|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | |  | **Wydział Informatyki i Zarządzania** kierunek studiów: Informatyka Praca dyplomowa - inżynierska**Implementacja algorytmów sztucznej inteligencji** **w grze Starcraft: Brood War**Marcin Żerko słowa kluczowe:  Neuroewolucja, 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:\**   1. *kategorii A (akta wieczyste)* 2. *kategorii BE 50 (po 50 latach podlegające ekspertyzie)*   *\* niepotrzebne skreślić*   |  | | --- | | pieczątka wydziałowa | |
|  |  |  | Wrocław 2017 |

**Streszczenie**

To do

**Abstract**

To do

**Spis treści**

[1. Wstęp 7](#_Toc497972333)

[1.1. Wprowadzenie 7](#_Toc497972334)

[1.2. Geneza pracy 7](#_Toc497972335)

[1.3. Cel i zakres pracy 7](#_Toc497972336)

[2. Stan wiedzy i techniki w zakresie tematyki pracy 8](#_Toc497972337)

[2.1. Porównanie Starcrafta do innych gier 8](#_Toc497972338)

[2.2. Przegląd istniejących technologii 8](#_Toc497972339)

[2.3. Przegląd istniejących technik 10](#_Toc497972340)

[3. Założenia projektowe i projekt aplikacji 11](#_Toc497972341)

[3.1. Wymagania funkcjonalne 11](#_Toc497972342)

[3.2. Wymagania niefunkcjonalne 11](#_Toc497972343)

[3.3. Diagram przypadków użycia 12](#_Toc497972344)

[3.4. Przypadki użycia 13](#_Toc497972345)

[4. Implementacja aplikacji 15](#_Toc497972346)

[4.1. Sposób realizacji implementacji 15](#_Toc497972347)

[4.2. Wykorzystywane biblioteki 15](#_Toc497972348)

[4.3. Struktura projektu 15](#_Toc497972349)

[4.4. Interfejs użytkownika 15](#_Toc497972350)

[5. Badania skuteczności 16](#_Toc497972351)

[6. Podsumowanie 17](#_Toc497972352)

[7. Bibliografia 18](#_Toc497972353)

1. Wstęp

1.1. Wprowadzenie

Starcraft jest jedną z najpopularniejszych strategicznych gier czasu rzeczywistego. 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 tą wersją zajmę się w tej pracy. Gra odniosła ogromny sukces głównie w Korei Południowej i stworzyła się wokół niej duża społeczność esportowa. 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. Możemy je wydobywać przy pomocy naszych jednostek zbierających, które produkujemy przy pomocy naszego 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 ludzkim przeciwnikom. Jednak nie zawsze jest taka możliwość, lub gracz może nie mieć na to ochoty. Wtedy na pomoc przychodzą boty, – czyli zaprojektowana sztuczna inteligencja, która próbuje udawać grę prawdziwego człowieka. Ta dołączona do instalacji gry jest prymitywna i łatwo z nią wygrać. W internecie jest dostępnych wiele innych botów, ale większość z nich skupia się wyłącznie na modyfikacji wysoko poziomej strategii, podchodząc bardzo podstawowo do kwestii 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 dwie dziedziny swoich zainteresowań – sztuczną inteligencję oraz Starcrafta – i spróbować wypracować własne rozwiązanie, które pozwoliłoby na skuteczne zarządzanie tymi to jednostkami w czasie rzeczywistym, podczas gry.

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 w projekcie, 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. Następny rozdział składa się z definicji miary skuteczności i wyników badań zastosowanych algorytmów. Na koniec opisane zostało podsumowanie projektu, wraz z dalszymi możliwościami rozwoju.

Należy jeszcze to zrewidować z faktycznym stanem rozdziałów w przyszłości. Wyszło ich pierwotnie 6.

2. Stan wiedzy i techniki w zakresie tematyki pracy

2.1. Porównanie Starcrafta do innych gier

W porównaniu do gier takich jak szachy, trudnością tej gry dla algorytmów sztucznej inteligencji są dwa fakty. Po pierwsze - jest ona 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 rezultacie zmniejsza szanse na wygraną w grze. Zaproponowane przez mnie rozwiązanie tego problemu zostanie przedstawione w dalszym rozdziale.

Dodatkowo też jest to gra niepełnej informacji. Gracz nie ma pełnej wiedzy o stanie i posunięciach przeciwnika, do póki nie widzi ich w danym momencie własnymi jednostkami lub budynkami. W związku z tym bot musi albo losować strategie, albo wybierać je z pewnym prawdopodobieństwem. Gdy już uda się zwiadowcom dowiedzieć o posunięciach przeciwnika, może wystąpić konieczność odpowiedniej modyfikacji naszych zachowań, aby nie zostać pokonanym. W pracy tej zajmuje się wyłącznie samym zarządzaniem jednostkami w starciu, gdy już wiemy, jakie siły posiada przeciwnik, w związku, z czym potencjalne możliwości na obejście tej kwestii nie zostaną tutaj przedstawione.

2.2. Przegląd istniejących technologii

Podstawową rzeczą, jaką trzeba wybrać przed implementacją programu, jest język programowania, jaki chcemy do tego celu użyć. Wiele projektów opartych na sztuczną inteligencji używa C++ - w to 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. Niestety moje umiejętności pisania w tym języku są zbyt małe, przez co zdecydowałem się używać do tego projektu bardziej znanego mi języka Java. Jest to język działający na własnej maszynie wirtualnej - dzięki temu jest bardziej przenośny, co pozwoli mi w łatwy sposób uruchamiać aplikację zarówno na komputerach z systemem operacyjnym Windows, jaki i Linux. Posiada on też automatyczny garbage collector, który zarządza pamięcią, pozbywając się nieużywanych obiektów, co przekłada się na łatwiejszą obsługę dla programisty. Wadą tego rozwiązania jest niemożliwość ręcznego zarządzania pamięcią, przez co jest on jednocześnie wolniejszy od swojego konkurenta. Uznałem jednak, że potencjalne zalety C++ mogą zostać przeważone przez ewentualne błędy w mojej implementacji, ze względu na brak mojego doświadczenia w tym języku w porównaniu do Javy.

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 naszego 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 wprowadzało kolejne poziomy abstrakcji. 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 cały problem. Jedynym kompletnym frameworkiem, który by nam na to pozwolił jest BWAPI [2]. Aby zachować sprawiedliwość 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 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 mocno pracochłonne, a nowe wersje gry nie wprowadzają do niej istotnych z tego punktu widzenia poprawek.

BWAPI jest napisane w C++, dlatego musiałem poszukać odpowiednego rozwiązania, które pozwoliłoby mi użyć go od strony Javy. Istnieje kilka technologii pozwalających złączyć ze sobą te dwa języki programowania, jednak zazwyczaj do własnej implementacji wymagają pewnej znajomości tego pierwszego języka programowania. Najbardziej aktualnym takim gotowym rozwiązaniem 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 portu 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 sztuczną inteligencję bez uruchamiania gry. Inaczej jest w przypadku JNIBWAPI [5] – tutaj jest możliwe rozszerzenie biblioteki, aby dane te po uruchomieniu gry były zapamiętane przez kod Javowy 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 natywnie 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 napisana w czasach Windowsa 98, co ogranicza ilość wykorzystywanej pamięci ram 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 wyuczania bota. Takim właśnie rozwiązaniem jest Sparcraft – część bota o nazwie UAlbertaBot. [6] Jest on także napisany w C++, w związku, z czym poszukiwałem jego odpowiednika 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 problemów. 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 wiele innych problematycznych modyfikacji, które utrudniał brak intuicyjności, dokumentacji i nie trzymanie się praktyk pisania czystego kodu. Okazało się również, że jest ona mocno 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 istniejących technik

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 to następnie go przeprocesuje funkcją aktywacji i może go podać dalej. 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 ilość ukrytych, przez co każda kolejna z nich może przykładowo coraz bardziej generalizować problem. Jest możliwe stworzenie takiej sieci, która będzie zachowywała się tak jak bramka XOR. Przy użyciu odpowiednich kombinacji tych bramek jest możliwe uzyskanie także innych funkcji logicznych, w związku z czym teoretycznie odpowiednio zbudowana sieć neuronowa jest w stanie przybliżyć każdą możliwą funkcję. Jednakże wymaga to użycia dużej ilości neuronów, co zwiększa wymagania i tworzy mnóstwo dodatkowych parametrów. Wszystkie one muszą być w jakiś sposób ustalone na odpowiednie wartości. Proces taki nazywamy uczeniem i jest na to wiele sposobów. Należy także pamiętać, że zbyt duża ilość neuronów może wprowadzić overfitting, dlatego potrzebny jest odpowiedni balans. [8]

Algorytmy genetyczne także zostały zainspirowane naturalnymi procesami, tym razem ewolucją gatunków i ich dopasowaniem do warunków otoczenia. Początkowo generowana jest startowa populacja i każdy z osobników jest oceniany za pomocą funkcji oceny. Każdy z nich ma swój genotyp, który mówi jak jest dany osobnik jest zbudowany. Następnie występuje selekcja osobników z populacji i krzyżowanie ich genów. Są to mechanizmy pozwalające eksploatować pobliskiej przestrzeni rozwiązań w poszukiwaniu coraz to lepszego rozwiązania. Kolejnym krokiem jest mutacja, gdzie geny poszczególnych odpowiedników zmieniają się. Pozwala to na eksploracje przestrzeni i potencjalne uniknięcie utknięcia w lokalnym minimum. Tak zmieniona populacja zostaje ponownie oceniona i cykl powtarza się, aż do osiągnięcia określonego wcześniej warunku stopu. Efektem pracy tej metaheurystyki jest najbardziej optymalny osobnik, jaki udało się jej odkryć. [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 właśnie algorytmów genetycznych. Mimo ich braku wiedzy o dziedzinie problemu, wyszukują one rozwiązanie, czyli uczą sieć neuronową, które w tym przypadku jest całym zestawem wag pomiędzy połączeniami w sieci neuronowej. Zdecydowałem się napisać ich implementacje własnoręcznie, 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 i projekt aplikacji

3.1. Wymagania funkcjonalne

Badania skuteczności zaimplementowanych przeze mnie algorytmów sztucznej inteligencji mogę przeprowadzić modyfikując sterujące nimi parametry bezpośrednio w kodzie. Jednakże takie rozwiązanie było by niewygodne i problematyczne dla innych potencjalnych użytkowników, dlatego postanowiłem stworzyć także interfejs graficzny dla programu. Oto wymagania funkcjonalne, jakie powinna spełniać moja aplikacja:

* Ustawienie parametrów neuroewolucji, aby można było dopasować jej działanie do własnych potrzeb
* Ustawianie maksymalnego czasu wykonania, aby można było wiedzieć, jaki jest maksymalny czas oczekiwania na nie
* Ustawienie maksymalnej ilości iteracji, aby można było porównywać pomiędzy różnymi parametrami, nie opierając się na bardzo zmiennym czasie, który może zależeć od danego 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 neuroewolucji
* Wypisanie oceny najlepszego uzyskanego osobnika, aby można było się dowiedzieć jak dobrze się sprawuje
* 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 wyniki w praktyce
* Wyświetlanie wybranych przez jednostkę decyzji, aby można było przeanalizować jak dana sieć neuronowa kieruje w starciu

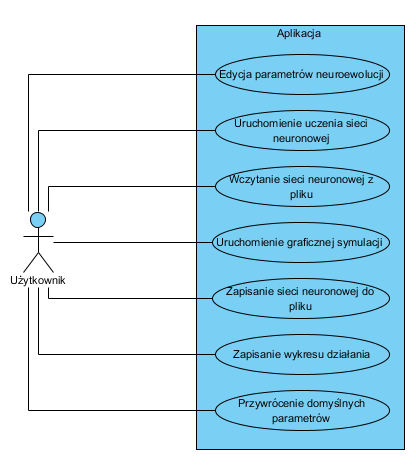
3.2. Wymagania niefunkcjonalne

Oprócz wymagań funkcjonalnych aplikacja powinna spełniać także pomocne wymagania niefunkcjonalne:

* Aplikacja powinna być responsywna i nie zawieszać interfejsu podczas symulacji, aby nie irytować użytkowników
* Aplikacja powinna móc uruchomić się na systemie Windows, jak i Linux, aby była bardziej przenośna
* Aplikacja powinna nie pozwalać na uruchomienie uczenia z błędnymi parametrami, aby użytkownik nie mógł jej zablokować
* Aplikacja powinna zapisywać i wczytywać się dane w formacie json, aby były możliwe do edycji także ręcznie
* 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 jak najbardziej generycznie, z wykorzystaniem interfejsów, aby pozwolić na jej łatwą rozbudowę w przyszłości

3.3. Diagram przypadków użycia

Aby zilustrować przypadki użycia, jakie chciałbym zawrzeć w swoim programie wykonałem diagram przy użyciu programu Visual Paradigm:



Rysunek 3.1: Diagram przypadków użycia

3.4. Przypadki użycia

**Przypadek użycia:** Edycja parametrów neuroewolucji

**Opis:** Przypadek ten opisuje możliwość edycji parametrów neuroewolucji, aby dostosować je do potrzeb użytkownika

**Warunki wstępne:** brak

**Przebieg:**

1. Aplikacja dostarcza domyślne parametry neuroewolucji
2. Użytkownik edytuje wybrane przez niego parametry

**Przypadek użycia:** Przywrócenie domyślnych parametrów

**Opis:** Przypadek ten opisuje możliwość przywrócenia domyślnych parametrów neuroewolucji, aby użytkownik mógł cofnąć swoje zmiany

**Warunki wstępne:** brak

**Przebieg:**

1. Użytkownik klika na przycisk przywrócenia domyślnych parametrów neuroewolucji
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:**

1. 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:**

1. Użytkownik rezygnuje z wyboru
2. Aplikacja wyświetla błąd, jeżeli wczytywanie nie powiodło się

**Przypadek użycia:** Zapisanie sieci neuronowej do pliku

**Opis:** Przypadek ten opisuje możliwość edycji parametrów neuroewolucji

**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:**

1. Użytkownik rezygnuje z wyboru
2. 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 neuroewolucji 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:**

1. Użytkownik rezygnuje z wyboru
2. Aplikacja wyświetla błąd, jeżeli zapisywanie nie powiodło się

4. Implementacja aplikacji

4.1. Sposób realizacji implementacji

Zgodnie z tym, co planowałem cała implementacje wykonałem w języku Java aktualnej wersji numer 8. Postanowiłem jednak nie korzystać ze wszystkich nowinek udostępnionych przez tą aktualizację. Zrezygnowałem z korzystania z streamów, ze względu na obawy o wydajność aplikacji. Pozwalają one na większe 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 wykonanie mojego programu, a zależało mi w nim na prędkości. Drugą nowinką, której nie wykorzystałem, jest 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 zdecydowałem się na użycie żadnej biblioteki do klonowania obiektów i wszystkie metody do tego służące zrobiłem ręcznie, co znacznie poprawiło wydajność.

Jako zintegrowane środowisko programistyczne postanowiłem wybrać IntelliJ IDEA firmy JetBrains. [11] Jest to jedno z najbardziej zaawansowanych i popularnych IDE, jakie są dostępne na rynku. Jego zdecydowana 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 kontekstowe. 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 wybrałem Gita. [12] Jest to rozproszony system, który pozwala na współpracę wielu deweloperów nad jednym kodem. W moim przypadku oczywiście pracowałem samemu, jednak dzięki niemu mogłem bezproblemowo pracować na kilku różnych komputerach, wprowadzać wersje na różne poboczne gałęzie, cofać poprzednie zmiany, czy przeglądać pełną historię. 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 to trzy biblioteki od firmy Apache z pakietu Apache Commons: Apache IO, Apache Math i Apache Lang. [14] Ta 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.

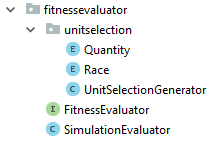
Drugim zestawem bibliotek, jest Gson i Gson Extras. [15] Zostały one wykonane przez Google. Pierwsza z nich pozwala na bardzo łatwą serializacje i deserializacje obiektów do formatu Json. Jej użycie pozwala ograniczyć modyfikacje w kodzie niż konkurencyjne biblioteki takie jak Jackson. W większości wypadków nie wymaga do działania publicznych konstruktorów, czy dodatkowych annotacji. 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 kawałek 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

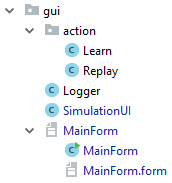
Pierwszą paczką w projekcie jest paczka fitnessevaluator:



Rysunek 4.1: Paczka fitness evaluator

Jest to interfejs, który służy do obliczania oceny danego osobnika. Obecnie zaimplementowana jest tylko jedna klasa, nazwana SimulationEvaluator, która do tego celu używa napisanej przeze mnie symulacji. W przyszłości można było by przykładowo rozbudować tutaj projekt i dodać także możliwość używania prawdziwej gry do oceniania osobników. W podpaczce unitseleciton znajduje się klasa generująca ustawienia jednostek do symulatora, oraz dwa pomocnicze enumy określające ilość jednostek i ich rasę.

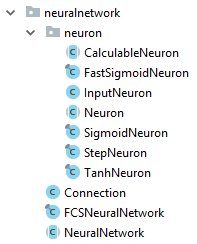
Kolejną paczką jest paczka gui:



Rysunek 4.2: Paczka gui

Klasa MainForm jest główną klasą w aplikacji i został zaimplementowany jej 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. W podpaczce action znajdują się klasy Learn i Replay zarządzające procesem nauki i ponownego uruchomienia neuroewolucji.

Trzecią paczką w projekcie jest neuralnetwork:

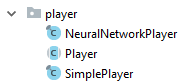


Rysunek 4.3: Paczka neuralnetwork

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, nie obliczaną z poprzednich neuronów. StepNeuron, jako funkcji aktywacji używa funkcji kroku, FastSigmoidNeuron szybszej (jednakże błędnej) wersji funkcji sigmoid, a TanhNeuron tangensa hiperbolicznego. Są one nieużywane obecnie, jednak pozostały one w kodzie aplikacji.

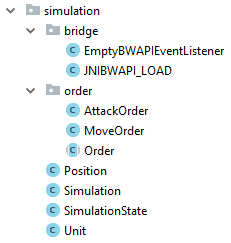
Czwartą paczką jest paczka player:



Rysunek 4.4: Paczka player

Jest to najmniejsza paczka zawierająca tylko klasę abstrakcyjną player i rozszerzające je klasy. SimplePlayer jest atakującym najbliższą jednostkę graczem, do którego porównujemy taktyki, a NeuralNetworkPlayer jest wykorzystującym sieci neuronowe graczem.

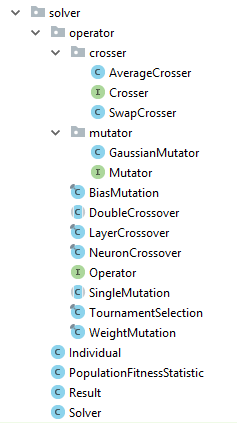
Następną paczka jest simulation:



Rysunek 4.5: Paczka simulation

Jest to paczka, w której znajdują się pliki służące do symulacji pola bitwy. Podpaczka bridge zawiera 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 ja uruchomić, a SimulationState reprezentuje jej stan w danym momencie w czasie.

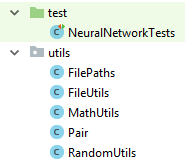
Szóstą paczką jest paczka solver:



Rysunek 4.6: Paczka solver

Zawiera ona solver, który 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 taboo search, czy symulowane wyżarzanie. Należałoby tylko napisać nowe operatory, które by je realizowały.

Dwie ostatnie paczki to paczki test i utils:

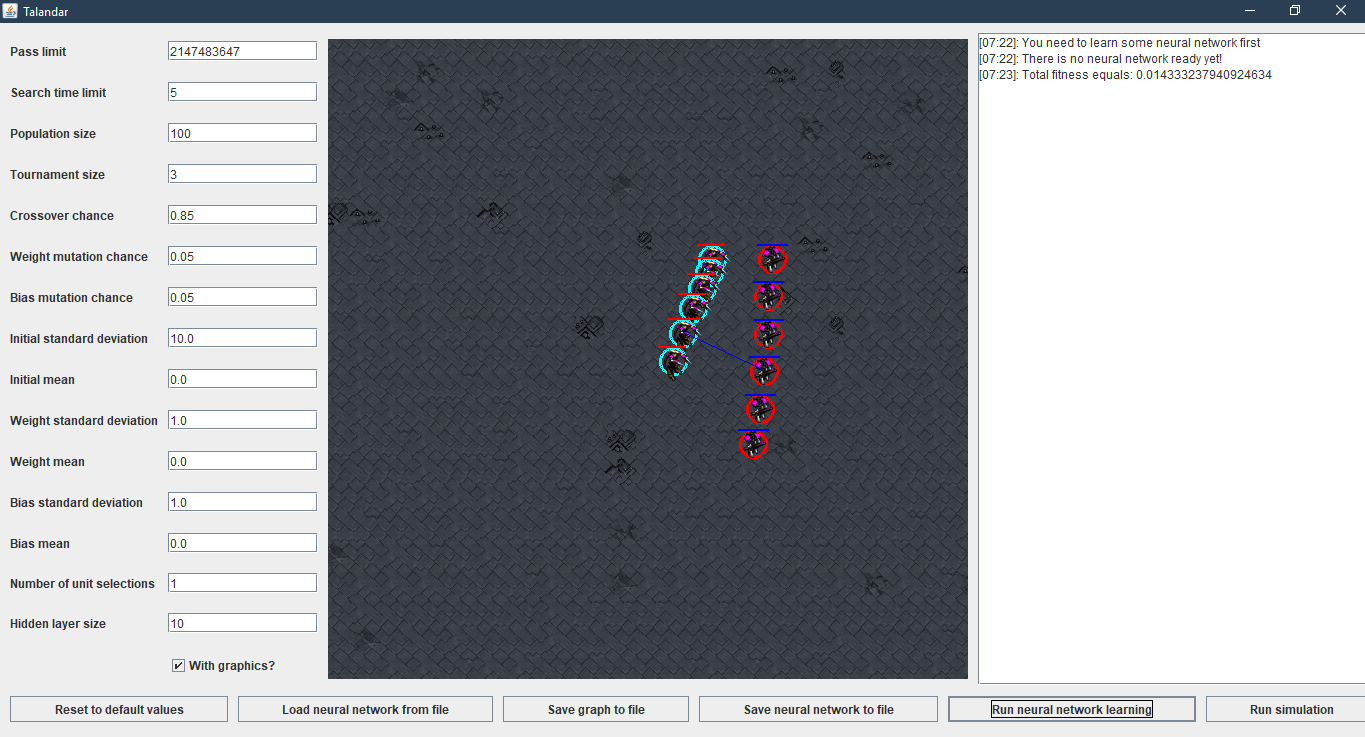


Rysunek 4.7: Paczki test i utils

Pierwsza z nich zawiera testy do sieci neuronowej, a druga z nich 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 wykonałem przy użyciu multiplatformowego toolkitu dostępnego w Javie nazywającego się Swing. Zaimplementowałem go 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 4.9: 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 pozostała ilość punktów wytrzymałości. Kolorowe kręgi pod nimi pokazują, jaki obecnie rozkaz jest przez nich wykonywany. Po lewej zaś mamy możliwość edycji różnych parametrów neuroewolucji, 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 neuroewolucji 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

6. Podsumowanie

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]. |

Todo: poprawić format bibliografii – chwilowo jest automatycznie generowana