Oomck Chatbot Project Plan

19th February, 2021

For COSC 310

By Owen Murovec, Opey Adeyemi, Mathew de Vin,
Carson Ricca, and Keyvan Khademi

Project Description

In this project, we are creating a chatbot that will allow a user to interact with the agent and maintain a conversation about the Fast and the Furious franchise. The agent will be able to maintain a conversation with the user for approximately 30 turns. The agent will play the role of a friend who has a vast knowledge of the Fast and the Furious. The user will have the role of another friend who also enjoys the series. The repository for our project can be found at https://github.com/oomck/oomck-bot.

SDLC & Rationale

An incremental development lifecycle is proposed for this project. More specifically, agile practices will be employed. An agile approach is preferred by the small and well-knit team of developers, as well as the scope of the work involving rapidly shifting requirements as the team becomes more acquainted with the tasks at hand. The chatbot is a small application with a specific purpose that is suited for agile development. Despite an agile approach being the primary choice, components of integration and configuration will be used. This is due to the prevalence of pre-existing, reliable, and robust systems that accomplish parts of what Oomck Chatbot aims to achieve.

SDLC Phases

SCRUM is an iterative development method that focuses on managing iterative development practices. Scrum has three main phases that we will implement in our project.

1. The initial phase is an outline planning phase in which the general objectives are established for the project, and the software architecture is designed.
    a. Creating a project plan:
        i. Brief project description.
        ii. Select SDLC, with rationale.
        iii. List of SDLC phases.
        iv. Initialize a WBS.
        v. Create a Gantt Chart.
    b. Determine dataset:
        i. Discover new datasets and discuss feasibility of usage.
        ii. Explore the possibility of using an NLP dataset, pattern matching, and API implementation.
        iii. Determine dataset(s) to use.
    c. DevOps setup:
        i. Set-Up automated unit testing.
        ii. Containerize ElasticSearch using Docker.
        iii. Transition Jira TODOs on successful build runs.
        iv. Create DevOps setup script(s).
2. The second phase is a series of sprint cycles. Details of each sprint can be found in the ***Sprint Log*** section of this document. Each cycle implements an increment of the system.
    a. Create command-line interface:
        i. Read input from command-line.
        ii. Print output back to the command-line.

    b. User Input Parsing:
        i. Retrieve user input.
        ii. Clean user input (i.e. standardize input into an easy to understand format).
        iii. Tokenize user input (i.e. remove words in the input that have little effect on the meaning of the sentence).
    c. Initialize ElasticSearch with Data (Script):
        i. Clone the data repo.
        ii. Filter and clean data into the correct form.
        iii. Tokenize data.
        iv. Indexing data into Docker.
        v. Upload data to the ElasticSearch container.
    d. Creating bot logic.
        i. Write ElasticSearch api query script.
        ii. Feed clean and tokenized user input into api query.
        iii. Simple handling for unknown input strings.
3. The final phase wraps up the project, required documentation is completed, and lessons learned from the project are assessed.
    a. Finish project plan document:
        i. Create a list of limitations.
        ii. Include sample output.
        iii. Document system architecture.
    b. Create project presentation:
        i. Showcase project's strengths and weaknesses.
        ii. Brief overview of WBS.
        iii. Show PM graphs on Github.
        iv. Showcase the Jira project.

Project Objectives

To develop a conversational agent capable of retaining dialogue with a user for 30 turns back and forth (a turn is defined as a prompt-response pair). This agent is to have a topic upon which it centers its responses as well as a distinct tone through which it responds.

Deadline: ~~March 5th, 2020~~ March 12th, 2020

Requirements

1. Capability to keep a 30 turn dialogue.
    a. A turn is defined as a prompt-response pair.
2. Focus conversation around the topic of the Fast and the Furious movie franchise.
3. Produce a distinctly conversational tone.

Project Scope

Technology used:

- Python.
- Docker.
- GitHub.
- ElasticSearch.
- Jira.
- Discord.

Techniques Used:

- NLP
- Containerization
- Data cleaning and wrangling

Sprint Log

Following the Scrum structure, our work was largely broken into 3 sprints. Initially, we used our extensive SDLC phases breakdown and used it as a basis for our WBS. In turn, this meant that we created issues for each of the tasks.

Sprint 1: Initial Bot Setup
Goals: Load dataset into ElasticSearch, and have a working user input parser.
Added to backlog: Problems with docker container setups and versioning errors. Many datasets were tried during this sprint, all leading to issues of compatibility with our system approach. Eventually a large dataset of Ubuntu support chat logs was used.

Sprint 2: Initial Bot Functionality
Goals: Resolve problem of large datasets and long start up times, connect the bot to ElasticSearch API to properly query the data, establish robust system architecture, and revise user input parsing techniques.
Added to backlog: CI scripts interacted with Jira intuitively, docker container initialization causes issues on certain machines, upon closer inspection the ubuntu-related dataset we used contained lots of jargon, leading to nonsensical conversations with the bot.

Sprint 3: Revise Bot Functionality
Goals: Fix CI scripts, reinitialize docker container creation, find and load new dataset, redo any necessary sections of the codebase to accommodate for these changes. The system architecture needed to be tweaked due to this.
Added to backlog: None.

Closing Sprint: Wrap Up
Goals: Finish documentation, create project presentation video, bug fixes and tweaks.

Work Breakdown Structure

[Included in an attached file]

Overall Schedule

The Agile process our project team is following inherently conflicts with the notion of a pre-defined schedule. We did, however, procedurally generate GANTT charts as issues were created and completed. Our underlying project structure was that of Scrum. In order to achieve this, we met every Thursday at noon in order to refocus on our goals as a team and keep things on track. The complexity of student schedules meant our sprints did not start and end precisely on Thursdays, but we did discuss progress and assign tasks during these meetings.

Limitations

1. As a result of the method we chose, the competence of Oomck-Bot's responses are highly dependent on the indexed data it uses. Even with strong data, depending on the context (i.e. if API-driven data was used), Oomck-Bot would quite often give nonsensical responses to user prompts. This leads to conversations that don't seem connected, despite being algorithmically sound. The current dataset remedies this problem, with the tradeoff that it is quite small, leading to a generally limited range of dialogue.

2. The current implementation of Oomck-Bot lacks a GUI. We currently use a basic command-line interface to interact with the user, which is terribly unintuitive. Going forward, Oomack-Bot will feature a web-based GUI to allow for easy chat history viewing and streamlined back-and-forth communication with users.

3. Oomck-Bot does not dynamically retrieve any relevant data to its responses, thus if the dataset it uses has many logs that say "Sunny" as a response to "How is the weather", then Oomck-Bot will say the same, without checking the current status of the weather.

4. Oomck-Bot currently only responds with the same line of dialogue when puzzled about a user's input. Expanding this would enhance the realism of Oomck-Bot's conversations.

5. Our bot is also only able to interpret a single line of input from users and output a single line of dialogue in response. Our command-line interface limits users such that they must wait for Oomck-Bot to respond before inputting a new prompt. However, in real conversations, users may want to add additional information to their prior message and may expect a similarly formatted response.

Sample Output