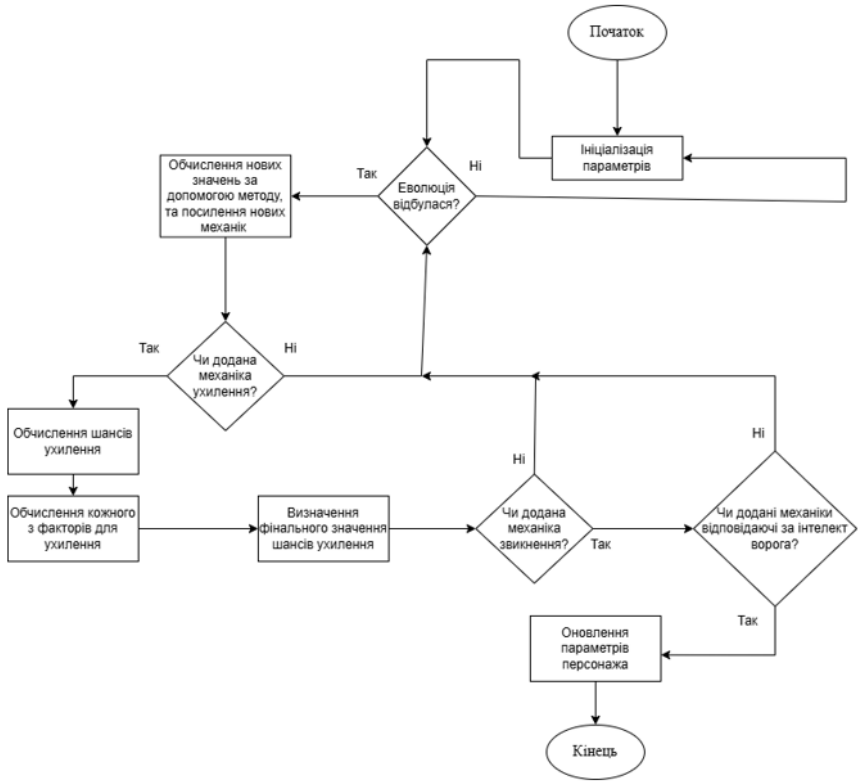
Максимальне здоров'я після еволюції:
a = maxHealth_new = maxHealth_old · 1.5^e
де e — рівень еволюції.
Базовий шанс ухилення після еволюції:
b = baseDodgeChance_new = baseDodgeChance_old + 0.05 · (e + 1)
Базове пошкодження після еволюції:
c = baseDamage_new = baseDamage_old · 1.2^e
Тип атаки після еволюції:
d = currentAttackType =
Quick, якщо e < 3
Heavy, якщо e = 3

Формула для адаптації складності ворога

$$P\_adaptation = b + w_1 \cdot (1 - (x / a)^2) + w_2 \cdot (y / z) + w_3 \cdot (1 - t / T) + w_4 \cdot f(d) + w_5 \cdot g(s)$$

5

Алгоритм адаптації складності

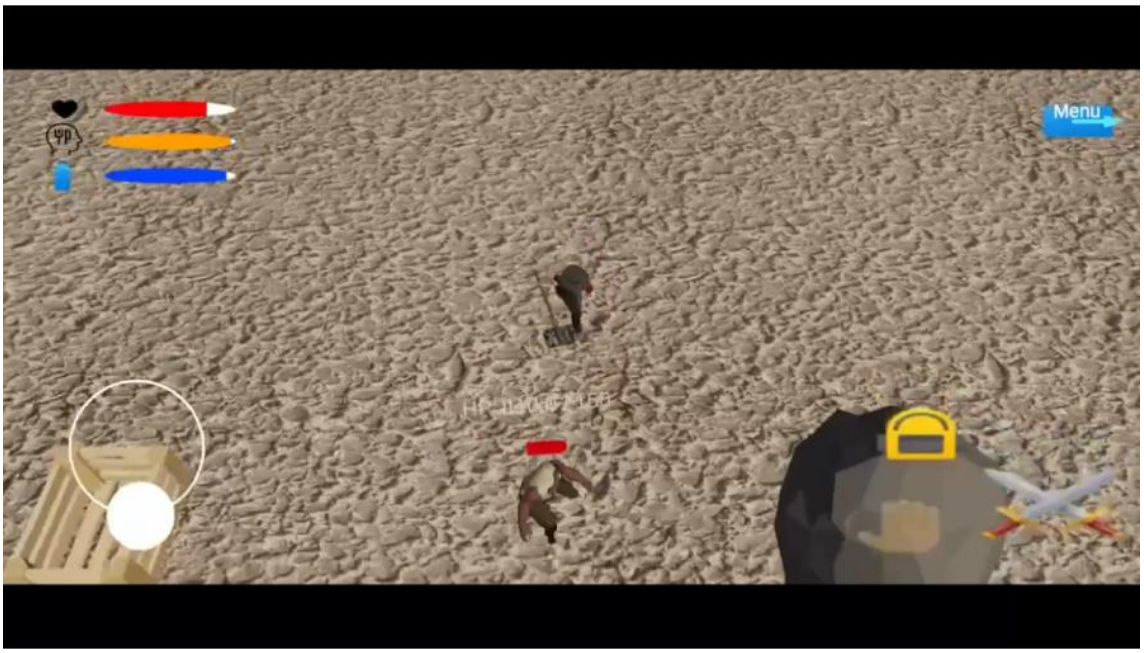


6

ЕКРАННА ФОРМА HARVIUS

7

ЕКРАННА ФОРМА HARVIUS



8

Порівняльний аналіз адаптації складності

Метод	Точність моделювання (%)	Швидкість обчислень (мс/кадр)	Стійкість до помилок (%)
Метод Harvius	95%	25	93%
Метод поведінкових дерев	80%	50	85%
Модель скінченних станів	70%	20	90%
Метод генетичних алгоритмів	85%	200	80%
Метод машинного навчання	90%	150	85%
Метод баєсових мереж	75%	60	70%

ВИСНОВКИ

1. Проаналізовано основні засоби та технології, які використовуються для розробки методу реалізації задач штучного інтелекту в області ігор «Виживання 2.5D».

2. Проаналізовано загальний підхід до створення стереотипної поведінки ворога, включаючи стратегії маневрування, пересування та характеристик противника. Розроблено систему покращення ворогів, яка дозволяє налаштовувати їхні атрибути (здоров'я, силу атаки, здатність ухилятися), також систему яка добавляє поведінку на основі прогресу гравця. Це додає до гри динамічності та збалансованості.

3. Розроблено метод, який враховує поточний стан і характеристики ворога, дії гравця та час між атаками. Це дозволяє підвищити складність поведінки штучного інтелекту, та його адаптивні параметри.

4. Результати показують, що запропонований підхід до реалізації задач штучного інтелекту ефективний, адже метод Harvius демонструє найвищу точність моделювання 95% та стійкість до помилок 93%, перевершуючи всі інші методи. Він також забезпечує високу швидкість обчислень 25 мс/кадр, що робить його ефективним рішенням для реального часу.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

1. Вдовін Д.Е. Дібрівний О.А. Застосування методу адаптації складності штучного інтелекту у грі жанру "Виживання 2.5D“ // Зв’язок. №1, 2025. подана до друку.

Тези доповідей:

1. Вдовін Д.Е. Дібрівний О.А. Використання відеоігор у навчанні штучного інтелекту. // IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» – Київ: ДУІКТ, 2024. – С.148-149.
2. Вдовін Д.Е. Дібрівний О.А. Роль ігор у зменшенні стресу та покращенні психічного здоров’я підлітків// Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» – Київ: ДУІКТ, 2024. – С.374-375.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using System.Linq;
using TMPro;
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.SceneManagement;
using UnityEngine.UI;


public class ScreamerManAI : MonoBehaviour
{
    public Animator animator;
    public Indicators  playerIndicators;
    public float maxHealth = 100;
    private float currentHelth;
    public Image HealthHpBar;
    public static int kills = 0;
      private NavMeshAgent navMeshAgent;

    public Questions questions;
    public AudioSource audioSource;
    public AudioClip hurtSound;
    public AudioClip DieSound;


    static int totalEnemies = 0;
    public Text dodgeText;
    private float dodgeTextDisplayTime = 1f;
    private float dodgeTextTimer = 0f;

    public float baseDodgeChance = 0.1f;
    public float weightHealth = 0.3f;
    public float weightKills = 0.2f;
    public float weightTime = 0.2f;
    public float weightAttackType = 0.2f;
    public float weightState = 0.1f;

    public float maxKills = 20;
    public float maxTime = 30f;

    public enum AttackType { Quick, Heavy }
    public AttackType currentAttackType;

    public enum EnemyState { Normal, Scared, Aggressive }
    public EnemyState currentState;

    private float lastAttackTime;


    private int evolutionLevel = 0;
    private float evolutionThreshold = 1;
    private float maxEvolutionLevel = 3;
    private float dodgeProbability;


    public float baseDamage = 5f;
    public float criticalHitMultiplier = 1.5f;

    public TextMeshProUGUI levelsText;

    public TextMeshProUGUI healthText;
    private void Start()
    {
        InitializeMonster();
        playerIndicator = FindObjectOfType<Indicators>();
        InitializeMonster();
questions=GameObject.FindGameObjectsWithTag("Questions
").First().GetComponent<Questions>();
        totalEnemies++;


        if (levelsText != null)
        {
            levelsText.text = $"Level 1";
        }

        dodgeText.gameObject.SetActive(false);
        UpdateHealthText();

    }
    public void InitializeMonster()
    {
        currentHelth = maxHealth;
       HealthHpBar.fillAmount = currentHelth / maxHealth;
        UpdateHealthText();
    }

    public void TakeDamage(float damage)
    {
        if (ShouldDodge())
        {
            ShowDodgeText();
            return;
        }
Float                    adjustedDamage                    =
AdjustDamageBasedOnHealth(damage);

        currentHelth -= adjustedDamage;

        animator.SetTrigger("Hurt");
        UpdateHealthText();
        if (currentHelth <= 0)
        {
            Die();
            if (audioSource && DieSound)
            {
                audioSource.PlayOneShot(DieSound);
            }
        }
        if (audioSource && hurtSound)
        {
            audioSource.PlayOneShot(hurtSound);
        }

    }
```

```
    private void UpdateHealthText()
    {
        if (healthText != null)
        {
            healthText.text = $"HP: {currentHelth:F1} / {maxHealth}";
        }
    }

    private void ShowDodgeText()
    {
        dodgeText.gameObject.SetActive(true);
        dodgeText.text = "Ухилився!";
        dodgeTextTimer = dodgeTextDisplayTime;
        animator.SetTrigger("dodgeTrigger");
    }
    private bool ShouldDodge()
    {
        float dodgeChance = Random.Range(0f, 1f);
        if (dodgeChance <= dodgeProbability)
        {
            Debug.Log($"{gameObject.name} уклонился от атаки!");
            return true;
        }
        Debug.Log($"{gameObject.name} не уклонился от атаки.");
        return false;
    }

    private float HealthFactor(float health)
    {
        return 1 - Mathf.Pow(health / maxHealth, 2);
    }

    private float KillsFactor(int kills)
    {
        return kills / maxKills;
    }

    private float TimeFactor(float time)
    {
        return 1 - (time / maxTime);
    }

    private float AttackTypeFactor(AttackType attackType)
    {
        switch (attackType)
        {
            case AttackType.Quick: return 0.1f;
            case AttackType.Heavy: return -0.1f;
            default: return 0;
        }
    }

    private float EnemyStateFactor(EnemyState state)
    {
        switch (state)
        {
            case EnemyState.Normal: return 0;
            case EnemyState.Scared: return 0.2f;
            case EnemyState.Aggressive: return -0.2f;
            default: return 0;
        }
    }
    public void Update()
    {
        if (currentHelth <= maxHealth * 0.3f)
        {
            currentState = EnemyState.Aggressive;

        }
        else
        {
            currentState = EnemyState.Normal;
        }
        HealthHpBar.fillAmount = currentHelth / maxHealth;
        dodgeText.transform.LookAt(Camera.main.transform);
        HealthHpBar.transform.LookAt(Camera.main.transform);
        levelsText.transform.LookAt(Camera.main.transform);
        healthText.transform.LookAt(Camera.main.transform);
        dodgeText.transform.Rotate(0, 180, 0);
        levelsText.transform.Rotate(0, 180, 0);
        healthText.transform.Rotate(0, 180, 0);
        if (dodgeTextTimer > 0)
        {
            dodgeTextTimer -= Time.deltaTime;
            if (dodgeTextTimer <= 0)
            {
                dodgeText.gameObject.SetActive(false);
            }
        }
        CheckEvolution();
    }
    public float AdjustDamageBasedOnHealth(float damage)
    {
        if (currentHelth <= maxHealth * 0.3f)
        {
            return damage * 0.7f;
        }
        return damage;
    }
    void Die()
    {
        animator.SetBool("IsDie", true);
        GetComponent<Collider>().enabled = false;
        this.enabled = false;
        GetComponent<NavMeshAgent>().speed = 0;
        HealthHpBar.fillAmount = 0;
        if (gameObject.tag == "enemy")
        {
            kills++;
            questions.UpdateKills();
            if (kills >= 20)
            {
                GameObject[] objectsToDestroy = GameObject.FindGameObjectsWithTag("ObjectToRemove");
                foreach (GameObject obj in objectsToDestroy)
                {
                    Destroy(obj);
                }

GameObject.FindObjectOfType<GameManager>().ShowBossMessage();

            }
        }
        Destroy(gameObject, 7f);
    }
    private void CheckEvolution()
    {
        if (kills >= evolutionThreshold * (evolutionLevel + 1) && evolutionLevel < maxEvolutionLevel)
        {
            evolutionLevel++;
            ApplyEvolutionChanges();
        }
    }
```

```
    private void ApplyEvolutionChanges()
    {

        if (levelsText != null)
        {
            levelsText.text = $"Level {evolutionLevel + 1}";
        }
        switch (evolutionLevel)
        {
            case 1:
                maxHealth *= 1.5f;
                baseDodgeChance += 0.05f;
                baseDamage *= 1.2f;
                break;

            case 2:
                maxHealth *= 1.5f;
                baseDodgeChance += 0.05f;
                baseDamage *= 1.2f;
                break;

            case 3:
                maxHealth *= 1.5f;
                baseDodgeChance += 0.05f;
                baseDamage *= 1.2f;
                currentAttackType = AttackType.Heavy;
                break;

            default:
                break;
        }
        if (currentHelth < maxHealth)
        {
            currentHelth = maxHealth;
        }
        UpdateHealthText();
        HealthHpBar.fillAmount = currentHelth / maxHealth;
        AdaptDodgeProbability();
    }


    private void AdaptDodgeProbability()
    {
        float timeSinceLastAttack = Time.time - lastAttackTime;
        dodgeProbability = baseDodgeChance +
                           weightHealth * HealthFactor(currentHelth) +
                           weightKills * KillsFactor(kills) +
                           weightTime                                  *
TimeFactor(timeSinceLastAttack) +
                           weightAttackType                            *
AttackTypeFactor(currentAttackType) +
                           weightState * EnemyStateFactor(currentState);
    }

    public void Attack(GameObject target)
    {
        Debug.Log("Атака начата");
        if (target.CompareTag("Player"))
        {
            bool isCriticalHit = Random.Range(0f, 1f) < 0.1f;
            float damageDealt = baseDamage * (1 + 0.2f *
evolutionLevel);


            if (isCriticalHit)

            {

                damageDealt *= criticalHitMultiplier;

            }

            else

            {

            }

            if (playerIndicators != null)

            {

                playerIndicators.TryChangeHealthAmount(
damageDealt);

            }

            Damage                damageComponent                =
target.GetComponent<Damage>();

            if (damageComponent != null)

            {

                damageComponent.TakeDamage(damageDealt);

            }


            lastAttackTime = Time.time;

        }

    }

}
```