**FSM Generator**

**Programmer Manual**

# Contributors

Chris Loos

Justin Butler

# Vision Statement

The goal of this software is to create a solution for simplifying the validation process in large, enterprise software. Our approach is based off a research paper written, presented, and provided by Dr. Finkbine. Potentially this software can be adopted into his research to develop an effective, large production version of the concept.

# Introduction

In traditional coding practices, programs may consist of modules or methods which contain large, interweaving blocks of code which may in turn trigger many other functions. This vast complexity increases the chance of unexpected errors and bugs from future development incompatibilities. Our solution utilizes the portability and simplicity of a finite state machine to build individual pieces of a large system. This approach allows heavy reusability and faster validation of code to save on debugging and maintenance costs.

# Components

This system has three major components: GUI, Exporter, and Processor.

# GUI

This component runs on JavaFX and allows users to create and manipulate finite state machines visually. This is done by taking canvas coordinates to calculate movement and closest points on connecting state nodes. This interface is accessible once a new project is created or an existing project is loaded.

# Exporter

This component consists of multiple code-generating classes for the purposes of exporting the finite state machine as it has been created. These exporters use the following formats:

Design view: Saved as a .FXML format

FSM code: Saved as a .FSM format

Python code: Passes a temporary .FSM file to the processor

# Processor

This component consists of the transpiler and code importers. The transpiler accepts a .FSM format input file to generate a valid Python script. The importers take valid .FSM format files or any valid XML file to draw a finite state machine on the GUI canvas for manipulation and processing.

# Tool Overview

**Java** – The entire program was written in Java. This provides the desired portability across systems via the java virtual machine (JVM). Due to the simplistic nature of the language and the access to ample learning material, this language is stable enough for lasting production.

**Python** – This program only transpiles Python code due to client specifications. There is easy potential to add other languages, but the need wasn’t there.

**XML** – The GUI designer saves the project in an XML format to save all of the information generated during the creation process. The system will accept .FXML, .XML, or .TXT format files as long as they contain valid XML markup.

**FSM** – Finite State Machines make up the entirety of this system’s purpose. These are mathematical models used to simulate sequential logic graphically.

# Further Development

With another year in development, we would like to implement other languages for transpiling and finish / improve the GUI with more features to make the process feasible for larger development. We are not sure of what extent new features would entail, so further research into the topic would be needed.

Missing / Un-implemented features as a pie in the sky

* + Delete Nodes
  + Edit Nodes
  + Delete Links
  + Edit Links
  + Export as image
  + Export into other languages
  + GUI Settings
  + Moving / Arcing Links
  + Self-linking visually
  + FSM validation