

Programowanie obiektowe

INEW0003P

Projekt

Wydział elektroniki	Kierunek: informatyka
Grupa zajęciowa: Pt 17:05	Semestr: 2017/2018 Lato
Nazwisko i Imię: Bączek Mateusz	Nr indeksu: 241330
Nazwisko i Imię: Szymon Filipiak	Nr indeksu: 241309
Nr grupy projektowej:	1
Prowadzący:	mgr inż. Adam Włodarczyk

Temat: Trenowanie sieci neuronowych w grze zręcznościowej

Ocena:
Punkty:
Data:

Założenia i opis funkcjonalny programu

Założenia

Program demonstruje możliwości uczenia sztucznych sieci neuronowych w symulowanym środowisku gry zręcznościowej - bitwy samolotów. Aplikacja

umożliwia wytrenowanie samolotów do walki między sobą, walkę sztucznej inteligencji z człowiekiem i zapis stanu wytrenowanej sieci do pliku w celu późniejszego kontynuowania treningu lub wczytania w celu gry z człowiekiem.

Dodatkowe funkcje

Program posiada funkcję trenowania sieci w tempie przyspieszonym, bez uruchomionej graficznej warstwy aplikacji, można w ten sposób zainstalować go na zdalnym serwerze i przeprowadzać na nim przyspieszoną symulację.

Mechanika gry

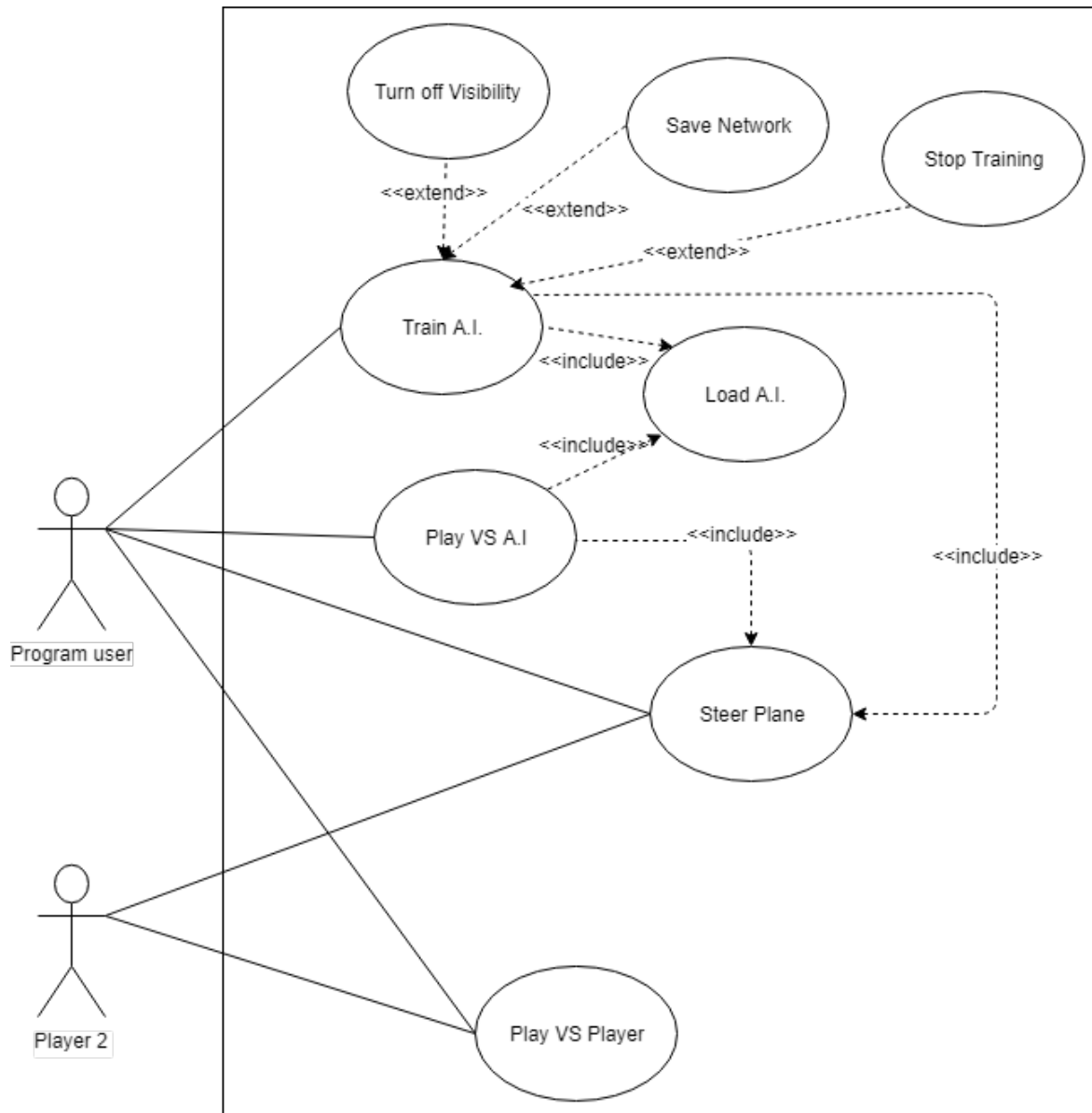
Gracz kontroluje jeden z kilku samolotów na planszy. Samoloty poruszają się do przodu przez cały czas, ale ich prędkość maleje wraz z wznoszeniem się, a rośnie podczas spadania. Gracz odpowiada za skręcanie samolotem, zwiększanie prędkości poprzez użycie specjalnej umiejętności (boost) oraz strzelanie. Samoloty ulegają zniszczeniu jeżeli wejdą w kolizję z innym obiektem (pociskiem lub innym samolotem). Gdy samolot doleci do krawędzi ekranu, pojawia się z jego drugiej strony. Przyspieszenie oraz karabin mają swoje czasy przeładowania.

Sztuczna inteligencja

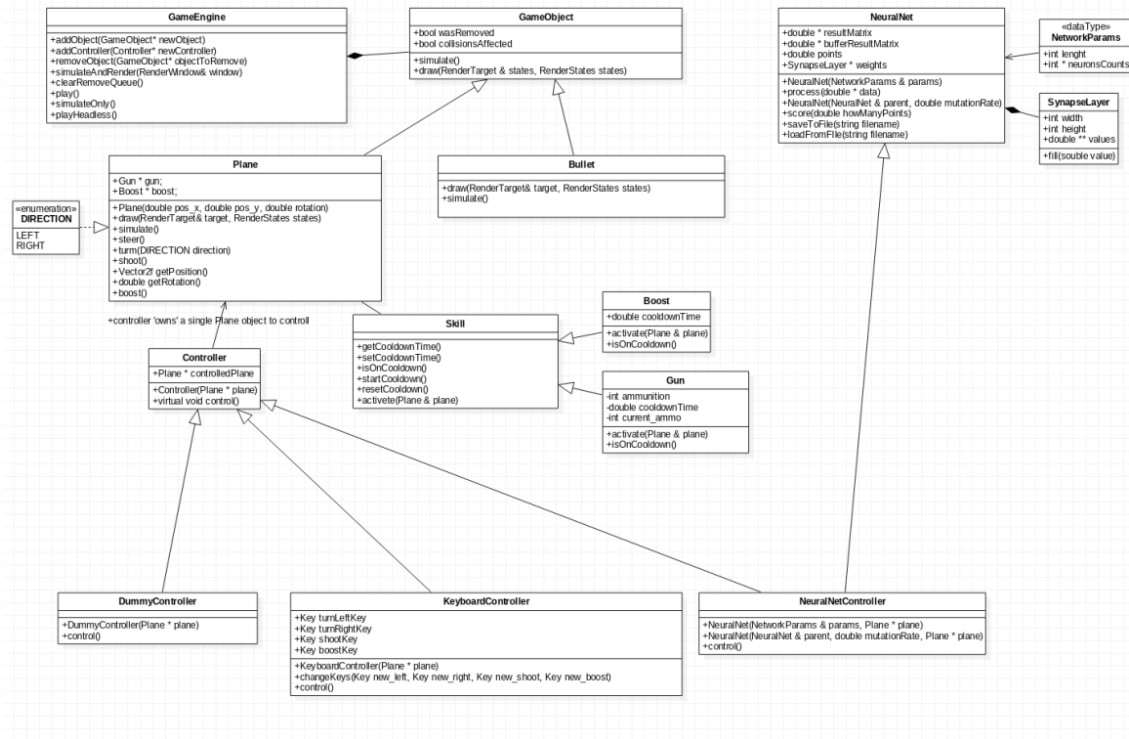
Samolot może być kontrolowany nie tylko przez człowieka, ale także przez bota - sztuczną sieć neuronową. Każdy bot jest wyposażony komórki wzrokowe odpowiedzialne za pozyskiwanie kierunku i odległości innych obiektów gry od samolotu, którym bot kieruje. Te informacje są przekazywane jako argumenty wejścia dla sieci neuronowej. Zależnie od danych wyjściowych sieci neuronowej, samolot wykonuje odpowiednie akcje (np skrety, strzelanie).

Diagramy

Use case



Klasy



Opis użytkowy programu

```

wint3rmute@gl553ve ➤ ./game
Machine Learning Plane Game Experiment
Usage:

--help - print this help message

--play2 - 2 planes, both controlled by keyboard, sort of a playground
--train-visible - show the process of network training
--train-visible-continue - same as above, but will use the best saved network as a starting point

--train-invisible - train the networks ASAP, no graphics
--train-invisible-continue - same as above, but will use the best saved network as a starting point

--play-ai - play against the best trained AI available
--play-ai-load <filename> play against the AI from the <filename>

wint3rmute@gl553ve ➤

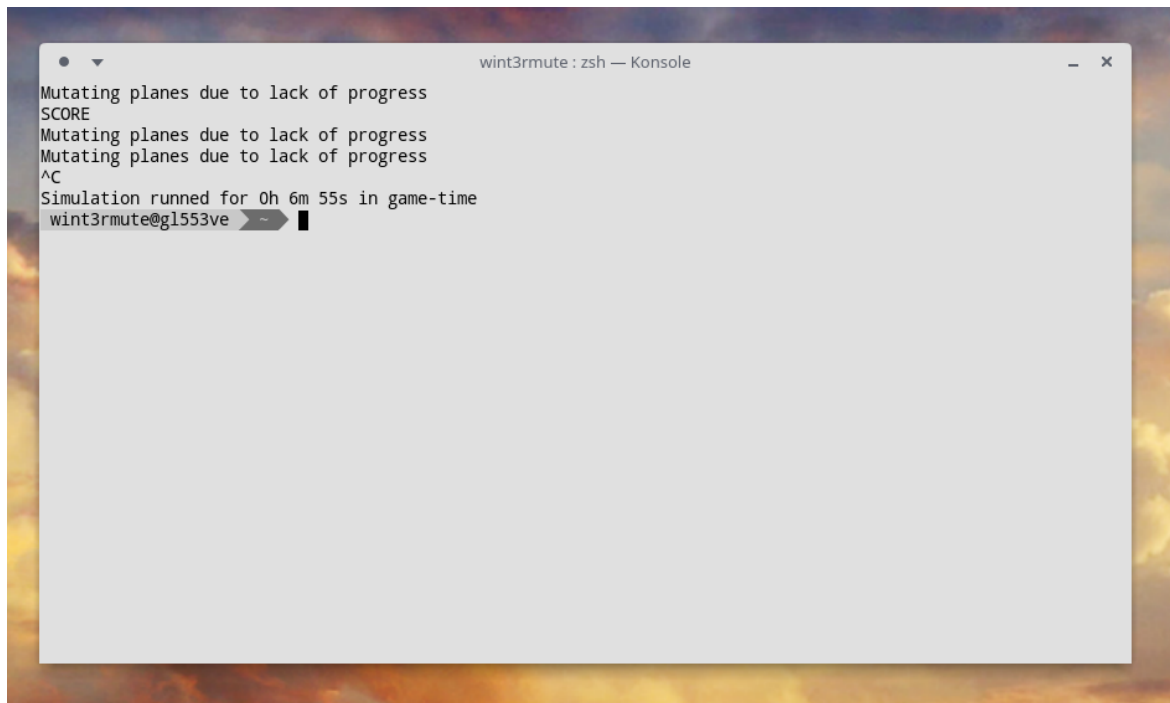
```

Program posiada standardowy unixowy interfejs konsolowy, po uruchomieniu bez wpisaniu argumentów użytkownikowi zostaje zaprezentowany ekran możliwych opcji wywołania programu.

Opis funkcji:

- `--help` - wyświetla taki sam ekran pomocy jak w przypadku wywołania programu bez argumentów
- `--play2` - Uruchamia grę z dwoma samolotami sterowanymi przez ludzkich graczy, pierwszy steruje za pomocą strzałek, drugi za pomocą klawiszy WSAD
- `--train-visible` - uruchamia proces trenowania z aktywną warstwą graficzną programu, użytkownik może obserwować postęp trenowania botów na ekranie
- `--train-visible-continue` - kontynuuje proces trenowania ostatniej zapisanej sieci (sieci zapisywane są automatycznie co każdą iterację trenowania)
- `--train-invisible` - rozpoczyna proces trenowania bez uruchomionej warstwy graficznej programu, w najszybszym możliwym tempie (na nowoczesnym procesorze i7 pozwala to na przyspieszenie symulacji około 450-500 razy)

Podczas trenowania w trybie niewidzialnym wyświetlany jest log symulacji, a po przerwaniu działania programu (za pomocą ctrl-c) wyświetlany jest czas, przez jaki symulacja działała (tj., ile czasu upłynęło wewnątrz symulacji)



```
wint3rmute: zsh — Konsole
Mutating planes due to lack of progress
SCORE
Mutating planes due to lack of progress
Mutating planes due to lack of progress
^C
Simulation runned for 0h 6m 55s in game-time
wint3rmute@gl1553ve ~
```

Przykładowe wyjście programu w przypadku przerwania symulacji w trybie przyspieszonym

- `--train-invisible-continue` - to samo co `--train-visible-continue`, tylko w trybie przyspieszonym
- `--play-ai` - pierwszy samolot jest ostatnią trenowaną sztuczną inteligencją, drugi jest sterowany przez gracza
- `--play-ai-load <filename>` - to samo co powyżej, dodatkowo wymagany jest parametr wskazujący na nazwę pliku z zapisaną siecią, która zostanie załadowana jako pierwszy gracz

Wnioski

Dzięki zrealizowaniu projektu poszerzyliśmy wiedzę związaną z C++ i Pythonem. Udało nam się zaimplementować i wytrenować sieć neuronową (do pewnego stopnia skuteczności).

Programując ten sam projekt w dwóch różnych językach programowania, zauważyliśmy różnice między C++em a Pythonem, najważniejsze z nich to:

- Znacznie łatwiejsze instalowanie bibliotek w Pythonie
- Większa dostępność łatwych do uruchomienia bibliotek do symulacji fizycznej i do grafiki (także w Pythonie)
- Zdecydowaną przewagę w wydajności programu napisanego w C++ (Chociaż Python pozwala na zwiększenie wydajności obliczeniowej za pomocą bibliotek natywnych, na przykład Numpy, z której też korzystaliśmy)
- Mniej wyraźny styl zarządzania pamięcią w Pythonie, utrudnione ręczne zarządzanie wyzwalaniem RAMu, język więcej rzeczy robi za programistę
- Wirtualne środowisko w Pythonie (virtualenv) ułatwia zarządzanie zainstalowanymi modułami, brak podobnie prostego w obsłudze narzędzia w C++

Implementacja czujników sieci neuronowej uświadomiła nam, że jest bardzo wiele sposobów, w jaki możliwe jest dobranie danych wejściowych dla sieci, co potencjalnie mogłoby pozytywnie wpłynąć na ich skuteczność w grze.

Dokumentacja kodu źródłowego umieszczona jest w załączniku do maila

Link do repozytorium projektu:

<https://github.com/Wint3rmute/project-oob-sem2>