

Team Project - Team C5

Final Report

24.03.2016

Thomas Clarke, Mihnea Patentasu, Piotr Wilczyński, Danyil
Ilchenko, Owen Pemberton, Botond Megyesfalvi

Table of Contents

- Introduction
- Design
 - Description
 - Modules
 - GUI - Main modules
 - GUI - Canvas
 - Automaton
 - Testing
- Visualisation, GUI and HCI Design
 - Automata Design
 - GUI Design
 - Error messages and Input Validation
 - Help and About Pages
- Software Engineering and Risk Management
 - Overview
 - Agile
 - Model-View-Controller
 - Pair programming
 - Risks
- Evaluation
- Team Work
 - General Organisation
 - Problems, conflicts and solutions
 - Communication
 - Google Drive
 - Facebook
 - Trello
 - Skype
 - Gantt Chart
 - SVN
- Summary
- Individual Summaries
- References
- Appendices

Introduction

The aim of the project was to create a piece of software that conveys knowledge about and demonstrates inner workings of an algorithm through computer generated visualisation. The choice of the algorithms was free and therefore we decided to visualise transforming a regular expression into finite-state machine of type “acceptor”. A finite-state machine (or automata) is a state machine that can represent a language. This decision was based on several factors; firstly as every member of the team was studying basic automata theory as part of the Models of Computation module, we were familiar with regular expression based automata and knew that algorithms for generating such state machines existed. However, since creating a state machine equivalent to a given regular expression is quite intuitive, most students taking the module do not use any of the available algorithms and do not possess explicit, formal knowledge on how they work. This presented an opportunity to create a learning tool which would familiarise students such as ourselves with various algorithms for converting regular expressions into automata. Secondly, since finite-state machines are essentially graphs (states connected with transitions), they are naturally fit for visualisation.

The algorithms that are implemented and visualised in the scope of the project are Thompson’s construction and powerset construction.

Thompson’s construction builds a nondeterministic finite automaton (NFA) that represents the same language as a given regular expression using a set of rules, each assigning a template-like NFA to one of the basic regular expression patterns (concatenation, disjunction, Kleene star and the base case - single character).

Powerset construction takes a NFA and turns it into a deterministic finite automata (DFA) by combining single NFA states into sets that represent the states of DFA.

The finished program allows user to enter a regular expression which will prompt the step-by-step generation of NFA. For the visualisation of this process, a top-down approach was picked: the initial automaton is represented by two states (a starting state and an accepting state) connected by a single edge that encompasses the regular expression in its entirety; at each consecutive step, the outermost component of the expression is expanded into its corresponding Thompson’s construction template until all the edges are either empty (epsilon), or single-character transitions. The sections of NFA which are being expanded in any moment are highlighted. Upon building the NFA the user is presented with an option to turn it into a DFA. Selecting this option launches the visualisation of powerset construction. At this stage, both NFA and DFA are displayed. As the program follows the edges available in NFA and combines its states into closures, new closures are added as states to the DFA.

Highlighting is utilised again to stress which parts of the NFA and the DFA are processed at every step.

Apart from visualising the process of building automata, the application includes functionality to test a string against an automaton and see if it is accepted by its language (the process is visualised) and other features such as a colorblind mode, saving and loading regular expressions as well as printing the NFA or the DFA at any stage of the generation.

The following report describes the development process in detail including the key class structure and the interface design concepts, the software engineering techniques applied, the test results and the general work order.

Design

Description

The original specification formed a basis for how the system would work as far as the user could see. The design documentation details how the system is actually implemented and for all of the classes, including the GUI classes, UML class diagrams have been created and an overall package diagram has also been generated. The diagrams can be seen in the appendix.

Furthermore, it is based on the modified model/view/controller (MVC) style of system architecture, but changes it slightly. The model is the automata (all backend work) and the view and the controller are the visualisation and GUI.

Modules

GUI - Main modules

The view and controls are held within this main GUI modules. The view takes the form of the visualisation, where the drawing of the actual graph of the automaton is done in the canvas module discussed later. The controls are the menu bar at the top and the set of controls at the bottom. We also implemented a tabbing system so the user can have multiple instances of automatas at once, doing different things (for example one tab could be showing generation, with another tab showing testing a string).

An important part of this structure is the Document interface. It defines the minimal functions of a document on a tab. This is mainly applicable to the RegexDocument class which holds a NFA animation followed by a DFA animation and the

TestStringDocument class which holds the animation and GUI controls for testing a string. There are also documents which simply hold the 'About' and 'Help' HTML pages which are displayed to the user in a web view. For the documents that show the visualisation, they add to the tab the canvas, controls specific to the type of document, and the playback controls (for playing animations).

The controls specific to the documents just allow the user to input a string and push that into the automaton processor. They do have other features including keeping track of what the state of the automaton is and what should be drawn. An important part of these classes are that they run the automaton generator parts of the program in another thread so as not to slow down the user interface. This also means we need to run all GUI updates in a JavaFX 'run later' thread to make the program thread safe.

The PlaybackControls object is the GUI panel for controlling the current frame to be passed from the animation to the canvas for drawing. When playing, it uses a new thread to wait, depending upon the user set speed, and move on to the next frame (or previous if in reverse). It can also be used by the user to skip frames, or skip to the end of animations in their entirety.

The menu bar itself is a class which holds all of the buttons presented, and has control over the tab list as it needs that for its operations (such as opening, closing etc.). With the Tab list, it keeps a Document list, where a Document is something keeping track of what the user is doing in the tab and supplying resources to it such as a RegexDocument. It is important for this object to also keep track of the currently selected tab, so that it can run 'save' or 'test a string' on what the user is currently looking at.

Also accessible via the menu bar are 'About' and 'Help' pages. These open in new tabs and either give the user our logo and names, or help with how to use the product. It is enforced by the main menu manager that a maximum of one of each of these documents are open at a time. The application switches to the currently open one if the user tries to open another.

An important feature of the main menu is also a 'toggle colour blind mode' button. This sets a flag (initially false) and sends the update events through to the canvas, which uses the flag to switch between normal colours and colourblind colours when drawing the automaton and the text on the screen. To choose the colours used, the team spoke to a colour blind user to help determine the best selection of colours to accommodate for the maximum number of colour blind people (as there are several types of colour blindness).

GUI - Canvas

We decided to give the main visualisation its own module to help reduce dependencies on other sections, so it was easier to maintain and use for different types of drawings (mainly generating automata and testing string automata). This means giving control of the canvas almost completely to the components classes described above. This also effectively forms the view of the MVC model.

The main class in the gui.canvas package extends a normal JavaFX canvas, so it is easy to change information about and add our own methods to. Alongside this, we created a class purpose built for holding a 'frame'. A 'frame' is one state of what we want to draw on screen, holding information about positions of nodes and edges and a few other small bits of information including the text displayed at the bottom of the screen.

The information about nodes and edges was stored in their own two classes, VisualState and VisualEdge, so that it was easier to build the frame (it is a collection of objects), but also so we could give them custom draw methods where we just pass the canvas we want to draw on. Furthermore, we also store information in the nodes and edges about the highlighting and the type of state for nodes and the type of curve for edges for use when drawing.

Using the above structure, it is easy to perform a draw routine. The canvas initially does some calculations, with information passed from the components, to set up sensible scaling and sizing of nodes. This mainly takes into account the size of the screen the user is using along with the number of nodes in the automata. It then goes through and draws the graph, edges then states. It draws them in this specific order because the edges extend into the middle of the states, so by drawing the states after the edges, we ensure no objects are drawn over each other.

Automaton

Under our revision of the MVC, this forms the core model. The automaton package contains everything related to interpreting user input, generating the automaton from that and processing it into frames of generation. It will then deliver that to the GUI section.

There is an automaton class which holds all the information about the automaton we need to perform calculation on, and an animation class which holds basic information about the frames of the animation which is extended by a NFA, a DFA and a test string class. These animation classes use information about the automata to generate their frames showing their respective purpose. Once they have finished

processing the information, they return the frames to the GUI module which displays a certain frame.

Moreover, this means that the automaton module handles the graph layout. For the prototype and early tests (before a custom one was written by the team), we used a library called JUNG which would lay the graph out. It gave us the knowledge that our graph was being generated correctly in terms of the edges and states and while we always wanted to write our own implementation for the actual layout, this emphasised the need for it even more. The key to the layout was having knowledge of the overall graph at each frame and what the final graph should look like. Our final implementation requires no external libraries and contains only code written by the team to lay out the graph.

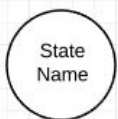
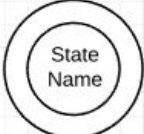
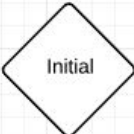
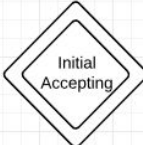
Testing

We made a package containing all the automated testing. This took the form of a series of JUnit tests to check that the algorithms were calculating the correct automata (and closures within them) and frames for the visualisation. This module, while not being core to the functionality, is vital to get correct output from as it helped us determine whether the program runs correctly and make the whole system more reliable and easier to maintain while working on other parts of it. This is also worth including in the source code as anyone who would like to extend the project is able to reuse our tests and repurpose them to check their own code.

Visualisation, GUI and HCI Design

Automata Design

As regards the automata, we wanted our design to be simplistic and clear and we intended to use the same shapes that we learnt in the Models of Computation module for different types of states.

Normal State	Accepting	Initial State	Initial and Accepting
			

For the nodes' sizes, we scale the entire automaton based on its width and height and also the size of the window. Whilst this implementation works effectively for realistically sized automata, for larger automata it does not work as well since we have a minimum size and the visualisation gets more difficult to see as it gets larger. This is one area for future expansion in the program - to allow the user to zoom (scale) and move the automata around the window.

Our visualisation colours the states depending on the type and also whether they are being used in the current step of building the automata or determining. Accepting states are coloured green for better differentiation and states being used in the current step are coloured in yellow.

For accessibility we included a colorblind option that changes our colour scheme. We used a program called Color Oracle that alters the colours of our program's window to simulate colour blindness. Using the colour blindness simulator allowed us to decide on a colour scheme that would allow most colour blind users to differentiate between highlighted and unhighlighted states.

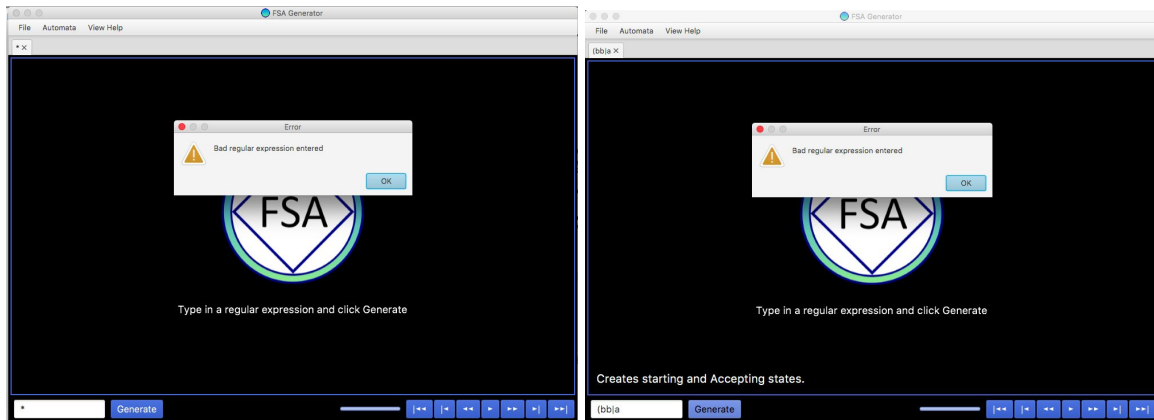
GUI Design

During the team meeting where we decided to build a Regex to FSA construction program, we also drew basic GUI designs so that we gained ideas about the project's direction and features from the end-user's perspective. Example of these ideas include the saving and loading of regex strings and play/pause controls for the animation of building an automata. Later these sketches were formalised digitally. These designs can be seen in appendix 1.

From early in these designs our team unanimously agreed that an MDI (multiple-document interface) style GUI would lead to a good user experience. Eventually we settled on a Tabbed style MDI where each automata or string testing window would be in its own tab with the tab controller along the top of the application window.

Error messages and Input Validation

For invalid regex strings such as "*" or where there are unbalanced brackets we display an error message prompting the user to fix the regex string so that an FSA can be generated. This is described in more detail in the testing documentation.



Help and About Pages

We decided to implement a help page with details of how to use the program. Our program's aim is to be a teaching aid for students learning about converting Regex to FSAs, so whilst we expect our users to have a basic understanding of Regular Expressions and Automata we decided to place hyperlinks in this help page to descriptions of Regex and FSAs for reference.

The help page is built as a tab with a basic web browser (from JavaFX) which loads a local html file (and images) from the packaged JAR resources. The decision to use a html page instead of creating the help page in Java allowed us to have these hyperlinks in the tab and open up reference websites inside our application, instead of the user having to open an external web browser.

Software Engineering and Risk Management

Overview

Our task was to select an existing algorithm and build a system that visualises the algorithm. We had to have a working prototype by week 6, and finish our whole project by week 11. For such a big project as this one it was evident that we had to use some sort of software engineering techniques to plan it and organise our work around one or more software development models. Since we are all university students, we needed a software development process with a flexible schedule, because other deadlines could interfere with our progress.

Agile

We decided to follow the agile software development model, because it fits our project best. Having a working application at the end of each iteration assures us that

even if we do not implement every planned functionality, we would still have a fully functioning prototype, and an application at the end. Moreover, it has a flexible schedule, so we can have shorter or longer sprints based on how much time we have. Another benefit of choosing this model is that it enables us to adapt to any change in the requirements document at any time, making it easy to adapt to any feedback we receive.

We split the whole project into small independent functionalities (product backlog) and we estimated how long it was going to take to implement each functionality (on the Gantt chart). We prioritised them based on how important they were and how fast we could implement them. We had scrum meetings every other day (every day during the last week), in which we discussed new ideas and selected the functionalities (sprint backlog) that we were going to implement during our next sprint. Usually we had sprints of 1-4 days, depending how complex the functionalities were. We always agreed on a deadline for our sprints in order to make sure we would all manage to complete our work. At each sprint we designed and implemented functionalities as well as tested the old and the newly implemented ones to make sure everything worked fine together.

Model-View-Controller

We structured our project based on the model–view–controller (MVC) software architectural pattern: we split our team into two groups, one working on the GUI/Visualisation part (view and controller) and the other group implementing the simulation of the algorithm (model). Those two groups worked mostly independently, and they worked together only when we integrated the different parts of our project. Even though we worked independently, we attended the meetings together, and the communication between the groups was really good, which made the integration of the different aspects relatively easy. We discussed how we would integrate different functionalities before we even started working on them so that both groups could plan ahead and implement them accordingly.

Pair programming

Every time we implemented a more complicated functionality, we worked in pairs to make the implementation more optimal, but also to reduce risks (reduce the bus factor). This guarantees that for each functionality there are always at least two members from our team who have full understanding of the code. This assures us that even if one of our members (for any reason) cannot work on the team project anymore, the other members would be able to continue the project and use and edit any of the already implemented functionalities.

Risks

Extended from the specification risk assessment, we are presented with the following list, scaled where:

- Probability: 1 (low likelihood) to 5 (very likely)
- Effect: 1 (minimal effect/disruption) to 5 (large effect/disruption)

Risk	Probability	Effect	Strategy Type	Strategy
Computer malfunctions	2	1	Avoidance plan	Ensure each team member can quickly switch from using their personal machine to a lab machine and vice-versa so if one system fails the team member can continue to work.
SVN connection issues	5	3	Contingency plan	If a team member (or all) cannot access the SVN repository, the team shall share code via other media and continue to work until it is restored and the team can all re-synchronise together.
Illness / unavailability to work	3	4	Minimisation strategy	For each piece of code, at least 2 people should fully understand it, so work can continue. Also, ensure everything is committed to SVN so everyone can access it.
Compatibility issues on various operating	1	3	Avoidance plan	Use Java which is can be used on almost every system, and use a few external libraries as possible

systems				to minimise risk of incompatibility. We should build in the newest version of Java, but bear in mind some more advanced functions may not work in older versions of Java.
Underestimation of length of various components	2	4	Impact reduction	Complete the project production schedule in order and only move on when a previous item is fully complete. Only commit working and tested code so that if time runs short then there is always something to show.
Greater difficulty implementing features in JavaFX	3	3	Impact reduction	Either find a sensible workaround to problem, or approach our target using a different method entirely using research to help with redirection.

Evaluation

In order to evaluate our project, we compared the final project results against the requirements and features stated in the Software System Requirements.

The SRS document shows that our system's purpose was to develop a learning tool to help Computer Science students learn about Finite State Automata and understand the algorithms used in building a deterministic finite automaton and visualising each step of the construction. In section 2.2 of the SRS, the "Product Function" section states what functions the application must have. It included typing in a regular expression, checking if the input is valid or not, watching how the automaton is built, being able to pause as well as manually go through each step, print or save as an image, zoom in, zoom out and check for bisimulation. The functions we did not implement were minimising, zooming in and out and checking

for bisimulation. Apart from these, all of the proposed functions were fully implemented.

We planned to make our application user friendly for the target audience. We focused on adding HCI features such as creating a help page for the users. On the help page, the user can see a general description of the algorithms involved in building automata with links to further information about the algorithms, a tutorial on the buttons' functionality and how to switch from one stage to another.

As part of our testing, we chose to do User Testing which included giving our app to users which met our target audience requirements. We first had to reach the final GUI concept and ensure that most of our functional requirements were implemented. We designed a questionnaire to cover all the important aspects of our learning tool and analyse the results. Once the results of the questionnaire from the users were collected, we reviewed their suggestions in detail and grouped them based on their relevance to our application's functionality. We managed to receive feedback from students and two lecturers that are familiar with the automata theory. We even tested our colour blind mode on a colourblind person to make sure that it is properly visible to people that require special arrangements to use our application. The high priority suggestions have been fully implemented in order to meet potential users' requirements. Some of the suggestions we received and considered important immediately were as follows: adding a help page, adding a title for the current phase of the algorithm, adding a speed slider and showing what character the algorithm is searching for next in the automaton. Unfortunately, there were still other suggestions but we considered that they would just extend the current functionality of the program and would not make a serious impact on the user ability to understand the algorithms. Overall, the feedback was very positive and we could rest assured that our project would be suitable as a learning tool for students to learn more about the automata theory.

Since we were not allowed to use external libraries that can help with our graph representation, we have focused on not using any extensive GUI libraries except for JavaFX. We have followed this constraint and have solely used standard Java and JavaFX to complete our project. This made the project more difficult but the challenge was overcome by implementing our own graph representation.

All functional requirements have been met except for bisimilarity and minimisation. The reason why we could not implement these functions is because we did not take into consideration the time and complexity of building an automaton from a regular expression and testing if a word belongs to that automaton. Another issue was to calculate the position of each state so that they could maintain their positions at each step in order to make the whole automaton clearer to the user. This issue along with calculating the scale factor proved to be a serious challenge as the layout of the

states on the screen is an important factor in making the user understand the algorithms steps.

All the non-functional requirements have been met. The performance of the application enables the user to have a smooth experience with a quick response time to user interaction. In terms of reliability, the application builds the same automaton when the same regular expression is inputted and it always scales itself based on the window size. In terms of portability, there were no errors encountered as the application works correctly on all platforms. Availability and maintainability have also been fulfilled as proposed in the specification. The application is available for any user who has an up-to-date version of Java without any requirements. In terms of maintainability, our application's code is well structured and easy to read so that the system can adapt to new changes without affecting other features of the program.

We are convinced that our application came out as we planned it, without changing the idea of how our program would look like. The features of our program that we consider most impressive are the graphics and the quick computation of the frames of the animation. These tasks were successful because we commenced them early and we had time at the end of our project to improve on our design and the backend algorithms of constructing the automata. Other reasons why we consider we have done a great job are teamwork and our organisational skills. We managed to work on our strengths and divide work equally.

If we were to start this project again, the main thing that we would do would be to have a more realistic approach of the planning stage. We underestimated the amount of time some of the tasks require and we thought we could reach all of our goals for this application but a more accurate Gantt chart would have helped stay on track with our work. Furthermore, we would focus more on early user testing because it would help with our application design in advance.

In conclusion, we consider that the project can be evaluated as an overall success as we have met the healthy majority of the requirements set in the SRS document at the beginning of the project.

Team Work

General Organisation

The organisation of work in our team was one of the most important aspects. It was crucial to allocate pieces of work to different members appropriately and establish a certain pace of making progress. Without that, it would have been impossible to

make a certain amount of progress in time and coordinate the development of the project appropriately.

During our project, we agreed on having three meetings per week. That was enough to do a significant amount of work every week and not too much for us to fall behind with other coursework. At those meetings, we were discussing ideas that we came up with and problems that we encountered as well as assigning new tasks to people, merging our code and making sure it was compatible.

At the very beginning, we decided that it would be best if we split into two teams. It would have been quite difficult to work on every task as six people and that would not have allowed us to make a significant amount of progress. One team was dealing with the GUI interface and the other one with implementing the algorithms. On the one hand, that was very useful in keeping one thing separate from the other, but on the other hand it was hard to keep everything compatible. Nevertheless, we took measures to maintain all the components coherent and succeeded in doing so. Both teams were constantly communicating with each other, exchanging ideas and asking what the other side of the project looked like. Everything was being adjusted continuously so that it matched the other team's code.

Our decisions about who was going to do which task within one team were based on the suggestions and the ideas that each member had and skills and good understanding of each of us. At the start, one of us suggested to use the JavaFX library and that person was appointed to start doing research on it and implementing it into our project. Later on, other people turned out to have an excellent understanding of one of our algorithms, therefore we decided that they would start its implementation.

We have been using pair programming. As it was a highly complex project, we would always make sure that no one programmed on their own to reduce the risk of various components being incompatible with each other. It also really helped us work as a team rather than individually and was beneficial in terms of sharing ideas and remembering about tasks that still needed to be done.

Problems, conflicts and solutions

The main issue that we encountered at the beginning was when we were to decide whether to use a top-down or bottom-up approach. The top-down approach creates one automaton which is constantly expanded, whereas the bottom-up one creates a number of small automata and joins them together. We had a lot of debate about both of them and we considered several pros and cons, but eventually, by means of consensus, we decided that the top-down approach would be better as it is clearer and more readable to the user. We thought that it would be easier to focus on one

automaton and its expansion instead of having a number of them scattered around the screen.

There were also a number of other minor problems, such as different methods having to be adjusted to be more compatible and universal. For example, we created a method determining the coordinates of a NFA on the canvas and later in the project we discovered that we needed the same method for a DFA. We did not predict that at the beginning, therefore the previously mentioned method had to be reimplemented.

In general, we have managed to overcome every difficulty and none of them set us back to a significant extent.

Communication

It was crucial to keep in touch with everyone at all times so that everyone knew what was happening with our project, what needed to be done and when we were going to have next meetings. There were a number of tools which turned out to be extremely useful for us in efficient communication.

Google Drive

Throughout the design and the development, all of our documentation including designs, plans and reports were stored on Google Drive. The documents could be edited by more than one person so everyone could see the changes made by others in real time.

Facebook

We were also using Facebook chat where almost three thousand messages have been sent in our group conversation. It really helped us get through to each other at any time and all important issues, such as new ideas, problems or planning future meetings, were always discussed there. Communication would have been much harder without it; thanks to that tool, we practically never fell out of touch with each other.

Trello

Another tool that we found helpful was the Trello board. We kept a number of lists of the tasks that had to be completed and the issues that had to be resolved there. It really facilitated the process of keeping track of the progress and reminding each other of the things that had not been done yet by the use of relevant notes on the tasks.

Skype

Sometimes we were not able to meet as some of us were away. An extremely useful tool in such situations turned out to be Skype. By means of it, we were able to have a discussion face to face without having to go out which was very convenient.

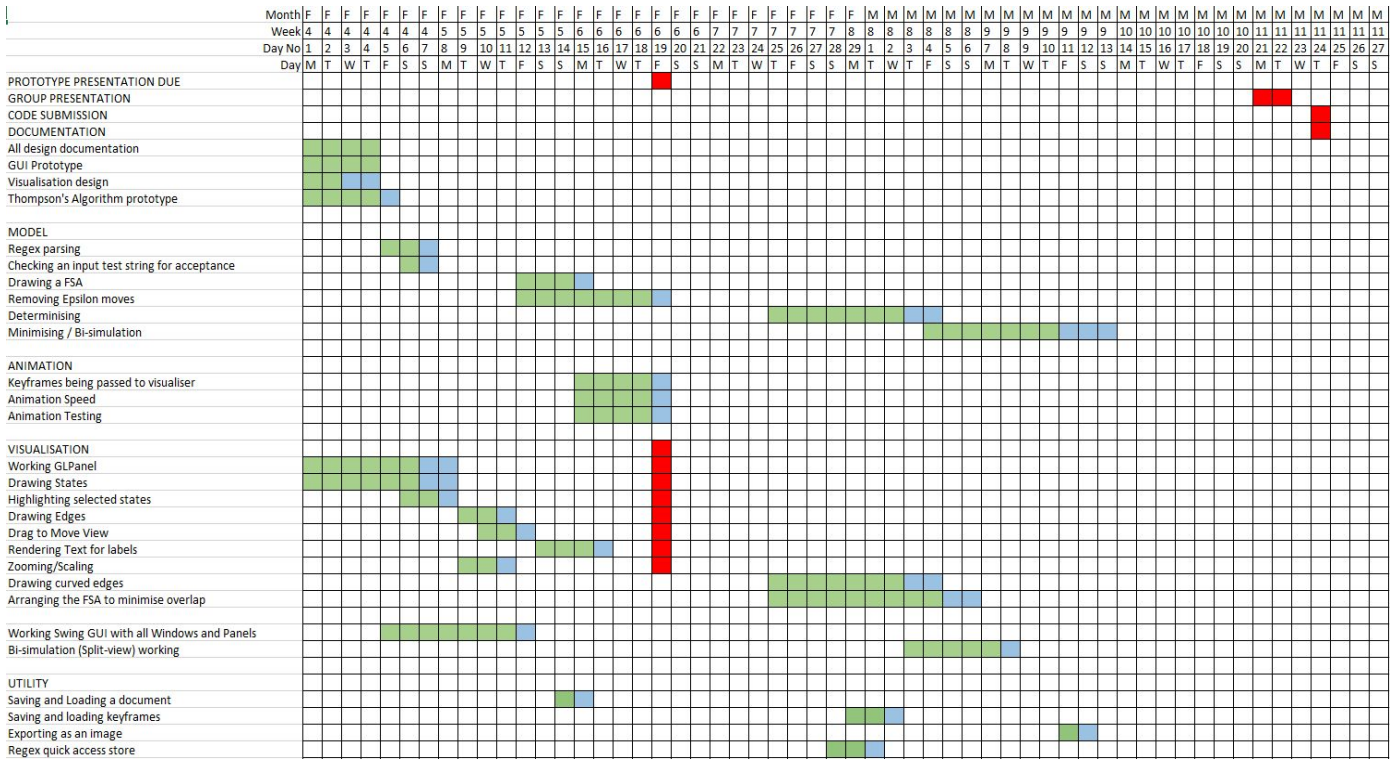
Gantt Chart

We designed a Gantt Chart and planned all the work with one day precision. It was extremely useful as an overview of all the tasks that needed to be dealt with and all the stages of the development of our application. However, it turned out that not everything was able to be done according to the plan. That was because some of the components required more time than we expected at the beginning. There were certain problems that occurred later and that we were not aware of before. There was also other coursework that we had to keep up with and sometimes we did not get as much time to devote to our project as we planned.

Nevertheless, thanks to a few mini scrum meetings that we organised, we always avoided falling behind with the project and managed to get the tasks done in time. At those meetings, we would spend a lot of time on intensive programming and we would not finish until the desired feature had been implemented.



















Our Gantt chart can be seen below:

Team Project - Team C5



SVN

We have been using SVN where our code was stored at all times. We were constantly updating all the changes to it so that each team member could download the latest version in any place at any time. That really helped us keep the whole project compatible and neat. A fragment of our SVN log can be seen below:

Revision		Actions		Author	Date	Message
257				bxm402	15 March 2016 19:56:21	Making NFA more user-friendly.
255				omp4...	15 March 2016 18:49:32	Fixed the regex frame slider values aft...
254				omp4...	15 March 2016 18:36:17	Changed the initial background to black...
253				tbc452	15 March 2016 18:34:44	changed tset string text
252				omp4...	15 March 2016 18:25:02	removed the test string class
251				omp4...	15 March 2016 18:23:21	Wrapped the drawDocument routine fo...
250				omp4...	15 March 2016 18:19:54	Wrapped the drawDocument routine in...
249				tbc452	15 March 2016 17:29:18	fixed text in dfa
248				dx459	15 March 2016 17:25:44	changed Automaton
247				omp4...	15 March 2016 17:20:00	Reverted automaton to previous versio...
246				omp4...	15 March 2016 16:53:54	added EdgeHighlights class
245				tbc452	15 March 2016 16:51:32	updated my stuff
244				omp4...	15 March 2016 16:10:31	code for Tom to fix (VisualState label)
243				omp4...	15 March 2016 16:07:43	Completed TestString and Edge Highlig...
242				tbc452	15 March 2016 15:33:14	dfa states now show what nfa states t...
241				tbc452	15 March 2016 15:14:40	working on better string on states just

Summary

Based on the massively positive user feedback and the overall test results, it can be concluded that the software produced meets its requirements when it comes to comprehensively visualising a regular expression to a NFA/DFA transformation. We believe that our program can be used to complement the automata theory related courses due to its ability to quickly create and neatly display various automata of a certain type, or as a standalone tool for learning Thompson's and powerset construction algorithms. The visualisation, along with the descriptions provided in the "Help" section of the application, should be sufficient for a user familiar with automata in general, but not with the concrete algorithms, to gain working knowledge of those algorithms.

We believe that the workflow was well-structured and organised throughout the project with all team members contributing and providing valuable input. The design decisions however, could have been more thoroughly thought-out in the early stages of the development, as the team ended up making some significant design adjustments later on. Although one of the most significant early design decisions (a top-down approach to visualisation instead of a bottom-up one) defined how the entire application functions, we believe it to be correct as it allowed us to gradually incorporate changes into an automaton while preserving and building up on the early established general structure. The choice of an agile development paradigm also proved to be a good one, since it helped us to stay flexible and quickly introduce decisive changes while keeping everyone up to date on the current state of the project.

Another thing that could admittedly have been done better is incorporating automated unit testing more deeply into the development as we repeatedly reintroduced already fixed mistakes into the code by accident on a number of occasions, mostly through SVN.

Individual Summaries

Thomas Clarke

Throughout the course of team project, I held an important and authoritative position. I was mainly teamed with Owen to produce the GUI and visualisation, of course while working with the whole team to make sure the different components actually fit together well. Myself and Owen put many hours into developing the visualisation and iterating upon it until it looks as we wanted it to, along with a user friendly GUI with a host of easily accessible features.

A big part of the experience for myself was being a key member of organisation and in particular managing team meetings and trying to drive the content of conversation at them. It was important from creating the original specification to submitting the final product that our group's ideas were synchronised and effectively used. For instance, I suggested and set up the Trello board we used to help keep track of current tasks and notes associated with them, so whoever wanted to work whenever could access as much information as possible (along with all other documentation hosted on Google Drive).

In terms of programming, I suggested the use of JavaFX as a good platform. Despite all of us being inexperienced with the library, I created the first mock-up of the GUI (while Owen created the first drawing of the visualisation) within a few days of the suggestion to prove its potential. I then went on to program many features of the GUI and visualisation, including (but not limited to) the playback controls, handling multiple animations and the file menu (save, open, print and close). I also found and worked with the colour blind user to help develop 'colour blind mode' to help the software be as accessible to as many people as possible.

I also worked with other team members to help smooth the connection between the backend and the frontend, namely the Animation class and the Document interface. Towards the end, I also did a lot of work in refactoring the project to make the GUI as modular as possible with as low dependencies as possible, so that features (such as the side bar with DFA) were easier to add, and providing appropriate JavaDoc throughout all GUI classes.

I feel as though I worked appropriately hard for the project, not over contributing and keeping to deadlines. I enjoyed learning JavaFX and working in this team environment to produce this large piece of software.

Mihnea Patentasu

I believe that working on the team project has been an overall enjoyable experience. My main role within the team was a backend developer of our application. I worked on implementing Thompson's algorithm that is being visualised by the program, but I also worked on the Testing Strategy by creating unit tests for our base classes.

I am confident that I have learnt a lot from this module, both academically and as a person. On the academic side, I learnt more about different software engineering methods and how to work with SVN. I have understood that you cannot just start coding, software projects require planning and documentation. My coding skills have improved, as well as my research skills. Our chosen class of algorithms was not implemented in Java so we had to research and analyse which data structure would be most suitable for our algorithms in order to reduce the complexity. Furthermore, I have also understood the importance of refactoring your code so that your team members will not struggle to understand it. Another useful technique I have learnt was testing which helped us find bugs in our code easily whenever someone had made changes to it. Using Trello has proved to be a useful way of organising and grouping all of our tasks. I am sure that these skills will help me further in my studies and even in the industry.

Working in a team that I did not choose was an interesting experience but I had the luck of finding myself in a skilful team. At first, it was a bit difficult as we had to adapt to the different styles of programming and we were not used to working with each other. We did not know our strengths or weaknesses. Notwithstanding, that changed after a few weeks when we decided on our application's design. Therefore, we quickly went from forming to storming. I have also learnt how to work effectively within a team to achieve a common goal before a deadline. I enjoyed employing pair programming because we could share ideas easier and work more efficiently.

If I could repeat the whole project, I would probably have done more tests before coding and tried to do more pair programming as I believe it generates better, safer code and sometimes you do more progress when there is someone from your team pushing you and working alongside you.

In conclusion, I believe that my team and I have worked hard to produce an innovative practical software that can help students understand the automata theory. I think that we should be proud of our product considering the short amount of time in which we managed to complete most of our initial project ideas.

Piotr Wilczyński

Working on a project with team C5 was an absolutely amazing and unforgettable experience. I have really enjoyed it and I am extremely proud of everyone's attitude and contribution. The fact that I have improved a number of skills, such as effective communication, planning and creativity, will undoubtedly be very beneficial for me in the future.

During the development of the project, I was part of the team implementing the algorithms. I contributed significantly to the initial discussions about what the interface should look like and how certain objects should be represented in the code. I was one of the people who implemented the method for string testing. I wrote a lot of unit testing to always make sure that the whole project was still working. Moreover, I helped with designing the questionnaire and collected feedback from several people who were also studying the module related to the automata theory.

However, my main role in the team was to coordinate work and keep track of what needed to be done. I was always making sure that everyone had something to do and trying to balance workload. A significant number of components, such as implementing classes, determinisation and testing strings, were kicked off by me. My task was to monitor the amount of progress that we were making and to ensure that we were not falling behind. I was the one to reflect on what had been done so far and remind everyone about the things that had not been dealt with yet.

One of the most significant aspects that I brought to everyone's attention was error handling. I was aware that it was very important for the application to deal with incorrect inputs appropriately. I implemented a method which checks if a given regular expression is well-formed and well-bracketed. I also suggested to the GUI team that they showed an error message when a wrong regular expression had been entered.

Apart from that, I made a significant contribution to refactoring. I was always watching the whole code, reviewing all the changes and trying to keep the project clear, neat and concise.

To sum up, I am extremely happy that I had a chance to be a member of team C5 and I think we have managed to produce a decent piece of software. We devoted a huge amount of time to make it an outstanding learning tool for students and we are pleased with our results. It has been a great pleasure to be part of this project and I think everyone had benefitted from it to the fullest.

Danyil Ilchenko

I was a part of the model team working on the implementation of the algorithms. I would consider my influence on the theoretic part of the project to be quite significant. The visualising strategy for both NFA and DFA was suggested by me and I worked hard on making sure the backend of the application runs smoothly and does not prevent those strategies from coming to life. Namely I implemented the routine that parses regular expressions into construction blocks used by Thompson's construction and as well as powerset construction algorithm. A lot of thought went into optimizing the latter in order to account for the overheads caused by the necessity to animate the algorithm's execution at each step. In general, my input is noticeable in most backend classes be it in the design or the implementation. I also occasionally collaborated with the GUI (view and control) team in order to devise the most effective way of transferring information about the automata to the graphical interface and drawing modules.

I believe that working on this project was an invaluable experience. Among other things it taught me just how powerful a team of developers can be: having separate parts of the application being worked on simultaneously creates a unique synergy that allows new concepts to be tested and either realised or discarded a lot quicker. And if a certain process in the development goes particularly slow or comes to a halt it is always possible to pull in more resources. Working alone does not provide such flexibility. Teamwork also implies a variety of different perspectives being merged together which ultimately leads to a more well-rounded product.

As a result of working on this project I learnt a bit about JavaFX which I never used before. I also mastered SVN which is now available to me as a version control system alternative to GIT. My knowledge of automata theory was expanded too.

If I could have had this experience again, I would insist on more careful consideration of design. In hindsight it seems to me that we sometimes rushed and started coding too early, which lead to increased workload in the later stages of the project.

Owen Pemberton

Having previously completed a visualisation project a few years ago, I contributed heavily at the beginning of the project, specifically when creating the Gantt chart as I had knowledge of the various components we would need and how long they would take to develop. Since I also had experience in Swing I put myself forward to work on the GUI and visualisation alongside Tom. Whilst our team decided on JavaFX over Swing after Tom developed a prototype, both technologies are very similar so I felt I still had valuable skills that would leave me best placed in the group to work in the GUI and visualisation. I also felt strongly regarding HCI and the importance of the usability of the GUI so I pushed for our team to use a MDI tabbed user interface which I then designed and worked. I believe the MDI interface results in a great user experience in the final product.

Very early into development I realised there were problems with JavaFX, most notably that the Canvas object that we intended to use for drawing on doesn't allow for resizing so I worked on writing a wrapper for this node so that it would receive events to update and redraw the canvas whenever its parent control/container resized. An early version of this involved our canvas object holding a reference to its parent container then adding listeners to the width and height properties on construction. This wasn't a clean and self-contained implementation so I spent some time later in the project on this problem, but I believe the final solution in which the container class implements abstract invalidation methods of our canvas object works effectively, is reusable and is very neat without being overly complex.

Nearing the time the prototype was due, our team had worked on getting most of the GUI layout and model side of the project working and we had overlooked the graph layout implementation. Whilst at first glance this isn't the most important part of the visualisation, it is crucial to getting the graph to display on the screen neatly. Since we didn't have much time, instead of implementing our own algorithm I used the JUNG graph library to calculate possible coordinates for our graph nodes. I created the VisualRepresentation, VisualState and VisualEdge classes that would store a basic representation of the automata as a graph. Using JUNG wasn't ideal as it produced graphs that would move positions between frames, but it ensured that we had a clear visual automata representation for the prototype presentation. After the prototype presentation we removed the library and used our own graph layout algorithm (developed by Botond).

As the project progressed I worked with Tom, fixing various bugs, ensuring that any new code written in the model worked with the GUI and implementing new features from the user feedback such as edge highlighting. I spent a lot of time near the end of the project refactoring and commenting any code that needed it.

Botond Megyesfalvi

It was a really good experience to be a member of Team C5. I am very pleased with my team members, they are all good programmers, and we all had a really positive attitude. We were always aiming for the best, trying to find the best solution for every aspect of our project, and being so perfectionist, we pushed ourselves to become even better.

I believe I made a really significant contribution to the team. I worked on the model component, but also helped planning the integration of our two components. I implemented the Thompson's construction that builds up a nondeterministic finite automaton from a conblock and worked on most of the functionalities of our model.

I thought of many ideas about how we could improve our project and found solutions for some of the occurring problems. It was my idea to visualise the process of checking whether a string is accepted by a regular expression or not. I came up with an optimal solution for the integration of the model and visualisation, by creating a visual representation of the graph and passing that to the visualisation team instead of creating a copy of the graph. This made their work a lot easier because they could visualise it without having to traverse the graph of the automaton every time. At halfway of our project we realised that we had to draw the automata ourselves, without using any external libraries to draw a graph, so I planned and implemented the methods that generate the coordinates for an NFA and a DFA as well. I think one of the hardest parts of our project was planning how to represent certain objects, and in my opinion, I came up with some good ideas that helped the team move forward and also made future functionalities easy to implement.

I also helped the team by getting some useful feedback. I and Owen presented our application to Dr Steve Vickers and Dr Paul Levy and asked for their professional feedback. I also helped Mihnea and Piotr to get feedback from students.

Overall, I had a really positive experience working in team C5. I think I gained a lot of experience in programming, but also just in working as a team. We all learned about different software engineering techniques last year, but actually using them for such a huge project has proven extremely useful. The project also taught me the importance of commenting our code properly. Good commenting made it possible to work individually or in smaller groups because we could understand (at least how to use) every part of the project, and we could add extra functionalities to classes that were made by others. I think this whole project was a really useful experience for all of us, and I am really proud of the application we managed to create as a team.

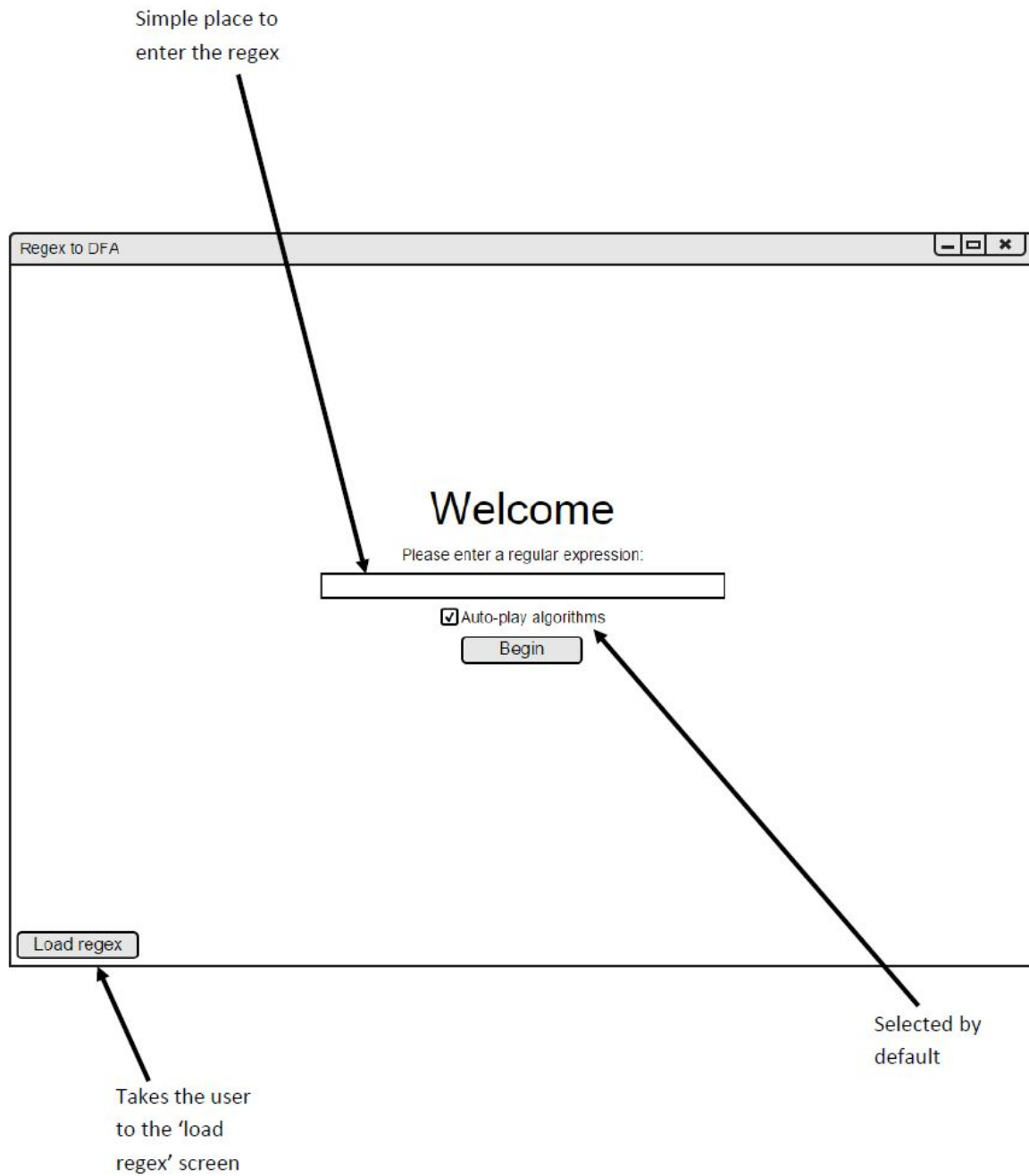
References

Ken Thompson (Jun 1968). "Programming Techniques: Regular expression search algorithm". *Communications of the ACM* **11** (6): 419–422.

Rabin M. O.; Scott D., (1959). "Finite automata and their decision problems". *IBM Journal of Research and Development* **3** (2): 114–125.

Appendices

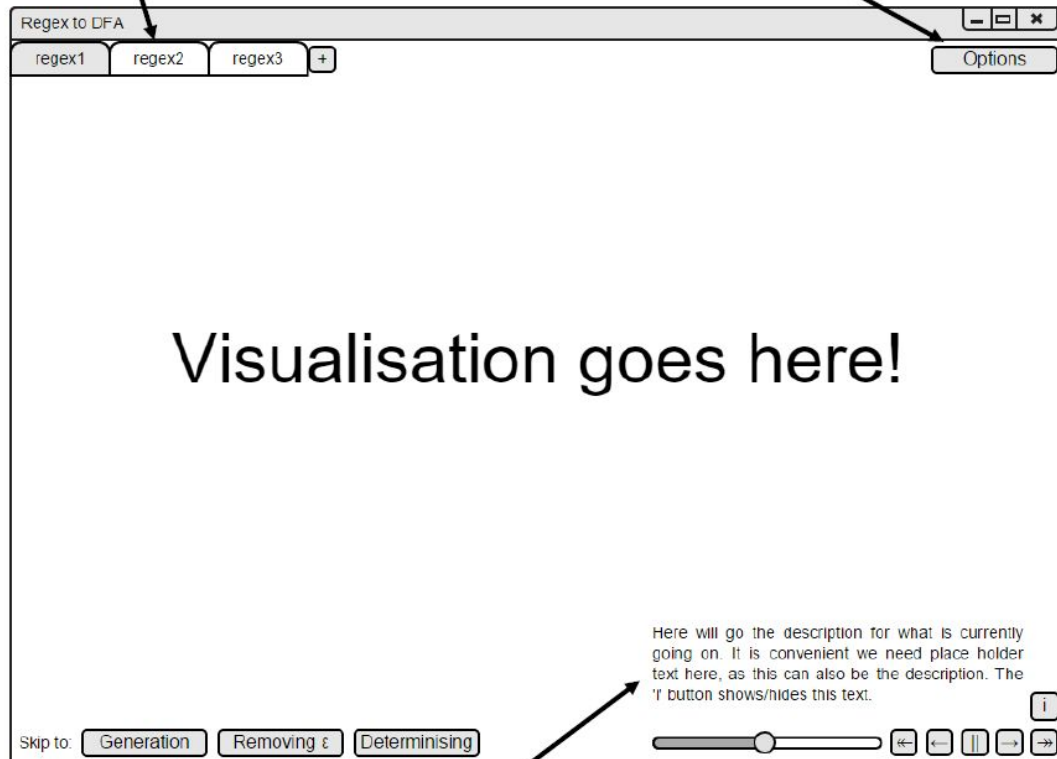
Appendix 1 - Early GUI Designs



Tabs for all the regexs (dynamically adds tabs when needed. The selected one is a different shade to the others. The plus button is a quick way to add a new one. 'New' tabs are the home screen (previous page).

Options button allows the user to:

- Close current regex
- Open a new one
- Save the current regex (opening the load/save screen)
- Allow them to test for bisimilarity (from regex in another tab).
- Allow them to print the current contents of the screen.



Allows the user to skip to a specific part of the visualisation.

The helper text for the current animation. Also with a hide button.

Navigation of the current section

- Slider indicating progress through current section.
- The buttons are (left to right) restart, rewind, play/pause, fast forward, skip to end

Load a regular expression	
1.	regex1
2.	regex2
3.	regex3
4.	(Empty slot)
5.	(Empty slot)
6.	(Empty slot)
7.	(Empty slot)
8.	(Empty slot)
9.	(Empty slot)
10.	(Empty slot)

Load Cancel

A list of all saved regular expressions. Empty slots are empty and not selectable. The same screen will be shown when saving, except the button at the bottom shall read save (and if saving, a prompt will be shown if the slot is already taken).

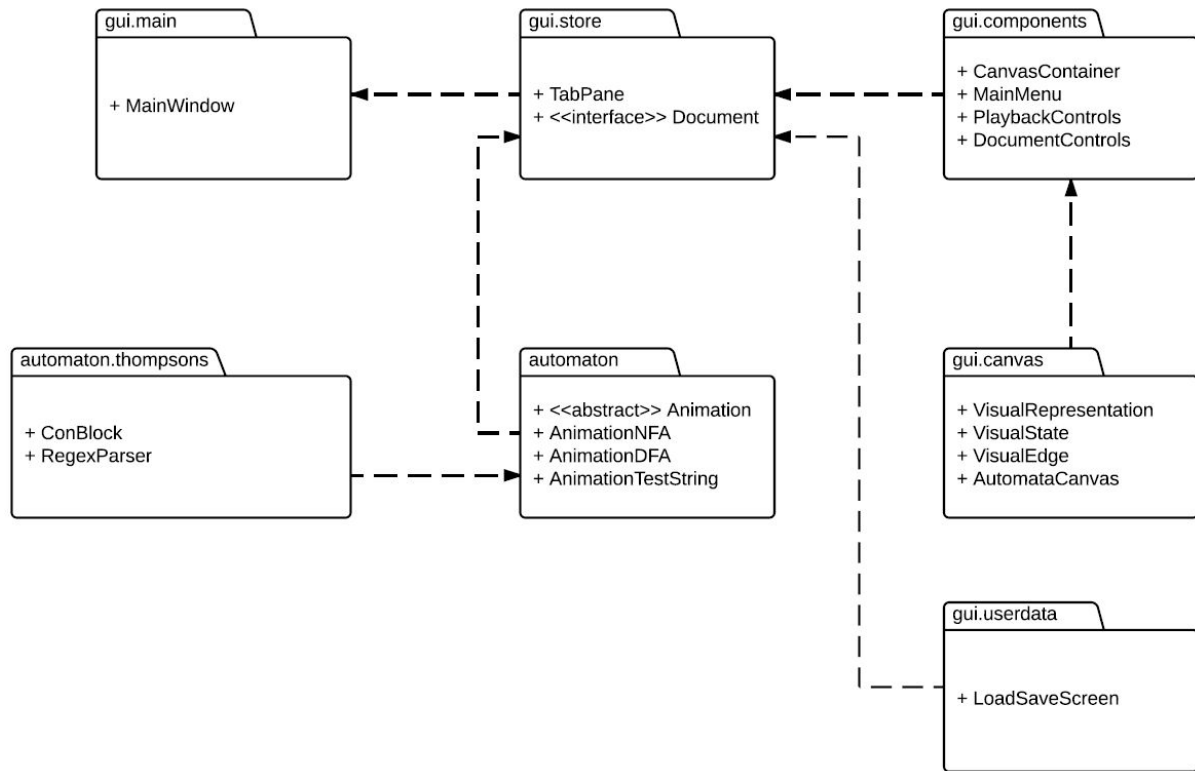
Load (or save) button shall load (or save to) the selected slot.

Cancel shall return the user to the previous screen.

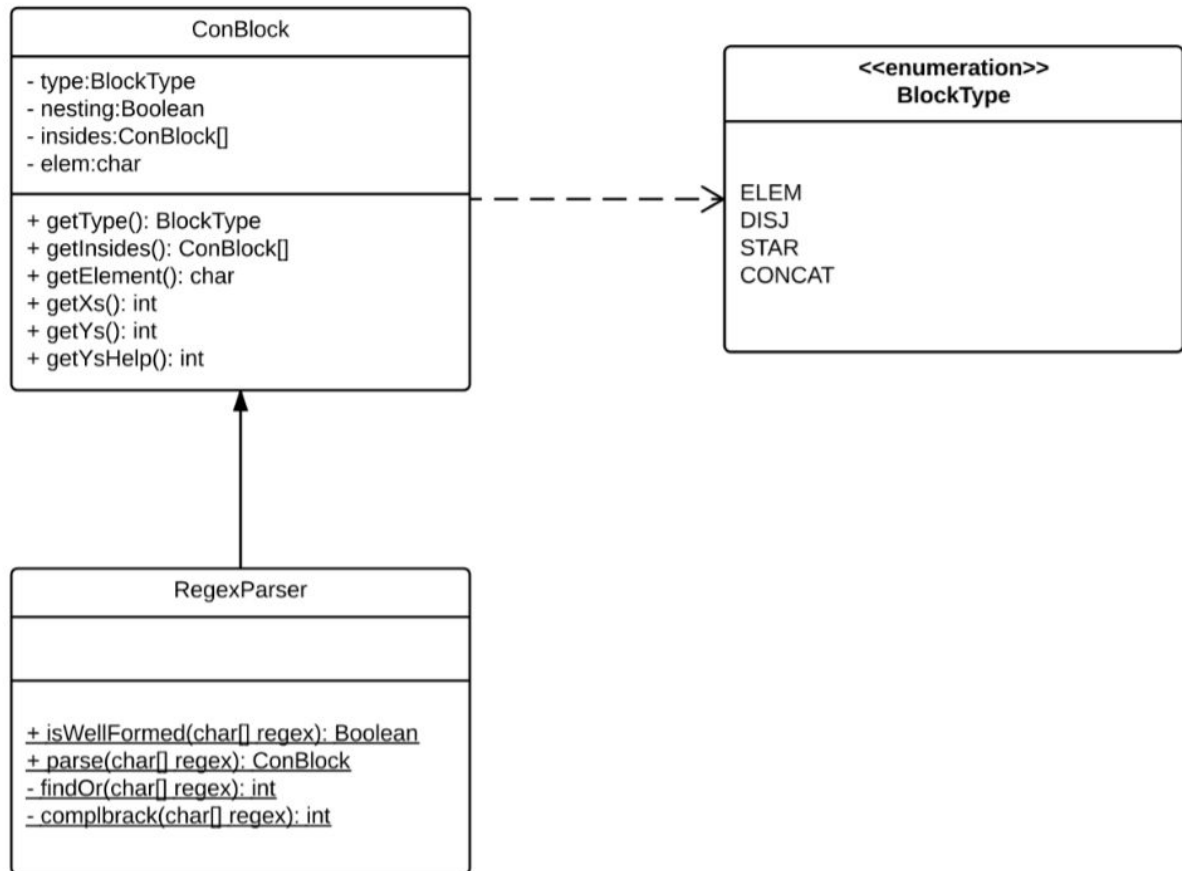
Appendix 2 - UML Class Diagrams and Package Diagram

*gui.main.MainWindow contains the starting method for the program and nothing else.

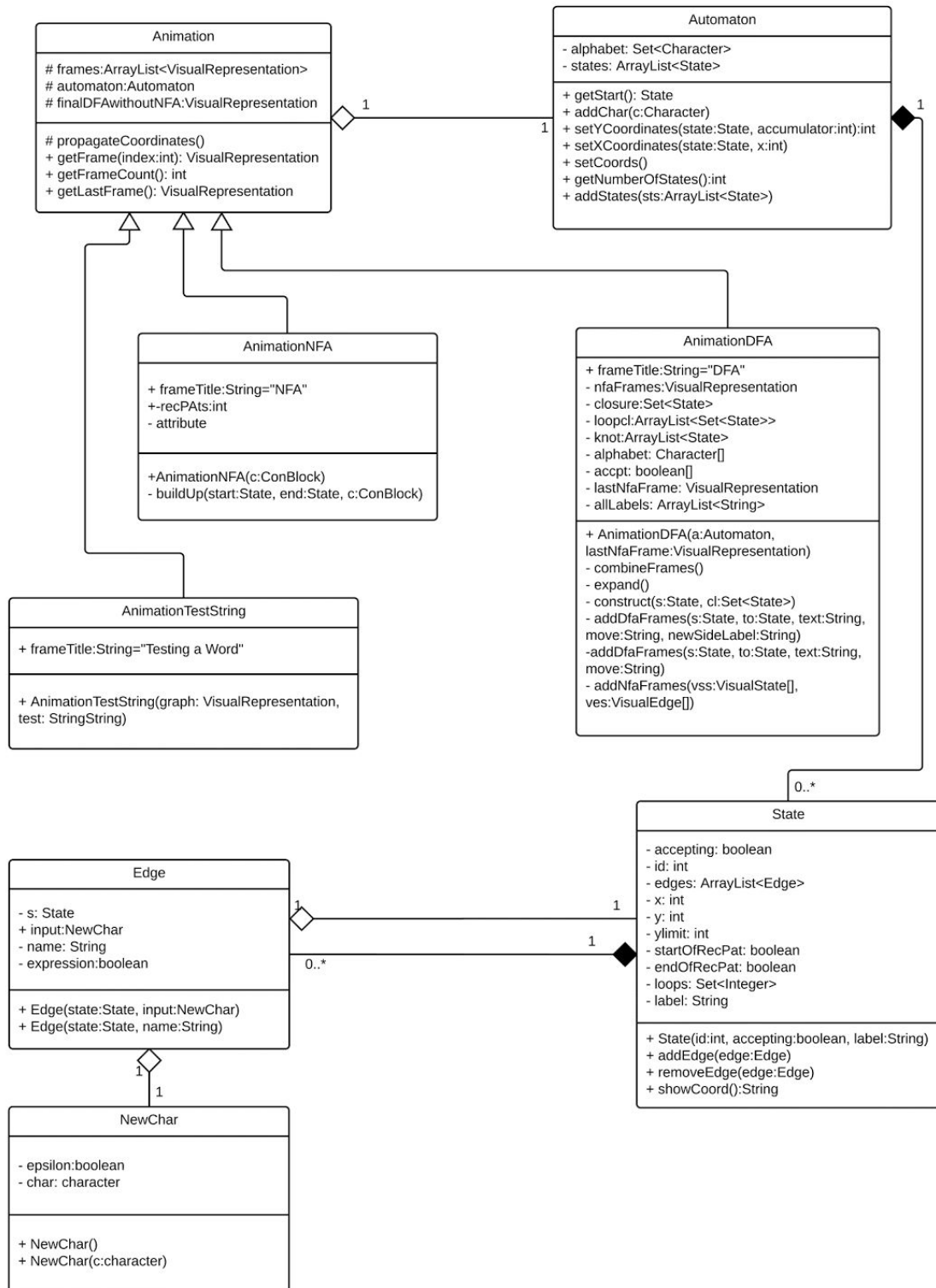
Packages



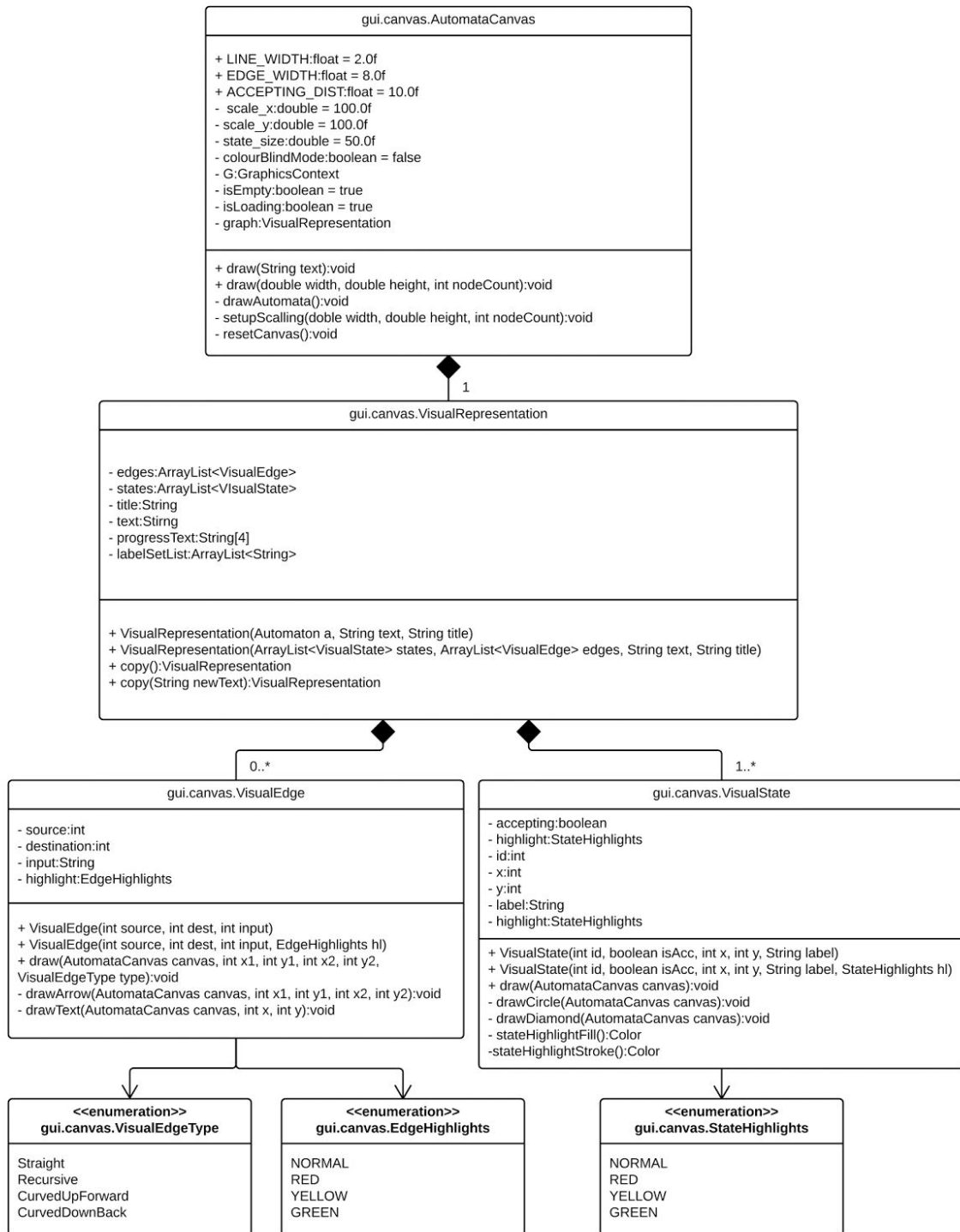
automaton.thompsons

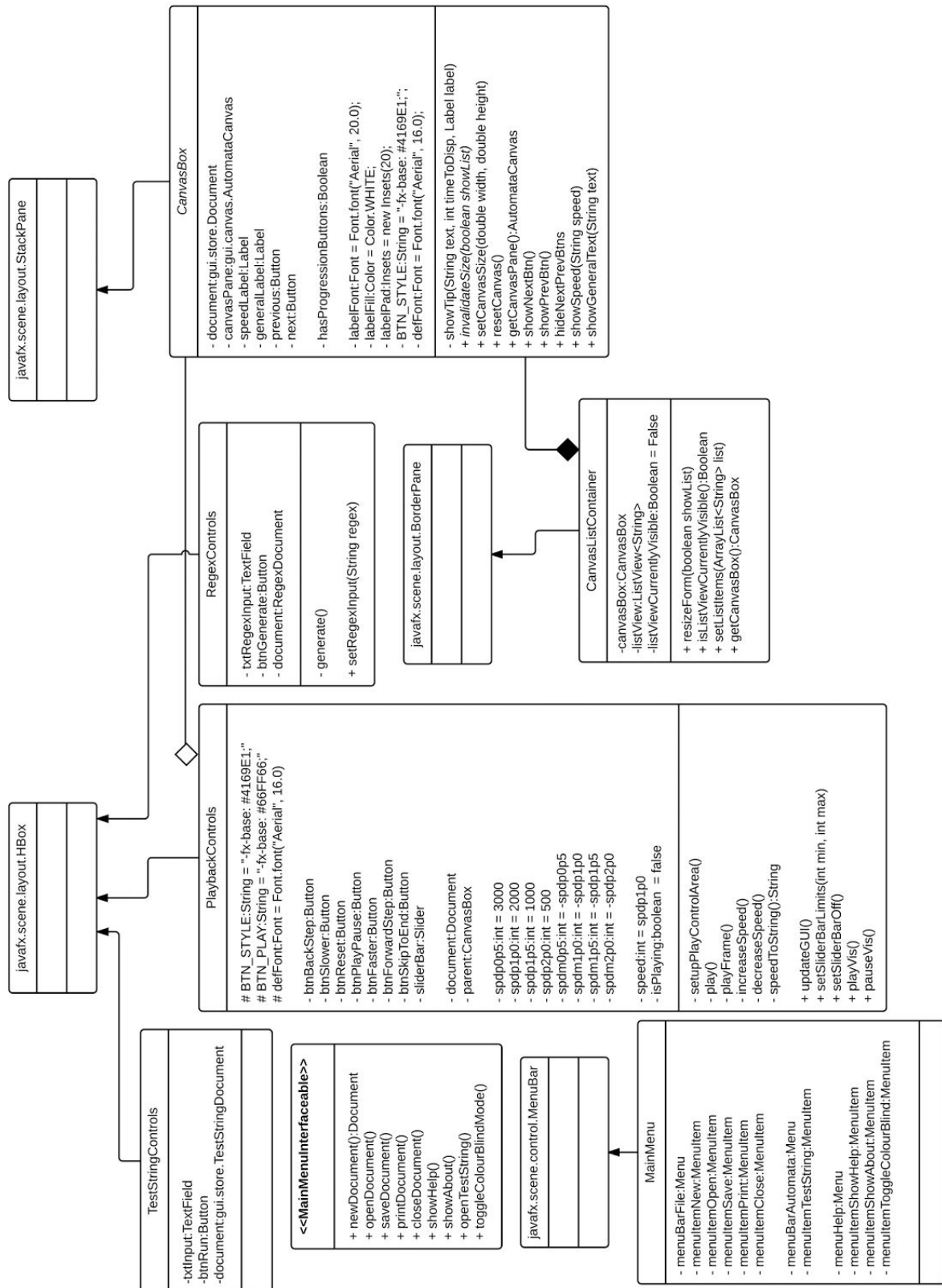


automaton

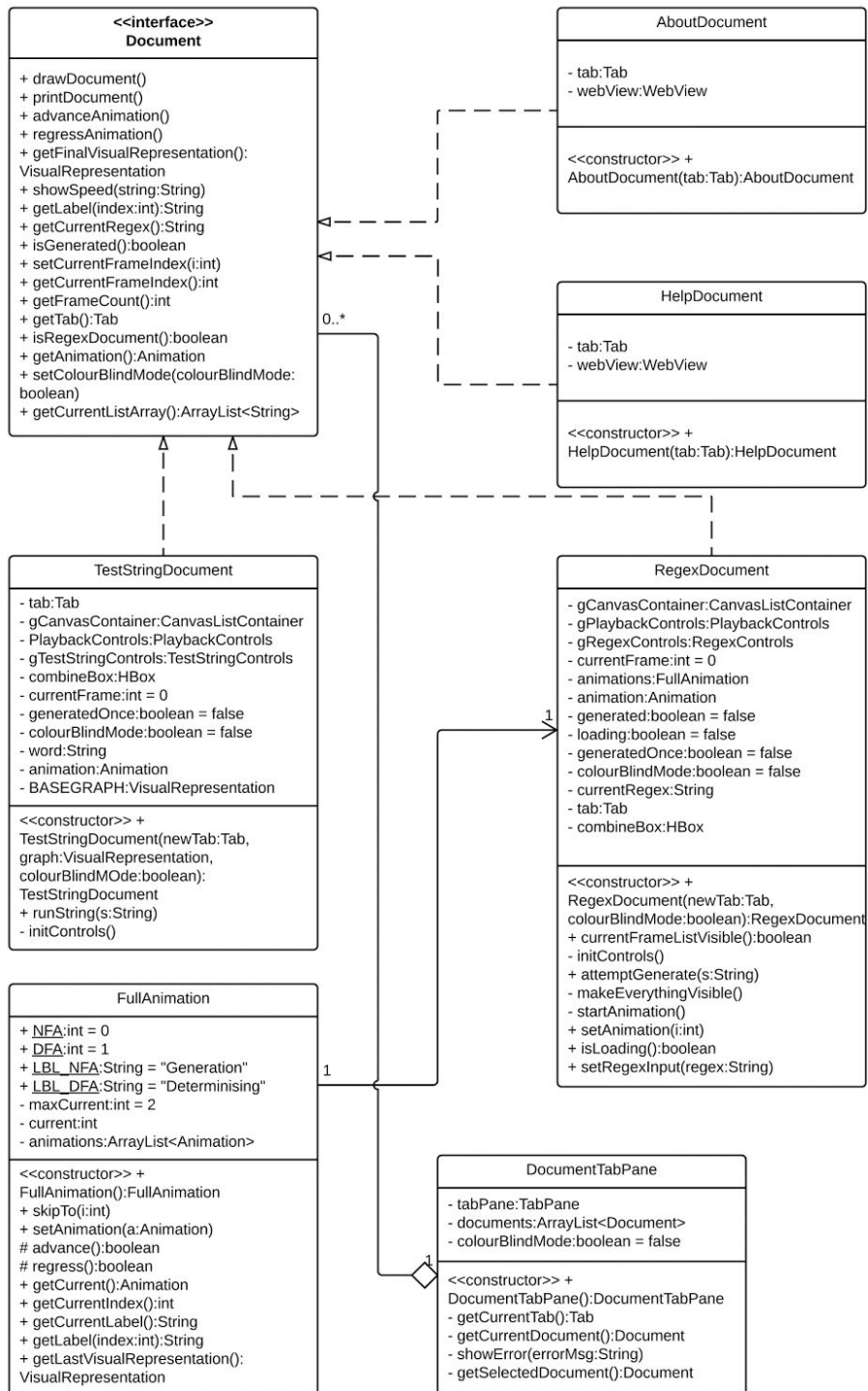


gui.canvas





gui.store



gui.userdata

