Politechnika Śląska w Gliwicach

Wydział Automatyki, Elektroniki i Informatyki



**Biologically Inspired Artificial Intelligence**

**Temat projektu:** *Gra „7 Cudów Świata”*

**Autorzy:** Piotr Kardyś

Marzena Kruszy

Rafał Potoczek

**Prowadzący:** mgr inż. Marcin Wierzchanowski

**Rok akademicki:** 2015/2016

**Kierunek:** Informatyka

**Rodzaj studiów:** SSI

**Semestr:** VI

**Grupa:** GKiO3

**Sekcja:** 2

**Data oddania sprawozdania:** 21.09.2016

1. **TREŚĆ ZADANIA**

Implementacja gry „7 Cudów Świata” oraz komputerowego przeciwnika z wykorzystaniem sieci neuronowej.

1. **ANALIZA ZADANIA**

„7 Cudów Świata” – gra planszowa, która podzielona jest na trzy epoki podczas których gracze kupują budowle umożliwiające rozwój jednego z siedmiu cudów świata. Celem gry jest zebranie jak największej ilości punktów pochodzących z następujących źródeł: punkty militarne (przyznawane za zwycięstwa militarne), posiadane monety, stopień rozbudowania posiadanego cudu świata, punkty z budynków cywilnych, punkty z budowli komercyjnych, punkty z kart gildii oraz punkty z budynków naukowych.

Głównym celem projektu było zapoznanie się z działaniem sieci neuronowych. Program zawiera implementację okrojonej wersji gry „7 Cudów Świata”, jak również komputerowego przeciwnika, którego ruch odbywa się za pomocą podania danych do sieci (w której następuje podjęcie decyzji na temat wyboru najbardziej wartościowej karty), a następnie pobrania danych wyjściowych. W pierwszym etapie sieć neuronowa musi zostać poddana procesowi uczenia się, aby dokonywała wyboru najcenniejszej karty z zestawu. Następnie można przystąpić do gry właściwej i spróbować wygrać stawiając sobie za przeciwnika AI.

* 1. **Algorytmy**

Projekt napisany jest w języku C++ z wykorzystaniem biblioteki FANN (Fast Artificial Neural Network Library).

Fast Artificial Neural Network (FANN) – biblioteka implementująca sztuczną sieć neuronową (SSN), którą można wykorzystać w C, C++, PHP, Pythonie, Delphi, a nawet w środowisku Mathematica. Stanowi ona potężne narzędzie dla twórców oprogramowania. SSN mogą być użyte w wielu dziedzinach m.in. tworzenie gier komputerowych, rozpoznawanie obiektów na obrazach czy przewidywanie trendów giełdy.

Do zaimplementowanej sieci neuronowej podawanych jest 6 danych wejściowych (numer epoki 1 – 3 oraz 5 liczb zmiennoprzecinkowych reprezentujących poszczególne karty rozdania). W wyniku działania sieci otrzymujemy 5 danych wyjściowych (wartość 1 na pozycji karty, która została wybrana oraz wartości 0 na pozostałych miejscach). Pierwszą rzeczą jaką należy zrobić jest nauczenie sieci wyboru odpowiedniej karty z zestawu. W tym celu przygotowany został plik AIWonder.data zawierający zestawy danych wejściowych oraz odpowiadające im zamierzone zestawy danych wyjściowych. Po zakończeniu procesu nauki tworzony jest plik AIWonder.net przechowujący już gotową sieć neuronową (dzięki temu nie ma potrzeby uczenia sieci każdorazowo przy uruchomieniu programu). Po wykonaniu tej czynności można przystąpić do gry ze sztuczną inteligencją.

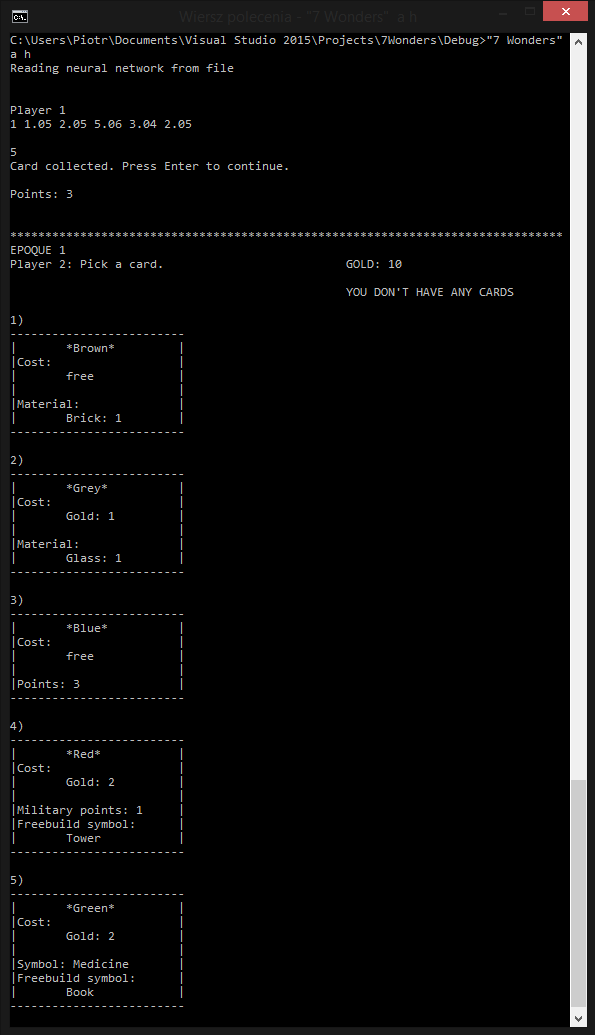
Dodatkowo program wyposażony został w możliwość wyboru graczy (HUMAN/AI) dzięki czemu może zostać uruchomiony w trybie:

* gry przeciw człowiekowi
* gry przeciw sztucznej inteligencji
* obserwatora (rozgrywka dwóch sztucznych inteligencji pomiędzy sobą)

1. **SPECYFIKACJA ZEWNĘTRZNA**

Uruchomienie programu następuje z linii komend z dwoma parametrami wejściowymi, określającymi rodzaj graczy:

* h – człowiek (HUMAN)
* a – sztuczna inteligencja (AI)

W wyniku uruchomienia programu otrzymujemy następujący widok:

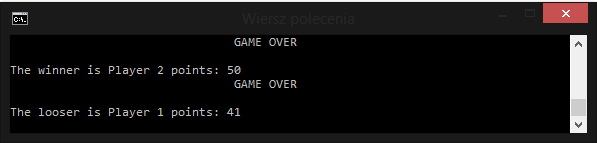
W tym przypadku Gracz 1 zdefiniowany został jako AI, natomiast Gracz 2 jako HUMAN. Następuje wczytanie sieci neuronowej z pliku AIWonder.net i wykonany zostaje ruch przeciwnika (wybór jednej karty z pięciu możliwych). Następnie mamy możliwość wykonania naszego ruchu. Widok gracza (człowieka) zawiera takie informacje jak:

* numer epoki
* ilość posiadanego złota
* listę posiadanych przez gracza kart
* informację o konieczności wyboru karty
* listę dostępnych kart w bieżącym rozdaniu

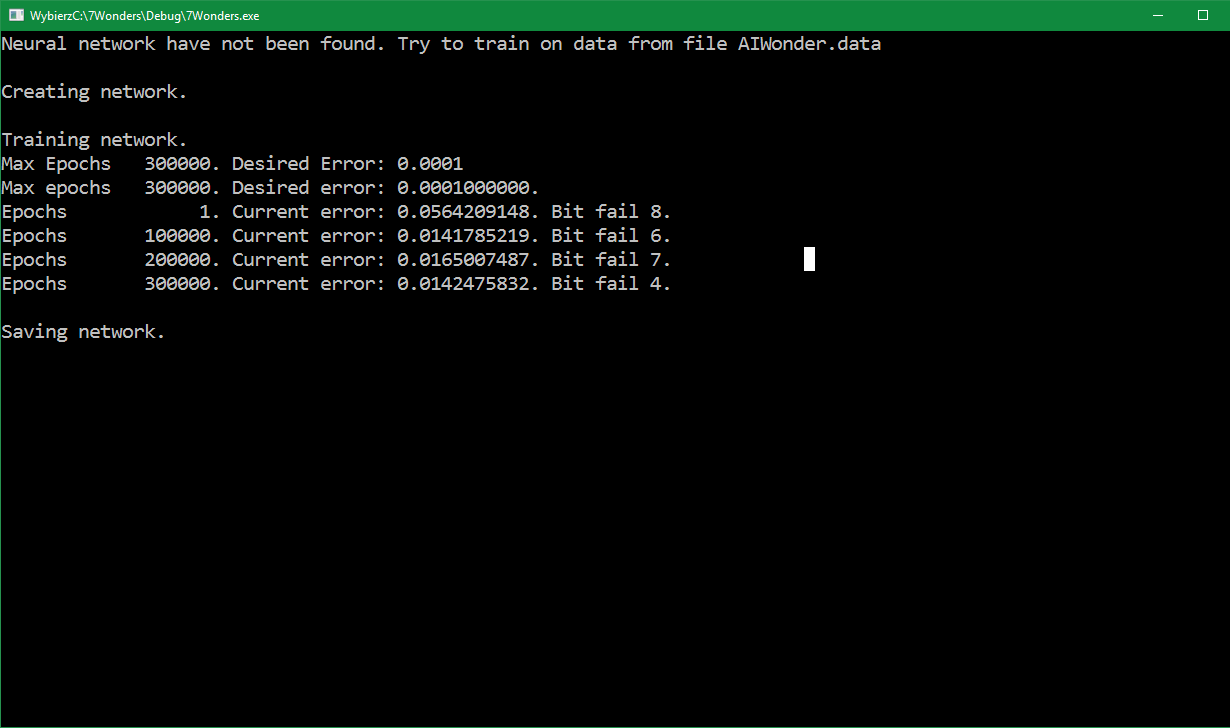
Po wybraniu numeru karty, którą jesteśmy zainteresowani otrzymujemy informację, czy karta została dodana do posiadanych przez nas kart, czy też została sprzedana (w przypadku, gdy nie stać nas na jej zakup). Ostatnią informacją tury

jest podanie ilości posiadanych przez nas punktów (punkty obliczane są na podstawie ilości punktów poszczególnych kart posiadanych przez gracza oraz ilości posiadanego złota podzielonego przez 3, ponieważ 3 sztuki złota odpowiadają 1 punktowi).

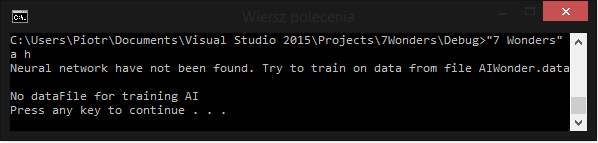
Karty wybierane są naprzemiennie, aż do zakończenia 3 epoki, a co za tym idzie zakończenia rozgrywki. W tym miejscu zostaje podana informacja na temat zwycięscy, podejmowane decyzje (tury) wygranego gracza zostają zapisane do pliku train.txt (w celu ułatwienia tworzenia pliku do nauki sieci) i działanie programu zostaje zakończone.



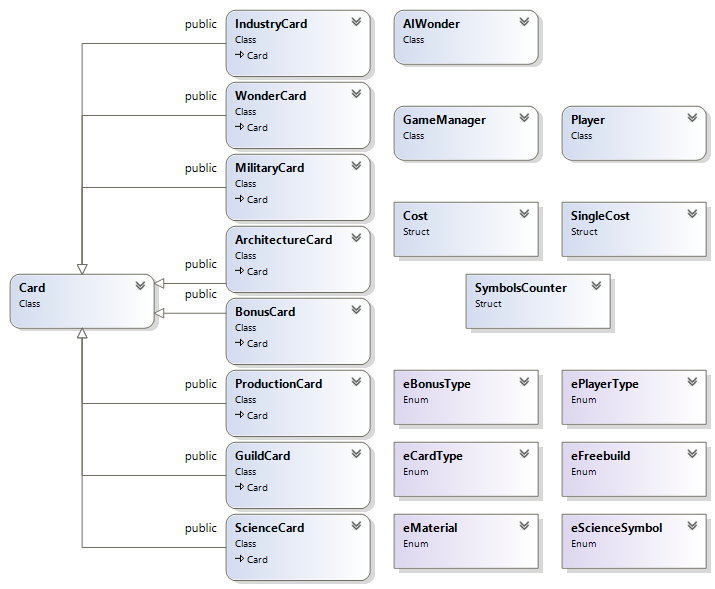
Program zabezpieczony został przed brakiem dostarczenia pliku AIWonder.net (zwierającego gotową sieć neuronową) oraz pliku AIWonder.data (zawierającego zestaw danych potrzebnych do procesu nauczania). W pierwszym przypadku wyświetla informację o tym, że plik z siecią nie został odnaleziony i został rozpoczęty proces uczenia i tworzenia nowego pliku AIWonder.net. Następnie program rozpoczyna swoje działanie.



W drugim przypadku wyświetlona zostaje informacja o braku pliku z danymi do nauki sieci, a wykonywanie programu zostaje przerwane.



1. **SPECYFIKACJA WEWNĘTRZNA**
   1. **Klasy**



* **main** – główna klasa programu, odpowiedzialna za wczytanie argumentów wejściowych (w przypadku ich braku zapewnia argumenty domyślne: Gracz 1 – sztuczna inteligencja „a”, Gracz 2 – człowiek „h”), inicjalizację sieci neuronowej, wczytanie zasobów (inicjalizację elementów gry) oraz rozpoczęcie rozgrywki
* **Card** – abstrakcyjna klasa zawierająca elementy wspólne dla wszystkich kart (pola: typ karty, koszt budowy, numer epoki, kod karty, reprezentację „graficzną” karty tzn. stringa przechowującego jak zostanie ona wyświetlona w konsoli oraz funkcje ustawiające/pobierające poszczególne pola klasy)
* **IndustryCard, WonderCard, MilitaryCard, ArchitectureCard, BonusCard, ProductionCard, GuildCard, ScienceCard** – klasy dziedziczące po klasie **Card**, reprezentujące odpowiednie typy kart w grze. Każda z nich posiada specyficzne dla siebie konstruktory, funkcje ustawiające/pobierające poszczególne składowe, jak i same pola:
  + **IndustryCard** – karty posiadają pola: produkowane materiały oraz bonus
  + **WonderCard** – karty cudów posiadają pola: produkowane materiały, bonusy, dodatkowe złoto, punkty, siła militarna
  + **MilitaryCard** – czerwone karty posiadają pola: siła oraz znak darmowej budowy
  + **ArchitectureCard** – niebieskie karty posiadają pola: punkty oraz znak darmowej budowy
  + **BonusCard** – złote karty posiadają pola: punkty, dodatkowe złoto, znak darmowej budowy oraz typ karty na jaką mają wpływ (efekt)
  + **ProductionCard** – brązowe i szare karty posiadają pole produkowanego materiału
  + **GuildCard** – fioletowe karty posiadają pola: punkty, dodatkowe złoto oraz typy kart na jakie mają wpływ (efekt)
  + **ScienceCard** – zielone karty posiadają pola: symbol naukowy oraz znak darmowej budowy
* **AIWonder** – klasa odpowiedzialna za inicjalizację, naukę oraz wykorzystanie sztucznej inteligencji w programie. Posiada ona również zdefiniowane stałe przechowujące ścieżki plików AIWonder.net oraz AIWonder.data
* **GameManager** – klasa reprezentująca rozgrywkę, odpowiedzialna jest ona za: wczytanie zasobów gry (z pliku Cards.xml) i przypisanie ich do odpowiednich obiektów klas, rozpoczęcie rozgrywki oraz obsługę głównej pętli gry (w tym interpretację wyborów graczy), wyświetlanie rozdań kart (jak i samych kart) oraz zapis poczynań wygranego gracza do pliku
* **Player** – klasa reprezentuje gracza, odpowiedzialna jest za: ustawienie typu gracza, obsługę zasobów gracza (złoto, punkty, posiadane karty oraz materiały), sprawdzanie czy gracz może sobie pozwolić na kupno danej karty oraz zapis tury gracza
* **SingleCost** – struktura przechowująca informacje na temat materiału i jego ilości (wykorzystywana do określenia produkowanego materiału przez brązowe i szare karty, jak również kosztów budowy)
* **Cost** – struktura reprezentująca koszty budowy poszczególnych budynków (zawiera ona wektor elementów struktury SingleCost oraz pole przechowujące znak darmowej budowy)
* **SymbolsCounter** – struktura będąca licznikiem zebranych symboli naukowych przez gracza (zawiera 2 pola: symbol oraz posiadaną ilość tych symboli)
* **eCardType** – typ wyliczeniowy przechowujący typy kart (brązowa, szara itp.)
* **eMaterial** – typ wyliczeniowy przechowujący typy materiałów (drewno, szkło itp.)
* **eFreebuild** – typ wyliczeniowy przechowujący znaki darmowej budowy
* **eScienceSymbol** – typ wyliczeniowy przechowujący symbole naukowe
* **eBonusType** – typ wyliczeniowy przechowujący dodatkowe bonusy (obniżony koszt budowy, ponowienie tury itp.)
* **ePlayerType** – typ wyliczeniowy przechowujący rodzaj gracza (AI/HUMAN)
  1. **Stałe**

Wszystkie stałe wykorzystywane w programie znajdują się w pliku nagłówkowym **Constants.h**. Zdefiniowane są w nim m.in. takie stałe jak :

* **strFileDirectory** – stała przechowująca ścieżkę do pliku Cards.xml, w którym zdefiniowane są wszystkie karty występujące w grze
* **strCardTypeBrown** – stała przechowująca typ karty w stringu (w tym przypadku jest to karta brązowa, dla innych kolorów kart zdefiniowane są analogiczne stałe)
* **strMaterialWood** – stała przechowująca typ materiału produkowanego/ potrzebnego do budowy w stringu (w tym przypadku jest to drewno, dla innych materiałów zdefiniowane są analogiczne stałe)
* **strFreebuildSword** – stała przechowująca znak darmowej budowy w stringu (w tym wypadku jest to miecz, dla innych znaków zdefiniowane są analogiczne stałe)
* **strSymbolMedicine** – stała przechowująca symbol naukowy w stringu (w tym wypadku jest to medycyna, dla innych symboli zdefiniowane są analogiczne stałe)
* **strBonusTypeLowerCost** – stała przechowująca dodatkowe bonusy w stringu (w tym wypadku jest to obniżony koszt produkcji budynków, dla innych bonusów zdefiniowane są analogiczne stałe)
  1. **Funkcje**

1. **TESTOWANIE**

Sztuczna inteligencja była testowana na danych zebranych z kilku gier(od gracza, który wygrał). Heurystycznie próbowaliśmy dobrać ustawienia SI, tak aby grała zgodnie z taktyką, którą obraliśmy. Udało się uzyskać zadowalającą skuteczność, niestety program nie jest idealny i zdarza się, że SI dokonuje wyboru, który nie jest najlepszy.

1. **WNIOSKI**

Trenowanie sztucznej inteligencji nie jest łatwym zadaniem. Trzeba tak dobrać specyfikację SI, aby znaleźć złoty środek między dokładnością, a czułością na dane – można doprowadzić do tego, aby dla danych testowych skuteczność AI była bardzo duża, ale przy innych danych uzyskamy całkowicie nieoczekiwane wyniki. Z drugiej strony będzie mała czułość na zmianę danych, ale także mała dokładność.