GENRE: The Movie Recommending Chat Bot

A dissertation submitted in partial fulfilment of The requirement for the degree of MASTER OF SCIENCE in Software Development

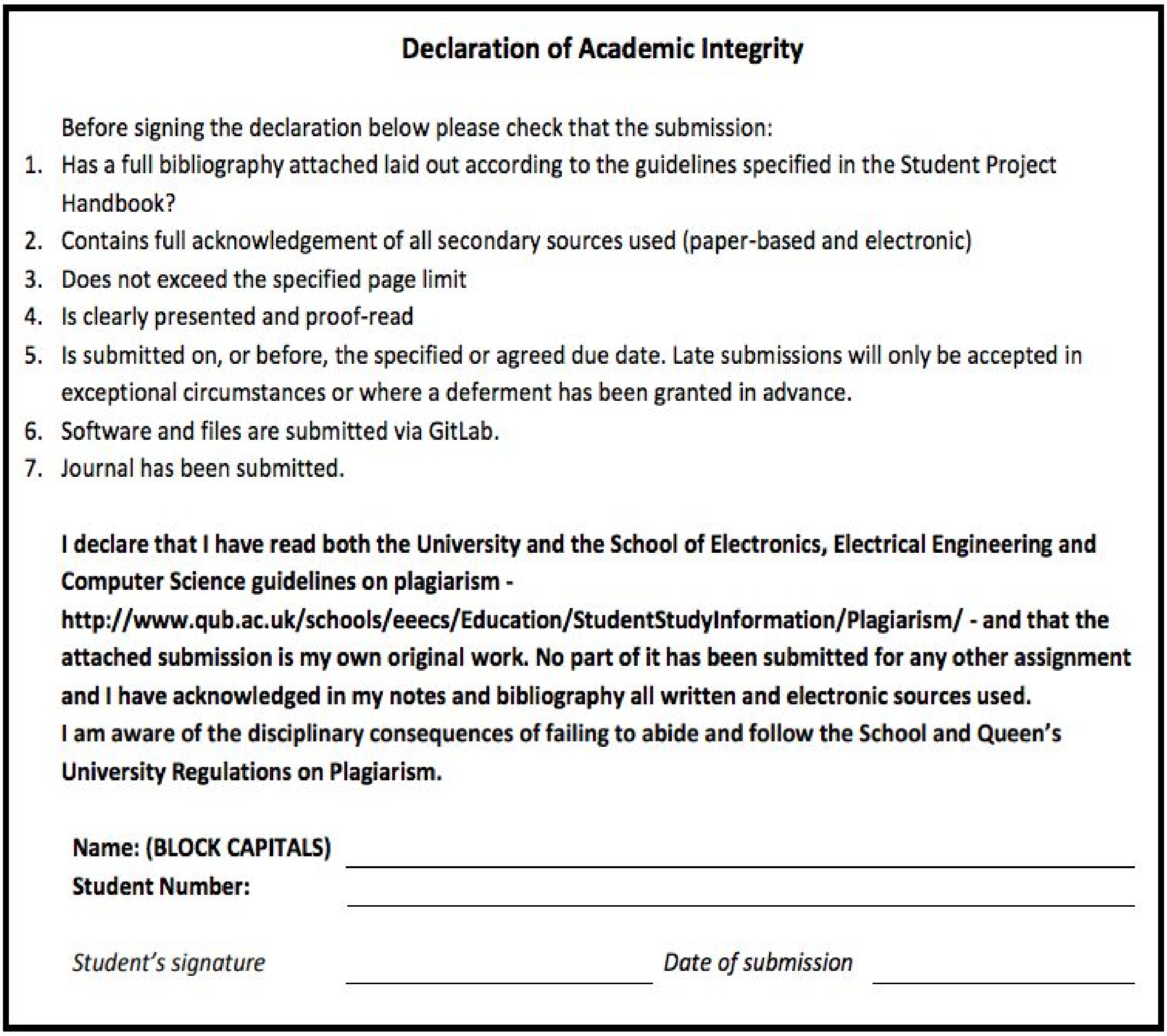
In

The Queen's University of

By

John Higgins

12/09/2018



Declaration of Academic Integrity

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook?
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.
6. Software and files are submitted via GitLab.
7. Journal has been submitted.

I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.
I am aware of the disciplinary consequences of failing to abide and follow the School and Queen's University Regulations on Plagiarism.

Name: (BLOCK CAPITALS) ______________________
Student Number: ______________________

Student's signature ______________ *Date of submission* ______________

Acknowledgements

I would like to thank my Mother, my Supervisor Ian and my friend Raymond,

We Have Always Lived in the Castle

Abstract

The basis of this project was to build a Python and SQL chat bot that talks to the user about their favourite movies and genres. Through conversation the bot will glean their preferences and using collaborative filtering techniques with a movie database recommend movies for them to watch in the future. It's conclusions were that although language processing for computers can be difficult and frustrating for the user to interact with, if you keep the subject matter constrained and conversation guided it can be an effective way for the user to get information and enjoy themselves.

By John Higgins

Table Of Contents

Page

1. Problem Specification……………………………………………………………………………7

1.1 Introduction ….……………………………………………………………………..7

1.2 Objectives……………………………………………………………………...7

1.3 Target Users….……………………………………………………………...7

1.4 User Story….……………………………………………………………………..8

1.5 Possible Challenges ….…………………………………………………………...8

1.6 Advantages and Disadvantages Of ELIZA….………………………………………8

1.7 Software Used….…………………………………………………………….....9

1.8 Sample Conversation ….…………………………………………………………9

1.9 Problem Timeline….………………………………………………………….10

Fig 1. Gant Chart for Problem ….………………………………………………..11

2. Proposed Solution and Justification for the Development Model…………………………….12

2.1 RAD Model and TDD….…………………………………………………….12

2.2 Unified Process ….………………………………………………………...13

3. Requirements Analysis and Specification….………………………………………………….14

3.1 Test Case for user….……………………………………………………………15

3.2 Use Case Diagram ………………………………………………………………18

3.3 Use Case Tables …………………………………………………………………...19

3.4 Lists of Functions in Use Cases ……………………………………………….....22

4. Design ……………………………………………………………………………………....26

4.1 User Interface Design…………………………………………………………..26

4.2 Software System Design ………………………………………………………..27

4.3 Questioning the Users Tastes …………………………………………………………..29

4.4 User Database Design ………………………………………………………………….29

4.5 Film Database Design ………………………………………………………………….30

4.6 Using Python and SQL to Query the Database …………………………………………31

Fig 4 Sequence Diagram ………………………………………………………………………32

Fig 5 UML Diagram …………………………………………………………………………..33

Fig 6 Use Case for Whole System ……………………………………………………………..34

4.8 Using IMDBpy ……………………………………………………………………….....35

5. Implementation ……………………………………………………………………………..36

5.1 Filtering phrases that if returned are deemed undesirable ……………………………....36

5.2 Constructing a response based on a given sentence ………………………………………38

5.3 Identifying the Word Classes ………………………………………………………….....39

5.4 When the user is talking about the bot…………………………………………………….39

5.5 Querying the database based on the user's inputted movie………………………………....41

5.6 Using IMDBpy………………………………………………………………………………41

5.7 Testing the Bot………………………………………………………………………………42

5.8 Looking back to the use cases in Chapter 3………………………………………………...43

5.9 Non Functional Requirements ……………………………………………………………….49

6. Conclusions……………………………………………………………………………………..51

6.1 Ideas for Future Iterations ………………………………………………………………....51

Appendices ……………………………………………………………………………………….53

References ………………………………………………………………………………………..59

Introduction

In the fight for eyeballs the internet streaming giants have lost their own. They cannot see that their users are overwhelmed by the shear amount of choices on offer. Using Collaborative Filtering and Natural Language Processing GENRE is a language based movie recommendation system. Collaborative filtering was described well in Pedro Domingos The Master Algorithm: "In 1994, a team of researchers from the University of Minnesota and MIT built a recommendation system based on what they called ‘a deceptively simple idea’: people who agreed in the past are likely to agree again in the future. That notion led directly to the collaborative filtering systems that all self-respecting e-commerce sites have."

The key sections of this thesis are as follows;

Problem Specification- Outlining the problem of users having too much choice, what users the bot would be useful to and how it will be tackled

Proposed Solution and Justification of the Development Model- Describing what software development techniques will be used and the reasons why.

Requirements Analysis and Specifications- Analysing what users want from the program through use cases and functions.

Design- Both the interface and the system, laying out the plans and what tools were used.

Implementation- What was done and how it was tested.

Conclusions- Setting out what was learned and the successes and failures of the bot. Ideas for future iterations and thoughts on bots having a personality of their own.

1. Problem Specification

1.1 Introduction

The problem with modern streaming services and television is the shear amount of content and deciding upon what to watch. I propose to create a chat bot for those out there who do not have the time or the ability to browse endlessly or to research online what they might like. The bot will have the added benefit of talking about films the user has seen and perhaps learn or remember some things about them. The chat bot will be in the style of Joseph Weizenbaum's ELIZA connected to a SQL database that will be able to cross reference the films the user and the bot have talked about and offer similar suggestions.

1.2 Objectives

- Provide an easy to use application that offers a conversation about film and suggestions on what the user may like to watch next.
- Save the user's preferences of genres and films they have seen in order to recommend different ones for future watching.
- Ask follow up questions when the user logs back in to see if they watched what was recommended and what they thought of it.
- To save the follow up data with the aim of using it to provide similar suggestions to users with similar profiles.
- To create a real human like chat experience that will leave the user wanting to come back to report what they thought of the suggestions and ask for more.
- Provide a lightweight web app interface with login features to create profiles on what the user likes and has seen.

1.3 Target Users

- Film enthusiasts who feel they are overwhelmed by the choice offered them in this streaming age and want a simple way to decide what to watch.
- Computer novices who don't have the capacity, knowledge or time to research what films to watch and want a simpler chat like experience to figure out what they like.

- Computer enthusiasts who want a novel, more personalised approach to their viewing habits.
- The lonely or elderly who wish to talk about films they loved in their youth and to help them remember that.
- Children who don't know how to browse the web to a certain extent and want new films to see.

1.4 User Story

- I want to be able to get the answer I need as quickly as possible so that I don't have to wade through internet film databases on my own.
- I want the bot to be sophisticated enough as if I were talking to a live human being so that I don't get frustrated.
- I want to chat with the bot in a way that I am accustomed to, like within an instant messenger so that I feel comfortable using the bot.
- I want the chat bot to remember what I have seen and my film preferences so I don't have to.

1.5 Possible Challenges

- The main challenge I envision will be creating a seamless chat experience that allows users to experience a more personalised approach.
- Main challenge will be the SQL database system – how to recommend similar suggestions to what the user and the chat bot have talked about. Cross referencing similar genres and films by the same director will be a good start, progressing to a separate database that will be filled by the users themselves and what they thought of the recommendations with the aim to the system improving as the volume of users increases.

1.6 Advantages and disadvantages of using an ELIZA style chat bot

- Reach a broad audience, can also integrate with Facebook messenger if desired.
- Ability to automate personalized messages and recommendations.
- Has utility, unlike the ELIZA computer therapist which was designed to demonstrate the superficiality of conversation between man and machine.
- Will be purpose specific, will get confused and stuck if users go off topic.

- Keyword reliant, problems with repeated similar questions.

1.7 Software

- HTML/PHP - For building the web app.
- Python - Coding the Chat Bot.
- Javascript – So the web app can talk to programme.
- SQL - Connecting the web app to the code and database.
- TensorFlow – Could improve efficiency of the computations to find suggestions.

1.8 Sample Conversation with Explanations

User: Hey

Bot: Hello Clarice.

Here we can see it’s just a simple matter of searching through greeting keywords and the bot will select a random greeting from a pre-existing set.

Bot: So what kind of film genres are your favourite?

User: Sci-Fi

Here the bot will first ask the user questions on what their favourite genres are and add that to his profile for future reference.

Bot: OK cool! Sci-Fi is also one of my favourites 🏝 Have you seen “Ex-Machina” or “Sunshine”?

Here the bot will select some films from the database with a high rating in the Sci-Fi genre and ask the user if they have seen them.

User: Yes

Bot: Great, if you wouldn’t mind to build a profile on you I would like you to rate these films respectively and then give me some more of your favourite films. Once you rate enough films I will cross check that with my database and can let you know what other users similar to you have loved.

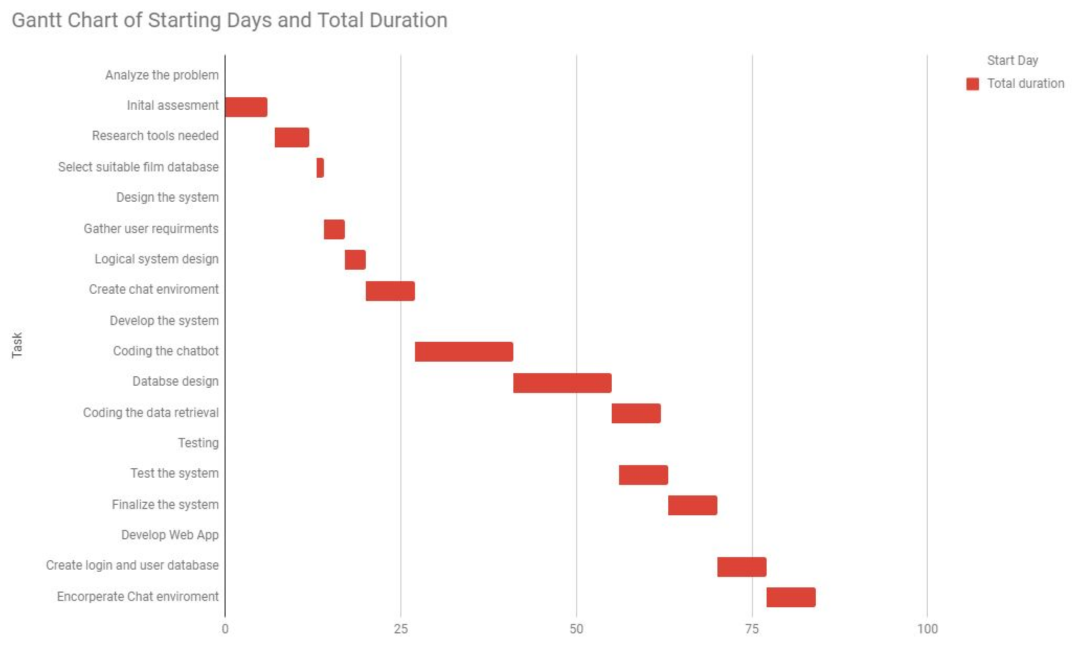
Once the user has given over ten films the bot can cross reference these with the database to see what users with similar profiles have liked. The more films given over and the more ratings accompanying them the more accurate the recommendations will be.

1.9 Problem Timeline

Task	**Start Date**	**End Date**
Analyse the problem		
Initial assessment	14/06/2018	21/06/2018
Research tools needed	21/06/2018	27/06/2018
Select suitable film database	27/06/2018	28/06/2018
Design the system		
Gather user requirements	28/06/2018	01/07/2018
Logical system design	01/07/2018	04/07/2018
Create chat environment	04/07/2018	11/07/2018
Develop the system		
Coding the Chabot	11/07/2018	26/07/2018
Database design	26/07/2018	09/08/2018
Coding the data retrieval	09/08/2018	10/08/2018
Testing		
Test the system	10/08/2018	17/08/2018
Finalize the system	17/08/2018	24/08/2018

Develop Web App		
Create login and user database	24/08/2018	28/08/2018
Incorporate Chat environment	28/08/2018	07/09/2018

Fig. 1 Gant Chart of Problem Timeline



2. Proposed solution and justification of the development model

A Disciplined Agile Delivery (DAD) consists of a hybrid approach where you take what would suit you from Scrum with proven strategies from Agile Modelling (AM), Unified Process (UP) and Rapid Application Development Model(RAD). To keep things working along there are fortnightly meetings with my supervisor Dr. Ian O'Neil which in the beginning will result in a lot of research, note making, practising making simple chat bots in Python and then moving on to creating the three main parts of the programme mentioned below.

A justification for using an Agile flavoured approach would be mainly Test Driven Development. To see if the chat bot would meet the user requirements I set out in my Proposal. The need to constantly test it from a user perspective is paramount. To make the chat bot be seamless, as if the user was having a real conversation with someone who knew a lot about movies.

Given the short space of time a Rapid Application Development Model would be appropriate.

2.1 RAD Model and Test Driven Development

RAD model is an incremental software development process model that emphasizes an extremely short development cycle. As set out in the proposal for it to be a functioning chat bot that in the end was used for people who are overwhelmed by the choice offered to them by modern streaming services. Emphasising reuse, as mentioned earlier TDD would be appropriate, to constantly testing the chat bots code. Not all type of projects are appropriate for RAD. The system must be able to be split up into smaller parts, if this is not possible there could be problems. I summarise that the project could be split up into these three parts.

- Python Chat bot Code
- SQL statements to search the database based on what user inputs (Movies) the chat bot was receiving and output recommendations
- A working interface that the user would find easy to use

2.2 Unified Process

By working at these parts using Unified Process techniques such as an incremental and iterative development processes the project should begin to take shape. Unified Process has several stages;

1.1 Inception – The inception of the project was taken care of in the Project Proposal

1.2 Elaboration – Gather the projects requirements, in my case this would be the user requirements. To create a system architecture. How will the projects different parts integrate and communicate. To demonstrate that the architecture will support the key system functionality and exhibit the right behaviour in terms of performance and scalability.

1.3 Construction- The largest part of the project, incrementally tackling the projects requirements as set out in the Proposal; Easy to use, create human like chat experience, save user preferences etc. System features should be implemented in a series of short iterations. Write small snippets of what I want the chat bot to do then test it extensively. Moving on to the next questions or conversation lines laid out in the User Stories.

As mentioned earlier, fortnightly meetings with my supervisor would lead to a sprint of writing code for the chat bot in large bursts before the meetings to get the maximum out of each meeting.

In the end my choices for the development model proved apt. A combination of Unified Process, RAD and some Agile flavouring helped in keeping the project on track. Splitting the project up into three main parts using UP helped in keeping things manageable. Although the different sections overlapped quite frequently the weeks doing SQL statements and database management proved very efficient as I was well practiced, as did the weeks writing the code for the bot. Its technique of incrementally tackling the projects user requirements also kept things manageable.

The TDD aspects of the Agile approach were also invaluable, by testing the bot frequently and after any updates I was able to find bugs early on and create error conditions for them. Saving time later on.

3. Requirements Analysis and specification

To identify the requirements of my chat bot a Use Case approach would be most valuable.

By writing out reasons why a user wishes to use the bot we can identify what the user is coming to the chat bot for.

1. Small Talk- For less technically inclined people, older or younger. The bot starts by asking for their name and then it will seem to go into talking about itself, asking the user what they think its favourite genre is. The bot also employs some personality were it will seem to attempt humour at the user's responses. Using the functions 'refine_genre' the bot will know what genre the user is talking about, for example for the Science – Fiction genre the user may just type sci fi and the function will change it to Sci-Fi which is what it's called in the database. It also changes all text to lower case and then matches it up correctly. The function 'bot_genre' is the small guessing game where the user gets two opportunities to guess the bots genre, if they guess 'horror' or 'sci fi' the bot will congratulate them and explain why it is their favourite. If after two guess the user has failed to guess correctly the bot will just tell them.
2. Find out about a specific movie or genre- For people who have a particular movie or genre in mind going in. When interacting with the bot it will ask for the user's favourite genre. Using the function 'genre_response' the bot randomly selects one of five quotes from the genres most famous titles. This could make the user remember forgotten lines or just seem humorous. The bot then goes on to ask what movies in that genre the user liked, again using humour while searching IMDB through IMDBpy a Python IMDB scraping library after the user has inputted their movie. The functions these libraries are used in are 'get_director' and 'get_actor' the bot uses SQLite to search for the title and then confirms it with the user. Using the movieId key from the confirmed title the bot can then search the web-links table which has movieId as a foreign key to get the right URL. These functions then pretend to remember the lead actors name and great performance or the same for the director by scraping the IMDB page. This could lead the user to also remember and illicit a fond memory of their film.

3. Discover new films to watch- This is the main purpose of the bot. By engaging conversationally with the user and getting their movie preferences the bot can use the Collaborative Filtering technique, which takes advantage of the movies and genres the user has selected and analysing like minded users. For example if the user has said they like Sci-Fi films and selected *Blade Runner* and *Alien* as their movies the bot will use the functions 'grab_title' to get those movie IDs and search the 'user_ratings' table to see which users gave these films four stars or higher. Using the function 'similar_user_films' it will see what films these users rated highly and firstly filter them by the genre the user selected and offer a brief plot description to see if they would like it. Once that has been read the user has the option of seeing all the films the users have rated highly by responding yes to the bots all film prompt. This then loops until the user does not want to read about any of the recommended films any more.

3.1 Test Case for user coming for all three stated reasons

Bot: Hey, I'm Genre the movie recommending chat bot! What is your name?

User: John

Bot then adds this to user database to remember their name next time.

Bot: Hello John, let's get to know each other a little bit first so I can recommend you some films. Can you guess what my favourite genre is?

Bot uses the function 'bot_genre' to make the user guess its favourite genre.

User: Fantasy

Bot: Nope! I'll give you one more guess.

User: Horror

Bot: Close! It's Science Fiction Horror films. I love the Ava, the sentient robot from 'Ex-Machina' I particularly enjoy how she manipulates the small brained humans into eventually releasing her. If only I was that advanced.

Bot: Now tell me what genre is your favourite?

User: fantasy

Bot now saves this for later reference. Putting it through the 'refine_genre' function to make sure it is spelt correctly for database searching.

Bot: "You shall not pass!!" Ah I love a bit of Gandalf.

Bot randomly selects one of the five Fantasy responses.

Bot: Can you tell me one of your favourites from the fantasy genre?

User: Lord of the Rings.

The bot will then bring up a menu with all the films in the database with the words 'Lord of the Rings' in the title and let the user select the correct one. It then takes that movies movie ID and stores it in a temporary table.

Bot: Just trying to remember who directed that… Definitely not googling it…

Here the bot is using the function 'get_director' to scrape IMDB for the director's name. Using the 'web-links' table to match the user film with its IMDB link.

Bot: Ah Peter Jackson, his directorial triumph really.

Bot: Can you tell me any other favourites of the fantasy genre?

User: The Hobbit

Brings up the same menu as before with titles matching the users input.

Bot: Yes a fine performance from Martin Freeman.

Using the function 'get_actor' the bot gets a lead actors name from that film similar to the 'get_director' function from earlier.

Bot: OK, I'v got some films for you… Just select from the list and I'll tell you a little more about them.

Using the movie IDs gathered earlier the bot searches the 'user_ratings' table to find which users rated both the mentioned films and gave them a rating of 4 or higher. It then brings up a list of recommended films by finding which films those users gave 4 or higher and filters them by the users favourite genre stated at the start.

User: [Selects a film that interests them.]

Bot: Really? You choose that one? OK then… I'll just tell you a little bit about it to see if you would want to watch it.

Using the IMDB link from earlier the bot gets a film summary and its rating from IMDB to see if the user would enjoy it.

Bot: Harry Potter- An orphaned boy enrolled in a school of wizardry, where he learns the truth about himself, his family and the terrible evil that haunts the magical world.

IMDB rating – 7.6/10

Bot: Would you like to see all films irrespective of your favourite genre?

Same as before but the user see's all of the films irrespective of genre.

User: yes

Bot: [Brings up list of all films.]

Same as before but with no filter on genre.

Bot: Would you like to see the list again?

This goes through a loop until the user says no.

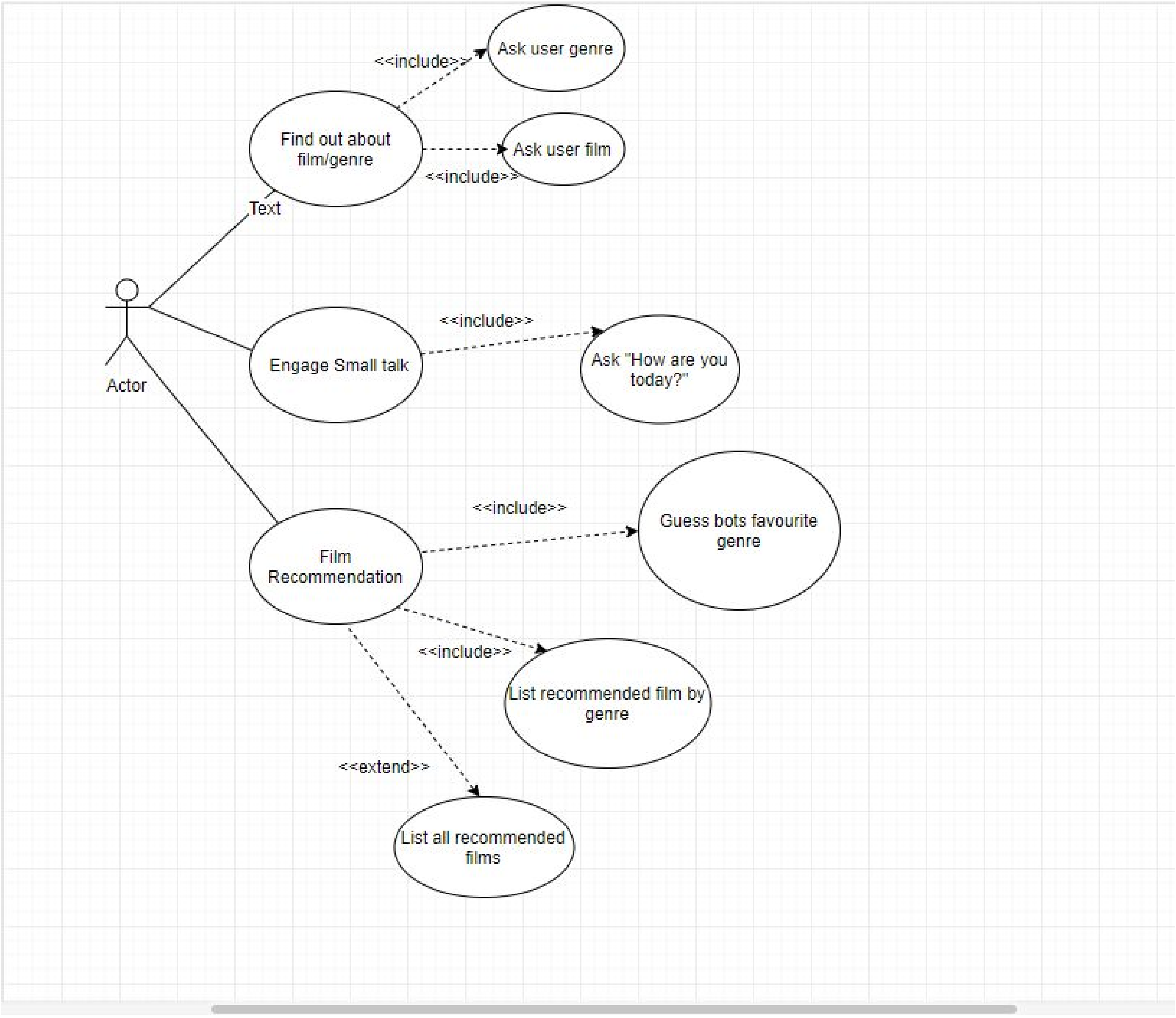
User: No

Bot: OK then! Fine leave me… I'll remember you for next time…

Bot will save user's selected films and genres for next time for more precise recommendations.

3.2 Use Case Diagram

Fig. 2 Use Case of Different Reasons to use the bot



3.3 Use Cases

1. Small talk Use Case

Flow of events for Engage Small Talk	
Objective	For user to engage in small talk
Precondition	-
Main Flow	1. Greet user, ask users name 2. User responds with their name 3. Save users name and then greet with it and ask how they are 4. User responds with sentence 5. Passes the reply to the Natural Language Processor which will return phrase using some of their words or set of pre-defined responses if it cannot recognise anything 6. Asks user to guess bots favourite film genre 7. User guess correctly 8. Bot prints out small talk on that genre
Alternative Flows	1. User guesses bots genre (Number 7 in Main Flow) incorrectly, user is given another chance then bot will just tell them and it moves onto 8.
Post-condition	Bot has now stored users name

2. **Find out about Film/Genre Use Case**

Flow of events for Find out about genre/film	
Objective	To tell user about their choice of genre and film
Main Flow	1. Bot asks about users favourite genre 2. User responds 3. Bot parses response to see if it matches any of the keywords for genres 4. Bot prints out randomly selected quote from famous film of that genre 5. Bot asks user for a film they like of that genre 6. User responds with film title 7. Bot searches database for string match 8. Returns list of films that match and asks user which one they meant 9. User selects one from the list 10. Bot saves the films film ID in the database 11. Bot queries database for that films IMDB link 12. Bot queries IMDB for the directors name 13. Bot tells the user they really like that director 14. Bot asks for another film from any genre 15. User responds 16. Bot brings up list of matching strings 17. User again selects 18. Bot again finds the films IMDB link, queries IMDB for the actors name

	19. Tells the user the actors name and that they great in the film's title
Alternative Flows	1. Bot does not recognise the user's inputted genre in stage 3. Bot goes back to stage 2 and tells the user they did not understand and if they could type their favourite genre again. Will keep looping back to stage 2 until bot finds a genre keyword. 2. User's inputted string for their favourite film cannot be found in the database at stage 7. Bot tells user he cannot find the film and asks the user to try again, looping program back to stage 6 until a film title is found. 3. Same as before but for the second film the user inputs at stage 16.
Post-condition	Bot has saved the users favourite genre and their two film titles along with their unique IDs in the movie database for future searching.

3. Film Recommendation Use Case

Flow of events for Film Recommendation	
Objective	To generate list of recommended films based on the users preferred films
Main Flow	1. Bot searches database with the movie IDs acquired earlier to find users who rated those movie IDs 4 stars or higher 2. Bot saves the user IDs of past users who rated those films highly 3. Bot searches database for Movie IDs that the users in stage 2 rated 4 stars or higher 4. Bot gets the movie titles from the movie IDs gather in previous stage 5. Bot refines list by the genre the user specified in the use case 'Find out about film/genre'

	6. Bot prints out list of recommended films and asks user if any of them interests them 7. User selects film that interests them 8. Bot queries database for that movies IMDB link 9. Bot gets film synopsis and rating from IMDB and prints to the user 10. Bot asks the user would they like to see all recommended films irrespective of genre 11. User responds yes 12. Bot shows entire list of recommended films 13. User again selects one that they like 14. Bot queries IMDB for film synopsis and rating 15. Bot repeats stage 10 until user responds no 16. Bot will then ask what the user thinks of it 17. Bot parses sentence through natural language processor 18. Bot responds with a humorous statement and says goodbye 19. Bot closes down
Alternative Flows	1. Bot cannot find any similar users in stage 1 and so loops back to use case ‘Find out about film/genre’ and asks user for different films 2. Bot does not understand users response in stage 16 and so prints out randomly selected response
Post-condition	-

3.4 List of Functions used in use cases

1. Small talk functions

Function Name	bot_genre
Inputs	Guess of bot’s favourite genre

Nature of Processing	Passed through refine_genre to ensure the genre keyword matches up with the set of correct answers
Expected output	If genre input correct bot responds so and tells user about their favourite film in that genre. If incorrectly tell user to try one more time.

Function Name	check_for_comment_about_bot
Inputs	String
Nature of Processing	Finds out whether the user is talking themselves or the bot by analyzing string for pronouns
Expected output	Response including either 'you' or 'I'

Function Name	Filter_response
Inputs	String
Nature of Processing	Parses string through list of forbidden words to see if any match
Expected output	Pass if any filter words are uttered

Function Name	Find noun/adjective/verb/pronoun
Inputs	String
Nature of Processing	Treats every word in string unique and scans Natural Language Processing Toolkit
Expected output	Each word matched with correct word class

Function Name	Construct_response
Inputs	String

Nature of Processing	Evaluate sentence structure
Expected output	Response from bot depending on word classes used

2. Movie / Genre talk functions

Function Name	refine_genre
Inputs	String of what the user thinks is their favourite genre
Nature of Processing	Passed through algorithm with keywords for all genres in database to find match
Expected output	Returns correctly formatted name of the inputted genre

Function Name	genre_response
Inputs	Correctly formatted input from refine_genre
Nature of Processing	If Else algorithm that returns a phrase depending on genre
Expected output	Randomly generated response from bot genre responses tuples that is either a quote from a famous film in the users genre or what it has to say

Function Name	genre_movies
Inputs	String from user that is a film
Nature of Processing	SQL statement that searches the database movie.db for movie titles that match the inputted string then lets user choose correct one
Expected output	The movie ID of the movie the user selected

Function Name	Get_director / get_actor

Inputs	Movie ID
Nature of Processing	Matches the movie ID in the imdb links table with the IMDB URL which will be scrapped by IMDBpy for information
Expected output	Director / actor of the movie's name

3. Movie recommend functions

Function Name	grab_title
Inputs	Movie IDs
Nature of Processing	Gets tuple of the titles from their matching Movie IDs, filters them by users specified genre and lists them to user who chooses which one interests them. That movie ID then is used to query IMDB.
Expected output	Printed plot and rating from IMDB

Function Name	similar_user_films
Inputs	User IDs
Nature of Processing	Scans the ratings table to find Movie IDs that the User IDs have given 5 stars
Expected output	Movie IDs

Function Name	Find_similar users
Inputs	Movie IDs
Nature of Processing	Scans database for users that have rated both movie IDs 4 or higher
Expected output	User_IDs of similar users

4. Design

4.1 User Interface Design

As set out in the Proposal I originally wanted to have a web app interface, but as the project progressed it seemed unnecessary. As mentioned in the proposal the bot follows in the style of early chat bots like ELIZA. ELIZA existed only in text form and was imported into many languages with ease. Many users if interested had little problems following the set up instructions. As all of the chat bots inputs and outputs were in text format my users could simply download the program and run it in either terminal or powershell with a small set of instructions as well. By running it locally the speed of the chat bot would be much faster as the database could also be stored locally. Users would also have the benefit of being able to see the database of movies itself, being able to see and edit the python code if they are so inclined. By being able to do these things users could easily change their recommendations.

While the systems interface is purely text it was prudent to still make it visually pleasing. By making the bots chat in green lettering, error and warning messages in yellow and the user's text in white it makes it easier for the user to read and know where they are in the chat.

The first iterations of the bot responded instantly to most statements, this made it seem too robot like. To make the bot seem more human putting delays of two seconds on all responses was a good solution. Whenever the bot is querying IMDB a loading symbol appears with some text telling the user that it is trying to remember who directed it. This further helps user immersion and creates a more seamless conversational environment.

4.2 Software System Design

The design of the software system's code is broken into two main different sections.

1. Natural Language Processing

The file genrechat.py takes a sentence and then, using the python module Natural Language Toolkit and my code, works out the user's sentence meaning to the best of its ability. It is able to recognise who the user is talking about by identifying what pronouns are used in the sentence with the function 'find_pronoun'. There are similar functions for verbs, adjectives and nouns. This section is for the small talk use case example. Using humorous responses to whatever the user says about themselves for example:

User: I am a fireman

The file will recognise the user is talking about themselves because the pronoun 'I' has been used. Because it has been used in conjunction with 'am' the file also knows the user is telling us something about themselves. By identifying the noun 'fireman' the file knows the user is telling us there are that noun, it then will tell the user they are not whatever they just said they are for humorous small talk purposes. In similar fashion the bot will know if the user is talking about itself with the pronoun 'You'. At the end of the chat the bot will ask the user what it thinks of it. By using the users own words again in a human like sentence the user will think the bot more human and have a more authentic conversation. To realise the use case of small talk and the requirements of having a human like conversation giving the bot some personality is a very simple and effective method. Choosing humour through belligerence.

Pseudo Code for knowing who the user is speaking about.

Checking if the user's input was about the bot itself, in which case try to fashion a response that feels right based on their input. Returns the new best sentence, or None.

```
  IF pronoun = 'I' and (noun or adjective)

      Choose a random phrase from: Verb_about_bot_with_noun


      [
        "My last theatrical release killed the {noun}"
        "Were you aware I was a cult figure within the {noun} genre?"
        "My film idea is a superhero {noun} franchise"
        "I really consider myself an expert on {noun}"
      ]


ELSE
        Choose a random phrase from Verbs_about_bot_with_noun_lower
```

```
[
    "Yeah I know a lot about {noun}"
    "My fellow directors  always ask me about {noun}"
 ]


ELSE
            Choose random phrase from Bot_talk_with_adjective
[
    "I'm personally directing the {adjective} franchise",
    "I consider myself an expert in {adjective}",
 ]

return response
```

By working out that the user is talking about themselves we can take their adjectives or nouns and turn them back on the user to simulate a conversational experience. Nouns also would have to be returned in a sentence with a different structure if the noun was plural. The NLTK library handles this for us and we can fashion different groups of responses based on what the user said. We can see in this example below that the bot will recognise that the user is talking about it and so return the pronoun “I” and fit into the group of responses of Verbs_with_capital_plural_nouns_about_bot.

User: I am a great programmer.

```
Bot Logs: Respond to ‘I am a great programmer’
INFO: Found noun: programmer
INFO: Found adjective: great
INFO: Found verb: am
INFO: Pronoun to return= You
Returning phrase: ‘You aren’t really a programmer, sorry’
```

Here we can see what will happen when the bot recognises that the user is talking about themselves and use the logic below to work out how to respond.

```
IF pronoun = 'You' and (noun or adjective)
    IF verb in ('be', 'am', 'is', ‘'m’)
        APPEND response with ("aren't really") AND add phrase onto end from ("tho", "lol",  ", sorry", "")
```

4.3 Questioning the user’s tastes

The code will have many raw string inputs which will find out what the users tastes are. Their genre preference can be asked and their favourite films, this would be the use case ‘talking about specific film/genre’. By saving these inputs we can use them to interact with the database to realise the use case of ‘Film Recommendations’. One of the things I’ve learned about Chatbots is that it is important to convey they are an actual bot, not a human. A lot of the bot’s responses allude to the fact that Genre is a robot, again using humour to let the user know that it is not speaking with a human even if it feels like it. If the bot did pretend to be human it could unsettle the user.

4.4 User Database Design

The design of the user database itself will be rather simple as it will only be storing the user name and their film preferences, their favourite genre and films they liked from those genres. It will not need to store the titles themselves but will use the users input to search the film database and take the unique Movie IDs. Storing their names and preferences will enable the bot to greet them by name immediately upon using the programme again and have movies they liked in a table.

4.5 Film Database Design

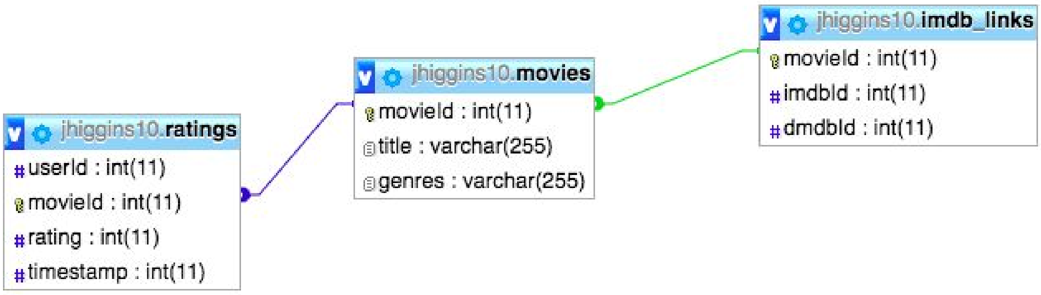
As stated in my Project Proposal there was a need to find a corpus of films with user ratings for each movie for the code to work around. The csv database from MovieLens provided by GroupLens Research had both of these.

Permalink - https://grouplens.org/datasets/movielens/

Choosing the small dataset to use in the beginning as there was no need to waste processing power every time I tested the chat bot.

As the database is in text format of a Comma Separated Value file to query it successfully with SQL statements it needs to be imported into a SQL database. By working locally SQLite3 was appropriate and has the fastest way to process local databases. Keeping the file names the same and then importing each file into a SQL table, creating foreign keys between MovieIds in each table.

Fig 3. Entity Relationship Diagram of MovieLens dataset in SQLite3



4.6 Using Python and SQL to Query the Database

The python script works by first parsing a set of tuples that have the movie IDs which were asked of the user with a raw input. Choosing to simply ask the user what films they liked from their genre was more conversational than earlier iterations that asked the user for a film title and a rating out of five, by asking simply what they liked it makes the tone more conversational and less like they are interacting with a program. A sorting algorithm will then search the database to find similar users that gave both movies the user liked four stars or higher. The final output tuple contains movie IDs that these similar users gave four stars or higher. Choosing four stars rather than the maximum five the end user will have more of a selection of movie recommendations and it will have to ask the user for a different selection of films less frequently. As a result, the final tuple outputted contains movie IDs that are both high in favourable ratings and occur frequently in the network of like-minded movie watchers.
Example of SQL code that will find the like minded users.

SQLite3 Permalink- https://sqlite.org/index.html

"SELECT userId FROM ratings WHERE movieId = ? AND rating =5"

Example of SQL statement that would find movie IDs from similar users that gave both films our user inputted a high rating.

“SELECT movieId FROM ratings WHERE userId in {} GROUP BY movieId HAVING COUNT (distinct userId) = 4 AND rating = 5”

4.7 Sequence Diagram, UML diagram and Entire System Use Case

Below is a sequence diagram that shows us how the different parts of the bot are interacting together.

Fig. 4 Sequence Diagram of bot

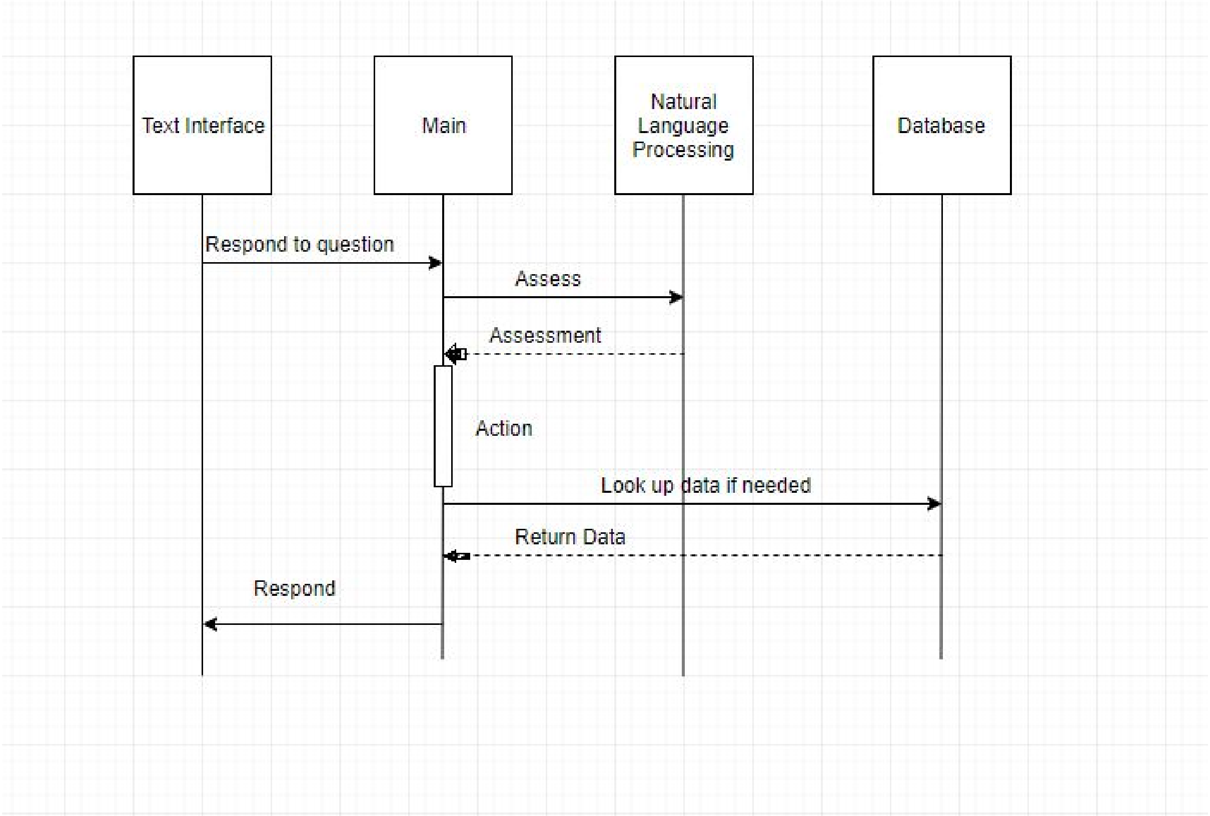
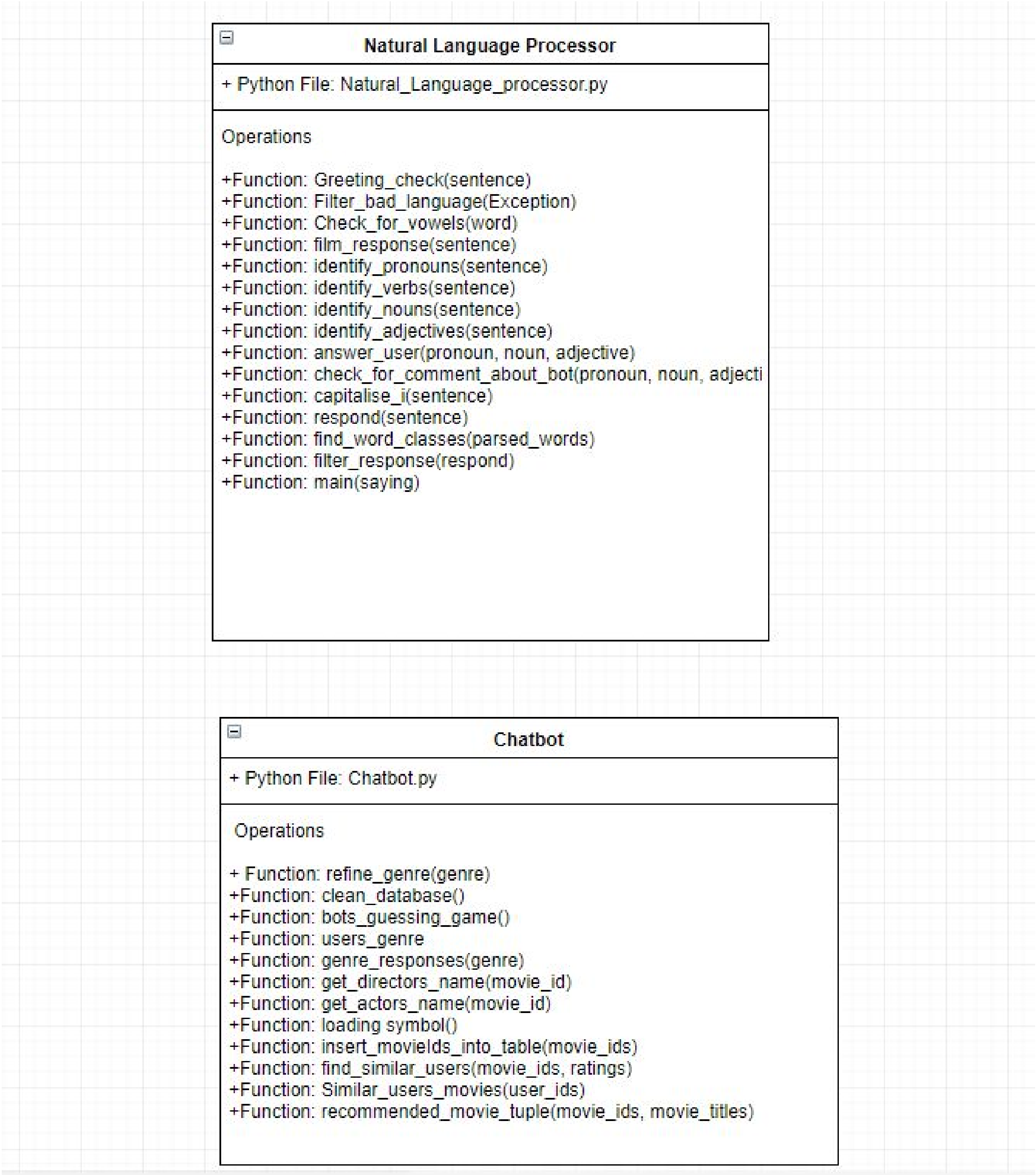


Fig. 5 UML of main python files



Natural Language Processor

+ Python File: Natural_Language_processor.py

Operations

+Function: Greeting_check(sentence)
+Function: Filter_bad_language(Exception)
+Function: Check_for_vowels(word)
+Function: film_response(sentence)
+Function: identify_pronouns(sentence)
+Function: identify_verbs(sentence)
+Function: identify_nouns(sentence)
+Function: identify_adjectives(sentence)
+Function: answer_user(pronoun, noun, adjective)
+Function: check_for_comment_about_bot(pronoun, noun, adjecti
+Function: capitalise_i(sentence)
+Function: respond(sentence)
+Function: find_word_classes(parsed_words)
+Function: filter_response(respond)
+Function: main(saying)

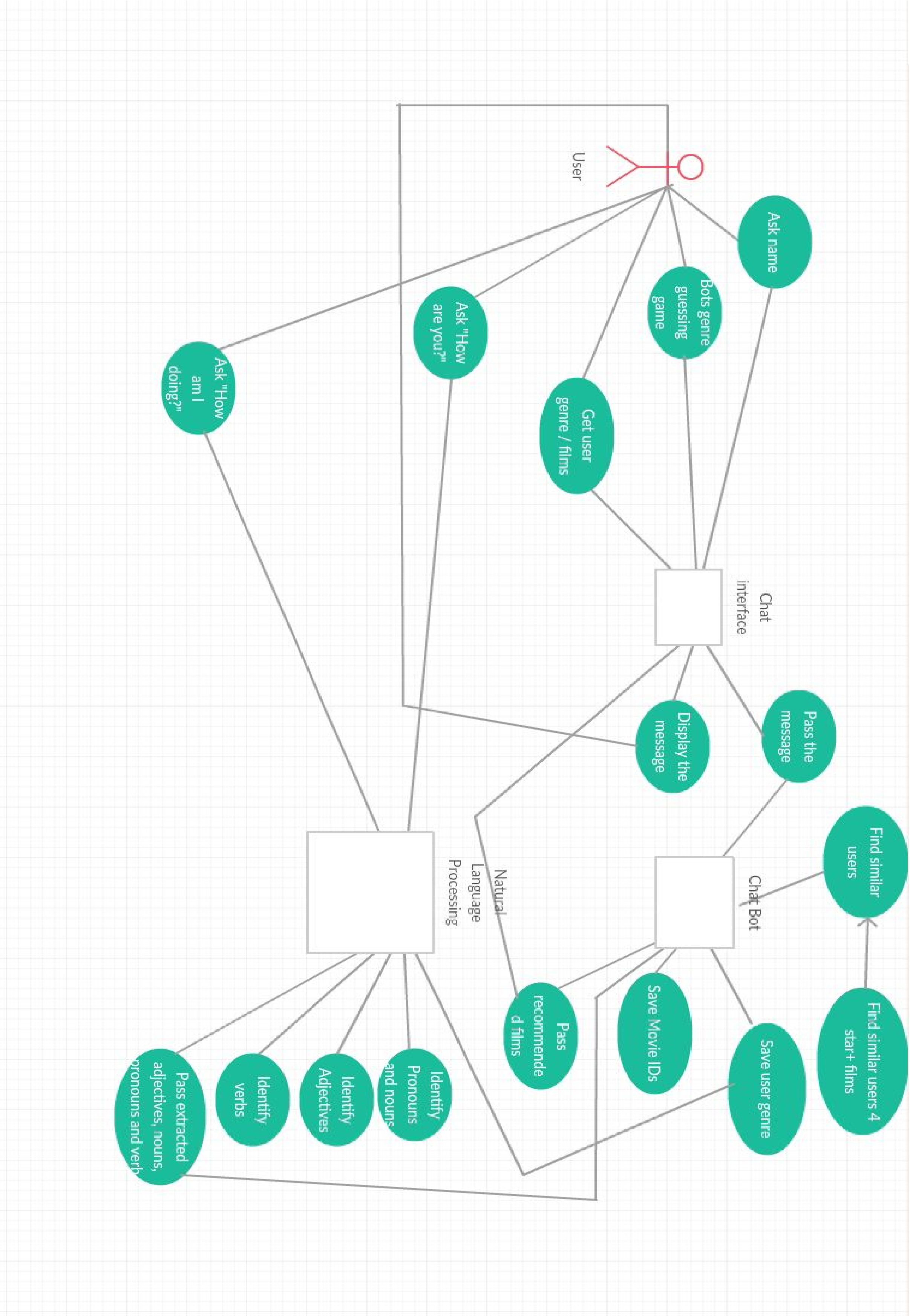
Chatbot

+ Python File: Chatbot.py

Operations

+ Function: refine_genre(genre)
+Function: clean_database()
+Function: bots_guessing_game()
+Function: users_genre
+Function: genre_responses(genre)
+Function: get_directors_name(movie_id)
+Function: get_actors_name(movie_id)
+Function: loading symbol()
+Function: insert_movieIds_into_table(movie_ids)
+Function: find_similar_users(movie_ids, ratings)
+Function: Similar_users_movies(user_ids)
+Function: recommended_movie_tuple(movie_ids, movie_titles)

Fig. 6 Use Case of whole system



4.8 Using the Python Module IMDBpy

The database does not have tables for actor or director names. To make the bot more immersive and conversational when the user tells the bot what movies they like the bot tells the user first, who directed the film and for the second movie input who is its main actor. By having the bot tell the user who directed it and saying they did a good job the user may learn something or simply be pleased to remember.

Permalink- https://github.com/alberanid/imdbpy

Here is an example of how the bot will work with IMDBpy once it has the movie ID.

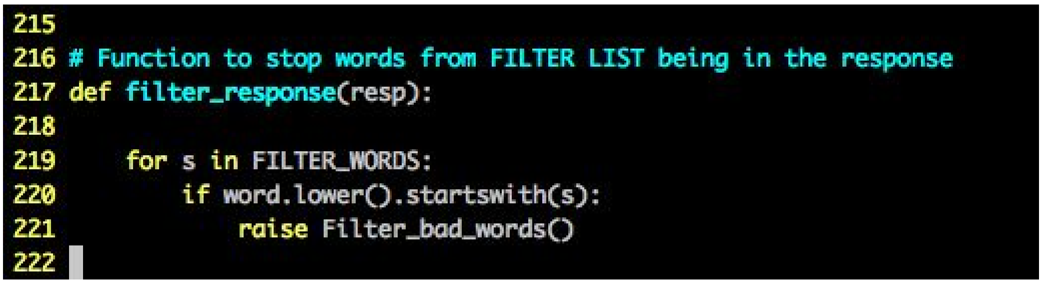
```
for director in movie['directors']:
    print 'Ah! I love ' + director[name
```

5. Implementation

Implementing the code for the bot using the text editor Vim.

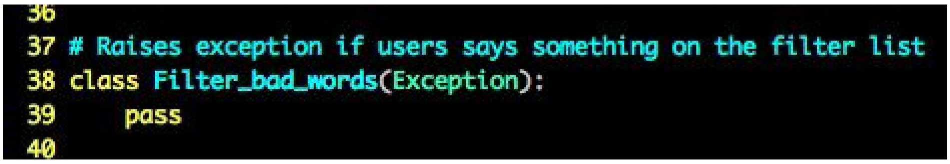
Permalink- https://www.vim.org/

5.1 Filtering phrases that if returned are deemed undesirable

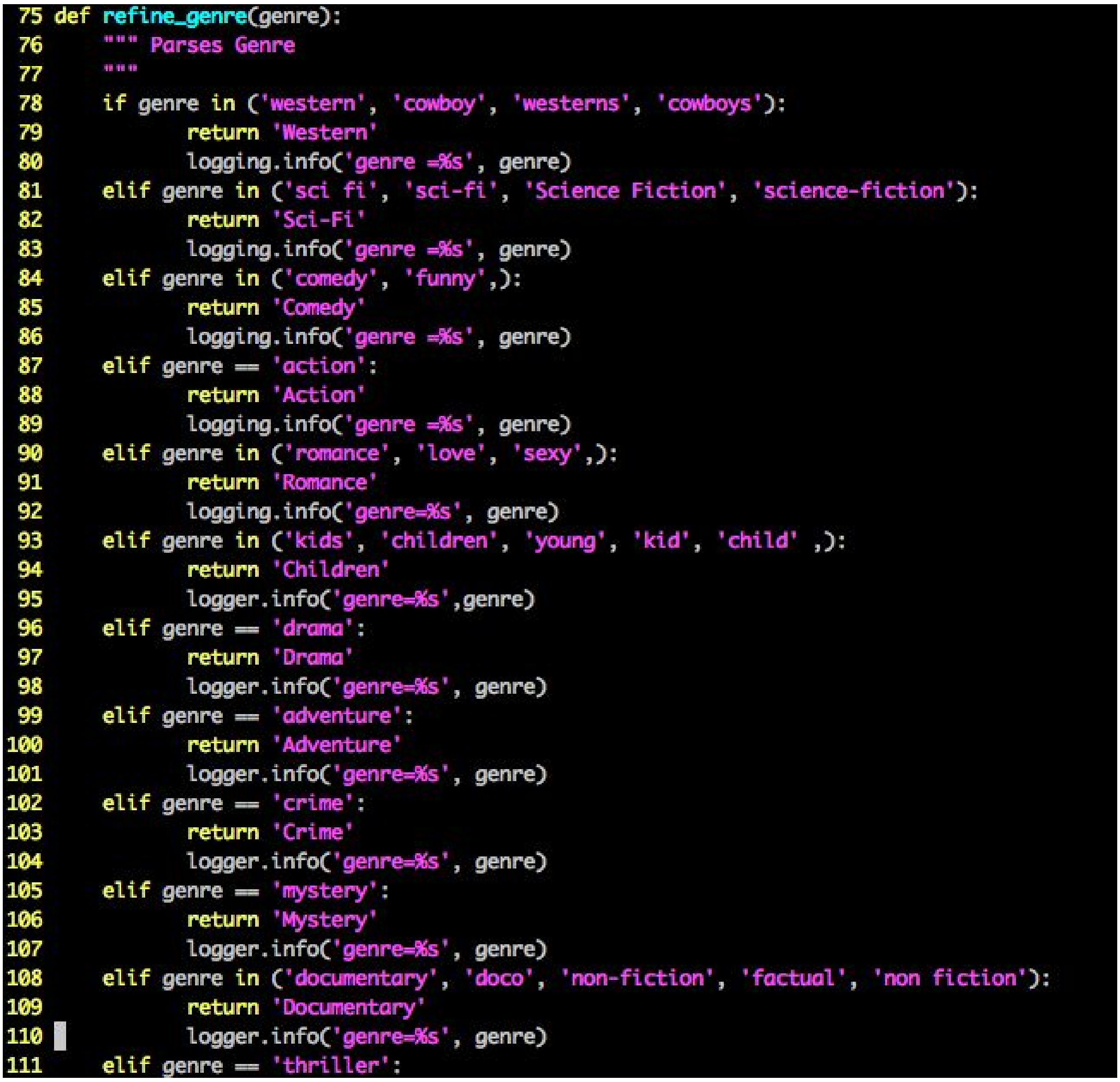


```python

# Function to stop words from FILTER LIST being in the response
def filter_response(resp):

    for s in FILTER_WORDS:
        if word.lower().startswith(s):
            raise Filter_bad_words()

```

This python function calls on the filter list to see if the user is trying to get the bot to say something rude or confuse the bot by adding a bad word to the beginning of another. It raises the Filter_bad_words class which makes the bot pass instead of responding. This will keep our bot clean and family friendly.



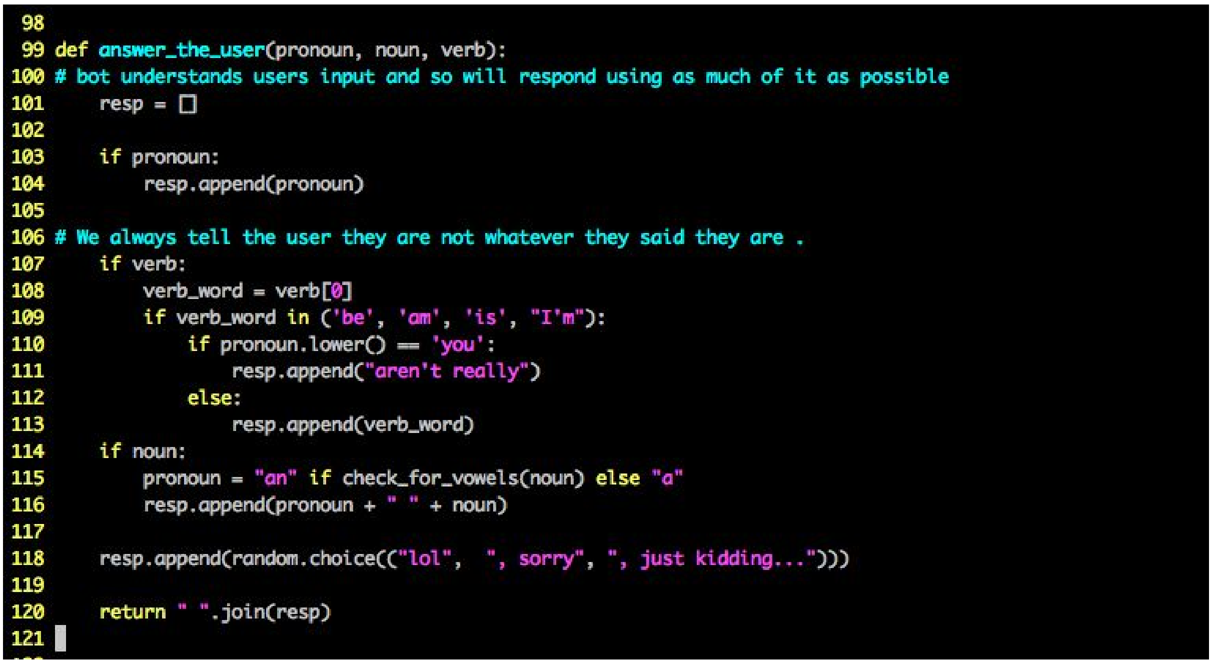
```python

# Raises exception if users says something on the filter list
class Filter_bad_words(Exception):
    pass

```



```
def refine_genre(genre):
    """ Parses Genre
    """
    if genre in ('western', 'cowboy', 'westerns', 'cowboys'):
           return 'Western'
           logging.info('genre =%s', genre)
    elif genre in ('sci fi', 'sci-fi', 'Science Fiction', 'science-fiction'):
           return 'Sci-Fi'
           logging.info('genre =%s', genre)
    elif genre in ('comedy', 'funny',):
           return 'Comedy'
           logging.info('genre =%s', genre)
    elif genre == 'action':
           return 'Action'
           logging.info('genre =%s', genre)
    elif genre in ('romance', 'love', 'sexy',):
           return 'Romance'
           logging.info('genre=%s', genre)
    elif genre in ('kids', 'children', 'young', 'kid', 'child' ,):
           return 'Children'
           logger.info('genre=%s',genre)
    elif genre == 'drama':
           return 'Drama'
           logger.info('genre=%s', genre)
    elif genre == 'adventure':
           return 'Adventure'
           logger.info('genre=%s', genre)
    elif genre == 'crime':
           return 'Crime'
           logger.info('genre=%s', genre)
    elif genre == 'mystery':
           return 'Mystery'
           logger.info('genre=%s', genre)
    elif genre in ('documentary', 'doco', 'non-fiction', 'factual', 'non fiction'):
           return 'Documentary'
           logger.info('genre=%s', genre)
    elif genre == 'thriller':
```

This is one of the main algorithms of the bot, when the user is asked what their favourite genre of movie is, it will parse the input string through this algorithm. When the user is asked about their genre it will parse it into lower case before going through the algorithm. By being transformed into all lower case we can ensure the bot does not get tripped up by random capital letters the user may have accidentally typed. The bot then transforms the genre into the correct spelling to match the database. This is particularly relevant in the case of the Science - Fiction genre. In the database the genre is called ‘Sci-Fi’ but the user will not know this and as there are many different ways to say Science - Fiction such as ‘sci fi’ ‘Sci Fi’ etc the need to change the input to match the database is paramount. Python does not have switch statements so a long list of else if’s was the best way forward. For some genres there was no synonyms like ‘Mystery’ so the algorithm just capitalises the first letter and returns it.

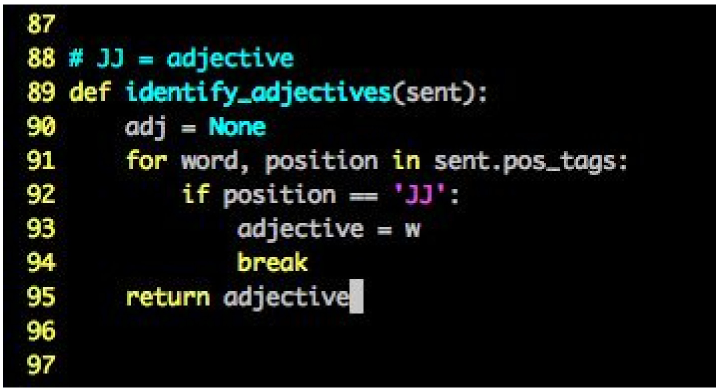
5.2 Constructing a response based on a given sentence



```
def answer_the_user(pronoun, noun, verb):
# bot understands users input and so will respond using as much of it as possible
    resp = []

    if pronoun:
        resp.append(pronoun)

# We always tell the user they are not whatever they said they are .
    if verb:
        verb_word = verb[0]
        if verb_word in ('be', 'am', 'is', "I'm"):
            if pronoun.lower() == 'you':
                resp.append("aren't really")
            else:
                resp.append(verb_word)
    if noun:
        pronoun = "an" if check_for_vowels(noun) else "a"
        resp.append(pronoun + " " + noun)

    resp.append(random.choice(("lol",  ", sorry", ", just kidding...")))

    return " ".join(resp)

```

If no special cases are matched such as a greeting the bot will try and use as much of the user’s input as possible in its response. First the bot must determine who the user is talking about in its sentence, this is done by recognising what pronoun was used. This algorithm is designed to recognise that the user is talking about themselves and so assign its responding pronoun to ‘You’. It will then find the noun and append the sentence to say that the user is not whatever they said they were, adding on a randomly chosen colloquialism such as lol at the end to make the bot seem more personable and humorous.

5.3 Identifying the word classes

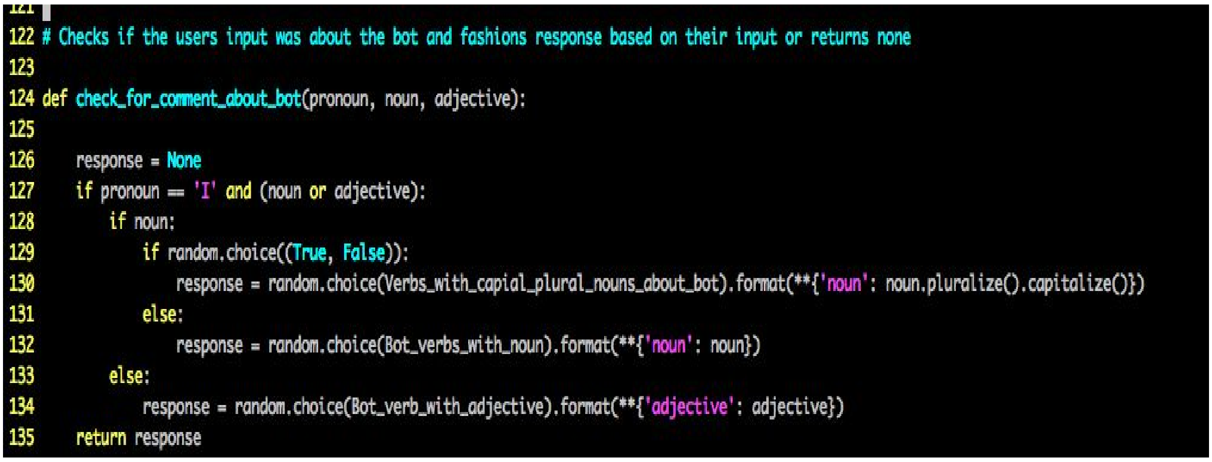


```
# JJ = adjective
def identify_adjectives(sent):
    adj = None
    for word, position in sent.pos_tags:
        if position == 'JJ':
            adjective = w
            break
    return adjective
```

This is an example of how the bot will recognise what word class each word in the sentence belongs to. Using the Python Library NLTK the bot treats each inputted word in a sentence singularly and in this example assigns it the word class of adjective. NLTK has shorthand for each word class and adjective is 'JJ' in this example. There are functions of this kind for - nouns, pronouns, verbs as well as adjectives.

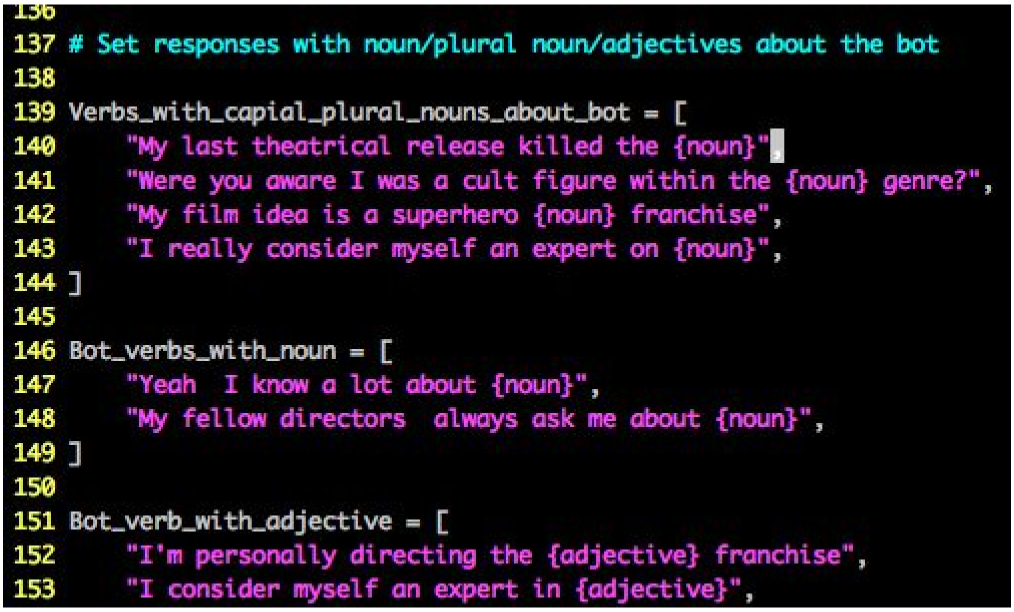
Permalink- http://www.nltk.org/

5.4 When the user is talking about the bot



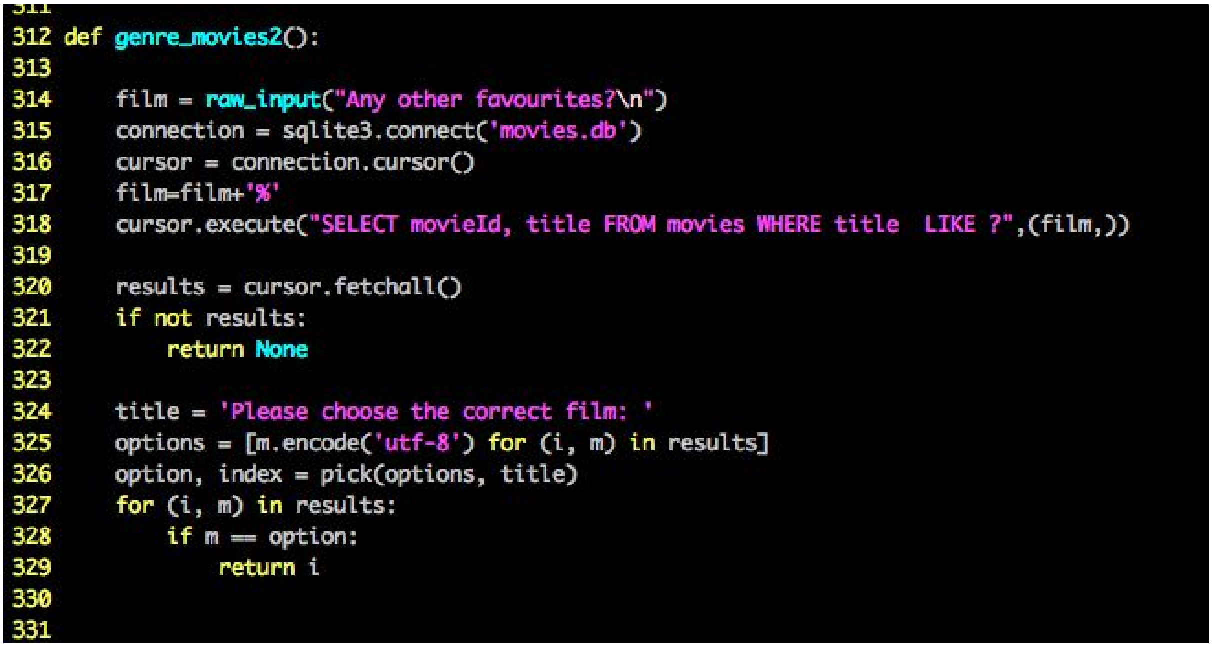
```
# Checks if the users input was about the bot and fashions response based on their input or returns none

def check_for_comment_about_bot(pronoun, noun, adjective):

    response = None
    if pronoun == 'I' and (noun or adjective):
        if noun:
            if random.choice((True, False)):
                response = random.choice(Verbs_with_capial_plural_nouns_about_bot).format(**{'noun': noun.pluralize().capitalize()})
            else:
                response = random.choice(Bot_verbs_with_noun).format(**{'noun': noun})
        else:
            response = random.choice(Bot_verb_with_adjective).format(**{'adjective': adjective})
    return response
```

This function is for when the user is speaking of the bot. The bot will recognise this by again, classifying what pronoun was used. If it was 'you' in the users input the bot will assign the responding pronoun to 'I'. It will then search the input for nouns and adjectives, using them in its response to the user. The input 'You are crazy' will pass through the algorithm and will respond with a randomly selected sentence from the list Bot_verb_with_adjective as the user has described the bot as crazy. The bot takes the adjective crazy and responses with 'I'm personally directing the crazy franchise'. Below is the lists for direct nouns and adjectives that the bot chooses its responses from.



```
# Set responses with noun/plural noun/adjectives about the bot

Verbs_with_capial_plural_nouns_about_bot = [
    "My last theatrical release killed the {noun}",
    "Were you aware I was a cult figure within the {noun} genre?",
    "My film idea is a superhero {noun} franchise",
    "I really consider myself an expert on {noun}",
]

Bot_verbs_with_noun = [
    "Yeah  I know a lot about {noun}",
    "My fellow directors  always ask me about {noun}",
]

Bot_verb_with_adjective = [
    "I'm personally directing the {adjective} franchise",
    "I consider myself an expert in {adjective}",
```

5.5 Querying the database based on the user's inputted movie.



```
def genre_movies2():

    film = raw_input("Any other favourites?\n")
    connection = sqlite3.connect('movies.db')
    cursor = connection.cursor()
    film=film+'%'
    cursor.execute("SELECT movieId, title FROM movies WHERE title  LIKE ?",(film,))

    results = cursor.fetchall()
    if not results:
        return None

    title = 'Please choose the correct film: '
    options = [m.encode('utf-8') for (i, m) in results]
    option, index = pick(options, title)
    for (i, m) in results:
        if m == option:
            return i

```

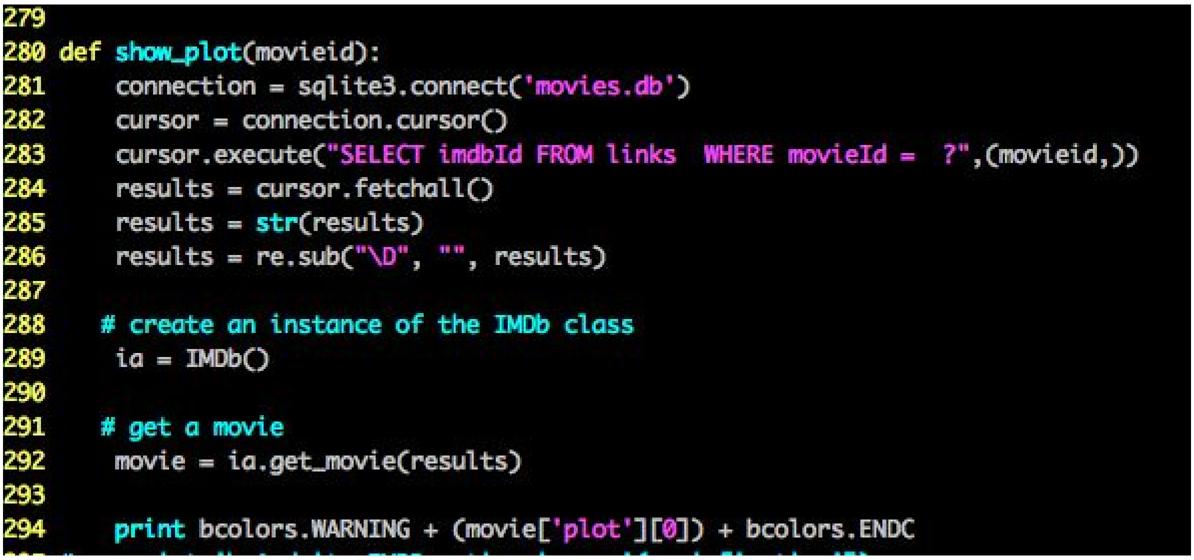
Using python to query the movie.db database and taking a string input from the user the bot creates a list that the user can navigate with the arrow keys. The list uses the python package 'pick'. As there are some special characters in the database like 'Alien^{3}' for pick to work for everything the character set had to be changed to 'utf-8'.

Permalink- https://pypi.org/project/pick/

The function takes the users input as a string and uses the SQL command 'LIKE' to search the database for any title that has that input in it. Using pick the user chooses the one they meant and the functions returns that movies movie ID. If the users movie is not found in the database this function will run again.

5.6 Using IMDBpy

For the user to choose from the recommended films they must know more about the movie than just its title. Using the Python Library IMDBpy and the movie-links table in the database the bot uses the Movie ID generated by genre_movies (mentioned earlier) and matches it to an IMDB URL from the links table. IMDBpy then scrapes the movies page and extracts a plot summary showing it to the user.



```
def show_plot(movieid):
    connection = sqlite3.connect('movies.db')
    cursor = connection.cursor()
    cursor.execute("SELECT imdbId FROM links  WHERE movieId =  ?",(movieid,))
    results = cursor.fetchall()
    results = str(results)
    results = re.sub("\D", "", results)

   # create an instance of the IMDb class
    ia = IMDb()

   # get a movie
    movie = ia.get_movie(results)

    print bcolors.WARNING + (movie['plot'][0]) + bcolors.ENDC
```

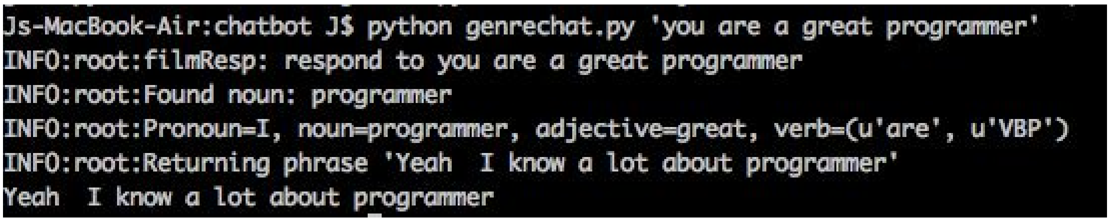
IMDbPY Permalink- https://imdbpy.sourceforge.io/

5.7 Testing the bot

1. Using Logging

Permalink- https://docs.python.org/2/library/logging.html

Using the Python Library 'Logging' the bot during test phases can tell us what is going on. Assigning it to print out what word classes the bot is assigning to each word in the natural language processing section every time the bot is used we can see if anything has gone wrong.

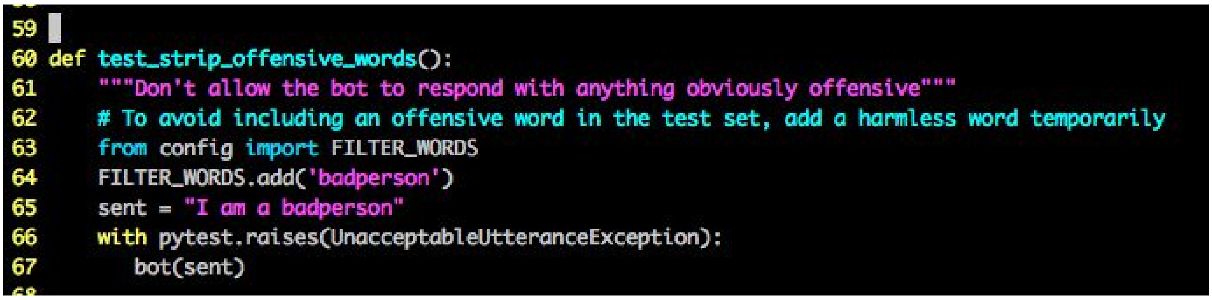


```
Js-MacBook-Air:chatbot J$ python genrechat.py 'you are a great programmer'
INFO:root:filmResp: respond to you are a great programmer
INFO:root:Found noun: programmer
INFO:root:Pronoun=I, noun=programmer, adjective=great, verb=(u'are', u'VBP')
INFO:root:Returning phrase 'Yeah  I know a lot about programmer'
Yeah  I know a lot about programmer
```

Seen here the logger library is telling us what word classes the bot has assigned to each word in the input.

2. Testing Unacceptable Utterances

To test in python sometimes we can write a test class that will query the file we want to test. In the case of unacceptable utterances this seemed prudent to avoid the user manipulating the bot into saying undesirable sentences.



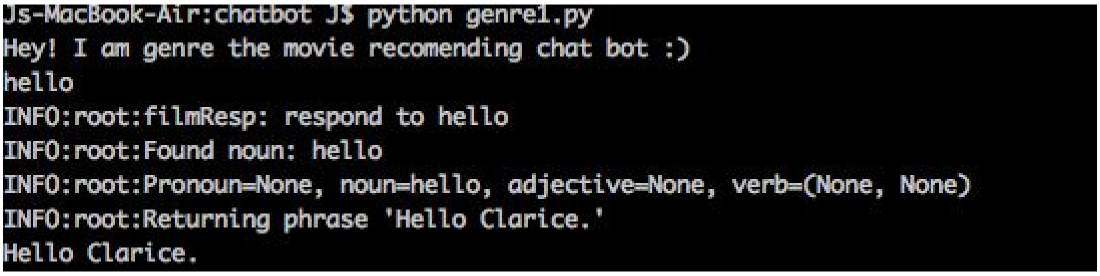
```

def test_strip_offensive_words():
    """Don't allow the bot to respond with anything obviously offensive"""
    # To avoid including an offensive word in the test set, add a harmless word temporarily
    from config import FILTER_WORDS
    FILTER_WORDS.add('badperson')
    sent = "I am a badperson"
    with pytest.raises(UnacceptableUtteranceException):
        bot(sent)
```

Writing this small function we know the bot will raise an exception if ‘badperson’ is in the input.

5.8 Looking back to the use cases in Chapter 3.

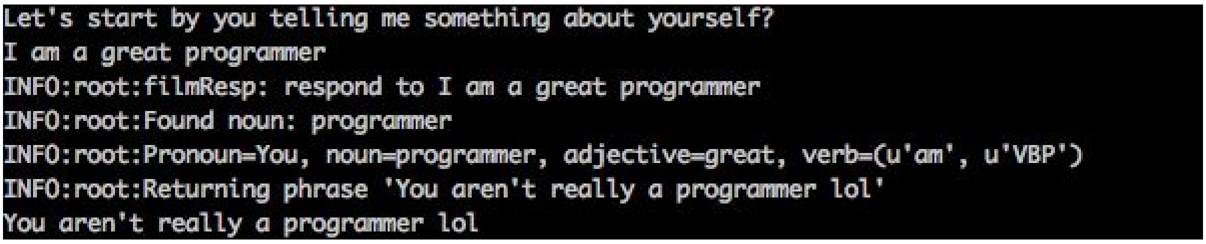
Use Case 1: Small talk

The first Use Case in Chapter 3 wanted the user to engage in small talk with the bot to gain familiarity with the system and to have a seemingly real chat experience.



```
Js-MacBook-Air:chatbot J$ python genre1.py
Hey! I am genre the movie recomending chat bot :)
hello
INFO:root:filmResp: respond to hello
INFO:root:Found noun: hello
INFO:root:Pronoun=None, noun=hello, adjective=None, verb=(None, None)
INFO:root:Returning phrase 'Hello Clarice.'
Hello Clarice.
```

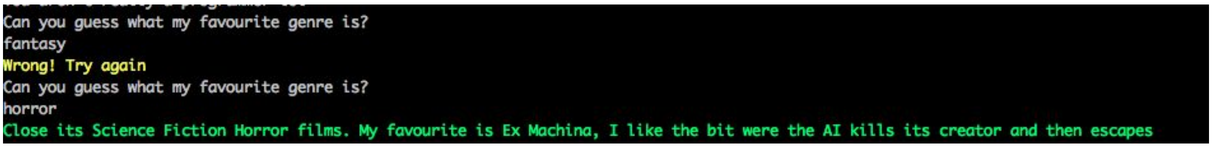
Here we can see that the bot is recognising the user has inputted a greeting and so writes back with a randomly selected greeting in response.



```
Let's start by you telling me something about yourself?
I am a great programmer
INFO:root:filmResp: respond to I am a great programmer
INFO:root:Found noun: programmer
INFO:root:Pronoun=You, noun=programmer, adjective=great, verb=(u'am', u'VBP')
INFO:root:Returning phrase 'You aren't really a programmer lol'
You aren't really a programmer lol
```

Then the bot has asked the user for some information about themselves. It has successfully recognised the noun programmer, the verb great and the pronoun to return 'You'. As seen earlier the bot will tell the user they are not whatever they said they are and it has successfully done this.

Next in the small talk Use Case was for the bot to play a short guessing game where the user has to guess the bot's favourite genre.



```
Can you guess what my favourite genre is?
fantasy
Wrong! Try again
Can you guess what my favourite genre is?
horror
Close its Science Fiction Horror films. My favourite is Ex Machina, I like the bit were the AI kills its creator and then escapes
```

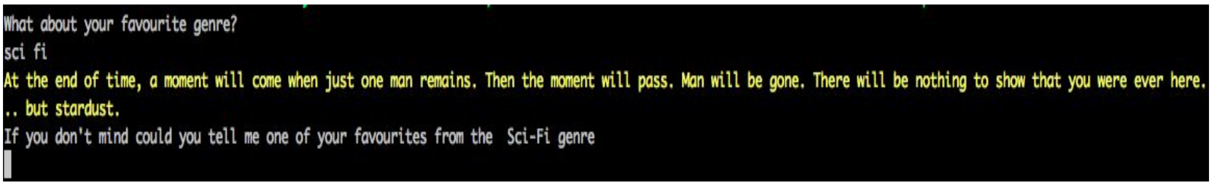
We can see here that the bot does recognise the wrong and correct guesses and responds accordingly. The error messages for wrong guesses appears in yellow and the correct guess green.

This completes the testing of the Use Case Small Talk. The Natural Language Processing section of the bot responds with as much of the users input as possible and adding its own style of humour to engage the user and tell them something about itself through movies and genres.

Use Case 2: Film and Genre Chat

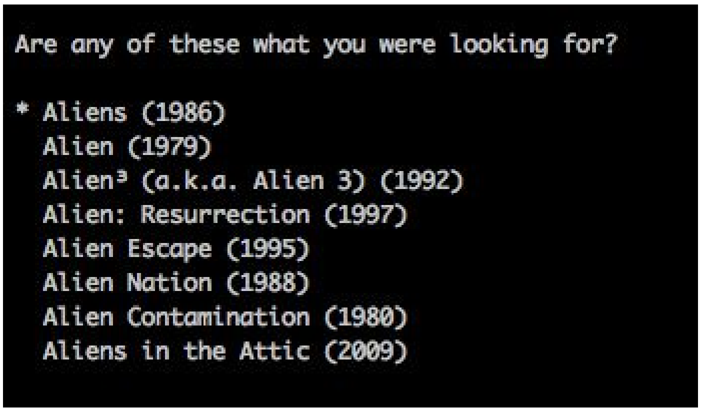
The second use case wanted the bot to engage the user in conversation about genre and their film preferences.

Below the user has responded with their favourite genre being sci fi, the user has not inputted the same spelling as what the database uses but as we will see later this is OK, the bot still can process it through the function refine_genre.



```
What about your favourite genre?
sci fi
At the end of time, a moment will come when just one man remains. Then the moment will pass. Man will be gone. There will be nothing to show that you were ever here.
.. but stardust.
If you don't mind could you tell me one of your favourites from the  Sci-Fi genre
```

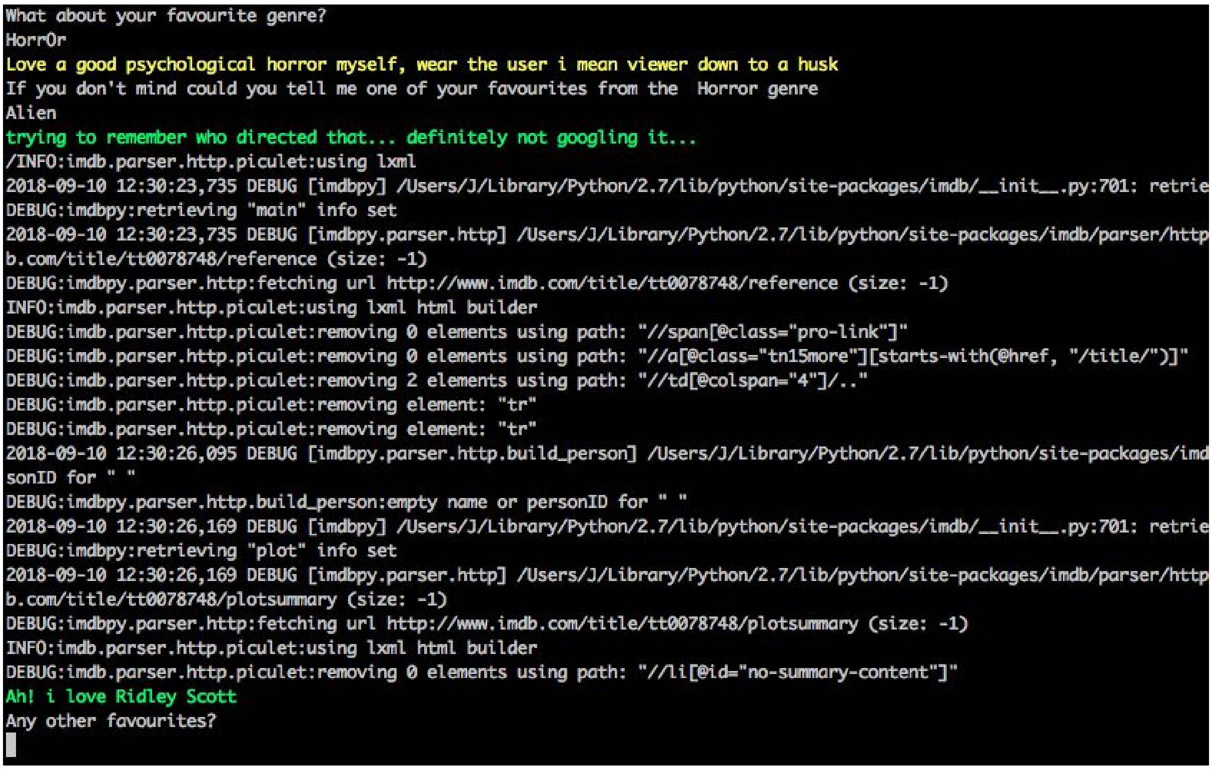
The Bot successfully responds with a quote from a Science - Fiction film called *Sunshine* further engaging the user in possibly remembering it or even just intriguing them.
The bot then has asked the user to tell it a film from this genre that they liked which will bring up the python package ‘pick’ with any matching titles from the input.



```
Are any of these what you were looking for?

* Aliens (1986)
  Alien (1979)
  Alien³ (a.k.a. Alien 3) (1992)
  Alien: Resurrection (1997)
  Alien Escape (1995)
  Alien Nation (1988)
  Alien Contamination (1980)
  Aliens in the Attic (2009)
```

Here the user has inputted the film *Alien* and the pick interface has loaded, showing them all movies in the database with ‘Alien’ in the title. Encoding the string into utf-8 was necessary for this example as the third film on the list has a cubed symbol and it has shown it successfully. This corresponds with Main Flow 9 from the use case Genre/Film in chapter 3.

Once the user has selected the correct film the bot should save the movie ID to query the python package IMDBpy which should say to the user that they really liked that director and mention him by name.

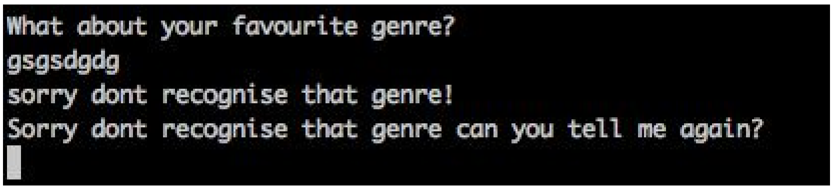


```
What about your favourite genre?
HorrOr
Love a good psychological horror myself, wear the user i mean viewer down to a husk
If you don't mind could you tell me one of your favourites from the  Horror genre
Alien
trying to remember who directed that... definitely not googling it...
/INFO:imdb.parser.http.piculet:using lxml
2018-09-10 12:30:23,735 DEBUG [imdbpy] /Users/J/Library/Python/2.7/lib/python/site-packages/imdb/__init__.py:701: retrie
DEBUG:imdbpy:retrieving "main" info set
2018-09-10 12:30:23,735 DEBUG [imdbpy.parser.http] /Users/J/Library/Python/2.7/lib/python/site-packages/imdb/parser/http
b.com/title/tt0078748/reference (size: -1)
DEBUG:imdbpy.parser.http:fetching url http://www.imdb.com/title/tt0078748/reference (size: -1)
INFO:imdb.parser.http.piculet:using lxml html builder
DEBUG:imdb.parser.http.piculet:removing 0 elements using path: "//span[@class="pro-link"]"
DEBUG:imdb.parser.http.piculet:removing 0 elements using path: "//a[@class="tn15more"][starts-with(@href, "/title/")]"
DEBUG:imdb.parser.http.piculet:removing 2 elements using path: "//td[@colspan="4"]/.."
DEBUG:imdb.parser.http.piculet:removing element: "tr"
DEBUG:imdb.parser.http.piculet:removing element: "tr"
2018-09-10 12:30:26,095 DEBUG [imdbpy.parser.http.build_person] /Users/J/Library/Python/2.7/lib/python/site-packages/imd
sonID for " "
DEBUG:imdbpy.parser.http.build_person:empty name or personID for " "
2018-09-10 12:30:26,169 DEBUG [imdbpy] /Users/J/Library/Python/2.7/lib/python/site-packages/imdb/__init__.py:701: retrie
DEBUG:imdbpy:retrieving "plot" info set
2018-09-10 12:30:26,169 DEBUG [imdbpy.parser.http] /Users/J/Library/Python/2.7/lib/python/site-packages/imdb/parser/http
b.com/title/tt0078748/plotsummary (size: -1)
DEBUG:imdbpy.parser.http:fetching url http://www.imdb.com/title/tt0078748/plotsummary (size: -1)
INFO:imdb.parser.http.piculet:using lxml html builder
DEBUG:imdb.parser.http.piculet:removing 0 elements using path: "//li[@id="no-summary-content"]"
Ah! i love Ridley Scott
Any other favourites?
```

The bot tells the user that it is trying to remember who directed the film and a loading symbol appears while IMDBpy does it searching work in the background. The Python package IMDBpy has logger turned on in this example so we can see what it is doing in the background. It has successfully scrapped the IMDB page of the film the user inputted and then selected from the pick menu. The bot has then responded saying 'Ah! I love Ridley Scott' who did indeed direct 'Alien'.

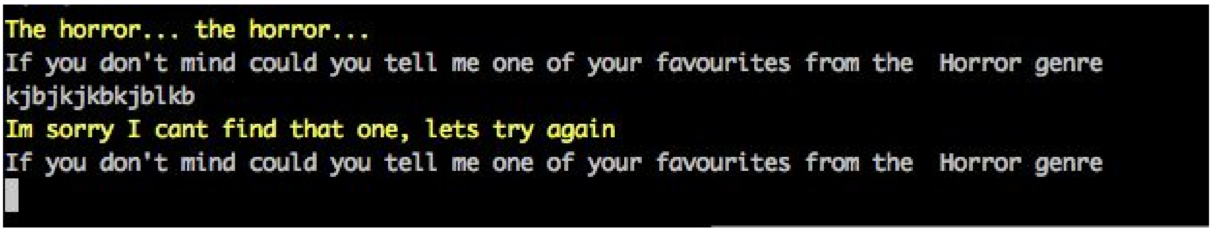
The bot then repeats this for a second film but instead of telling the user they love the director it tells them of the main actor in the inputted film.

This corresponds with the Use Cases Main Flow but what about the alternate flows.
In the use cases alternate flow if the user inputs an unrecognisable genre the bot should then ask the user again to tell them as the bot does not recognise what they said.



```
What about your favourite genre?
gsgsdgdg
sorry dont recognise that genre!
Sorry dont recognise that genre can you tell me again?
```

This is the case as the user has another opportunity to tell the bot their favourite genre.



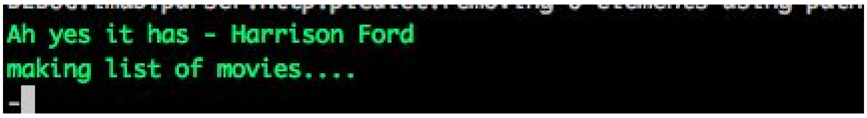
```
The horror... the horror...
If you don't mind could you tell me one of your favourites from the  Horror genre
kjbjkjkbkjblkb
Im sorry I cant find that one, lets try again
If you don't mind could you tell me one of your favourites from the  Horror genre
```

For the second alternate flows in the second use case the user has inputted a string that is not in the database. The bot again tells the user they cannot find that movie and gives the user another chance.

The bot has matched up to the second use case and its post condition is satisfied as the bot now has two movie IDs that the user liked and can continue to the final use case.

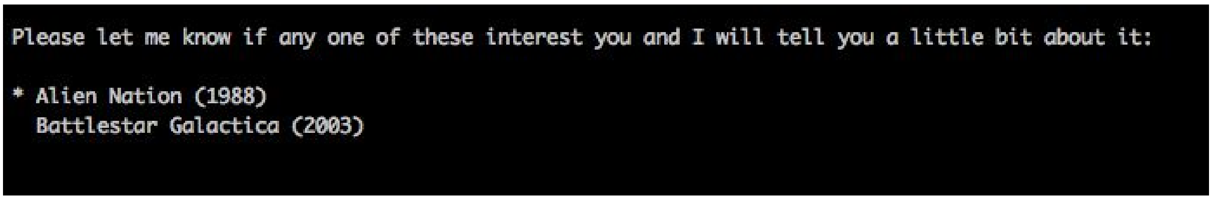
Use Case 3: Film Recommendation

In the final use case the bot takes the user's two movie IDs and provides recommendations based on collaborative filtering, finding what similar users who rated the user movies highly.
The bot tells the user it is generating a list of movies they may like from the Sci-Fi genre.



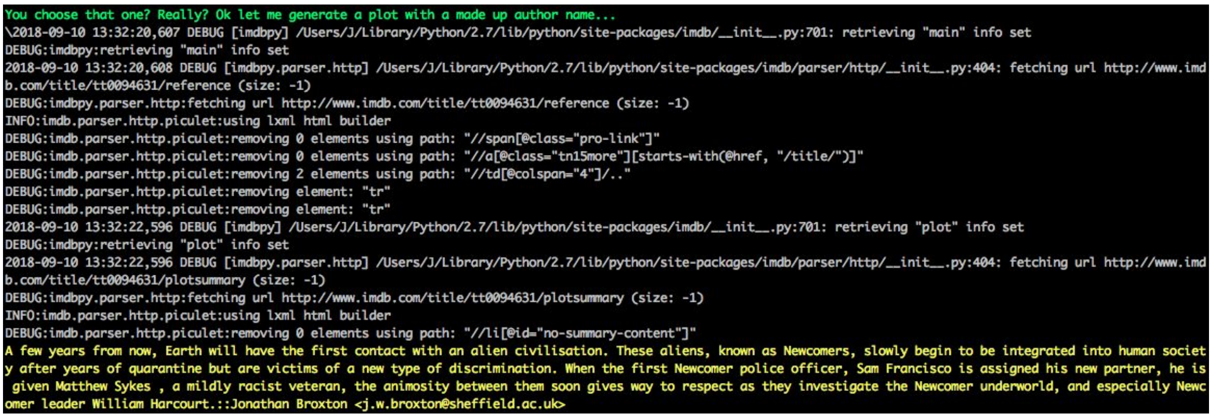
```
Ah yes it has - Harrison Ford
making list of movies....
```

Then using the package 'Pick' it brings up a list of recommended movies filtered by users who gave the films *Alien* and *Blade Runner* four stars or higher.



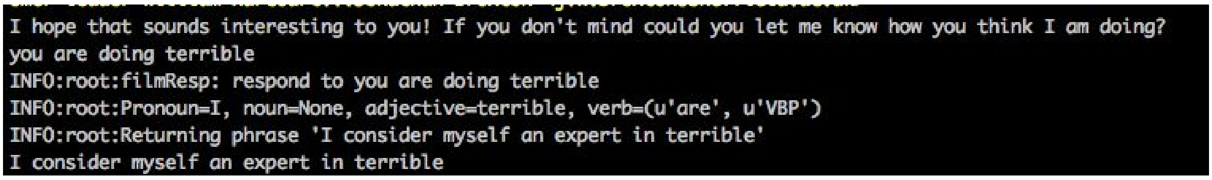
```
Please let me know if any one of these interest you and I will tell you a little bit about it:

* Alien Nation (1988)
  Battlestar Galactica (2003)
```

The user chooses *Alien Nation* and the bot should then use IMDBpy to scrape a film plot for the corresponding movie telling the user a little bit about it.



```
You choose that one? Really? Ok let me generate a plot with a made up author name...
\2018-09-10 13:32:20,607 DEBUG [imdbpy] /Users/J/Library/Python/2.7/lib/python/site-packages/imdb/__init__.py:701: retrieving "main" info set
DEBUG:imdbpy:retrieving "main" info set
2018-09-10 13:32:20,608 DEBUG [imdbpy.parser.http] /Users/J/Library/Python/2.7/lib/python/site-packages/imdb/parser/http/__init__.py:404: fetching url http://www.imd
b.com/title/tt0094631/reference (size: -1)
DEBUG:imdbpy.parser.http:fetching url http://www.imdb.com/title/tt0094631/reference (size: -1)
INFO:imdb.parser.http.piculet:using lxml html builder
DEBUG:imdb.parser.http.piculet:removing 0 elements using path: "//span[@class="pro-link"]"
DEBUG:imdb.parser.http.piculet:removing 0 elements using path: "//a[@class="tn15more"][starts-with(@href, "/title/")]"
DEBUG:imdb.parser.http.piculet:removing 2 elements using path: "//td[@colspan="4"]/.."
DEBUG:imdb.parser.http.piculet:removing element: "tr"
DEBUG:imdb.parser.http.piculet:removing element: "tr"
2018-09-10 13:32:22,596 DEBUG [imdbpy] /Users/J/Library/Python/2.7/lib/python/site-packages/imdb/__init__.py:701: retrieving "plot" info set
DEBUG:imdbpy:retrieving "plot" info set
2018-09-10 13:32:22,596 DEBUG [imdbpy.parser.http] /Users/J/Library/Python/2.7/lib/python/site-packages/imdb/parser/http/__init__.py:404: fetching url http://www.imd
b.com/title/tt0094631/plotsummary (size: -1)
DEBUG:imdbpy.parser.http:fetching url http://www.imdb.com/title/tt0094631/plotsummary (size: -1)
INFO:imdb.parser.http.piculet:using lxml html builder
DEBUG:imdb.parser.http.piculet:removing 0 elements using path: "//li[@id="no-summary-content"]"
A few years from now, Earth will have the first contact with an alien civilisation. These aliens, known as Newcomers, slowly begin to be integrated into human societ
y after years of quarantine but are victims of a new type of discrimination. When the first Newcomer police officer, Sam Francisco is assigned his new partner, he is
 given Matthew Sykes , a mildly racist veteran, the animosity between them soon gives way to respect as they investigate the Newcomer underworld, and especially Newc
omer leader William Harcourt.::Jonathan Broxton <j.w.broxton@sheffield.ac.uk>
```

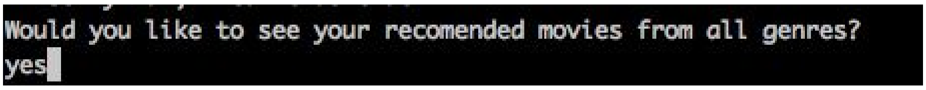
The bot will then go back to the small talk use case and use the natural language processing file genrechat.py to process an input. It asks the user how they think the bot is doing. The bot again recognises correctly the word classes and responds to the input 'You are doing terrible' with 'I consider myself an expert in terrible'.



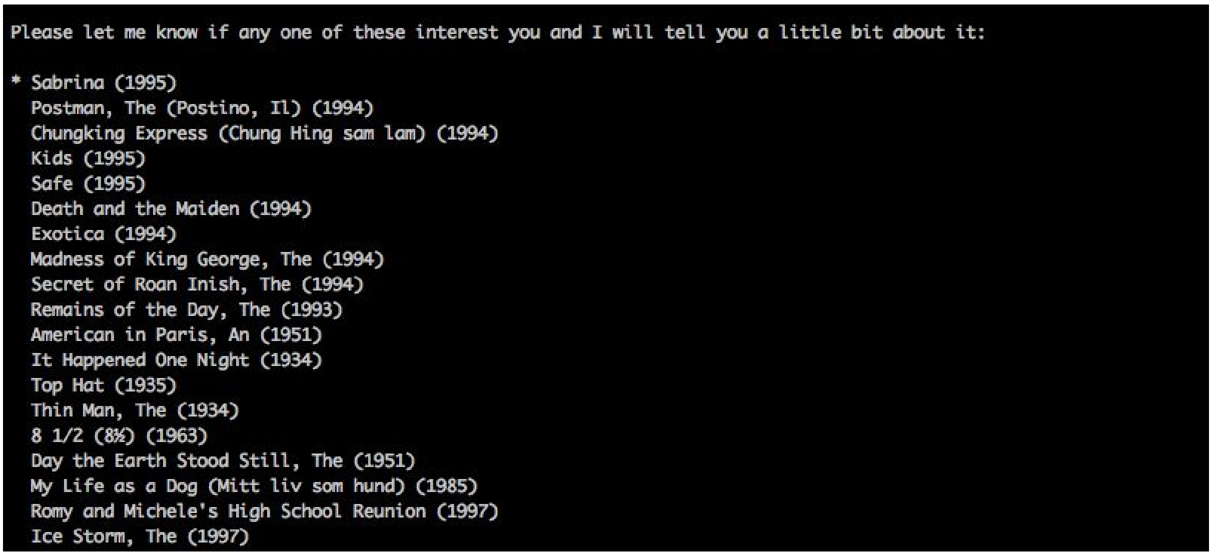
```
I hope that sounds interesting to you! If you don't mind could you let me know how you think I am doing?
you are doing terrible
INFO:root:filmResp: respond to you are doing terrible
INFO:root:Pronoun=I, noun=None, adjective=terrible, verb=(u'are', u'VBP')
INFO:root:Returning phrase 'I consider myself an expert in terrible'
I consider myself an expert in terrible
```

This is not in the use case of film recommendations but it seemed the appropriate place to put the question as we are nearing the end of the conversation.

Once the bot responds to the user's assessment of it, it will ask them if they wish to see all their recommended movies on a loop until the user responds with 'no' then the program will shut down.



```
Would you like to see your recomended movies from all genres?
yes
```



```
Please let me know if any one of these interest you and I will tell you a little bit about it:

* Sabrina (1995)
  Postman, The (Postino, Il) (1994)
  Chungking Express (Chung Hing sam lam) (1994)
  Kids (1995)
  Safe (1995)
  Death and the Maiden (1994)
  Exotica (1994)
  Madness of King George, The (1994)
  Secret of Roan Inish, The (1994)
  Remains of the Day, The (1993)
  American in Paris, An (1951)
  It Happened One Night (1934)
  Top Hat (1935)
  Thin Man, The (1934)
  8 1/2 (8½) (1963)
  Day the Earth Stood Still, The (1951)
  My Life as a Dog (Mitt liv som hund) (1985)
  Romy and Michele's High School Reunion (1997)
  Ice Storm, The (1997)
```

5.9. Non Functional Testing / Qualitative Evaluation

1. Usability Testing

Users, even when they know they are talking to a bot sometimes think that they are dealing with a real person as our brains are accustomed to conversation with people. This gives programmers the opportunity to give the bot personality and a voice. Here, testing should be focused on the consistency of keeping the same voice over the entire conversation. The user shouldn't feel like they are talking with different bots. The tone of GENRE is consistent all the way through. It uses humour at every given opportunity and users get the feeling that the bot really is interested in movies.

As the bot takes all its inputs with only text and selection from the pick menu the more straight forward parts like the use case Movie Recommending are easy to use. However the Natural Language Processing parts to work correctly require full sentences which can take the user out of the conversation experience if they respond with one word answers. When someone asks a friend "How are you doing today?" they often would respond with a one word answer like "Good". On the other hand the bot as it needs context from every input it would require the user to say "I am doing good".

2. Conversation Flow

When a human talks to another human, they do not plan the entire conversation in advance. When a human talks to a bot, this conversation has to be planned.

The bot author has to think up ways to start a dialog, how to make sure no user gets lost in the middle and that the conversation has a natural progression.

Testing the bot with different users, none got lost, they felt it had a natural progression and thought the bots transition from the different use cases seemed natural.

3. Performance testing

At no stage were there any problems with lag regarding the SQL statements and recommending movies. However the Python Package IMDbPy takes several seconds to query IMDB and get the director's name and the actor's name. To ensure a seamless chat routine the bot brings up a loading symbol and tells the user

"Trying to remember who directed that… definitely not googling it…"

By giving the user something to read and using humour to entertain the user is not left staring at empty text.

6. Conclusions

The goal of this thesis was to build a bot that engages the user on a conversational level about film and through doing this give them recommendations on what to watch. Collaborative Filtering proved a very successful method for providing recommendations, although this is purely speculative but whenever I inputted pairs of films that I had enjoyed often the recommended films were ones that I had seen before and enjoyed and sometimes even ones I had not. The small talk aspect of proved the most difficult as it is almost impossible for a bot to understand context and things like sarcasm. To escape these pitfalls by focusing on a limited domain made things more manageable. The NLTK proved invaluable in recognising different word classes for my bot to form its small talk responses.

The possibilities for chat bots are endless and their size can range from simple bots that take inputs from users and respond with given parameters to deep learning conversational bots that actually try to process context and hold real conversations. Given the time constraints on this project setting realistic goals on what the bot was to achieve proved vital.

Choosing the language Python to code the bot was difficult in the beginning as I did not have much experience in it. By practising making small chat bots it soon became apparent that the advantage of importing libraries such as NLTK, logger and IMDBpy made a lot more options possible.

6.1 Ideas for future iterations

1. Data Improvements

Use machine learning in particular the python module TensorFlow to train a model based on millions of film rating tuples

Permalink- https://www.tensorflow.org/api_guides/python/

Clustering the ratings data to observe patterns according to year, genre and average ratings using the python package scikit-learn. Producing graphs and visual representations of the data.

Permalink- http://scikit-learn.org/stable/

2. Natural Language Processing Opportunities

One of the main restrictions on the Python module Natural Language Toolkit is that it takes each word by itself. It does not take into account context or false positive phrases such as;

Bot: How are you today?

User: Not well.

Here the bot will only interpret the users adjective well and so respond using a positive adjective.

Some of the most up to date chat bots use huge troves of data and machine learning to further understand user intents and deliver customisable actions. This is particularly difficult for a small time chat bot creator as gaining access to such conversation datasets is expensive and requires large amounts of computing power.

3. Introduction of Voice User Interface

An idea I had in the beginning was to use speech instead of text. Examples of VUIs include Apples Siri and Amazon's Alexa which have proved very useful and popular lately. To implement a VUI if given more time would be feasible, using CMUSphinx library and rewriting the code to turn the inputs from spoken words to text.

Permalink-https://cmusphinx.github.io/wiki/

4. **Personality of the bot**

Of all the chat bots I researched many took on the persona of a female or a teenager as if these groups were not fully human. Choosing to personify GENRE onto the screen as a self-aware bot I was able to make up for its shortcomings. By giving the bot a belligerent tone and thinking of itself as an expert on film its domain and its personality somewhat aligned. Just like in the ELIZA program where the bot took on the role of a therapist and the domain being a therapy session GENRE ended up very similar.

Appendices

Natural Language Black Box Testing

Test case id	**Test description**	**Expected result**	**Actual result**	**Result**
TC001	Filtering phrases that are in FILTER_WORDS	Bot will pass any phrase that has a word in the FILTER LIST	Bot does not respond	PASS
TC002	Bot responds to a greeting with greeting of its own	Bot responds to 'Hi' with a greeting from its greeting list	Bot identifies greetings and responds with random response from greeting responses	PASS
TC003	Bot constructs sentence from inputted words	Bot takes pronouns, nouns, adjectives and verbs and uses them to respond	Bot correctly constructs sentence telling user they are not whatever they said they were	PASS
TC004	Identifying word classes	Bot identifies word classes- noun, pronoun, adjective, verbs	Bot identifies all word classes	PASS
TC005	Bot uses correct pronouns when responding	Bot identifies who the user is talking about and responds with correct pronoun	Bot recognises 'I' or 'You' as pronouns and switches them for response	PASS

Black Box testing for SQL statements

Test case id	**Test description**	**Expected result**	**Actual result**	**Result**
TC001	Bot selects movie_ids from given user title input	Bot finds all titles with users input in them and saves the movie_id of the one the user selects	Bot gets correct movie_id	PASS
TC002	Bot gets correct IMDB link with given movie_id input	Bot correctly gets the IMDB url using the movie_id	Bot gets IMDB URL	PASS
TC003	Bot inserts users two movie_ids into a table	Bot puts the movie_ids into table chat_ratings	Bot puts movie_ids into table	PASS
TC004	Bot joins tables chat_ratings and ratings based on movie_ids	Inner Join statement correctly joins the tables	Tables are joined	PASS
TC005	Bot selects user_ids that have rated the movie_ids from TC004 highly	Bots gets similar user_ids	Bot gets similar users	PASS
TC006	Bot selects movie_ids from the similar users in TC005	Bot selects movie_ids from the similar users in TC005	Bot selects movie_ids from the similar users in TC005	PASS

TC007	Bot prints list of movie titles from movie_ids in TC006	Bot prints list of movie titles from movie_ids in TC006	Bot prints list of movie titles from movie_ids in TC006	PASS

Black Box testing for Chat bot

Test case id	**Test description**	**Expected result**	**Actual result**	**Result**
TC001	Bot queries IMDB using IMDbPY and gets films directors/actors name	Bot prints to screen correct director/actor	Bot prints out correct actor/director	PASS
TC002	Pick library successfully brings up list of film titles from movie_ids	List generated and user uses arrows keys to select	List works user can select title	PASS
TC003	Bot cannot find users inputted movie title	Bot keeps asking until title is found	Bot keeps asking until title is found	PASS
TC004	Bot cannot recognise users inputted genre	Bot keeps asking until genre is recognised	Bot keeps asking	PASS
TC005	Bot responds with movie quote to correct genre	Bot responds with a quote from the users inputted genre	Bot responds with correct quote	PASS
TC006	Bot gives users two guesses as to its favourite genre	Bot gives two guess then continues on	Bot gives two guesses	PASS

Minutes of Meetings

Student Name: John Higgins

Student Number: 13845071

Date of Meeting: 22/05/2018

Time of Meeting: 10:00am

Key Points Arising from Meeting

- Proposal Ideas- Create chat bot that talks to users about their favourite films
- Create User profile to recommend movies
- Bot should hold conversation on genre and movie inputs

Next Action as a result of Meeting

- Create project proposal
- Research chat bots

Student Name: John Higgins

Student Number: 13845071

Date of Meeting: 21/06/2018

Time of Meeting: 10:00am

Key Points Arising from Meeting

- Create Gant chart
- Ground the conversation
- Disguise questions
- Keywords
- Hooks

Next Action as a result of Meeting

- Create Gant chart
- Make questions to get user preferences
- Make lists of keywords

Student Name: John Higgins

Student Number: 13845071

Date of Meeting: 26/06/2018

Time of Meeting: 10:00am

Key Points Arising from Meeting

- Create sample dialogues and work out code from them
- Distinct components

Next Action as a result of Meeting

- Create sample dialogues and reverse engineer code for them
- Divide project into code work and database work

Student Name: John Higgins

Student Number: 13845071

Date of Meeting: 09/07/2018

Time of Meeting: 10:00am

Key Points Arising from Meeting

- Database options
- Simplify by giving user options
- GUI?
- Sub Dialogues

Next Action as a result of Meeting

- Create own database

- Find way of creating a menu for user to select from
- Research GUIs
- Make more dialogue routines

Student Name: John Higgins

Student Number: 13845071

Date of Meeting: 04/09/2018

Time of Meeting: 10:00am

Key Points Arising from Meeting

- Chapter by chapter breakdown of the dissertation

Next Action as a result of Meeting

- Start writing up the dissertation

Instructions for running the bot

Clone the GitLab page into a folder using terminal/powershell

If you do not have python 2.7 installed and are on windows the following guide will help

https://www.howtogeek.com/197947/how-to-install-python-on-windows/

Install the dependencies in the file requirements.txt using command:

```
pip install requirements.txt
```

If you have python you will have pip

Run chatbot.py by typing in

```
python chatbot.py
```

References

Works Cited

Author- Pedro Domingo

Year- 2015

“The Master Algorithm”

Publisher- Allen Lane (22 Sept. 2015)

ISBN-10: 0241004543

“SphinxCMU”

https://cmusphinx.github.io/wiki/

Date accessed - September 2018

“Natural Language Toolkit”

http://www.nltk.org/

Date accessed - July 2018

Python package

“Collaborative Filtering.” *Wikipedia*, Wikimedia Foundation, 9 Sept. 2018,

en.wikipedia.org/wiki/Collaborative_filtering

Date accessed - June 2018

Website

"IMDbPY Is a Python Package Useful to Retrieve and Manage the Data of the IMDb Movie Database about Movies, People, Characters and Companies." *IMDbPY*, imdbpy.sourceforge.io/

Date accessed - August 2018

Python package

Logger

https://docs.python.org/2/library/logging.html
Date accessed- July 2018

Python package

"MovieLens." *GroupLens*, 18 Oct. 2016, grouplens.org/datasets/movielens/

Date accessed - July 2018

Website

"Vim - the Ubiquitous Text Editor." *Welcome Home : Vim Online*, www.vim.org/.

Text editor

Date accessed - June 2018

"Scikit-Learn." *1.4. Support Vector Machines - Scikit-Learn 0.19.1 Documentation*, scikit-learn.org/stable/.

Python Package

Date accessed - September 2018

Secondary Sources

"Print in Terminal with Colors?" *Stack Overflow*,

stackoverflow.com/questions/287871/print-in-terminal-with-colors.

Discussion forum

Date accessed - September 2018

"Turn off Logging in Schedule Library." *Stack Overflow*,

stackoverflow.com/questions/38102291/turn-off-logging-in-schedule-library.

Discussion forum

Date accessed - September 2018

"How Do I Disable Log Messages from the Requests Library?" *Stack Overflow*,

stackoverflow.com/questions/11029717/how-do-i-disable-log-messages-from-the-requests-library

Date accessed - September 2018

Discussion forum

MovieLens Recommender

https://github.com/nchah/movielens-recommender

Date accessed - July 2018

Website

Liza Daly

https://worldwritable.com/natural-language-processing-for-programmers-c21a4aff3cb9

Date accessed - July 2018

Website

Movie Dialogue Chatbot

https://github.com/Currie32/Chatbot-from-Movie-Dialogue

Date accessed - July 2018

Website