System and Unit Test Report

Product Name: Dave
Team Name: Dave

Personnel: Kenneth High – Product Owner
Eunika Wu
Cyrus Farsoudi
John Wilde
Alec Reid

Overview:

Dave is a highly dependent piece of software. The major components were tested individual. Some part were standalone pieces which were for gathering and processing data for the generator.

Component Unit Tests:

Web-scraper ("Discovery One"):

The unit test for the module is a manual test which check that the information is being fetch successfully.

1. Start the DiscoveryOne.bat in powershell
2. First argument is test.links file
3. Second argument is test
4. Run
5. The output should be a file called "test.part0.data"
6. In that file should be "This a test for Disovery One"

Parser:

should be in the same folder/directory. Also in this same folder/directory should be the folder/directory Testing Materials. This Testing Materials contains an input file with ideal script formatted and a set of six output files with the correct output that should be produced by Parse.py with this input file. The shell script mk.test will compare the output generated by running Parse.py with the given input file against the pre-written ideal output files. If no differences are returned then there are no errors, otherwise there is an error.

PDF Writer ("Clarke"):

The pdf writer is a manual test because formatting can only be check visually. The attributes which need to pass the test are the following:

- Printing to pdf
- PDF check
- Has proper headings
- Characters, action, dialogue, parentheticals and awkward page breaks
- spacing and indentation is consistent
- no text runs off the page
- page numbers

NLTK Generator ("HAL"):

This test is a manual test where the data is taken from file and output a few loosely coherent sentences.

The Director ("Stanley"):

The purpose of Stanley (also called the director algorithm) is to select what kind of text to create next in the script. Screenplay text exists in 6 categories: Scene headings, actions, dialogue, character names, parentheticals, and transitions. These follow one another in very specific ways (for example: you can have several paragraphs of action following one another, but after a character name you must have either a parenthetical or a bit of dialogue). Because this is complex, it's extremely difficult if not impossible to automate the test for this module. Therefore we here have a guide to manually test for the correct functioning of Stanley.

UI interface (Cinema-Scope):

The purpose of the GUI is give the user a pleasing and intuitive way to interact with the program. I was able to write small PyUnit tests for the Genre helper class and for some functions for daveApp.py, but the only equivalence classes I had were set to run a small integer number of genres, between 10-30 genres. However, since the GUI is only running 12 genres, which the user cannot change, this was the only equivalence class needed for testing. I was not able to write significant unit tests for most of the GUI functionality, because most of the widgets, like buttons, are built into the tkinter package. With more time I may have been able to setup mock objects for the frames and test further functionality. Therefore, most of the testing has been manual. I simply open the GUI and check whether the images and buttons are properly displaying, and whether the functions are being called when the buttons are pressed.

System Test scenarios:

- Sprint 1
  - User Story 1
    - As a user I would like a program that can return an output based on my input so that I know the output of the program is not totally random.
    - Scenario:
      - Open the program, you should see a GUI that will allow you to generate scripts
      - Click the GENERATE NOW button
      - This should generate a script
      - You can now open the program again but click the GENERATE button under one of the movie posters representing genres
      - The script generated by this iteration should be significantly different from the previous one as it should be using a different set of input scripts.
      - You can repeat this test any number of times until satisfied, just choosing two different buttons each time.
  - User Story 2
    - As a non-software engineer, I would like the program to return output in the form of a word or pdf file, because I don't like .txt
    - Scenario:
      - Open the program, you should see a GUI that will allow you to generate scripts
      - Click the GENERATE NOW button
      - This should generate a script
      - Check that the file created is an Adobe pdf
- Sprint 2
  - User Story 1
    - As a scriptwriter, I would like a program that can return stories in conventional screenplay format because screenplay format is crucial to readability.
    - Scenario:
      - Open the program, you should see a GUI that will allow you to generate scripts
      - Click the GENERATE NOW button
      - This should generate a script
      - You can now check for screenplay format by eye. If you need a systemic/algorithmic way to tell if this is working, this means that the correct output will:
        - start with a scene heading. All scene headings will be followed by actions.
        - Actions can be followed either by a transition, a character name, a scene heading or another action.
        - A transition can be followed by a character name, a scene heading or an action.
        - A character name must be followed by either dialogue or a parenthetical.
        - A Parenthetical must be followed by dialogue.
        - Dialogue can be followed by either a character name or by action.
        - Any deviation from these is an error.
  - User Story 2
    - As a non-software engineer, I would like the program to have an intuitive and inviting GUI because I do not like the command line.
    - Scenario:
      - Open the program, you should see a GUI that will allow you to generate scripts
      - Judge if this GUI is aesthetically pleasing enough for you

- Sprint 3
    - User Story 1
        - As a writer I would like to read a correctly formatted series of grammatically valid sentences because word jumbles are hard to read.
        - Scenario:
            - Open the program, you should see a GUI that will allow you to generate scripts
            - Click the GENERATE NOW button
            - This should generate a script
            - You can now check the screenplay for coherancy by eye.
    - User Story 2
        - As a person, I would like a program that can generate humorously insane/random stories on a whim because I enjoy laughing.
        - Scenario:
            - Open the program, you should see a GUI that will allow you to generate scripts
            - Click the GENERATE NOW button
            - This should generate a script
            - You can now check the screenplay for nonsensicalness by eye.

Acceptance tests:

Discovery One (Webscraper):

- Acceptance criteria
    - Should gather scripts from imsdb.com and store them in plain text files with the .data extensions, with 15 scripts per file
    - Run unit test for the webscraper to verify this works. Look through the various .data files by hand

GUI:

- Acceptance criteria
    - Should be a GUI with buttons and pictures. Pressing the generate buttons should generate a script based on the inputs from the webscraper's .data files.
    - Run unit tests for the GUI, if it passes then it works

HAL (natural language generator):

- Acceptance criteria
    - Should generate natural language strings based on n-grams from the input data selected by the button pressed on the GUI.
    - The results generated should be in English
    - The results should consist of strung together sentence fragments with no evaluation for f any logical sense is made
    - To run the test for HAL, just run the module independently and look at the output

PDF generator:

- Acceptance criteria
    - Final output should be in a .pdf file
    - Run manual test (formattest) to verify

Parse:

- Acceptance criteria
    - Correctly separates headings, dialogue, actions, etc. with approx. 90% accuracy
    - Run unit test for Parse.py (described in README for Parse.txt) to verify

Stanley (the director algorithm):

- Acceptance criteria
    - Outputs text paired with formats in a "correct" sequence for script formatting
    - Run module with either fixed input or input from other modules and check the results by eye.