Visualization of a Turing Machine

Design Document

James Merenda - Computer Science

Daniel Urban - Science and Technology Multidisciplinary Studies

Ji Sue Lee - Computer Science

Brett George - Computer Science

California University of Pennsylvania

CSC 490: Senior Project

December 4, 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. The Visualization is a web-based application which allows for user interaction and animation to achieve a better understanding of Turing Machines. This design document is to be used as a blueprint when implementing and realizing the software. Below, developers will find details regarding software design, class structure, and data flow.

Description of Document

Purpose and Use

This document is a detailed design of the software implementation for this project. By planning details such as classes, data flow, and program architecture, we can show a more in-depth outline of the project. Those more in-depth details come in the form of module cohesion and coupling, function input and output, and testing plans. This document should be used as a reference by developers to save time and cost during the implantation phase. An implementation timeline will be included to provide a scheduled list of expectations for project development.

Ties to the Specification Document

This document further explains and details the plans made in the specification document for the Visualization of a Turing Machine project. This document is a more technical breakdown of the software development and design.

Intended Audience

This document is intended to be read by developers of the Visualization of a Turing Machine project. Users will be unable to use this document as a manual or tutorial for the web application.

Project Block Diagram

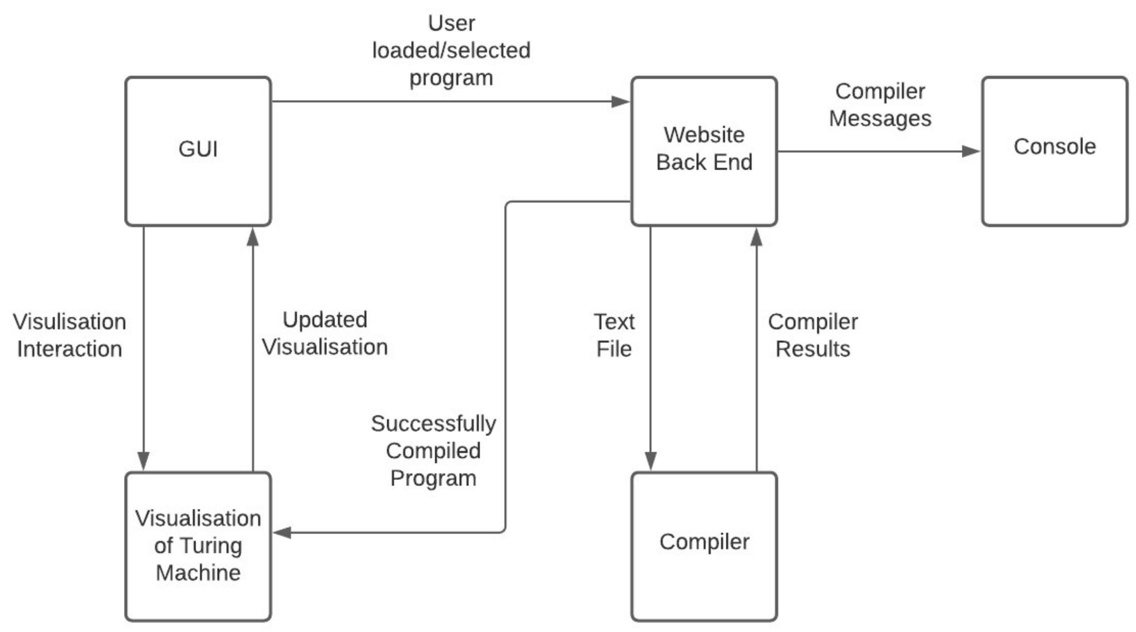

Figure 1: Project Block Diagram

Description

The web application will consist of four parts, which include the graphical user interface (GUI), back-end, compiler, and the console. A user can load the web page and then choose to select a pre-made program or write their own program using a text editor. That chosen program will be sent to the compiler. The results of the compilation will be sent to the console. A successfully compiled program will be able to be visualized and animated. The user can step through and animate the visual program using the GUI. The user can choose to edit the selected program or choose a new one at any point, and the compilation process will need to be repeated.

Design Details

System Modules and Responsibilities

Architectural Diagram

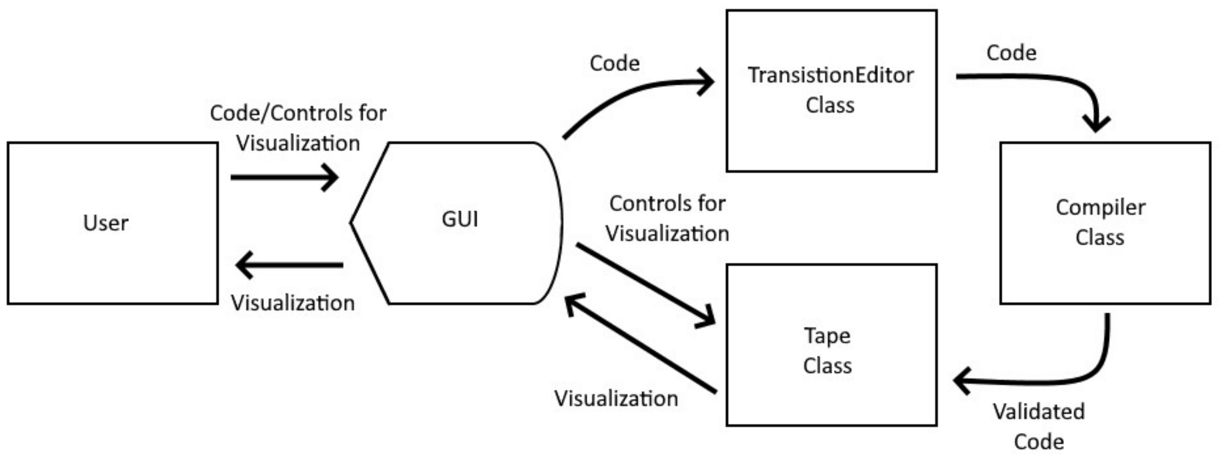

Figure 2: Architectural Diagram

Description:

The user can input their own code or select from a handful of preinstalled code options via the GUI on the website. If they already have a visualization in progress, they can also control the visualization's speed, whether it is playing or paused in the GUI. The GUI passes the code to the TransisitionEditor class and/or the Controls for the Visualization to the Tape Class. The TransisitionEditor processes the code and passes it to the Compiler Class, where it ensures the code is error-free before passing the validated code to the Tape class. There, the Tape class processes the visualization and sends it back to the GUI to display. If the tape class received user input for controlling the currently active visualization, it will manipulate the visualization as requested and send it back to the GUI for the user to receive.

Module Cohesion:

Module Cohesion is present in our project in the way that each class is laid out and programmed. While they all work together, they are only responsible for one or a few specific tasks each, which in turn makes the project easy to read, edit, and debug.

Most of the main classes also have a separate View class, which handles the visibility of certain functions and displaying the GUI, results, etc. cleanly and effectively. This allows for an easy way to check if errors are caused by the actual program itself or just the View classes and allows for changes to be made to the GUI easily without editing the main code.

Module Coupling:

This project utilizes Module Coupling by breaking each process into different steps which are handled by separate classes. Each class must work together and process the data appropriately for the result to be presentable and accurate. For instance, the TransistionEditor class must receive code, either in the form of preinstalled examples or custom user submissions, to pass on to the Compiler class, which then must check the code before sending it to the Tape class, where the visualization must be properly processed and made before being passed to the website. All the pieces must work together to achieve a product.

Design Analysis

Data Flow Analysis

The data flow is handled by the visualizer's compiler class. After the user either selects an example Turing Machine, selects a previously created one or creates their own, the TransitionEditor takes the input and passes it to the Compiler.

The compiler then processes the data into a visualization. If there are any errors, the compiler will output to the console as necessary and the conversion will fail. If the input is successfully translated, the animated visualization is then presented to the user.

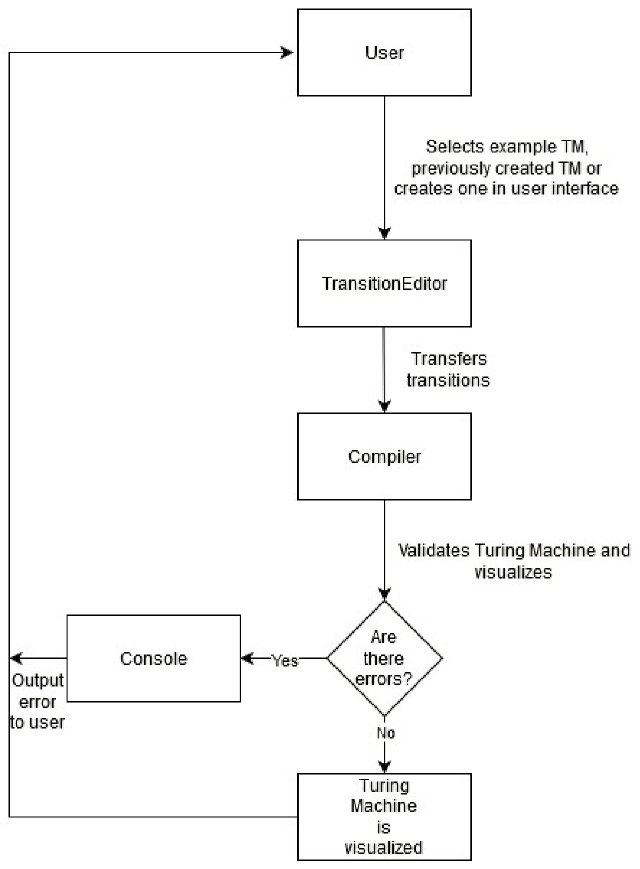
User
Selects example TM, previously created TM or creates one in user interface
TransitionEditor
Transfers transitions
Compiler
Validates Turing Machine and visualizes
Are there errors?
Yes
Console
Output error to user
No
Turing Machine is visualized

Figure 3: Data Flow Diagram

Design Organization

Detailed Tabular Description of Classes/Objects

Class name: ConsoleView

Class description: The Console View class is a boundary object used to manipulate the visibility of the console on the webpage. This class handles interaction between the user and the console.

Class data members: Console Object, View Object

Class member functions: toggleConsoleView()

Class name: TransitionEditorView

Class description: The Transition Editor View class is a boundary object used to manipulate the visibility of the transition editor on the webpage. This class handles interaction between the user and the transition editor.

Class data members: TextArea Object, View Object

Class member functions: toggleTransitionEditorView()

Class name: ProgramDropdownView

Class description: The Program Dropdown View class is a boundary object used to manipulate the visibility of the example program dropdown menu on the webpage. This class handles interaction between the user and the program dropdown selector.

Class data members: View Object

Class member functions: toggleProgramDropdownView()

Class name: TapeView

Class description: The Tape View class is a boundary object used to manipulate the visibility of the animated magnetic tape on the webpage. This class handles interaction between the user and the animated magnetic tape.

Class data members: View Object

Class member functions: toggleTapeView()

Class name: Console

Class description: The Console class is a control class used to output information to the user such as error messages or compilation messages. Each message contains an error code specifying the types of errors and possible solutions.

Class data members: View Object, TextArea Object

Class member functions: clearConsole()

Class name: Compiler

Class description: The Compiler class is a control class used to configure user input into animations for the Turing Machine. If there are issues with the user's input or transitions, error messages are sent to the console.

Class data members: String transitionCode, String message, Int errorCode

Class member functions: checkSyntax(), checkErrors(), checkWarnings(), writeToConsole()

Class name: TransitionEditor

Class description: The Transition Editor class is a control class used to transfer the user's transitions to the compiler for processing. This class also handles saving and executing programs written by the user or loaded from example programs.

Class data members: String transitionCode

Class member functions: toggleUserInput(), save(), compile(), execute()

Class name: ProgramDropdown

Class description: The Program Dropdown class is a control class used to insert example transitions and input for the user. These example programs are accessed by a dropdown menu for the user's convenience.

Class data members: String transitionCode, String examplePrograms

Class member functions: loadProgram()

Class name: Tape

Class description: The Tape class is a control class used to manipulate the magnetic tape. Manipulations include animations, playback speed, and toggle playback.

Class data members: String inputText

Class member functions: removeInputFromTape(), moveTapeRight(), moveTapeLeft(), toggleTapePlayback(), terminateTapePlayback(), setTapeSpeed()

Class name: ConsoleOutput

Class description: ConsoleOutput will contain any feedback from the compiler and an error code.

Class data members: String feed, Integer errorCode

Class member functions: feedString(), errorCodeInteger()

Class name: Program

Class description: Program will contain transition code and the name of the program.

Class data members: String name, String code

Class member functions: nameString(), codeString()

Class name: TapeInput

Class description: TapeInput will contain an input value for the program.

Class data members: String input

Class member functions: inputString()

Functional Description

ConsoleView Class

toggleConsoleView()

Input:

The toggleConsoleView() function has no input.

Output:

The toggleConsoleView() function will output the console field to the screen.

Return Parameters:

The toggleConsoleView() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

TransitionEditorView Class

toggleTransitionEditorView()

Input:

The toggleTransitionEditorView() function has no input.

Output:

The toggleTransitionEditorView() function will output the transition editor field to the screen.

Return Parameters:

The toggleTransitionEditorView() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

ProgramDropdownView Class

toggleProgramDropdownView()

Input:

The toggleProgramDropdownView() function has no input.

Output:

The toggleProgramDropdownView() function will output the program dropdown menu to the screen.

Return Parameters:

The toggleProgramDropdownView() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

TapeView Class

toggleTapeView()

Input:

The toggleTapeView() function has no input.

Output:

The toggleTapeView() function will output the animated magnetic tape to the screen.

Return Parameters:

The toggleTapeView() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

Console Class

clearConsole()

Input:

The clearConsole() function has no input.

Output:

The clearConsole() function will output multiple new lines to the console, clearing clutter from previous messages.

Return Parameters:

The clearConsole() function has no return parameters.

Types:

String.

Compiler Class

checkSyntax()

Input:

The checkSyntax() function will accept transition code from the transition editor field, checking if proper formatting was used.

Output:

The checkSyntax() function will output messages relating to formatting errors to the console.

Return Parameters:

The checkSyntax() function returns strings appropriate for each type of syntax error written by the user.

Types:

String.

checkErrors()

Input:

The checkError() function will accept transition code from the transition editor field, checking if any compilation errors occurred.

Output:

The checkError() function will output messages relating to transition errors to the console.

Return Parameters:

The checkSyntax() function returns strings appropriate for each type of transition error written by the user.

Types:

String.

checkWarnings()

Input:

The checkWarnings() function will accept transition code from the transition editor field, checking if any undesirable transition patterns are detected.

Output:

The checkWarnings() function will output messages relating to undesirable transition patterns to the console.

Return Parameters:

The checkWarnings() function returns strings appropriate for each type of undesirable transition pattern detected by the compiler.

Types:

String.

writeToConsole()

Input:

The writeToConsole() function will accept strings from the programmer.

Output:

The writeToConsole() function will output strings written by the programmer to the console.

Return Parameters:

The writeToConsole() function has no return parameters.

Types:

String.

TransitionEditor Class

toggleUserInput()

Input:

The toggleUserInput() function has no input.

Output:

The toggleUserInput() function will inhibit user input in the transition editor and magnetic tape input field.

Return Parameters:

The toggleUserInput() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

save()

Input:

The save() function accepts transitions from the transition editor.

Output:

The save() function creates a text file containing all transitions from the transition editor.

Return Parameters:

The save() function will return messages relating to creating the text file.

Types:

I/O, String.

compile()

Input:

The compile() function accepts transitions from the transition editor.

Output:

The compile() function will output messages relating to undesirable transition patterns to the console.

Return Parameters:

The compile() function has no return parameters.

Types:

String.

ProgramDropdown Class

loadProgram()

Input:

The loadProgram() function has no input.

Output:

The loadProgram() function inserts a prewritten list of transitions into the transition editor.

Return Parameters:

The loadProgram() function has no return parameters.

Types:

String.

Tape Class

removeInputFromTape()

Input:

The removeInputFromTape() function has no input.

Output:

The removeInputFromTape() function removes the displayed values from the animated Turing Machine.

Return Parameters:

The removeInputFromTape() function has no return parameters.

Types:

String.

moveTapeRight()

Input:

The moveTapeRight() function has no input.

Output:

The moveTapeRight() function moves the animated magnetic tape to the right, moving relevant data in the tape in the same direction, respectively.

Return Parameters:

The moveTapeRight() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

moveTapeLeft()

Input:

The moveTapeLeft() function has no input.

Output:

The moveTapeRight() function moves the animated magnetic tape to the left, moving relevant data in the tape in the same direction, respectively.

Return Parameters:

The moveTapeLeft() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

toggleTapePlayback()

Input:

The toggleTapePlayback() function accepts the previous playback speed.

Output:

The toggleTapePlayback() function either switches from some speed to no speed or vice versa.

Return Parameters:

The toggleTapePlayback() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

terminateTapePlayback()

Input:

The terminateTapePlayback() function has no input.

Output:

The terminateTapePlayback() function stops playback, resets tape input, and moves the read-head to the beginning of the input.

Return Parameters:

The terminateTapePlayback() function has no return parameters.

Types:

The data types used within this function are based on modern browsers to edit visibility of CSS and HTML elements.

setTapeSpeed()

Input:

The setTapeSpeed() function takes a value between 1 and 0.

Output:

The setTapeSpeed() function sets the current playback speed to the user defined speed.

Return Parameters:

The setTapeSpeed() function has no return parameters.

Types:

Float

Modules Used

At this time, the development team does not feel that modules will be needed for this project. Modules that are specifically designed with webapp animation in mind are too bloated for the usefulness they may have.

Files Accessed

A server will host source files necessary for the website to function. These files include HTML, CSS, and JavaScript files, respectively. The user may download transitions from the website in the form of a text file.

Real-time Requirements

The website needs to run at a reasonable speed. If the website feels bloated and slow, the user may find this tiresome and find different learning solutions. Ideally, the interaction between the user and the website should be simple and straightforward with seamless response times.

Messages

Messages sent in this project will be communicated to the user using the console. Other messages in the console will be made by the software development team to introduce the user to the environment. This includes the following data:

Message Type	**Source / Destination**	**Data**
writeToConsole()	Backend->Console	Interaction I/O String
checkWarnings()	Console->User	Compilation Results String
checkErrors()	Console->User	Compilation Results String
checkSyntax()	Console->User	Compilation Results String
clearConsole()	User->Console	User Interaction String

Figure 4: Message Table

Narrative / PDL

```
Open website:
    1. Create transitions.
    2. Load example transitions from dropdown.
    3. Clear console
Create transitions/Load transitions from dropdown:
    1. User edits transitions in the transition editor or loads example transitions
       from dropdown.
    2. Compile transitions
       Any Errors?
           No? Go to Play animation.
           Yes? Check errors and return to (1).
Play animation:
    1. Add input to tape.
       Input in tape?
           Yes? Go to (2).
           No? Display message in console and go back to (1).
2.      Acquire playback speed.
```

```
Custom playback speed set?
        No? Go to (3)
        Yes? Set playback speed to user-defined playback speed.
3.      Wait for user input.
    a.  Play
        No? Go to (3).
        Yes? Play animation and go to (3).
    b.  Step
        No? Go to (3).
        Yes? Step animation by one frame and go to (3).
    c.  Pause
        No? Go to (3).
        Yes? Stop animation at the current frame and go to (3).
    d.  Stop
        No? Go to (3).
        Yes? Stop animation, return to the first frame, and go to (3).
    e.  Change tape input.
        Return to (1).
    f.  Otherwise
        Return to the open website state.
```

Decision: Programming Language / Reuse / Portability

The visualizer itself will be written in the programming language JavaScript, while the website it is hosted on will be written in the scripting languages HTML and CSS. Since they are standard web languages, most if not all modern Internet browsers support them, which in turn allows portability across most computers and phones with little or no extra code needed.

Implementation Timeline

	January	February	March	April	May
Module development	🟥	🟥			
Module integration		🟥	🟥		
Validation			🟩	🟩	
Manuals				🟩	🟩
Testing	🟥	🟥	🟥	🟥	🟥

Figure 5: Implementation Timeline Table

Design Testing

In earlier stages, testing will generally be non-execution based since modules are still being written. Later, modules will be tested individually with test data to ensure they function alone before testing in conjunction with other modules. Since Turing Machines can be solved by hand, modules' correctness can be easily evaluated to see if and where faults may occur.

Appendix: Team Details

This Design Document was written by the following individuals, with their individual contributions cited as follows.

James Merenda: James was the main editor of the document and wrote the Tabular Description of Classes along with its requirements and the Functional Description, as well as handled the Writing Center discussion.

Ji Sue Lee: Ji Sue wrote the Data Flow or Transaction Analysis section, the Decision: Programming Language/Reuse/Portability section, and the Design Testing section, along with creating the Implementation Timeline.

Daniel Urban: Dan handled the System Modules and Responsibilities portion of the Design Details segment, which included creating the Architectural Diagram and writing the Module Cohesion and Module Coupling sections.

Brett George: Brett wrote the Title Page, Abstract, Purpose and Use, Ties to the Specification Document, and Intended Audience section, along with creating the Project Block Diagram with its description.

Appendix: Writing Center Report

Appendix: 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: _______________

I, Brett George, 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/

JavaScript HTML DOM Animation. (n.d.). Retrieved December 03, 2020, from https://www.w3schools.com/js/js_htmldom_animate.asp