Advanced Lab Semester Project Report

Omar Nada Tarek Samy Mohannad Banayosy

February 2014

1 Introduction

Multiple options were given to choose the semester project from. we decided to add voice recognition to the game developed in Semester 4 (Entanglement). We also implemented a general Grammar generator and used it to generate the game grammar file. In this section we explain briefly what is Entanglement.

Entanglement is a game originally developed by gropherwood studios. We were asked to implement a simpler version of it in Semester 4 as a part of the CSEN401 course.

The game supports one to 4 players, each player starts on a tile and the objective is to connect tiles on the board using ropes. The player with the longest connection wins. A player loses if he hits the board boundaries or returns to the starting tile. Players can connect ropes by fixing the tiles and rotating them.

2 Extending Entanglement

We used CMU Sphinx to extend the old version of Entanglement. The user is allowed to start the game using 1-4 players and play it using voice commands. The project was developed in a way similar to Mini project I (The calculator). The grammar generator was used to generate a grammar file to be used by Sphinx recognizer. List of words mentioned in the grammar file were translated to phonemes in the dictionary file.

The old Entanglement implementation was embedded inside the new CMU Sphinx project. The GUI package contains files related to the GUI and the Engine package contains classes that are more related to the logic behind the game. And in the main package (edu.cmu.sphinx.demo.game) the only newly implemented class (Entanglement.java) is placed. The class uses CMU Sphinx to detect whatever the user says and reflect that on the game itself. For example, if the user says ”play two players” a new game is initalized with two players. This is done by loading the JFrames that actually shows the game itself and passing the board configuration to the JFrames.

3 How To Run Entanglement

To start playing Entanglement, simply build the project using the demo.xml file and then run the Entanglement.java class.

4 Grammar Generator

As a part of this project, we implemented a general Grammar Generator. Where given a list of sentences in a file, returns a file with a grammar that generates these sentences. The generator was implemented in Ruby. We chose Ruby because it makes it easier for us to use previously implemented Ruby Gems that could help us achieve this task.

In the following sections, we discuss how we implemented the generator and what Gems we used in this project.

5 Grouping Similar Sentences

In order to generate grammar from list of sentences, it is sensible to group similar sentences together and try to generate a single rule from a similar group of sentences. There are many ways of detecting similarity between differrent pieces of text. In our implementation, we group sentences by their Levenshtein distance. The Levenshtein distance is defined as the minimum number of characters that should be changed in a piece of text to make it exactly similar to another one. We used a Ruby Gem called Text to calcualte the Levenshtein distance between the given sentences and group them to extract a single rule for each group.

Once we group our sentences, we parse them and use the words in them to a single rule in the grammar file. The rule generated would be enough to generate all sentences in it's group.

6 How to Run The Generator

In a terminal, simply run ruby grammar_generator.rb. The script will write the grammar to a file called out.txt. it assumes the sentences to be located in a file called sentences.txt