Visualization of a Turing Machine

Specification Document

James Merenda - Computer Science

Daniel Urban - Science and Technology Multidisciplinary Studies

Ji Sue Lee - Computer Science

California University of Pennsylvania

CSC 490: Senior Project

November 12, 2020

Instructor Comments/Evaluation

Table of Contents

Abstract

The Visualization of a Turing Machine project is an alternative teaching tool using visual elements and a programming interface. We decided to create this web-based Turing Machine program to further aid the process of understanding the concept of a Turing Machine in a more accessible way. Most hard copy texts describing the concept are extremely verbose and complicated. The Visualization of a Turing Machine is meant to alleviate the difficulties of the concept by providing a visual demonstration as well as an interface to try a different program or write custom programs. The project will be available on an online domain hosted by California University of Pennsylvania or available to download directly. This specification document entails the required steps necessary to create this project.

Description of the Document

Purpose and Use

The purpose of this document is to clarify the engineering required to develop this project as an online learning tool. Specifications for this project include, but are not limited to, the project's criteria and features. If this document's contents should not follow the business model or the initially provided guidelines, our professor may dispute their concerns. Developers and the professor will be able to properly discuss revisions to improve the final product in a timely manner. Should the specifications be satisfactory to both parties, this document becomes a binding contract.

Intended Audience

The intended audience for this document is the professor and software developers. Any further amendments or additions to this document may be included upon the client's request. Contained in this document are boundaries of the applications for the software developers to follow. Within these boundaries, the software developers can create software to the client's specifications. Additional diagrams are included in this document to assist the development team in writing structured software.

System Description

Overview

The Visualization of a Turing Machine project is a website that allows users to visualize Turing Machines by either loading preset programs or creating their own. The user can view a preset program simply by selecting the program at the landing screen. Alternatively, the user can select the option to create their own Turing Machine and will be presented with a menu to customize it. After selecting a preset program or creating one, the program will simulate the Turing Machine, providing feedback if the Turing Machine is erroneous.

Environment and Constraints

End User Profile

The most common end user will be students looking to improve their knowledge of Turing Machines. Users do not necessarily need to have experience with Turing Machines, but having experience allows for less erroneous Turing Machines to be created. The user will receive feedback on their Turing Machine if it is incorrect.

User Interaction

To interact with the website, the user needs a monitor and a pointing device, most likely a mouse, but alternatives such as a tablet pen will work. The user will use the pointing device to navigate through menus and customize Turing Machines.

Hardware Constraints

Since the project is web based, the minimum hardware required is a computer capable of running a modern web browser, which most students either own or have access to.

Software Constraints

The minimum software constraint is a web browser capable of running JavaScript, which most modern web browsers can do. Mozilla Firefox, Google Chrome, Microsoft Edge, and Safari will all be given special notice as common or default browsers.

Time Constraints

To facilitate feedback to the user, the simulation should compute the inputted program in a short span of time, about fifteen seconds or less.

Cost Constraints

The site itself will be free to use. Other cost constraints include the potential cost of a computer with the ability to run a modern web browser and connect to the Internet, which starts at around two hundred USD. Alternatively, since the project is targeted toward students, they may have access to a public library or other places with free access to computers.

Acceptance Test Criteria

Testers

Testing the program will be handled primarily by the Visualization of the Turing Machine development team. Bugs will be reported and fixed in a timely fashion. Once the team can no longer find any issues on their own, and if there is enough time to do so, a beta of the website can be sent to some volunteer testers to ensure any remaining bugs are ironed out. The development team will observe all testing to ensure everything runs smoothly and any problems are sorted before the final release.

Criteria for User Acceptance

For the Visualization of a Turing Machine project to be successful, it must meet the guidelines outlined by the user and proposed by the development team. These requirements include that the visualization must:

- Be on an easy-to-use and visually appealing website.
- Provide the user with instructions on how to operate the site and provide examples that can be edited and manipulated to the user's specifications.
- Allow the user to input their own code for the visualization to process.
- Display the visualizations correctly in a satisfying and easy-to-understand way.
- Give the user control over how the visualization plays out (speed, play, pause, etc.)
- Provide a link to access more information about the Turing Machine on a separate page of the site.

Integration of Separate Parts and Installation

Because the visualization will be presented on an interactive webpage, the user will not directly interact with or need to install any sort of software to their computer. Rather, the absolute bare minimum the user requires to use the visualization is simply access to the webpage. However, if the user does have coding experience, they can alter the examples or write their own that will get visualized by the webpage, providing them with a graphical representation of the resulting process. Any changes made by the user will be reflected in this visualization, including error messages if the user writes something unusable. The results of the program are directly influenced by what the user puts into it, even if it is merely the preinstalled examples.

System Modeling:

Functional: Use Case and Scenarios

The use case is the same as the one shown in the requirements document.

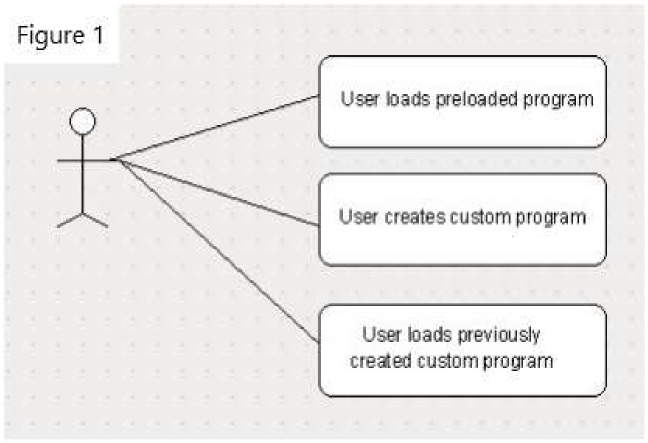


Figure 1
User loads preloaded program
User creates custom program
User loads previously created custom program

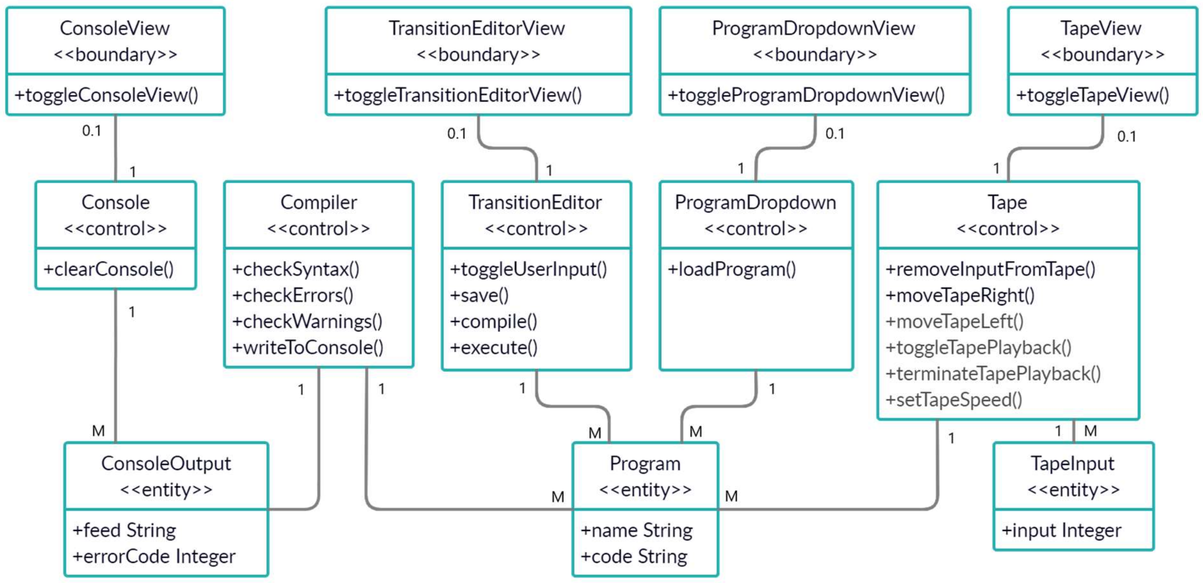
Description:
The user may either select a preloaded program to load or create a new program (Figure 1). If a preloaded program is selected, the user can then run it immediately. If the option to create a custom program is chosen, the editing interface appears, allowing the user to immediately create a new program and run it. Additionally, if the user has previously created a custom program, the option to load that program is also available (Figure 1).

The following scenarios will explain the steps followed by the user and program.

1) The user opens the webpage on their computer of choice.
2) The user can either choose from a list of preinstalled examples or write their own code for the program to visualize.
3) Whichever option is used, the on-screen code will be sent to the visualization program where it will be processed and broken up into a displayable animation that will be sent back to the client for the user to interact with.
4) The user can then either continue editing their code or use different code to repeat the process of visualization.

Entity: Class Diagrams

Figure 2



Class name / Description / Type

ConsoleView Class

The ConsoleView class is a boundary object used to toggle visibility for the console. This class handles output from the console.

TransitionEditorView Class

The TransitionEditorView class is a boundary object used to toggle visibility for the transition editor. This class handles input from the user and preloaded programs.

ProgramDropdownView Class

The ProgramDropdownView class is a boundary class used to toggle visibility for the dropdown menu used to access preloaded programs built into the webpage. This class allows users to select from a list of programs to use as an example run of the Turing Machine.

TapeView Class

The TapeView class is a boundary class object used to toggle visibility for the tape animation. This class handles animations from the transition editor.

Console Class

The Console class is a control object used to handle output from the compiler. The user may clear the console if the window becomes cluttered.

Compiler Class

The Compiler class is a control object used to interpret user input from the transition editor and send potential warnings and errors to the console.

TransitionEditor Class

The TransitionEditor class is a control object used to accept user-created programs. The user's programs may be compiled, executed, or saved to a file.

ProgramDropdown Class

The ProgramDropdown class is a control object used to load example programs for the user to observe. Included in these programs are comments for the user to follow along.

Tape Class

The Tape class is a control object used to control the movement of the magnetic tape to represent the transitions defined by the program. The class also may clear its input field for different inputs.

Attributes

ConsoleOutput Class

The ConsoleOutput class is an entity object used to display strings of potential errors. If no errors arise, then compilation succeeds.

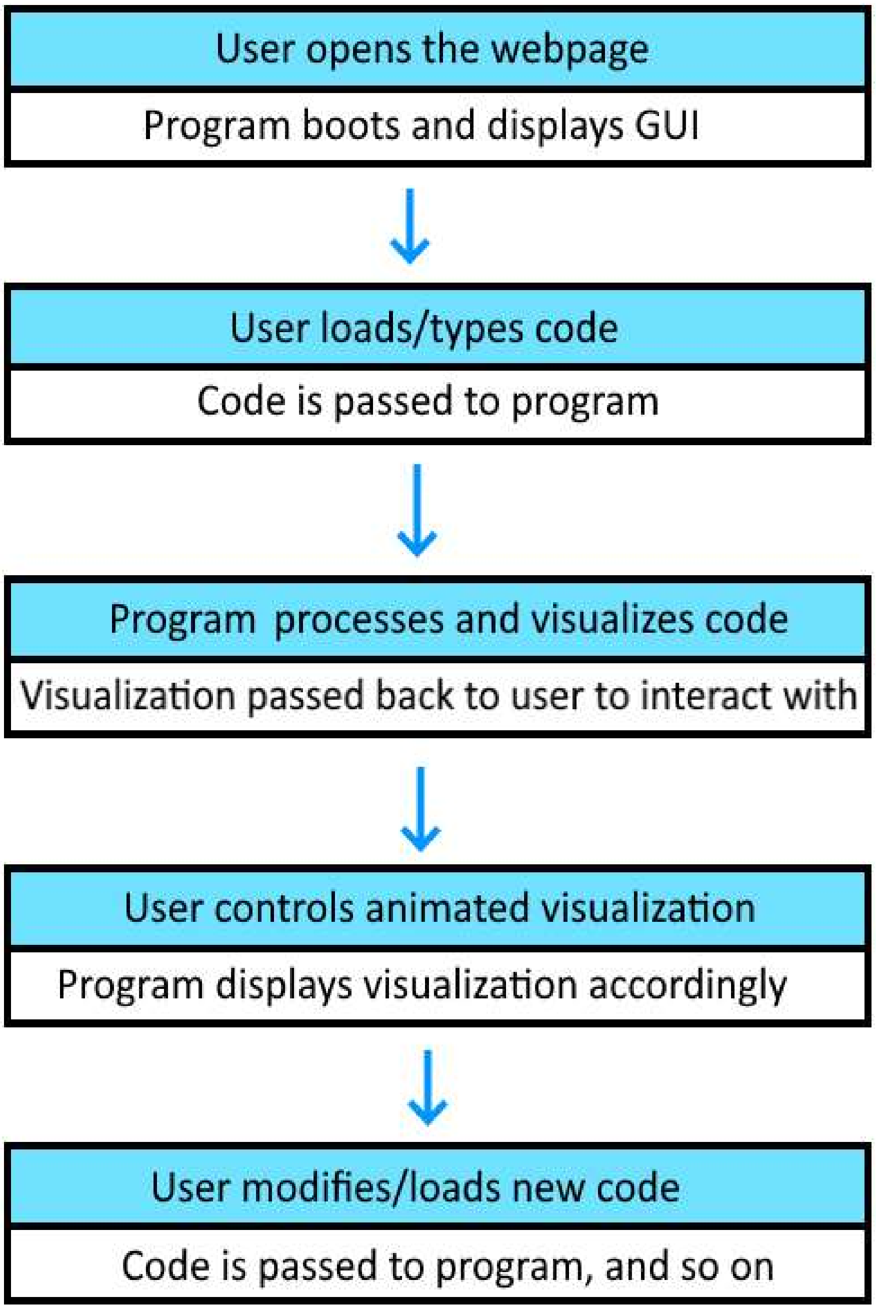
Program Class

The Program class is an entity object used to contain the user's custom program. The program's contents may be saved to the user's device or replaced with an example program on the webpage.

TapeInput Class

The TapeInput class is an entity object used to accept user input for the magnetic tape. The contents of this input may only be 0's or 1's representing binary.

Dynamic: Statechart


User opens the webpage
Program boots and displays GUI
User loads/types code
Code is passed to program
Program processes and visualizes code
Visualization passed back to user to interact with
User controls animated visualization
Program displays visualization accordingly
User modifies/loads new code
Code is passed to program, and so on

States

- User opens the webpage: The user navigates to the visualization website using their internet browser.
- Program boots and displays GUI: The visualization program loads and displays the initial state of the GUI with selectable/modifiable example code, etc.
- User loads/types of code: The user either selects example code on the website or types their own code, both of which can be modified by the user before submitting.
- Code is passed to program: The user's code is sent to be processed by the visualization program.
- Program processes and visualizes code: The program tests, compiles, and executes the provided code to ensure that it works correctly (if not, an error message is returned). If it works, the program processes the code into the interactable visualization.
- Visualization passed back to the user to interact with: The visualization (or the error message if the code did not compile successfully) is passed back to the client.
- User controls animated visualization: The user can control the animation's speed, whether it is playing/paused, jump forward/back a step, etc.
- Program displays visualization accordingly: Whichever commands the user submits to control the animation, the webpage will respond to and display on the visualization.
- User modifies/loads new code: The user either edits the code they have submitted, or they can load/type new code for the program to process.
- Code is passed to the program, and so on: Like before, the code is sent to the visualization program to be processed, and the cycle repeats.

Events

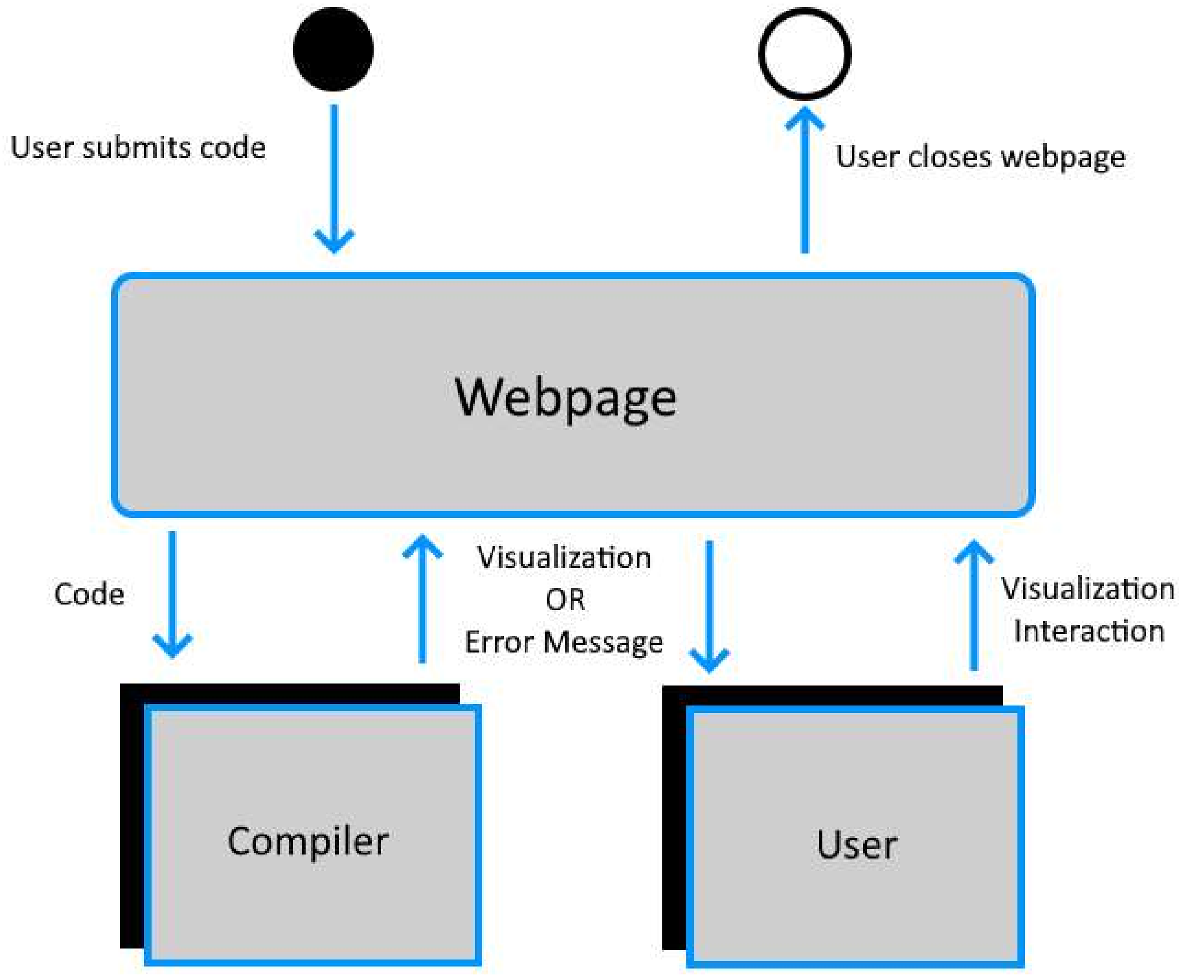
Most of the events processed deal with how the user interacts with the code and how the user interacts with controlling the visualization. When modifying the code, nothing is sent to the program to process until the user submits it by pressing a button indicated on the webpage. With controlling the visualization, however, any interactions the user makes with the controls are picked up immediately, requiring constant checks for interrupts provided by the user.

Transitions

Transitions will occur when the user interacts with the controls of the program by pressing buttons on the webpage. The initial state of the program is to display a blank webpage and prompt the user to input their own code or select an example. Once the user inputs their code of choice and presses the submit button, the code is sent to the webpage and, once error-checked, processed into a visualization that the user can interact with. The user can also still access their code and resubmit it at any time, which will be resent to the program and refresh the visualization accordingly.

Dataflow Diagram

Figure 3



Components / Tools Needed

The components and tools needed for this project includes an internet-enabled device, a web browser, and an internet connection. The device will open a webpage containing the project. A text editor and a web browser in tandem will be used to arrange assets and program logic into the project.

Appendix I: Glossary of Terms

Web Browser:

an application used to access and view websites.

Webpage:

a hypertext document on the World Wide Web.

Text Editor:

a system or program that allows a user to edit text.

Turing Machine:

a mathematical model of computation that defines an abstract machine, which manipulates symbols on a strip of tape according to a table of rules.

JavaScript:

an object-oriented computer programming language commonly used to create interactive effects within web browsers.

Mozilla Firefox:

a free and open-source web browser developed by the Mozilla Foundation and its subsidiary, the Mozilla Corporation.

Google Chrome:

a cross-platform web browser developed by Google.

Microsoft Edge:

a cross-platform web browser developed by Microsoft.

Safari:

a graphical web browser developed by Apple, based on the WebKit engine.

Appendix II: Team Details

This document was written and edited by the following individuals, with their specific contributions listed as follows:

- Ji Sue Lee: Ji Sue wrote up to and including the Environment and Constraints section of the System Description.

- James Merenda: James handled writing the Abstract, Description of Document, Components/Tools needed, and was the one who spoke with the writing center, along with being the chief editor of the document.

- Daniel Urban: Dan wrote the rest of the System Description (Acceptance Test Criteria and Separate Parts and Installation), part of the System Modeling section, and the Dataflow section.

Each team member also provided editing and feedback to the entirety of the document and worked together in writing the Appendix section.

Appendix III: Writing Center Report

Appendix IV: Workflow Authentication

I, James Merenda, hereby attest that I have done the work documented herein.

Signature: ____

Date: _______________

I, Ji Sue Lee, hereby attest that I have done the work documented herein.

Signature: ____

Date: _______________

I, Daniel Urban, hereby attest that I have done the work documented herein.

Signature: ____

Date: _______________

References

Online Turing Machine Simulator. (n.d.). Retrieved November 12, 2020, from https://turingmachinesimulator.com/

Turing Machine Simulator. (n.d.). Retrieved November 12, 2020, from http://morphett.info/turing/

Turing machine visualization. (n.d.). Retrieved November 12, 2020, from https://turingmachine.io/