Imouto NLP Documentation

INFR 4320U – Artificial Intelligence for Gaming

Kevin Pang

Kenny Difei Wang

Contents

[Summary 3](#_Toc436060646)

[Program Components 4](#_Toc436060647)

[Natural Language Processing 4](#_Toc436060648)

[TextProcessing.cs 4](#_Toc436060649)

[ThreadedTextProcessing 4](#_Toc436060650)

[Response Generation 5](#_Toc436060651)

[Finite State Machine 7](#_Toc436060652)

[ImoutoFSM 8](#_Toc436060653)

[Imouto FSM Diagram 10](#_Toc436060654)

# Summary

The Imouto NLP program is an experimental framework that integrates a soft Natural Language Processing component with a Finite State Machine in order to create an environment where the user can experience having an authentic conversation.

The program does not use a standard parser or dictionary; instead it uses a custom built parsing method and dictionary to process text. As a result, there are some limitations to the processing ability and understanding of the program.

Imouto means “little sister” in Japanese, and as the name suggests, the game places the player as an elder sibling to a little-sister type character. The whole game centers on communicating with the AI imouto by entering text into the chat box in an attempt to sway her into a better mood.

# Program Components

The Imouto NLP program features the following main components.

## Natural Language Processing

The following section will go over how Natural Language Processing is done in the program.

### TextProcessing.cs

TextProcessing.cs contains the methods that make up the core of the language processing component of the program. It processes text using text processing modules derived from the ThreadedTextProcessing class. As the name of the class suggests, the text processing modules run on their own threads.

Once the modules are finished scanning the entered text, the program moves on to calculate a “sway value”. This value is used to determine state changes within the ImoutoFSM system. The formula for calculating the sway:



Essentially, the sway value is the sum of the weighted numbers of modifier words, keywords and intent words. The sway value is only calculated when a keyword is found in the entered text, meaning if a keyword is not found, it will default to zero. Values greater than 0 represent a positive reaction from the imouto while values lower than 0 (I.E. In the negatives) represent a negative reaction. Furthermore, if a negative keyword is detected the sway value will be inversed, meaning that the slightest mention of a negative word will result in a negative reaction from the imouto.

### ThreadedTextProcessing

The ThreadedTextProcessing class (also found in TextProcessor.cs) is the processing module that is used to process the text entered by the user. It does so by using the string.Contains() method and list of words to search for.



Each module contains a list of words that it will search for. For example, the positive keyword module would contain a list of words considered as positive keywords, while a swear word module would contain a list of expletives. If a matching word is found, it is logged onto a separate list called MatchedWords. The MatchedWords list is cleared at the start of each processing loop in order to eliminate the chance of overlapping results.

All ThreadedTextProcessing modules are managed by a ThreadedTextProcessingController object. The controller is also able to start all tasks on the main thread in the event multithreaded processing produces errors or unwanted results.

A multithreading approach was chosen in favor for its speed; while running tasks one by one on the main thread is more effective for small inputs and a small reference list, a multithreaded approach provides better insurance in the event the reference lists grow too large.

## Response Generation

Responses in Imouto NLP are not Generated in the sense that the program constructs a sentence from a list of words, following a grammar structure. Instead, responses are loaded from a text file and are picked depending on the sway value calculated from the user’s input, as seen in the code below.



The function, GenerateResponse(), accepts the calculated sway value from the user’s entered text as well as the number of keywords that were found and picks an appropriate response. Additional suffixes and extras are appended to the response by the code below, which is also part of the GenerateResponse() function. Currently, the program only appends the player’s name and some grammar particles.



## Finite State Machine

The ImoutoStates script contains the necessary base class of the finite state machine used by the imouto. The actual code of the FSM is a modified version of the deterministic FSM that is publicly available on the unity wiki at the following link: <http://wiki.unity3d.com/index.php?title=Finite_State_Machine>.

The main loop of the FSM calls the Act() and Reason() function of the current state. The Act() function is the specific behaviour that the state exhibits and the Reason() function checks if the conditions for a transition to another state have been met, and executes the relevant processes if so. The system itself contains a list of states, with the transitions kept inside the state itself (I.E. State A could have transitions to State B and State C, and all the States are kept in the List that is managed by the FSM system).

Transitions are performed by calling the PerformTransition() Function, shown below.



The function first checks whether the user is attempting to access a null transition or transition to a null state, in which case the function will abort.

If the transition is deemed valid, the system will execute the currentstate’s DoBeforeLeaving() function, which generally contains clean up functions to reset the state should the AI return to it at a later time. It then switches the current state to the new one before executing the new state’s DoBeforeEntering() function, which contains initialization functions.

### ImoutoFSM

The ImoutoFSM class is the implementation of the FSMSystem defined previously. It also introduces the EmotionState class, which inherits properties from the FSMState Class. The purpose of the EmotionState class is to serve as a multiuse State from which multiple emotions of the imouto can be derived from.

The EmotionState uses a high and low value in order to determine when to transition to another Emotion. The Imouto’s sway value is updated in real time and used in comparison with the aforementioned values in order to determine whether to transition to a more positive emotion or a more negative emotion. The sway value is reset to 0 every time a transition occurs.

The transitions to each Emotion is hard coded into the EmotionState’s Reason function, as seen in the next page. The reason all the transitions are predetermined is to reduce the amount of redundant code (i.E having to define a new class for a new emotion) as there are very few differences between two Emotions.



## Imouto FSM Diagram

The following flowchart illustrates the finite state machine used by the ImoutoObject. The sway values are generally balanced between all states, with the exception being the transitions between the angry and annoyed states. It is slightly easier for the player to lose than to win, mimicking the trait that humans respond greater to negative remarks as opposed to positive ones.

