Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**ВИКОРИСТАННЯ МАШИННОГО НАВЧАННЯ ДЛЯ АДАПТАЦІЇ СКЛАДНОСТІ МОБІЛЬНИХ ІГОР**»

Виконав: студент 3-го року навчання,

Освітньої програми «Інженерія програмного забезпечення», 121

Пермяков Андрій Ігорович

Керівник Франків О.О.,_____

асистент

Рецензент _____

(прізвище та ініціали)

Секретар ЕК

_____

«____» _____ 20____ р.

Київ – 2023

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики технологій факультету інформатики

ЗАТВЕРДЖУЮ
Викладач кафедри інформатики ,
канд. фіз-мат. наук, доц. _____ Гороховський С.С.
(підпис)
„____” _____2023р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Пермякову Андрію Ігоровичу
факультету інформатики 3 курсу бакалаврської програми
ТЕМА: Використання машинного навчання для адаптації складності мобільних ігор

Зміст ТЧ до курсової роботи:
Індивідуальне завдання
Вступ
Теоритичні основи
Обґрунтування підходу до вирішення проблеми
Деталі імплементації
Висновки
Джерела

Дата видачі „___” _____ 2022 р. Керівник_____
(підпис)
Завдання отримав _____
(підпис)

Тема: **Використання машинного навчання для адаптації складності мобільних ігор**

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	17.10.2022	
2.	Огляд літератури та документації за темою роботи.	20.12.2022	
3.	Проведення досліджень.	27.12.2022	
3.	Опис результатів дослідження.	10.03.2023	
4.	Аналіз отриманих результатів з керівником.	31.03.2023	
6.	Корегування роботи.	03.04.2023	
7.	Створення презентації та написання доповіді.	3.05.2023	
8.	Остаточне оформлення пояснювальної роботи та слайдів.	16.05.2023	
9.	Захист курсової роботи.	25.05.2023	

Студент _____

Керівник _____

“_____” _____

Зміст

Анотація 7

ВСТУП 8

Розділ 1. Теоритичні основи 11

1.1 Reinforcement Learning як теоретична основа в дослідженні. 11

1.2 Підходи RL 12

1.3 Стан світу та простір станів в RL[6] 12

1.4 Простір дій в RL[7] 14

1.5 Метод Actor Critic у вирішенні завдань Reinforcement Learning 15

1.6 Висновки 17

Розділ 2. Обґрунтування підходу до вирішення проблеми 18

2.1 Опис проблеми 18

2.2 Спосіб збирання даних для навчання моделі 19

2.3 Аугментація даних та відмова від неї 19

2.4 Обґрунтування вибору підходу Actor-Critic 20

2.5 Вибір бібліотеки для навчання моделі 21

2.6 Порівняння Actor-Critic з іншими методами, використами в схожих проєктах 22

2.7 Висновки 23

Розділ 3. Деталі імплементації 24

3.1 Опис імплементації гри 24

3.2 Опис імплементації підробоного агента, способу збирання та зберігання даних 25

3.3 Стан світу, обробка та підготовка даних для використаввня в тренуванні моделі 27
3.4 Опис гіперпараметрів 29
3.5 Імплементація Actor-Critic 32
3.6 Навчання моделі 34
3.7 Доєднання моделі до мобільного застосунку 36
3.8 Висновки 37
ВИСНОВКИ 39
ДЖЕРЕЛА 42

Список графіків

Графік 0.1[4] 9
Графік 1.1: Архітектура Actor-Critic[8] 16
Графік 3.1. Функція Softplus 31

Анотація

У даній роботі розглянуто спосіб адаптації складності мобільних ігор за допомогою машинного навчання. У рамках цієї роботи розроблено модель, яка використовує підхід, заснований на навчанні з підкріпленням (RL) та алгоритмі Actor-Critic, для адаптації складності мобільної гри відповідно до перфомансу гравця. Використовуючи бібліотеку PyTorch, модель була навчена на великому обсязі даних з гри, з метою вивчення оптимальної стратегії адаптації. Результати експериментів свідчать про ефективність запропонованого підходу, де складність гри динамічно змінюється залежно від досвіду та успішності гравця. Цей дослід демонструє потенціал RL-підходу та алгоритму Actor-Critic у вирішенні задачі адаптивної складності мобільних ігор.

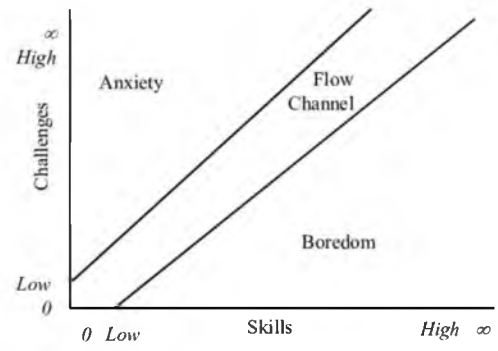
ВСТУП

Усе більша кількість людей користуються мобільними іграми як способом розваги і відпочинку. Це призводить до популярності геймінгу та збільшення ринку мобільних ігор. Ринок розвивається швидко, особливо під час пандемії, і очікується, що його доходи у 2023 році перевищать 100 мільярдів доларів США. Звіт Newzoo про світовий ринок мобільних ігор підтверджує цей тренд, показуючи збільшення доходів на 13% у річному обчисленні до 77 мільярдів доларів США в 2020 році[1]. У 2023 році цей ринок очікується досягнути вартості понад 102 мільярдів доларів США.

У багатьох відеоіграх є набір заздалегідь визначених рівнів складностей, які охоплюють діапазон від легкого до важкого. Проте такий підхід має низку недоліків. Гравець має обрати складність до початку гри, не маючи можливості оцінити, який саме рівень обрати. Окрім того, соціальний вплив може сприяти тому, що слабкі гравці обиратимуть рівень, який їм не підходить, адже у геймерській спільноті проходити гру на низькому рівні складності – «не круто». До того ж деякі ігри заохочують цю концепцію. Наприклад, однією з фішок серії ігор Wolfenstein є вибір складностей[2]. Окрім назв, іконка головного героя змінюється під час вибору складності. Список її рівнів з класичної гри:

- "Can I play, Daddy?": на іконці головний персонаж в чепчику з соскою, що символізує легкість цього рівня і є знущанням над гравцями, які його оберуть.
- "Don't hurt me": персонаж уже без соски й чепчика, проте виглядає дуже наляканим.
- "Bring 'em on!": персонаж упевнено дивиться вперед жорстоким поглядом.
- "I am Death Incarnate!": очі персонажа сяють червоним, він пронзає маніакальним поглядом, закусивши нижню губу.

У цьому графіку, створеному Михаєм Чиксентміхалі, описано поняття «потік», де гравець найбільш залучений і найщасливіший[3]. Пропорційно рівню навичок гравця має зростати й рівень запропонованих йому випробувань. Досвідчений гравець буде вважати просту гру нудною, в той самий час як недосвідчений гравець може вважати ту саму гру заскладною.



Графік 0.1[4]

Виходячи з тенденцій популярності мобільних ігор та різноманіття навичок бази користувачів, ***за мету даної роботи*** було поставлено розробку алгоритму адаптації складності мобільних ігор за допомогою машинного навчання. Це дозволить збільшити середню задоволеність користувачів від гри та відповідно підвищити її популярність, розширивши користувацьку базу.

Мета роботи зумовила наступне ***наукове завдання:***

1. Розробити гру-макет з можливістю збору даних.
2. Зібрати дані про різні рівні складності гри від різних користувачів.
3. Дослідити різні алгоритми машинного навчання та обрати найбільш релевантний для поставленої задачі.
4. Розробити модель машинного навчання для адаптації складності гри.
5. Порівняти дані, отримані з використанням навченої моделі з даними без її використання.
6. Проаналізувати результати експерименту та запропонувати можливі покращення.

Наукова новизна та практичне значення результатів: було розроблено модель, яка може самостійно адаптувати складність мобільної гри за допомогою машинного навчання. Це дозволяє створювати ігри, які можуть адаптуватися до навичок та вмінь гравців, забезпечуючи оптимальну геймплейну взаємодію. Результати дослідження можуть бути корисні для розробників мобільних ігор, які зацікавлені в розробці інноваційних методів адаптації складності гри, явно не залучаючи для цього самого гравця.

Розділ 1. Теоритичні основи

1.1 Reinforcement Learning як теоретична основа в дослідженні.

Reinforcement Learning (надалі RL) – навчання з підкріпленням. Це метод машинного навчання, який передбачає навчання агента, який взаємодіє з оточенням, щоб максимізувати отриману нагороду. В RL агент навчається шляхом взаємодії з оточенням, де він приймає рішення про дії, які повинен виконати для максимізації отриманої винагороди. Цей підхід оснований на ідеї навчання за допомогою проб і помилок, де агент підтримує свій досвід та вдосконалює свої дії на основі отриманих нагород.

RL має широкі застосування в таких областях як автоматичне керування, робототехніка, відеоігри, фінанси та інші. В комп'ютерних іграх RL може використовуватись для навчання ігрових агентів, які можуть взаємодіяти з гравцем або грати проти інших агентів. Це дозволяє створювати більш інтелектуальних гравців з більш складними стратегіями гри.

Саме RL зазвичай використовується для навчання моделі, що грає в гру, адже він доволяє агенту самостійно вивчати ефективність певної дії в складному ігровому середовищі. RL складається з 3 основних фаз: спостереження, дія та винагорода. На етапі спостереження передається будь-яка відповідна інформація про стан світу, яка може знадобитися штучному інтелекту для прийняття рішення. Фаза дії - алгоритм приймає рішення на основі даних про стан світу, переданих у нього, і остання фаза винагороди використовується для винагороди або покарання штучного інтелекту за прийняте ним рішення. Алгоритм навчається приймати найоптимальніші рішення, які призведуть до найвищої сумарної винагороди.

1.2 Підходи RL

У RL існує безліч підходів для навчання та оптимізації моделей[5]. Найпоширенішими з них є наступні:

- Q-learning: це один з найбільш популярних підходів RL, де агент навчається максимізувати очікувану нагороду, вивчаючи оптимальні дії, які має виконати в певних станах.
- SARSA: цей підхід також навчає агента максимізувати очікувану нагороду, проте він використовує іншу стратегію вивчення, де агент використовує свої попередні дії для вивчення оптимальної стратегії.
- Actor-Critic: це підход RL, де агент навчається за допомогою двох нейронних мереж - критика та актора. Критик оцінює, наскільки добре агент діє в певному стані, а актор визначає оптимальні дії на основі оцінок критика.
- Deep Q-Network (DQN): це підхід RL, де нейронна мережа використовується для оцінки функції Q. Агент вчиться максимізувати очікувану нагороду, використовуючи цю функцію Q для визначення оптимальних дій.
- Policy Gradients: це підход RL, де агент навчається безпосередньо максимізувати очікувану нагороду, вивчаючи оптимальну політику дій. Цей підхід використовує градієнтний спуск для навчання.
- Proximal Policy Optimization (PPO): це підхід RL, який базується на Policy Gradients, але використовує обмеження на зміну політики, що дозволяє досягати більш стабільної та ефективної навчання.

1.3 Стан світу та простір станів в RL[6]

Стан світу у RL описує поточне середовище, у якому діє агент. Це представлення містить всю необхідну інформацію про середовище, яку агент використовує для прийняття рішень, і може бути представлене у вигляді вектора значень або більш складної структури даних. Стан зберігається в пам'яті агента і може бути викликаний в будь-який момент навчання. Важливою функцією стану є те, що він дозволяє агенту відстежувати свій прогрес і приймати рішення на основі попереднього досвіду. Наприклад, стан світу гри може визначатися як набір інформації про здоров’я гравця, місце його знаходження, кількість видимих ворогів, поточний час матчу тощо. Агент, який, наприклад, адаптує складність гри, може навчитись за станом світу визначати чи складно гравцю чи легко й на основі цього виконувати дію.

У RL простір станів представляє собою перелік усіх можливих станів, які може отримати агент, включаючи поточний та майбутні стани, які доступні з поточного стану. Важливо, що простір станів може бути нескінченним, в залежності від моделі. Наприклад, у грі в шахи, простір станів включає в себе всі можливі розташування фігур на дошці. Такий простір є скінченним, адже можна порахувати скільки можливих варіантів існує. Якщо ж розглянути, наприклад, мобільну гру - космічний шутер, де гравець керує космічним кораблем та збиває інші космічні кораблі, маємо нескінченний простір станів. Кожен рівень може бути створений випадковим чином з різними ворожими кораблями, які можуть мати різні швидкості, здоров'я, щосекунди гравець і вороги рухаються і так далі. Кожна така дія змінює стан світу, тому їх простір ввжається нескінченним. Хоча простір станів може бути складним і великим, алгоритми навчання з підкріпленням призначені для ефективного вивчення та дослідження просторів станів з метою знаходження оптимальних політик. Агент повинен мати можливість відслідковувати всі можливі стани, щоб вибрати найкращу дію на кожному кроці.

1.4 Простір дій в RL[7]

Простір дій (action space) в RL визначає множину можливих дій, які агент може здійснювати в середовищі. Різне середовище очевидно дозволяє різні простори дій. Цей простір може бути *дискретним* або *неперервним*, залежно від того, які дії може здійснити агент у конкретній задачі.

У дискретному просторі дій, агент може обирати лише з обмеженого набору можливих дій, які зазвичай є конкретними та окремими діями. Наприклад, рухатися в певному напрямку без деталей про швидкість тощо або виконувати певну дію з скінченного переліку можливих.

У неперервному просторі дій, агент може здійснювати безліч можливих дій у формі неперервних значень. Це означає, що агент може змінювати параметри своєї дії з високою точністю, наприклад, окрім напрямку змінюючи ще й швидкість руху або кут огляду відповідно до вимог завдання. Однак, неперервний простір дій зазвичай є складнішим у порівнянні з дискретним, оскільки потребує від агента вміння більш точно налаштовувати свої дії.

Ця відмінність має деякі досить глибокі наслідки для методів у RL. Деякі сімейства алгоритмів можуть бути безпосередньо застосовані лише в одному випадку, і їх доведеться суттєво переробити для іншого. Саме тому для вибору підходу важливо зрозуміти над яким простором дій цей алгоритм буде оперувати.

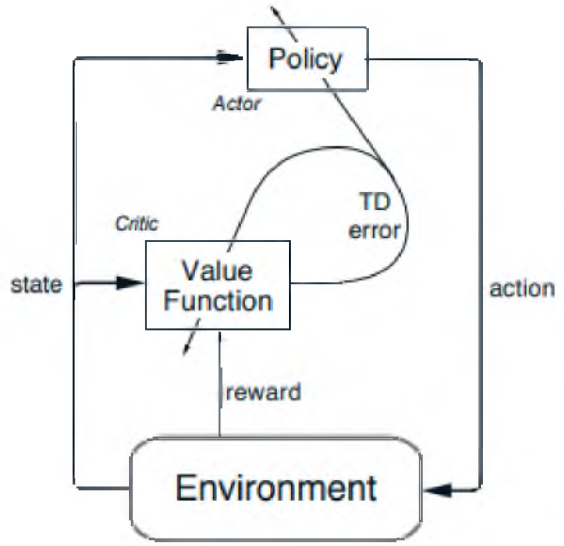
При формалізації простору дій для задачі про зміну складності гри в RL можна використовувати різні підходи, залежно від особливостей конкретної задачі та постановки вимог до системи. Наприклад, можна використати дискретний простій дій. На кожному кроці агент може вибрати одну з трьох можливих дій: збільшити складність гри, зменшити її або залишити без змін. Інший можливий підхід може полягати в використанні нескінченного простору дій, де кожна дія

відповідає числу, яке вказує на рівень зміни складності гри. Наприклад, додатнє число може вказувати на збільшення складності, а від'ємне на зменшення. Такий простір дій може бути забезпечити більшу гнучкість в управлінні складністю гри та можливість виконувати дрібні зміни в кожен момент часу замість того, щоб просто не змінювати нічого.

1.5 Метод Actor Critic у вирішенні завдань Reinforcement Learning

Метод Actor Critic (AC) є одним з популярних підходів до розв'язання задач RL. У методі Actor Critic ми маємо два компоненти: актора (Actor) та критика (Critic). Актор відповідає за прийняття рішень: вибір дій на основі поточного стану світу, тоді як критик оцінює якість прийнятих рішень. Задача актора полягає в максимізації очікуваної винагороди, тоді як критик відповідає за оцінку якості цих рішень, що дозволяє покращувати актора шляхом корекції його рішень.

Актор використовує Policy Gradient (PG) - метод, який заснований на градієнті функції цілі. Актор намагається максимізувати очікувану винагороду, отриману від навчальних даних, визначаючи оптимальну стратегію прийняття рішень. Щоб покращити стратегію, актор навчається тому, які дії повинні бути виконані в кожному стані світу, щоб максимізувати очікувану винагороду.



Графік 1.1: Архітектура Actor-Critic[8]

Критик використовує метод оцінки функції цінності (Value Function), щоб навчитися оцінювати очікувану винагороду для кожного стану світу. Він навчається на основі зворотного поширення помилки, коли винагорода, отримана актором, порівнюється з оцінкою винагороди, що надається критиком. Критик може бути реалізований як нейронна мережа, яка навчається на основі навчальних даних.

Загалом, метод Actor Critic - це надзвичайно потужний та універсальний підхід, який знайшов своє застосування в різних галузях. Він широко використовується в робототехніці для навчання роботів виконувати складні завдання, такі як монтаж складних механізмів або навіть пілотування дронів у небезпечних умовах. Крім того, він довів свою ефективність в галузі комп'ютерних ігор, де агенти повинні приймати рішення в складних середовищах.

Що стосується нашого проєкту, ми обрали саме цей підхід через його швидкість та ефективність навчання в купі з відносною непрожерливістю до кількості тренувальних даних.

1.6 Висновки

Reinforcement learning є однією з найбільш амбітних галузей штучного інтелекту, що вивчається дослідниками по всьому світу. Мета reinforcement learning полягає в тому, щоб навчити комп'ютер взаємодіяти з навколишнім середовищем і вчиняти дії, які допоможуть досягнути максимального результату. Існує безліч різноманітних підходів до RL, таких як deep reinforcement learning, Q-learning та policy gradients тощо. Кожен з цих підходів має свої переваги та недоліки, і вибір підходу залежить від характеру конкретної задачі, доступних ресурсів, кількості та типу даних.
Метод RL під назвою Actor-Critic є одним з найефективніших за швидкістю навчання та при відносно невеликих обсягах тренувальних даних підходів до розв'язання задач RL, який базується на використанні двох моделей: актора та критика. Актор визначає, яку дію слід виконати в даному стані світу, тоді як критик оцінює якість виконаної дії та надає актору зворотну інформацію про те, які дії були правильними або неправильними. Оскільки модель критика працює з оцінкою якості виконаної дії, то він може забезпечити актору зворотній зв'язок для налаштування параметрів стратегії прийняття рішень. Це дозволяє методу Actor-Critic ефективно працювати з великими просторами станів та дій, тобто саме такими, як в нашому проєкті.

Розділ 2. Обґрунтування підходу до вирішення проблеми

2.1 Опис проблеми

Задача полягає в написанні «голої» гри, на якій можна навчити модель ШІ, яка адаптуватиме складність гри відповідно до продуктивності гравця та власне навчанні моделі. Основні кроки:

- Обрано гру «космічний шутер»: гравець управляє космічним кораблем, який збиває ворожі космічні кораблі. Імплементація достатньо легка, при цьому можно доволі наочно змінювати складність в реальному часі. Якщо зробити орієнтований час матчу в три хвилини, то в будь-який момент часу можна дізнатися, чи складно гравцю, чи легко і на скільки. Наприклад, якщо пройшло півтори хвилини, тобто половина орієнтовного часу матчу, то в гравця має бути приблизно 50% здоров'я. Якщо ж в нього 80% здоров'я, то йому легко. А якщо за ті ж півтори хвилини в гравця лишилось 20% здоров'я, то йому складно.
- Ідейно складність має змінюватися максимально непомітно для гравця. Адже якщо користувач починає це помічати, він втрачає відчуття власного внеску у результат своєї гри і може відчувати невдоволення, оскільки його навички не мають впливу на загальний результат. Проте для наочності й простоти визначення успіху експеременту в цьому проєкті було прийнято рішення адаптувати складність більш радикально. Чим легше гра, тим швидше гравець літає і стріляє, тим менше він ураження отримує від куль і зіткнень з ворогами і навпаки.
- За дію агенту обрано наступне: для певного стану світу видавати число від -1.0 до 1.0. Від'ємне число спрошує гру, додатнє – поскладнює. Простір дій відповідно – нескінченний.

– Відповідно до простору дій обрано метод RL Actor-Critic.

2.2 Спосіб збирання даних для навчання моделі

Для генерації даних було розроблено підробного агента: реальна людина грає в гру, агент опрошується кожні 15 секунд. Він отримує дані про стан світу, відповідає на них рандомною дією. Початкова ідея була дати агенту дію від -1.0 до 1.0, уручну тримати складність гри завжди в проміжку від 0.0 до 2.0 невключно. Проте через радномність дій скачки були надто сильні, через що експериментальним шляхом було прийнято рішення скоротити дії агента до проміжку від -0.2 до 0.2.

Дії підробного агента оцінювались за описаною в розділі 2.1 функцією: якщо на даний момент гравець має здоров'я нижче, ніж повинен орієнтовно мати, то йому складно. Якщо агент послабив складність, коли гравцю складно – отримує позитивну винагороду і навпаки. Такий підхід дозволив зібрати різноманітні дані про правильні та неправильні дії в різних станах світу. Це зробило навчання моделі більш репрезентативним та узагальненим.

Дані збирались мною особисто та ще декількома добровольцями, які просто грали в гру. Для цього спеціально були обрані люди з різною кількістю досвіду в мобільних іграх, адже кінцевим результатом проєкту було задовольнити будь-яку людину незалежно від її вправності.

Незважаючи на те, що для збирання даних було залучено декілька людей, із самого початку було зрозуміло, що кінцева кількість даних буде відносно невеликою.

2.3 Аугментація даних та відмова від неї

Аугментація даних (або збільшення даних) у RL - це процес створення нових даних шляхом модифікації наявних даних для підвищення ефективності навчання агента. Цей процес зазвичай використовується в умовах обмеженої кількості даних для навчання, коли потрібно використовувати ті самі дані більше одного разу, або коли необхідно навчити агента здійснювати коректні дії в різних умовах.

Існує багато технік аугментації даних, які можна використовувати для покращення якості даних та ефективності навчання агента в RL. Наприклад, при тренуванні моделі, яка розпізнає котиків на картинках можна використовувати геометричні перетворення, такі як зміщення, поворот та збільшення/зменшення масштабу зображення, щоб створити нові зображення, які будуть відрізнятися від оригінальних, але все ще будуть коректними зображеннями котиків. Також можна застосовувати фільтри, що знижують чіткість зображення, додавати шум, який може репрезентувати забруднення або пошкодження зображення в реальному світі.

Однак, важливо розуміти, що використання технік аугментації даних повинно бути обґрунтовано та ретельно перевірене, щоб уникнути спотворення даних та перенавчання моделі. У нашому випадку навіть ретельно продумавши техніку аугментації, легко створити варіанти світів, які теоритично можливі, проте фізично є майже недосяжними. Це може призвести до того, що агент буде навчатися на неправдивих або неможливих даних, що може погіршити його результати в реальному світі. Саме тому було прийнято рішення не використовувати аугментацію для цього прокту.

2.4 Обґрунтування вибору підходу Actor-Critic

Дослідження, що були проведені у статті "Asynchronous Methods for Deep Reinforcement Learning"[9] описують різні підходи RL, щоб знайти найбільш ефективний спосіб навчання. Автори заявляють, що метод Actor-Critic може бути ефективнішим за інші методи в RL, особливо при використанні обмеженого обсягу даних, що робить його особливо корисним для проєктів, де зібрати велику кількість даних може бути складно. Також там стверджується, що цей метод є гарним вибором для неперервних просторів дій та світів через те, що він використовує дві окремі мережі для навчання.
Обидві з цих переваг є істотними для даного проєкту. Таким чином, зважаючи на ефективність при відносно невеликих кількостях даних та в середовищах з неперевними просторами дій та станів світів, саме метод Actor-Critic і було використано для цього проєкту.

2.5 Вибір бібліотеки для навчання моделі

Для навчання моделі було обрано PyTorch - фреймворк машинного навчання з відкритим вихідним кодом, розроблений компанією Facebook[10]. Він забезпечує зручний і простий інтерфейс для розробки глибоких нейронних мереж та інших моделей машинного навчання. Він має чудову документацію та безліч прикладів, що допомагає новачкам в машинному навчанні швидко освоїти фреймворк.
Крім того, PyTorch є одним з найбільш популярних фреймворків для машинного навчання в наші дні, що означає, що з ним пов'язано безліч матеріалів та ресурсів, включаючи навчальні курси, блоги, статті, форуми та інші ресурси для допомоги в розробці моделей машинного навчання.
Таким чином, зважаючи на те, що я є новачком у машинному навчанні, мій вибір пав саме на PyTorch з-поміж аналогів таких як TensorFlow чи Keras тощо.

2.6 Порівняння Actor-Critic з іншими методами, використами в схожих проєктах

У пості «How I trained a neural network to play my mobile game»[11] користувача під нікнеймом blazarious на Reddit описано розробку моделі штучного інтелекту, яка грає в класичну мобільну гру «3 в ряд». Цей проєкт вартий уваги, адже агент, що підкручує складність, теж в деякому сенсі грає в свою «гру». Blazarious оповів як було успішно реалізовано підхід до RL під назвою Curriculum Learning. Цей підхід реалізовує ідею того, що навчальні задачі мають бути представлені в певному порядку зі зростанням складності. Таким поскладненням там виступало збільшення простору станів світу: як тільки модель навчилась проходити простий рівень гри – їй починали давати більш складний.

Удаліть цього експерименту показує, що Curriculum Learning може бути ефективним для того, щоб змусити агента робити сенсовні рішення в задачах з дискретними просторами станів світу та дій. Проте в середовищі з неперервними станами, яке розглядається в нашому проєкті, такий підхід неможливо було б реалізувати.

Гонг Хонгіу в книзі «Dynamic Difficulty Adjustment: Developing an Adaptive Game AI with Machine Learning»[12] описав створення моделі, яка адаптує складність десктопної гри в реальному часі залежно від навичок гравця. Реалізовано було це за допомого RL з підходом PPO. Навчання відбувалось в реальному часі, адже для створення гри було використано двигун Unity і плагін до нього під назвою ML-Agents, який дозволяє доєднатися до TensorFlow прямо з гри.

Порівняно з Actor-Critic підхід PPO більш складний в реалізції: він має більше гіперпараметрів та вимагає більшої кількість тренувальних даних в цілому.

Саме тому було зроблено вибір на користь Actor-Critic. Останній має значно менше гіперпараметрів, що робить його більш легким у реалізації, тестуванні та адаптації, а також ефективніше тренується на відносно невеликих об'ємах даних. Хоча PPO може привести до кращих результатів в перспективі, його реалізація та підготовка вимагали б значно більше часу та зусиль. За тих самих умов, які наявні в нашому проєкті, модель натренована з PPO скоріше за все працювала б гірше. З урахуванням масштабів завдання та гіперпараметрів, Actor-Critic є оптимальним варіантом.

2.7 Висновки

Застосування RL та Actor-Critic алгоритмів для адаптації складності мобільної гри є дієвим підходом за обмеженої кількості даних в середовищах з неперевними просторами дій та станів світів, що дає можливість оптимізувати взаємодію гравця з грою та покращувати користувацький досвід за допомогою адаптивної складності.

Для реалізації моделі використано PyTorch - потужний інструмент для розробки та тренування нейронних мереж. Використання цієї бібліотеки сприяє ефективному використанню ресурсів та дозволяє розробникам швидко створювати та налаштовувати моделі.

Відмова від аугментації даних обґрунтована тим, що в даній задачі важливіше мати реальні дані про продуктивність гравців та їх взаємодію з грою, ніж створювати штучні уявні ситуації. Оскільки дані були зібрані вже від реальних гравців, то вони відображають реальні умови та проблеми, з якими може стикнутись модель у реальному житті.

Розділ 3. Деталі імплементації

3.1 Опис імплементації гри

Суть гри полягає в тому, щоб управляти космічним кораблем та збивати ворожі кораблі. Гравець управляється віртуальним джойстиком та постійно стріляє вперед. У межах видимої на екрані області корабель гравця може рухатись куди завгодно, проте не може її покинути. Ураження отримується від влучань ворожих куль та лобових зіткнень з іншими кораблями. Гра закінчується, коли здоров'я гравця досягає нуля. Час від часу за картою з'являються вороги, ціль яких підбити корабель гравця пострілами або зіткненням. Є 4 різні типи ворогів:

- найпростіший під назвою "Just emeny". Він летить прямолінійно з середньою незмінною швидкістю та стріляє вперед.
- "Tank enemy". Броньований ворог: перші кілька влучань в нього не мають ефекту. Як і попередній, рухається вперед та стріляє лінійно перед собою. Прото рухається цей ворог дещо повільніше, а стріляє трохи частіше.
- "Cannon enemy". Виїжджає на свою позицію, де зупиняється і починає обстрілювати всю мапу півколом. Зазвичай ховається по краях екрану.
- "Chasing enemy". Ворог-камікадзе: з високою швидкістю летить прямо на корабель гравця з метою лобового зіткнення. Не стріляє.

На початковому екрані гри є так звана «пасхалка» - прихований функціонал. Тапнувши тричі по фону можна обрати з яким агентом гратимемо: підробним або справжнім. По натиску на кнопку «Play» починається гра. Посередині екрану корабель гравця, зліва знизу віртуальний джойстик, а зліва згори – таймер. На цьому екрані теж присутній прихований функціонал: по потрійному тапу на фон можна показати або приховати дані про поточну складність гри та про останню дію, яку виконав агент. Коли здоров'я гравця опускається до 0

показується фінальний екран, на якому видно скільки часу корустивач протримався.

«Пасхалки» дозволили зручно підмінювати агентів та аналізувати процес змінення складності без відкриття зайвої інформації для тих, хто награвав дані для тренування моделі.

Гру було розроблено нативно під iOS мовою Swift з використанням IDE Xcode. Навігацію між екранами було зроблено за допомогою фреймворку UIKit, а у якості ігрового двигуна було обрано SpriteKit. У цілому, SpriteKit – це фреймворк для малювання двовимірних фігур, тексту, зображень тощо[13]. Він використовує Metal для досягнення високої продуктивності візуалізації, водночас пропонуючи простий інтерфейс програмування, що полегшує створення додатків із інтенсивним використанням графіки, зокрема ігор. Metal – це низькорівневий фреймворк від Apple, який надає додатку прямий доступ до графічного процесора (GPU) пристрою[14]. Таким чином для написання гри не було використано жодної сторонньої бібліотеки.

3.2 Опис імплементації підробоного агента, способу збирання та зберігання даних

Як уже було описано в підрозділі 2.2, спочатку підробний агент видавав випадкову дію в проміжку від -1.0 до 1.0, але швидко стало зрозуміло, що скачки занадто сильні й було прийнято рішення обмежити проміжок до [-0.2;0.2]. Задля централізованого регулювання складності було винесено загальну константу під назвою gameDifficultyKnob з початковим значенням рівним 1.0. Цей множник впливав на складність гри наступним чином:

– час між пострілами гравця (в секундах):

$$gameDifficultyKnob * 0.2$$

– швидкість руху гравця:

$$\frac{4}{1 + (gameDifficultyKnob - 1) * 0.3}$$

– ураження, яке гравець отримує від ворожої кулі

$$gameDifficultyKnob * 0.015$$

– ураження, яке гравець отримує від лобового зіткнення:

$$gameDifficultyKnob * 0.05$$

Ще до початку роботи над проєктом було зрозуміло, що після збирання перед тим як навчати модель дані потрібно буде обробити. Цей процес буде детально пояснено згодом, але тут варто зазначити, що для його спрощення одразу було закладено мінімальне здоров'я гравця рівне нулю і максимальне – одиниці. Оцінювання дії відбувалось наступним чином: обраховується орієнтовне здоров'я, яке мало б бути у гравця при лінійному його зменшення протягом цільового часу матчу – трьох хвилин. Формула:

$$\text{орієнтовне здоров}'\text{я} = 1 - \max\left(0, \min\left(1, \frac{\text{поточний час}}{\text{орієнтовний час матчу}}\right)\right)$$

Далі йде логічна оцінка дії агента. Для прикладу уявимо, що поточний рівень корабля гравця виявляється нижче за орієнтовний. Тоді якщо модель послабила

складність гри, їй видається винагорода у розмірі додатнього значення різниці здоров'я фактичного та орієновного. Якщо ж модель поскладнила гру в цьому випадки – видається штраф у вигляді від'ємної різниці здоров'я.
Варіант, коли орієнтвне здоров'я в цей момент часу дорівнює фактичному здоров'ю можна не розглядати, адже здоров'я представлено як дійсне число від 0 до 1. За теорією ймовірностей очевидно, що вірогідність обрати конкретну точку на відрізку дорівнює нулю, адже на відрізку будь-якої довжини є безліч точок, які можна представити дійсним числом.
За збирання даних відповідає клас GameDataCollector. Функція запису одного екземпляру даних приймає на вхід примірник структури стану світу, яку роз'яснено нижче, дію агента та оцінку цієї дії. Кожне поле структури стану світу, дія та оцінка записуються в один рядок у форматі CSV, тобто через кому. Дані зберігаються за допомогою нативного класу FileManager[15] у файлі з розширенням «.csv», що дозволяє відкривати й зручно переглядати його в застосунку Numbers[16], який є аналогом Exel від Apple.

3.3 Стан світу, обробка та підготовка даних для використаввня в тренуванні моделі

Для струтури стану світу було обрано наступні поля:

- поточне здоров'я;
- відношення здоров'я до часу матчу;
- час матчу;
- ураження, отримане за останню хвилю ворогів;
- середнє враження, яке вороги наносять за хвилю;
- поточну зміну множника складності гри від початкового значення.

Ідея полягала в тому, щоб модель навчилась асоціювати здоров'я гравця та швидкість його змінення з часом матчу й почала робити сенсовні дії, які б привели до бажаного результату.

Нормалізація – техніка, яка часто використовується як один з кроків підготовки даних для навчання моделі[17]. Метою її є приведення даних з різних діапазонів до певного спільного проміжку, зазвичай [-1;1] або [0;1] без спотворення. У статті «Why Data Normalization is necessary for Machine Learning models»[18] це наочно пояснено. У експерименті, проведеному автором, модель, навчена на ненормалізованих даних видає лише біля 49% точності на валідаційному наборі даних, в той час як модель, яку навчали на нормалізованих даних з незмінними всіма іншими параметрами мала цей показник близько 89%. Валідаційний набір даних - це частина даних, відмінна від тренувальних. Її використовують для перевірки ефективності моделі, яка навчалась на тренувальному наборі даних. Точність на валідаційному наборі даних є важливим критерієм для оцінки якості моделі та допомагає уникнути проблеми перенавчання моделі на тренувальних даних[19].

Як уже було зазначено, максимальне здоров'я корабля гравця було одразу закладено ріним одиниці, тому параметри, які його стосуються, тобто власне здоров'я гравця, його відношення до часу матчу та всі параметри, які стосуются враження, яке отримав гравець не потребують нормалізації.

Параметр «time elapsed», час матчу, було нормалізовано за допомогою знаходження найбільшого часу в усьому сеті даних та подальшого ділення всього кожного елемента стовпця на це значення.

Параметр «current difficulty», який означає поточну складність і має значення в проміжку [0;2] було нормалізовано діленням кожного значення стовпця на 2.

Усю роботу з csv за межами гри: нормалізцію та подальше використання для безпосередньо навчання моделі було прорoблено за допомогою Python бібліотеки для обробки та аналізу даних з відкритим кодом під назвою pandas[20].

3.4 Опис гіперпараметрів

Як зазначає офіційна документація Scikit-learn[21], програмного забезпечення для машинного навчання, гіперпараметри – це параметри, які встановлюються перед початком навчання моделі і не змінюються під час процесу навчання. Звичайні ж параметри, на відміну від них, є внутрішніми величинами, які можуть бути оновлені в процесі навчання. Вибір правильних гіперпараметрів є критичним для успішного навчання моделі, тому що вони впливають на її швидкість навчання, точність та стійкість. Гіперпараметри не вивчаються з даних і не можуть бути оціненими під час процесу навчання, тому їх необхідно встановлювати на основі досвіду, експертизи та емпіричних досліджень. Через повну відсутність досвіду та нестачу експертизи, усі гіперпараметри підбиралися за допомогою емпіричного методу. Було використано наступні гіперпараметри:

- Кількість епох. Початково було використано 50 епох та поступово збільшувалось і було прийнято рішення зупинитись на 250.
- Швидкість навчання – наскільки швидко модель навчається в процесі градієнтного спуску який буде описано в розділі 3.5. Менше значення цього параметру означає менші кроки в зміні політики. Використано значення 0.001.
- Гамма - використовується для оцінки впливу майбутніх нагород на поточні дії агента, забезпечуючи збалансовану роботу в умовах

невизначеності. Чим більша гамма, тим більший вплив майбутніх нагород на вибір поточної дії. Якщо значення гамма дуже маленьке, то агент приймає рішення, що залежить від нагород, які майже одразу отримує, не зважаючи на майбутні наслідки. Однак, якщо значення гамма дуже велике, агент забуває про наслідки своїх дій в більш віддаленому майбутньому. Було використано значення 0.99.

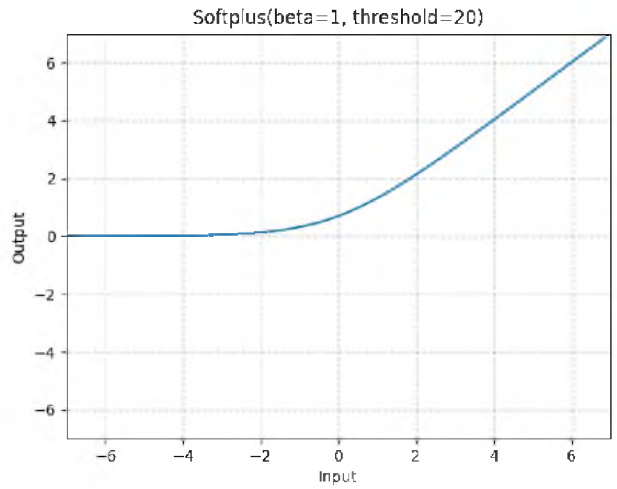
– Розмір партії (batch size). Описує скільки прикладів модель використає перед тим, як оновити свої параметри. Використано значення 32.
– Функція активації - математична функція, яка використовується в нейронних мережах для введення нелінійності в модель та забезпечення можливості моделі вирішувати більш складні завдання. Кожен штучний нейрон в має вхідні зв'язки, кожен з яких має вагу, а також вихід, який вираховується з ваг і вхідних сигналів. Функція активації застосовується до цього вихідного значення, щоб вирішити, чи має нейрон передавати сигнал до наступного шару мережі. Використано реалізацію функції Softplus з PyTorch. Softplus є нелінійною функцією, яка є гладкою та диференційованою на всій числовій вісі[22]. Її формула:

$$Softplus(x) = \frac{1}{\beta} * \log(1 + e^{\beta * x})$$

Параметри:

- beta – наскільки швидко функція зростає зі збільшенням вхідних значень. Більший параметр beta означає швидше зростання.
- threshold – поріг, який регулює, коли функція перейде від експоненційного згладжування до лініійної функції.

Графік функції Softplus з визначеними за замовчувванням параметрами beta = 1, threshold = 20:



Графік 3.1. Функція Softplus

– Розмір вхідного шару. Означає скільки властивостей має стан світу, тобто 7 у нашому випадку.
– Розмір прихованого шару, який є проміжним шаром між вхідним і вихідним. Використано значення 15.
– Розмір вихідного шар. Означає яку кількість значень повертає модель, тобто 1 у нашому випадку.
– Функція втрат – визначає рівнь похибки алгоритму, тобто різницю прогнозованого й фактичного значення[23]. Для оцінки того, наскільки точно нейронна мережа передбачає очікувані винагороди та для оцінки того, наскільки добре нейронна мережа вибирає дії, що максимізують нагороду було використано фукнцію втрат середньоквадратичної помилки (MSELoss). У PyTorch її визначено наступним чином[24]:

$$loss(x, y) = (x - y)^2$$

x представляє фактичне значення, а y – прогнозоване значення. Зрозуміло, що за такого підходу чим більша помилка, тим більше покарання моделі.

3.5 Імплементація Actor-Critic

У PyTorch одним з головних «будівельних блоків» для моделей є структура даних під назвою тензор[25]. Вона є багатовимірним масивом і може використовуватися для представлення вхідних даних, параметрів, проміжних розрахунків та вихідних даних. Тензор станів є двовимірним з розмірами: (кількість зразків даних, кількість характеристик світу). Тензори дій та винагород є одновимірними з розмірами в кількість зразків даних.
Оптимізатор – це алгоритм, який викристовується для налаштування параметрів моделі з метою мінімізації функції втрат та тренувальному наборі даних. Одним із найпопулярніших у PyTorch таких алгоритмів є Adam (Adaptive Moment Estimation), який власне й було використано[26]. Adam це адаптимний алгоритм оптимізації швидкості навчання, який оновлює коефіцієнти моделі за допомогою градієнтного спуску.
Градієнтний спуск – алгорим для мінімізації функції втрат, який шукає мінімум функції втрат, змінюючи параметри моделі у напрямку, протилежному до напрямку найшвидшого зростання функції втрат[27]. Процес вимірювання градієнту в поточній точці, обчислення кроку зміни паарметрів та їх власне оновлення виконується в циклі до тих пір, поки не досягнеться заданий рівень точності або функція втрат не досягне мінімального значення.
Оптимізатор Adam приймає в якості аргументів параметри моделі та вищеописану швидкість навчання. Сама модель реалізовано окремим класом ActorCritic, який є нащадком базового класу torch.nn.Module[28], базового класу

для всіх модулів нейронних мереж. В конструктор приймаються три параметри: розміри вхідного, прихованого та вихідного шарів, значення яких були пояснені вище.

Актора та критика створено за допомогою контейнера Sequential[29] з PyTorch, який дозволяє поєднати кілька шарів нейронної мережі в єдину модель. Кожен елемент в послідовності є примірником модуля PyTorch. Вихідні дані одного модуля є вхідними даними наступного. Повністю зв'язаний шар (fully connected layer або dense layer) – це особливий тим шару нейронної мережі, де кожен нейрон у шарі зв'язаний з кожним нейроном у попередньому шарі.

Перший шар актора – повністю зв'язаний шар, імплементований модулем Linear[30] з кількістю вхідних даних рівній гіперпараметру розміру вхідного шару і кількістю вихідних даних рівній гіперпараметру розміру прихованого шару. Ініціалізується з випадковими вагами, які будуть вивчені в процесі навчання за допомогою алгоритму градієнтного спуску - звоторнього поширення помилки.

Вихідні дані першого шару актора пропускаються через функцію ReLU[31], яка замінює кожен від'ємний елемент тензора на нулі, не змінюючи при цьому додатні. Її формула:

$$ReLU(x) = \max(0, x)$$

Далі пара Linear + ReLU повторюється ще раз з розмірами вхідних та вихідних шарів рівних розміру прихованого шару, а потім ще раз з кількістю вхідних даних рівній розміру прихованого шару, а вихдних – розміру вихідного шару.

Закінчується актор функцією tanh, тобто гіперболічним тангенсом, для масштабування вихідних даних до проміжку [-1;1].

Перші чотири шари критика – точно такі самі, як і в актора. Далі йде останній Linear шар, який має кількість вихідних даних рівну одиниці, адже метою критика є оцінка стану, яка є скалярною величиною.
Функція forward() визначає передачу даних через мережу нейронів. Вона приймає тензор x, який пропускається через актора й критика і повертає дію, отриману від актора та оцінку цієї дію, отриману від критика. При цьому дія домножається на 0.2 для того, щоб мастабувати інтервал після вищеописаної функції tanh в [-0.2, 0.2].

3.6 Навчання моделі

Оскільки під час тренування градієнти накопичуються при ітерації кожної партії (batch), перед кожною ітерцаєю їх варто обнуляти. Це робиться за допомогою фукнції ".zero_grad()" на оптимізаторі, виклик якої очищує градієнти всіх параметрів моделі, які були розраховані в попередній ітерації. Це прибирає вплив градієнтів з минулої ітерації на результати з нової ітерації.
Наступним кроком модель Actor-Critic використовується для отримання дії та її оцінки. Потім створюється розподіл політики з використанням передбаченої дії і нормального розподілу[32] зі стандартним відхиленням 0.1. Розподіл політики використовується для генерації дій на основі поточного стану. Він представляє розподіл ймовірностей щодо можливих дій за поточного стану. Більше стандартне відхилення призводить до дій, які більш тісно згруповані навколо середнього значення.
Далі з розподілу політики функція log_prob() обчислює логарифм імовірності, що випадкова величина, що має цей розподіл, прийме певне значення. Тобто результуючий тензор log_probs є мірою того, наскільки вірогідними є дії агента відповідно до політики розподілу.

Наступним кроком до тензору дій поелементно застосовується фукнція softplus, яку було описано в розділі 3.4. Отриманий тензор логаритмується в log_softplus_actions. Невелика константа значенням в 1e-8 додається в тілі логаритму аби запобігти повернення -infinity або NaN, коли її аргумент дуже близький до нуля, що може спричинити проблеми з числовою стабільністю через невідповідність машинної точності чисел.

Показник втрати актора є ознакою ефективності політики та використовується для її оновлення під час навчання. У нашому проєкті вона обчислюється як від'ємне середнє значення різниці log_probs та log_softplus_actions за допомогою вбудованої функції mean[33] з PyTorch:

actor_loss = -torch.mean(log_probs – log_softplus_actions)

Показник втрати критика є мірою похибки між прогнозованою винагородою для даного стану та фактичною винагородою, отриманою від середовища. На меті в критика стоїть мінімізація цієї втрати шляхом коригування вагів під час навчання. У нашому проєкті втрата критика обчислюється за допомогою середньоквадратичної помилки між передбаченими значеннями та фактичними винагородами:

critic_loss = nn.MSELoss()(винагороди передбачені, винагороди справжні)

Далі загальна втрата вираховується наступним чином:

loss = actor_loss + critic_loss

Після цього на *loss* викликається функція “backwoard()”[34], яка обчислює градієнт втрат з урахуванням параметрів моделі. Далі викликається метод “step()”[35] на оптимізаторі, щоб оновити параметри моделі на основі обчислених градієнтів.

Весь процес повторюється для кожної партії даних з датасету. Клас DataLoader[36] ітерує через всі партії, для кожної обчислюючи градієнти та оновлюючи параметри. Нарешті, після кількості повних проходів по датасету рівній кількості епох, натренована модель зберігається у файл з розширенням .pt і є готовою до використання.

3.7 Доєднання моделі до мобільного застосунку

Напряму використовувати модель PyTorch з мобільного засосунку неможливо, тому було пропрацьовано декілька варіантів:

- конвертувати в нативну для iOS CoreML[37], використовуючи тип ONNX[38, 39] як проміжний крок. Конвертацію в ONNX вийшло успішно провести, проте згодом виявилось, що конвертація ONNX в CoreML більше не підтримується[40].
- запускати з iOS проєкту python скрипт, який вже у свою чергу доступається до моделі, проте це не вийшло зробити, бо не знайшлось способу запустити скрипт з мобільного додатку нормальним шляхом без воркераундів.
- нарешті було прийнято рішення запустити на локальному хості сервер, який прийматиме стан світу як http body, пропускатиме його через модель і повертатиме дію. Для цього було використано Flask[41], легковаговий веб-фреймворк для Python, який дозволяє розробникам швидко розгортати веб-додатки та API з мінімальними зусиллями.

Підхід з локальним сервером довзоляє доступатися до моделі лише з симулятора, який запущено на тому ж комп'ютері, де й розгорнуто сервер, але не на фізичному телефоні. Для вирішення цієї проблеми було використано ngrok[42] - інструмент для тунелювання мережі, який дозволяє локальним комп'ютерам та серверам відкривати публічний тунель в Інтернеті. Він генерує унікальну URL-адресу, яку можна використовувати для доступу до локального сервера з будь-якої точки Інтернету, що дозволяє легко використовувати модель з будь-якого фізичного девайсу, навіть якщо він за межами локальної мережі. Була спроба «зашити» запуск і відповідно припинення ngrok прямо в запуск проєкту, адже Xcode дозволяє додавати власні bash скрипти до етапу збірки. Проте виявилось, що навіть якщо в скрипті поставити виконання процесу на фон, Xcode не продовжить збірку допоки цей процес не буде завершено, тому від цієї ідеї довелось відмовитись.

Натомість було прийнято рішення запускати окремо сервер і окремо проєкт. Запуск сервера реалізовано за допомогою bash скрипта, який власне й запускає ngrok. Після цього скрипт використовує jq[43] - інструмент командного рядка для обробки та маніпулювання JSON-даними для того, щоб витягнути згенеровану сервісом публічну адресу. Після цього відбувається запис згенерованої адреси до файлу, який потім вичитується з iOS додатку. Відповідно ця адреса й використовується для отримання дій агента. Таким чином будь-який телефон, на якому встановлено застосунок, може доступатися до моделі, якщо сервер запущено й не залежати при цьому від того, чи запущений він під дебагером Xcode.

3.8 Висновки

Було розроблено систему, що складається з мобільного додатку та моделі машинного навчання, що взаємодіють між собою через мережу. У результаті успішної імплементації моделі на основі RL з використанням Acrtor-Critic у середовищі PyTorch, її було інтегровано в мобільну гру "космічний шутер", яка була написана з нуля мовою Swift з використанням фреймворку SpriteKit. Для забезпечення зв'язку між грою та моделлю використовувався локальний сервер на Python Flask та сервіс ngrok, який забезпечував доступ до моделі з мережі. Усі ці етапи вимагали великої роботи та уваги до деталей. Для тренування моделі було потрібно створити та оптимізувати набір даних, обрати підходящу архітектуру та налаштувати параметри моделі. Крім того, щоб забезпечити взаємодію між грою та моделлю було перебрано багато різних способів конвертації та доєднанням моделі напряму. Після цього було прийнято рішення розробити локальний сервер та налаштувати сервіс ngrok, що вимагало деякої експертизи в написанні bash скриптів.

Загалом, усі ці зусилля виправдали себе, адже мобільний додаток з моделлю машинного навчання відкриває нові можливості для більш широкої вибірки користувачів, що можуть насолоджуватися грою з оптимальним рівнем складності та відповідно цікавості.

ВИСНОВКИ

Версію гри з використанням натренованого агента було перетестовано на тій самій групі гравців. У версії гри з підробним агентом середній час матчу всіх учасників експерименту становив близько 70 секунд. При цьому двадцять найдовших матчів в середньому займали 282 секунди. У версії гри з справжнім натренованим агентом середній час матчу з тими самими учасниками становив бильзко 87 секунд, а двадцять найдовших матчів займали в середньому 172 секунди.

Щоб поліпшити якість навчання моделі, можна зробити декілька кроків. Один з найбільш очевидних - залучення більшої кількості гравців для збору даних. Це дозволить не лише збільшити кількість даних, а й розширить діапазон зібраних даних від користувачів з різними рівнями навичок. Збільшення загального розміру датасету може допомогти знизити ризик перенавчання або недонавчання моделі.

Альтернативою цьому є навчання додаткових моделей, які гратимуть в гру й репрезентуватимуть гравців різного рівня досвіду. Для цього потрібно було б зібрати групу добровольців, поділити їх за досвідом на групи й назбирати дані, за якими навчити окремі моделі. Ці моделі у свою чергу вже можна було б використати для тренування моделі, що адаптує складність. Такий підхід вимагає великої кількості додаткових дій, але в довгостроковій перспективі може багато часу.

Доповнити минулий крок можна було б вибором іншого алгоритму навчання, який потребує більше даних, але на великому датасеті може бути більш ефективним за Actor-Critic за інших рівних умов. Наприклад Proximal Policy Optimization, який описано в розділі 2.5 цього документа.

Існує й інший варіант покращення, а саме впровадження донавчання, оскільки дані про стан світу все одно надходять на сервер щоразу при використанні моделі. Ці дані можна зберігати та використовувати в майбутньому для дотренування моделі. Дотренування - це процес подальшого навчання моделі на нових даних, що допомагає покращити її точність та ефективність у роботі з новими даними. Його можна проводити регулярно, щоб модель завжди була актуальною.

Крім того, можна позбутися проміжного кроку у вигляді сервера та впровадити модель прямо на девайс. Це дозволить уникнути надмірних витрат на http запити для визначення кожної дії. Персоналізоване донавчання прямо на девайсі є також можливим кроком покращення. Це посприяє створенню унікального досвіду для кожного окремого користувача, незалежно від інших. Проте, на відміну від серверного донавчання, за такого підходу модель важче контролювати, щоб вона не з'їхала з глузду й не почала видавати нерелевантні дії.

Повертаючись до результатів нашого експерименту, хоча початкова ідея, що модель має утримати користувача в грі протягом трьох хвилин, не була досягнута прямим шляхом, його можна вважати успішним. Слід зазначити, що більшість учасників експерименту не мали досвіду в мобільному геймінгу, через що страждає загальний середній час матчу. Також модель була налаштована на відносно невеликі кроки в адаптації складності задля згладжування скачків і відповідно меншої очевидності, що складність змінюється.

У цілому ж, середній час матчу став дещо ближчим до цільового. При цьому середній час найдовших матчів став помітно ближчим до орієнтовних трьох хвилин. Це свідчить про те, що модель відповідає потребам більш досвідчених

гравців і може забезпечити для них у середньому складніші виклики. Отже, можна зробити висновок, що експеримент є успішним.

ДЖЕРЕЛА

1. У 2023 році ринок мобільних ігор перевищить $100 млрд [Електронний ресурс] // profi-forex. – 2023. – Режим доступу до ресурсу: http://www.profi-forex.org/hi-tech/kompjuternye-igry/entry1008319624.html.
2. Difficulty levels [Електронний ресурс] – Режим доступу до ресурсу: https://wolfenstein.fandom.com/wiki/Difficulty_levels.
3. Flow Theory by Mihaly Csikszentmihalyi (1975) [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=oGJi7i1Ife0.
4. Mihaly Csikszentmihalyi showed us flow. Writers need flow. And tech destroys it. [Електронний ресурс] – Режим доступу до ресурсу: https://withoutbullshit.com/blog/mihaly-csikszentmihalyi-showed-us-flow-writers-need-flow-and-tech-destroys-it.
5. Reinforcement learning [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Reinforcement_learning.
6. Bowyer C. M. What is State in Reinforcement Learning? It is What the Engineer Says it is! [Електронний ресурс] / Caleb M. Bowyer. – 2022. – Режим доступу до ресурсу: https://medium.com/mlearning-ai/what-is-state-in-reinforcement-learning-it-is-what-the-engineer-says-it-is-47add99a1121.
7. Part 1: Key Concepts in RL [Електронний ресурс] // OpenAI. – 2018. – Режим доступу до ресурсу: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#action-spaces.
8. 4 - Sutton R. S. Chapter 11 Policy Approximation / R. S. Sutton, A. G. Barto // Reinforcement Learning: An Introduction / R. S. Sutton, A. G. Barto. – Cambridge, Massachusetts: The MIT Press, 2018. – (друге). – С. 258.
9. Asynchronous Methods for Deep Reinforcement Learning [Електронний ресурс] – Режим доступу до ресурсу: http://proceedings.mlr.press/v48/mniha16.pdf.

10. PyTorch [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/PyTorch.
11. blazarious. How I trained a neural network to play my mobile game [Електронний ресурс] / blazarious // Reddit. – 2022. – Режим доступу до ресурсу: https://www.reddit.com/r/reinforcementlearning/comments/w3oh3o/how_i_trained_a_neural_network_to_play_my_mobile/.
12. Xiong H. Dynamic Difficulty Adjustment: Developing an Adaptive Game AI with Machine Learning / Hongyou Xiong. – Northridge: California State University, 2019. – 54 с.
13. SpriteKit [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/spritekit.
14. Metal [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/metal/.
15. FileManager [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/foundation/filemanager.
16. Numbers [Електронний ресурс] – Режим доступу до ресурсу: https://www.apple.com/numbers/.
17. Normalize Data component [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/normalize-data?view=azureml-api-2.
18. Urvashi J. Why Data Normalization is necessary for Machine Learning models [Електронний ресурс] / Jaitley Urvashi – Режим доступу до ресурсу: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029.

19. Asad M. I. Training and Validation Data in PyTorch [Електронний ресурс] / Muhammad Iqbal Khan Asad – Режим доступу до ресурсу: https://machinelearningmastery.com/training-and-validation-data-in-pytorch/.
20. pandas [Електронний ресурс] – Режим доступу до ресурсу: https://pandas.pydata.org/.
21. 3.2. Tuning the hyper-parameters of an estimator [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-learn.org/stable/modules/grid_search.html#grid-search.
22. SOFTPLUS [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.nn.Softplus.html.
23. Функція втрат [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F_%D0%B2%D1%82%D1%80%D0%B0%D1%82.
24. PyTorch Loss Functions: The Ultimate Guide [Електронний ресурс] – Режим доступу до ресурсу: https://neptune.ai/blog/pytorch-loss-functions.
25. TORCH.TENSOR [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/tensors.html.
26. TORCH.OPTIM [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/optim.html.
27. Градієнтний спуск [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D1%96%D1%94%D0%BD%D1%82%D0%BD%D0%B8%D0%B9_%D1%81%D0%BF%D1%83%D1%81%D0%BA.
28. MODULE [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.nn.Module.html.
29. SEQUENTIAL [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html.

30. LINEAR [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html.
31. RELU [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html.
32. Нормальний розподіл [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D1%80%D0%BE%D0%B7%D0%BF%D0%BE%D0%B4%D1%96%D0%BB.
33. TORCH.MEAN [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.mean.html.
34. TORCH.TENSOR.BACKWARD [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.Tensor.backward.html
35. TORCH.OPTIM.OPTIMIZER.STEP [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/docs/stable/generated/torch.optim.Optimizer.step.html.
36. DATASETS & DATALOADERS [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html.
37. Core ML Models [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/machine-learning/models/.
38. Open Neural Network Exchange [Електронний ресурс] – Режим доступу до ресурсу: https://onnx.ai/.
39. EXPORTING A MODEL FROM PYTORCH TO ONNX AND RUNNING IT USING ONNX RUNTIME [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html#export-to-coreml.

40. coremltools 6.0 [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/apple/coremltools/releases/tag/6.0.
41. Flask [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/2.3.x/.
42. ngrok [Електронний ресурс] – Режим доступу до ресурсу: https://ngrok.com/.
43. ./jq [Електронний ресурс] – Режим доступу до ресурсу: https://stedolan.github.io/jq/.