

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. Wytrenowany model ma sam podejmować decyzje o używanych akcjach w środowisku gry. 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 znajduje się podsumowanie, zawierające wyniki oraz wnioski wynikające z implementacji rozwiązania.

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 trained model is supposed to make decisions about the actions used in the game environment on its own. 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 a summary containing the results and conclusions drawn from the implementation of the solution.

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	12
2.2.4	Własny silnik z wykorzystaniem różnych pakietów do uczenia maszynowego	12
2.3	Algorytmy do uczenia maszynowego	13
2.4	Systemy walki w turowych grach RPG	13
2.4.1	RPG Maker	14
2.4.2	Darkest Dungeon	14
2.4.3	Active Time Battle	15
2.4.4	Press Turn System	15
3	Opis działania aplikacji	17
3.1	Aplikacja testująca model	17
3.1.1	Menu główne	17
3.1.2	Kreator postaci	19
4	Implementacja (OBSOLETE)	20
4.1	Część aplikacji testująca model	20
4.1.1	Scena MainMenu	20
4.1.2	Scena CharacterCreator	22
4.1.3	Scena BattleScene	25
4.1.4	Implementacja systemu Press Turn	27

4.2	Część aplikacji automatycznie trenująca model	30
4.2.1	Implementacja ML-Agents	30
4.2.2	UI	35
5	Instrukcja instalacji i korzystania	37
5.1	Instalacja	37
5.1.1	Wymagane programy	37
5.1.2	Konfiguracja wirtualnego środowiska Pythona	37
5.2	Korzystanie z aplikacji	38
5.2.1	Uruchomienie trenowania modelu	38
5.2.2	Uruchomienie symulacji	40
6	Podsumowanie	41
6.1	Przykładowy model wynikowy	41
6.1.1	Parametry uczenia	41
6.1.2	Przebieg uczenia oraz jego wyniki	42
6.1.3	Uruchomienie i analiza zachowania modelu	42
6.2	Wnioski	43
7	Bibliografia	44

1 Wstęp

Sztuczna inteligencja w ciągu ostatnich lat objęła wiele dziedzin życia i ciągle dynamicznie się rozwija. 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 tę technologię do generowania zadań, poziomów czy postaci [1], stworzenia systemu do wykrywania graczy używających programów oszukujących w grach wieloosobowych [2], oraz do stworzenia sztucznej inteligencji dla przeciwników gracza. Ostatnie z tych przykładowych zastosowań zostanie omówione pod względem teoretycznym i praktycznym zgodnie z tematem pracy.

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 walki². Wynikiem pracy będą dwie aplikacje: jedna trenująca model, a druga testująca model w postaci symulatora walki. Treść pracy będzie stanowić opisy technologii użytych w pracy wraz z alternatywami, opis implementacji rozwiązania na podstawie wybranych wcześniej technologii, instrukcję obsługi napisanej aplikacji, przykładowe wyniki powstałe w wyniku uruchomienia obu aplikacji oraz wnioski poimplementacyjne.

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.

²Turowy system walki - algorytm systemu walki, który zarządza kolejnością poruszania się jego uczestników przekazując możliwość wykonania akcji według uporządkowanej kolejności określonej przez ten system. Można go porównać do topologii sieciowej token ring, gdzie żądania w danym momencie może przekazywać tylko osoba posiadająca token (w przypadku turowego systemu walki token jest nazywany turą).

2 Technologie

W tym rozdziale zostaną szczegółowo omówione komponenty potrzebne do stworzenia całego tytułowego projektu. Opisane są również technologie, które nie zostały użyte w projekcie aby pokazać, że implementację można przeprowadzić na wiele różnych sposobów. 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 rozróżnić trzy jego główne rodzaje (w oparciu o drugi rozdział z [3]):

- Nadzorowane (ang. Supervised) - model jest trenowany na danych wejściowych oraz odpowiednich dla nich wartościach wynikowych. Na ich podstawie uczy się przewidywania wyniku dla nowych, nieznanych 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 agenta w wykreowanym otoczeniu. Agent wykonując akcje w otoczeniu, dostaje nagrody lub kary, które zależą od wyników jego akcji.

W przypadku tej pracy zostanie użyty rodzaj uczenia przez wzmacnianie, z uwagi na 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 jak gracz grający w grę, uczyć się wykonywania najlepszych możliwych akcji w danym momencie poprzez ingerencję w otoczenie za pomocą akcji. Większa ilość informacji na temat działania uczenia przez wzmacnianie jest zawarta w [4].

2.2 Silniki graficzne oraz pakiety do uczenia

Silnik graficzny to program odpowiedzialny za generowanie grafiki na monitorze.¹ W przypadku tworzenia gier, silnik graficzny służy do projektowania aplikacji oraz jej kompilacji. Pakiety do uczenia maszynowego to gotowe biblioteki służące do uczenia modeli sztucznej inteligencji. W przypadku tego projektu, pakiet do uczenia najlepiej wybrać w oparciu o używany silnik graficzny. Poniżej znajduje się lista niektórych silników graficznych wraz z kompatybilnymi pakietami do uczenia maszynowego.

2.2.1 Unity z pakietem ML-Agents

Wśród wymienionych gotowych pakietów, ML-Agents jest najstarszym rozwiązaniem (udostępnione w 2017r.²). Dostępne w pakiecie są dwa algorytmy uczenia przez wzmocnienie - PPO oraz SAC³. Pakiet posiada własną dokumentację [5]. Sam silnik Unity umożliwia tworzenie aplikacji z grafiką 2D oraz 3D, operuje na języku C# oraz ma rozbudowaną dokumentację [6].

2.2.2 Godot z pakietem RL Agents

Godot jest silnikiem OpenSource na licencji MIT. Według dokumentacji [7], silnik wspiera oficjalnie dwa języki programowania: autorski język GDScript oraz C#. Twórcy za pośrednictwem technologii GDExtension zapewnili możliwość stworzenia wsparcia dla innych języków przez każdego użytkownika tego silnika. 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 czterech innych pakietach⁴ do uczenia przez wzmocnienie oraz wspiera ponad 12 algorytmów uczenia. Więcej informacji o RL Agents można znaleźć w artykule naukowym poświęconym tej technologii [8] oraz w dokumentacji dostępnej w repozytorium projektu [9].

¹Definicja pochodzi ze źródła (stan na 06.01.2024r.): https://www.linfo.org/graphic_engine.html

²Repozytorium powstało 8.09.2017r, pierwsza wersja pakietu pochodzi z 16.09.2017r - link do repozytorium <https://github.com/Unity-Technologies/ml-agents>

³Więcej informacji o tych algorytmach w rozdziale „Algorytmy do uczenia maszynowego”

⁴Pakiety na których bazuje RL Agents - StableBaseLines3, SampleFactory, CleanRL oraz Ray rllib (stan na 06.01.2024r.)

2.2.3 Unreal Engine z pakietem Learning Agents

W porównaniu do innych pakietów, Learning Agents jest najmłodszy (powstał na początku 2023r.) oraz najslabiej udokumentowany. Pakiet operuje algorytmami PPO oraz BC (ang. Behavioral Cloning)⁵. Sam silnik graficzny Unreal Engine jest narzędziem z bardzo dużym wsparciem i możliwościami. Posiada wsparcie dla języka C++ oraz funkcji Blueprints umożliwiającej programowanie wizualne (bez pisania kodu). Dokumentacja zarówno silnika Unreal Engine jak i pakietu Learning Agents znajduje się w [10].

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 PyTorch⁶ lub też jeden z czterech pakietów wykorzystywanych przez RL Agents.

Wybrany silnik i pakiet do uczenia maszynowego

Do pracy wybrałem Unity z pakietem ML-Agents przez wzgląd na moje wcześniejsze doświadczenie z Unity i językiem 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.

⁵Informacja pochodzi ze źródła (stan na 06.01.2024r.): <https://dev.epicgames.com/community/learning/tutorials/80WY/unreal-engine-learning-agents-introduction>

⁶Używany w ML-Agents oraz Learning Agents

2.3 Algorytmy do uczenia maszynowego

Unity ML-Agents posiada dwa algorytmy uczące PPO (ang. Proximal Policy Optimization) oraz SAC (ang. Soft Actor-Critic). Pierwszy z nich jest podstawowym algorytmem używanym w pakiecie.

PPO jest algorytmem „on-policy”, kiedy SAC jest algorytmem „off-policy”. Samo określenie Policy odnosi się do strategii wyboru akcji przez agenta w aktualnym stanie środowiska. W skład tej strategii wchodzi zarówno wybór akcji agenta jak i aktualizacja modelu zależna od nagród i kar zwracanych przez środowisko po wykonaniu akcji.

Różnica między algorytmami „on-policy” i „off-policy” zależy od tego czy obie funkcjonalności w Policy są od siebie odseparowane. Jeśli Policy wybierające akcję agenta jest inne od Policy aktualizującego model to algorytm jest „off-policy”, natomiast jeśli Policy odpowiedzialne za wybór akcji jest takie same co aktualizujące model, to algorytm jest „on-policy”. Szczegółowe informacje na temat Policy znajdują się w [11].

Według [5] algorytm PPO jest algorytmem ogólnego przeznaczenia i jest bardziej stabilny w porównaniu z innymi algorytmami uczenia przez wzmocnienie. Dodatkowe informacje o algorytmie PPO znajdują się w [12], a o SAC w [13].

Do projektu wybrany został algorytm PPO z uwagi na wcześniej podaną stabilność oraz fakt, że jest on algorytmem będącym ponad 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, lecz większość gier opiera się o znane już systemy wprowadzając do nich tylko minimalne zmiany. Pewne elementy są jednak stałe i można je zauważyć w każdej grze z tego gatunku.

- Rozgrywka polega na wzajemnym przekazywaniu 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 zginą.

Powyższe punkty zostały oparte o własne doświadczenia z grami tego gatunku, jak i o wybrane rozdziały z [14, 15]. Dalej przedstawię parę różnych podejść do walki w systemie turowym. Niektóre zawierają nazwy gier lub aplikacji, z których zostały zaczerpnięte, przez brak możliwości odnalezienia źródła zawierającego ustandaryzowaną nazwę danego systemu.

2.4.1 RPG Maker

RPG Maker to program do tworzenia gier RPG nie wymagający programowania. System walki w nim użyty jest dobrą bazą do tworzenia innych, bardziej skomplikowanych systemów przez swoją prostotę. Na początku walki jest tworzona kolejka złożona z postaci występujących na scenie (zarówno wrogich jak i sojusznicznych), która jest posortowana według statystyki Szybkości. Im większa owa statystyka, tym postać jest wyżej w kolejce i szybciej wykona ruch. Po stworzeniu kolejki gracz wybiera ruchy wszystkich sojusznicznych postaci, a po wyborze ruchu ostatniej postaci system przechodzi przez kolejkę, uruchamiając wybrane przez gracza oraz sztuczną inteligencję przeciwników akcje. Po wszystkim, system daje graczowi ponowną możliwość wyboru umiejętności. System ten był również zaimplementowany w grach z serii Etrian Odyssey oraz pierwszych częściach Final Fantasy.

2.4.2 Darkest Dungeon

Darkest Dungeon jest grą autorstwa Red Hook Studios. System walki w niej przedstawiony jest zmodyfikowaną wersją poprzedniego systemu, jednak wprowadza większą dynamikę. Gdy kolejka postaci zostanie stworzona oraz postać w drużynie gracza ma aktualnie swoją turę, gracz wybiera akcję, którą postać ma wykonać. Po jej wybraniu, postać wykonuje akcję i przechodzi na koniec kolejki, przekazując turę następnej postaci się w niej znajdującej. System ten został również użyty w grach Yakuza: Like a Dragon, Like a Dragon: Infinite Wealth czy Super Mario RPG.

2.4.3 Active Time Battle

System walki charakteryzujący się tym, że każda postać na początku zaczyna z pustym paskiem tury, ładującym się w równych interwałach. Gdy tura jakiejś postaci zostanie w pełni naładowana, walka zatrzymuje się, a postacie z naładowaną turą wybierają akcję, jaką mają wykonać. Następnie pasek tury jest dalej ładowany. Gdy ładowanie dobiegnie do końca, postać wykona wybraną wcześniej akcję i resetuje ładowanie tury do wartości początkowej. System ten jest znany z gier Chrono Trigger, Child of Light oraz serii Final Fantasy. Dodatkowe informacje na temat systemu walki oraz jego implementacji znajdują się w [15, 16].

2.4.4 Press Turn System

Jest to nazwa systemu walki opracowanego przez firmę Atlus na potrzeby gry [17]. Opiera się na zwiększeniu znaczenia tury jako zasobu poprzez rozdzielenie tur na zwykłe i naznaczone, oraz wprowadzenie oceny ruchu postaci. Ocena opiera się na 2 powiązanych ze sobą kryteriach:

1. Rodzaj obrażeń - np. żywioły typu ogień, woda, ziemia itp.,
2. Relację postaci do użytego rodzaju obrażeń.

Pierwotna implementacja systemu zawiera 6 typów relacji do rodzaju obrażeń:

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

System na początku rozdziela fazę gracza oraz fazę przeciwnika. Start każdej fazy rozpoczyna się od liczenia liczby tur jaką posiada poruszając się strona. Jest to zazwyczaj liczba postaci po poruszającej się stronie, wykluczając martwe jednostki (istnieją wyjątki od tej reguły np. silniejsza, samotna jednostka otrzymuje jedną dodatkową turę). Poza liczeniem liczby tur jest tworzona kolejka postaci w której kolejności będą się one poruszać. Gdy postać aktualnie posiadająca ruch wykona akcję zaczyna się jej ocena, która zmienia ilość posiadanych przez stronę tur. W zależności od oceny tury mogą być odejmowane lubznaczane. Zasady oceny wyglądają następująco:

- Jeśli postać trafi w słaby punkt innej postaci lub trafi cios krytyczny, oraz posiada 1 lub więcej zwykłych tur, zwykła tura jestznaczana.
- Jeśli postać trafi w słaby punkt innej postaci lub trafi cios krytyczny, oraz posiada same turyznaczane, odejmowana jestznaczona tura.
- Trafienie niebędące trafieniem krytycznym lub trafienie w inny niż Słabość typ relacji, powoduje odjęcie tur o konkretną ich liczbę priorytetyzując turyznaczane.
- Jeśli postać postanowi przeczekać turę, system zachowuje się tak samo jak przy trafieniu krytycznym lub trafieniu w słaby punkt.

Gdy stronie wykonującej akcje skończą się tury, następuje zmiana stron. Więcej informacji o tym systemie walki można znaleźć w pracy licencjackiej [18].

Wybrany system walki

Do implementacji został wybrany system Press Turn. 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 Opis działania aplikacji

Projekt składa się z dwóch aplikacji:

- Testującej model (symulator walki)
- Uczącej model

Aplikacje nie są kompilowane do plików uruchomieniowych, gdyż działają na funkcji testowania gry w Unity. Funkcja ta jest wymagana przez pakiet ML-Agents do uczenia modelu. Powoduje to konieczność zainstalowania silnika Unity, aby móc uruchomić obie aplikacje.

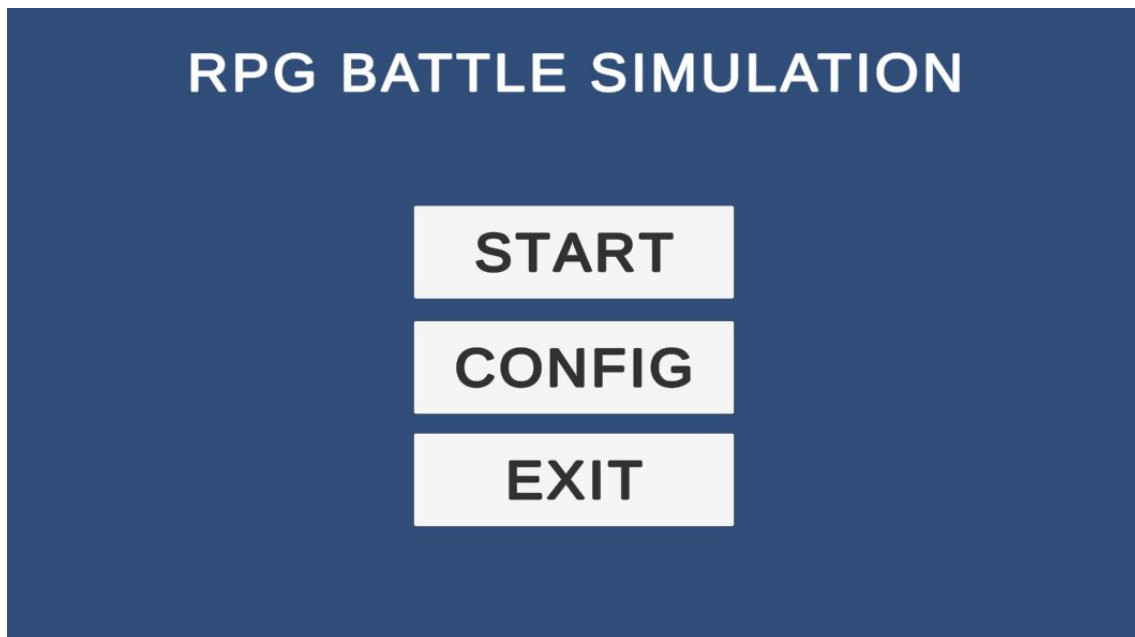
3.1 Aplikacja testująca model

Aplikacja testująca model została stworzona w postaci symulatora walki dla systemu Press Turn. Umożliwia ona dostosowanie statystyk postaci gracza jak i przeciwnika sterowanego przez wytrenowany model. Składa się z 3 widoków:

1. Menu głównego
2. Kreatora postaci
3. Symulatora walki

3.1.1 Menu główne

Zawiera przycisk przechodzący do kreatora postaci, przycisk umożliwiający wyjście z aplikacji oraz menu opcji w którym jest możliwość dostosowania głośności muzyki i dźwięków w symulatorze. Ustawienia graficzne nie były potrzebne w aplikacji przez uruchamianie jej w środowisku testowym Unity, które dostosowuje rozdzielczość do okna aplikacji.



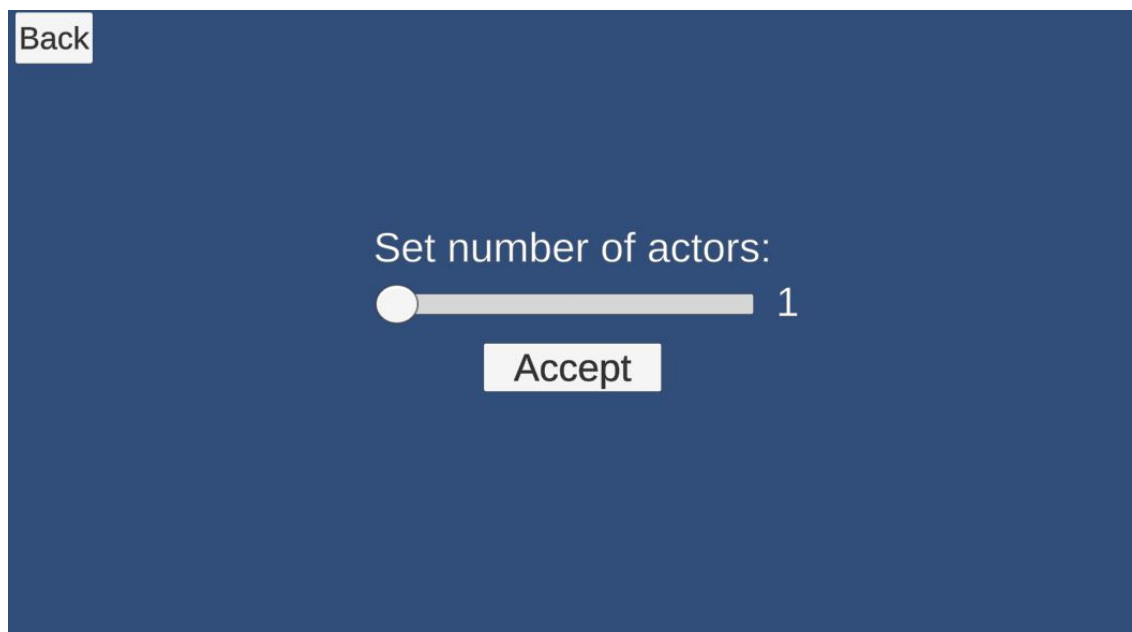
Rysunek 1. Widok menu głównego, *Źródło: własna inicjatywa*



Rysunek 2. Widok menu opcji, *Źródło: własna inicjatywa*

3.1.2 Kreator postaci

Kreator postaci umożliwia dostosowanie ilości postaci po stronie gracza oraz imienia i cech sojuszników i przeciwnika. Informacje o tym czym są, i za co odpowiadają cechy znajdują się w rozdziale „[tutaj odnośnik do highendowego opisu implementacji systemu Press Turn]”. Liczba przeciwników w każdej rozgrywce wynosi 1, aby wykluczyć zmieniającą się liczbę przeciwników, która może skomplikować proces uczenia.



Rysunek 3. Widok panelu wyboru liczby aktorów, Źródło: własna inicjatywa



Rysunek 4. Widok kreatora postaci, Źródło: własna inicjatywa

4 Implementacja (OBSOLETE)

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

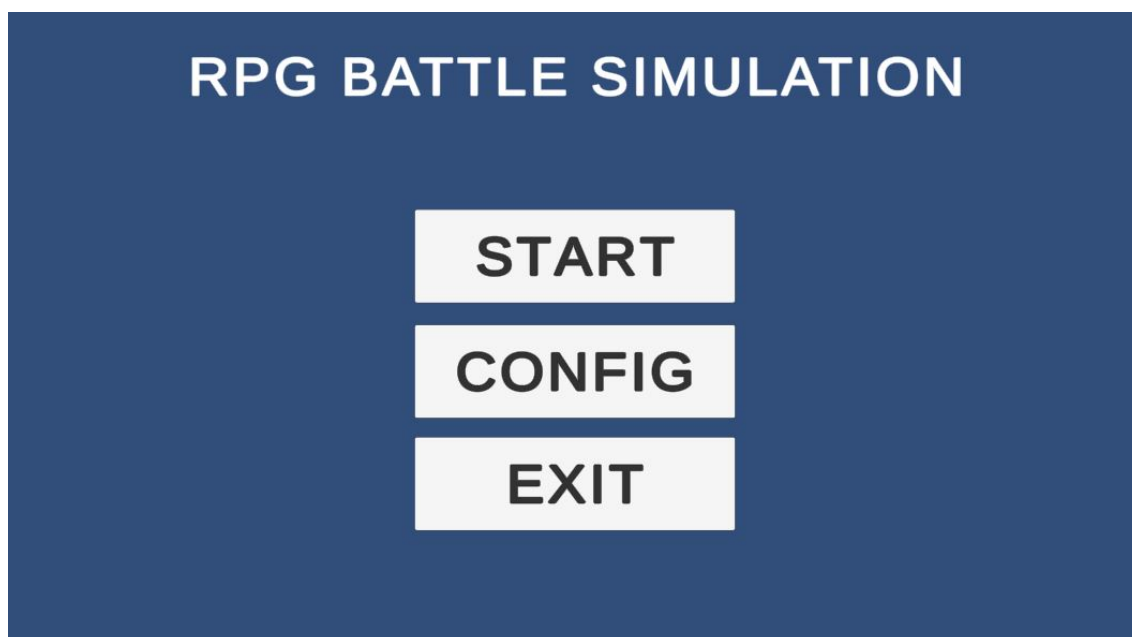
4.1 Część aplikacji testująca model

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

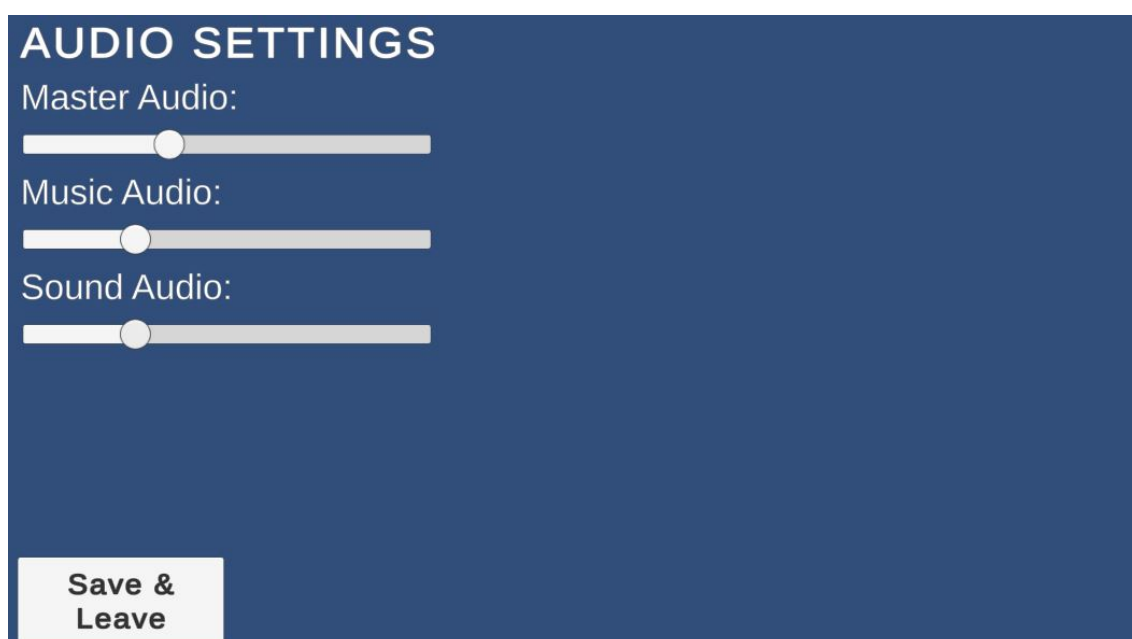
4.1.1 Scena MainMenu

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.

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



(a) Widok panelu głównego, Źródło: własna inicjatywa



(b) Widok opcji, Źródło: własna inicjatywa

Rysunek 1. Zrzuty ekranu przedstawiające menu główne

4.1.2 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.

Panel wyboru liczby aktorów

Panel składa się z suwaka przyjmującego wartości całkowite w zakresie $\langle 1, 4 \rangle$ 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 postaci²:

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

²Przeznaczenie statystyk zostanie wyjaśniony podczas omawiania sceny walki

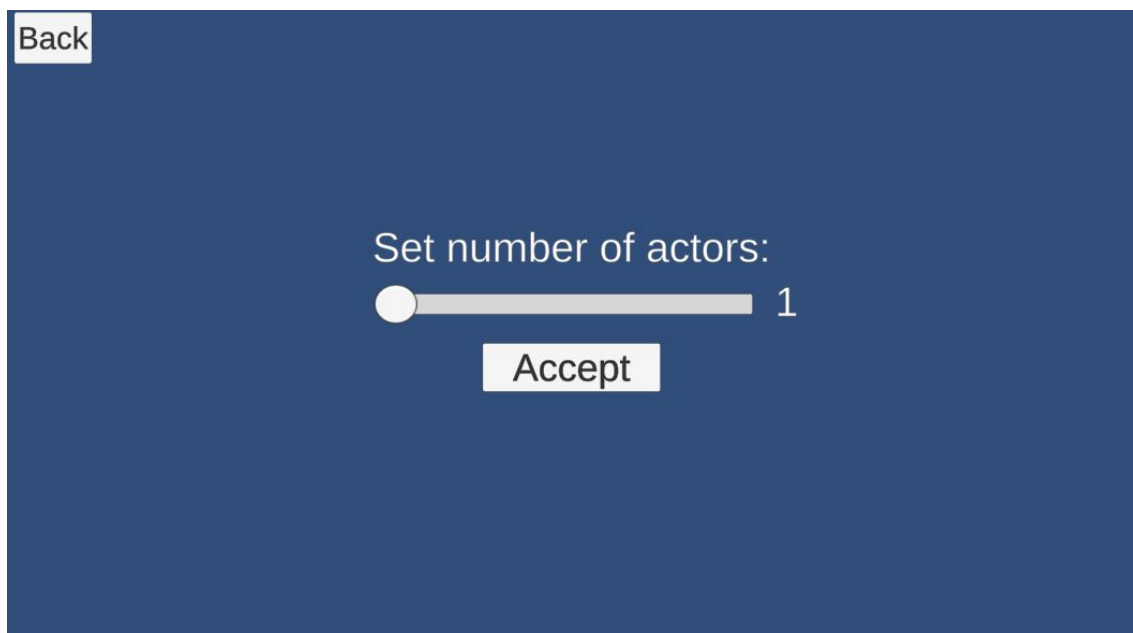
Postacie posiadają również Elementy i odpowiadające im relacje. Lista Elementów:

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: naznaczenie 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, Źródło: własna inicjatywa



(b) Widok kreatora postaci, Źródło: własna inicjatywa

Rysunek 2. Zrzuty ekranu przedstawiające kreator postaci

4.1.3 Scena BattleScene

Rozbudowana scena przedstawiająca informacje na temat aktualnego stanu walki. Stworzona na podstawie UI³ z gry [17]. Scena posiada następujące elementy UI:

- Przycisk Leave Battle - służy do wyjścia z walki,
- Lista tur - czerwone znaczniki oznaczają naznaczone 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 podczas walki,
- Stan drużyny gracza - pokazuje paski z imionami wszystkich postaci gracza, jak i ich stanem zdrowia i many.

³UI (ang. User Interface, pl. Interfejs Użytkownika) - Elementy wizualne umożliwiające interakcję użytkownika z aplikacją.



(a) UI sceny walki z gry Shin Megami Tensei III: Nocturne, Źródło: [17]



(b) UI sceny walki w projekcie, Źródło: własna inicjatywa

Rysunek 3. Zrzuty ekranu porównujące UI stworzone w projekcie do występującego w grze [17]

4.1.4 Implementacja systemu Press Turn

System Press Turn zaimplementowany w projekcie działa na 2 obiektach sceny 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 postacie 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.

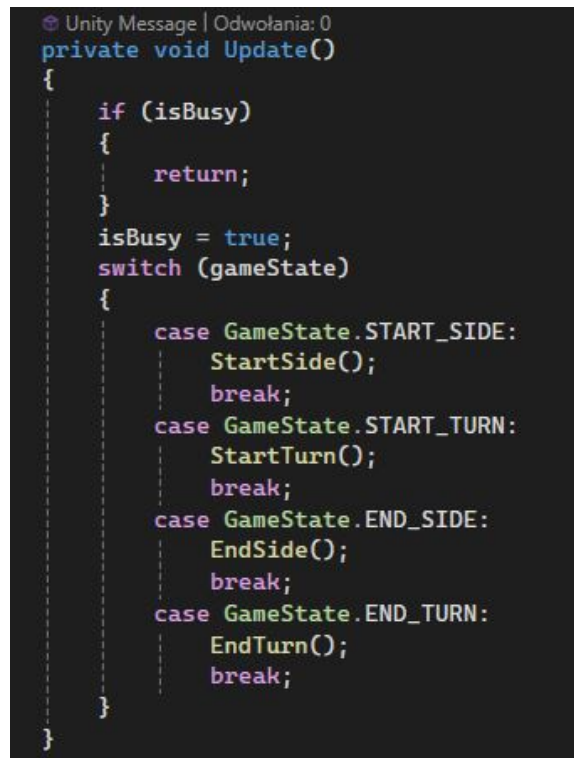
Po wczytaniu się sceny, BattleManager zostaje zainicjowany. Posiada on następujące wartości:

- selectedSkill - przechowuje informacje na temat wybranej przez postać umiejętności,
- isPlayerTurn - wartość prawda/fałsz mówiąca czy rusza się gracz, bazowo ustawiona na prawdę,
- isBusy - wartość prawda/fałsz odpowiadająca za przejście do następnego stadium walki, bazowo ustawiona 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, naznaczone 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 4. Funkcja Update w obiekcie BattleManager, Źródło: własna inicjatywa

Funkcja obsługuje tzw. pętlę gry czyli algorytm zapętlaający rozgrywkę. Pętla gry operuje na wywoływaniu 4 funkcji widocznych na Rysunek 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 rolę 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:

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

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

gdzie

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

Jedyną niewykorzystaną nigdzie statystyką jest `Luck`, z uwagi na brak planowanej wcześniejszej implementacji ciosów krytycznych w systemie walki. Statystyka `Agility` jest wykorzystywana do określenia kolejności w kolejce poruszania się postaci.

4.2 Część aplikacji automatycznie trenująca model

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

4.2.1 Implementacja ML-Agents

Aby zacząć uczyć model w technice uczenia przez wzmacnianie potrzebne jest 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 $\langle 1, \text{pointsMax} \rangle$,
- pointsMax - maksymalna wartość wykorzystywana do losowania HP oraz MP postaci. Może przyjąć wartość z zakresu $(\text{pointsMin}, 9999)$,
- statMin - minimalna wartość wykorzystywana do losowania statystyk postaci. Może przyjąć wartość z zakresu $\langle 1, \text{statMax} \rangle$,
- statMax - maksymalna wartość wykorzystywana do losowania statystyk postaci. Może przyjąć wartość z zakresu $(\text{statMin}, 99)$,
- minActors - minimalna wartość wykorzystywana do losowania ilości postaci po stronie gracza. Może przyjąć wartość z zakresu $\langle 1, \text{maxActors} \rangle$,
- maxActors - maksymalna wartość wykorzystywana do losowania ilości postaci po stronie gracza. Może przyjąć wartość z zakresu $(\text{minActors}, 4)$.

Agent

Implementacja oparta o artykuł [19]. Aby zaimplementować Agentą za pomocą pakietu ML-Agents należało stworzyć nową klasę dziedziczącą po klasie Agent oraz określić jakie wartości będzie on przewidywał. W przypadku tej pracy Agent będzie przewidywał dwie 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 nieużywanie 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 nieżyjących lub nieistniejących,
- void OnEpisodeBegin() - metoda przywraca stan początkowy środowiska w którym się znajduje.

```

Odwołania: 0
public override void Heuristic(in ActionBuffers actionsOut)
{ }
Odwołania: 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(), Źródło: własna inicjatywa

```

Odwołania: 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(), Źródło: własna inicjatywa

```

Odwołania: 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(), Źródło: własna inicjatywa


```

Odwołania: 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(), Źródło: własna inicjatywa

Rysunek 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() Agent. Powoduje to również konieczność zaimplementowania manualnego wykonywania kroku środowiskowego⁴. Dodatkowo do klasy zostały zaimplementowane dodatkowe metody oraz właściwości pomocnicze opisane w [19]. 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.

⁴Krok środowiskowy - naznacza moment zmiany wartości w środowisku

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 pętli 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 naznaczonych 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.

4.2.2 UI

Env 1

Env 2

Turns:0

Pressed Turns:0

Chosen Skill:9

Chosen Actor:1

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:-

Max HP:6187

Current HP:456

Max MP:5243

Current MP:4787

Strength:5

Magic:25

Dex:23

Agility:61

Luck:9

Phy:wk

Fire:abs

Ice:rep

Elec:-

Wind:abs

Light:rep

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: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

Ice:str

Elec:rep

Wind:rep

Light:abs

Dark:str

Reset Env

Rysunek 6. Widok UI do nadzoru trenowania modelu, *Źródło: własna inicjatywa*

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.

5 Instrukcja instalacji i korzystania

5.1 Instalacja

5.1.1 Wymagane programy

Aby aplikacja była poprawnie uruchamiana potrzebne są określone komponenty. W przypadku wersji innych 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.

5.1.2 Konfiguracja wirtualnego środowiska Pythona

Wirtualne środowisko umożliwia instalację pakietów bezpośrednio w folderze z projektem, dzięki czemu może on zostać zapisany na dysk przenośny 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
```

5.2 Korzystanie z aplikacji

5.2.1 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. Sklonować 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:

```
magents -learn
```

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

```
magents -learn --force
```

8. Gdy ML-Agents wyświetli komunikat w którym czeka na Unity, należy uruchomić scenę `TrainingScene` w silniku Unity za pomocą funkcji testowania sceny.
9. Gdy będziemy uważać, że model jest dostatecznie wytrenowany, należy wyłączyć testowanie sceny, a następnie zatrzymać uczenie w wierszu poleceń za pomocą skrótu klawiszowego `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/`. Podczas uczenia, w konsoli jest wypisywana wartość średnia oraz odchylenie standardowe nagrody jaką otrzymuje sztuczna inteligencja. Do dokładniejszej analizy uczenia modelu może posłużyć aplikacja `Tensorboard`, lecz z uwagi na to, że pakiety wykorzystywane przez ML-Agents są przestarzałe i nie są kompatybilne z tymi obsługującymi wyżej wymienioną aplikację, nie zostało to rozwiązanie zaimplementowane.

5.2.2 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 MIAgentsAI należy przypisać wytrenowany model. Robi się to przeciągając z widoku zasobów projektu plik o rozszerzeniu .onnx do właściwości Model obiektu MIAgentsAI.
4. Otworzyć scenę MainMenu i uruchomić symulację.

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

6 Podsumowanie

6.1 Przykładowy model wynikowy

6.1.1 Parametry uczenia

Przykładowy model został wytrenowany na 3 środowiskach uczących o takich samych wartościach nagród za akcje. Nagrody mają następujące wartości:

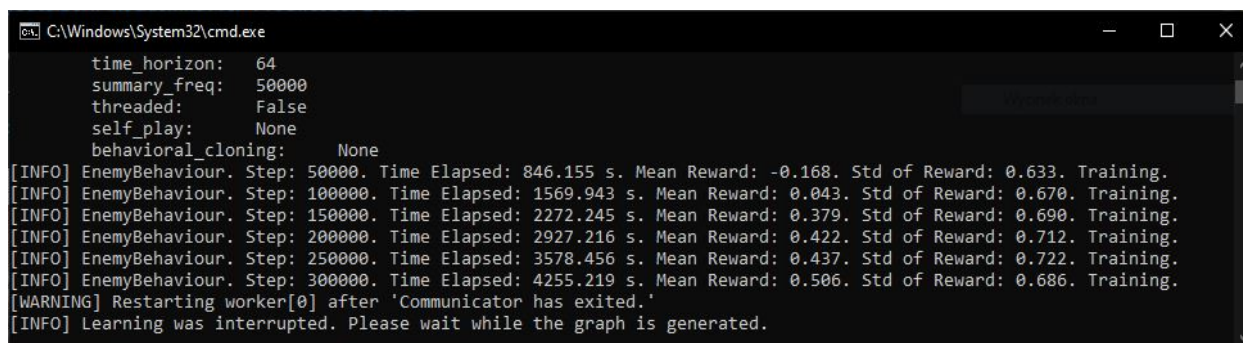
- Wygrana walka: 1.0,
- Przegrana walka: -1.0,
- Uderzenie w Element z relacją Neutralny: 0.2,
- Uderzenie w Element z relacją Słabość: 0.5,
- Uderzenie w Element z relacją Odporność: -0.1,
- Uderzenie w Element z relacją Niewrażliwość: -0.3,
- Uderzenie w Element z relacją Absorpcja: -0.5,
- Uderzenie w Element z relacją Odbicie: -0.5,
- Wyleczenie obrażeń: 0.3.

Środowiska zostały skonfigurowane według następujących parametrów:

- Punkty Życia oraz Many aktorów są losowane dla środowiska:
 1. z liczb o zakresie $\langle 50, 300 \rangle$,
 2. z liczb o zakresie $\langle 300, 500 \rangle$,
 3. z liczb o zakresie $\langle 500, 700 \rangle$,
- Statystyki aktorów są losowane dla środowiska:
 1. z liczb o zakresie $\langle 3, 20 \rangle$,
 2. z liczb o zakresie $\langle 25, 50 \rangle$,
 3. z liczb o zakresie $\langle 50, 75 \rangle$,
- Ilość aktorów jest w każdym środowisku losowana z liczb o zakresie $\langle 1, 4 \rangle$,
- Maksymalna ilość tur na epizod uczenia została ustawiona na 400 dla każdego środowiska.

6.1.2 Przebieg uczenia oraz jego wyniki

Uczenie zostało przeprowadzone na domyślnych hiperparametrach. Trwało ono 300 000 kroków co przełożyło się na czas 4255 sekund oraz wyniki w postaci średniej nagrody za akcję równej 0.506 oraz odchylenia standardowego o wartości 0.686.



```
C:\Windows\System32\cmd.exe
time_horizon: 64
summary_freq: 50000
threaded: False
self_play: None
behavioral_cloning: None
[INFO] EnemyBehaviour. Step: 50000. Time Elapsed: 846.155 s. Mean Reward: -0.168. Std of Reward: 0.633. Training.
[INFO] EnemyBehaviour. Step: 100000. Time Elapsed: 1569.943 s. Mean Reward: 0.043. Std of Reward: 0.670. Training.
[INFO] EnemyBehaviour. Step: 150000. Time Elapsed: 2272.245 s. Mean Reward: 0.379. Std of Reward: 0.690. Training.
[INFO] EnemyBehaviour. Step: 200000. Time Elapsed: 2927.216 s. Mean Reward: 0.422. Std of Reward: 0.712. Training.
[INFO] EnemyBehaviour. Step: 250000. Time Elapsed: 3578.456 s. Mean Reward: 0.437. Std of Reward: 0.722. Training.
[INFO] EnemyBehaviour. Step: 300000. Time Elapsed: 4255.219 s. Mean Reward: 0.506. Std of Reward: 0.686. Training.
[WARNING] Restarting worker[0] after 'Communicator has exited.'
[INFO] Learning was interrupted. Please wait while the graph is generated.
```

Rysunek 7. Wyniki wyświetlone w aplikacji konsolowej ML-Agents, *Źródło: własna inicjatywa*

6.1.3 Uruchomienie i analiza zachowania modelu

Na początku przypisałem model do sztucznej inteligencji przeciwnika oraz uruchomiłem symulator walki. Po uruchomieniu symulacji z różną konfiguracją umiejętności oraz liczbą aktorów w drużynie gracza, mogę wyodrębnić parę charakterystycznych zachowań:

- Model w pierwszej kolejności skupia się na używaniu umiejętności o typie obrażeń Nadzwyczajnym. Może to być spowodowane faktem, że obrażeń z tym typem obrażeń nie da się zablokować,
- Jeśli model nie posiada umiejętności o typie obrażeń Nadzwyczajnym, używa naprzemiennie leczenia oraz umiejętności ofensywnych. Rzadko używa umiejętności, które mogą odebrać tury jego stronie, jednak nie wykorzystuje też zbyt często tych, które dadzą mu dodatkowe naznaczone tury.
- Gdy model nie ma dostępu do umiejętności leczących ani o typie obrażeń Nadzwyczajnym, zaczyna priorytetyzować słabe punkty aktorów gracza. Jeśli w danym ruchu postanowi nie zaatakować, przeczeka turę aby zyskać naznaczoną turę.
- Jak model nie posiada umiejętności potrafiących uderzyć w słabe punkty, najczęściej przeczeka turę.

Wynikowy model nie jest idealny, lecz mimo to opanował on w zadowalającym stopniu zasady systemu walki. Wykorzystuje on mechanikę naznaczania tur oraz wybrał priorytetowe umiejętności zapewniające mu gwarantowane obrażenia, jak i leczenie ran.

6.2 Wnioski

Po implementacji tytułowej aplikacji oraz analizie wynikowego modelu uważam, iż jest w obecnych czasach możliwość wykorzystania technologii uczenia maszynowego w nowoczesnych grach. Wytrenowane wcześniej modele, mogłyby zastąpić manualne tworzenie sztucznej inteligencji dla przeciwników w grach, co jednocześnie skróciłoby czas ich produkcji. Chociaż wytrenowane wcześniej modele są realnym conceptem, który można zastosować już teraz, to modele trenowane na bieżąco podczas rozgrywki są w tym momencie bardzo ciężkie, lub niemożliwe do zaimplementowania na technologii ML-Agents. Jest to spowodowane wymaganymi komponentami, jakie muszą działać razem z samą grą, czyli program konsolowy uczący model w języku Python oraz środowisko testujące rozgrywkę silnika Unity. Na stan mojej obecnej wiedzy, bez tych dwóch komponentów nie da się uruchomić uczenia modelu. Problem pojawia się również w ilości obserwacji jakich potrzebuje sieć neuronowa, aby wyciągnąć wnioski z uczenia. Nie jest pewne, czy standardowa rozgrywka jest w stanie wyprodukować na tyle dużą ilość obserwacji dla sztucznej inteligencji, aby ta mogła nauczyć się podejmować dobre akcje w środowisku.

7 Bibliografia

- [1] Sebastian Risi, Julian Togelius *Increasing generality in machine learning through procedural content generation*, (2020 r.)
- [2] José Pedro Pinto, André Pimenta, Paulo Novais *Deep learning and multivariate time series for cheat detection in video games*, (2021 r.)
- [3] Shagan Sah, *Machine Learning: A Review of Learning Types*, (2020 r.)
- [4] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, Joelle Pineau, *An Introduction to Deep Reinforcement Learning*, (2018 r.)
- [5] *Unity ML-Agents Toolkit Documentation*, <https://unity-technologies.github.io/ml-agents/ML-Agents-Toolkit-Documentation/>, (dostęp 06.01.2024r)
- [6] *Unity Documentation*, <https://docs.unity.com/>, (dostęp 06.01.2024r)
- [7] *Godot Documentation*, <https://docs.godotengine.org/en/stable/>, (dostęp 06.01.2024r)
- [8] Edward Beeching, Jilles Debangoye, Olivier Simonin, Christian Wolf, *Godot Reinforcement Learning Agents*, (2021 r.)
- [9] *Godot RL Agent Repository*, https://github.com/edbeeching/godot_rl_agents/, (dostęp 06.01.2024r)
- [10] *Unreal Engine Documentation*, <https://docs.unrealengine.com/>, (dostęp 06.01.2024r)
- [11] Richard S. Sutton, Andrew G. Barto *Reinforcement Learning: An Introduction, second edition*, (2020 r.)
- [12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, *Proximal Policy Optimization Algorithms*, (2017 r.)
- [13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, (2018 r.)

- [14] SangGyu Nam, Kokolo Ikeda, *Generation of Diverse Stages in Turn-Based Role-Playing Game using Reinforcement Learning*, (2019 r.)
- [15] Ville Mäkelä, Albrecht Schmidt, *I Don't Care as Long as It's Good: Player Preferences for Real-Time and Turn-Based Combat Systems in Computer RPGs*, (2020 r.)
- [16] Agrivian Anditya, Agus Sihabuddin, *An Optimal Input for Role-Playing Game's Combat Pace using an Active Time Battle System Algorithm*, (2016 r.)
- [17] Atlus, *Shin Megami Tensei III: Nocturne*, (2003 r.)
- [18] Ryan Ganzke, *Designing a Battle Simulator For a Popular RPG Series*, (2023 r.)
- [19] 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)

