Politechnika Wrocławska
Wydział Elektroniki, Fotoniki i Mikrosystemów

KIERUNEK: Automatyka i Robotyka

PRACA DYPLOMOWA INŻYNIERSKA

TYTUŁ PRACY:

Sztuczna inteligencja w roli gracza w grze zręcznościowej

Artificial intelligence as a player in an arcade game

AUTOR:
Jan Bronicki

PROMOTOR:
Dr inz. Mariusz Uchroński

WROCŁAW 2022

Spis treści

Skróty

HTML (ang. *HyperText Markup Language*)
JSON (ang. *JavaScript Object Notation*)
JIT (ang. *Just In Time compiler*)
PyPI (ang. *Python Package Index*)
NumPy (ang. *Numerical Python*)
SciPy (ang. *Scientific Python*)
SymPy (ang. *Symbolic Python*)
API (ang. *Application Programming Interface*)
GUI (ang. *Graphical User Interface*)
GTK (ang. *GIMP ToolKit*)
PEP (ang. *Python Enhancement Proposal*)
PIP (ang. *Preferred Installer Program*)
POSIX (ang. *The Portable Operating System Interface*)
DLL (ang. *Dynamic-Link Library*)
CI (ang. *Continuous Integration*)
CD (ang. *Continuous Delivery*)

Z dedykacją, dla Łukasza Stanisławowskiego

Rozdział 1

Wstęp

Tempore autem est minus et vel commodi. Magni explicabo eos enim placeat natus consectetur. Molestiae dolores nihil illum asperiores error praesentium excepturi. Ut itaque aliquid. Et vero voluptatem occaecati voluptatum.

Quaerat quis quia sit amet sed illum. Vitae quo nemo sit reprehenderit cum placeat dolor ipsum officiis. Omnis laudantium reprehenderit beatae tempora enim vel fugit dolorem. Earum ut quia cupiditate quo.

Animi inventore beatae ad repellat. Repellat commodi neque. Delectus ullam aut assumenda quaerat magnam repellendus optio. Libero eveniet consequatur qui nobis animi ea. Harum dicta voluptatem amet qui nulla. Illo accusantium nostrum ad animi architecto sint tempora quia.[**?**]

Rozdział 2

Cel pracy

2.1. Cel pracy

2.1.1. Podpodrozdział

Nemo voluptatum earum praesentium. Inventore ab fuga cum esse sit ullam facilis ipsa voluptate. Quod consequatur non porro alias at non. Przykład TODO:

Rozdział 3

Użyte narzędzia i technologie

3.1. Język Python

Język **Python** [17] [23] [20] jest bardzo popularnym, nowoczesnym oraz wysokopoziomowym językiem programowania. Czytając artykuły i inne treści na temat historii **Python**'a [24] [22] dowiadujemy się, że język powstał w 1991 roku. Stworzony przez Guido van Rossum podczas swojej pracy w laboratorium w Centrum Matematyki i Informatyki w Amsterdamie, pierwotnie tworzony był z myślą zastąpienia rozwijanego w latach osiemdziesiątych języka **ABC**. Samą nazwę język **Python** nazwę zawdzięcza popularnemu serialowi komediowemu emitowanemu przez BBC w latach siedemdziesiątych - "Latający Cyrk Monty Pythona", którego Guido był fanem. Projekt początkowo zakładał stworzenie prostego w użyciu, zwięzłego i wysokopoziomowego języka, głównie na potrzeby pracy w Amsterdamskim laboratorium. Z biegiem czasu **Python** stał się rozwijanym przez społeczność programistyczną projektem **Open Source**, nad którym czuwała organizacja non-profit założona przez Guido von Rossum'a, **Python Software Foundation**.

Python jest multi-paradygmatowym językiem, gdzie programowanie obiektowe i strukturalne są w pełni wspierane, wiele cech języka wspiera programowanie funkcyjne i aspektowe. Wiele innych paradygmatów jest wspieranych dzięki modularnym rozszerzeniom do języka. **Python** w przeciwieństwie do języków statycznie typowanych takich jak **C**, **Rust** czy **TypeScript**, stosuje typowanie dynamiczne, które sprawdza poprawność i bezpieczeństwo typów w programie, dynamicznie, podczas jego egzekucji, w przeciwieństwie do typowania statycznego, gdzie typy sprawdzane są podczas kompilacji kodu. Dodatkowo **Python** posiada kombinacje zliczania referencji oraz cyklicznego **Garbage Collector**'a. Architektura języka **Python** oferuje wsparcie, dla programowanie funkcyjnego zgodnego z tradycją języka **Lisp**. Język posiada typowe, dla języków funkcyjnych funkcje takie jak `filter`, `map`, `reduce`, `lambda`, list comprehensions, słowniki oraz generatory. Standardowa biblioteka języka posiada moduły `itertools` oraz `functools`, które implementują narzędzia funkcjonalne zapożyczone z języków **Haskell** oraz **Standard ML**.

Zamiast polegać na wbudowanej funkcjonalności w jądrze języka, **Python** został zaprojektowany tak, aby być najmożliwiej elastyczny, aby mógł współdziałać z różnymi odrębnymi modułami. Sama kompaktowa modularność sprawiła, że język stał się popularnym sposobem, na dodawanie programowalnego interfejsu to istniejących już aplikacji oraz języków programowania. **CPython** jest referencyjną implementacją **Python**'a, napisaną w **C**, która spełnia standard **C89** z niektórymi cechami standardu **C99**. Jedną z charakterystyk

implementacji **CPython**'a jest to, że zespół odpowiedzialny za rozwój oraz utrzymanie kodu jego interpretera preferuje odrzucanie poprawek do kodu mających na celu marginalną poprawę szybkości i sprawności interpretera w zamian za zachowanie czystości i czytelności kodu źródłowego. Taka długoterminowa postawa umożliwiła powstanie powstanie innych implementacji języka **Python** opartych o inne rozwiązania, bądź inne języki za pomocą których napisany został sam interpreter. Dzięki różnorodności implementacji **Python**'a, jego modularności i łatwej rozbudowie o obce rozszerzenia programista może zdecydować się na przeniesienie wrażliwych na czas wykonywania funkcjonalności do odrębnych modułów napisanych w innych językach takich jak **C**, lub **Rust**, albo użyć wyspecjalizowanej do tego odmiany samego interpretera takiej jak **PyPy**, która posiada tak zwany **JIT**, czyli **Just-In-Time compiler**, pozwalający na optymalizację kodu na system bądź architekturę docelową danego programu.

Dzięki tak rozbudowanemu środowisku oraz społeczności otaczającej język powstała ogromna biblioteka paczek, dla języka **Python**, która często podawana za jedną z największych zalet języka. W ten sposób **Python** stał się swoistym odpowiednikiem *Lingua Franca* wśród programistów. Z powodu wyżej wymienionych cech i okoliczności **Python** jest wykorzystywany w bardzo różniących się, oraz na pozór nie połączonych ze sobą dziedzinach takich jak Web Development, Automatyzacja, Bazy Danych, Aplikacje Mobilne, Testowanie Oprogramowania, Analiza Danych oraz Uczenie Maszynowe.

3.2. Biblioteka PyTorch

PyTorch [11] [18] jest Open Source'ową biblioteką służącą do uczenia maszynowego. Jest bazowana na bibliotece **Torch** napisanej w języku **Lua**. Z powodu niszowości języka **Lua**, oraz braku modularności i możliwości rozbudowywania go o nowe funkcjonalności za pomocą zewnętrznych modułów i paczek, powstał **PyTorch**, czyli, biblioteka **Torch**, ale zaimplementowana w języku **Python**. Dzięki temu **PyTorch** może korzystać z bardzo rozbudowanego środowiska **Python**, które oferuje dużą ilość naukowych paczek, między innymi takich jak **NumPy**.

Jedną z największych zalet biblioteki **PyTorch** jest możliwość programowanie imperatywnego. Jest to przeciwieństwem do bibliotek takich jak **TensorFlow** i **Keras**, które ze względu na poleganie głównie na językach takich jak **C** i **C++**, oferują jedynie możliwość programowania symbolicznego. Większość **Python**'owego kodu jest imperatywne jako, że jest to dynamicznie interpretowany język. W sytuacji symbolicznej zachodzi przeciwieństwo, ponieważ istnieje bardzo wyraźne rozróżnienie pomiędzy zdefiniowaniem grafu komputacyjnego, a jego kompilacją. W przypadku imperatywnym komputacja zachodzi w momencie jej wywołania, nie we wcześniej zoptymalizowanym punkcie w kodzie. Podejście symboliczne pozwala na większą optymalizację, a imperatywne takie jak **PyTorch** pozwalaj na większą swobodność, oraz używanie natywnych cech, funkcjonalności i rozszerzających modułów języka **Python**. Drugą największą zaletą biblioteki **PyTorch** są dynamiczne grafy komputacyjne, które w przeciwieństwie do bibliotek takich jak **TensorFlow** generują je statycznie przed uruchomieniem programu. **PyTorch** umie generować i modyfikować je dynamicznie podczas działania programu.

3.3. Biblioteka NumPy

NumPy [19] [15] [6] [5] jest Open Source biblioteką stworzoną, dla języka programowania **Python**, dodaje wsparcie, dla dużych wielowymiarowych tablic i macierzy wraz z dużą kolekcją wysokopoziomowych funkcji matematycznych, które pozwalają operować na wspomnianych macierzach. Poprzednikiem biblioteki **NumPy** był, **Numeric**, oryginalnie stworzony przez Jim'a Hugunin'a wraz z kontrybucjami kilku innych developerów. W roku 2005, Travis Oliphant stworzył projekt **NumPy** włączając w to właściwości oraz funkcjonalności **Numeric**'a. **Python** nie był oryginalnie stworzony do numerycznej komputacji, ale już we wczesnym życiu języka różne towarzystwa naukowe i inżynieryjne wyrażały swoje zainteresowanie językiem. **NumPy** adresuje problem powolności języka **Python** poprzez zapewnienie wielowymiarowych macierzy, funkcji i operacji, które są wydajne obliczeniowo operując na macierzach.

Używanie biblioteki **NumPy** w **Python**'ie funkcjonalnością przypomina programowanie w środowisku **MATLAB**, jako, że oba są interpretowane, mają podobną składnie, oba pozwalają użytkownikowi pisać szybkie i wydaje programy tak długo jak operacje przeprowadzane są na macierzach. W przeciwieństwie do **MATLAB**'a, **NumPy** nie oferuje tak wielkiej ilości dodatkowych narzędzi. Oferuje odwrotne podejście, gdzie to inne paczki/narzędzia korzystają u swoich podstaw z biblioteki **NumPy**. Dzięki zaawansowanej integracji z **Python**'em oraz byciu podłożem, dla zasadniczej większości paczek i narzędzi służących celom obliczeniowo naukowym takim jak **SciPy**, **SymPy**, **Scikit-Learn** i wielu innym **NumPy**, stał się podstawową warstwą, dla takich narzędzi a tym samym umożliwia im prostą komunikację między sobą jako, że macierze na których operują są macierzami biblioteki **NumPy**.

Główną funkcjonalnością biblioteki **NumPy** jest jej `ndarray`, struktura danych, reprezentująca n-wymiarową macierz. Wewnętrzna niskopoziomowa implementacja owych macierzy polega na kroczących widokach w pamięci. Takowe dane muszą być homogenicznego typu. Takie widoki pamięci mogą być również bufferami zaalokowanymi z poziomu innych języków takich jak **C**, **C++**, czy **Fortran**. Powoduje to ogromną optymalizację, jako, że eliminuje to potrzebę kopiowania i przenoszenia danych.

3.4. Biblioteka Matplotlib

Biblioteka **Matplotlib** [14] [21] [19] [3] jest najpopularniejszym **Python**'owym narzędziem służącym do tworzenia wykresów, grafów, histogramów, obrazów, wielowymiarowych grafów i rysunków służących do wizualizacji danych. Oryginalnie stworzona przez John D. Hunter'a miała za zadanie tworzyć przyzwoicie wyglądające wykresy, które można by poddać publikacji. Mimo, że inne biblioteki są dostępne większość programistów używa **Matplotlib**'a jako, że jest on najpopularniejszy oraz najbardziej i najlepiej rozbudowany ze wszystkich nie wspominając o jego wpasowaniu w istniejący ekosystem Data Science.

Matplotlib zapewnia obiektowo zorientowane **API** do tworzenia i używania generowanych wykresów w toolkitach **GUI** takich jak **Tkinter wxPython**, **Qt**, lub **GTK**. Dodatkowo istnieje również proceduralny interfejs "**pylab**" bazujący na maszynie stanu (podobnie do **OpenGL**), zaprojektowany, aby przypominać interface **MATLAB**'a.

Pyplot to moduł **Matplotlib**'a, który jest zaprojektowany z myślą bycia używalnym pod kątem programistycznym tak jak **MATLAB**, ale z dodatkową zaletą pozostawania w przestrzeni języka **Python**, która jest Open Source i darmowa.

3.5. PyGame

PyGame [9] [8] jest cross-platform'ową biblioteką stworzoną, dla języka **Python** zaprojektowaną z myślą tworzenia prostych gier i animacji. Zawiera w sobie biblioteki odpowiadające za grafikę i dźwięk. **PyGame**, był oryginalnie napisany przez Pete Shinners'a, w celu zastąpienia **PySDL** po tym jak jego developemnt został zatrzymany. Od roku 2000 jest to projekt społeczności i został objęty licencją **GNU Lesser General Public License**, która pozwala na dystrybucje **PyGame**'a zarówno wraz z Open Source'owym oprogramowaniem jak i tym prawnie zastrzeżonym prawami autorskimi.

Rozdział 4

Sieć neuronowa

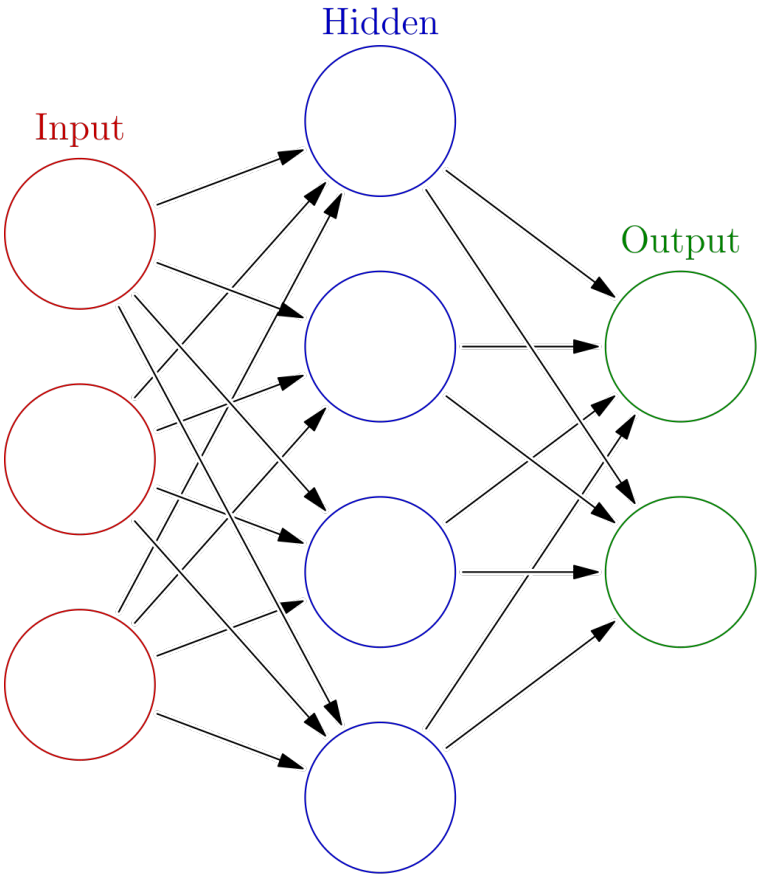
4.1. Sieć neuronowa

4.1.1. Budowa sieci

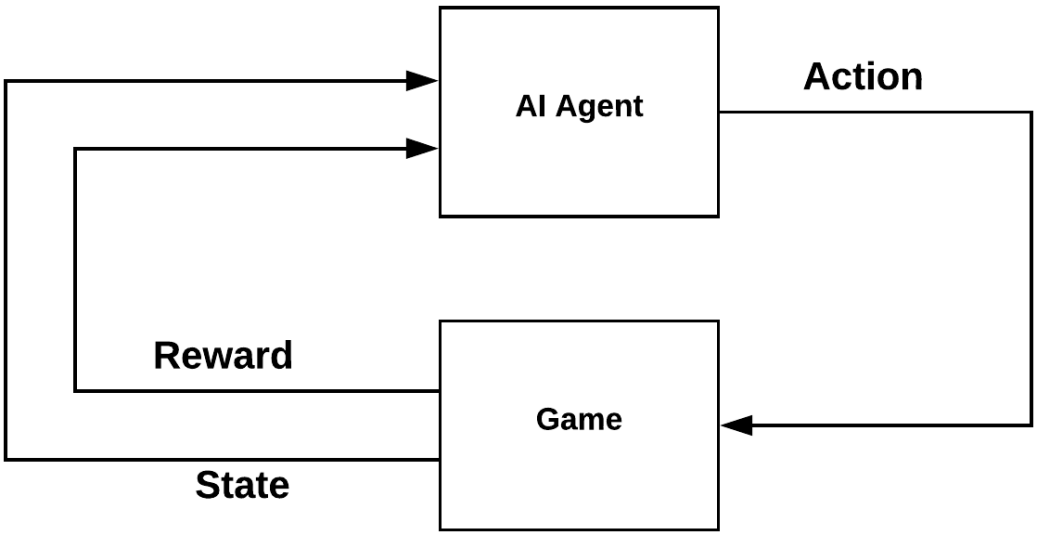
4.1.2. Optymizator

4.1.3. Strategia nauczania

4.1.4. Dane wejściowe i wyjściowe modelu



Rys. 4.1: Wykres nauki sieci neuronowej



Rys. 4.2: Wykres nauki sieci neuronowej

Rozdział 5

Paczka Python

5.1. PyPI

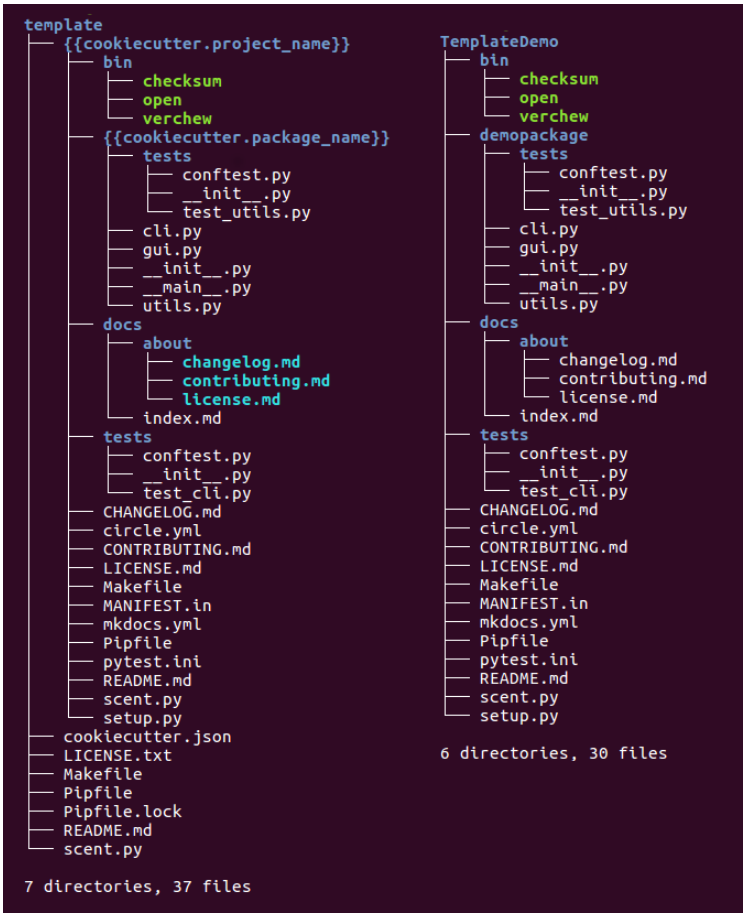
PyPI, [10] [23] czyli **Python Package Index** jest oficjalnym repozytorium, dla języka **Python** przeznaczonym na third-party paczki przeznaczone, dla języka, które może zainstalować każdy użytkownik. **PyPI** jest operowany przez **Python Software Foundation**, która jest non-profit organizacją odpowiedzialną za utrzymywanie języka **Python** i jego środowiska. Niektóre menadżery paczek takie jak **pip** [7] używają **PyPI** jako domyślnego źródła paczek i ich zależności. Na dzień 30 października 2021 roku, ponad $336\ 000$ **Python**'owych paczek może zostać w ten sposób zainstalowane. **PyPI** głównie hostuje **Python**'owe paczki w formie nazywanej "**sdists**", czyli z *ang. "**source distributions**"*, lub w formie pre-kompilowanej tak zwanej "**wheels**". **PyPI** jako indeks paczek pozwala użytkownikowi na wyszukiwanie paczek za pomocą konkretnych słów lub filtrów, przeszukując metadane paczek, takie jak licencje czy kompatybilność ze standardem **POSIX**. Pojedyncza paczka na **PyPI** może posiadać, pomijając metadane, poprzednie wersje paczki, pre-kompilowane **wheel**'e (na przykład zawierające **DLL**'e na system Windows), lub wiele innych form przewidzianych na inne systemy operacyjne i wersje **Python**'a.

Python Distribution Utilities, czyli w skrócie nazywane "**distutils**", to moduł dodany do **Python**'a, we wrześniu roku 2000, jako część standardowej biblioteki języka w wersji `1.6.1`, 9 lat po wyjściu pierwszej oficjalnej wersji **Python**'a, mając na celu uproszczenie instalacji third-party paczek. Jednakże, **distutils** dawało jedynie narzędzie służące do tworzenia paczek, dla kodu napisanego w **Python**'ie, i nic więcej. Umiało zebrać, i dystrybuować metadane paczki, ale nie używało ich w żaden inny sposób. **Python** ciągle potrzebował scentralizowanego katalogu paczek znajdującego się w internecie. **PEP 241** [1] zaproponował standaryzacje indeksów, która została sfinalizowana w marcu 2001 roku.

5.2. Cookiecutter

Cookiecutter [2] [12] to command-line'owe narzędzie, którego zadaniem jest tworzenie projektów z tak zwanych **cookiecutters**, czyli gotowych szablonów, za pomocą, których **Cookiecutter** umie stworzyć szablon paczki.

W tym celu użyty jest system templatowanie **Jinja2**, która potrafi inteligentnie zastąpić imiona i strukturę folderów oraz plików i ich zawartości. Co bardzo dobrze pokazuje poniżej zamieszczony przykład:



Rys. 5.1: Przykład działania Cookiecutter'a [16]

5.3. Tworzenie paczki

Używając gotowego szablonu **Cookiecutter**'a `https://github.com/giswqs/geodemo` tworzymy szkielet naszej paczki:

Listing 5.1: Użycie Cookiecutter'a

```
$ cookiecutter gh:giswqs/pypackage
full_name [Qiusheng Wu]: Jan Bronicki
email [admin@example.com]: janbronicki@gmail.com
github_username [giswqs]: John15321
project_name [Python Boilerplate]: Simple Game AI
project_slug [simple_game_ai]: sgai
project_short_description: Simple Game AI package allows to easily define an
interface for a game and train a simple neural net to play it
pypi_username [John15321]: John15321
version [0.0.1]: 0.0.1
use_pytest [n]:
add_pyup_badge [n]:
create_author_file [n]:
Select command_line_interface:
1 - No command-line interface
2 - Click
3 - Argparse
Choose from 1, 2, 3 [1]: 1
Select open_source_license:
1 - MIT license
2 - BSD license
3 - ISC license
4 - Apache Software License 2.0
5 - GNU General Public License v3
6 - Not open source
Choose from 1, 2, 3, 4, 5, 6 [1]: 1
Select github_default_branch:
1 - main
2 - master
Choose from 1, 2 [1]: 2
```

Wybrany szablon przygotowuje nam wiele narzędzi takie jak na przykład `mkdcos`, narzędzie służące do generowania dokumetnacji w języku **Markdown**. Dodatkowo tworzy on automatycznie configurację **CI/CD Pipeline**.

Za pomocą komendy `tree` możemy zobczyć jaką strukturę paczki wygenerował, dla nas **Cookiecutter**:

Listing 5.2: Wygenerowana struktura paczki za pomocą Cookiecutter'a

```
$ tree
.
|-- docs
|   |-- contributing.md
|   |-- faq.md
|   |-- index.md
|   |-- installation.md
|   |-- overrides
|   |   `-- main.html
|   |-- sgai.md
|   `-- usage.md
|-- LICENSE
|-- MANIFEST.in
|-- mkdocs.yml
|-- README.md
|-- requirements_dev.txt
|-- requirements.txt
|-- setup.cfg
|-- setup.py
|-- sgai
|   |-- __init__.py
|   `-- sgai.py
`-- tests
    |-- __init__.py
    `-- test_sgai.py
```

Widzimy, że **Cookiecutter** stworzył (zaczynając alfabetycznie), folder `docs` w, którym mieści się dokumentacja napisana w języku **Markdown**, plik `LICENSE` opisujący licencję **MIT** [4] na, której wydana została paczka, plik `MANIFEST.in`, który opisuje dodatkowe pliki zawarte w paczce, które nie są kodem źródłowym takie jak `README.md` i `requirements.txt`, plik konfiguracyjny `mkdocs.yml` odpowiadający za konfiguracje dokumentacji **Markdown**, plik `README.md` opisujący podstawowe informacje apropos paczki, `requirements.txt`, który wskazuje **pip**'owi wymagane zależności wymagane, dla poprawnego działania paczki, `requirements_dev.txt`, czyli wymagane zależności do pracy nad paczką, takie jak formater kodu `black`, lub `pytest` służący do automatyzowania testów, `setup.cfg`, który funkcjonuje jako plik konfiguracyjny, dla `setup.py`. Następnie widzimy folder `sgai`, który jest miejscem na kod źródłowy.

Tak prezentuje się struktura projektu po dodaniu odpowiednich plików kodu źródłowego, gdzie najważniejsze to:

- `agent.py` - tworzy i reguluje parametry sieci neuronowej
- `config.py` - posiada konfiguracje parametrów procesu nauki sieci neuronowej takie jak `LEARNING_RATE`
- `data_helper.py` - funkcje odpowiedzialne za rysowanie wykresów wizualizujących postępy nauki sieci neuronowej
- `model.py` - posiada model sieci neuronowej i jej mechanizmy nauki
- `trainer.py` - definiuje interfejs klasy abstrakcyjnej za pomocą, której użytkownik paczki łączy grę z treningiem sieci neuronowej

Listing 5.3: Struktura gotowej paczki

```
$ tree
.
|-- AUTHORS.rst
|-- docs
|   |-- authors.rst
|   |-- contributing.md
|   |-- faq.md
|   |-- index.md
|   |-- installation.md
|   |-- overrides
|   |   `-- main.html
|   |-- sgai.md
|   `-- usage.md
|-- LICENSE
|-- MANIFEST.in
|-- mkdocs.yml
|-- README.md
|-- requirements_dev.txt
|-- requirements.txt
|-- setup.cfg
|-- setup.py
|-- sgai
|   |-- agent
|   |   |-- agent.py
|   |   |-- config.py
|   |   |-- data_helper.py
|   |   |-- __init__.py
|   |   |-- model.py
|   |   `-- trainer.py
|   |-- __init__.py
|   `-- sgai.py
`-- tests
    |-- __init__.py
    `-- test_sgai.py
```

5.4. Publikowanie paczki

Następnie za pomocą skryptu `setup.py` tworzymy źródłową dystrybucje paczki, uzyskując plik z rozszerzeniem `.tar.gz`:

Listing 5.4: Generacja paczki

```
$ python setup.py sdist
...
Writing sgai-0.0.3/setup.cfg
creating dist
Creating tar archive
```

Owy plik znajduje się w nowo powstałym folderze `dist` w **root**'cie projektu:

Listing 5.5: Zbudowana source distribution paczki

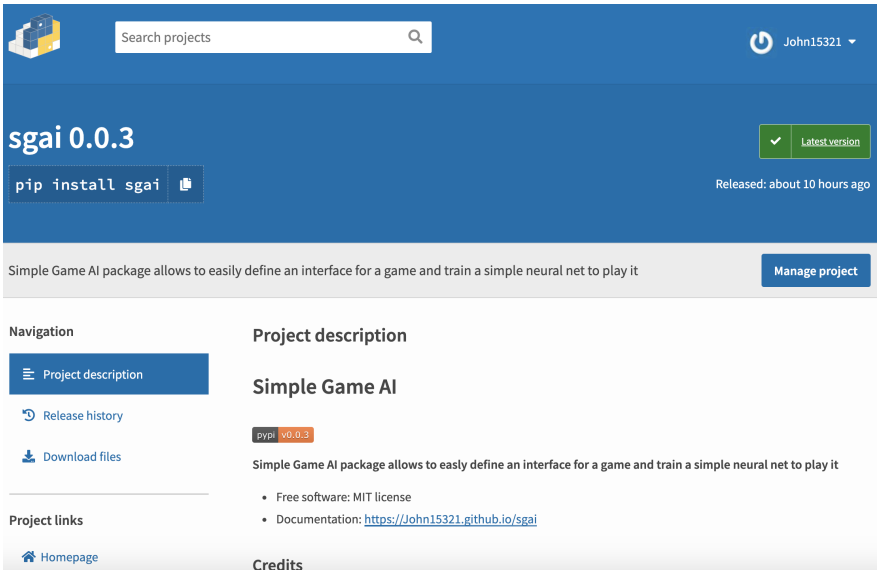
```
$ tree
.
...
|-- dist
|   `-- sgai-0.0.3.tar.gz
...
```

Następnie za pomocą narzędzia `twine` uploadujemy wygenerowany plik do indeksu **PyPI**:

Listing 5.6: Upload zbudowanej paczki na **PyPI**

```
$ twine upload ./dist/sgai-0.0.3.tar.gz
Uploading distributions to https://upload.pypi.org/legacy/
Enter your username: John15321
Enter your password:
Uploading sgai-0.0.3.tar.gz

View at:
https://pypi.org/project/sgai/0.0.3/
```

Otwierając stronę pod adresem `https://pypi.org/project/sgai` widoczna i gotowa to instalacji jest paczka `sgai`:



Rys. 5.2: Paczka sgai widoczna na stronie **PyPI**

Aby sprawdzić czy paczka została poprawnie dodana do globalnego indeksu **PyPI** w nowym środowisku możemy spróbować zainstalować paczkę `sgai` w następujący sposób:

Listing 5.7: Instalacja paczki używając narzędzia **pip**

```
$ pip install sgai

Collecting sgai
  Downloading sgai-0.0.3.tar.gz (7.4 kB)
  Preparing metadata (setup.py) ... done
  ...
```

Po zakończonej sukcesem instalacji, możemy sprawdzić czy pip widzi naszą paczke i jej wersje wykonując poniższą komende:

Listing 5.8: Wylistowanie paczki za pomocą narzędzia **pip**

```
$ pip list

Package            Version
------------------ -------
...
sgai               0.0.3
...
```

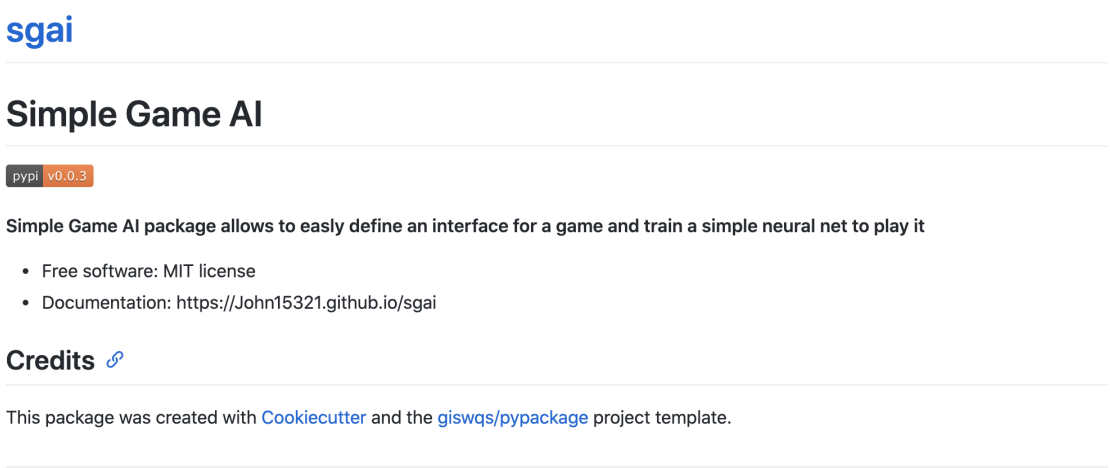
Korzystając z interaktywnego interpretera **Python**'a, **IPython**, możemy spróbować zaimportować zainstalowaną paczkę i wyświetlić jej metadane:

Listing 5.9: Importowanie zainstalowanej paczki

```
$ ipython

In [1]: import sgai


In [2]: sgai?
Type:        module
String form: <module 'sgai' from '/Users/jbron/sgai/sgai/__init__.py'>
File:        ~/sgai/sgai/__init__.py
Docstring:   Top-level package for Simple Game AI.
```

Wszystko się zgadza co oznacza, że paczka została poprawnie udostępniona i zainstalowana.

Dodatkowo możemy zobaczyć sporządzoną dokumentację na temat paczki na stronie `https://john15321.github.io/sgai/`, która jest hostowana dzięki serwisowy **GitHub** na, którym znajduje się kod źródłowy paczki:



Rys. 5.3: Dokumentacja paczki **sgai**

Rozdział 6

Użycie

6.1. Gra Snake

Snake [13] jest popularnym konceptem prostej gry komputerowej, gdzie gracz ma za zadanie manewrować poruszającą się linią w taki sposób, aby nie kolidowała ze sobą lub, dodatkowo ściankami planszy, w między czasie zbierając punkty losowo porozmieszczane po mapie. Linia ma za zadanie reprezentować ciało węża, który pełza, a punkty reprezentowane są w postaci owocu, lub prostej grupki pikseli. Odmiana użyta w celu testowania sieci neuronowej została na pisana w języku **Python**, przy użyciu paczki **PyGame**, która posłuży do reprezentacji graficznej postępowania sztucznej inteligencji, oraz samej rozgrywki. Wąż został przestawiony w postaci linii składającej się z pojedynczych zielonych kwadratów. Wąż zaczyna z trzema zielonymi kwadratami, każdy dodatkowo zdobyty punkt podłuża węża o jeden kwadrat. Punkty możliwe do zdobycia są reprezentowany w postaci czerwonych kwadratów.

Rys. 6.1: Gra Snake

Większość mechaniki gry została zaimplementowana w klasie `SnakeGame`, która odpowiada za mechanikę gry oraz przechowywanie jej stanu. Znajdują się w niej między innymi mechaniki odpowiedzialne za wykrywanie zderzeń, poruszanie się węża, sterowanie wężem, czy losowanie miejsca pojawienia się następnych punktów. Dodatkowo zaimplementowane zostały klasy enumeratorowe, takie jak `Direction` i `GamePoint` w celu zwiększenia czytelności kodu:

Listing 6.1: Importowanie gry

```
import numpy as np
import torch
from sgai.agent.trainer import Trainer

from snake import Direction, GamePoint, SnakeGame
```

Następnie definiujemy potrzebny interfejs `Trainer`, aby paczka `sgai` umiała się komunikować z grą. Zaczynami od definicji klasy i jej konstruktora. W tym przykładzie wielkość warstwy ukrytej pozostawiamy nie zmienioną ponieważ podstawowa w pełni wystarczy do poziomu skomplikowania przygotowanej gry:

Listing 6.2: Definicja klasy `MyTrainer`

```
class MyTrainer(Trainer):
    def __init__(self, game, input_size: int, output_size: int):
        super().__init__(game, input_size, output_size)
```

Następnie definiujemy dwie funkcje `get_state()`, oraz `perform_action()`. Są one interfejsem pomiędzy paczką `sgai`, a kodem gry, dlatego to użytkownik paczki musi je zaimplementować, ponieważ paczka nie ma możliwości posiadania wiedzy na temat własności działania kodu gry na, której będzie szkolić sieć neuronową. Na samym początku definiujemy stan gry, pobierając konkretne elementy rozgrywki takie jak pozycja węża, pozycja nagrody itp.:

Listing 6.3: Definicja funkcji `get_state()`

```
    def get_state(self) -> np.ndarray:
        head = self.game.snake[0]
        point_left = GamePoint(head.x - 20, head.y)
        point_right = GamePoint(head.x + 20, head.y)
        point_up = GamePoint(head.x, head.y - 20)
        point_down = GamePoint(head.x, head.y + 20)

        left_direction = self.game.direction == Direction.LEFT
        right_direction = self.game.direction == Direction.RIGHT
        up_direction = self.game.direction == Direction.UP
        down_direction = self.game.direction == Direction.DOWN
```

W tym przykładzie zebranych będzie łącznie 11 parametrów opisujących stan gry, które następnie zostaną przekazane sieci neuronowej jako wejście. Gdzie każda z wartości może być jedynie 0, lub 1, 0 oznaczające `False`, a 1 `True`:

- Czy jakieś niebezpieczeństwo znajduje się naprzeciwko?
- Czy niebezpieczeństwo znajduje się po prawej?
- Czy niebezpieczeństwo znajduje się po lewej?
- Czy wąż porusza się w lewo?
- Czy wąż porusza się w prawo?
- Czy wąż porusza się w górę?
- Czy wąż porusza się w dół?
- Czy jedzenie znajduje się po lewej?
- Czy jedzenie znajduje się po prawej?
- Czy jedzenie znajduje się nad wężem?
- Czy jedzenie znajduje się pod wężem?

Listing 6.4: Zwracany stan

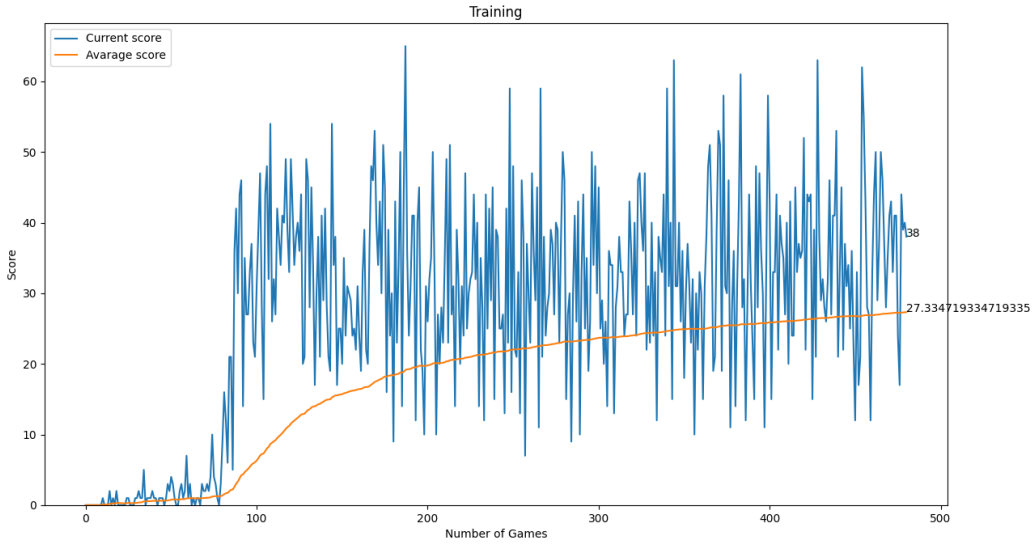
```
state = [
    # Danger straight
    (right_direction and self.game.is_collision(point_right))
    or (left_direction and self.game.is_collision(point_left))
    or (up_direction and self.game.is_collision(point_up))
    or (down_direction and self.game.is_collision(point_down)),
    # Danger right
    (up_direction and self.game.is_collision(point_right))
    or (down_direction and self.game.is_collision(point_left))
    or (left_direction and self.game.is_collision(point_up))
    or (right_direction and self.game.is_collision(point_down)),
    # Danger left
    (down_direction and self.game.is_collision(point_right))
    or (up_direction and self.game.is_collision(point_left))
    or (right_direction and self.game.is_collision(point_up))
    or (left_direction and self.game.is_collision(point_down)),
    # Move direction
    left_direction,
    right_direction,
    up_direction,
    down_direction,
    # Food location
    self.game.food.x < self.game.head.x,  # Food left
    self.game.food.x > self.game.head.x,  # Food right
    self.game.food.y < self.game.head.y,  # Food up
    self.game.food.y > self.game.head.y,  # Food down
]

return np.array(state, dtype=int)
```

Ponieważ wewnętrzna funkcja `play_step()` gry sama zwraca nagrodę, stan oraz wynik gry (które są wynikiem podjętej przez nas akcji), zaimplementowanie funkcji `perform_action()` jest bardzo proste:

Listing 6.5: Implementacja funkcji `perform_action()`

```
def perform_action(self, final_move) -> Tuple[int, bool, int]:
    reward, game_over, score = self.game.play_step(final_move)
    return reward, game_over, score
```

Jak widać na poniższym wykresie w okolicach setnej rozgrywki, sieć neuronowa znacznie się poprawia. Następnie przez kolejne kilkaset gier widoczny jest jedynie marginalny postęp, przez to można stwierdzić że nastąpiło nasycenie spowodowane prostą budową sieci.



Rys. 6.2: Wykres nauki sieci neuronowej

Literatura

[1] Pep 241 – metadata for python software packages. https://www.python.org/dev/peps/pep-0241/, 2001.

[2] Cookiecutter: Better project templates. https://cookiecutter.readthedocs.io/, 2022.

[3] Matplotlib. https://en.wikipedia.org/wiki/Matplotlib, 2022.

[4] Mit license. https://en.wikipedia.org/wiki/MIT_License, 2022.

[5] Numpy. https://en.wikipedia.org/wiki/NumPy, 2022.

[6] Numpy manual. https://numpy.org/doc/stable/, 2022.

[7] pip (package manager). https://en.wikipedia.org/wiki/Pip_(package_manager), 2022.

[8] Pygame. https://en.wikipedia.org/wiki/Pygame, 2022.

[9] Pygame documentation. https://www.pygame.org/wiki/GettingStarted, 2022.

[10] Python package index. https://en.wikipedia.org/wiki/Python_Package_Index, 2022.

[11] Pytorch. https://pytorch.org/docs/stable/index.html, 2022.

[12] Readmme.md. https://github.com/cookiecutter/cookiecutter, 2022.

[13] Snake (video game genre). https://en.wikipedia.org/wiki/Snake_(video_game_genre), 2022.

[14] Users guide. https://matplotlib.org/stable/users/index, 2022.

[15] E. Bressert. *SciPy and NumPy*. O'Reilly Media, Inc., 2012.

[16] H. Canto. Project templates and cookiecutter. https://medium.com/worldsensing-techblog/project-templates-and-cookiecutter-6d8f99a06374, 2018.

[17] B. Lubanovic. *Introducing Python, 2nd Edition*. O'Reilly Media, Inc., 2019.

[18] E. S. Luca Pietro Giovanni Antiga, Thomas Viehmann. *Deep Learning with PyTorch*. Manning Publications, 2020.

[19] W. McKinney. *Python for Data Analysis, 3rd Edition*. O'Reilly Media, Inc., 2021.

[20] T. Z. Michal Jaworski. *Expert Python Programming - Fourth Edition*. Packt Publishing, 2021.

[21] S. R. Poladi. *Matplotlib 3.0 Cookbook*. Packt Publishing, 2018.

[22] Wikipedia. Python (programming language). `https://en.wikipedia.org/wiki/Python_(programming_language)`.

[23] M. Wilkes. *Advanced Python Development: Using Powerful Language Features in Real-World Applications*. Apress, 2020.

[24] A. Zamczała. Python ma już 30 lat! od czego wszystko się zaczęło? 2021.

Spis rysunków

Spis tabel

Indeks rzeczowy