# General description

The chatbot will be based off of “The Witcher” series, focusing mainly on “The Witcher 3: The Wild Hunt” video game. The Witcher 3 is a video game with complex lore where the primary goal is killing various monsters. These monsters will usually have a set of weaknesses which can be exploited by the player to kill that monster more easily. Some example can include certain oils that the player can craft and apply to their weapon and signs (a sign is effectively a spell) that the player can use such as Igni (Sign that produces a small burst of fire). The chatbot will allow the user to ask some questions about the game, for a description of the enemies in the game and what weakness they have that the player can exploit.

System requirements:

* The chatbot should allow for some “small talk” such saying hello and asking how they are
* Chatbot should allow the user to tell them their name and remember it in future outputs
* The chatbot should allow the user to ask for a description all of the different monsters
* The chatbot should allow the user to ask for a list of things that a particular monster is weak to
* The chatbot should be able to answer some basic questions about The Witcher 3 and the Witcher series
* The chatbot should allow the user to exit the program by saying goodbye

# AI techniques:

## Stage 1

One of the AI techniques that will be employed is Rule based. I will use an AIML file to define patterns for user input. When the user input matches that pattern, there will be a defined template to decide what the output should.

Another technique that was used was TF\*IDF, using the bag of words model and Cosine similarity. This is used to calculate how similar a string is to a set of other strings. TF is term frequency. This refers to the number of times a term is present in a particular document. A document in this context being a predefined question or the input from the user. IDF is inverse document frequency. This is calculated by doing the number of documents divided by the number of documents that contain a word. The Log function is used to dampen the weighting of the IDF score. Tf and idf are represented as matrices and multiplied together. The result is a matrix containing values close to 0 which is used to determine the importance of each word.Cosine similarity is then used calculate the similarity of each document to each other document in the in the data set. Cosine similarity works by applying a formula to two matrices to work out the cosine angle between them in a multi-dimensional space. The smaller the angle, the less of a difference between the two documents. This is used to find the question with the most similarity to the users input.

## Stage 2

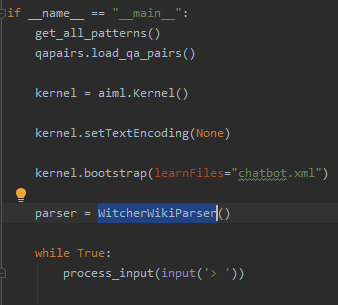
The main AI technique used for stage 2 is a Convolutional Neural Network which is used to create an image classifier. Neural Networks work by building a model of nodes with 3 main layers. An input layer, a hidden layer and an output layer. The network is the provided labelled data, known as training data which is reshaped to fit the nodes of input layer. **(Each node in each hidden layer takes a sum of the nodes at the previous layer, but multiplies the value of every node by a weight to determine how influential each node at the previous layer is to the current node. An extra number known as a bias can be added to this weighted sum to influence how high this weighted sum of nodes needs to be before the current node is significantly activated)**. At the output layer the node with the highest activation is considered to be the best fit. These output nodes will usually correspond to different categories or classes that the programmer defines. During the training process, if the output does not match the label of the data, back propagation is used to alter the weights of the hidden layers to alter the activation of the outputs nodes to be closer to what it should be.

Convolutional Neural Networks are Neural Networks that use image pixel data as the input layer, and convolutions layers for its hidden layers. The convolutional layers apply arbitrary kernel convolutions to manipulate the image. **(Describe what kernel convolutions actually are)**

# Explaining the code

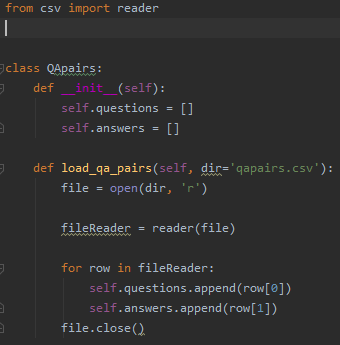
## Stage 1

At the beginning of code being executed a few different variables are initialised.

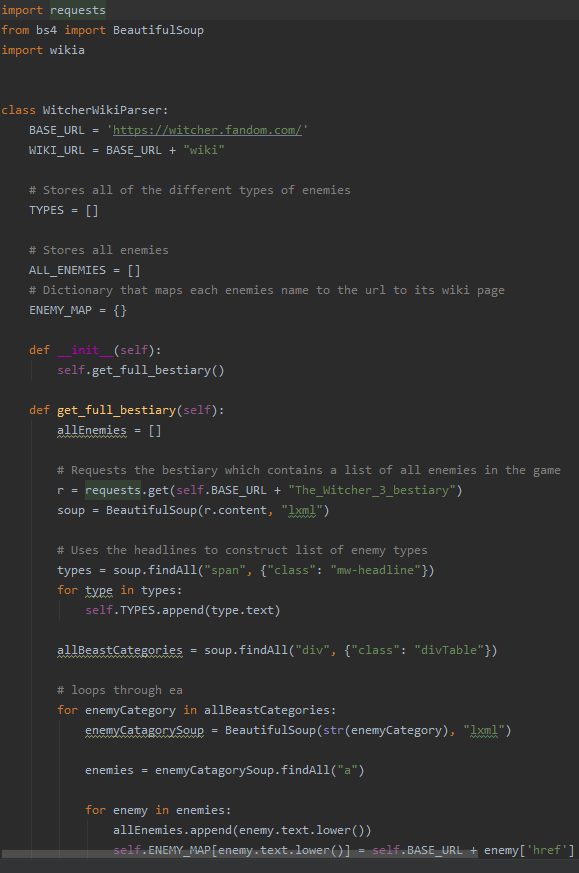


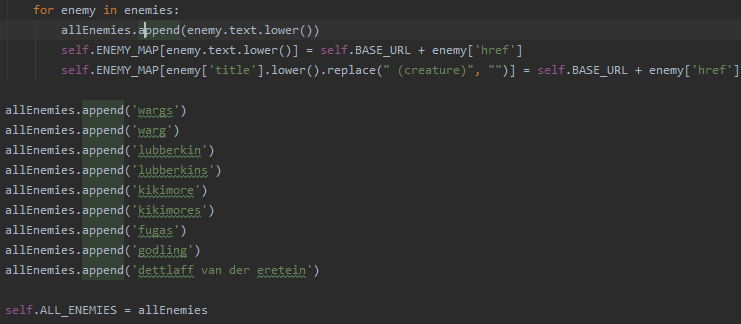
The kernel is initialised and passed the AIML file.

A function is called to populate a list two lists with questions and answers from the csv file:

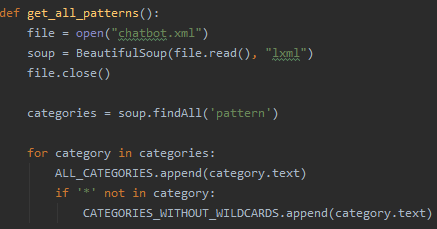


An instance of the Witcher wiki parser is created. This is used to get descriptions and weaknesses of enemies. The constructor pre loads a list of all of the different enemies, as well as a dictionary mapping each enemies name to it wiki page url.





Back in the main function, another function is called to load all of the patterns in the AIML file. This will be used later:



An infinite while loop calls the process input function, passing in the console input as a parameter



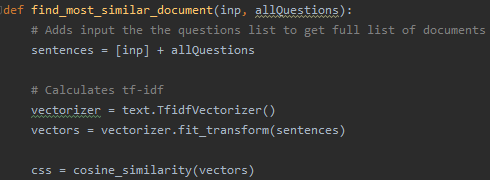
The process input function is where the chatbot logic is defined. Here the user input is stripped of any punctuation, converted to lower case, and passes the input to the kernel. The output is then processed, using a $ to signify that the input needs further processing and # followed by a number to determine which function is run what numbers and parameters is returned is determined by the AIML file:



The default reaction in the AIML file will signal the python code to do tf-ifd and cosine similarity. First the users input, questions from the csv file and patters without wild cards are passed into the function.



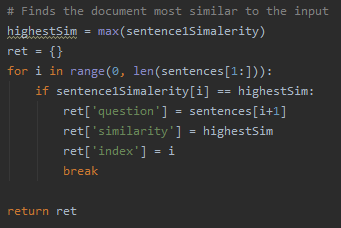
These three inputs are concatenated into one list. The Sci-kit learn package is then used to perform the TF-IDF and cosine similarity operations:



Once that has been done, the input similarity vector is extracted from the matrix. Its similarity to itself is stripped from the list



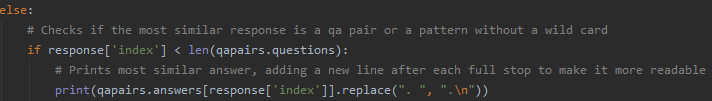
The most similar question is found. The question its self, the similarity score, and its position in the list is returned by the function in a dictionary:



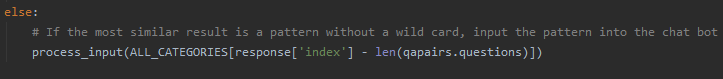
If the similarity score is 0 then the chat bot will ask the user to try and rephrase there sentence:



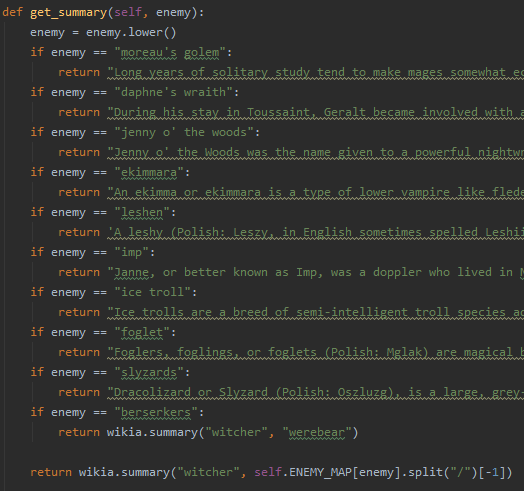
If the most similar answer was in the Q and A pairs, then the answer for that question is indexed and printed by the chat bot.



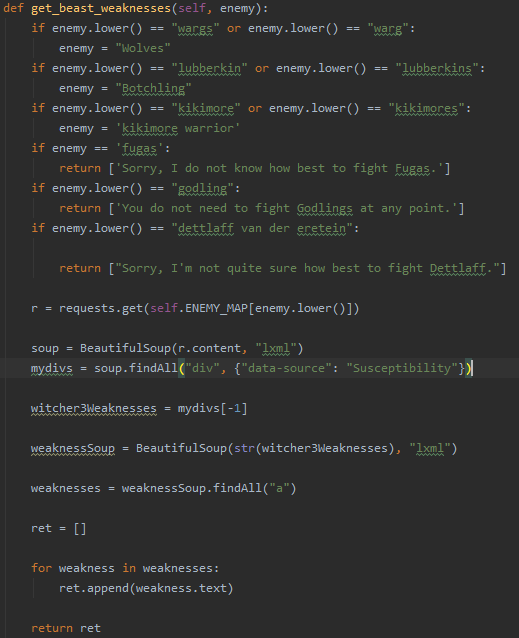
If the most similar document was one of the patterns that do not contain any wild cards, then that patter is entered into the chatbot again. This allows AIML file to be simplified and not include lots of very similar patterns where the patterns do not have any. Implementing this with patterns with wild cards would be slightly more difficult.



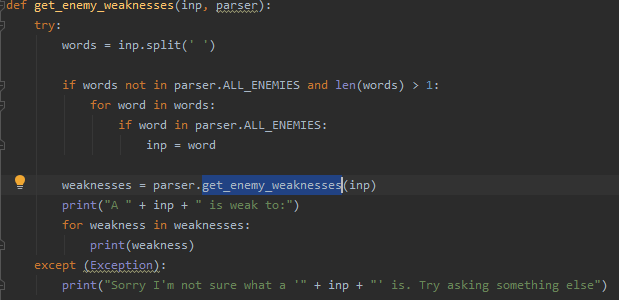
If the users requests a description of an enemy, the wikia python package is used to get this from the Witcher wikia page. This did not work with all enemies, hard coded return values where returned in these cases.



The wikia package did not have a working way that I could find to extract the weaknesses of an enemy. When the user requests the weaknesses of an enemy, wiki page for the enemy is parsed to scrape the weaknesses. Work arounds have been included for enemies that did not have susceptibility on their wiki page:



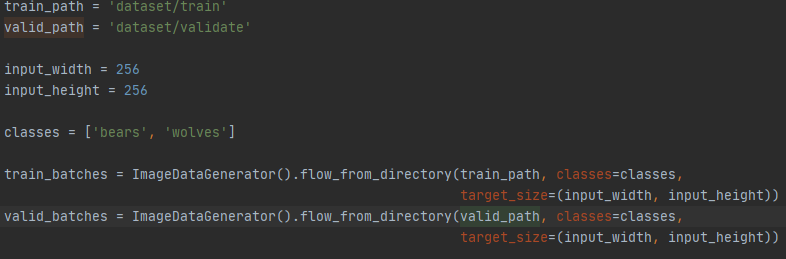
Both functions which call the get weakness and get description are very similar and use a snippet of code which tries to find the input in the list of all enemies. If it cannot be found, then it will try splitting the input into words to see if any unwanted words were also caught by the wild card:



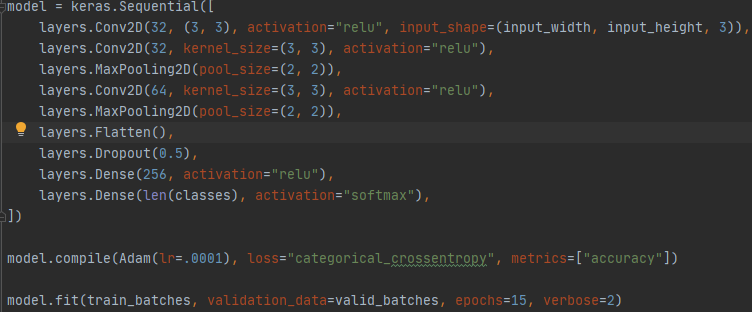
## Stage 2

The Python script which trains the classifier is in Train\_CNN.py

First the training and validation data is loaded into image data generator

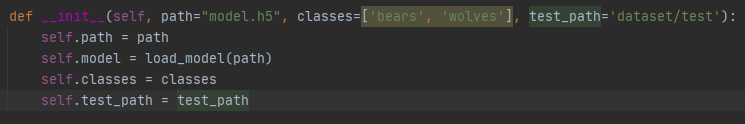


The model is constructed, compiled, and then trained on the training data



The file Trained\_CNN\_Wrapper contains a class which wraps the code required to load the trained model and make a prediction on an image.

The constructor loads the model



The predict\_local\_image member takes a file path as an argument, loads the specified image file, processes the image into the format that the model expects, and runs the image through the trained model, and returns the class that the model predicted.

