Seth Nielsen
seth.emanuel.nielsen@gmail.com

Fairytale Story Generator

1. Introduction

We have been tasked with creating a program that randomly generates a story. The goal is to create a comprehensive story, as in no nonsense or incorrect grammar allowed, while using what can be described as a Markov-Chain.

The content is written by the author of this report, but the technical solution in itself is not dependent on the content being a fairytale per se, and much higher levels of complexity that can now be found are able to be implemented with little work. (Well, beyond means of keeping track of the complexity and said story).

2. Purpose of this document

This document aims to explain the technical side of the solution, meaning we will provide a chart over the structure of the program, and then provide a walkthrough of each of the classes found in the solution.

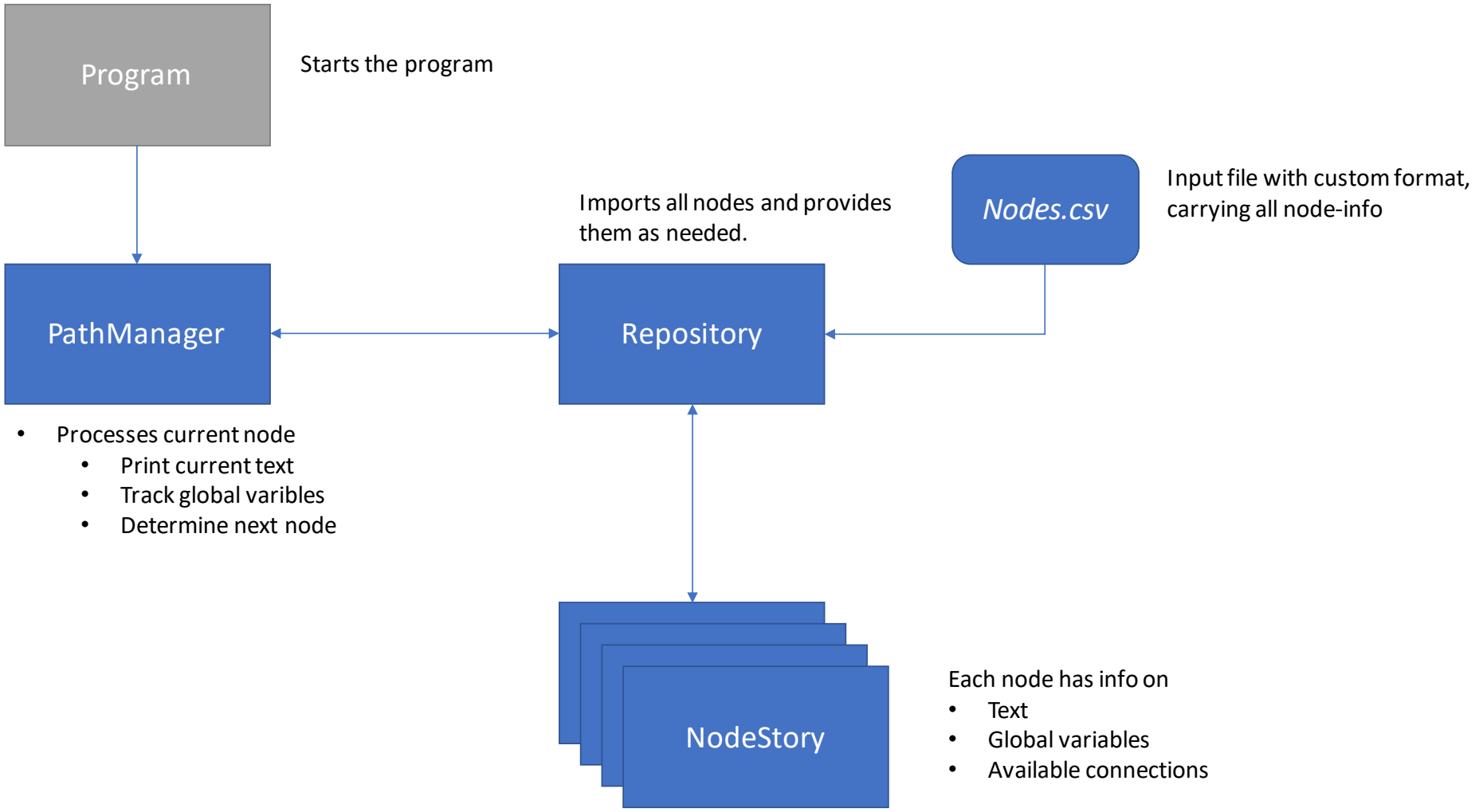
It should be noted though, that the document alone is reasonably not enough to reproduce the solution, and ideally the reader is intended to have the code available during reading, alternating between the two in order to get the full understanding.

3. Program

In this section we will provide a chart over the structure, of the program, as well as provide the aforementioned walkthrough of the key classes.

The reader is reminded here to not fret if not understanding the solution by simply checking the chart, but rather keep reading with the image kept in mind.

3.1. Structure chart



3.2. Classes

In this section we will provide a brief walkthrough of each class.

3.2.1. NodeStory

This is the most basic of all the components in the program. A given node contains information on the associated text that carries the story forward, information on any global variables that are to be permanently set, and finally links to connected nodes that are to follow along with their respective probabilities.

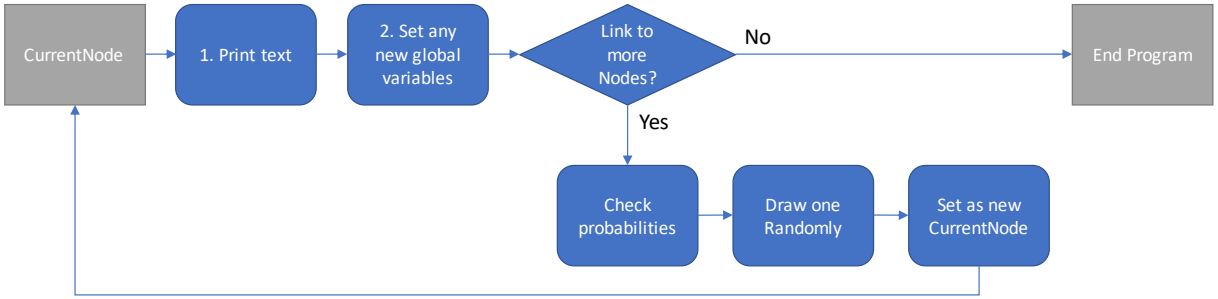
3.2.2. Repository

For starters this class is responsible for reading the .csv file that accompanies the program, which carries the information needed to create all nodes.

Second, this class is responsible for providing the rest of the program with access to desired nodes as needed.

3.2.3. PathManager

This class processes each node, and then asks the repository for any randomly drawn connected node if one exists. The process can be illustrated with the following flowchart:



3.2.4. Program

The main program is a very sparse one, it simply sets up an instance of the PathManager, and then asks it to start processing nodes, always starting with the “first” one in the story.

Appendix: Story Structure

While not explicitly needed to understand the program, we here provide a brief illustration of the story structure in the current implementation.