

Praca Dyplomowa Inżynierska

Rafał Kuligowski
205835

Zastosowanie metod uczenia maszynowego do stworzenia sztucznej inteligencji w turowych grach RPG

The use of machine learning methods to create artificial intelligence in
turn-based RPG games

Praca dyplomowa na kierunku:
Informatyka

Praca wykonana pod kierunkiem
dra Marka Karwańskiego
Instytut Informatyki Technicznej
Katedra Zastosowań Matematyki

Warszawa, rok 2023



SZKOŁA GŁÓWNA
GOSPODARSTWA
WIEJSKIEGO

Wydział Zastosowań
Informatyki
i Matematyki

Oświadczenie Promotora pracy

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia tej pracy w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis promotora

Oświadczenie autora pracy

Świadom/a odpowiedzialności prawnej, w tym odpowiedzialności karnej za złożenie fałszywego oświadczenia, oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami prawa, w szczególności z ustawą z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. 2019 poz. 1231 z późn. zm.)

Oświadczam, że przedstawiona praca nie była wcześniej podstawą żadnej procedury związanej z nadaniem dyplomu lub uzyskaniem tytułu zawodowego.

Oświadczam, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną. Przyjmuję do wiadomości, że praca dyplomowa poddana zostanie procedurze antyplagiatowej.

Data

Podpis autora pracy

Streszczenie

Zastosowanie metod uczenia maszynowego do stworzenia sztucznej inteligencji w turowych grach RPG

Tematem pracy było zaimplementowanie sztucznej inteligencji uczącej się za pomocą metod uczenia maszynowego w turowej grze RPG. Praca składa się z czterech głównych części. Pierwsza część omawia technologie wykorzystane w pracy oraz ich alternatywy. Druga część skupia się na implementacji wcześniej wspomnianych technologii. Trzecia część zawiera instrukcję użytkownika gotowej aplikacji. W czwartej części znajdują się wnioski dotyczące implementacji tematycznego rozwiązania w produktach komercyjnych.

Słowa kluczowe – uczenie maszynowe, sztuczna inteligencja, gry RPG, uczenie przez wzmocnianie, turowe systemy walki

Summary

The use of machine learning methods to create artificial intelligence in turn-based RPG games

The topic of the paper was the implementation of artificial intelligence, which learns through machine learning methods, in a turn-based RPG game. The paper consists of four main parts. The first part discusses the technologies used in the work and their alternatives. The second part focuses on the implementation of the aforementioned technologies. The third part contains user instructions for the finished application. The fourth part includes conclusions regarding the implementation of the thematic solution in commercial products.

Keywords – machine learning, artificial intelligence, RPG games, reinforcement learning, turn-based battle system

Spis treści

| | | |
|----------|---|-----------|
| 1 | Wstęp | 9 |
| 1.1 | Cel i zakres pracy | 9 |
| 2 | Technologie | 10 |
| 2.1 | Rodzaje uczenia maszynowego | 10 |
| 2.2 | Silniki graficzne oraz pakiety do uczenia | 11 |
| 2.2.1 | Unity z pakietem ML-Agents | 11 |
| 2.2.2 | Godot z pakietem RL Agents | 11 |
| 2.2.3 | Unreal Engine z pakietem Learning Agents | 11 |
| 2.2.4 | Własny silnik z wykorzystaniem różnych pakietów do uczenia maszynowego | 12 |
| 2.3 | Algorytmy do uczenia maszynowego | 12 |
| 2.4 | Systemy walki w turowych grach RPG | 13 |
| 2.4.1 | Standard Turn Battle | 13 |
| 2.4.2 | Active Turn Battle | 14 |
| 2.4.3 | Press Turn System | 14 |
| 3 | Implementacja | 16 |
| 3.1 | Część aplikacji testująca model | 16 |
| 3.1.1 | Scena MainMenu | 16 |
| 3.1.2 | Scena CharacterCreator | 16 |
| 3.1.3 | Scena BattleScene | 18 |
| 3.1.4 | Implementacja systemu Press Turn | 19 |
| 3.2 | Część aplikacji automatycznie trenująca model | 22 |
| 3.2.1 | Implementacja ML-Agents | 22 |
| 3.2.2 | UI | 26 |
| 4 | Instrukcja instalacji i korzystania | 28 |
| 4.1 | Instalacja | 28 |

| | | |
|----------|---|-----------|
| 4.1.1 | Wymagane programy | 28 |
| 4.1.2 | Konfiguracja wirtualnego środowiska Pythona | 28 |
| 4.2 | Korzystanie z aplikacji | 29 |
| 4.2.1 | Uruchomienie trenowania modelu | 29 |
| 4.2.2 | Uruchomienie symulacji | 30 |
| 4.2.3 | Sposób oceny modelu | 31 |
| 5 | Wnioski | 32 |
| 6 | Bibliografia | 33 |

1 Wstęp

Sztuczna Inteligencja w przeciągu ostatnich lat objęła wiele dziedzin życia i jest ciągle dynamicznie rozwijana. Jej zastosowanie jest bardzo rozległe, a jedną z branż o dużych perspektywach rozwoju tej technologii jest przemysł rozrywkowy w który wchodzi gry wideo. W grach można zastosować tą technologię do chociażby generowania misji i zadań dla graczy aby nie były one powtarzalne, stworzenia mądrego systemu automatycznej walki dzięki któremu gracz nie lubiący tego elementu w grze może go pominąć, czy też do stworzenia sztucznej inteligencji dla przeciwników gracza. Ostatnie z tych przykładowych zastosowań zostanie zgodnie z tematem pracy omówione pod względem teoretycznym i praktycznym.

1.1 Cel i zakres pracy

Praca ma na celu zbadanie możliwości implementacji sztucznej inteligencji dla przeciwników w grach wideo z gatunku RPG¹ z turowym systemem walki. Wynikowo otrzymamy 2 aplikacje: trenującą model oraz testującą model w postaci symulatora walki. Treść pracy będą stanowić opisy technologii użytych w pracy wraz z alternatywami, opis implementacji rozwiązania na podstawie wybranych wcześniej technologii, instrukcję obsługi napisanej aplikacji oraz wnioski na temat implementacji takiego rozwiązania w grach komercyjnych na podstawie tego projektu.

¹RPG (ang. Role Playing Games) - gry w których gracz wciela się w rolę postaci występujących w fikcyjnym świecie. Gracze ponoszą wszelkie konsekwencje swoich akcji jako postać w świecie gry.

2 Technologie

W tym rozdziale zostaną szczegółowo omówione komponenty potrzebne do implementacji takowej sztucznej inteligencji. Według kolejności wyboru są to:

- Rodzaj uczenia maszynowego
- Silnik do stworzenia gry oraz pakiet do uczenia maszynowego kompatybilny z silnikiem
- Wykorzystywany algorytm do uczenia maszynowego
- System walki (w przypadku tej pracy - system oparty o tury)

2.1 Rodzaje uczenia maszynowego

W uczeniu maszynowym można rozgraniczyć 3 jego główne rodzaje (w oparciu o drugi rozdział z [1]):

- Nadzorowane (ang. Supervised) - model jest trenowany na danych wejściowych oraz odpowiednich dla nich wartościach wynikowych. Na ich podstawie uczy się przewidywania wyniku dla nowych, nieznanymi danych wejściowych.
- Nienadzorowane (ang. Unsupervised) - model jest trenowany na samych danych wejściowych nie dostając oczekiwanych wyników. Model sam musi odkryć sens jaki stoi za danymi wejściowymi.
- Przez wzmacnianie (ang. Reinforcement) - model trenowany za pomocą interakcji trenowanego agenta z wykreowanym otoczeniem. Agent wykonując akcje w otoczeniu dostaje nagrody lub kary w zależności od wyników jego akcji.

W przypadku tej pracy użyty zostanie rodzaj uczenia przez wzmacnianie przez charakterystyczną dla niego metodę uczenia przez interakcje. Metoda ta pasuje do gier wideo, gdyż z założenia polegają one na interakcji gracza ze światem gry. Model będzie mógł, tak samo jak gracz grający w grę, poprzez ingerowanie za pomocą akcji w otoczenie uczyć się wykonywania najlepszych możliwych akcji w danym momencie. Więcej wiedzy o tym jak działa uczenie przez wzmacnianie jest zawarte w [2].

2.2 Silniki graficzne oraz pakiety do uczenia

Wybór silnika graficznego oraz pakietu do uczenia maszynowego jest ze sobą głęboko powiązany. Ponadto jest on mocno subiektywny, gdyż jest uzależniony od umiejętności autora projektu. Lista popularnych wyborów w zakresie silnika i pakietu do uczenia maszynowego:

2.2.1 Unity z pakietem ML-Agents

Wśród wymienionych gotowych pakietów, ML-Agents jest najstarszym rozwiązaniem (udostępnione w 2017r.¹). Dostępne w pakiecie są 2 algorytmy uczenia przez wzmacnianie - PPO oraz SAC². Pakiet posiada własną dokumentację[3]. Sam silnik Unity umożliwia tworzenie aplikacji z grafiką 2D oraz 3D, operuje na łatwym w nauce języku C# oraz ma rozbudowaną dokumentację[4].

2.2.2 Godot z pakietem RL Agents

Godot jest silnikiem OpenSource o coraz mocniejszej pozycji na rynku. Według dokumentacji[5] silnik wspiera oficjalnie 2 języki programowania: autorski język GDScript oraz C#. Twórcy za pośrednictwem technologii GDEExtension dodali możliwość dodania wsparcia dla innych języków przez społeczność. Dzięki temu, można na tym silniku programować również w innych językach programowania takimi jak Rust, C++ czy Swift. Pakiet RL Agents jest najbardziej rozbudowanym pakietem wśród wymienionych. Bazuje na 4 innych pakietach³ do uczenia przez wzmacnianie oraz wspiera ponad 12 algorytmów uczenia. Więcej informacji o RL Agents można znaleźć w artykule naukowym poświęconym tej technologii [6] oraz w dokumentacji dostępnej w repozytorium projektu [7].

2.2.3 Unreal Engine z pakietem Learning Agents

Pod względem innych pakietów Learning Agents jest najmłodszy (powstał na początku 2023r.) oraz najslabiej udokumentowany. Pakiet operuje algorytmami PPO oraz BC (ang. Behavioral Cloning)⁴. Mimo to sam silnik Unreal Engine jest potężnym narzędziem z

¹Repozytorium powstało 8.09.2017r, pierwsza wersja pakietu pochodzi z 16.09.2017r - link do repozytorium <https://github.com/Unity-Technologies/ml-agents>

²Więcej informacji o tych algorytmach w rozdziale "Algorytmy do uczenia maszynowego"

³Pakiety na których bazuje RL Agents - StableBaseLines3, SampleFactory, CleanRL oraz Ray rllib (stan na 06.01.2024r.)

⁴Informacja pochodzi ze źródła (stan na 06.01.2024r.): <https://dev.epicgames.com/community/learning/tutorials/80WY/unreal-engine-learning-agents-introduction>

bardzo dużym wsparciem i możliwościami. Posiada wsparcie dla języka C++ oraz funkcji Blueprints umożliwiającej programowanie wizualne (bez pisanie kodu). Dokumentacja zarówno silnika Unreal Engine jak i pakietu Learning Agents znajduje się w [8].

2.2.4 Własny silnik z wykorzystaniem różnych pakietów do uczenia maszynowego

Rozwiązanie oferujące największą swobodę, ale też najtrudniejsze w implementacji. Do stworzenia środowiska gry potrzebna jest biblioteka graficzna pomagająca stworzyć silnik np. OpenGL lub pakiet do stworzenia aplikacji okienkowej np. PyGame. Wybierając pakiet do uczenia maszynowego warto szukać wśród pakietów Pythona gdyż ma on ich bardzo dużo. Dla uczenia przez wzmocnienie można zastosować pakiet PyTorch⁵ lub też jeden z 4 pakietów wykorzystywanych przez RL Agents.

Wybrany silnik i pakiet do uczenia maszynowego

Do pracy wybrałem Unity z pakietem ML-Agents przez moje wcześniejsze doświadczenie z Unity oraz zaawansowanie w języku C#. Uważam jednak, że dużo bardziej rozbudowany RL Agents na silniku Godot jest lepszym wyborem do tego typu projektu, gdyż daje dużo większe pole do dostosowania uczenia dla osiągnięcia lepszych rezultatów.

2.3 Algorytmy do uczenia maszynowego

Unity ML-Agents posiada 2 algorytmy uczące PPO (ang. Proximal Policy Optimization) oraz SAC (ang. Soft Actor-Critic). Pierwszy z nich jest podstawowym algorytmem używanym w pakiecie. Według [3] algorytm PPO jest algorytmem ogólnego przeznaczenia i jest bardziej stabilny w porównaniu z innymi algorytmami uczenia przez wzmocnienie. To samo źródło opisuje również różnicę między algorytmem SAC i PPO. SAC jest algorytmem "off-policy" kiedy PPO jest algorytmem "on-policy". Oznacza to że SAC uczy się z doświadczeń zebranych w dowolnym momencie w przeszłości, a podczas uczenia są losowo dobierane do treningu, gdzie w PPO występuje "policy", które reguluje podejmowanie decyzji i jest ulepszane z każdą iteracją uczenia. Dodatkowe informacje o algorytmie PPO znajdują się w [9], a o SAC w [10]. Do projektu wybrany został algorytm PPO z uwagi na wcześniej podaną stabilność oraz fakt, że jest on algorytmem będącym ponad

⁵Używany w ML-Agents oraz Learning Agents

6 lat w pakiecie ML-Agents co sugeruje, że jest on lepiej rozwiniętym rozwiązaniem w pakiecie.

2.4 Systemy walki w turowych grach RPG

Systemów walki turowej w grach RPG jest bardzo dużo, ale wszystkie charakteryzują się paroma wspólnymi cechami:

- Rozgrywka polega na wymianie tur postaci gracza z przeciwnikami
- Podczas tury gracz oraz przeciwnicy wykorzystują swoje tury jako zasób do wykonywania akcji
- Postacie gracza oraz przeciwnicy posiadają statystyki, które określają ich cechy. Do takich statystyk może należeć np. Obrona zmniejszająca obrażenia otrzymywane przez postać, czy też Atak zwiększający zadawane obrażenia
- Postacie gracza oraz przeciwnicy posiadają umiejętności, których mogą używać zużywając tury do wykonywania akcji w środowisku gry
- Walka kończy się gdy wszystkie postacie gracza lub wszyscy przeciwnicy umrą

Poniżej przedstawię parę podejść do turowego systemu walki wraz z przykładową grą w której został taki system walki użyty:

2.4.1 Standard Turn Battle

Najbardziej podstawowy, najpowszechniejszy i zarazem najprostszy system walki turowej. Na początku walki jest tworzona kolejka tur złożona ze wszystkich postaci na scenie na podstawie statystyki Szybkości. Gdy postać wykorzysta swoją turę jest przenoszona na koniec kolejki i następna postać w kolejce dostaje możliwość wykorzystania tury. Posiada on 2 główne warianty:

1. Postać posiadająca turę od razu po wybraniu ruchu go wykonuje i kolejka przechodzi do następnej osoby. Jest to system wykorzystywany np. w grach z serii Pokemon, czy też jako podstawowy system walki w kreatorze gier RPG o nazwie RPG Maker MV.
2. Wszystkie postacie na początku wybierają ruch jakiego będą używać podczas aktualnego przejścia przez kolejkę. Po wybraniu walka trwa jedno całe przejście kolejki, aby po ruchu ostatniej postaci w kolejce znów poprosić o wybranie ruchu. Jest to rozwiązanie znane np. z gier z serii Etrian Odyssey.

2.4.2 Active Turn Battle

System walki charakteryzuje się tym, że każda postać na początku zaczyna z pustym paskiem tury ładującym się w równych interwałach o wartość zależną od statystyki Szybkości. Im większa owa statystyka, tym szybciej tura jest ładowana. Gdy tura jakiejś postaci zostanie w pełni naładowana, walka zatrzymuje się, a postaci z naładowaną turą wybierają akcję jaką mają wykonać. Następnie pasek tury jest ponownie ładowany. Gdy dobiegnie do końca postać wykona wybraną wcześniej akcję i resetuje ładowanie tury do wartości początkowej. System ten jest znany z gier z serii Final Fantasy.

2.4.3 Press Turn System

Został stworzony i w głównej mierze jest używany w grach firmy Atlus począwszy od gry [11]. System opiera się o zwiększenie znaczenia tury jako zasobu gdzie na początku gra oddziela fazę ruchu gracza i przeciwnika. Następnie tworzy kolejkę ruchów dla uczestników danej fazy na podstawie statystyki Szybkości. Następne kroki jakie podejmuje system są zależne od wariantu tego systemu.

Wariant klasyczny

Występujący w grach z serii Shin Megami Tensei począwszy od [11]. Ilość tur dla strony wykonującej ruchy jest liczona na podstawie ilości postaci po tej stronie. Każda akcja wykonana przez postać jest oceniana przez system i na podstawie oceny zmienia się ilość tur. W przypadku tego modelu musi występować system Elementów, które są przypisane do każdej umiejętności ofensywnej, oraz relacji postaci do Elementu. Elementem mogą być np. żywioły takie jak Ogień, Lód czy Wiatr, natomiast relacjami są skutki Elementu na postać, które potem edytują ilość tur. W grach z tej serii występuje 6 relacji do Elementów:

1. Normalna
2. Słabość
3. Odporność
4. Niewrażliwość
5. Odbicie
6. Absorpcja

Jeśli postać trafi w słaby punkt przeciwnika nie traci całej tury, a tura jest "wciskana" (ang. Pressed). To samo dzieje się, gdy postać trafi cios krytyczny. "Wciśnięta" tura za-

chowuje się jak zwykła tura, ale nie może być wciśnięta ponownie. Tury można "wciskać" trafiając w słaby punkt przeciwnika oraz uderzając ciosami krytycznymi aż do momentu, w którym nie będziemy mieli dostępnych zwykłych tur. Jeśli posiadamy zarówno zwykłą turę, jak i "wciśniętą" turę to akcja z normalną relacją do Elementu spowoduje zużycie "wciśniętej" tury. Inne typy relacji zmniejszają ilość tur o określoną ilość np. trafienie w Element z relacją Niewrażliwości może obniżyć ilość tur atakującego o 2 priorytetyzując tury "wciśnięte". Jeśli postać postanowi nic nie robić podczas tury, jest ona "wciskana". Gdy stronie wykonującej akcje skończą się tury następuje zmiana stron. W przypadku gdy walka jest z osamotnioną, silniejszą postacią (tzw. Bossem), system daje tej postaci dodatkową turę.

Wariant uproszczony

Występujący w grach z serii Persona począwszy od części trzeciej. W tym przypadku również musi występować system Elementów i relacji opisany jak w wersji klasycznej. Postacie mogą wykonać jedną akcję według kolejki określonej na początku fazy. Jeśli po użyciu umiejętności ofensywnej postać trafi w słaby punkt przeciwnika lub uderzy cios krytyczny, nie traci ruchu i może użyć czegoś jeszcze raz, a przeciwnik dostaje status "Powalenia". Jeśli trafi się przeciwnika o tym statusie ciosem krytycznym lub w słaby punkt, nie dostaje się dodatkowego ruchu i rusza się kolejna osoba z drużyny. Status "Powalenia" znika gdy postać ma się ruszyć. W każdym innym przypadku ruch jest tracony i rusza się kolejna osoba. W grze Persona 5 dodano dodatkowo mechanikę Baton Pass, która umożliwia oddanie swojej tury po trafieniu krytycznym lub w słaby punkt innej osobie z drużyny. Jeśli kolejna osoba trafi cios krytyczny lub w słaby punkt może oddać swoją turę kolejnej osobie z drużyny wyłączając każdą osobę która korzystała z tej mechaniki wcześniej. Po wykonaniu akcji nie dającej dodatkowego ruchu, ruch dostaje następna postać.

Wybrany system walki

Do implementacji został wybrany klasyczny wariant systemu Press Turn z gry [11]. Mimo swojego skomplikowania (co może spowodować dłuższy czas uczenia modelu) posiada on bardzo dużo ciekawych interakcji związanych z turami, które sztuczna inteligencja będzie mogła wykorzystać na swoją korzyść.

3 Implementacja

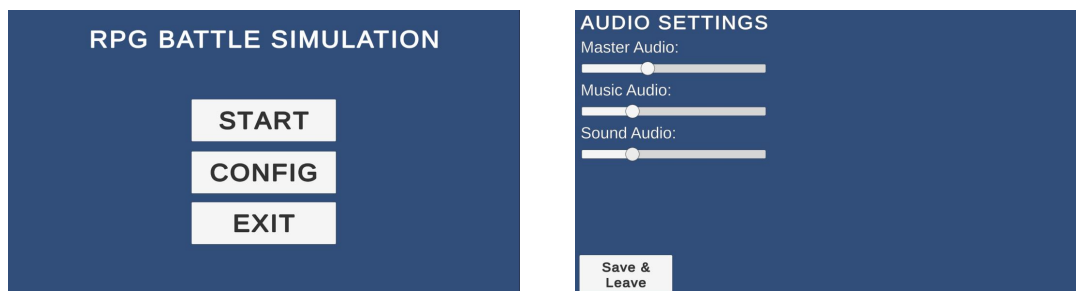
Aplikacja jest podzielona na 2 części: automatycznie trenującą model oraz testującą model w symulatorze walki.

3.1 Część aplikacji testująca model

Ta część aplikacji została zaimplementowana jako pierwsza w celu przygotowania systemu Press-Turn. Składa się z 3 scen Unity: MainMenu, CharacterCreator oraz BattleScene.

3.1.1 Scena MainMenu

Menu Główne składa się z 2 widoków: Panelu głównego oraz opcji. W panelu głównym są dostępne przyciski START, CONFIG oraz EXIT. START służy do uruchomienia kreatora postaci, CONFIG otwiera widok opcji, a EXIT służy do zamknięcia aplikacji¹. Widok opcji posiada suwaki zmieniające głośność muzyki oraz dźwięków w aplikacji i przycisk zapisujący ustawienia.



(a) Widok panelu głównego

(b) Widok opcji

Rysunek 3.1. Zrzuty ekranu przedstawiające menu główne

3.1.2 Scena CharacterCreator

Kreator postaci składa się z 2 widoków: panelu wyboru liczby aktorów w drużynie oraz kreatora postaci. Oba widoki posiadają też przycisk BACK powracający do menu głównego.

¹Przycisk zamknięcia aplikacji nie działa podczas testowania aplikacji na silniku, tylko gdy aplikacja zostanie wyeksportowana.

Panel wyboru liczby aktorów

Panel składa się z suwaka przyjmującego wartości całkowite w zakresie 1-4 oraz przycisk potwierdzający ilość aktorów o treści ACCEPT. Po jego wciśnięciu widok zmienia się na kreator postaci.

Kreator postaci

Składa się z przycisku Restart Actors wracającego do widoku wyboru liczby aktorów, przycisku Start Battle zaczynającego walkę gdy wszyscy aktorzy mają wypełnione pola statystyk, pola wyboru aktualnie edytowanego aktora oraz obszernego ekranu do wprowadzania statystyk postaci. Lista statystyk postaci (sens statystyk zostanie wyjaśniony podczas omawiania sceny walki):

1. HP (ang. Health Points, pl. Punkty Życia) - zakres 1-9999
2. MP (ang. Mana Points, pl. Punkty Many) - zakres 1-9999
3. Strength (pl. Siła) - zakres 1-99
4. Magic (pl. Magia) - zakres 1-99
5. Dexterity (pl. Sprawność) - zakres 1-99
6. Agility (pl. Zwinność) - zakres 1-99
7. Luck (pl. Szczęście) - zakres 1-99

Postacie posiadają również Elementy i odpowiadające im relacje. Lista Elementów (kolejność od lewej według ustawienia w widoku):

1. Obrażenia fizyczne
2. Ogień
3. Lód
4. Wiatr
5. Elektryczność
6. Światłość
7. Ciemność
8. Nadzwyczajny - ukryty element, zawsze o relacji neutralnej

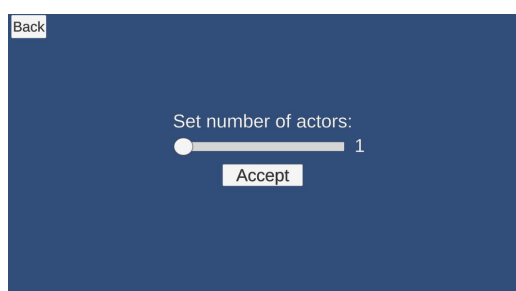
Lista relacji Elementu z postacią (wraz ze skrótem widocznym w kreatorze, wpływem na tury oraz wpływem na obrażenia):

1. Neutralny, skrót: "-", wpływ na turę: utrata 1 tury, wpływ na obrażenia: brak
2. Słabość, skrót: "Wk", wpływ na turę: "wciśnięcie" 1 tury, wpływ na obrażenia:

+20%

3. Odporność, skrót: "Str", wpływ na turę: utrata 1 tury, wpływ na obrażenia: -20%
4. Niewrażliwość, skrót: "Null", wpływ na turę: utrata 2 tur, wpływ na obrażenia: zniwelowanie obrażeń
5. Odbicie, skrót: "Rep", wpływ na turę: utrata 4 tur, wpływ na obrażenia: odbicie wartości obrażeń (obrażenia odbite również podlegają ocenie, ale nie wpływają na tury oraz nie mogą odbić się drugi raz)
6. Absorbacja, skrót: "Abs", wpływ na turę: utrata 4 tur, wpływ na obrażenia: wyleczenie o wartość obrażeń

W skład kreatora wchodzi również lista z wyborem umiejętności. Każdy rekord listy posiada: pole wyboru umiejętności, jej nazwę, koszt oraz przypisany Element.



(a) Widok panelu wyboru liczby aktorów



(b) Widok kreatora postaci

Rysunek 3.2. Zrzuty ekranu przedstawiające kreator postaci

3.1.3 Scena BattleScene

Rozbudowana scena przedstawiająca informacje na temat aktualnego stanu walki. Stworzona na podstawie UI z gry [11].

Scena posiada następujące elementy UI:

- Przycisk Leave Battle - służy do wyjścia z walki
- Lista tur - czerwone znaczniki oznaczają "wciśnięte" tury, a zielone zwykłe tury
- Widok drużyny gracza - generowany automatycznie, obramowanie pokazuje kto aktualnie posiada ruch
- Widok przeciwnika - sterowany przez wytrenowany model przeciwnik z widocznym paskiem zdrowia
- Lista umiejętności - umożliwia wybór umiejętności aktualnie ruszającej się postaci. Po wyborze umiejętności gracz jest proszony o cel.

- Dziennik zdarzeń walki - pokazuje umiejętności użyte od początku walki przez jakąkolwiek postać
- Stan drużyny gracza - pokazuje paski z imionami wszystkich postaci gracza, jak i ich stanem zdrowia i many



(a) UI gry [11]



(b) UI sceny walki w projekcie

Rysunek 3.3. Zrzuty ekranu porównujące UI stworzone w projekcie do występującego w grze [11]

3.1.4 Implementacja systemu Press Turn

System Press Turn zaimplementowany w projekcie działa na 2 obiektach na scenie BattleScene:

- BattleManager - odpowiada za logikę walki
- BattleData - przechowuje dane o postaciach, stanach i umiejętnościach

Zaczytywanie danych oraz startowa konfiguracja obiektów

Zanim wczyta się scena BattleScene, w kreatorze jest tworzony obiekt BattleData. Przed wypełnieniem go danymi o postaciach posiada listę ze stanami, jakich postaci mogą doznać (aktualnie zaimplementowany jest tylko stan Śmierci), oraz listę wszystkich umiejętności. Obiekt jest wypełniany danymi z kreatora dotyczącymi postaci. Dane te są zapisywane do dwóch list:

- party - przechowująca postaci sterowane przez gracza
- enemies - przechowująca informacje o postaci przeciwnika

Po wczytaniu się sceny, BattleManager zostaje zainicjowany. Posiada on następujące wartości:

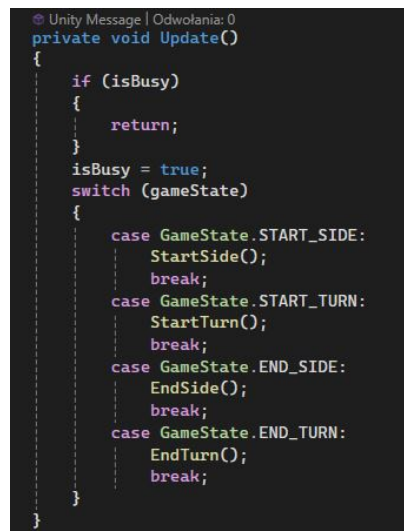
- selectedSkill - przechowuje informacje na temat wybranej przez postać umiejętności

- isPlayerTurn - wartość prawda/fałsz mówiąca czy rusza się gracz, bazowo ustawiony na prawdę
- isBusy - wartość prawda/fałsz odpowiadająca za przejście do następnego stadium walki, bazowo ustawiony na prawdę
- gameState - enumerator mówiący jakie jest stadium walki, może przyjąć jedną z następujących wartości: START_SIDE, START_TURN, END_SIDE, END_TURN
- currentCharacter - przechowuje informacje o aktualnie poruszającej się postaci
- moveQueue - lista z indeksami postaci ułożona w kolejności poruszania się (sortowana po statystyce Agility, od największej wartości do najmniejszej)
- currentIndex - wartość pomocnicza przechowująca indeks aktualnie poruszającej się postaci
- turnQueue - lista z wartościami prawda/fałsz odpowiadająca ilości tur (zwykle tury to wartości true, "wciśnięte" tury to wartości false)

Po wczytaniu wszystkich potrzebnych zasobów, BattleManager ustawia wartości gameState na START_SIDE oraz isBusy na false.

Główna pętla systemu walki

System walki operuje na funkcji Update wykonującej się co klatkę.



```

@ Unity Message | Odwołania: 0
private void Update()
{
    if (isBusy)
    {
        return;
    }
    isBusy = true;
    switch (gameState)
    {
        case GameState.START_SIDE:
            StartSide();
            break;
        case GameState.START_TURN:
            StartTurn();
            break;
        case GameState.END_SIDE:
            EndSide();
            break;
        case GameState.END_TURN:
            EndTurn();
            break;
    }
}

```

Rysunek 3.4. Funkcja Update w obiekcie BattleManager

Funkcja obsługuje tzw. pętlę gry czyli algorytm zapętłający rozgrywkę. Pętla gry operuje na wywoływaniu 4 funkcji widocznych na Rysunek 3.4. Pętla gry przedstawiona w postaci listy kroków prezentuje się następująco:

1. Wywołuje się StartSide(), które uzupełnia listę tur - turnQueue oraz listę indeksów postaci - moveQueue
2. Następuje aktualizacja currentIndex o pierwszy indeks z moveQueue, przenosząc ten index później na koniec listy moveQueue. Potem zmienia gameState na START_TURN
3. Wywołuje się StartTurn(), które ustawia wartość currentCharacter na aktualnie poruszającą się postać.
4. Jeśli ruch wykonuje gracz aplikacja pokazuje dostępne umiejętności postaci i czeka na reakcję użytkownika
5. Jeśli ruch wykonuje przeciwnik uruchamiana jest funkcja AIActions() która prosi o podjęcie decyzji wytrenowany model ML-Agents w wyborze celu oraz umiejętności
6. Po wyborze umiejętności i celu wykonuje się funkcja UseSkill() uruchamiająca logikę systemu Press Turn
7. Na koniec gameState jest zmieniany na END_TURN
8. Wywołuje się EndTurn(), które sprawdza warunki wygranej i przegranej (jeśli któryś z warunków jest prawdziwy, przejdź do 11.)
9. Jeśli turnQueue jest puste, zmień stan na END_SIDE, w przeciwnym wypadku przejdź do punktu 2.
10. Wywołaj funkcję END_SIDE, która zmienia wartość isPlayerTurn na przeciwną sobie i zmienia wartość gameState na START_SIDE. Przejdź do punktu 1.
11. Pokaż ekran końcowy i opuść pętlę gry

Działanie funkcji UseSkill() oraz wykorzystanie statystyk postaci

Wspomniana w pętli gry funkcja UseSkill() pełni kluczową funkcję w działaniu systemu walki. Głównym jej zadaniem jest pobranie zasobów (many oraz zdrowia) potrzebnych do użycia wybranej umiejętności, aktywacja umiejętności na wybranym celu oraz zmiana tur na podstawie oceny relacji postaci do elementu umiejętności. Gdy dochodzi do aktywacji umiejętności, statystyki postaci atakującej i broniącej są przekazywane do funkcji przeliczającej obrażenia. Formuły obrażeń prezentują się następująco:

$$PhysicalAttack = 5 * \sqrt{\frac{a.Strength}{b.Dexterity}} * m$$

$$MagicalAttack = 5 * \sqrt{\frac{a.Magic}{b.Dexterity} * m}$$

gdzie,

- a - atakujący
- b - broniący
- m - stała moc umiejętności

Jedyna niewykorzystywana nigdzie statystyka to Luck, z uwagi na brak planowanej wcześniej implementacji ciosów krytycznych w systemie walki.

3.2 Część aplikacji automatycznie trenująca model

Ta część aplikacji służy do trenowania modelu za pomocą pakietu ML-Agents. Składa się z 1 sceny Unity: TrainingScene.

3.2.1 Implementacja ML-Agents

Aby zacząć uczyć model w technice uczenia przez wzmacnianie potrzebne są środowisko oraz uczący się w nim agent.

Środowisko

Środowiskiem jest stworzony wcześniej system Press Turn z którego zostały usunięte niepotrzebne funkcje związane z UI. W rolę gracza wcieli się generator liczb losowych. Będzie on wybierał losową, dostępną umiejętność. Ilość postaci po stronie gracza jak i statystyki, umiejętności oraz relacje z Elementami będą losowane dla wszystkich postaci. Aby umożliwić powyższe zmiany, dodane zostały następujące elementy do środowiska:

- turns - ilość minionych tur
- maxTurns - maksymalna ilość tur po której środowisko powróci do stanu początkowego
- pointsMin - minimalna wartość wykorzystywana do losowania HP oraz MP postaci. Może przyjąć wartość z zakresu $<1, pointsMax)$
- pointsMax - maksymalna wartość wykorzystywana do losowania HP oraz MP postaci. Może przyjąć wartość z zakresu $(pointsMin, 9999>$
- statMin - minimalna wartość wykorzystywana do losowania statystyk postaci. Może przyjąć wartość z zakresu $<1, statMax)$

- statMax - maksymalna wartość wykorzystywana do losowania statystyk postaci. Może przyjąć wartość z zakresu (statMin,99>
- minActors - minimalna wartość wykorzystywana do losowania ilości postaci po stronie gracza. Może przyjąć wartość z zakresu <1,maxActors)
- maxActors - maksymalna wartość wykorzystywana do losowania ilości postaci po stronie gracza. Może przyjąć wartość z zakresu (minActors,4>

Agent

Implementacja oparta o artykuł [12]. Aby zaimplementować Agentą za pomocą pakietu ML-Agents należało stworzyć nową klasę dziedziczącą po klasie Agent oraz określenie jakie wartości będzie on przewidywał. W przypadku tej pracy, Agent będzie przewidywał 2 wartości:

- Indeks wybranej umiejętności
- Indeks wybranego aktora

Klasa Agent wymaga do działania nadpisania niektórych swoich metod. Lista przeciążonych metod wraz z opisem nowej funkcjonalności:

- void Heuristic() - z metody została usunięta cała funkcjonalność, przez fakt powodowania niepotrzebnych ostrzeżeń w konsoli Unity. Funkcja pozostaje pusta ze względu na nie używanie jej w projekcie.
- void CollectObservations() - metoda zajmuje się zebraniem danych ze środowiska oraz wysłanie ich do analizy(spis wszystkich zmiennych branych pod uwagę podczas uczenia został opisany niżej)
- void OnActionReceived() - metoda odczytuje wybrane przez Agentą wartości i przekazuje je do środowiska
- void WriteDiscreteMask() - metoda tworzy maskę uniemożliwiającą wybranie przez sztuczną inteligencję umiejętności, do których nie ma dostępu oraz postaci nie żyjących lub nie istniejących
- void OnEpisodeBegin() - metoda przywraca stan początkowy środowiska w którym się znajduje

```
Odwolania.0
public override void Heuristic(in ActionBuffers actionsOut)
{
}
Odwolania.0
public override void CollectObservations(VectorSensor sensor)
{
    if(sensor == null)
    {
        Debug.LogWarning("Input is null");
        return;
    }
    sensor.AddObservation(trainingManager.PrepareObservations());
}
```

(a) Funkcja Heuristic() oraz CollectObservations()

```
Odwolania.0
public override void OnActionReceived(ActionBuffers actions)
{
    resultGenerated = true;
    chosenSkill = actions.DiscreteActions[0];
    chosenActor = actions.DiscreteActions[1];
    //Debug.Log($"Actions Called:\n[chosenSkill] - Skill Index\n[chosenActor] - Actor Index");
}
```

(b) Funkcja OnActionReceived()

```
Odwolania.0
public override void WriteDiscreteActionMask(IDiscreteActionMask actionMask)
{
    //SetSkills
    List<int> skills = trainingManager.PrepareDataForAI();
    for (int i = 0; i < 11; i++)
    {
        if (skills.Contains(i))
        {
            actionMask.SetActionEnabled(0, i, true);
        }
        else
        {
            actionMask.SetActionEnabled(0, i, false);
        }
    }
    //SetActors
    bool isSomeoneAlive = false;
    for (int i = 0; i < trainingManager.party.Count; i++)
    {
        if(!trainingManager.party[i].CheckStatesByName("Death"))
        {
            isSomeoneAlive = true;
            break;
        }
    }
    if(isSomeoneAlive)
    {
        for (int i = 0; i < trainingManager.party.Count; i++)
        {
            if (!trainingManager.party[i].CheckStatesByName("Death"))
            {
                actionMask.SetActionEnabled(1, i, true);
            }
            else
            {
                actionMask.SetActionEnabled(1, i, false);
            }
        }
        for (int i = trainingManager.party.Count; i < 4; i++)
        {
            actionMask.SetActionEnabled(1, i, false);
        }
    }
}
```

(c) Funkcja WriteDiscreteMask()

```
Odwolania.0
public override void OnEpisodeBegin()
{
    if (firstRun)
    {
        firstRun = false;
        return;
    }
    if (tryingToReset)
    {
        Debug.LogWarning("Already trying to reset...");
    }
    tryingToReset = true;
    trainingManager.ResetEnvironment();
    positiveRewards = 0f;
    negativeRewards = 0f;
    episodeOver = false;
    /*trainingManager.confirmation.SetActive(true);
    trainingManager.startButton.gameObject.SetActive(false);*/
}
```

(d) Funkcja OnEpisodeBegin()

Rysunek 3.5. Zrzuty ekranu prezentujące ciała nadpisanych funkcji

Aby ML-Agents mogło działać poprawnie w systemie turowym, trzeba było dodatkowo wyłączyć Automatyczne Kroki Środowiskowe poprzez funkcję Awake() Agenta. Do ciała funkcji należało wpisać: `Academy.Instance.AutomaticSteppingEnabled = false;` Powoduje to również konieczność zaimplementowania manualnego wykonywania kroku środowiskowego za pomocą: `Academy.Instance.EnvironmentStep();` Dodatkowo do klasy zostały zaimplementowane dodatkowe metody oraz właściwości pomocnicze opisane w [12]. Są to:

- `firstRun` - wartość prawda/fałsz określająca czy jest to pierwsze wywołanie metody `OnEpisodeBegin()`
- `tryingToReset` - wartość prawda/fałsz informująca, że środowisko jest w trakcie resetu
- `resultGenerated` - wartość prawda/fałsz informująca, że model skończył przewidywać wartość i ją zwrócił do środowiska

- NotifyEndEpisode() - metoda daje nagrodę sztucznej inteligencji za wykonane akcje i kończy epizod
- GetOutput() - metoda czekająca na to, aż ML-Agents zwróci wynik

Interakcja środowiska z agentem

Aby cała struktura uczenia działała, trzeba sprawić aby agent posiadał interakcje ze środowiskiem. W tym celu dostosowano wspomniany w sekcji Główna pętla systemu walki algorytm petli gry na potrzeby trenowania modelu. Nowy algorytm pętli gry wygląda następująco:

1. Wywołuje się funkcja OnEpisodeBegin() i całe środowisko jest na nowo losowane, następnie ustawiana jest wartość gameState na START_SIDE
2. Wywołuje się StartSide(), które uzupełnia listę tur - turnQueue oraz listę indeksów postaci - moveQueue
3. Następuje aktualizacja currentIndex o pierwszy indeks z moveQueue, przenosząc ten index później na koniec listy moveQueue. Potem zmienia gameState na START_TURN
4. Wywołuje się StartTurn(), które ustawia wartość currentCharacter na aktualnie poruszającą się postać.
5. Jeśli ruch wykonuje gracz aplikacja losuje umiejętność i cel dla aktualnej postaci
6. Jeśli ruch wykonuje przeciwnik uruchamiana jest funkcja AIActions() która prosi o podjęcie decyzji trenowany model ML-Agents w wyborze celu oraz umiejętności. Po otrzymaniu wyniku wykonuje się funkcja EnvironmentStep()
7. Po wyborze umiejętności i celu wykonuje się funkcja UseSkill() uruchamiająca logikę systemu Press Turn
8. Na koniec gameState jest zmieniany na END_TURN
9. Wywołuje się EndTurn(), które sprawdza warunki wygranej, przegranej oraz limitu tur (jeśli któryś z warunków jest prawdziwy, przejdź do 12.)
10. Jeśli turnQueue jest puste, zmień stan na END_SIDE, w przeciwnym wypadku przejdź do punktu 3.
11. Wywołaj funkcję END_SIDE, która zmienia wartość isPlayerTurn na przeciwną sobie i zmienia wartość gameState na START_SIDE. Przejdź do punktu 2.
12. Wywołaj funkcję NotifyEndEpisode(), która daje nagrodę dla modelu i powoduje przejście do punktu 1.

Zmienne wybrane do uczenia

Zmiennych uczących jest łącznie 87. W ich skład wchodzi:

1. Ilość zwykłych tur
2. Ilość "wciśniętych" tur
3. Statystyki Aktora 1 - 17 zmiennych
4. Statystyki Aktora 2 - 17 zmiennych
5. Statystyki Aktora 3 - 17 zmiennych
6. Statystyki Aktora 4 - 17 zmiennych
7. Statystyki Przeciwnika - 17 zmiennych

Wszystkie zmienne są poddane normalizacji według wzoru:

$$normValue = \frac{value - minValue}{maxValue - minValue}$$

W przypadku gdy aktor nie został stworzony w danym epizodzie, wszystkie 17 przypisanych do niego zmiennych przyjmuje wartość 0.

3.2.2 UI

| Env 1 | Turns: 0 | Pressed Turns: 0 | Chosen Skill: 9 | Chosen Actor: 1 | | | | | | | | | | | | |
|-------|--------------|------------------|-----------------|------------------|--------------|-----------|---------|-------------|----------|-----------|------------|-----------|------------|------------|-------------|------------|
| Env 2 | Max HP: 14 | Current HP: 0 | Max MP: 9018 | Current MP: 9002 | Strength: 22 | Magic: 90 | Dex: 36 | Agility: 69 | Luck: 70 | Phy: - | Fire: - | Ice: abs | Elec: - | Wind: null | Light: rep | Dark: - |
| | 6187 | 456 | 5243 | 4787 | 5 | 25 | 23 | 61 | 9 | wk | abs | rep | - | abs | rep | null |
| | Max HP: 0 | Current HP: 0 | Max MP: 0 | Current MP: 0 | Strength: 0 | Magic: 0 | Dex: 0 | Agility: 0 | Luck: 0 | Phy: null | Fire: null | Ice: null | Elec: null | Wind: null | Light: null | Dark: null |
| | Max HP: 0 | Current HP: 0 | Max MP: 0 | Current MP: 0 | Strength: 0 | Magic: 0 | Dex: 0 | Agility: 0 | Luck: 0 | Phy: null | Fire: null | Ice: null | Elec: null | Wind: null | Light: null | Dark: null |
| | Max HP: 4855 | Current HP: 4780 | Max MP: 3415 | Current MP: 2507 | Strength: 91 | Magic: 33 | Dex: 18 | Agility: 12 | Luck: 82 | Phy: null | Fire: abs | Elec: str | Wind: rep | Light: rep | Dark: abs | str |
| | Reset Env | | | | | | | | | | | | | | | |

Rysunek 3.6. Widok UI do nadzoru trenowania modelu

Widok składa się z 2 elementów:

1. Listy środowisk
2. Panelu szczegółowego

Elementy te są ze sobą głęboko powiązane. Po naciśnięciu przycisku w liście środowisk, panel szczegółowy zmienia pokazywane dane na te z wybranego środowiska. Panel szczegółowy pokazuje wszystkie zmienne w nieznormalizowanej postaci. Są one ustawione w taki sam sposób jak to opisuje lista w sekcji Zmienne wybrane do uczenia. Ponadto wyświetlane są ostatnio wybrane wartości przez sztuczną inteligencję w postaci indeksów poszczególnych danych. Dostępny jest również przycisk restartu środowiska Reset Env, przywracający je do stanu początkowego.

4 Instrukcja instalacji i korzystania

4.1 Instalacja

4.1.1 Wymagane programy

Aby aplikacja była poprawnie uruchamiana potrzebne są określone komponenty. W przypadku wersji nowszych/starszych od podanych nie ma gwarancji prawidłowego działania aplikacji. Lista potrzebnych komponentów:

- Unity Hub - używany do instalacji silnika Unity
- Unity wersja 2022.2.11f1 - program Unity Hub sam zapyta o zainstalowanie tej wersji silnika podczas pierwszej próby uruchomienia projektu
- Python v3.9.13
- (opcjonalnie) Visual Studio 2022 lub Visual Studio Code - do edycji kodu źródłowego aplikacji

4.1.2 Konfiguracja wirtualnego środowiska Pythona

Wirtualne środowisko umożliwia instalację pakietów bezpośrednio w folderze z projektem, dzięki czemu może on zostać zabrany na przenośny dysk i uruchomiony na każdym komputerze posiadającym wymienione wcześniej komponenty. Poniższa instrukcja jest napisana z myślą o systemie operacyjnym Windows, gdyż na tym systemie aplikacja była tworzona i testowana. Aby skonfigurować wirtualne środowisko, należy:

1. Uruchomić Wiersz Poleceń (cmd) w folderze bazowym projektu
2. Sprawdzić za pomocą wiersza poleceń swoją wersję Pythona za pomocą polecenia:

```
python
```

jeśli jest poprawnie zainstalowany, zostanie otworzony program Python CLI pokazujący zainstalowaną wersję Pythona. Aby z niego wyjść należy użyć komendy:

```
exit()
```

3. Utworzyć wirtualne środowisko za pomocą komendy:

```
python -m venv venv
```

4. Aktywować wirtualne środowisko w wierszu poleceń za pomocą polecenia:

```
venv\Scripts\activate
```

5. Zainstalować narzędzie pip do wirtualnego środowiska odpowiadające za pobieranie pakietów Pythona wpisując:

```
python -m pip install --upgrade pip
```

6. Za pomocą narzędzia pip pobrać pakiet ML-Agents wpisując:

```
pip install mlagents
```

7. Zainstalować pakiety wchodzące w skład biblioteki PyTorch wpisując:

```
pip install torch torchvision torchaudio
```

8. Zmienić wersję pakietu protobuf, gdyż ta instalowana razem z ML-Agents nie działa poprawnie za pomocą komendy:

```
pip install protobuf==3.20.3
```

9. Pobrać pakiet ONNX odpowiadający za tworzenie plików przechowywujących modele wpisując polecenie:

```
pip install onnx
```

10. Sprawdzić, czy ML-Agents zostało poprawnie zainstalowane za pomocą komendy:

```
mlagents-learn -h
```

4.2 Korzystanie z aplikacji

4.2.1 Uruchomienie trenowania modelu

Pierwszym etapem jaki trzeba wykonać jest wytrenowanie modelu. Aby to zrobić trzeba wykonać następujące kroki:

1. Uruchomić projekt poprzez Unity Hub
2. Otworzyć scenę TrainingScene znajdującą się w folderze Assets/Scenes
3. Na panelu Hierarchy należy wyszukać obiekt Environments. Służy on do przechowywania Środowisk uczących model
4. Zklonować obiekt Environment będący dzieckiem obiektu Environments tyle razy, ile ma być środowisk uczących

5. Skonfigurować Środowiska. Obiekt `TrainingManager` służy do konfiguracji losowych statystyk postaci w środowisku, a `MLAgentsTrainer` ustawia wartości nagród za akcje
6. Uruchomić wiersz poleceń w folderze z projektem następnie aktywując wirtualne środowisko
7. Uruchomić ML-Agents za pomocą komendy:

```
mlagents -learn
```

Natomiast, jeśli już wcześniej model był uczony możemy zmusić ML-Agents do ponownego uczenia wpisując:

```
mlagents -learn --force
```

8. Gdy ML-Agents wyświetli komunikat w którym czeka na Unity, należy uruchomić scenę `TrainingScene` w Unity za pomocą funkcji `playtest`
9. Gdy będziemy uważać, że model jest dostatecznie wytrenowany, należy wpierw opuścić `playtest`, a następnie w zatrzymać uczenie w wierszu poleceń za pomocą skrótu `CTRL+C`. Po zatrzymaniu, `ml-agents` wyświetli nazwę oraz ścieżkę do pliku z rozszerzeniem `.onnx` w której model został zapisany

Wszystkie pliki wynikowe ML-Agents znajdują się w folderze `results/ppo/`. Jest możliwość uruchomienia aplikacji `Tensorboard` aby obserwować przebieg uczenia, lecz z uwagi na to, że pakiety wykorzystywane przez ML-Agents są przestarzałe, nie są kompatybilne z pakietami wymaganymi przez pakiet `Tensorflow` (potrzebny do działania aplikacji `Tensorboard`) wykorzystujący te same pakiety ale w nowszej wersji. Po próbach instalacji starszych wersji pakietów, nie udało się uzyskać działającej aplikacji `Tensorboard`, przez co finalnie nie jest dostępna w implementacji.

4.2.2 Uruchomienie symulacji

Po wytrenowaniu modelu, trzeba go przypisać do sztucznej inteligencji na scenie z symulacją. Aby to zrobić należy wykonać następujące kroki:

1. Skopiować plik z modelem o ścieżce podanej po zakończeniu uczenia do folderu `Brains` znajdującego się w katalogu `[Ścieżka Projektu]/Assets/Data/`
2. Otworzyć scenę `BattleScene` znajdującą się w folderze `Scenes`
3. Do obiektu `MLAgentsAI` należy przypisać wytrenowany model. Robi się to przeciągając z widoku folderów projektu plik o rozszerzeniu `.onnx` do właściwości `Model` obiektu `MLAgentsAI`

4. Otworzyć scenę MainMenu i uruchomić symulację

Po tych krokach można dostosować statystyki postaci gracza oraz przeciwnika i zasymulować walkę na danym modelu.

4.2.3 Sposób oceny modelu

Z uwagi na wcześniej wspomniany brak możliwości implementacji aplikacji Tensorboard, nie da się dokładnie sprawdzić jakości modelu. Można jednak sprawdzić czy model dokonuje poprawnych decyzji podczas różnych prób symulacji. W moim przypadku oceniałem czy model po zobaczeniu słabości na jakiś konkretny żywioł np. Ogień, posiadając też inne dostępne umiejętności będzie bardziej skłonny do atakowania tym typem obrażeń.

5 Wnioski

W mojej ocenie, oparcie sztucznej inteligencji w grach na przetrenowanych modelach jest w obecnych czasach możliwe. Problemem jest dodatkowe dotrenowanie lub całkowite trenowanie od podstaw modelu podczas trwania gry. ML-Agents wymaga uruchomionego środowiska Unity, które jest edytorem gry, aby móc trenować model co wyklucza taką możliwość. Jednak technologie uczenia maszynowego rozwijają się bardzo pręźnie i jak jeszcze w roku 2020 technologie uczenia maszynowego były domeną naukową, bardzo mało znaną poza tym kręgiem, to aktualnie są bardzo ważną i dynamicznie rozwijaną dziedziną. Może to spowodować na tyle duży rozwój tej technologii, że uczenie sztucznej inteligencji podczas gry będzie możliwe.

6 Bibliografia

- [1] Shagan Sah, *Machine Learning: A Review of Learning Types*, (2020 r.)
- [2] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, Joelle Pineau, *An Introduction to Deep Reinforcement Learning*, (2018 r.)
- [3] *Unity ML-Agents Toolkit Documentation* <https://unity-technologies.github.io/ml-agents/ML-Agents-Toolkit-Documentation/> (dostęp 06.01.2024r)
- [4] *Unity Documentation* <https://docs.unity.com> (dostęp 06.01.2024r)
- [5] *Godot Documentation* <https://docs.godotengine.org/en/stable> (dostęp 06.01.2024r)
- [6] Edward Beeching, Jilles Debangoye, Olivier Simonin, Christian Wolf, *Godot Reinforcement Learning Agents*, (2021 r.)
- [7] *Godot RL Agent Repository* https://github.com/edbeeching/godot_rl_agents (dostęp 06.01.2024r)
- [8] *Unreal Engine Documentation* <https://docs.unrealengine.com> (dostęp 06.01.2024r)
- [9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, *Proximal Policy Optimization Algorithms*, (2017 r.)
- [10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, (2018 r.)
- [11] Atlus, *Shin Megami Tensei III*, (2003 r.)
- [12] Andrew Zuo, *Guide On Using Unity ML Agents For A Turn Based Strategy Game*, (2021 r.), <https://andrewzuo.com/guide-on-using-unity-ml-agents-for-a-turn-based-strategy-game-9e52e6945a31> (dostęp 06.01.2024r)

Wyrażam zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW w tym w Archiwum Prac Dyplomowych SGGW.

.....
(czytelny podpis autora pracy)

