|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | |  | **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](#_Toc497701834)

[1.1. Wprowadzenie 7](#_Toc497701835)

[1.2. Geneza pracy 7](#_Toc497701836)

[1.3. Cel i zakres pracy 7](#_Toc497701837)

[2. Stan wiedzy i techniki w zakresie tematyki pracy 8](#_Toc497701838)

[2.1. Porównanie Starcrafta do innych gier 8](#_Toc497701839)

[2.2. Przegląd istniejących technologii 8](#_Toc497701840)

[2.3. Przegląd istniejących technik 10](#_Toc497701841)

[3. Założenia projektowe i projekt aplikacji 11](#_Toc497701842)

[3.1. Wymagania funkcjonalne 11](#_Toc497701843)

[3.2. Wymagania niefunkcjonalne 11](#_Toc497701844)

[3.3. Diagram przypadków użycia 12](#_Toc497701845)

[3.4. Przypadki użycia 12](#_Toc497701846)

[4. Implementacja aplikacji 14](#_Toc497701847)

[4.1. Sposób realizacji implementacji 14](#_Toc497701848)

[5. Badania skuteczności 15](#_Toc497701849)

[6. Podsumowanie 16](#_Toc497701850)

[7. Bibliografia 17](#_Toc497701851)

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

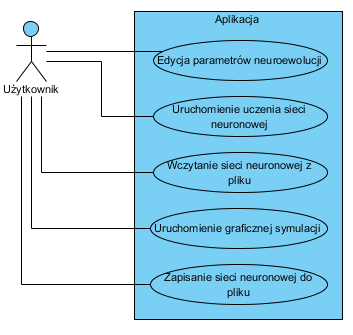
3.2. Wymagania niefunkcjonalne

Oprócz wymagań funkcjonalnych aplikacja powinna spełniać także następujące 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 korzystać z wielowątkowości, aby przyśpieszyć jej działanie, jeżeli użytkownik się na to zdecyduje
* 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:** 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 w nowym oknie dialogowym

**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 w nowym oknie dialogowym, jeżeli wczytywanie nie powiodło się

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

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

**Warunki wstępne:** uruchomione zostało wcześniej uczenie sieci neuronowej

**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 w nowym oknie dialogowym, 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:** wczytana została sieć neuronowa

**Przebieg:**

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

4. Implementacja aplikacji

4.1. Sposób realizacji implementacji

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]. |

Poprawić format bibliografii