***Inżynieria Oprogramowania***

**Grupa: N22-31**

**Rok akademicki: 2020/2021**

**Skład zespołu:**

Aleksander Witocha - 269944

Magdalena Staniec - 269947

**Tytuł projektu:**

Snake – with – AI

**Opis instalacji aplikacji:**

**Cel projektu:**

Projekt miał na celu stworzenie nienadzorowanej sieci neuronowej uczącej się grać w grę Snake.

**Biblioteki Zastosowane w projekcie:**

Pygame, które daje nam do dyspozycji graficzny interfejs, co za tym idzie umożliwia wyświetlanie grafiki, odtwarzanie dźwięków, śledzenie czasu, obsługę myszy itd.

Biblioteki PyTorch, która ułatwia budowanie projektów głębokiego uczenia.

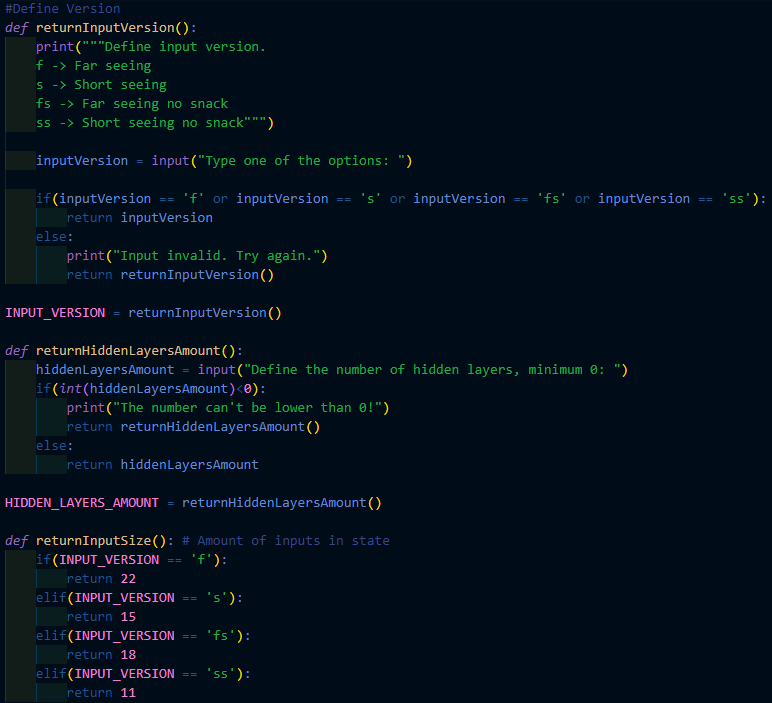
Matplotlib ipython ze względu na niezliczone biblioteki rozszerzające możliwości, pozwalające na tworzenie interaktywnych arkuszy mogący zawierać kod wykonywalny, opisy, tabele, wykresy i wiele innych danych.

DeepQ Learning (rozszerzenie Reinforced Learning) – zastosowanie w dziedzinach, które wymagają danych nienumerycznych np. w klasyfikacji obrazów, wykrywaniu obiektów i opisywaniu zawartości takich zbiorów

**Funkcjonalność:**

Zbieranie danych o otoczeniu głowy węża, zapętlone kroki oceniające sytuację i wykonujące ruch, rozdawanie punktów za odpowiednio wykonane działanie.

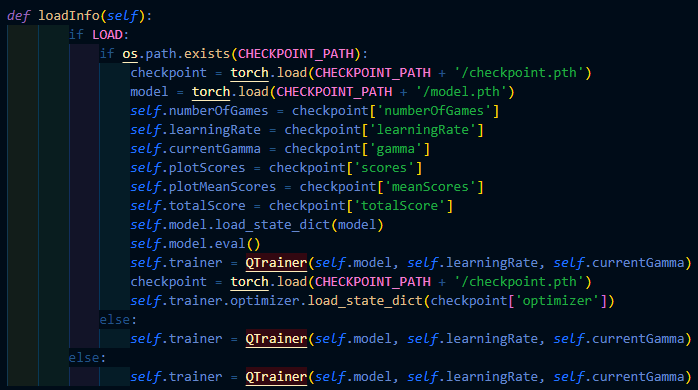
Dodatkową funkcjonalnością jest interfejs wspomagający wybór wersji:



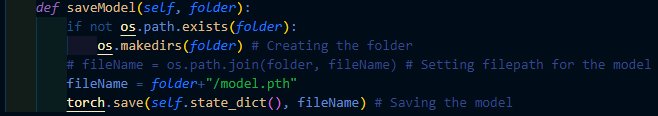
Takie wpłynięcie na pracę programu okazało się jednak bardzo szkodliwe dla funkcjonowania całego modelu obniżając jego wyniki bądź, w niektórych przypadkach, całkowite uniemożliwienie uczenia się modelu.

Dodatkowo całość została wyposażona w możliwość wczytywania i zapisywania obecnie używanego modelu, nie wpłynęło to na pracę programu.

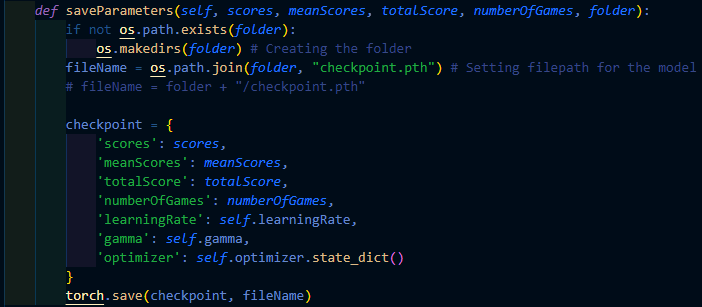
**Wczytywanie danych:**



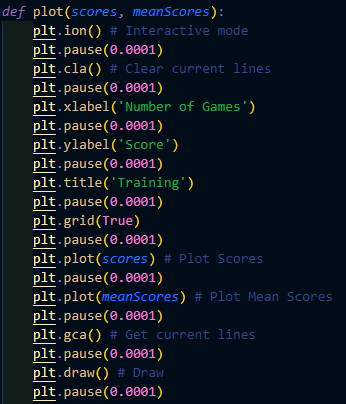
**Zapisywanie modelu:**



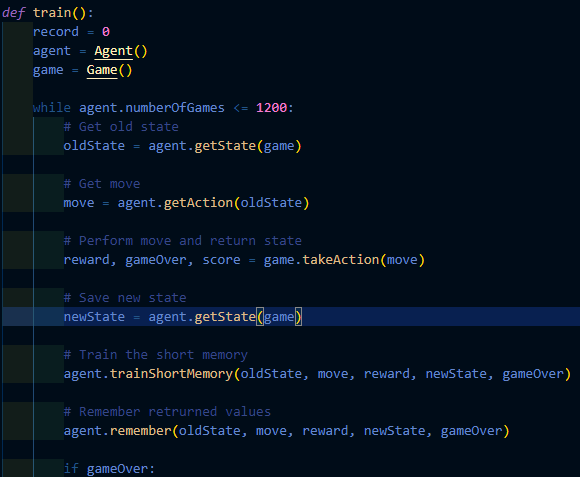
**Zapisywanie parametrów modelu:**



**Dane uzyskane w ten sposób są zapisywane w formie wykresu przy użyciu biblioteki matplotlib**



### Flow przewodni aplikacji:

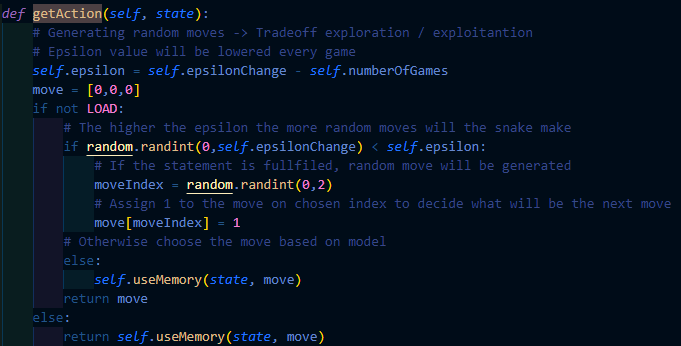
****

Każdy krok wykonany przez agenta (sterownik modelu) jest pętlą wykonujących się po sobie funkcji:

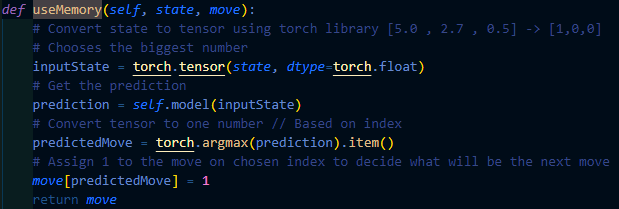
**1. getState(game)** - zbiera informacje o otoczeniu głowy snake’a. (na screenie kawałek kodu ponieważ jest bardzo długi)



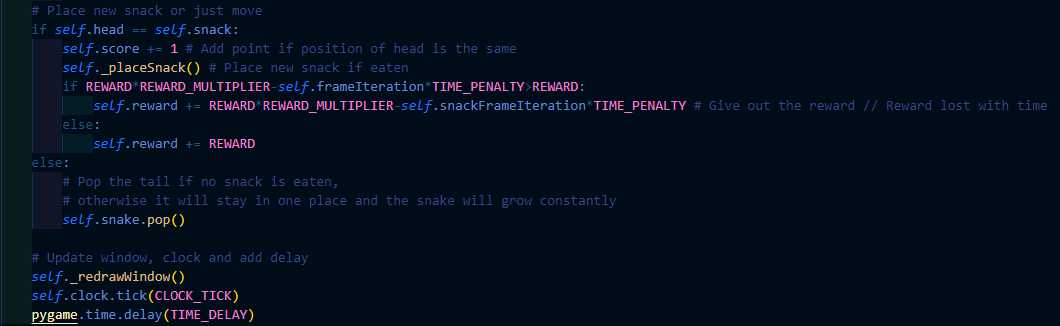
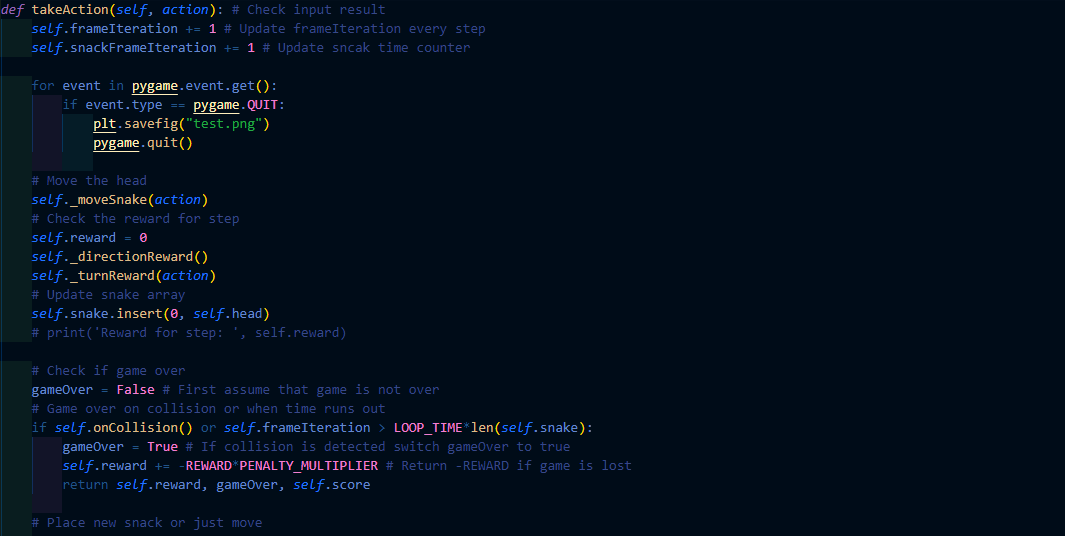
**2.1 getAction(oldState)** - podjecie decyzji o następnym kroku na bazie informacji o otoczeniu.



**2.2. useMemory(state, move)** - na podstawie informacji o poprzednich krokach podejmowana jest decyzja jaki krok powinien zostać podjęty jako następny. Dzieje się to za pomocą wewnętrznej funkcji biblioteki PyTorch, wybierany jest najbardziej prawdopodobny właściwy krok.



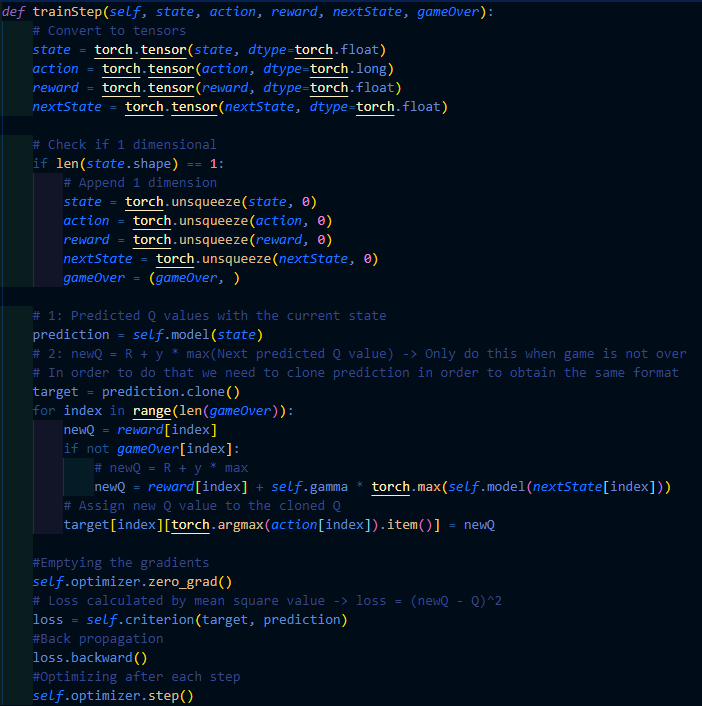
**3. takeAction(move)** - funkcja środowiskowa odpowiadająca za mechanikę poruszania się snake’a, rozdanie nagrody za ruch, sprawdzanie czy spełniło się kryterium końca gry (kolizja) oraz przekazanie obecnego wyniku. Konkretne funkcje używane w tej metodzie są mechaniczne odnoszące się do biblioteki PyGame bądź poszczególnymi funkcjami wpływającymi na nagrodę pod kątem ruchu.



**4.** Następnie wywoływana jest drugi raz funkcja getState zbierająca informacje po wykonanym kroku.

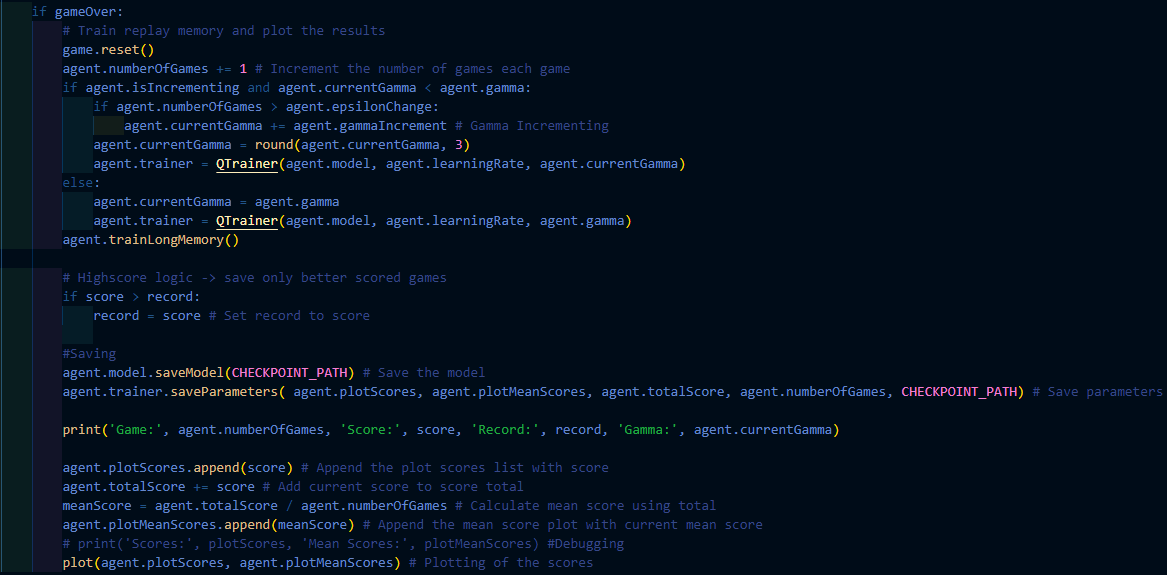
**5. trainShortMemory(oldState, move, reward, newState, gameOver)** - Pozyskane informacje używane są do treningu pamięci krótkotrwałej, porównującej informacje z kroków. Do porównania używamy modelu regresji liniowej i równania Bellmana.





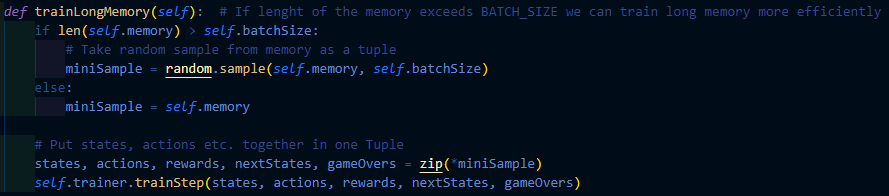
**6. remember(oldState, move, reward, newState, gameOver)** - na zakończenie każdego kroku zebrane dane są zapisywane w pamięci w celu uczenia pamięci długotrwałej.

**W przypadku spełnienia się kryterium końca gry plansza oraz stan gry zostają zresetowane, liczba gier oraz niektóre parametry uczenia się zaktualizowane, a stan modelu wraz z obecnymi parametrami, zapisany.**

****

Wywoływana jest także funkcja uczenia pamięci długotrwałej.

**7.1** **trainLongMemory()** - funkcja ta pobiera losową grę z pamięci i używa jej do trenowania modelu



**7.2 trainStep()** -

