Politechnika Wrocławska

Wydział Informatyki i Zarządzania
kierunek studiów: Informatyka

Praca dyplomowa - inżynierska

Implementacja algorytmów sztucznej inteligencji w grze Starcraft: Brood War

Marcin Żerko

słowa kluczowe:
Algorytm genetyczny, sieć neuronowa
Strategia czasu rzeczywistego, gry komputerowe
Starcraft: Brood War

krótkie streszczenie:

Celem pracy jest implementacja algorytmów sztucznej inteligencji w grze Starcraft: Brood War. Zakres obejmuje przegląd możliwych technologii do wykorzystania w projekcie, ich implementacje, oraz badania ich skuteczności.

opiekun pracy dyplomowej	..... *Tytuł/stopień naukowy/imię i nazwisko*	..... *ocena*	..... *podpis*

Ostateczna ocena za pracę dyplomową			
Przewodniczący Komisji egzaminu dyplomowego	..... *Tytuł/stopień naukowy/imię i nazwisko*	..... *ocena*	..... *podpis*

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**

a) kategorii A (akta wieczyste)

b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)

** niepotrzebne skreślić*

pieczątka wydziałowa

Wrocław 2017

Streszczenie

Celem niniejszej pracy jest implementacja algorytmów sztucznej inteligencji w grze Starcraft: Brood War do zarządzania jednostkami podczas starcia. Został wykonany przegląd możliwych do wykorzystania technologii implementacji aplikacji, oraz technik sztucznej inteligencji. Na podstawie wybranych z nich został stworzony projekt aplikacji, wraz z wymaganiami i przypadkami użycia, a następnie jej implementacja w języku Java. Algorytmy genetyczne zostały użyte do nauki sieci neuronowych w stworzonym symulatorze. Na koniec zdefiniowano miary skuteczności, oraz zamieszczono wyniki badań skuteczności aplikacji.

Abstract

The aim of this thesis is the implementation of the artificial intelligence algorithms in the game of Starcraft: Brood War to micromanage units in battles. An overview of possible application implementation technologies and artificial intelligence techniques has been made. With the selected ones, the application design was created, along with the requirements and use cases, and then implemented in Java. Genetic algorithms were used for learning neural networks in the created simulator. At the end, the measures of effectiveness were defined, and the results of testing the effectiveness of the application were included.

Spis treści

1. Wstęp

Starcraft jest jedną z najpopularniejszych strategicznych gier czasu rzeczywistego (ang. *real time strategy*). Gra została wydana w 1998 roku na komputery osobiste przez firmę Blizzard Entertainment. Rok później wydany został dodatek do niej, o podtytule Starcraft: Brood War. Rozpowszechnił się on na tyle, że niemal zawsze jest dodawany do podstawowej gry i tej wersji dotyczy niniejsza praca. Gra odniosła ogromny sukces głównie w Korei Południowej i stworzyła się wokół niej duża społeczność esportowa.

1.1. Wprowadzenie do gry

Celem gry jest zniszczenie wszystkich budynków przeciwników. Bardzo często gracze poddają się wcześniej, gdy uznają, że w ich sytuacji wygrana jest już niemożliwa. Aby to osiągnąć należy budować własne struktury, rozwijać w nich różne pomocne technologie, trenować jednostki bojowe i tak nimi pokierować, aby wyeliminować obronę przeciwnika. Wszystko to kosztuje – walutą w grze są kryształy minerałów, oraz rzadszy, ale potrzebny do bardziej zaawansowanych operacji, gaz – wespan. Można go wydobywać przy pomocy jednostek zbierających, które są produkowane przy pomocy głównego budynku. W grze są dostępne trzy rasy, które różnią się sposobem rozgrywki, oraz wyglądem.

1.2. Geneza pracy

Starcraft jest grą wieloosobową i może być rozgrywany przeciwko innym graczom. Jednak nie zawsze jest taka możliwość, lub gracz może nie mieć na to ochoty. Wtedy na pomoc przychodzą boty, czyli algorytmy, często oparte na metodach sztucznej inteligencji, które zastępują ludzkiego przeciwnika. Te dołączone do instalacji gry są prymitywne i łatwo z nimi wygrać. W internecie jest dostępnych wiele innych botów, ale większość z nich skupia się wyłącznie na modyfikacji wysokopoziomowej strategii, pomijając kwestię taktycznego poruszania się jednostek. Przykładowo dzięki wycofywaniu pojedynczych rannych jednostek za linię stworzoną z pozostałych, jest możliwość uniknięcia jej śmierci – może ona wtedy wciąż zadawać pełne obrażenia przeciwnikom, będąc chroniona przez swoje jednostki sojusznicze, które przejmą na siebie ostrzał wroga i pozwolą jej przeżyć. W związku z tym postanowiłem połączyć dwa obszary swoich zainteresowań – sztuczną inteligencję oraz gry strategiczne czasu rzeczywistego – i spróbować wypracować własne rozwiązanie, które pozwoliłoby na skuteczne zarządzanie tymi to jednostkami w czasie rzeczywistym.

1.3. Cel i zakres pracy

Celem pracy jest implementacja algorytmów sztucznej inteligencji w grze Starcraft: Brood War do zarządzania jednostkami podczas starcia. Zakres obejmuje przegląd możliwych technologii do wykorzystania, ich implementacje, oraz badania ich skuteczności. W drugim rozdziale zostały opisane możliwe do wykorzystania technologie i podejścia, wraz z analizą istniejących rozwiązań. Rozdział trzeci zawiera informacje dotyczące projektu aplikacji, jego wymagań i przypadków użycia. W kolejnym rozdziale opisany jest sposób implementacji programu, diagram klas, oraz zostały zamieszczone widoki z aplikacji. Na aplikacje składają się samodzielnie stworzone implementacje sieci neuronowej, algorytmu genetycznego interfejsu użytkownika, oraz symulacji gry Starcraft. Następny rozdział składa się z definicji miary skuteczności i wyników badań zastosowanych algorytmów. Na koniec opisane zostało podsumowanie projektu, wraz z dalszymi możliwościami rozwoju.

2. Stan wiedzy i techniki w zakresie tematyki pracy

W poniższym rozdziale zostanie przedstawiony stan wiedzy i techniki – porównanie gry Starcraft do innych gier, przegląd możliwych do wykorzystania technologii implementacji, oraz możliwych do użycia technik sztucznej inteligencji. Na tej podstawie zostanie wykonany projekt aplikacji i jej implementacja.

2.1. Porównanie gry Starcraft do innych gier

W porównaniu do gier takich jak szachy, występują dwie podstawowe trudności. Po pierwsze - gra Starcraft jest rozgrywana w czasie rzeczywistym, a nie turowym. Powoduje to, że czas na obliczenia jest znacznie ograniczony. Program musi odpowiadać na ruchy przeciwnika jak najszybciej, gdyż każde opóźnienie może skutkować utratą jednostek w starciu, lub niedopasowaniem strategii do odpowiedzi przeciwnika, co w rezultacie zmniejsza szanse na wygraną w grze. Zaproponowane przeze mnie rozwiązanie tego problemu zostanie przedstawione w dalszym rozdziale.

Dodatkowo żadna ze stron nie posiada pełnej informacji o stanie gry. Nie wiedzą o poczynaniach przeciwnika dopóki jego jednostki nie wejdą w pole widzenia ich jednostek lub budynków. W związku z tym bot musi dostosowywać swój plan gry, do posiadanej przez niego wiedzy. Gdy już uda się zwiadowcom dowiedzieć o posunięciach przeciwnika, może wystąpić konieczność odpowiedniej modyfikacji naszych zachowań, aby nie zostać pokonanym. Niniejsza praca dotyczy wyłącznie zarządzania jednostkami w starciu, gdy już wiemy, jakie siły posiada przeciwnik, w związku z czym potencjalne możliwości modyfikacji strategii na podstawie odkrytych ruchów przeciwnika nie zostaną tutaj przedstawione.

2.2. Przegląd możliwych technologii implementacji aplikacji

Podstawową rzeczą, jaką trzeba wybrać przed implementacją programu, jest język programowania, jaki chcemy do tego celu użyć. Wiele metod opartych na sztucznej inteligencji wykorzystuje C++ - w tym także te związane ze Starcrafem. Jest to język ogólnego przeznaczenia, który między innymi dzięki możliwości bezpośredniego zarządzania pamięcią pozwala na bardzo efektywne wykorzystywanie zasobów sprzętowych. W związku z tym wymaga on od programisty więcej uwagi i ostrożności podczas implementacji, by uniknąć wycieków pamięci. Zdecydowałem się jednak używać do tego projektu bardziej znanego mi języka Java. Jest to język korzystający z maszyny wirtualnej - dzięki czemu wzrasta przenośność, co pozwoli mi w łatwy sposób uruchamiać aplikację zarówno na komputerach z systemem operacyjnym Windows, jaki i Linux. Posiada on też automat do zarządzania pamięcią (ang. *garbage collector*). Pozbywając się nieużywanych obiektów, pozwala na łatwiejszą implementacje aplikacji dla programisty. Wadą tego rozwiązania jest brak możliwości ręcznego zarządzania pamięcią, co może wpływać negatywnie na prędkość działania. Uznałem jednak, że potencjalne zalety C++ mogą zostać przeważone przez ewentualne błędy w mojej implementacji, ze względu na mój brak doświadczenia w pisaniu programów w tym języku.

Głównym źródłem informacji na temat zastosowania metod sztucznej inteligencji w Starcrafcie jest Starcraft AI Wiki [1]. Zawiera ona opis najpopularniejszych bibliotek, linki do dalszych stron, jak i wiele przydatnych porad dla osób próbujących swoich sił w tej dziedzinie. Najbardziej sprawiedliwym w stosunku dla ludzkich przeciwników bota, było by odczytywanie ekranu gry, analizowanie tego, co widzimy i odpowiednie wykonywanie akcji, tak żeby udawać manualne użycie klawiatury i myszki. Było by to jednocześnie dość trudne i wymagało dodatkowej implementacji. Zamiast tego lepszym rozwiązaniem wydaje się użycie odpowiedniej biblioteki, która udostępniłaby nam interfejs do otrzymywania danych z gry i wydawania rozkazów, dzięki czemu moglibyśmy pominąć ten problem. Jedynym kompletnym rozwiązaniem, który by nam na to pozwoliło jest BWAPI [2]. Aby gracz miał równe szanse, domyślnie nie są udostępniane żadne wiadomości na temat przeciwnika, których bot nie mógłby zobaczyć grając w normalny sposób. Są także blokowane potencjalne rozkazy wydawane manualnie, aby człowiek nie mógł w żaden sposób pomagać algorytmom sztucznej inteligencji. BWAPI nie wspiera najnowszej poprawki do gry – używa wersji 1.16.1 – rozwój Starcrafta był przez długi czas zawieszony i dopiero w tym roku został wznowiony. Aby dostosować się do zmian, musiałyby zostać zmienione konkretne offsety pamięci gry, z których korzysta do odczytywania i wydawania rozkazów. Było by to pracochłonne, a nowe wersje gry nie wprowadzają do niej istotnych z tego punktu widzenia poprawek.

BWAPI jest napisane w C++, dlatego niezbędne jest odpowiednie rozwiązanie, które pozwoliłoby użyć go od strony Javy. Istnieje kilka technologii pozwalających łączyć ze sobą te dwa języki programowania, jednak zazwyczaj do własnej implementacji wymagają pewnej znajomości C++. Jednym z aktualnych rozwiązań jest BWMirror [3]. Używanym w nim sposobem jest Java Native Interface [4]. Pozwala on na wykonywanie metod napisanych w obydwóch językach i przekazywanie pomiędzy nimi obiektów. Cechą tego rozwiązania jest to, że dane dotyczące parametrów jednostek i budynków wczytywane z gry są przechowywane po stronie kodu w części C++. W związku z tym nie jest możliwe stworzenie tylko i wyłącznie za pomocą Javy symulatora, który pozwoliłby trenować algorytmy sztucznej inteligencji bez uruchamiania gry. Inaczej jest w przypadku JNIBWAPI [5] – tutaj jest możliwe rozszerzenie biblioteki, aby dane te po uruchomieniu gry były zapamiętane do plików i następnie wczytywane w razie potrzeby, bez konieczności ponownego uruchamiana gry. W przeciwieństwie do BWMirror, nie wspiera ona najnowszej, czwartej wersji BWAPI, pozostając na wersji trzeciej.

Starcraft nie wspiera wielu rozwiązań, które mogłyby uprościć uczenie botów – takich jak możliwość uruchomienia gry bez interfejsu, lub przyśpieszenia jej do wyższej prędkości. Dodatkowo jest to aplikacja 32 bitowa, dostosowana pod system Windows 98, co ogranicza ilość wykorzystywanej pamięci i powoduje problemy z kompatybilnością na nowszych systemach, oraz na tych spoza rodziny Microsoftu. Nie pozwala także w prosty sposób na przygotowanie odpowiedniego scenariusza i przerwania go, będąc nastawiony na rozgrywanie całości partii. Kwestie te powodują, że odpowiedni symulator może znacznie przyśpieszyć proces uczenia bota. Takim właśnie rozwiązaniem jest Sparcraft – część bota o nazwie UAlbertaBot [6]. Jest on także napisany w C++, w związku, z czym niezbędny jest jego odpowiednik w Javie. Taka implementacja opierająca się na tym symulatorze, jednak niełącząca się z oryginalnym kodem, jak we wcześniej przywołanych przypadkach, nazywa się Jarcraft [7]. Niestety ma ona wiele wad. Po pierwsze kod po pobraniu z repozytorium nie kompiluje się. Jeden z plików gdzie jest przechowywana baza danych parametrów jednostek i budynków przekracza dopuszczalny rozmiar dla kompilatora Javy. Brakuje także jednego z plików źródłowych wykorzystywanego przez niego JNIBWAPI w nieokreślonej przez autora

wersji. Uruchomienie jej wymagało modyfikacji sposobu zapisu bazy danych do zewnętrznych plików, modyfikacji kodu symulatora na nowszą wersję JNIBWAPI i innych zmian, które utrudniał brak intuicyjności, dokumentacji i łamanie dobrych praktyk pisania kodu. Ponadto jest ona ograniczona, jeśli chodzi o poruszanie się jednostek – można zrobić to tylko o stałą odległość, w jednym z czterech podstawowych kierunków. Powodowało to, że wszystkie próby bardziej skomplikowanych taktyk niż zwykłe atakowanie najbliższej jednostki były skazane na porażkę. W związku z tym, nauczony doświadczeniem zdobytym z nią, postanowiłem napisać własny symulator.

2.3. Przegląd technik sztucznej inteligencji

Sieci neuronowe zostały zainspirowane biologicznymi sieciami neuronowymi, które występują w mózgach zwierząt. Składają się z neuronów i połączeń pomiędzy nimi. Neuron może przekazywać sygnał przez połączenia do innego neuronu, który przekaże na swoje wyjście zmodyfikowaną przez funkcję aktywacji sumę wartości wejść. Stan neuronu jest zwykle określany przez liczbę rzeczywistą. Każde połączenie ma swoją wagę, która określa jak mocny ma wpływ w porównaniu do innych. Zwykle są one też połączone w warstwy: wejścia, wyjścia i dowolną liczbę ukrytych. Jest możliwe stworzenie takiej sieci, która będzie zachowywała się tak jak bramka XOR. Bardziej skomplikowane zastosowania wymagają użycia dużej liczby neuronów, co zwiększa liczbę dodatkowych parametrów. Wszystkie one muszą być w jakiś sposób ustalone na odpowiednie wartości. Doskonali się je poprzez proces ich wielokrotnego strojenia i oceniania, który nazywamy uczeniem. Można je zrealizować na wiele sposobów. Należy pamiętać o zjawisku przeuczenia (ang. *overfitting*). Objawia się ono zbyt mocnym dopasowaniem do danych, na których zostały wyuczone i może mieć negatywny wpływ na zachowanie przy innych zestawach danych [8].

Algorytmy genetyczne także zostały zainspirowane naturalnymi procesami, tym razem ewolucją gatunków i ich dostosowaniem do warunków otoczenia. Początkowo generowana jest startowa populacja i każdy z osobników jest oceniany za pomocą funkcji przystosowania. Każdy osobnik ma swój genotyp, który opisuje jak jest on zbudowany. Następnie występuje selekcja z populacji i krzyżowanie ich genów, a kolejnym krokiem jest mutacja, gdzie geny poszczególnych odpowiedników zmieniają się. Tak zmieniona populacja zostaje ponownie oceniona i cykl powtarza się, aż do osiągnięcia określonego wcześniej warunku stopu. Efektem pracy tej metaheurystyki jest najlepszy osobnik, jakiego udało się jej odkryć. Są to mechanizmy pozwalające eksploatować pobliską przestrzeń rozwiązań w poszukiwaniu coraz to lepszego rozwiązania, oraz eksploracje, które ma na celu uniknięcie utknięcia w lokalnym optimum [9].

W swojej pracy postanowiłem wykorzystać sieci neuronowe, ponieważ raz nauczone, mogą w czasie rzeczywistym reagować na poczynania jednostek przeciwnika. Aby je do tego wyuczyć zdecydowałem się użyć do tego celu algorytmów genetycznych, które są znanym i stosowanym narzędziem do optymalizacji trudnych problemów obliczeniowych. Mimo ich braku wiedzy o dziedzinie problemu, wyszukują one rozwiązanie, w tym przypadku uczą sieć neuronową, a rozwiązaniem jest zestaw wag pomiędzy połączeniami w sieci neuronowej. Zdecydowałem się napisać implementacje samodzielnie, zamiast korzystać z gotowych bibliotek. Aby uprościć implementację, w moim rozwiązaniu topologia sieci neuronowej jest stała – w przeciwieństwie do niektórych rozwiązań, które potrafią dynamicznie dodawać, lub usuwać połączenia, czy też nawet całe neurony z sieci [10].

3. Założenia projektowe

W rozdziale tym zostanie zaprezentowany projekt aplikacji, który umożliwi jej implementację.

3.1. Wymagania funkcjonalne

Badania skuteczności zaimplementowanych przeze mnie algorytmów sztucznej inteligencji mogę przeprowadzić modyfikując sterujące nimi parametry bezpośrednio w kodzie. Jednakże takie rozwiązanie byłoby niewygodne i trudne w użyciu dla użytkowników, dlatego postanowiłem stworzyć także interfejs graficzny dla programu. Poniżej przedstawiam wymagania funkcjonalne, jakie ma spełniać przygotowana aplikacja:

- Możliwość ustawienia parametrów uczenia sieci neuronowej algorytmem genetycznym, aby można było dopasować jego działanie do własnych potrzeb
- Możliwość ustawiania maksymalnego czasu wykonania, aby można było wiedzieć, jaki jest maksymalny czas oczekiwania na nie
- Ustawienie maksymalnej liczby wyliczeń funkcji przystosowania, aby można było porównywać pomiędzy różnymi parametrami, nie opierając się na bardzo zmiennym czasie, który może zależeć w danym momencie od obciążenia systemu
- Uruchamianie procesu uczenia sieci neuronowej, aby uzyskać jak najlepszego osobnika
- Zapisywanie najlepszej wyuczonej sieci neuronowej do pliku, aby można było wykorzystać go w późniejszym czasie
- Generowanie wykresu minimalnej, średniej i maksymalnej oceny wśród osobników względem numeru generacji, aby można było zobaczyć wpływ parametrów na działanie uczenia sieci neuronowej
- Wypisanie oceny najlepszego uzyskanego osobnika
- Wczytanie parametrów sieci neuronowej z pliku, aby można było wykorzystać wcześniej wyuczoną sieć
- Uruchomienie symulacji i jej podgląd na żywo w oknie aplikacji, aby można było zobaczyć jej działanie w praktyce
- Wyświetlanie wybranych przez jednostkę decyzji, aby można było przeanalizować jak dana sieć neuronowa kieruje w starciu
- Aplikacja powinna nie pozwalać na uruchomienie uczenia z błędnymi parametrami, aby użytkownik nie mógł jej zablokować
- Aplikacja powinna zapisywać i wczytywać dane w formacie json, aby były możliwe do edycji także ręcznie

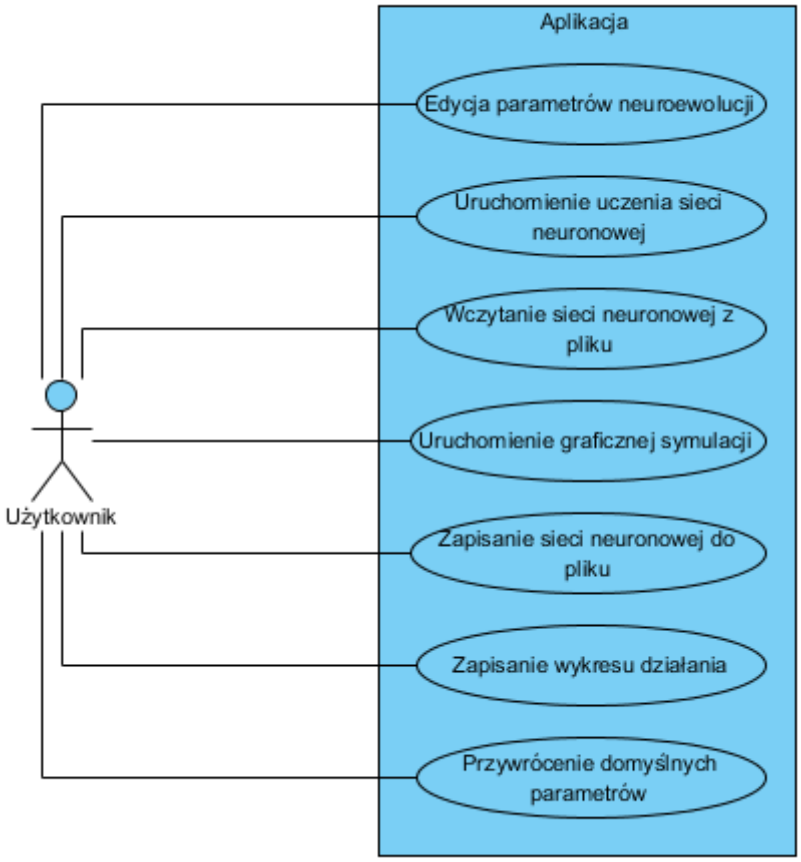
3.2. Wymagania niefunkcjonalne

Oprócz wymagań funkcjonalnych aplikacja powinna spełniać także pomocne wymagania niefunkcjonalne:

- Aplikacja powinna szybko reagować na komendy wydawane przez użytkownika i nie zawieszać interfejsu podczas symulacji
- Aplikacja powinna uruchamiać się zarówno na systemie Windows, jak i Linux
- Aplikacja powinna zawierać w sobie wszystkie potrzebne jej do działania biblioteki i zasoby, aby nie wymagała od użytkownika pobierania dodatkowych danych
- Aplikacja powinna być napisana z wykorzystaniem interfejsów, aby pozwolić na jej łatwą rozbudowę w przyszłości

3.3. Diagram przypadków użycia

Diagram przypadków użycia, jakie chciałbym zawrzeć w swoim programie wykonałem przy użyciu programu Visual Paradigm:



Rysunek 3.1: Diagram przypadków użycia

3.4. Przypadki użycia

Przypadek użycia: Edycja parametrów uczenia sieci neuronowej
Opis: Przypadek ten opisuje możliwość edycji parametrów uczenia sieci neuronowej, aby dostosować je do potrzeb użytkownika
Warunki wstępne: brak
Przebieg:

1. Aplikacja dostarcza domyślne parametry uczenia sieci neuronowej
2. Użytkownik edytuje wybrane przez niego parametry

Przypadek użycia: Przywrócenie domyślnych parametrów
Opis: Przypadek ten opisuje możliwość przywrócenia domyślnych parametrów uczenia sieci neuronowej, aby użytkownik mógł cofnąć swoje zmiany
Warunki wstępne: brak
Przebieg:

1. Użytkownik klika na przycisk przywrócenia domyślnych parametrów uczenia sieci neuronowej
2. Aplikacja przywraca parametry do stanu pierwotnego

Przypadek użycia: Uruchomienia uczenia sieci neuronowej
Opis: Przypadek ten opisuje możliwość uruchomienia uczenia sieci neuronowej, aby uzyskać jak najlepszego osobnika
Warunki wstępne: brak
Przebieg:

1. Użytkownik klika w przycisk odpowiedzialny za uruchomienie uczenia sieci neuronowej
2. Aplikacja sprawdza poprawność podanych parametrów
3. Aplikacja uruchamia uczenie sieci neuronowej
4. Aplikacja prezentuje wyniki użytkownikowi w nowym oknie dialogowym

Alternatywny przebieg:

2. Aplikacja wykrywa nieprawidłowe wartości parametrów i informuje o tym

Przypadek użycia: Wczytanie sieci neuronowej z pliku
Opis: Przypadek ten opisuje możliwość zapisania najlepszej wyuczonej sieci neuronowej do pliku, aby można było wykorzystać ją w późniejszym czasie
Warunki wstępne: brak
Przebieg:

1. Użytkownik klika w przycisk odpowiedzialny za wczytanie sieci neuronowej z pliku
2. Aplikacja wyświetla okno z wyborem pliku
3. Użytkownik wybiera i potwierdza wybór pliku
4. Aplikacja wczytuje sieć neuronową z pliku

Alternatywny przebieg:

3. Użytkownik rezygnuje z wyboru
4. Aplikacja wyświetla błąd, jeżeli wczytywanie nie powiodło się

Przypadek użycia: Zapisanie sieci neuronowej do pliku
Opis: Przypadek ten opisuje możliwość edycji parametrów uczenia sieci neuronowej
Warunki wstępne: sieć neuronowa została wczytana lub wyuczona
Przebieg:

1. Użytkownik klika w przycisk odpowiedzialny za zapisanie sieci neuronowej do pliku
2. System wyświetla okno z wyborem miejsca zapisu pliku i domyślnymi wartościami
3. Użytkownik wybiera i potwierdza wybór miejsca zapisu
4. System zapisuje sieć neuronową do pliku

Alternatywny przebieg:

3. Użytkownik rezygnuje z wyboru
4. Aplikacja wyświetla błąd, jeżeli zapisywanie nie powiodło się

Przypadek użycia: Uruchomienie graficznej symulacji
Opis: Przypadek ten opisuje możliwość uruchomienia symulacji i jej podgląd na żywo w oknie aplikacji, aby można było zobaczyć jej wyniki w praktyce i przeanalizować zachowanie jednostek
Warunki wstępne: sieć neuronowa została wczytana lub wyuczona
Przebieg:

1. Użytkownik klika w przycisk odpowiedzialny za uruchomienie graficznej symulacji
2. System prezentuje graficzną symulacje działania sieci neuronowej

Przypadek użycia: Zapisanie wykresu działania do pliku
Opis: Przypadek ten opisuje możliwość zapisania wykresu działania uczenia sieci neuronowej do pliku, aby można było go przeanalizować
Warunki wstępne: sieć neuronowa została wyuczona
Przebieg:

1. Użytkownik klika w przycisk odpowiedzialny za zapisanie sieci neuronowej do pliku
2. System wyświetla okno z wyborem miejsca zapisu pliku i domyślnymi wartościami
3. Użytkownik wybiera i potwierdza wybór miejsca zapisu
4. System zapisuje sieć neuronową do pliku

Alternatywny przebieg:

3. Użytkownik rezygnuje z wyboru
4. Aplikacja wyświetla błąd, jeżeli zapisywanie nie powiodło się

4. Wybrane aspekty implementacji aplikacji

Zgodnie z przedstawionym w poprzednim rozdziale projektem została wykonana aplikacja. Wybrane aspekty jej implementacji zostaną przedstawione w poniższym rozdziale.

4.1. Sposób realizacji implementacji

Program został wykonany w języku Java, w wersji numer 8. Postanowiłem jednak nie korzystać ze wszystkich nowinek udostępnionych przez tą aktualizację. Nie zostały wykorzystane strumienie (ang. *stream*), ze względu na obawy o wydajność aplikacji. Pozwalają one na wprowadzanie elementów języków opartych na paradygmacie funkcyjnym. Dzięki temu można tworzyć bardziej czytelny kod i uniknąć niezbędnego w innym przypadku wielokrotnego zagnieżdżenia kolejnych pętli. Niestety wprowadza to także narzut, który spowolniłby proces wyuczania sieci neuronowej. Drugą nowinka, która nie została wykorzystana, to klasa Optional. Pozwala ona na zachowanie większego bezpieczeństwa w przypadku, gdy dana zmienna mogłaby być nullem – zamiast tego otrzymujemy obiekt Optional.empty z wygodnymi metodami isPresent() i get() do sprawdzania oraz pobierania wartości. Nie została także użyta żadna biblioteka do klonowania obiektów i wszystkie metody do tego służące zrobiłem ręcznie, co znacząco poprawiło wydajność.

Jako zintegrowane środowisko programistyczne postanowiłem wybrać IntelliJ IDEA firmy JetBrains [11]. Jest to obecnie jedno z najbardziej zaawansowanych i popularnych IDE. Jego zdecydowaną zaletą są rozbudowane funkcje podpowiadania, oraz uzupełniania kodu, jak i duże wsparcie dla ewentualnego refactoru. Kod jest nieustannie analizowany przez IntelliJ, aby wsparcie to mogło być jak najbardziej dostosowane do znaczenia aktualnie pisanego kodu. Jakość tego programu podkreśla też fakt, że został on wybrany przez Google, aby jego odmiana została nowym IDE dla tworzenia aplikacji na system operacyjny Android.

Jako system kontroli wersji został wybrany Git [12]. Jest to rozproszony system, który pozwala na współpracę wielu deweloperów nad jednym kodem. Umożliwiało to bezproblemową pracę na kilku różnych komputerach, oraz rozgałęzianie i wersjonowanie kodu aplikacji. Narzędzie to posiada interfejs dostępny jedynie z linii komend, lecz istnieje wiele różnych nakładek ułatwiających korzystanie z niego, takich jak Git Extensions [13].

4.2. Wykorzystywane biblioteki

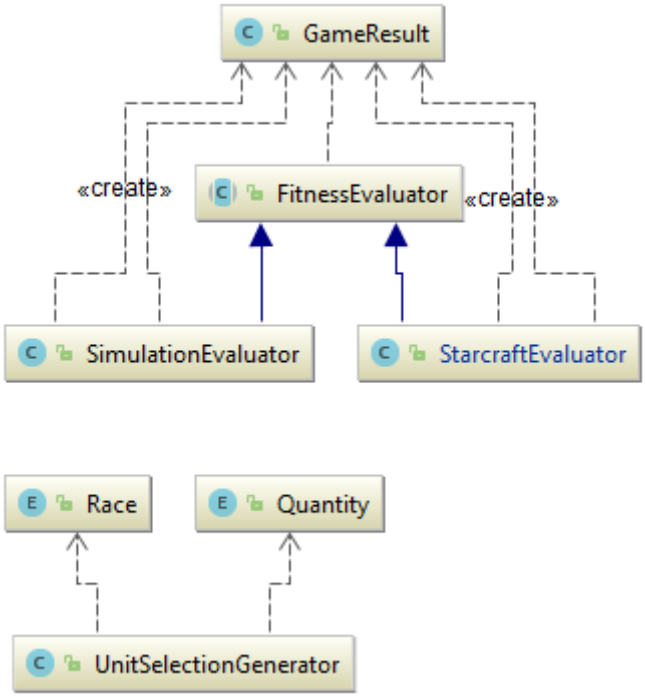
Jednym z zestawów bibliotek, jakie wykorzystałem przy implementacji swojego programu, są trzy biblioteki od firmy Apache z pakietu Apache Commons: Apache IO, Apache Math i Apache Lang [14]. Pierwsza z nich ułatwia zapisywanie plików, druga udostępnia funkcje matematyczne, które były przydatne przy implementacji sieci neuronowych, a trzecia dodatkowe metody do generowania losowych wartości. Są to biblioteki rozszerzające możliwości języka w dość standardowy sposób i mają bardzo dużo zastosowań.

Drugim zestawem bibliotek, jest Gson i Gson Extras [15]. Zostały one wykonane przez Google. Pierwsza z nich pozwala na bardzo łatwą serializacje i deserializacje obiektów do formatu Json. Jej użycie pozwala w większym stopniu ograniczyć modyfikacje w kodzie niż konkurencyjne biblioteki takie jak Jackson. W większości wypadków nie wymaga do działania publicznych konstruktorów, czy dodatkowych adnotacji. Dodatkową biblioteką jest Gson Extras, które jest zbiorem eksperymentalnych funkcji, które nie są jeszcze dostępne w podstawowej wersji biblioteki. W tym projekcie użyłem jej do ułatwienia serializacji różnych podtypów tej samej klasy.

Trzecim zestawem bibliotek jest JUnit [16] i Hamcrest [17]. Pierwsza z nich jest najpopularniejszą biblioteką do pisania testów jednostkowych w Javie i standardem wspieranym przez prawie każde środowisko programistyczne. Druga z nich udostępnia wiele własnych matcherów, które pozwalają na łatwiejsze pisanie bardziej czytelnych asercji do testów. Wykonałem przy ich pomocy testy jednostkowe klas dotyczących sieci neuronowych, ponieważ jest to potencjalnie część kodu, w którym mogą powstać najtrudniejsze do wykrycia i naprawienia błędy.

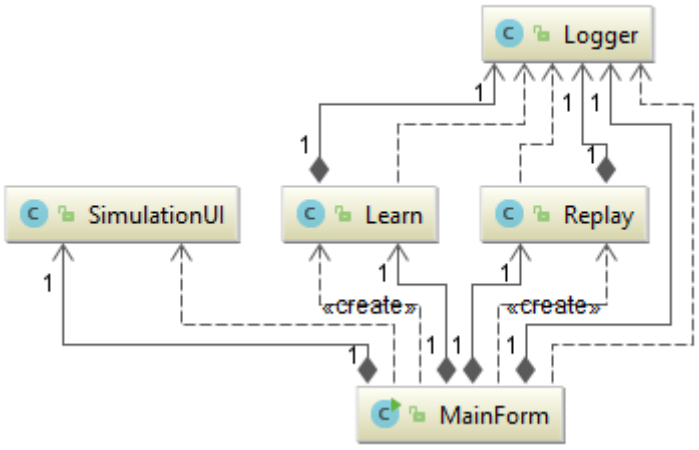
Ostatnią biblioteką, jaką wykorzystałem jest JFreeChart [18]. Jest to biblioteka pozwalająca w łatwy sposób oprogramować rysowanie wykresów. Możemy je potem zapisywać do pliku, czy też pokazywać na bieżąco użytkownikowi. Jest wiele możliwych typów wykresów, jednak ja zdecydowałem się na wykresy liniowe, na których łatwo można zobaczyć zmiany oceny populacji w zależności od czasu i dzięki temu przeanalizować zachowanie użytych rozwiązań.

4.3. Struktura projektu



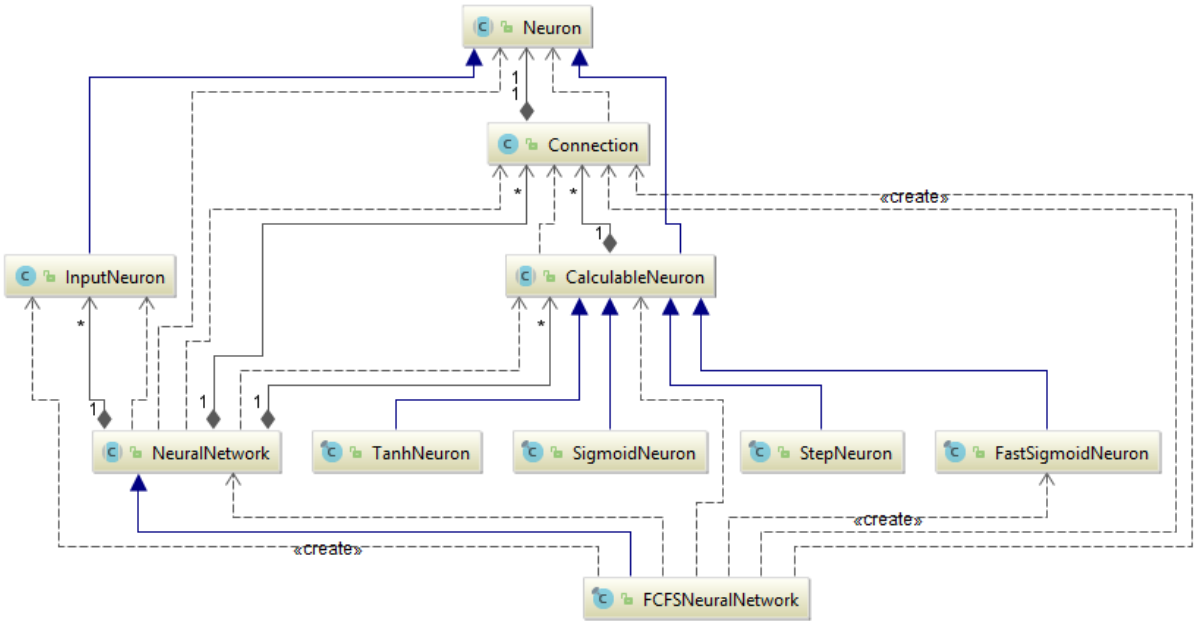
Rysunek 4.1: Pakiet fitness evaluator

FitnessEvaluator to interfejs, który służy do obliczania oceny danego osobnika. Powstały dwie klasy implementujące go, jedna używająca stworzonego symulatora, a druga gry Starcraft. UnitSelecionGenerator to klasa generująca ustawienia jednostek do symulatora, są także dwa pomocnicze typy wyliczeniowe określające liczbę jednostek i ich rasę. Klasa GameResult to klasa pomocnicza, służącą do przechowywania wyników gry



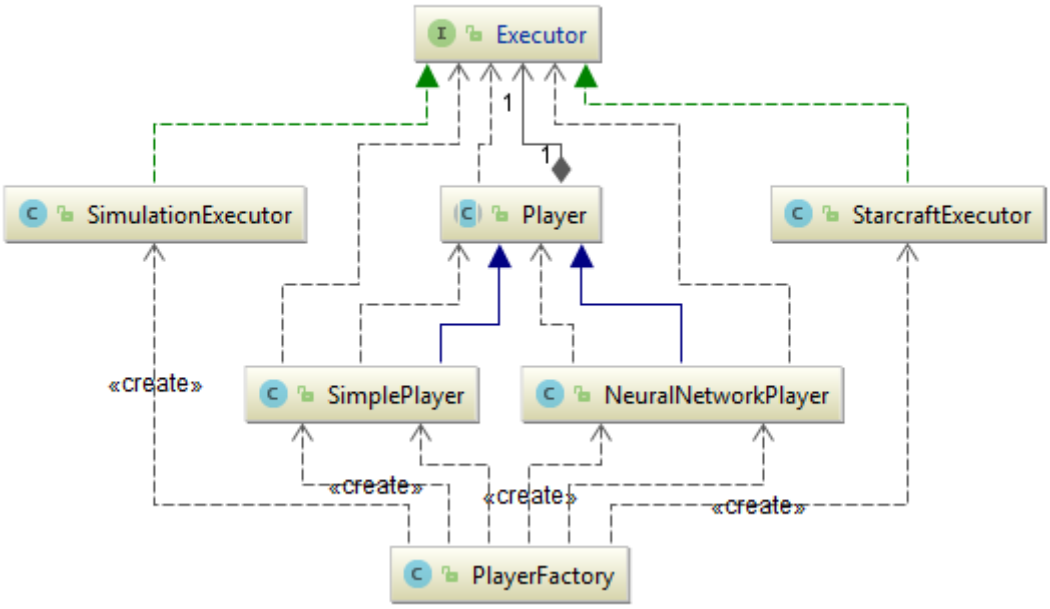
Rysunek 4.2: Pakiet gui

Klasa MainForm jest główną klasą w aplikacji i został zaimplementowany w niej interfejs graficzny. Towarzyszy jej plik MainForm.form, który jest XMLem odpowiadającym za rozmieszczenie, wygląd i zachowanie elementów interfejsu. Klasa Logger zajmuje się logowaniem i wyświetlaniem wiadomości z aplikacji w oknie po prawej stronie. SimulationUI to stworzony przeze mnie komponent graficzny pozwalający wyświetlać na bieżąco wyniki symulacji. Klasy Learn i Replay zarządzają procesem nauki i ponownego uruchomienia uczenia sieci neuronowej.



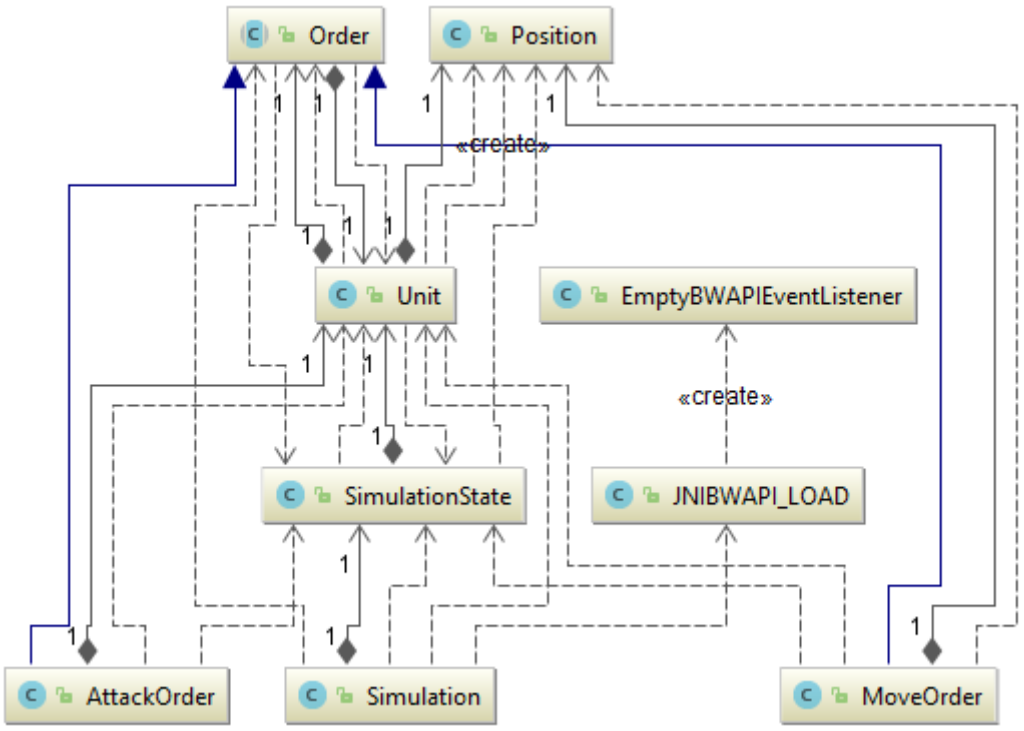
Rysunek 4.3: Pakiet neural network

NeuralNetwork jest klasą abstrakcyjna zawierającą już w sobie przydatne metody do implementacji własnych sieci neuronowych. FCSNeuralNetwork to skrótowiec, od Fully Conntected Sigmoid Neural Network, czyli sieci neuronowej, w której funkcją aktywacji jest sigmoid, a każdy neuron jest w pełni połączony z każdym innym neuronem sąsiedniej warstwy. Podobna sytuacja jest z klasą Neuron i rozszerzająca klasą CalculableNeuron – jest ona w zasadzie przeciwieństwem InputNeuron, który jest neuronem z podaną z zewnątrz wartością, a nie obliczaną z poprzednich neuronów. StepNeuron, jako funkcji aktywacji używa funkcji kroku, FastSigmoidNeuron szybszej wersji funkcji sigmoid, a TanhNeuron tangensa hiperbolicznego. Są one nieużywane obecnie, jednak pozostały one w kodzie aplikacji, do łatwiejszego późniejszego wykorzystania



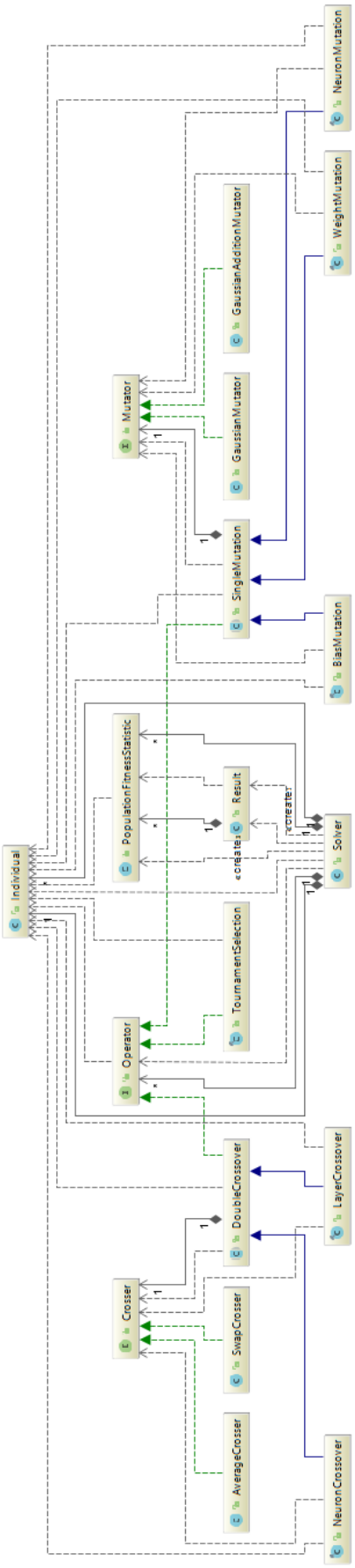
Rysunek 4.4: Pakiet player

Jest to pakiet zawierająca klasę abstrakcyjną player i rozszerzające je klasy. SimplePlayer jest atakującym najbliższą jednostkę graczem, do którego porównujemy taktyki, a NeuralNetworkPlayer jest wykorzystującym sieci neuronowe graczem. Interfejs Executor jest wykorzystywany przez te klasy do wykonywania akcji po stronie stworzonej symulacji, lub gry Starcraft. Została stworzona także klasa pomocnicza PlayerFactory, która uproszcza tworzenie odpowiednich konfiguracji gracy.



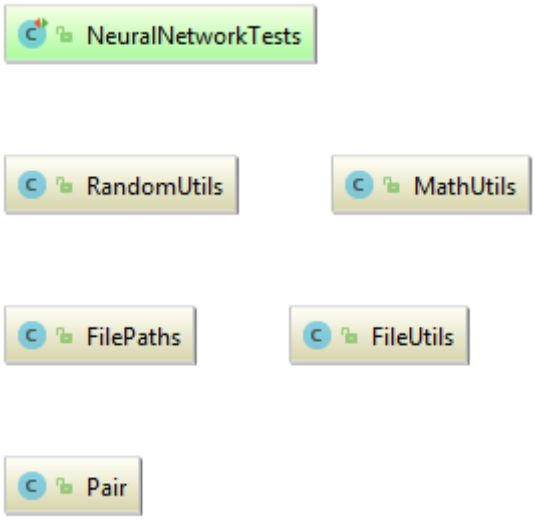
Rysunek 4.5: Pakiet simulation

Jest to paczka, w której znajdują się pliki służące do symulacji pola bitwy. JNIBWAPI_LOAD i EmptyBWAPIEventListener to klasy potrzebne do komunikacji z JNIBWAPI, zaś w order znajduje się klasa abstrakcyjna rozkazu i rozszerzające je klasy move order i attack order, które pozwalają na wydawanie rozkazów jednostkom. Position to klasa, w której przechowywane są współrzędne jednostki, reprezentowanej przez klasę Unit. Simulation zaś jest główną klasa, która pozwala przeprowadzić symulację, a SimulationState reprezentuje jej stan w danym momencie w czasie.



Rysunek 4.6: Pakiet solver

Klasa solver przez odpowiedni dobór operatorów pozwala na uruchomienie algorytmu genetycznego i zaprezentowanie jego wyników. Jest on jednak napisany generycznie i pozwala nam wykorzystywać także inne metaheurystyki, takie jak tabu search, czy symulowane wyżarzanie. Należałoby tylko napisać nowe operatory, które by je realizowały. Zawarte są różne operatory dla algorytmu genetycznego, które pozwalają na duży ich dobór.

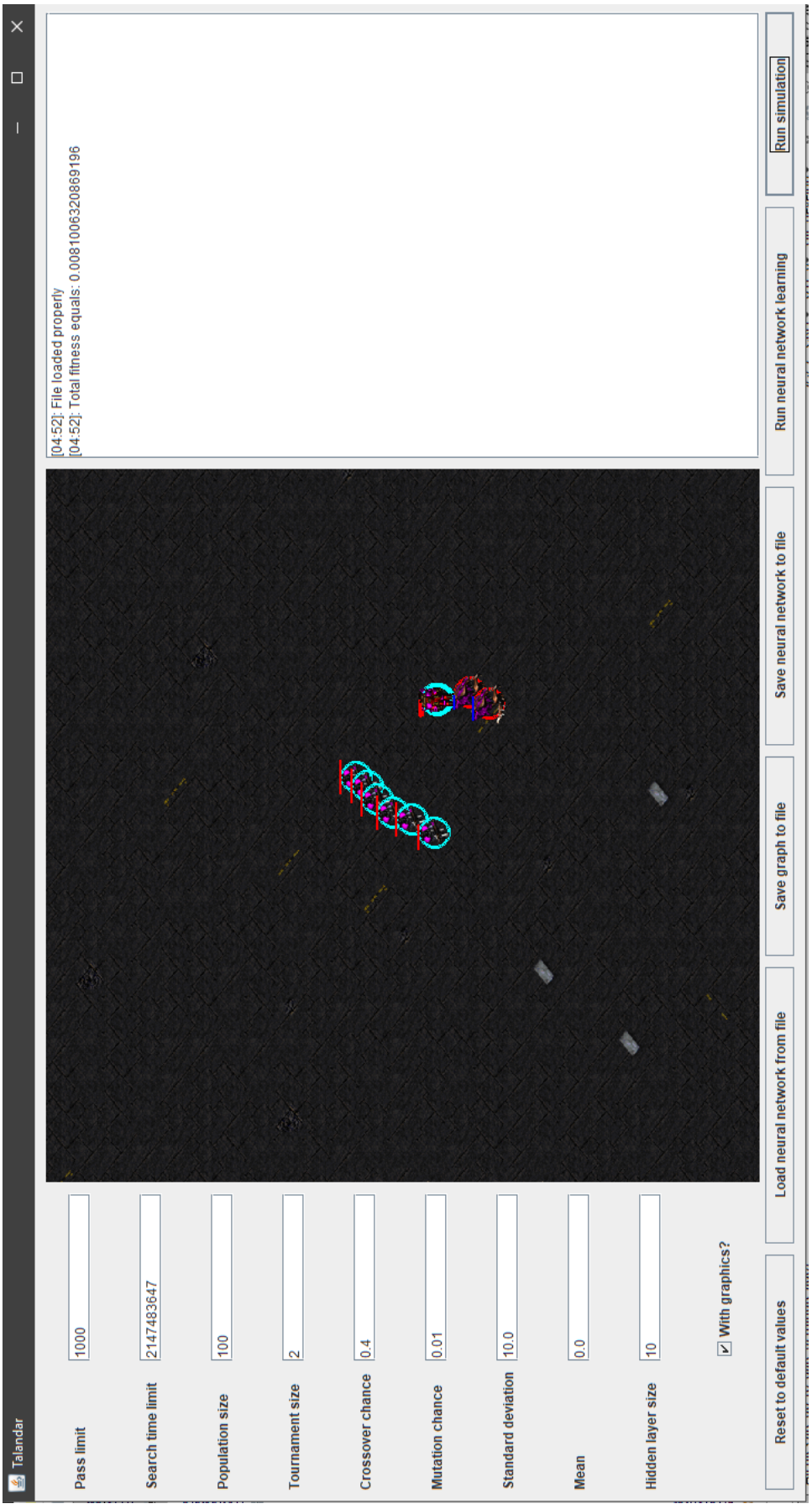


Rysunek 4.7: Pozostałe klasy

NeuralNetworkTests zawiera testy do sieci neuronowej, a reszta z nich to różne klasy i metody, jakie były pomocne przy implementacji, takie jak klasa Pair, czy różne operacje matematyczne, lub na plikach.

4.4. Interfejs użytkownika

Interfejs użytkownika został wykonany przy użyciu multiplatformowego pakietu dostępnego w Javie nazywającego się Swing. Został on zaimplementowany w taki sposób, aby w miarę możliwości próbował imitować wygląd systemu używanego przez użytkownika. Jeżeli będzie to niemożliwe, to wróci do swojego podstawowego wyglądu. Tak prezentuje się interfejs na zrzucie ekranu:



Rysunek 4.8: Interfejs programu

Z prawej części dostępne są logi programu, gdzie podawane są wyniki skuteczności wyuczonych botów i ewentualne komunikaty błędów. Na środku jest podgląd starcia w czasie rzeczywistym. Jednostki po lewej są jednostkami, którymi steruje sieć neuronowa, a jednostki po prawej są sterowane przez proste AI dla porównania. Można je także odróżnić dzięki innym kolorom pasków nad ich głowami wskazującym na pozostałą liczbę punktów wytrzymałości. Kolorowe kręgi pod nimi pokazują, jaki obecnie rozkaz jest przez nich wykonywany. Po lewej zaś mamy możliwość edycji różnych parametrów uczenia sieci neuronowej, wraz z domyślnymi wartościami, jakie są wpisane po uruchomieniu programu.

Pierwszy przycisk na dole od lewej pozwala zresetować zmiany, jakie wprowadziliśmy do konfiguracji. Drugi przycisk pozwala na wczytanie sieci neuronowej z pliku, wyświetlając okno do jej wyboru. Kolejne dwa przyciski pozwalają zapisać odpowiednio wykres nauki i samą sieć do pliku, tak samo jak poprzednio wyświetlając to samo okno wyboru miejsca. Przedostatni przycisk pozwala na uruchomienie procesu uczenia sieci neuronowej od zera, zaś ostatni z nich pozwala ponownie powtórzyć jak wczytana, lub wyuczona sieć sprawuje się podczas bitwy. Uruchamianie bez interfejsu jest o wiele szybsze niż z jego użyciem, ponieważ celowo spowalnia on działanie programu, aby człowiek mógł zobaczyć ruch w normalnym tempie.

5. Badania skuteczności

W poniższym rozdziale zostanie przedstawiona metodyka badań, podane wyniki uzyskane przy użyciu wytworzonej aplikacji, oraz ich zostanie przedstawiona ich analiza.

5.1. Wejście i wyjście sieci neuronowej

Na wejściach sieć neuronowa ma przekazywane następujące informacje:

- Ilość punktów wytrzymałości jednostki
- Zasięg broni jednostki
- Zasięg broni najbliższej jednostki przeciwnika
- Odległość najbliżej jednostki przeciwnika
- Odległość jednostki przeciwnika z najmniejszą ilością punktów życia

Na wyjściach sieć neuronowa może wybrać z następujących taktyk:

- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku, to zaatakuj ją. Jeżeli nie, to, jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą odległością
- Oddalaj się od jednostek przeciwnika
- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą ilością życia
- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku, to zaatakuj ją. Jeżeli nie, to, jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą ilością życia
- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to, jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to oddalaj się od jednostek przeciwnika
- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to, jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku i można ją zaatakować, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą odległością
- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą ilością życia
- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to, jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to, jeżeli jednostka jest w zasięgu ataku najbliżej jednostki przeciwnika, to oddalaj się od jednostek przeciwnika. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą odległością
- Jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą odległością
- Jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą ilością życia
- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to oddalaj się od jednostek przeciwnika.
- Jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą odległością
- Jeżeli jednostka wroga z najmniejszą ilością życia jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą ilością życia
- Jeżeli jednostka wroga z najmniejszą odległością jest w zasięgu ataku i broń jest przeładowana, to zaatakuj ją. Jeżeli nie, to, jeżeli jednostka jest w zasięgu ataku najbliżej jednostki przeciwnika, to oddalaj się od jednostek przeciwnika. Jeżeli nie, to poruszaj się w kierunku jednostki wroga z najmniejszą odległością

5.2. Definicja miary skuteczności

Każda sieć neuronowa była porównywana do algorytmów sztucznej inteligencji, które miały za zadanie symulować te użyte w grze Starcraft przez jej twórców. Oznacza to, że jednostki kierują się jak najkrótszą drogą do najbliżej jednostki wroga i jak tylko znajdą się w jej zasięgu, atakują ją.

Aby był możliwy proces uczenia sieci neuronowej i ocena uzyskanych rozwiązań, należało ustalić funkcję przystosowania. Funkcja ta powinna posiadać dwie cechy. Pierwszą z nich jest priorytetyzowanie jednostek droższych w produkcji, które wymagają do tego celu rzadszych zasobów. A drugą zachowanie przy życiu jak największej liczby jednostek i jak najbardziej równe rozłożenie ich punktów wytrzymałości. Ponieważ tak długo jak jednostki żyją, to mogą one zadawać obrażenia innym jednostkom czy korzystać ze swoich umiejętności specjalnych. Została opracowana następująca funkcja przystosowania:

$$u_{m_i} = c_{m_i} + 2 * c_{w_i}$$

$$u_{c_i} = u_{m_i} * \left(\frac{d_{c_i}}{d_{m_i}}\right)^{\frac{3}{4}}$$

$$f = \frac{\sum_{i=1}^{n_p} u_{c_i}}{\sum_{i=1}^{n_p} u_{m_i}} - \frac{\sum_{i=1}^{n_e} u_{c_i}}{\sum_{i=1}^{n_e} u_{m_i}}$$

Wzór matematyczny 5.1: Funkcja przystosowania

Oznaczenie symboli:

- c_m to koszt wyprodukowania jednostki w minerałach
- c_w to koszt wyprodukowania jednostki w wespanie – jest on liczony podwójnie, jako że jest to rzadszy zasób niż minerały
- u_m to wartość jednostki w momencie, gdy ma pełną ilość punktów wytrzymałości
- u_c to wartość jednostki po zakończeniu starcia, gdy może mieć mniejszą ilość punktów wytrzymałości
- d_c to ilość punktów wytrzymałości po zakończeniu starcia
- d_m to ilość punktów wytrzymałości, gdy jednostka jest w pełni zdrowa
- n_p to ilość jednostek gracza
- n_e to ilość jednostek przeciwnika
- $\frac{3}{4}$ to współczynnik potęgowania, który sprawia, że większa ilość jednostek, które będą miały razem większą ocenę niż mniejsza, która miałaby łącznie tyle samo punktów życia – został on dobrany doświadczalnie
- f to wartość funkcji przystosowania w pojedynczym starciu

Tak opracowana funkcja przystosowania przyjmuje wartości z zakresu, od -1 (gdy zginą wszystkie jednostki gracza, a jednostki przeciwnika pozostaną z pełna ilością punktów wytrzymałości), do 1 (gdy zdarzy się sytuacja przeciwna). Ponieważ nauka sieci neuronowej przy użyciu wyłącznie jednego scenariusza testowego mogłaby doprowadzić do przeuczenia, zostało przygotowane 576 przypadków testowych dla zbioru walidacyjnego, dla których liczona jest średnia arytmetyczna ocen. Są to kombinacje następujących parametrów:

- Rasy jednostek obu graczy. Jeżeli rasą gracza jest zerg, jego jednostkami będą zerglingi (małe i zwinne, walczące w zwarciu) i hydraliski (strzelające z daleka kwasem). Jeżeli protoss, to zostaną mu przydzielone zealoty (wytrzymałe i walczące wręcz) i dragoony (pokaźnych rozmiarów jednostki zasięgowe), W przypadku terrana będą to mariny (wyposażone w karabin podstawowe jednostki terran) oraz firebaty (posługujące się krótkodystansowymi miotaczami ognia). Są to najpopularniejsze kombinacje jednostek, jakimi posługują się gracze danej rasy.
- Liczba jednostek walczących wręcz, oraz dystansowych. Każdego typu jednostek może być 0, 6 lub 12 – są to wartości nawiązujące do istniejącego w grze limitu wielkości grupy kontrolnej, który wynosi właśnie 12 jednostek. W związku z tym podczas rozgrywki starcia większych grup są bardzo rzadkie. Zostały wyłączone także przypadki, gdzie dana strona nie posiadałaby wcale jednostek, jako że zakończą się one zawsze tym samym wynikiem. Możliwe jest, aby jedna ze stron posiadała więcej jednostek, dotyczy to zarówno ich łącznej liczby, jak i określonego typu.

Początkowo jako zbiór uczący była losowana określona liczba przypadków i ich lustrzanych odbić, jednak okazało się, że zbyt mała ich liczba powoduje dużą zmienność wyników i okazjonalnie powodowała przeuczenie. W związku z tym doświadczalnie zbiór uczący został ustalony na kombinacje wszystkich ras jednostek, jednak z liczbą jednostek każdego typu równą 6 – zredukowało to jego rozmiar do 9, jednocześnie zachowując bardzo zbliżone wyniki do pełnego zbioru walidacyjnego.

5.3. Metodyka badań i uzyskane wyniki przy użyciu symulatora

Badania przeprowadzono z następującymi parametrami:

- Warunek stopu - liczba wyliczeń funkcji przystosowania - 10000
- Brak ograniczeń czasowych – ze względu na ograniczoną mapę symulacji nie jest możliwa ciągła ucieczka jednostek, dlatego zawsze przebiegnie ona w skończonym czasie
- Wielkość selekcji turniejowej - 2
- Wagi sieci neuronowej w początkowej populacji były generowane zgodnie z rozkładem normalnym
- Krok symulacji – 1 klatka – jest możliwość ich pomijania w symulacji, jednak powoduje to straty dokładności
- Wielkość mapy – 640 piksele długości i szerokości
- Liczba ukrytych warstw sieci neuronowej – 1
- Liczba ukrytych neuronów – 10
- Szansa na mutacje – trzy możliwości: 0.1%, 1% i 10%
- Szansa na krzyżowanie – trzy możliwości: 10%, 40% i 70%
- Rozmiar populacji – trzy możliwości: 10, 100, 1000
- Ilość powtórzeń dla danej kombinacji możliwości: 10

Następnie obliczono średnią arytmetyczną, oraz odchylenie standardowe najlepszych wyników z powtórzeń dla danych możliwości parametrów:

Szansa na krzyżowanie	Szansa na mutacje	Liczba osobników w populacji	Średnia ocena najlepszego osobnika	Odchylenie standardowe
10,00%	0,10%	10	0,1275	0,0192
10,00%	0,10%	100	0,1827	0,0547
10,00%	0,10%	1000	0,1875	0,0249
10,00%	1,00%	10	0,1540	0,0232
10,00%	1,00%	100	0,1666	0,0277
10,00%	1,00%	1000	0,2034	0,0298
10,00%	10,00%	10	0,1801	0,0158
10,00%	10,00%	100	0,1930	0,0230
10,00%	10,00%	1000	0,1997	0,0309
40,00%	0,10%	10	0,1271	0,0370
40,00%	0,10%	100	0,2311	0,0316
40,00%	0,10%	1000	0,2032	0,0304
40,00%	1,00%	10	0,1483	0,0201
40,00%	1,00%	100	0,2087	0,0339
40,00%	1,00%	1000	0,2055	0,0136
40,00%	10,00%	10	0,2170	0,0330
40,00%	10,00%	100	0,2235	0,0273
40,00%	10,00%	1000	0,2114	0,0141
70,00%	0,10%	10	0,1332	0,0209
70,00%	0,10%	100	0,1681	0,0194
70,00%	0,10%	1000	0,2120	0,0186
70,00%	1,00%	10	0,2031	0,0460
70,00%	1,00%	100	0,1973	0,0383
70,00%	1,00%	1000	0,2103	0,0245
70,00%	10,00%	10	0,1986	0,0193
70,00%	10,00%	100	0,2028	0,0244
70,00%	10,00%	1000	0,2080	0,0141

Tabela 5.1: Wyniki badań przy użyciu symulatora

Badania pokazują dość duże odchylenie standardowe, oraz ciężko jest zauważyć zależność użytych parametrów od uzyskanych wyników. Niestety został one wykonane na zbyt małej ilości powtórzeń i iteracji. Łączna ilość obliczeń związanych z sieciami neuronowymi, ich uczeniem algorytmami genetycznymi i użyciem ich w symulacjach jest tak ogromna, że nie pozwoliła na wykonanie bardziej dokładnych badań w rozsądnym czasie. Najlepsze uzyskane pojedyncze rozwiązania osiągały ocenę około 0.30, co oznacza wygrane starcia z zachowanymi 30% wartości jednostek. Dla porównania rozwiązania korzystające zawsze tylko z jednej z dostępnych taktyk sieci neuronowej osiągają maksymalną ocenę równą prawie 0.09. Można z tego wywnioskować, że sieć neuronowa nie mogła się trzymać tylko jednej z nich i musiała zmieniać swoje wybory w zależności od sytuacji na polu bitwy.

5.4. Metodyka badań i uzyskane wyniki przy użyciu gry Starcraft: Brood War

Aby porównać uzyskane wyniki przy pomocy symulatora, z wynikami uzyskanymi przy użyciu gry Starcraft: Brood War została wybrana najlepsza wyuczona sieć neuronowa otrzymana w poprzednich badaniach. Algorytmy sztucznej inteligencji domyślnie zaprogramowane w grze, do których chcemy porównać nasze rozwiązania, oddalają się ze swojej pozycji tylko na pewną odległość, a potem wracają na pozycję startową. Inaczej zachowują się algorytmy sztucznej inteligencji, które przybliżają je w symulatorze, nieustannie goniąc jednostki gracza kontrolowanego przez sieć neuronową. W związku z tym część sieci z najlepszymi wynikami musiała zostać pominięta, ponieważ ich starcia nigdy by się nie zakończyły.

Przygotowano 9 map do wykorzystania w grze, które odpowiadają zbiorowi uczącemu w symulatorze – nie zostały stworzone wszystkie mapy dla zbioru walidacyjnego, ponieważ musiały być one tworzone ręcznie. Nazwano je XvY, gdzie X i Y to pierwsze litery rasy jednostek sieci neuronowej, Rozgrywka została uruchomiona na każdej z niej 10 razy i uzyskano następujące wyniki:

Próba	PvT	TvP	TvZ	ZvT	PvP	PvZ	TvT	ZvP	ZvZ
1	0,7899	-0,8645	-0,2140	-0,1449	-0,1550	0,8636	-0,2120	-0,8381	-0,0786
2	0,8169	-0,8217	-0,2368	-0,2612	-0,0421	0,8444	-0,1451	-0,8254	-0,3864
3	0,8074	-0,8658	0,1149	-0,1495	0,0179	0,8516	-0,1329	-0,8925	-0,1563
4	0,8192	-0,8726	0,0955	-0,1159	0,0787	0,8625	-0,0059	-0,8260	-0,4089
5	0,8429	-0,8676	0,0863	-0,1846	-0,1638	0,8646	-0,1139	-0,8492	-0,5654
6	0,8126	-0,8528	-0,1082	-0,0895	0,1281	0,8610	-0,1684	-0,8376	-0,1817
7	0,8317	-0,8533	0,1698	-0,1473	0,0610	0,8578	-0,1550	-0,8635	-0,4049
8	0,8098	-0,8488	0,2159	-0,0893	-0,2062	0,8668	-0,2185	-0,8728	0,0792
9	0,8215	-0,8615	0,1484	-0,2444	0,0853	0,8720	-0,2232	-0,8117	-0,2052
10	0,8127	-0,8397	0,1885	-0,1755	-0,1941	0,8501	-0,2686	-0,8627	-0,1898

Tabela 5.2: Wyniki badań przy użyciu gry Starcraft: Brood War

Całkowita średnia arytmetyczna wartości funkcji przystosowania wyniosła -0,0660, a jej odchylenie standardowo miało wartość 0,0758. Wyniki te są bardzo zmienne - powodem jest niedeterministyczność zaprogramowanych w grze algorytmów sztucznej inteligencji. Za każdym razem wykonuje ona inne ruchy, mimo identycznych warunków początkowych i deterministyczności sieci neuronowej. Symulacja nie uwzględnia także kolizji pomiędzy sąsiadującymi jednostkami, w przeciwieństwie do gry, co w dalszym stopniu wpływa na wyniki. Dodatkowo ze względu na niedoskonałości programistyczne, dochodziło czasem do zawieszenia się jednostek w miejscu i ignorowania przez pewien czas rozkazów, oraz występowały problemy z poruszaniem się jednostek po najkrótszej możliwej ścieżce. Są to znane problemy, które jednakże nigdy nie zostały naprawione, ponieważ mogłyby spowodować zbyt duże zmiany w grze, aby zostały zaakceptowane przez jej fanów. Wszystko to składa się na słabsze wyniki uzyskane w grze, niż te w symulatorze, przez te same wyuczone sieci neuronowe.

6. Podsumowanie

Na podstawie wykonanego przeglądu technologii i technik, a następnie projektu udało się pomyślnie zaimplementować aplikację. Zostało w niej zawarte uczenie sieci neuronowej algorytmami genetycznymi, wraz ze stworzeniem symulatora przyśpieszającego ten proces. Ze względu na duże wymagania sprzętowe aplikacji nie udało się wykonać bardzo dogłębnych badań. Otrzymane rezultaty pokazują, że osiąga ona lepsze rezultaty w symulatorze, niż w faktycznej grze.

Pierwszą rzeczą, która mogłaby być w aplikacji ulepszona, jest jej prędkość działania. Można by to osiągnąć przez jej dalszą optymalizację, zmianę języka implementacji, albo wykorzystania możliwości nowoczesnych kart graficznych do przyśpieszenia obliczeń. Kolejnym ulepszeniem mogłoby być dodanie obsługi gry Starcraft: Brood War także poza symulatorem, oraz rozbudowanie bota o możliwości bardziej strategiczne, takie jak budowanie własnej bazy i szkolenie jednostek. Ciekawym było by także zobaczyć jak radzi sobie ona z ludzkimi przeciwnikami czy innymi algorytmami sztucznej inteligencji, niż te, do których była ona porównywana. Warto by także wprowadzić zmiany do symulatora, aby lepiej odwzorowywał grę, co mogłoby poprawić osiągane wyniki poza nim.

7. Bibliografia

[1] „StarCraft AI,” [Online]. Available: http://www.starcraftai.com. [Data uzyskania dostępu: 04 10 2017].

[2] „BWAPI,” [Online]. Available: https://bwapi.github.io/. [Data uzyskania dostępu: 04 10 2017].

[3] „BWMirror API,” [Online]. Available: http://bwmirror.jurenka.sk/. [Data uzyskania dostępu: 04 10 2017].

[4] „Java Native Interface,” Oracle, [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jni/. [Data uzyskania dostępu: 04 10 2017].

[5] „JNIBWAPI,” [Online]. Available: https://github.com/JNIBWAPI/JNIBWAPI. [Data uzyskania dostępu: 04 10 2017].

[6] „UAlbertaBot,” [Online]. Available: https://github.com/davechurchill/ualbertabot. [Data uzyskania dostępu: 04 10 2017].

[7] „Jarcraft,” [Online]. Available: https://github.com/tbalint/JarCraft. [Data uzyskania dostępu: 04 10 2017].

[8] M. Nielsen, „Neural Networks and Deep Learning,” [Online]. Available: http://neuralnetworksanddeeplearning.com/. [Data uzyskania dostępu: 04 10 2017].

[9] Z. Michalewicz i D. B. Fogel, Jak to rozwiązać, czyli nowoczesna heurystyka, Wydawnictwa Naukowo Techniczne, 2006.

[10] S. Kenneth, „NeuroEvolution of Augmenting Topologies,” [Online]. Available: http://www.cs.ucf.edu/~kstanley/neat.html. [Data uzyskania dostępu: 04 10 2017].

[11] JetBrains, „IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains,” [Online]. Available: https://www.jetbrains.com/idea/. [Data uzyskania dostępu: 07 10 2017].

[12] „Git,” [Online]. Available: https://git-scm.com/. [Data uzyskania dostępu: 09 10 2017].

[13] „Git Extensions,” [Online]. Available: http://gitextensions.github.io/. [Data uzyskania dostępu: 09 10 2017].

[14] Apache, „Apache Commons,” [Online]. Available: http://commons.apache.org/. [Data uzyskania dostępu: 09 10 2017].

[15] Google, „GitHub - google/gson: A Java serialization/deserialization library to convert Java Objects into JSON and back,” [Online]. Available: https://github.com/google/gson. [Data uzyskania dostępu: 09 10 2017].

[16] „JUnit,” [Online]. Available: http://junit.org/junit5/. [Data uzyskania dostępu: 09 10 2017].

[17] „Hamcrest,” [Online]. Available: http://hamcrest.org/. [Data uzyskania dostępu: 09 10 2017].

[18] „JFreeChart,” [Online]. Available: http://www.jfree.org/jfreechart/. [Data uzyskania dostępu: 09 10 2017].

8. Spis grafik

9. Spis tabel

10. Spis wzorów matematycznych