The Creation of a Serious Game to Teach Turing Machines

Kieran Warwick
Master of Computing in Computer Science with Honours

2019

Abstract

In this dissertation, the development process of producing the serious game "Interactive Turing Machines" will be described. The serious game is aimed to education players with no prior knowledge about Turing machines, while providing a logical puzzle game to the player to take advantage of the benefits of serious games, which can prevent the player from making errors, providing instant feedback on why it was an error.

Turing machines are a fundamental mathematical concept, which can be described without the use of mathematics through the use of state transition diagrams. This dissertation will add to that state transition diagram through the use of subroutines, which are Turing machines contained within a state. The topic of Turing machines is taught at A-level and undergraduate levels, so the game allows for levels to be created by users such as teachers to support education.

The serious game constructed by this project can be found at `https://interactive-turing.herokuapp.com/`

Contents

Chapter 1

Introduction

1.1 Problem Description

A Turing machine is a fundamental concept, used to formalise a computation by using a mathematical model. The concept is taught at an undergraduate level to mathematics and computer science students, and to A level students studying AQA computer science (AQA, 2019). It is currently taught to undergraduates by introducing them to the informal model of a Turing machine, and then formalising it into a mathematical model. The students can then create their own machines by constructing a transition function.

This is practical for trivial Turing machines, but as the complexity increases, the computation gets more complicated to follow as with every new state adds to the transition function, so the solution is to use a state transition diagram to represent the Turing machine, which visually presents the transition function. This approach helps to see the whole machine without the need for the mathematical knowledge about sets of states or how to make the transition function.

This works up to a point because as the Turing machine gets more complex it becomes harder to see the whole state of the system. The current method to represent a complex Turing machine such as this is to describe what actions the Turing machine performs at a higher level, for example, to place a marker or to move to a certain side of the word. The problem with this is that it is now it is harder to construct a formal Turing machine as it does not describe a real transition function. The goal is to put these higher level functions into the Turing machine's transition function and thus state transition diagram.

With the traditional method of learning, a mistake becomes apparent in the Turing machine when the input results in a incorrect output. This can be due to a logical error in the transition function, which can be hard to detect if the student cannot conceptualise the computation of the machine correctly. A solution to this problem can be formed by the creation of a serious game which, as a game can restrict the choices made by the player; ensure that they do not make any simple mistakes that could result in a logical error. This will be useful tool for a student in learning how to construct Turing machines without making unnecessary errors. A serious game can also help assist in the traditional methods by being used as a supplement resource to the pen and paper techniques.

The aim of this project is to construct a serious game to be used as a learning resource that will help students, with little or no prior mathematical knowledge, understand the

concept of a Turing machine. It is also to introduce higher level actions into the Turing machine to improve the understanding of how the Turing machine functions. The game should also be a teaching tool to support educational institutions.

1.2 Structure of dissertation

Literature and Technology Survey

This was conducted in order to better understand the serious game genre and how it can be used to teach the user about educational topics in an effective manner. It then explains how to design serious games and then show examples of such. It also provides background on Turing machines and how they are currently taught in education.

Requirements and analysis

This section takes ideas from the literature and technology survey and generates requirements for the serious game to meet for it to become a successful learning resource.

Design

The high level design on how to meet the requirements set are discussed, including the design approaches considered and why the approaches were chosen.

Detailed Design and Implementation

This now takes the design and discusses the software and techniques used to implement it into a serious game. The challenges of creating the components required for the serious game are considered in order to fulfil the design criteria and the requirements.

Testing

Tests were performed to determine if the requirements set out were met. This was done using unit testing, observed playthroughs and a questionnaire which were designed in this stage.

Results and Discussion

The results of the observed playthrough and the questionnaire are discussed to evaluate the quality of the application built for the purpose of a serious game.

Conclusion

A discussion of whether the aims and goals of this dissertation were meet.It include a self refection on the highlights of the project, the downfalls, and the what future work that could be done to improve the application.

Chapter 2

Literature and Technology Survey

A literature and technology Survey was performed to investigate what serious games are and how to use them for teaching students about Turing machines in the most effective manner. This involves learning how to design a game such that the player will want to continue to play while still maintaining an educational benefit.

2.1 Serious Games

A serious game can seem like an oxymoron, as most games are presented as an entertaining, enjoyable, and fun recreational activity. The word 'serious' however only describes the nature of the game, which is to educate and train the player. This does not mean that a serious game cannot be fun and entertaining, as fun is a subjective measure to the player (Michael and Chen, 2005). Abt describes a serious game in his book 'Serious Games' as:

> "Games may be played seriously or casually. We are concerned with serious games in the sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement. This does not mean that serious games are not, or should not be, entertaining"(Abt, 1987).

Digital serious games have been in use since 1951 and, emerging from the popularity of war games came America's Army, which was developed by the United States Army for recruiting (Wilkinson, 2016). When played casually outside the army it was an entertaining game, inside the army, it was re-purposed to train and test the players about intelligence skills, first aid, and survival training. The US Army's use for the game did not consider entertainment the primary goal of the game, but to educate the player sufficiently about the challenges of warfare (Michael and Chen, 2005). The advantages of using a serious game in this way is that it can simulate scenarios which are dangerous and costly to perform in the real world, without real world consequences.

Micheal and Chen state that the main point of serious games is to get the players to learn something while having fun, if possible (Michael and Chen, 2005). But the effectiveness of the learning process has been debated, so in order to assess this, a literature survey was conducted, using forty peer reviewed papers, written between 2002 to 2012. The literature reviews claim that mathematics integrates well into game-based learning, with seven out

of thirteen showing a positive learning effect; however five reviews showed no learning effect (Backlund and Hendrix, 2013). In a follow up review by All, Castellar and Looy (2016), it was concluded that a more standardised approach was needed in order to be able to improve the rigour of a serious game's effectiveness, and to define guidelines.

2.2 Serious Games for the Purpose of Education

To consider the use of serious games in education, the benefits they offer student should be considered, or whether they actually do make a difference, compared to the more traditional approach, and how to design a game which includes these benefits.

2.2.1 Benefits of Serious Games for Education

Engagement

Abt suggests that the study of mathematics can be enriched with the introduction of serious games, as a student can spend hours on card games that require mathematics, strategy or conceptualisation. It was noted that when games were played in the classroom, the students would compete to win the game and argue about the correctness of a move, and Abt recommends that this engagement should be exploited for educational gain (Abt, 1987)

Reinforcement of Learnt Studies

Serious games can reinforce skills and concepts that the student has mastered. Once a student is playing a game, it is unlikely they will quit or make uncalculated moves. The understanding of principles is also reinforced, not only by the student playing the game, but by what moves the opponent makes (Abt, 1987).

Simulation of Environments

Serious games can create an environment which simulates a laboratory situation, by proving the player with objectives and procedures. This can also allow creative freedom when it comes to finding solutions to the objectives, as it allows the student to experiment with the environment, while maintaining a realistic set of constraints (Abt, 1987).

2.2.2 Evaluating the effectiveness of gamification

Serious games and 'edutainment' are considered a form of gamification, and Kapp constructs an improved definition for gamification:

> "Gamification is using game-based mechanics, aesthetics and game thinking to engage people, motivate action, promote learning, and solve problems."(Kapp, 2012)

Serious games are different from edutainment games, which is education through entertainment, when serious games have the same objective as edutainment games, but teach more than just facts, and can include more aspects of education (Michael and Chen, 2005). Therefore the role of entertainment should not be ignored within serious games, as a fun game can increase the player's interest and also give the player more motivation to begin the learning process, which is useful when learning a new topic. Thus, serious games should be considered a viable tool for an entry point into learning a topic.(Iten and Petko, 2016)

2.2.3 Developing a Serious Game for Education

To construct a serious game for education, the following game characteristics defined by Charsky (2010) should be considered: competition and goals, rules, challenges, choices, and fantasy elements; these can influence motivation and facilitate learning.

Competition and goals

Most games have the objective of winning so the player has a competitive reason to keep playing the game. This is normally achieved by playing against an opponent, or completing the goal within a time limit. In edutainment games, the goal of the game matches the learning goals, with added competition to motive the player to complete the activities as the player wants to win, and keep on winning, which motivates them to complete more activities. Within serious games, the goal can be defined outside the win-lose conditions, allowing games to be longer and to get the player to complete more complex goals.

Rules

These are the constraints that limit what actions the player can perform. In edutainment games, the rules are strict and cannot be broken, which represents reality. The fixed rules help the player to practice a specific skill set, but this makes the game have the same experience every time it is played. Serious games have a more relaxed rules, permitting some breaking to let the player create new and unique solutions to problems, and test theories in the game. This means there is no "right" way to achieve the goal of the game, allowing the player be more creative.

Challenges

The games tasks and activities that almost all games provide, offer the player challenges. Serious games attempt to integrate the learning elements within the game itself, making it harder to distinguish between the fun and the learning. As the player completes challenges, the player gains skills which can lead to more complex levels, giving the player more opportunity to learn more skills and gain more knowledge. If the game can be viewed as an experience, then the player can reflect back on the game to relive the experience, this helps the player understand the knowledge more effectively.

Choices

There are a number of options and decisions a player has prior to and during the game. There are three types of choices: expressive, strategic, and tactical.

Expressive choices tend to have little effect on the learning of the player, but help to keep the player motivated as they become invested in the game. These choices can range from equipping vanity items to an in game character, to muting the game's sounds. These choices tend to be in role playing games, with the player developing empathy for their in-game character. This creates a level of immersion which helps the player keep motivated.

Strategic choices affect how the game is played, such as changing game attributes like difficulty or number of players. In serious games, the concept of 'levelling up' is applied, where the difficulty increases with each level, with the beginning levels teaching the mechanics of the game. This allows the player to learn the prerequisite skills within the game, so the player can apply them to the harder levels. Another approach is to let all the levels be available at the start. This promotes learning by failure, but it requires a form of feedback to the player on how they failed and clues about how to fix it.

Tactical choices refers to how the player chooses to play the game, such as accepting help. In the context of serious games, it is the method the player chooses to take that is key, as it results in various outcomes. If that method is successful, the player will choose the same method until it stops working as well. This is a trial and error approach, which can only work with integrated assistance.

Fantasy

Almost every game contains fantasy elements to attempt to motivate and excite the player. The fantasy can be used to reinforce correct behaviours or a response to a challenge, usually giving the player a reward. This is what edutainment games provide, and serious games should also use fantasy elements to help develop the players knowledge.

2.3 Serious Games for Computer Science

2.3.1 Scratch

Scratch is a visual educational programming language designed primarily for children, aged eight to sixteen. It lets the users create scratch projects, such as stories and games, without any previous programming experience (Maloney et al., 2010). It is used in schools to teach children how to program by letting them create a project that they are passionate about, with the view that passion can assist the child to push through challenges (Resnick et al., 2009).

Scratch uses blocks instead of writing code (see Figure 2.1), drawing from the idea of Lego bricks, as it allows the user to snap together blocks to create programs. The connectors on the programming blocks suggest how they are used, just like Lego bricks. There is no obscure syntax of traditional programming languages, which allows the user to tinker with the blocks to see that output, and is an example of instant feedback(Resnick et al.,

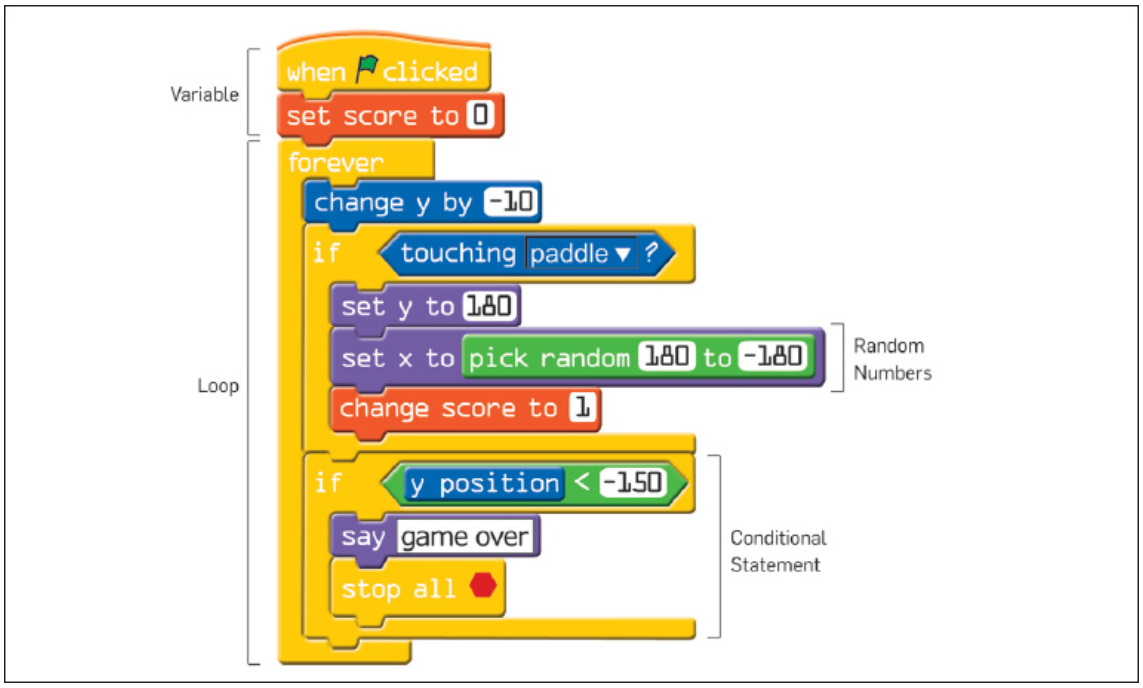

Figure 2.1: Scratch code for ping pong

2009). Also the use of blocks avoids complex notation. Overall, Scratch is an example of a successful learning resource, as some children took less than eight hours to learn how to use scratch and consequently as a result, learnt programming concepts (Wilson, Hainey and Connolly, 2012).

2.3.2 JFLAP (v7.1)

Java Formal Languages and Automata Package (JFLAP) is a interactive tool, created by Rodger and Finley (2006), to experiment with formal languages and includes Turing machines. To create a Turing machine using JFLAP, the user drags and drops states onto a canvas, and switches tools to create links between these states; one initial state and a number of final states must be made. This shows that the user is expected to have prior knowledge of how to construct a Turing machine, and will have to find documentation online on how to construct them. An example Turing machine can be seen in Figure 2.2.

This tool allows for states to represent writes and moves as states, which is an interesting model for the state transition diagram, allowing the transition to be split up into states. This could increase the understandability of the Turing machine that is constructed, allowing to see how the Turing machine performs a step in the computation. This idea could be extended the other way, allowing for Turing machines to be compressed into a single state.

2.3.3 TuringKara

TuringKara is a tool which AQA currently advertises as a resource to use to learn Turing machines. It is within the tool Kara, made by Reichert (2003). The tool teaches Turing machines by using a two dimensional tape, which allows for the Turing machine to move

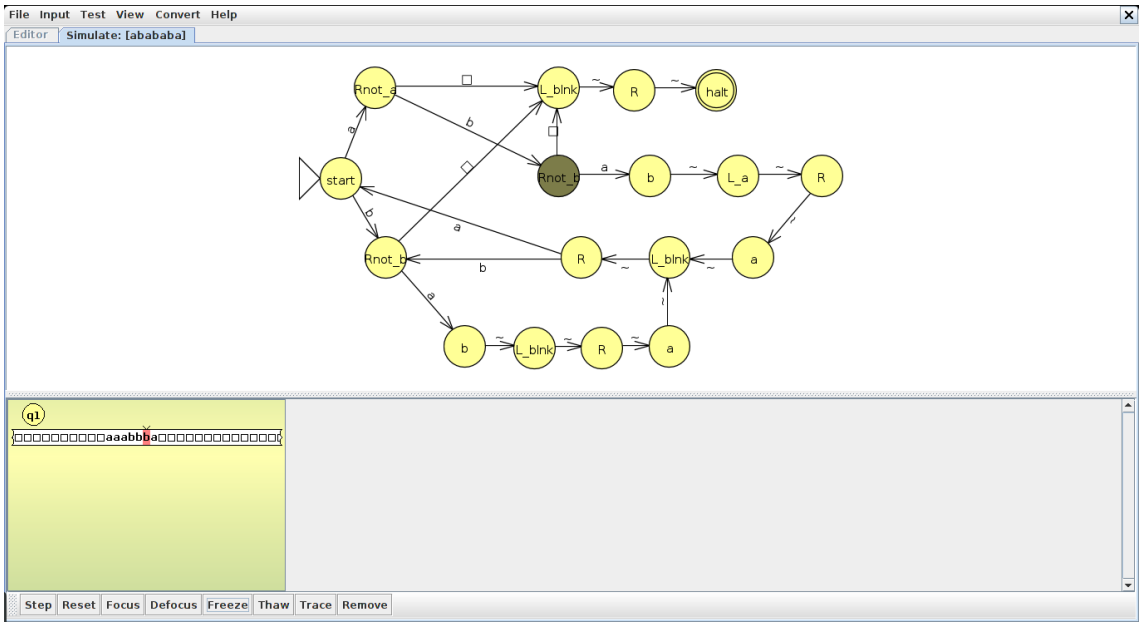

Figure 2.2: JFLAP

up and down on the tape. The user selects a state and can assign transitions to it. It also has a feature to assign a blank transition if it does not find a transition on the tape.

The whole environment can be seen in figure 2.3, with two windows - one for the Turing machine and one for the tape. This program is a useful tool to teach Turing machines, particularly the blank transition, which runs when no transition can be found in the transition table. This concept will be explored in the design as it reduces the number of states required for most Turing machines (Reichert, 2003) so will be useful in reducing complexity of the Turing machines

2.4 Turing machines

A Turing machine is a mathematical model of a computer. It can model the computing capability of a general-purpose computer. It has an infinitely long tape, containing cells, with each cell containing a symbol. There is a tape head that is positioned above one of the tapes cell; this cell is what the Turing machine scans. After this, the Turing machine will move once. It performs three actions it during a transition: changes state, writes a tape symbol, and moves the tape head.

The formal definition of a Turing machine is described by Hopcroft (2003), where a Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where

Q: A finite set of states

Σ: A finite set of input symbols

Γ: The complete set of tape symbols, so Σ is a subset of Γ

δ: The transition function, the arguments of $\delta(q, X)$ are a state q and a tape symbol X. The value of $\delta(q, X)$ is a triple (p, Y, D), where

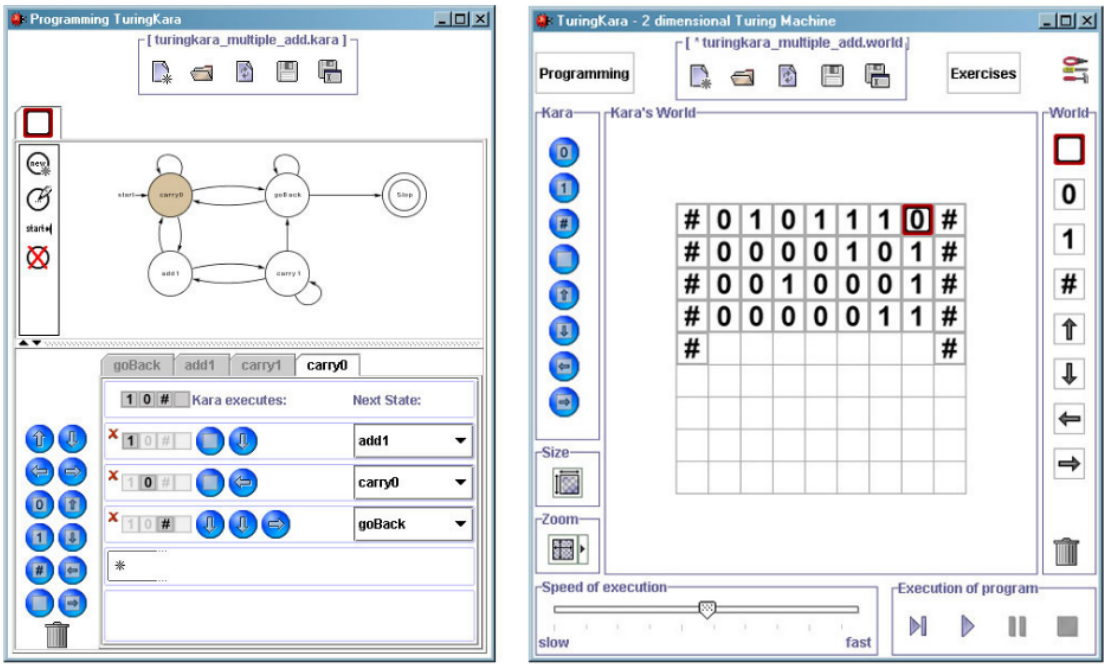

Figure 2.3: TuringKara's user interface

p: the next state, in Q

Y: the symbol, in Γ, written in the cell being scanned

D: a direction, L or R, to move the tape head

q_0: The initial state, a member of Q.

B: The blank symbol, which is in Γ but not Σ

F: The set of final states, a subset of Q

To describe the current configuration of the Turing machine, let $Q \times \Gamma^* \underline{\Gamma} \Gamma^*$ describe it. For example, a word *abababa* where the tape head is on the second symbol and in the state q, can be represented as $(q, a\underline{b}ababa)$ (Lewis and Papadimitriou, 1997). Finally, to describe a computation, it can be defined as a series of configurations which leads to a final state.

Turing machines can also be represented as a state transition diagram. This contains nodes which represent the states of the Turing machine. The arcs from states is labelled in the form $X/Y, D$, where X and Y are tape symbols and D is the direction to move the tape head. Figure 2.4 shows an example of a state transition diagram.

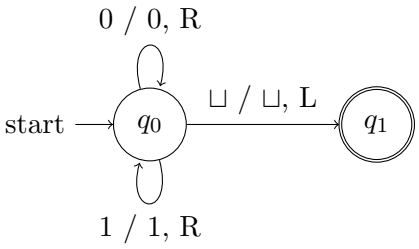

Figure 2.4: Turing machine to find RHS of the tape

	symbol		
state	0	1	⊔
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	$(q_1, \sqcup, L)$

Table 2.1: Transition table for Figure 2.4

Another way to represent a Turing machine is to describe it, for example, to perform a binary increment:

- Move to the right hand side of the word;
- Read the symbol, if it is a one then change it to a zero and move left;
- Otherwise write one onto the tape, move to the left hand side of the tape and, halt.

This is how a Turing machine is represented when the complexity is too high to show it as a state transition diagram. The actions to move to the right and the left can be called subroutines, where subroutines can simulate any type of subroutines found in programming languages, including recursive functions. A subroutine is a collection of states designed to perform a single function, calling the function by entering the subroutines initial state and exiting the function by moving to a return state (Hopcroft, 2003).

2.5 How are Turing machines taught currently

Turing machines are taught to students at a university level, with the expectation of knowledge of set theory and basic automatons, to understand the formal definition of the Turing machine. The tool JFLAP (see section 2.3.2) has been used in universities as a resource for learning, however the students would have needed to be taught about the Turing machine before being able to use it effectively. Another way to learn about Turing machines is to attend lectures, read textbooks and do problem sheets about constructing Turing machines for proofs. This is how most universities teach the topic of Turing machines. To construct a Turing machine by hand is tedious, requiring the writing out of the transition table on paper; it is also much harder to work out what the Turing machine does with a tape input as it is necessary to manually trace it using the table. An improvement could be made by introducing the Transition diagrams earlier, as it would be much easier to trace what the Turing machine computes with a tape input.

Chapter 3

Requirements and analysis

Requirements were gathered from the literature review, through researching into the design of game characteristics in section 2.2.3, and the AQA (2019) specification.

Requirements have three priorities, high, medium and low, and shall be defined with key words:

- **Must** The requirement is essential
- **Should** The requirement is ideal, but not necessary
- **May** The requirement is an extension

3.1 Non-functional Requirements

These requirements were gathered from the literature review, and are aims the serious game should accomplish to be successful

1.1 **Must improve the user knowledge of Turing machines.**
The primary goal of a serious game is to educate the user, so this is the most essential requirement. This was sourced from Abt's definition of a serious game (Abt, 1987).

1.2 **Should be easy to access.**
This is to reach as much of the target audience as possible. This requirement was elicited from discovering previous serious games, such as Scratch, which was a program but is now hosted on the internet for ease of access.

1.3 **Should receive feedback about what choices the user makes.**
A serious game uses feedback to keep the user motivated, and allows the user to see how their actions affect the outcome.

1.4 **Should be entertaining to play.**
Serious games that are entertaining can increase the user's interest and gives more motivation to start the learning process (Iten and Petko, 2016)

1.5 **Should not assume the player has a mathematical background, or any existing knowledge on the subject.**

This tool is to learn about Turing machines and no prior knowledge is required; it should be used to teach the model of computation, not the mathematics behind the model.

1.6 **Should feel intuitive to play.**
This can be achieved by reducing the number of ways to interact with the system, such as not using any presses on the keyboard and only using the mouse or touch interactions.

1.7 **May not assume that the player speaks a particular language.**
This is to increase the accessibility in order to reach a wider audience.

3.2 Functional Requirements

2.1 **Must be able to construct and display the Turing machine's Transitions and states.**
A Turing machine can be represented as a state transition diagram, or by the formal components of the mathematical model. This is important as this will be what the user interacts with so it should be designed with great care.

2.2 **Must be able to construct and display the tape of the Turing machine.**
The user should be able to give the Turing machine inputs to receive outputs, which is performed by the tape. The tape should also be able to move left and right and display the head where the tape is reading from.

2.3 **Must be able to represent a Turing machine subroutine.**
The subroutine needs a representation, as it is going to be used within the display of the Turing machine.

2.4 **Should give visual feedback when the machine is performing a step of the computation.**
This can be done by highlighting the transition and the state for which the Turing machine is currently in.

2.5 **Should be able to construct every Turing machine possible.**
This is so that every algorithm possible can be made in this game, increasing the possibilities of what can be done with the game.

2.6 **Should be able to construct levels.**
The levels should be an object containing the levels' properties which should be easy to change to make level creation as simple as possible. This requirement was elicited from Scratch (Resnick et al., 2009).

2.7 **Should prevent an non-deterministic Turing machine being constructed**
A non-deterministic Turing machine can have more than one transition to be performed for any given situation. This should be avoided as the game is teaching a deterministic Turing machine in which there are no uncertainties when performing a computation.

2.8 **Should be a web application that is runnable on modern browsers**
This is to ensure that the greatest number of people can play the game

2.9 **May use icons and symbols.**
To create a language independent game that can be used worldwide. This is ideal, but it should not interfere with the goal of teaching Turing machines, thus it is not a high requirement.

2.10 **May provide hints and tips**
These will be shown when the constructed Turing machine is invalid; The game should however still run the Turing machine to show why it is invalid. This promotes a tinkering approach which is what Scratch does.

2.11 **May have mobile support**
This is to make it accessible from almost every device that a user owns.

Chapter 4

Design

This chapter discusses the key design choices and methods used to create a serious game which meets the requirements specified in the previous chapter, focusing on the creation of the Turing machine and how it should be represented.

4.1 Turing Machine State Transition Diagram

This is a key part of the design as it is required to fulfil the essential requirement 2.1. It is also stated in requirement 1.6 that the game should feel intuitive to play and as the user must construct a Turing machine as the main game mechanic, much thought needs to be put into how the user interacts with this part of the design. The approach taken to make the game intuitive is to keep the design as simplistic as possible. State transition diagrams help in this aspect of simplistic design, as the other format representing a Turing machine is its formal definition: defining the transition table, a set of states, with the final and initial states, along with other items. The formal definition was avoided to reduce the mathematical notation, which follows requirement 1.5 of this project, as the transition table is harder to trace when compared to a state transition diagram.

In order to create a state transition diagram, the user must be able to create states, assign transitions between these states, set an initial state, set final states, and use subroutines. This is needed to fulfil the requirements 2.1 and 2.3 and therefore need a design for how the user will perform these actions. The design also should allow for any Turing machine to be build to follow requirement 2.5. To best determine the most simplistic way to perform each of these actions, two different approaches were considered. The first approach changes how state transition diagrams are defined, and the second does not redefine them, but merely extends it.

4.1.1 Approach 1

This approach introduces a new theoretical model on how to represent the Turing machine, by modifying the current representation of a Turing machine via state transition diagrams. The new state transition diagram contains:

- States, which could be either: empty, a set of transitions, or a subroutine.
- Arrows, which represent a change of state, as before.
- Transitions, which are reduced down into a read → write assignment, removing the movement of the tape from the transitions.

Movement of the tape is handled by a subroutine state, which is labelled left or right. This simplifies the transition down to only write onto the tape, removing the movement of the tape and the new state to change to, as that is handled by arrows.

A state containing a set of transitions can be seen in figure 4.1a. The left symbols represent the read of each transition, and the right on the write. The arrows from each write symbol show what state to change into after performing the transition. This was built on the idea of tracing the route of the computation of the Turing machine, by following the arrow into the state then branching into the correct transition, finally following the corresponding arrow out of the state.

A state that is a subroutine is labelled by the function it performs. The left and right subroutines will always be available, which move the tape in the corresponding direction. This is needed to ensure that the user can build all types of Turing machines, thus being Turing complete as required. Subroutines should be made by the creator of the level, so their internal representation should be a transition table, as they are not visually presented to the player. By not showing every transition, the state transition diagram is simplified. An example of a created subroutine can be seen in figure 4.1b, which moves the head of the tape to the right hand side of the tape.

(a) A transition set state (b) A subroutine state

Figure 4.1: New notation

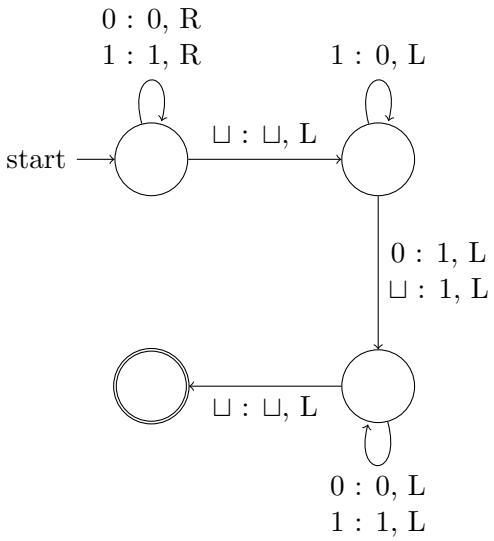

Figure 4.2: Standard notation for a binary incrementer

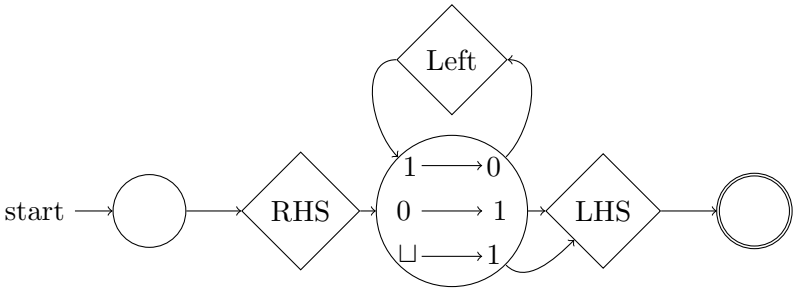

Figure 4.3: New Binary incrementer Turing machines

Analysis of the Approach

A Turing machine that increments a binary number was constructed to test this design. It was first constructed in the standard state transition diagram notation, as seen in figure 4.2, then the subroutines were figured out to be a Turing machine that moves to the right of the tape and one that moves to the left of it. It then was possible to create the machine in the new notation, as is shown in figure 4.3.

The number of states may have increased by two however, it is more obvious what happens during a computation, as the subroutines state what function they perform, instead of the user having to interpret what the transitions do. This avoids the unessential transitions to better understand how the overall machine functions.

For the user to construct this machine, it is necessary to add a state which contains all reads from the tape alphabet, and a Kleene star read, acting as the catch all (similar to TuringKara's blank transition (Reichert, 2003)). This state can be seen in figure 4.4. The user then needs to click the 'write' symbol in order to change it, which allows them to construct the transition set state. The user can then add the subroutines LHS, RHS, and left, and then arrows between each state to fully construct the machine.

The downfall with this approach is the question of whether it strays too far from the standard notation of the state transition diagrams with the transition set state. The

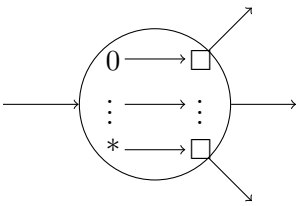

Figure 4.4: How the player can add transitions

transitions are split, which could run the risk of teaching that transitions do not change state or move the head on the tape. This is a serious matter, as teaching the wrong concepts would impede the users understanding of how transitions work within the Turing machine and therefore, would not meet the essential requirement 1.1.

4.1.2 Approach 2

This approach keeps as close as possible to the original state transition diagrams, by only adding a subroutine state. This subroutine is labelled by the function it performs, and when this subroutine state is entered, the subroutine Turing machine runs until it halts. It then moves on to pick a transition, if one exists from the subroutine state, otherwise it will halt.

The transitions are represented as normal: an arrow pointing to the new state and labelled as 'Read : Write, Move'. This is the true representation of a transition, as it has not been split up or modified in any way, as was done in the first approach. It will also not allow a non-deterministic Turing machine to be built by not allowing some assignment of transitions between states, which follows requirement 2.7. The transitions will be assigned to the states by providing a limited subset of transitions, as to provide a hint on how to approach building such a Turing machine, then dragging between two states to select the new state.

This is the most simplistic design approach, as it only extends the current state transition diagrams to include a subroutine node and keeps everything else the same. The idea being, that to be as simple as possible, anything that does not need to be changed should be kept the same

Analysis of the Approach

The Turing machine that increments a binary input, constructed for the normal state transition diagram, will be used again to test this design approach. This Turing machine can be seen in figure 4.2. The one constructed for this design approach will add the subroutines LHS and RHS as before, however the left subroutine is not needed as it is handled by the transitions now in this approach. The machine constructed using this design can be see in figure 4.5

This design approach adds only one extra state and reduces the number of transitions through the use of subroutines. It also maintains the look of the normal state transition diagrams, which is helpful for users who have prior knowledge about state transition diagrams, making it easier to construct Turing machines in this design approach.

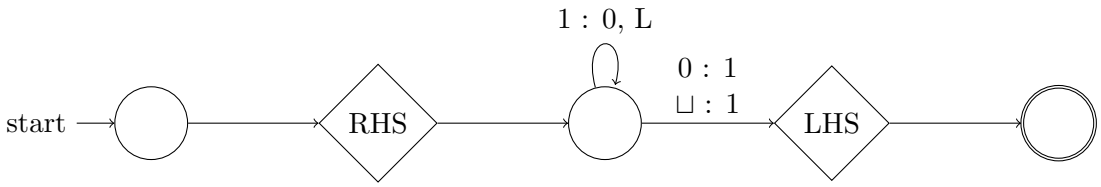

Figure 4.5: New notation

Overall this approach is simple, and clearly shows the current state of the Turing machine with the addition of subroutines, removing non-essential transitions, while maintaining the look of the normal state transition diagrams. This fulfils requirement 1.1 so this will be the approach taken in the implementation stage.

4.2 Tape and Computations

There will only be one tape for this Turing machine, and this tape will only have one controllable head. This design choice was made to keep the Turing machine as simple as possible, following the simple design approach of the Turing machine state transition diagram. Having more than two heads or tapes would complicate the machine unnecessarily, and as Church-Turing thesis states, all variations of Turing machines are computationally equivalent therefore, only a single tape and a single head is needed. This allows for all Turing machines to be constructed as requirement 2.5 states.

The tape will be a row of boxes for each tape cell, and will have a arrow head pointing to the head of the tape. This tape can then move left or right by moving all the tape cells respectively while keeping the head in the same position. This will be the representation used to show to the user. This fulfils the requirement 2.2, while maintaining the simplistic look of the design. This design can also allow for an seeming infinite tape.

To perform a step in the computation, the given transition should be highlighted. This is to meet requirements 2.4 and 1.3, as it would provide feedback to the user on what the step the Turing machine is performed. To perform a computation, there needs to be a way to run the Turing machine, so controls need to be designed for this purpose.

The controls available to use the Turing machine constructed will be:

- Play: runs the Turing machine at normal speed.
- Step: runs a single step of the Turing machine.
- Skip: runs the Turing machine at maximum speed.
- Step back: reverses the step taken.
- Skip back: reverses the Turing machine back to its original state.

4.3 Levels

To create levels for the requirement 2.6, it is necessary to define the components within the levels. A level will contain the following components:

- Transitions: a limited amount of transitions will be provided
- Subroutines: Added by defining the transition table of the subroutine to be added.
- Goal: A human readable version of the Turing machine to construct.
- Tests: Tape inputs and expected outputs to verify that the Turing machine achieved the goal.

A limited number of transitions are provided to attempt to create an entertaining game, as specified in requirement 1.4, by creating it into a puzzle game. This can allow for red herrings and partially correct solutions, to which the game can give feedback to assist the learning process.

4.4 Level Creator

This would allow the user to create there own levels, allowing for teachers to use the game as a tool for education. The level creator will be designed to allow for the user to create their own custom subroutine, allowing for more levels to be created. Then inputs needed to create a level are

- The goal to reach.
- The transitions to display.
- The tests to run on the Turing machine.
- The subroutines to provide to the player.

These inputs can be provided by forms, and by using the Turing machine state diagram creator to create subroutines.

Chapter 5

Detailed Design and Implementation

This chapter focuses on how to implement the designs using software, ensuring that the correct approaches were taken. It will discuss the design of the systems that are needed to structure the serious game, and the challenges encountered.

5.1 Languages and Software Used

The language used to create the game library was JavaScript ECMAScript 2015+, with the use of the Babel compiler to create a backwards compatible version of JavaScript so that it is runnable on older browsers. This was done to fulfil requirement 2.8 and 1.2, as the backwards compatible code can run on almost every browser.

The server was made using the Node.js runtime, using Express for the web routing and mongoDB for a NoSQL database. This was needed for requirement 2.6, as it is needed to store the created levels. The NoSQL database stores the level and then Express can use the URL to get the level from the database. The server is hosted on Heroku, which is a cloud platform that supports Git. This is useful as Git and GitHub were used for version control.

The choice to use a database with a web server allows for the server to query the database to fetch levels, and to generate a web page which adds that level to the JavaScript. This allows for a level creator to be made within the game.

5.2 Drawing State Transition Diagrams

To draw the state transition diagrams, the Scalable Vector Graphics (SVG) format was chosen instead of the Canvas format as it scales to the device using the application, and allows for DOM manipulation of the SVG objects. The library Snap.svg was chosen as it is a pure SVG library that was created by Adobe, which means it has a lot of documentation online as it is a popular library. It is also was easier to learn than D3.js, which is another popular SVG library.

The user can create states by clicking a fixed state object, which will generate a state which can be dragged and have transitions assigned to and from the state. Drawing the states was simple and the Snap.svg library provided a drag function for the state.

Drawing the arrows labelled with the transition was trivial, as a SVG path was constructed for the line and then text was put upon the same SVG path. There were challenges on the assignment of the transitions between states, as the head of the arrow would be placed on the center of the state and not at the border of the state. To fix this issue, the angle of the incoming transition for the state was taken, and then the sine and cosine were calculated with this angle. It is now possible to find the x and y coordinates for any angle for the edge of the circle by multiplying the result by the radius of the state. So the coordinates x and y were calculated using the equation:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} cx + r\cos(\theta) \\ cy + r\sin(\theta) \end{bmatrix} \tag{5.1}$$

where cx and cy are the coordinates for the origin of the circle, and r is the radius of the circle. This resulted in an error as the first point was correctly on the edge of the state, however the second point going to the next state was 180 degrees out of phase, so this was fixed by adding 180 to the angle. Listing 5.1 shows how it is implemented in JavaScript.

```
1  // get angle between -180 and 180 from the x axis
2  var angle = Snap.angle(this.x1, this.y1,this.x2, this.y2)-180
3
4  //Create path from the edge of a state to the edge of another state, hard
       coded 50 radius
5  line.attr({
6    // (x1,y1) uses the angle normally as it is exiting there
7    // (x2,y2) adds 180 to the angle, as incoming angle is 180 degrees out of
         phase.
8    d : `M  ${this.x1 + Snap.cos(angle)*50},
9            ${this.y1 + Snap.sin(angle)*50}
10         L  ${this.x2  + Snap.cos(angle+180)*50},
11           ${this.y2 + Snap.sin(angle+180)*50}`
12 })
```

Listing 5.1: Drawing an edge to edge arrow

A issue with overlapping occurs however, as when states q_1, q_2 have transitions such that $q_1 \rightarrow q_2$ and $q_1 \leftarrow q_2$, they overlapped and could not see what transitions went to which state. This was solved with a new SVG path for the arrow. Using an arc curve, it allowed for a bend to occur when this event happens. The construction of the path was trivial as it was similar to the previous issue. It also allowed for the transitions to go back to itself when it detects that the length of the arrow is less than the radius of the circle

Now that states and transitions can be drawn, it is possible to construct a state transition diagram. It however does not have any functionality or subroutines to use.

5.3 Turing Machine

This section focuses on the Turing machine functionality of the game, as it is required to be Turing complete as stated in requirement 2.5. The components needed to make the Turing machine are a transition function and a tape. The Turing machine must also be

able to perform a computation and verify that the Turing machine performs a set goal, which was done through the use of tests.

5.3.1 Transition Function

To create a Turing machine from the state transition diagram constructed, the Turing machines' transition function needs to be reconstructed from the state diagram. To follow the formal definition in defined in section 2.4, it should be a function such that:

$$\delta(q, x) = (q', x', d) \tag{5.2}$$

where q and q' are states, x and x' are tape symbols, and d is the direction to move the head.

This was done with a JavaScript object by using key : value pairs. It is possible to create a transition table object that can be represented by a JavaScript data structure of the form found in the listing 5.2. The key 'Read' is replaced by the symbol that the transition object holds. This was done to follow the formal definition of the transition function.

```
var Program = {
  State : {Read : Transition,
           Read : Transition,
           ...},
  State : {Read : Transition,
           Read : Transition,
           ...},
  ...
}
```

Listing 5.2: Transition table object

This data structure was chosen because when accessing `Program[`q`][`x`]` it returns the transition, which holds the the triple (q', x', d). This simulates the transition function δ perfectly, thus providing an essential component to the Turing machine.

5.3.2 Tape

The tape can be considered as a separate component of the Turing machine, it is the object that acts as the input and output to the Turing machine, and the tape is what the machine operates on. To create a tape for the machine to operate on, it was a design choice to have the Tape and Turing machine separated, and have the Turing machine perform calls onto the tape, with the main functions called being `read`, `write`, `moveRight` and `moveLeft`. This allowed for the issues with the tape to be related only with the tape and not the Turing machine.

The main issue with the tape was making it seem infinitely long. The issue being that when adding a tape cell to the beginning of the tape, the read function would return an incorrect value. This was solved by shifting the indexing of the head by increasing where the head is indexed from by one. The code listing seen in 5.3 performs this action.

```
// Add tapecell to start of tape when needed
if(head+min < 9){
  min++;
```

```
4     tapeCells.unshift(new TapeCell("",head))
5     tapeArray.unshift(blank)
6     tapeSVG.add(tapeCells[0].svg);
7   }
```

Listing 5.3: function called to prepend to tape

This function would only be performed when moving left at the edge of the left hand of the tape. When moving right at the edge of the right, a similar function was required. It can be found in listing 5.4. This function adds a tape cell to the end of the tape. The `max` value was taken at the start, which is the length of the tape at the start, so the left function would not interfere with this right function.

```
1  // When this happens, the tape must extend from the right hand side
2  if(max-head == 18){
3    tapeCells.push(new TapeCell("",max))
4    max++;
5    tapeArray.push(blank)
6    tapeSVG.add(tapeCells[tapeCells.length-1].svg);
7  }
```

Listing 5.4: function called to append to tape

5.3.3 Computation

To perform a step in the computation, the symbol read on the head of the tape and the current state are are passed into the transition function from before, which returns a transition object. This object contains the new state, the symbol to write to the tape, and the direction to move the head. The Turing machine can the perform a computation by the following steps if a transition exists:

1. Write the symbol to the tape.
2. Move the head of the tape.
3. Transition to the new state.

If a transition does not exist then halt the Turing machine. This however does not naturally include the kleene star into the computation. To solve this issue, the machine will check if a kleene star transition exists if no transition can be found.

```
1  // Read the tape and select corresponding transition
2  var read = tape.read()
3  transition = program[currentState.n][read]
4  if(!transition){ // If no transition found, look for a kleene transition
5    transition = program[currentState.n]["*"]
6  }
7
8  // Check transition exists before starting the computation
9  if(transition){
10   //If the write is not a klenne star, perform the write to the tape
11   if(transition.write != "*"){
12     tape.write(transition.write)
13   }
14   // Move the tape head
15   ...
```

```
16    // Change state to the new state
17    ...
18  }else //halt
```

Listing 5.5: Turing machine step

The code listing 5.5 shows how this solution was implemented, along with another fix to allow the kleene star to be used to not write anything to the tape, allowing for empty transitions.

5.4 Subroutines

These were implemented by reusing code from the states and the Turing machine. To create a subroutine, a program, initial state and final states should provided. The program is in the same form as found in listing 5.2, the transition object is just an object containing only the symbol to write to the tape, the direction to move the head and the new state to enter, so in JavaScript this is equal to {`write:"`x'`", move:"`d`", newState:"`q'`"`}.

This allowed for the subroutines to be independent from the Turing machine calling the subroutine. The main issues with the subroutines was when creating levels: to create a subroutine's program, when it contains a subroutine, would create a lot of issues as some states would be labelled the same. To solve this issue, a name hierarchy was created. Each subroutines can then be converted into a program, for example for a subroutine with two internal states, initial state 0 and final state 1, it would be converted into:`sr1, sr1_1`, and then the final states would exit using the subroutine exit transition. This made it possible to create subroutines with subroutines which can handle any level of recursion, and allows for insertion into the transition function. The listing 5.6 shows how this is implemented.

```
1  // for each state
2  Object.entries(programCopy).forEach(
3    ([state, read]) => {
4      // for each transition, convert nextState to form sr+n_n
5      Object.entries(programCopy[state]).forEach(
6        ([read, transition]) => {
7          if(transition["n"] != initialState){
8            programCopy[state][read]["n"] = this.n + "_"+ programCopy[state][
                 read]["n"]
9          }else{
10           programCopy[state][read]["n"] = this.n;
11         }
12       }
13     );
14   }
15 );
```

Listing 5.6: Function to convert into the new format

5.5 Levels and their creation

From the level design section 4.3, the components will be stored into a JSON format, which allows for the levels to be stored in a NoSQL database. To do this, a schema was

created for the level to define how the structure of the JSON object should be, and this schema can be found in listing 5.7.

```
const levelSchema = new mongoose.Schema({
  "goal": String,
  "transitions":[{"read" : String,"write": String,"move" : String, "
     quantity" : Number}],
  "tests":[{"input" :String, "output": String, "accepts" : Boolean }],
  "subroutines":[{
    "name" : String,
    "program" : String,
    "initialState" : String,
    "finalStates" : [String]
  }]
}, {minimize: false,
    timestamps: true});
```

Listing 5.7: Schema to validate level

Now levels that the user creates can be stored on the NoSQL database with ease as mongoDB supports JSON objects. This increases the reach of the game, as teachers can create custom levels to teach and give the students a link to the level to play.

The main challenge of these levels was how to get the data from the client to the server and then to the database. An AJAX call was made in JQuery to send a JSON object containing the details of the level to the server via a POST request, then the server will create a level object using the schema and the details received, and then add the level object created into to the database.

Chapter 6

Testing

This chapter describes the testing strategy, detailing test plans for unit testing, an observed playthrough and a questionnaire.

6.1 Unit Testing

The purpose of this section is to help find and remove bugs that are in the implementation of the game. Unit tests are normally functions that run a task to test if it returns the expected result, if it does then it passes. This helps to find bugs in the game as it can be tracked down to that test and then the functions called can be inspected. Unit tests like this are useful in teams as the code base is spread across a number of people, however as a single person knows the entire code base, unit tests such as this were deemed unnecessary.

The unit testing in this stage was to playthrough the game as normal first, which involved creating a Turing machines and running it on a tape input. Then more abnormal behaviour was tested such as, unexpected inputs, mobile devices and different browsers. This allowed bugs to be discovered and the function responsible found quickly.

This form of testing was performed when new features and code were added, to test if any new bugs were introduced into the game, allowing for the game to be in a playable state before performing observed playthroughs.

This process was performed throughout the implementation of the game, testing if each function performs the expected task without any unexpected outcomes.

6.2 Observed Playthroughs

An observed playthrough lets a user have free control over the game, letting them play and create levels. This was done to gather information on how a user with no prior knowledge of how the game functions would use the interface. This information can then be used to evaluate the Non-functional requirements of the game, and can be used to determine what when well and want did not.

This will be performed during a group supervisor meeting, by students who are also

developing a serious game for the purpose of education. The task for the students to perform is to create a Turing machine which inverses a binary word. This task was chosen as it is simple and tests the mechanics of the Turing machine creator. Comments made about the game will be noted down, and improvement will be made to the game based on this feedback.

6.3 Questionnaire

A questionnaire was constructed to find out if the game achieved the non-functional requirements, thus reviewing the game in its final state. This also gives the chance to receive feedback on how the game feels to play, and suggestions for future improvements. The questionnaire was constructed using Google forms, and will be posted online on social media, meaning that the audience will be friends and family, thus many will have no prior experience with Turing machines.

The questionnaire is separated into 3 stages, and the full questionnaire can be found in appendix C

6.3.1 Stage 1

In this stage, the participant will attempt to create two Turing machines through the game. The Turing machines being built are to: inverse a binary word, increment a binary word. These machines were chosen as they are thought of as simple Turing machines to construct.

The participant is first asked to read the home page of the game, which informally describes what a Turing machine is, and how they can perform a computations. Then they will read the instructions on how to construct Turing machines using the game. Once completed, the participant can then start creating Turing machines.

The participant will not be asked to create a level as it requires prior knowledge of a Turing machine, and this feature should only be used by teachers.

Once the Turing machines have been attempted to be constructed,the participant will then continue to the next stage of this questionnaire.

6.3.2 Stage 2

The participant is asked to rate statements about the game. These statements are ranked from one to five, with one being strongly disagree, and five being strongly agree. The statements will also alternate from positive and negative statements as an attempt to remove bias.

The statements are as followed:

1. **It was easy to achieve the goal set out**
   This is for requirement 1.6, as the goals are thought to be simple machines, thus

difficulty creating such machines should be a measure for the intuitiveness of the game.

2. **The interface does not present itself in a very attractive way**
   To be able to justify the design choice made in the Design chapter; to be as simplistic as possible.

3. **I enjoyed the time I spent creating the Turing machine**
   A serious game benefits from being entertaining, as stated in requirement 1.4.

4. **It was difficult to create a Turing machine**
   This is to measure the intuitiveness of the controls, and the feedback system of hints. This statement relates to the requirements 1.3 and 1.6.

5. **I learnt more about Turing machines and how they function**
   This is to determine the success of the serious game, as in requirement 1.1, it is the main goal of this game.

6. **The error messages are not adequate**
   This is determine if the hints and error system help the user learn though feedback, which is requirement 1.3

7. **I did not need to know about any mathematics while performing the task**
   For a direct measure of the requirement 1.5

8. **Subroutines did not help me understand how a Turing machine works**
   Subroutines were added to reduce the complexity of a Truing machine and to help understand how it works, thus measuring requirement 1.1

Once completed, the participant will move onto the next stage.

6.3.3 Stage 3

The stage uses open ended questions to determine what went well and what needs improving, which allows for new requirements to be gathered. It also allows for any missed bugs to be reported within the questionnaire. This qualitative approach was the best for the purpose of generating new requirements.

The participant will be asked what they liked and disliked about the game, and how they would improve it. There is also a extra input to give anything they personally want to say about the game.

Chapter 7

Results and Discussion

This chapter discusses the results of the observed playthrough and the questionnaire. It goes onto explore new requirements and future work.

7.1 Observed playthrough

The critical notes made from the observed playthrough will be discussed, and state what changes will be implemented and what should be future work.

- **Should be a way to remove states**
  This was attempted to be implemented but was too difficult to do so within time constraints, thus will be future work.

- **Transition selected should be more apparent that it was selected**
  Transitions now grow in size and stick to that size when active, but its growth is still too subtle so more could be done to show this.

- **When reversing the computation, the old states remained red**
  This bug was too difficult to find, so the feature was removed and should be added into future work as will help to see what state the machine is in.

- **Information about what a Turing machine is should be added as it had to be explained what it is**
  This resulted in adding a home page with a description about what a Turing machine is and how to construct one.

- **Verify does not fully work as it was a correct Turing machine and it rejected it**
  This was an undiscovered bug in the verify function and was fixed.

The most positive comments were made about the clean and simple design, and the animations which showed a change of state were liked.

Once the changes were implemented, it was possible to send out the questionnaire.

Question	Mean	S.d.
It was easy to achieve the goal set out	2.167	0.786
The interface does not present itself in a very attractive way	2.444	1.097
I enjoyed the time I spent creating the Turing machine	3.222	1.060
It was difficult to create a Turing machine	3.833	1.200
I learnt more about Turing machines and how they function	3.722	1.074
The error messages are not adequate	2.333	1.188
I did not need to know about any mathematics while performing the task	3.056	1.349
Subroutines did not help me understand how a Turing machine works	3.333	1.029

Table 7.1: Results of the second stage questionnaire

7.2 Questionnaire

The raw results from the questionnaire can be seen in appendix D. For the second stage of the questionnaire, which can be seen the the table 7.1, it shows that the primary goal of this game was achieved, and that is to teach the user what Turing machines are. Unfortunately however, it was both difficult to construct the Turing machine and to achieve the goal set out, which would seem that either the instructions were too complex or the controls were too difficult to handle. This is supported by the third stage of the questionnaire, as comments made to improve the game included:

> "A tutorial would go a long way towards making it all really clear for somebody who doesn't know much about bitwise computing, perhaps a guided tour of the game designer solving one of the levels with some text to explain each step."

> "Explain a little more clearly how to put together a Turing machine, maybe include a guided example of creating one, even just using like images."

> "Forced tutorial due to laborious instructions on how to play"

These quotes clearly show that some participants may have struggled making the Turing machines as they may not have understood the instructions that were given to them, and that an interactive tutorial would have been the correct way to teach the mechanics. The subroutines did not help either as showed by the results, however this is because they were not explained very well.

The second stage also shows that the interface is attractive, as thought during the design, which is supported by comments made in the final stage:

> "Very clear layout, clean design."

> "The flow chart layout of how the created machine 'thinks' is really nice. Once a machine has been made you can easily follow the steps it will take when it follows the tape."

It is important not to ignore the critical comments about the interface, as some participants were confused about how to assign a transition and what the controls under the tape do. The main feedback received to improve this was to explain things better and more clearly, as stated in this comment

> "It took some guessing to figure out how to remove an arrow once it was placed. Also some things were not explained very well, like what 'B' means in a read:write direction arrow. It was also not clear what to expect from the subroutines and how to include them, so that also required some trial and error before it clicked."

The results show mixed reviews on how entertaining the game was, which could be due to lack of a tutorial for the main mechanics of the game, and thus if a game does not simply work it will annoy the user, which is bad for creating a game for enjoyment.

The result of the mathematics needed can be explained not as the Turing machine construction, but as the binary tasks to perform, as many participants did not know what a binary word is and thus how to perform the task. This could be improved by adding a description box to the goal, which can describe the terminology. This description could also provide the user information about the subroutines available.

Chapter 8

Conclusion

8.1 Achievements and Reflection

The aim of this project was to construct a serious game to be used as a learning resource to help students,with little or no prior mathematical knowledge, understand the concept of a Turing machine. The topic of serious games was reviewed, to see the education benefits to the player of the game. Examples of serious games were also reviewed along with the effectiveness of the games, which produced ideas on how to construct a serious game that teaches Turing machines.

The design and implementation of this game was produced, following a set of requirements which should be met to make an effective serious game. The approach taken with adding only subroutines went well in the implementation stage as it did not make it more complex, however the subroutines could have been explained better to the player of the game. This was discovered when performing the questionnaire during the testing phase.

The results of this project produces a serious game, which meets the goal of teaching the concept of Turing machines however, there is a lot of room for improvement, such as an interactive tutorial rather than only instructions on how to play the game. The game excelled in showing the flow of the Turing machines computation through the use of animations.

The design choice to let the users create levels was a highlight of this project as it provides limitless possibilities for the levels, allowing for teachers to assign tasks to students to perform a level that they have constructed. The level creator also allows for subroutines to be constructed by the teacher, but on self reflection, it is not explained very well so should be improved.

Overall, the serious game has serious potential to be a learning tool that can be used in schools to teach students about the fundamentals of computation. In its current state however, it does not explain how to use the tool effectively, which will result in the tool to be misused. However to prevent this, further changes must be made to reach the maximum potential of this serious game.

8.2 Future work

The section will discuss how to improve the game in its current state, and how to add the possibility of monetisation.

8.2.1 Interactive tutorial

This was the most requested feature from the questionnaire, which could be implemented either through the use of animated gifs or videos. The most effective tutorial would be one that forces the user to play though a series of levels that gets them to learn the very basic mechanics so they can go onto more complex levels.

The series of levels could be implemented in the game's current state by creating a new section of levels that are the tutorial levels.

8.2.2 Create, Read, Update, Delete (CRUD)

The current game supports creation and reading of levels from a database, however it would be foolish to add a remove and update feature in the current state, as it would leave the system vulnerable, since there are no permissions system in place, consequently anyone could delete levels.

Thus, to make this into a full CRUD application, a user database is required. This would require the database to be restructured, and levels would need to have owners. The number of levels created could also be limited to a certain number of free levels, allowing for monetisation by offering a premium service to the users, which allows for unlimited level creation and perhaps extra features such as updating their levels.

Bibliography

Abt, C.C., 1987. *Serious games*. University press of America.

All, A., Castellar, E.P.N. and Looy, J.V., 2016. Assessing the effectiveness of digital game-based learning: Best practices. *Computers & Education*, 92-93, pp.90 – 103. Available from: http://doi.org/10.1016/j.compedu.2015.10.007.

AQA, 2019. *As and a-level computer science specification specifications for first teaching in 2015*. Manchester: AQA. Available from: https://filestore.aqa.org.uk/resources/computing/specifications/AQA-7516-7517-SP-2015.PDF.

Backlund, P. and Hendrix, M., 2013. Educational games - are they worth the effort? a literature survey of the effectiveness of serious games. *2013 5th international conference on games and virtual worlds for serious applications (vs-games)*. pp.1–8. Available from: http://doi.org/10.1109/VS-GAMES.2013.6624226.

Charsky, D., 2010. From edutainment to serious games: A change in the use of game characteristics. *Games and Culture*, 5(2), pp.177–198. https://doi.org/10.1177/1555412009354727, Available from: http://doi.org/10.1177/1555412009354727.

Hopcroft, J.E., 2003. *Introduction to automata theory, language and computation*. 2nd ed. Upper Saddle River, N.J.: Addison Wesley.

Iten, N. and Petko, D., 2016. Learning with serious games: Is fun playing the game a predictor of learning success? *British Journal of Educational Technology*, 47(1), pp.151–163. Available from: http://doi.org/10.1111/bjet.12226 [Accessed May 4, 2019].

Kapp, K.M., 2012. *The gamification of learning and instruction: game-based methods and strategies for training and education*. John Wiley & Sons.

Lewis, H.R. and Papadimitriou, C.H., 1997. *Elements of the theory of computation*. 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Maloney, J., Resnick, M., Rusk, N., Silverman, B. and Eastmond, E., 2010. The scratch programming language and environment. *Trans. Comput. Educ.*, 10(4), pp.16:1–16:15. Available from: http://doi.org/10.1145/1868358.1868363.

Michael, D.R. and Chen, S.L., 2005. *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade.

Reichert, R., 2003. *Theory of computation as a vehicle for teaching fundamental concepts of computer science*. Ph.D. thesis. ETH Zurich. Diss., Naturwissenschaften ETH Zürich, Nr. 15035, 2003. Available from: http://doi.org/10.3929/ethz-a-004526601.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y., 2009. Scratch:

Programming for all. *Commun. ACM*, 52(11), pp.60–67. Available from: `http://doi.org/10.1145/1592761.1592779`.

Rodger, S.H. and Finley, T.W., 2006. *Jflap: an interactive formal languages and automata package*. Jones & Bartlett Learning.

Wilkinson, P., 2016. A brief history of serious games. *Entertainment computing and serious games*, Springer, pp.17–41. Available from: `http://doi.org/10.1007/978-3-319-46152-6_2`.

Wilson, A., Hainey, T. and Connolly, T., 2012. Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. *6th european conference on games-based learning (ecgbl)*. pp.4–5.

Appendix A

Code

A.1 Arrow.js


```
function Arrow(x1,y1,x2,y2,txt){
  this.x1 = x1;
  this.x2 = x2;
  this.y1 = y1;
  this.y2 = y2;

  //save initial points
  this.oldx1 = x1;
  this.oldy1 = y1;
  this.oldx2 = x2;
  this.oldy2 = y2;

  this.txt = txt;
  this.curveFlag = 0;
  var ghostSvg;

  /**
   * Construct SVG elements
   */
  // Triangle to put at the end of the real line
  var marker = canvas.polyline("0,10 5,0 10,10").attr({fill: "#000"}).
      transform('r90').marker(0,0,10,10,9,5);
  var line = canvas.path(`M ${x1},${y1} L${x2},${y2}`)
  .attr({
    stroke: "#000",
    strokeWidth: 2,
    markerEnd: marker,
    fill :"none",
    cursor: 'pointer'
  });
  var linetxt = canvas.text(0,0,txt).attr({
    textpath : line,
    "text-anchor": "middle",
    "font": "25px sans-serif",
    "dominant-baseline" : "middle",
    dy:"-1em",
    cursor: 'pointer'
  });
  linetxt.textPath.attr({ startOffset: '50%' });
  // Create a SVG group for the arrow
  this.svg = canvas.group(line,linetxt)

  // Triangle to put at the end of the ghost line
  var ghostmarker = canvas.polyline("0,10 5,0 10,10").attr({fill: "#999"}).
      transform('r90').marker(0,0,10,10,9,5);
  var ghostline = canvas.path(`M ${this.oldx1},${this.oldy1} L${this.oldx2},${
      this.oldy2}`)
  .attr({
    stroke: "#999",
    strokeWidth: 2,
    markerEnd: ghostmarker,
    fill :"none",
    cursor: 'pointer'
  });
  var ghostlinetxt = canvas.text(0,0,txt).attr({
    textpath : ghostline,
    "text-anchor": "middle",
    "font": "25px sans-serif",
    "dominant-baseline" : "middle",
```

```
    dy:"-1em",
    fill: "#999",
    cursor: 'pointer'
  });
  ghostlinetxt.textPath.attr({ startOffset: '50%' });
  ghostSvg = canvas.group(ghostline,ghostlinetxt)
  ghostSvg.remove()

  /**
   * Line control functions
   */
  this.straight = function() {
    // get angle between -180 and 180 from the x axis
    var angle = Snap.angle(this.x1, this.y1,this.x2, this.y2)-180

    //Create path from the edge of a state to the edge of another state, hard coded 50 radius
    line.attr({
      // (x1,y1) uses the angle normally as it is exiting there
      // (x2,y2) adds 180 to the angle, as incoming angle is 180 degrees out of phase.
      d : `M  ${this.x1 + Snap.cos(angle)*50},
              ${this.y1 + Snap.sin(angle)*50}
           L  ${this.x2  + Snap.cos(angle+180)*50},
              ${this.y2 + Snap.sin(angle+180)*50}`
    })
  }

  this.curve = function(){
    var flag = 0;

    //calculate length and angle between the states
    var length = Snap.len(this.x1, this.y1,this.x2, this.y2)
    var angle = Snap.angle(this.x1, this.y1,this.x2, this.y2)-180

    //if the length is less than this distance, then draw it to itself
    if (length < 150){
      length = 60
      flag = 1;
      angle = -135
    }

    // Create path from edge of state to another edge with a curve line, hard coded 50 radius
    // If not the same state, flag = 0 so (x2,y2) needs to be 180 degrees out of phase
    line.attr({
      d : `M  ${this.x1 + Snap.cos(angle-30)*50},
              ${this.y1 + Snap.sin(angle-30)*50}
           A  ${length},${length},0,${flag},1,
              ${this.x2 + Snap.cos(angle+30+!flag*180)*50},
              ${this.y2 + Snap.sin(angle+30+!flag*180)*50}`
    })
  }

  this.update = function (){

    // Update line to follow rules
    if(this.curveFlag){
      this.curve();
    }else{
      this.straight();
    }

    //update text rotation.
    if(this.x2 - this.x1 < 0){
      linetxt.transform('r180');
    }else{
      linetxt.transform('r0');
    }
  }

  this.reset = function(){
    line.attr({
      d : `M ${this.oldx1},${this.oldy1} L${this.oldx2},${this.oldy2}`
    })
    linetxt.transform('r0');
    ghostSvg.remove()
  }

  /**
  * Movement control functions
  */
  this.dragArrow =  function (newX,newY){
    this.x2 = newX;
    this.y2 = newY;
    this.update();
  }

  this.dragState = function (newX, newY){
    this.x1 = newX;
    this.y1 = newY;
    this.update();
  }

  /**
  * Misc Functions
  */

  this.remove = function (){
    this.line.remove();
  }

  this.offsetLabel = function(n){
    linetxt.attr({
      dy:`-${n}em`
    })
    this.update()
  }

  this.animate = function(){
    line.animate({
      strokeWidth : 4
    },200,
    function(){
      line.animate({
        strokeWidth : 2
      },200)
    })
  }
```

```
173
174     this.showGhost = function(){
175       this.svg.add(ghostSvg);
176     }
177   }
```

A.2 State.js

```
1     function State(x, y, n){
2     var finalState = 0;
3     var initialState = 0;
4     var transitionsOut = [];
5     var transitionsIn = [];
6     this.n = ""+n;
7
8     /**
9      * Construct SVG elements
10     */
11    // Final state outline
12    var finalStateCir = canvas.circle(x,y,55).attr({
13      fill: "#fff",
14      stroke: "#000",
15      strokeWidth: 1,
16      cursor: 'pointer'
17    });
18    finalStateCir.remove()
19
20    // Initial state triangle pointing in
21    var initialStateTri = canvas.polygon(-50,0,-70,20,-70,-20).attr({
22      fill: "#000"
23    });
24    initialStateTri.transform(`t${x},${y}`)
25    initialStateTri.remove()
26
27    // States Circle
28    const cir = canvas.circle(x,y,50).attr({
29      fill: "#FAFAFA",
30      stroke: "#000",
31      strokeWidth: 1,
32      cursor: 'pointer'
33    });
34
35    // Text within the state
36    var txt = canvas.text(50, 50, states.length+1).attr({
37      "text-anchor": "middle",
38      "dominant-baseline" : "middle",
39      "font": "bold 30px sans-serif",
40      x : x,
41      y : y,
42      cursor: 'pointer'
43    })
44
45    this.svg = canvas.group(cir,txt);
46    this.svg.state = this;
47
48    /**
49     * Event listeners
50     */
51    this.svg.hover(function() {
52      // on hover
53      cir.animate({
54        r: 55
55      }, 200)
56      finalStateCir.animate({
57        r: 60
58      }, 200)
59    }, function(){
60      // on unhover
61      cir.animate({
62        r: 50
63      }, 200);
64      finalStateCir.animate({
65        r: 55
66      }, 200);
67    });
68
69    /**
70    * Drag controllers
71    */
72    var startcx;
73    var startcy;
74    var move = function (dx,dy,x,y){
75      //find corrected coordinates
76      var tdx, tdy;
77      var clientX, clientY;
78      var snapInvMatrix = this.transform().diffMatrix.invert();
79
80      if( (typeof dx == 'object') && ( dx.type == 'touchmove') ){
81        clientX = dx.changedTouches[0].clientX;
82        clientY = dx.changedTouches[0].clientY;
83        dx = clientX - this.data('ox');
84        dy = clientY - this.data('oy');
85      }
86
87      snapInvMatrix.e = snapInvMatrix.f = 0;
88      tdx = snapInvMatrix.x( dx,dy );
89      tdy = snapInvMatrix.y( dx,dy );
90
91      //transform center coordinates for the circles, text and initial State
            mark
92      cir.attr({
93        cx : tdx+startcx,
94        cy : tdy+startcy
95      })
96      txt.attr({
97        x : tdx+startcx,
98        y : tdy+startcy
99      })
100     finalStateCir.attr({
101       cx : tdx+startcx,
102       cy : tdy+startcy
103     })
104     initialStateTri.transform(`t${tdx+startcx},${tdy+startcy}`)
105
106     // For each transition out and in, transform the line
107     for (var i = 0; i < transitionsOut.length; i++) {
```

```
      transitionsOut[i].dragState(tdx+startcx,tdy+startcy)
    }
    for (var i = 0; i < transitionsIn.length; i++) {
      transitionsIn[i].dragArrow(tdx+startcx,tdy+startcy)
    }


  }

  var touchtime = 0;
  var start = function(x,y){
    //get the point where clicked
    if( (typeof x == 'object') && ( x.type == 'touchstart') ) {
        x.preventDefault();
        this.data('ox', x.changedTouches[0].clientX );
        this.data('oy', x.changedTouches[0].clientY );
    }
    startcx = this.getBBox().cx;
    startcy = this.getBBox().cy;

    if (touchtime == 0) {
      // set first click
      touchtime = new Date().getTime();
    } else {
      // compare first click to this click and see if they occurred within double click threshold
      if (((new Date().getTime()) - touchtime) < 800) {
        // double click occurred
        if(n != null){
          finalState = !finalState;
        }
        if(!finalState){
          finalStateCir.remove()
        }else{
          this.prepend(finalStateCir)
        }
        touchtime = 0;
      } else {
        // not a double click so set as a new first click
        touchtime = new Date().getTime();
      }
    }

  }

  this.undrag = function(){
    this.svg.undrag();
    this.svg.untouchstart();
    this.svg.untouchmove();
    this.svg.untouchend();
  }

  this.drag = function(){
    this.svg.drag(move, start);
    this.svg.touchstart(start);
    this.svg.touchmove(move);
  }

  /**
   * Transition getters + mutators
   */

  this.transitionsOutPush = function(transition){
    transitionsOut.push(transition)
  }

  this.transitionsInPush = function(transition){
    transitionsIn.push(transition)
  }

  this.transitionsOutDelete = function(transition){
    var i = transitionsOut.indexOf(transition)
    if(i >=0){
      transitionsOut.splice(i,1)
    }
  }

  this.transitionsInDelete = function(transition){
    var i = transitionsIn.indexOf(transition)
    if(i >=0){
      transitionsIn.splice(i,1)
    }
  }

  this.transitionsToState = function(state){
    var transitions = []
    for (var i = 0; i < transitionsOut.length; i++) {
      if(transitionsOut[i].newState() == state){
        transitions.push(transitionsOut[i]);
      }
    }
    return transitions;
  }

  this.getTransitions = function(){
    var object = {};
    for (var i = 0; i < transitionsOut.length; i++) {
      var read = transitionsOut[i].read
      object[read] = transitionsOut[i]
    }
    return object
  }
  this.getTransitionsOut = function(){
    return transitionsOut;
  }

  this.getTransitionsIn = function(){
    return transitionsIn;
  }

  /**
   * Misc functions
   */
  this.animate = function(){
    cir.animate({
      r:55
    },200,function(){cir.animate({
      r:50
    },200)})
  }
```

```
227     this.reset = function(){
228       cir.animate({
229         fill: "#FAFAFA"
230       },200)
231     }
232
233     this.clearTxt = function(){
234       txt.remove()
235     }
236     this.setInitialState = function(){
237       this.svg.prepend(initialStateTri)
238       initialState = 1;
239
240     }
241
242     this.isFinalState = function(){
243       return finalState;
244     }
245     this.isInitialState = function(){
246       return initialState;
247     }
248     this.isSubroutine = function(){
249       return 0;
250     }
251
252     /**
253      * Export the transitions in the form
254      * {State : {r : {w : _, m : _, n : _}}}
255      */
256     this.export = function(){
257       var json = {[this.n] : {}}
258       //assign
259       transitionsOut.forEach((value,index,array)=>{
260         if(value.read != "*"){
261           json[this.n] = Object.assign(value.export(), json[this.n]);
262         }
263       })
264
265       // assign kleene catch all
266       transitionsOut.forEach((value,index,array)=>{
267         if(value.read == "*"){
268           json[this.n] = Object.assign(value.export(), json[this.n]);
269         }
270       })
271       return json;
272     }
273   }
```

A.3 Subroutine.js

```
1    /**
2   * x,y : Int, inital coors
3   * name : String, To write on node
4   * program : Object, {state : {read : { w : a, m : b, n : c}}}
5   * initialState : String, of the inital state
6   * finalStates : [String], of final states
7   * tape : Tape Object
8   */
9   function Subroutine(x,y,n,name,program,initialState,finalStates,tape) {
10    this.name = name
11    this.program = program
12    this.initialState = initialState
13    this.finalStates = finalStates
14
15    var currentState = initialState;
16    var transition;
17
18    var transitionsOut = [];
19    var transitionsIn = [];
20    this.i = n;
21    this.n = "sr"+n;
22    var history = [];
23    //setup
24    const rhombus = canvas.rect(0,0,100,100).attr({
25      fill: "#FAFAFA",
26      stroke: "#000",
27      strokeWidth: 1,
28      cursor: 'pointer'
29    }).transform(`t${x},${y} r45`)
30
31    var txt = canvas.text(50, 50, name).attr({
32      "text-anchor": "middle",
33      "dominant-baseline" : "middle",
34      "font": "bold 30px sans-serif",
35      x : x+50,
36      y : y+50,
37      cursor: 'pointer'
38    })
39    this.svg = canvas.group(rhombus,txt);
40    this.svg.state = this;
41
42    this.svg.hover(function() {
43      rhombus.animate({ // on hover
44        "width": 110,
45        "height": 110,
46        "x":-5,
47        "y":-5
48      }, 200)
49    }, function(){ // on unhover
50      rhombus.animate({
51        "width" : 100,
52        "height": 100,
53        "x":0,
54        "y":0
55      }, 200);
56    });
57
58    this.animate = function(){
59      rhombus.animate({ // on hover
60        "width": 110,
61        "height": 110,
62        "x":-5,
63        "y":-5
64      }, 200, function(){rhombus.animate({
65        "width" : 100,
66        "height": 100,
```

```
      "x":0,
      "y":0
      }, 200);
    });
  }
  this.step = function(){}

  this.run = function(tape,speed) {
    var currentState = initialState
    while(true){

      //read
      var currentStateTransitions = program[currentState];
      var read = tape.read()

      transition = currentStateTransitions[read]

      //If kleene star exists
      if(!transition && currentStateTransitions["*"]){
        transition = currentStateTransitions["*"]
      }

      //check transition exists
      if (transition) {

        if(transition.w != "*"){
          // Write the new symbol where the head is on the tape.
          tape.write(transition.w)
        }

        // Move the tape head
        if(transition.m>0){
          tape.moveRight();
        }else if(transition.m<0){
          tape.moveLeft();
        }

        currentState = transition.n
        history.push({read : read, transition : transition});

      }else{
        if(finalStates.includes(currentState)){
          console.log("halted on final state")
          return 1;
        }
        else{
          console.log("halted on non-final state")
          return 0;
        }
      }
    }
  }
  this.unrun = function(offsetInput,tapeArrayInput,headInput){
    offset = offsetInput;
    tapeArray = tapeArrayInput;
    head = headInput;

    while(true){
      var action = history.pop()
      if (action == null){
        return 1;
      }
      var transition = action.transition

      head -= transition.m

      if(transition.m<0){
        tapeobj.moveRight();
      }else if(transition.m>0){
        tapeobj.moveLeft();
      }
      tapeArray[head] = action.read
      tapeobj.changeSymbolAt(head+offset,action.read)
    }
    this.animate();
  }

  var startcx;
  var startcy;
  /**
  * State drag controlers
  */

  var move = function (dx,dy,x,y){
    //find canvas coors
    var tdx, tdy;
    var clientX,clientY;
    var snapInvMatrix = this.transform().diffMatrix.invert();
    if( (typeof dx == 'object') && ( dx.type == 'touchmove') ){
      clientX = dx.changedTouches[0].clientX;
      clientY = dx.changedTouches[0].clientY;
      dx = clientX - this.data('ox');
      dy = clientY - this.data('oy');
    }
      snapInvMatrix.e = snapInvMatrix.f = 0;
      tdx = snapInvMatrix.x( dx,dy );
      tdy = snapInvMatrix.y( dx,dy );


    rhombus.transform(`t${tdx+startcx-50},${tdy+startcy-50} r45`);

    rhombus.attr({
      cx : tdx+startcx,
      cy : tdy+startcy
    })
    //text
    txt.attr({
      x : tdx+startcx,
      y : tdy+startcy
    })

    //transitionsOut
    for (var i = 0; i < transitionsOut.length; i++) {
      transitionsOut[i].dragState(tdx+startcx,tdy+startcy)
    }
    //transitionsIn
    for (var i = 0; i < transitionsIn.length; i++) {
      transitionsIn[i].dragArrow(tdx+startcx,tdy+startcy)
    }
```

```
187     }
188
189     var start = function(x,y){
190       startcx = this.getBBox().cx;
191       startcy = this.getBBox().cy;
192       if( (typeof x == 'object') && ( x.type == 'touchstart') ) {
193         x.preventDefault();
194         this.data('ox', x.changedTouches[0].clientX );
195         this.data('oy', x.changedTouches[0].clientY );
196       }
197     }
198
199     this.undrag = function(){
200       this.svg.undrag();
201       this.svg.untouchstart();
202       this.svg.untouchmove();
203       this.svg.untouchend();
204     }
205     this.drag = function(){
206       this.svg.drag(move, start);
207       this.svg.touchstart(start);
208       this.svg.touchmove(move);
209     }
210     this.isSubroutine = function(){
211       return 1;
212     }
213     this.getOffset = function(){
214       return offset;
215     }
216     this.getHead = function(){
217       return head
218     }
219     this.getTapeArray = function(){
220       return tapeArray
221     }
222
223     this.transitionsOutDelete = function(transition){
224       var i = transitionsOut.indexOf(transition)
225       if(i >=0){
226         transitionsOut.splice(i,1)
227       }
228     }
229
230     this.transitionsInDelete = function(transition){
231       var i = transitionsIn.indexOf(transition)
232       if(i >=0){
233         transitionsIn.splice(i,1)
234       }
235     }
236     this.getTransitionsOut = function(){
237       return transitionsOut;
238     }
239     this.getTransitionsIn = function(){
240       return transitionsIn;
241     }
242
243     this.transitionsOutPush = function(transition){
244       transitionsOut.push(transition)
245     }
246     this.transitionsInPush = function(transition){
247       transitionsIn.push(transition)
248     }
249     this.transitionsToState = function(state){
250       var transitions = []
251       for (var i = 0; i < transitionsOut.length; i++) {
252         if(transitionsOut[i].newState() == state){
253           transitions.push(transitionsOut[i]);
254         }
255       }
256       return transitions;
257     }
258     this.getTransitions = function(){
259       var r = {};
260       //{state : { read : {w : write, m : move, n : newState},
261       //           read : {w : write, m : move, n : newState}}}
262
263       // for each tape alphabet??
264       for (var i = 0; i < transitionsOut.length; i++) {
265         var read = transitionsOut[i].read
266         r[read] = transitionsOut[i]
267       }
268       return r
269     }
270     this.export = function(){
271       // copy program
272       var programCopy = JSON.parse(JSON.stringify(program))
273
274       // for each state
275       Object.entries(programCopy).forEach(
276         ([state, read]) => {
277
278           // for each transition, convert nextState to form sr+n_n
279           Object.entries(programCopy[state]).forEach(
280             ([read, transition]) => {
281               if(transition["n"] != initialState){
282                 programCopy[state][read]["n"] = this.n + "_"+ programCopy[state
                        ][read]["n"]
283               }else{
284                 programCopy[state][read]["n"] = this.n;
285               }
286             }
287           );
288
289
290           if(state != initialState){
291             if(finalStates.includes(state)){ // just final state
292
293               // assign transtions to final states
294               for (var i = 0; i < transitionsOut.length; i++) {
295                 var transition = transitionsOut[i].export()
296                 read = Object.assign(transition,read)
297               }
298
299               programCopy[this.n+"_"+state] = read
300               delete programCopy[state]
301             }
302             else{ // everything else but the inital state
303               programCopy[this.n+"_"+state] = read
304               delete programCopy[state]
305             }
```

```
306           }else if(finalStates.includes(state)){ //inital ++ finalstate
307             // assign transtions to final states
308             // BUG : if final state contains the same transtion as the exiting
                    transtion from
309             //       subroutine, this will create a logical error in the export
                    of subroutine
310             for (var i = 0; i < transitionsOut.length; i++) {
311               var transition = transitionsOut[i].export()
312               read = Object.assign(transition,read)
313             }
314
315             programCopy[this.n] = read
316             delete programCopy[state]
317           }
318           else{ //just inital state
319             programCopy[this.n] = read
320             delete programCopy[state]
321           }
322         }
323       );
324
325       return programCopy;
326     }
327   }
```

A.4 Tape.js

```
1     /*
2   *
3   * Input: initial tape input
4   */
5   function Tape(string,show){
6     if(show==null){
7       show = true;
8     }
9     var tapeCells = [];
10    var head = 0;
11    var min = 9;
12    var splitter = new GraphemeSplitter();
13    var tapeArray = splitter.splitGraphemes(string);
14
15    // set global
16    inputAlphabet = new Set(tapeArray)
17
18    // generate blank tapecells up untill head pos
19    for (var i = 0; i < min; i++) {
20      tapeCells.push(new TapeCell("",i))
21      tapeArray.unshift(blank)
22    }
23
24    // Write the string to the tape
25    for (var i = min; i < tapeArray.length; i++) {
26      tapeCells.push(new TapeCell(tapeArray[i],i))
27    }
28
29    // Fill up the rest of the space with tapecells
30    for (var i = tapeArray.length; i<19; i++){
31      tapeCells.push(new TapeCell("",i))
32      tapeArray.push(blank)
33    }
34
35    var max = tapeCells.length;
36    var tapeSVG = tapeCanvas.group()
37
38    // render the tape cells
39    for(var i=0;i < 19; i++){
40      tapeSVG.add(tapeCells[i].svg)
41    }
42    if(!show)tapeSVG.remove()
43
44
45    /**
46    * Movement Functions
47    */
48
49    // Move head left
50    this.moveLeft = function(){
51      head-=1
52
53      // Add tapecell to start of tape when needed
54      if(head+min < 9){
55        min++;
56        tapeCells.unshift(new TapeCell("",head))
57        tapeArray.unshift(blank)
58        tapeSVG.add(tapeCells[0].svg);
59      }
60
61      //render tape cell offscreen, before the animation
62      tapeSVG.add(tapeCells[(head+min - 9)].svg)
63
64      //remove the tape cell that is not showing before the animation
65      //check if it exists first
66      if (tapeCells[(head+min + 11)]) {
67        tapeCells[(head+min + 11)].svg.remove()
68      }
69
70      // animate
71      tapeSVG.animate({
72        transform:`t${(-head)*100}`
73      },200)
74    }
75
76    // Move head right
77    this.moveRight = function(){
78      head+=1
79
80      // When this happens, the tape must extend from the right hand side
81      if(max-head == 18){
82
83        tapeCells.push(new TapeCell("",max))
84        max++;
85
86        tapeArray.push(blank)
87        tapeSVG.add(tapeCells[tapeCells.length-1].svg);
88      }
89
```

```
    //render the tapecell offscreen, before the animation
    tapeSVG.add(tapeCells[(head+min+9)].svg)

    //remove the tape cell that is not showing before the animation
    if(tapeCells[(head+min - 11)]){
      tapeCells[(head+min - 11)].svg.remove()
    }


    //animate
    tapeSVG.animate({
      transform:`t${(-head)*100}`
    },200)
  }

  /**
  * Misc Functions
  */
  this.read = function(){
    return tapeCells[head+min].symbol
  }
  this.write = function(symbol){
    tapeCells[head+min].writeSymbol(symbol)
    tapeArray[head+min] = symbol
  }
  this.getTape =function(){
    for (var i = 0; i < tapeArray.length; i++) {
      if(tapeArray[i] != "B"){
        return tapeArray.slice(i)
      }
    }
  }
  this.getMin = function(){
    return min;
  }
}
```

A.5 TapeCell.js

```
function TapeCell(symbol,x){
this.symbol = symbol;
/**
* SVG elements
*/
var rect = tapeCanvas.rect(x*100, 0, 100, 100).attr({
  fill: "none",
  stroke: "#000",
  strokeWidth: 1
});
var txt = tapeCanvas.text(50+100*x,50,symbol).attr({
"text-anchor": "middle",
"dominant-baseline" : "middle",
"font": "40px sans-serif"
})
if(symbol == ""){
  this.symbol = "B";
}
// Create svg group
this.svg = tapeCanvas.group(rect,txt)

/**
 * Misc functions
 */
this.writeSymbol = function(symbol){
  txt.attr({
    text : symbol
  })

  if(symbol == ""){
    this.symbol = "B";
  }else{
    this.symbol = symbol;
  }
}
}
```

A.6 Transition.js

```

function Transition(x,y, read, write, move,n){
  var state = null;
  var newState = null;
  var transition = this;

  this.moveValue = 0;
  this.readHistory = []
  this.read = read;
  this.write = write
  this.move = move;

  if (this.move == "L"){
    this.moveValue = -1;
  }else if(this.move == "R"){
    this.moveValue = 1;
  }

  const arrow = new Arrow(x+20, y+50,x+125,y+50, `${read} : ${write} ${move}`)
  const rect = canvas.rect(x, y, 140, 70, 10).attr({
    fill: "#eaeaea",
    strokeWidth: 1,
    cursor: 'pointer',
    "opacity": 0.5
  });
  this.svg = canvas.group(rect,arrow.svg);
  const svg = this.svg

  this.svg.hover(function() {
    rect.animate({ // on hover
      "width": 150,
      "height": 80,
      "x":x-5,
      "y":y-5,
```

```
35        "opacity": 1
36      }, 200)
37    }, function(){ // on unhover
38      if(currentTransition != transition){
39        rect.animate({
40          "width" : 140,
41          "height": 70,
42          "x":x,
43          "y":y,
44          "opacity": 0.5
45        }, 200);
46      }
47    });
48    this.animateToNormal = function(){
49      if(currentTransition != transition){
50        rect.animate({
51          "width" : 140,
52          "height": 70,
53          "x":x,
54          "y":y,
55          "opacity": 0.5
56        }, 200);
57      }
58    }
59
60    this.svg.mousedown(function (){
61      currentTransition = transition;
62      for (var i = 0; i < globalTransitions.length; i++) {
63        for(var j = 0; j < globalTransitions[i].length;j++)
64        globalTransitions[i][j].animateToNormal()
65      }
66      for (var i = 0; i < states.length; i++) {
67        states[i].undrag();
68        drag(states[i]);
69      }
70      for (var i = 0; i < subroutines.length; i++) {
71        subroutines[i].undrag();
72        drag(subroutines[i]);
73      }
74    })
75
76
77    /**
78    * Drag controllers.
79    */
80    var startcx;
81    var startcy;
82    var dragMove = function(dx,dy,x,y){
83      var tdx, tdy;
84      var clientX,clientY;
85      var snapInvMatrix = this.transform().diffMatrix.invert();
86      if( (typeof dx == 'object') && ( dx.type == 'touchmove') ){
87        //x.preventDefault();
88        clientX = dx.changedTouches[0].clientX;
89        clientY = dx.changedTouches[0].clientY;
90        dx = clientX - this.data('ox');
91        dy = clientY - this.data('oy');
92      }
93        snapInvMatrix.e = snapInvMatrix.f = 0;
94        tdx = snapInvMatrix.x( dx,dy );
95        tdy = snapInvMatrix.y( dx,dy );
96      arrow.dragArrow(startcx+tdx,startcy+tdy);
97    }
98
99    var dragStart = function(x,y){
100     startcx = this.getBBox().cx;
101     startcy = this.getBBox().cy;
102     if( (typeof x == 'object') && ( x.type == 'touchstart') ) {
103         x.preventDefault();
104         this.data('ox', x.changedTouches[0].clientX );
105         this.data('oy', x.changedTouches[0].clientY );
106     }
107     arrow.dragState(startcx,startcy)
108     arrow.dragArrow(startcx,startcy)
109     if(n == 0){
110       arrow.showGhost()
111     }else{
112       rect.remove();
113     }
114
115   }
116
117   var dragEnd = function(e){
118     //e.preventDefault();
119     newState = null;
120     for (var i = 0; i < states.length; i++) {
121       var bbox = states[i].svg.getBBox()
122       if(Snap.path.isPointInsideBBox(bbox,arrow.x2,arrow.y2)){
123         arrow.dragArrow(bbox.cx,bbox.cy);
124         newState = states[i];
125       }
126
127       states[i].transitionsOutDelete(transition)
128       states[i].transitionsInDelete(transition)
129
130       states[i].undrag()
131       states[i].drag()
132     }
133     for (var i = 0; i < subroutines.length; i++) {
134       var bbox = subroutines[i].svg.getBBox()
135       if(Snap.path.isPointInsideBBox(bbox,arrow.x2,arrow.y2)){
136         arrow.dragArrow(bbox.cx,bbox.cy);
137         newState = subroutines[i];
138         console.log("test")
139       }
140
141       subroutines[i].transitionsOutDelete(transition)
142       subroutines[i].transitionsInDelete(transition)
143
144       subroutines[i].undrag()
145       subroutines[i].drag()
146     }
147
148     if(newState == null){
149       arrow.reset()
150       svg.prepend(rect)
151     }else{
152       var transitions = this.state.getTransitionsOut();
153       var repeated = 0;
154       for (var i = 0; i < transitions.length; i++) {
```

```
155         if(transitions[i].read == transition.read){
156           repeated = 1;
157         }
158       }
159       if(repeated){
160         arrow.reset()
161         svg.prepend(rect)
162       }else{
163         newState.transitionsInPush(transition)
164         this.state.transitionsOutPush(transition)
165
166         var transitionsFromState = this.state.transitionsToState(newState);
167         var transitionsFromNewState = newState.transitionsToState(this.state)
168
169         var curve = 0;
170         if(this.state == newState ||(transitionsFromState.length > 0 &&
              transitionsFromNewState.length > 0)){
171           curve = 1;
172         }
173         for (var i = 0; i < transitionsFromState.length; i++) {
174           transitionsFromState[i].offsetArrowLabel(i+1)
175           transitionsFromState[i].curveArrow(curve);
176         }
177         for (var i = 0; i < transitionsFromNewState.length; i++) {
178           transitionsFromNewState[i].curveArrow(curve)
179         }
180
181         state = this.state
182       }
183       currentTransition = 0;
184       transition.animateToNormal();
185     }
186   }
187
188   var drag = function(state){
189     state.svg.drag(dragMove,dragStart,dragEnd)
190     state.svg.touchstart(dragStart);
191     state.svg.touchmove(dragMove);
192     state.svg.touchend(dragEnd);
193   }
194
195   /**
196   * Event handlers
197   */
198
199
200
201   /**
202   * Getters
203   */
204
205   this.dragArrow = function(newX,newY){
206     arrow.dragArrow(newX,newY)
207   }
208   this.dragState = function(newX,newY){
209     arrow.dragState(newX,newY)
210   }
211   this.export = function(){
212     var movenumber=0;
213     if(move =="R"){
214       movenumber = 1;
215     }else if(move == "L"){
216       movenumber =-1;
217     }else{
218       movenumber = 0;
219     }
220     /*
221     if (read == "*" && write != "*"){
222       var json;
223       inputAlphabet.forEach((value,key,set) => {
224         var transition = { [value] : {w : write ,m : movenumber, n : newState.
              n}}
225         json = Object.assign(transition,json)
226       })
227       json = Object.assign({ [blank] : {w : write ,m : movenumber, n :
            newState.n}},json)
228       return json;
229     }
230     else if(write == "*") {
231       var json;
232       inputAlphabet.forEach((value,key,set) => {
233         console.log(value)
234         var transition = { [value] : {w : value ,m : movenumber, n : newState.
              n}}
235         json = Object.assign(transition,json)
236       })
237       json = Object.assign({ [blank] : {w : blank ,m : movenumber, n :
            newState.n}},json)
238       return json;
239     }*/
240
241     if (newState){
242       return {[read] : {w : write ,m : movenumber, n : newState.n}}
243     }else{svg
244       return 0;
245     }
246
247   }
248   this.newState = function(){
249     return newState;
250   }
251   this.oldState = function(){
252     return state;
253   }
254   this.offsetArrowLabel = function(n){
255     arrow.offsetLabel(n)
256   }
257   this.curveArrow = function(curveFlag){
258     arrow.curveFlag = curveFlag;
259     arrow.update();
260   }
261   this.animate = function(){
262     arrow.animate()
263   }
264   this.isSubroutine = function(){
265     return 0;
266   }
267 }
```

A.7 TuringMachine.js

```
/**
*
* Inputs types: transition table, state, state array, tape object
*/
function TuringMachine(program,initialState,finalStates,tape){

  var currentState = initialState;
  var transition;
  var interval;
  var history = [];

  this.step = function(){

    if(currentState.isSubroutine()){
      // Run the subroutine and save output
      var accepted = currentState.run(tape,200)
      var read = tape.read()
      // Save to history to enable reversal
      history.push(currentState)

      // Determine what state to go to next
      transition = program[currentState.n][read]

      // Check if there exists a klenne star if no transition can be found
      if(!transition && program[currentState.n]["*"]){
        transition = program[currentState.n]["*"]
      }

      // Ensure transistion exists before continuing
      if(transition){
        transition.animate()
        currentState = transition.newState()
        currentState.animate()
      }
      else{
        clearInterval(interval)
        return 0;
      }
    }
    else{
      // Read the tape and select corresponding transition
      var read = tape.read()
      transition = program[currentState.n][read]
      if(!transition){ // If no transition found, look for a kleene transition
        transition = program[currentState.n]["*"]
        if(transition){
          program[currentState.n]["*"].readHistory.push(read)
        }
      }

      // Check transition exists before starting the computation
      if(transition){

        //If the write is not a klenne star, perform the write to the tape
        if(transition.write != "*"){
          tape.write(transition.write)
        }
        transition.animate()


        // Move the tape head
        if(transition.moveValue > 0) tape.moveRight();
        if(transition.moveValue < 0) tape.moveLeft()

        // Change state to the new state
        currentState.reset()
        currentState = transition.newState()
        currentState.animate()

        // Save to history to be abled to be reversed
        history.push(transition)
      }
      else{
        // Halt machine
        clearInterval(interval)

        // If on final state then accept, else reject
        if(finalStates.includes(currentState)){
          return 1;
        }
        else{
          return 0;
        }
      }
    }
  }
  this.run = function(){
    currentState = initialState;
    interval = setInterval(this.step,500)
  }
  this.skip = function(){
    currentState = initialState;
    var accept = -1;
    while(true){
      accept = this.step();
      if(accept>-1){
        break;
      }
    }
    return accept;
  }

  this.unskip = function(){
    interval = setInterval(this.unstep,0)
  }

  this.unstep = function(){
    console.log(history)
    transition = history.pop();
    if(transition == null){
      console.log("At initialState")
      clearInterval(interval)
      return 1;
    }else if(transition.isSubroutine()){
      transition.unrun(tape)
    }
    else
```

```
    {
      currentState = transition.oldState()
      currentState.animate()

      if(transition.moveValue<0){
        tape.moveRight();
      }else if(transition.moveValue>0){
        tape.moveLeft();
      }
      if(transition.read == "*"){
        tape.write(transition.readHistory.pop())
      }else{
        tape.write(transition.read)
      }


      transition.animate()
    }
  }
  this.verify = function(input,output,accept){
    tape = new Tape(input,false);
    var accepted = this.skip();
    var tmout = tape.getTape()
    var i;
    for(i = 0 ; i<output.length;i++){
      if(output[i] != tmout[i]){
        return 0
      }
    }

    if(accepted != accept){
      return 0;
    }

    return 1
  }
  this.export = function(){
    var json = {}
    for (var i = 0; i < states.length; i++) {
      json.program = Object.assign(states[i].export(),json.program)
    }
    for (var i = 0; i < subroutines.length; i++) {
      json.program = Object.assign(subroutines[i].export(),json.program)
    }
    json.program = JSON.stringify(json.program)
    json.initialState = initialState.n;
    json.finalStates = [];
    for (var i = 0; i < finalStates.length; i++) {
      json.finalStates.push(finalStates[i].n)
    }
    return json
  }
}
```

A.8 create.js

```
var canvas = Snap(document.getElementById('canvasSVG'));
var tapeCanvas = Snap(document.getElementById('tapeSVG'));
var tapeobj
var states = [];
var subroutines = [];
var globalTransitions = [];
var tapeString = "010101"
var compiledtm;
var inputAlphabet;
var currentTransition;
var blank = "B";

var customtransitions = [];
var customsubroutines = [];
var tests = []

var preloadedSubroutines = [
    {
      "name" : "RHS",
      "program" : '{"0":{"0":{"w":"0","m":1,"n":"0"},"1":{"w":"1","m":1,"n
          ":"0"},"B":{"w":"B","m":-1,"n":"1"}},"1":{}}',
      "initialState" : "0",
      "finalStates" : ["1"]
    },
    {
      "name" : "LHS",
      "program" : '{"0":{"0" : {"w" :"0", "m" :-1, "n" : "0" },"1" : {"w"
          :"1", "m" :-1, "n" : "0" },"B" : {"w" :"B", "m" :1, "n" : "1" }},"1"
           : {}}',
      "initialState" : "0",
      "finalStates" : ["1"]
    },
    {
      "name":"INV",
      "program" : '{"0":{"0":{"w":"1","m":1,"n":"0"},"1":{"w":"0","m":1,"n
          ":"0"},"B":{"w":"B","m":-1,"n":"1"}},"1":{"0":{"w":"0","m":-1,"n
          ":"1"},"1":{"w":"1","m":-1,"n":"1"},"B":{"w":"B","m":1,"n
          ":"2"}},"2":{}}',
      "initialState" : "0",
      "finalStates" : ["2"]
    },
    {
      "name":"INC",
      "program" : '{"0":{"*":{"w":"*","m":0,"n":"sr0"}},"1":{"0":{"w":"1","m
          ":0,"n":"sr1"},"1":{"w":"0","m":-1,"n":"1"},"B":{"w":"1","m":0,"n":"
          sr1"}},"2":{},"sr1":{"*":{"w":"*","m":-1,"n":"sr1"},"B":{"w":"B","m
          ":1,"n":"sr1_1"}},"sr1_1":{"*":{"w":"*","m":0,"n":"2"}},"sr0
          ":{"*":{"w":"*","m":1,"n":"sr0"},"B":{"w":"B","m":-1,"n":"sr0_1"}},"
          sr0_1":{"*":{"w":"*","m":0,"n":"1"}}}',
      "initialState" : "0",
      "finalStates" : ["2"]
    }
  ]


function init(){

  for (var i = customtransitions.length - 1; i >= 0; i--) {
    var arrayOfTransitions = []
    for(var j = 0; j < customtransitions[i].quantity; j++){
```

```
        arrayOfTransitions.push(new Transition(1600 + 150*(i%2), 200 +((i+1)%2)*45, customtransitions[i].read, customtransitions[i].write, customtransitions[i].move,j));
      }
      globalTransitions[i] = arrayOfTransitions;
    }
    for (var i = customsubroutines.length - 1; i >= 0; i--) {
      const fixsub = new Subroutine(50 + i*170, 50,0, customsubroutines[i].name, JSON.parse(customsubroutines[i].program),customsubroutines[i].initialState,customsubroutines[i].finalStates)
      fixsub.svg.node.onclick = function(){
        var subroutine = new Subroutine(500,500,subroutines.length,fixsub.name,fixsub.program,fixsub.initialState,fixsub.finalStates)
        subroutine.drag()
        subroutines.push(subroutine)
      }
    }

    tapeobj = new Tape(tapeString);
    var fixedState = new State(1700, 100);
    fixedState.clearTxt();
    var state = new State(100,440,states.length);
    state.setInitialState();
    state.drag();
    states.push(state);

    fixedState.svg.node.onclick = function () {
      var state = new State(1500,100,states.length);
      state.drag();
      states.push(state);
    };
    // Mobile
    $('#canvasSVG').on('touchmove', function(e){e.preventDefault()});
  }

  function clearCanvas(){
    canvas.clear();
    tapeCanvas.clear()
    states = [];
    subroutines = [];
    globalTransitions = [];
    compiledtm = 0;
    init();
  }

  function step(){
    if(compiledtm){
      compiledtm.step();
    }else if(compileTM()){
      compiledtm.step()
    }
  }

  function reverseStep(){
    if(compiledtm){
      compiledtm.unstep();
    }else if(compileTM()){
      compiledtm.unstep()
    }
  }

  function playClicked(){
    compileTM()
    compiledtm.run()
  }

  function compileTM(){
    $('#alerts').empty()
    var transitions = {}
    var finalStates = []
    var initialState = "";
    var run = 1;
    var finalTransWarning = []
    for (var i = 0; i < states.length; i++) {
      transitions[states[i].n] = states[i].getTransitions();
      if(states[i].isFinalState()){
        finalStates.push(states[i])
        if(states[i].getTransitionsIn().length == 0){
          finalTransWarning.push(i+1)
        }
      }
      if(states[i].isInitialState()){
        initialState = states[i]
        if(states[i].getTransitionsOut().length == 0){
          $('#alerts').append(`<div class="alert alert-warning alert-dismissible fade show" role="alert">\
            <strong>Warning: No transitions from inital state ${i}!\
            <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button></div>`)
        }
      }
    }
    for (var i = 0; i < subroutines.length; i++) {
      transitions[subroutines[i].n] = subroutines[i].getTransitions();
    }
    if(finalTransWarning.length){
      if(finalStates.length == 1){
        $('#alerts').append(`<div class="alert alert-warning alert-dismissible fade show" role="alert">\
            <strong>Warning: No transitions to final state ${finalTransWarning.toString()}!\
            <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button></div>`)
      }else if (finalStates.length == 2){
        $('#alerts').append(`<div class="alert alert-warning alert-dismissible fade show" role="alert">\
            <strong>Warning: No transitions to final states ${finalTransWarning.join(' and ')} !</strong>\
            <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button></div>`)
      }else{
        $('#alerts').append(`<div class="alert alert-warning alert-dismissible fade show" role="alert">\
            <strong>Warning: No transitions to final states ${finalTransWarning.slice(0, finalTransWarning.length-1).join(', ')} and ${finalTransWarning[finalTransWarning.length-1]}!</strong>\
            <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button></div>`)
      }
    }
  }
```

```
    console.log(transitions)
    if(finalStates.length == 0){
      console.log("Error: No final states")
      run = 0;
      $('#alerts').append('<div class="alert alert-danger alert-dismissible fade show" role="alert">\
        <strong>Error: no final states!</strong>\
         You should add one by double clicking a state!.\
        <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button></div>')
    }
    if(run){
      compiledtm = new TuringMachine(transitions,initialState,finalStates,tapeobj)
      return compiledtm;
    }else{
      return 0;
    }

}
function tapeinput(){
    tapeString = document.getElementById("tapeinput").value;
    tapeCanvas.clear()
    tapeobj = new Tape(tapeString);
    compiledtm = 0;
}
function fastforward(){
    if(compiledtm){
      compiledtm.skip();
    }else if(compileTM()){
      compiledtm.skip()
    }
}
function reverseskip(){
    if(compiledtm){
      compiledtm.unskip();
    }else if(compileTM()){
      compiledtm.unskip()
    }
}
$(document).ready(() => {
    init();
    $('#transition').click(() =>{

      if(customtransitions.length == 0){
        $('#transitionList').empty()
      }

      customtransitions.push({"read" : $('#read').val(), "write" : $('#write').val(), "move" : $('#move').val(), "quantity" : $('#number').val()})
      var arrayOfTransitions = []
      for(var j = 0; j < $('#number').val(); j++){
        arrayOfTransitions.push(new Transition(1600 + 150*(globalTransitions.length%2), 200 +((globalTransitions.length+1)%2 + globalTransitions.length)*45, $('#read').val(), $('#write').val(), $('#move').val(),j));
      }
      console.log(arrayOfTransitions)
      globalTransitions.push(arrayOfTransitions)


    $("#transitionList").append('<li class="list-group-item text-center">' +
                                    $("#number").val() + ' x  If input is '+
                                    $("#read").val() + ', write ' +
                                    $("#write").val() + ', then move ' +
                                    $("#move").val() +
                                  ' <button type="button" class="close transition pl-3">\
                                      <span aria-hidden="true">&times;</span>\
                                    </button></li>');
})

$('#test').click(() =>{
    if(tests.length == 0){
      $('#testList').empty()
    }

    tests.push({"input" : $('#input').val(), "output" : $('#output').val(), "accepts" : $('#accepted').prop('checked')});

    $("#testList").append('<li class="list-group-item text-center">' +
                                    $('#input').val() + "->" +
                                    $('#output').val() + ': ' +
                                    $('#accepted').prop('checked') +
                                  ' <button type="button" class="close test pl-3">\
                                      <span aria-hidden="true">&times;</span>\
                                    </button></li>');

});

$('#subroutine').click(()=>{
    if(customsubroutines.length == 0){
      $('#subroutineList').empty()
    }

    var subroutine = preloadedSubroutines[$(select).val()]
    if(subroutine){
      $("#subroutineList").append('<li class="list-group-item text-center">' +
                                    subroutine.name +
                                  ' <button type="button" class="close subroutine pl-3">\
                                      <span aria-hidden="true">&times;</span>\
                                    </button></li>');
      customsubroutines.push(subroutine)

      const fixsub = new Subroutine(50 + (customsubroutines.length-1)*170, 50,0, subroutine.name,JSON.parse(subroutine.program),subroutine.initialState,subroutine.finalStates)
      fixsub.svg.node.onclick = function(){
        var subroutine = new Subroutine(500,500,subroutines.length,fixsub.name,fixsub.program,fixsub.initialState,fixsub.finalStates)
        subroutine.drag()
        subroutines.push(subroutine)
      }
    }
```

```
256
257   });
258
259   $('#createSubroutine').click(()=>{
260     compileTM();
261     var tmObj = compiledtm.export()
262     tmObj.name = $('#name').val();
263
264     $('#select').append('<option value="'+ preloadedSubroutines.length +'">'
265                         $('#name').val() +
266                         '</option>')
267     preloadedSubroutines.push(tmObj);
268     console.log(tmObj)
269   })
270
271   $('#submit').click(() =>{
272
273     var json = {
274     "goal": $('#goal').val(),
275     "transitions": customtransitions,
276     "tests": tests,
277     "subroutines": customsubroutines
278     }
279
280     $.ajax({
281       url: '/creator',
282       type: 'POST',
283       dataType: 'json',
284       data: json,
285     })
286     .done(function(id) {
287       console.log(id);
288       $( location ).attr("href", 'levels/'+id);
289     })
290     .fail(function() {
291       console.log("error");
292     })
293     .always(function() {
294       console.log("complete");
295     });
296
297     return json;
298   })
299
300   //remove transiton
301   $(document).on('click', '.transition.close', function() {
302     var item = $(this).parent()
303     var index = item.parent().children().index(item)
304
305     for(var j = 0; j < customtransitions[index].quantity; j++){
306       globalTransitions[index][j].svg.remove()
307       globalTransitions[index][j] = null
308     }
309
310     globalTransitions.splice(index,1)
311     customtransitions.splice(index,1);
312
313     //redraw transions
314     for (var i = index; i < globalTransitions.length; i++) {
315
316       for(var j = 0; j < customtransitions[i].quantity; j++){
317         globalTransitions[i][j].svg.remove()
318         globalTransitions[i][j] = new Transition(1600 + 150*(i%2), 200 +((i+1)
                %2 + i)*45, customtransitions[i].read, customtransitions[i].write,
                 customtransitions[i].move,j)
319       }
320     }
321
322     if(customtransitions.length == 0){
323       item.parent().append('<li class="list-group-item text-center" id="
              transitionListPlaceholder">No Transitions</li>')
324     }
325     $(item).remove()
326
327   });
328
329
330   //remove test
331   $(document).on('click', '.test.close', function() {
332     var item = $(this).parent()
333     var index = $('#testList').children().index(item)
334     tests.splice(index,1);
335     if(tests.length == 0){
336       item.parent().append('<li class="list-group-item text-center" id="
              testListPlaceholder">No Tests</li>')
337     }
338     $(item).remove()
339   });
340
341   //remove subroutine
342   $(document).on('click', '.subroutine.close', function() {
343     var item = $(this).parent()
344     var index = $('#subroutineList').children().index(item)
345     customsubroutines.splice(index,1);
346     if(customsubroutines.length == 0){
347       item.parent().append('<li class="list-group-item text-center" id="
              subroutineListPlaceholder" >No Subroutines</li>')
348     }
349     $(item).remove()
350   });
351 });
```

A.9 level.js

```
1  var canvas = Snap(document.getElementById('canvasSVG'));
2  var tapeCanvas = Snap(document.getElementById('tapeSVG'));
3  var tapeobj
4  var states = [];
5  var subroutines = [];
6  var globalTransitions = [];
7  var tapeString = "010101"
8  var compiledtm;
9  var inputAlphabet;
10 var currentTransition;
11 var blank = "B";
12 var tests = []
```

```

function init(){
  tapeobj = new Tape(tapeString);
  for (var i = json.transitions.length - 1; i >= 0; i--) {
    var arrayOfTransitions = []
    for(var j = 0; j < json.transitions[i].quantity; j++){
      arrayOfTransitions.push(new Transition(1600 + 150*(i%2), 200 +((i+1)%2 +
          i)*45, json.transitions[i].read, json.transitions[i].write, json.
          transitions[i].move,j));
    }
    globalTransitions[i] = arrayOfTransitions;
  }
  for (var i = json.subroutines.length - 1; i >= 0; i--) {
    const fixsub = new Subroutine(50 + i*170, 50,0, json.subroutines[i].name,
        JSON.parse(json.subroutines[i].program),json.subroutines[i].
        initialState,json.subroutines[i].finalStates,tapeobj)
    fixsub.svg.node.onclick = function(){
      var subroutine = new Subroutine(500,500,subroutines.length,fixsub.name,
          fixsub.program,fixsub.initialState,fixsub.finalStates,tapeobj)
      subroutine.drag()
      subroutines.push(subroutine)
    }
  }
  for (var i = json.tests.length - 1; i >= 0; i--) {
    tests.push(json.tests[i]);
  }
  $('#goal').empty()
  $('#goal').append('<h3>'+json.goal+'</h3>')

  // Mobile
  $('#canvasSVG').on('touchmove', function(e){e.preventDefault()});


  var fixedState = new State(1700, 100);
  fixedState.clearTxt();
  var state = new State(100,440,states.length);
  state.setInitialState();
  state.drag();
  states.push(state);

  fixedState.svg.node.onclick = function () {
    var state = new State(1500,100,states.length);
    state.drag();
    states.push(state);
  };

}

function clearCanvas(){
  canvas.clear();
  tapeCanvas.clear()
  states = [];
  compiledtm = 0;
  init();
}

function step(){
  if(!compiledtm){
    compileTM()
  }
  compiledtm.step()
}

function reverseStep(){
  if(compiledtm){
    compiledtm.unstep()
  }
}

function playClicked(){
  if(compileTM()){
    compiledtm.run()
  }
}

function submit(){
  compileTM()
  $('#alerts').empty()
  for (var i = 0; i < tests.length; i++) {
    if(!compiledtm.verify(tests[i].input,tests[i].output,tests[i].accepts)){
      $('#alerts').append(`<div class="alert alert-danger alert-dismissible
          fade show" role="alert">\
        <strong>Failed Test number ${i}!</strong>\
         Try input ${tests[i].input}\
        <button type="button" class="close" data-dismiss="alert" aria-label="
            Close"><span aria-hidden="true">&times;</span></button></div>`)
      return 0;
    }
  }
  $('#alerts').append(`<div class="alert alert-success alert-dismissible fade
      show" role="alert">\
        <strong>Passed all tests!</strong>\
        <button type="button" class="close" data-dismiss="alert" aria-label="
            Close"><span aria-hidden="true">&times;</span></button></div>`)
}
function compileTM(){
  $('#alerts').empty()
  var transitions = {}
  var finalStates = []
  var initialState = "";
  var run = 1;
  var finalTransWarning = []

  //create the transtion table
  for (var i = 0; i < states.length; i++) {
    //fetch the current states transtions
    transitions[states[i].n] = states[i].getTransitions();

    //find the inital and final states
    if(states[i].isFinalState()){
      finalStates.push(states[i])
      if(states[i].getTransitionsIn().length == 0){
        finalTransWarning.push(i+1)
      }
    }
    if(states[i].isInitialState()){
      initialState = states[i]
      if(states[i].getTransitionsOut().length == 0){
        $('#alerts').append(`<div class="alert alert-warning alert-dismissible
            fade show" role="alert">\
```

```
            <strong>Warning: No transitions from inital state ${i}!</strong>\
            <button type="button" class="close" data-dismiss="alert" aria-label="
                Close"><span aria-hidden="true">&times;</span></button></div>')
        }
      }
    }

    //get subroutines states
    for (var i = 0; i < subroutines.length; i++) {
      transitions[subroutines[i].n] = subroutines[i].getTransitions();
    }

    // Display warnings
    if(finalTransWarning.length){
      if(finalStates.length == 1){
        $('#alerts').append('<div class="alert alert-warning alert-dismissible
             fade show" role="alert">\
            <strong>Warning: No transitions to final state ${finalTransWarning.
                toString()}!</strong>\
            <button type="button" class="close" data-dismiss="alert" aria-label="
                Close"><span aria-hidden="true">&times;</span></button></div>')
      }else if (finalStates.length == 2){
        $('#alerts').append('<div class="alert alert-warning alert-dismissible
             fade show" role="alert">\
            <strong>Warning: No transitions to final states ${finalTransWarning.
                join(' and ')} !</strong>\
            <button type="button" class="close" data-dismiss="alert" aria-label="
                Close"><span aria-hidden="true">&times;</span></button></div>')
      }else{
        $('#alerts').append('<div class="alert alert-warning alert-dismissible
             fade show" role="alert">\
            <strong>Warning: No transitions to final states ${finalTransWarning.
                slice(0, finalTransWarning.length-1).join(', ')} and ${
                finalTransWarning[finalTransWarning.length-1]}!</strong>\
            <button type="button" class="close" data-dismiss="alert" aria-label="
                Close"><span aria-hidden="true">&times;</span></button></div>')
      }
    }
    //console.log(transitions)
    // no final states
    if(finalStates.length == 0){
      $('#alerts').append('<div class="alert alert-danger alert-dismissible fade
           show" role="alert">\
        <strong>Error: no final states!</strong>\
         You should add one by double clicking a state!.\
        <button type="button" class="close" data-dismiss="alert" aria-label="
            Close"><span aria-hidden="true">&times;</span></button></div>')
    }
    if(run){
      compiledtm = new TuringMachine(transitions,initialState,finalStates,
          tapeobj)
      return compiledtm;
    }else{
      return 0;
    }

}

function tapeinput(){
  tapeString = document.getElementById("tapeinput").value;
  tapeCanvas.clear()
  tapeobj = new Tape(tapeString);
  compiledtm = 0;
}
function fastforward(){
  if(compiledtm){
    compiledtm.skip();
  }else if(compileTM()){
    compiledtm.skip()
  }
}
function reverseskip(){
  if(compiledtm){
    compiledtm.unskip();
  }else if(compileTM()){
    compiledtm.unskip()
  }
}
init();
```

Appendix B

Ethics check list

UNIVERSITY OF BATH

Department of Computer Science

13-POINT ETHICS CHECK LIST

This document describes the 13 issues that need to be considered carefully before students or staff involve other people ("participants") for the collection of information as part of their project or research.

1. **Have you prepared a briefing script for volunteers?**
   You must explain to people what they will be required to do, the kind of data you will be collecting from them and how it will be used.

   Briefing scripts have been prepared for all volunteers on at the start of the questionnaire

2. **Will the participants be using any non-standard hardware?**
   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen and paper or typical interaction with PCs on desks is considered non-standard.

   Participants will be using their own hardware, no-standard hardware is used

3. **Is there any intentional deception of the participants?**
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

   There is no deception as the briefing scripts describes the study and what the data is used for

4. **How will participants voluntarily give consent?**
   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

Consent will be given by completion of the questionnaire which is stated to the participants in the briefing script. Signed consent is not needed as the results of the evaluation are to be contained within the project

5. **Will the participants be exposed to any risks greater than those encountered in their normal work life?**
   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

   There are no risks involved, as it only involves playing a serious game

6. **Are you offering any incentive to the participants?**
   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

   No incentive is offered to the participants

7. **Are any of your participants under the age of 16?**
   Parental consent is required for participants under the age of 16.

   All participants are over the age of 16

8. **Do any of your participants have an impairment that will limit their understanding or communication?**
   Additional consent is required for participants with impairments.

   No participants have impairments

9. **Are you in a position of authority or influence over any of your participants?**
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

   Participants are free to leave to study at any point, and are not pressured to take part in the questionnaire

10. **Will the participants be informed that they could withdraw at any time?**
    All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

    Participants will be informed they can withdraw from the study at any point during the study without reason

11. **Will the participants be informed of your contact details?**
    All participants must be able to contact the investigator after the investigation. They should be given the details of the Unit Lecturer or Supervisor as part of the debriefing.

    Participants will be provided contact details on the de-briefing script

12. **Will participants be de-briefed?**
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation.

All participants will be provided a debrief on the purpose of the study when the questionnaire is submitted

13. **Will the data collected from the participants be stored in an anonymous form?**
    All participant data (hard copy and soft copy) should be stored securely, and in anonymous form. *All data collected will be collected anonymously through the use of Google Forms*

Appendix C

Questionnaire

Interactive Turing Machines

By participating in is questionnaire, you agree for your response to be recorded and used within this project.

Go to https://lit-crag-54251.herokuapp.com/

Read what a Turing machine is and how to play
Go to Created Levels
Select “Goal: Create a Turing machine which inverses a binary word”
Try to make a Turing Machine that meets that goal.
Then go to the level ”Goal: Create a binary incrementer” and try that level

Do not worry if you cannot do it or encounter bugs, this is the purpose of this questionnaire

Once done, please continue to the next section

On a scale of 1 to 5, rate the accuracy of the following

It was easy to achieve the goal set out

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

The interface does not present itself in a very attractive way

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

I enjoyed the time I spent creating the Turing machine

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

It was difficult to create a Turing machine

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

I learnt more about Turing machines and how they function

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

The error messages are not adequate

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

I did not need to know about any mathematics while performing the task

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

Subroutines did not help me understand how a Turing machine works

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

Optional questions

What aspects of the game did you like?

What aspects of the game did you dislike?

What would you do to improve the game?

Any extra comments?

_____
_____
_____
_____

Debrief

Thank you for completed this questionnaire, Your response has been recorded and will be used to improve the serious game, you have the right to remove your response by contacting either Kieran Warwick at kw510@bath.ac.uk or Willem Heijltjes at wbh22@bath.ac.uk.

Appendix D

Questionnaire results

This shows the raw results of the questionnaire, which was provided by Google Forms.

D.1 Stage two results

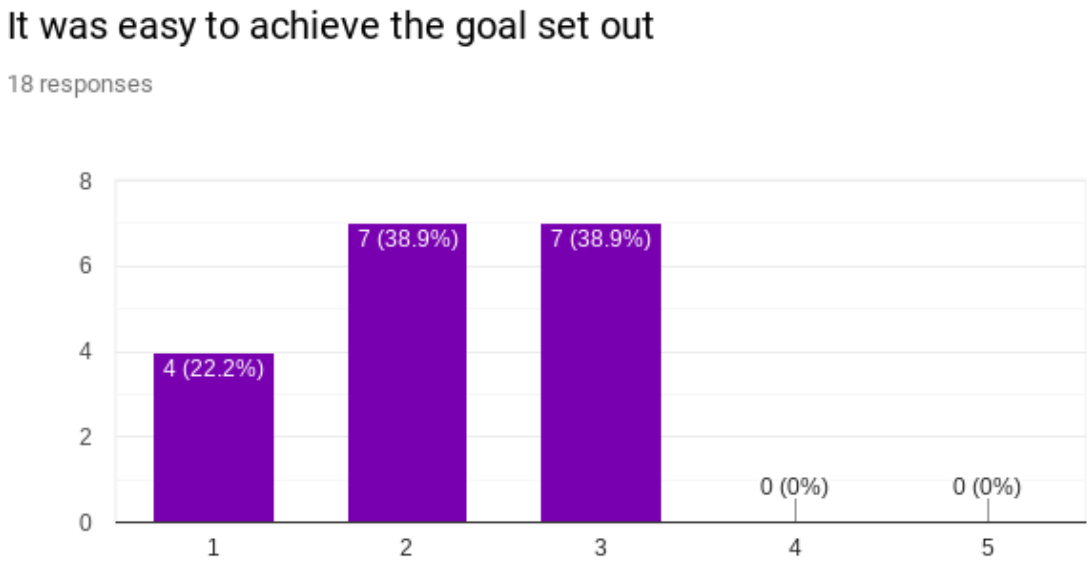

Figure D.1: Question 1

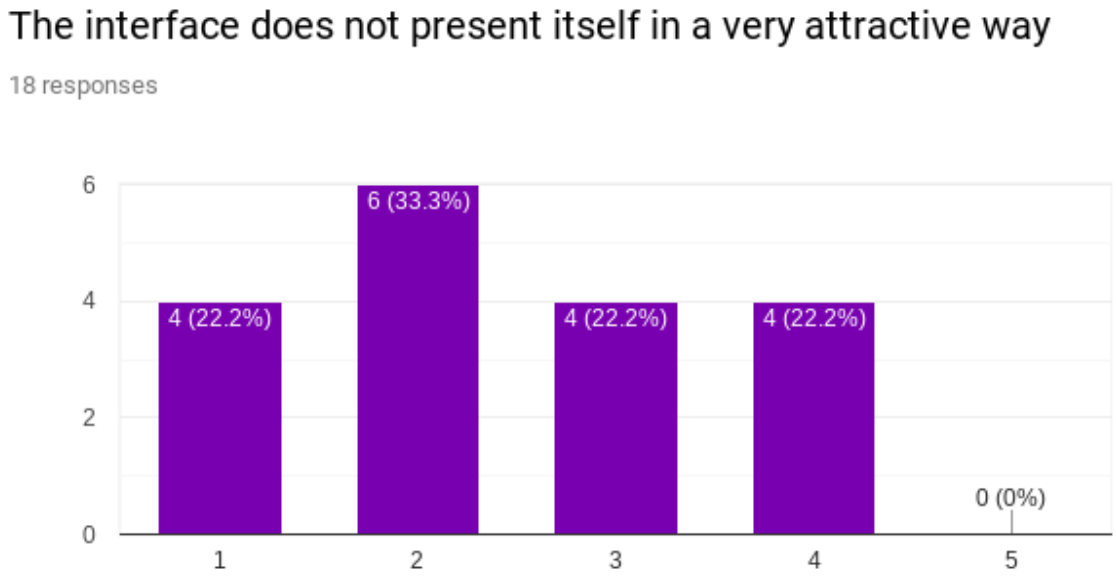

Figure D.2: Question 2

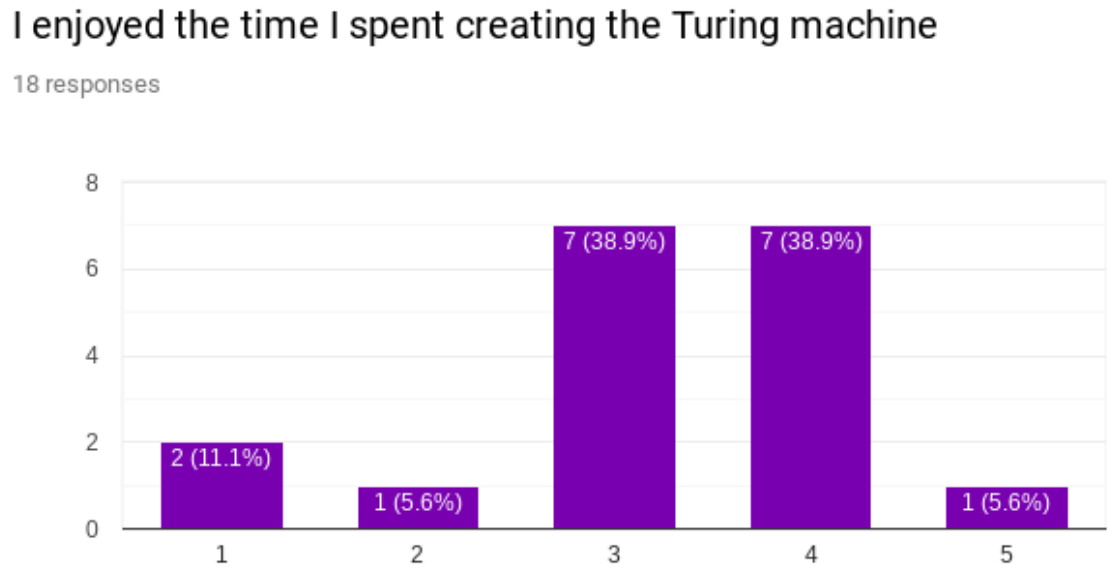

Figure D.3: Question 3

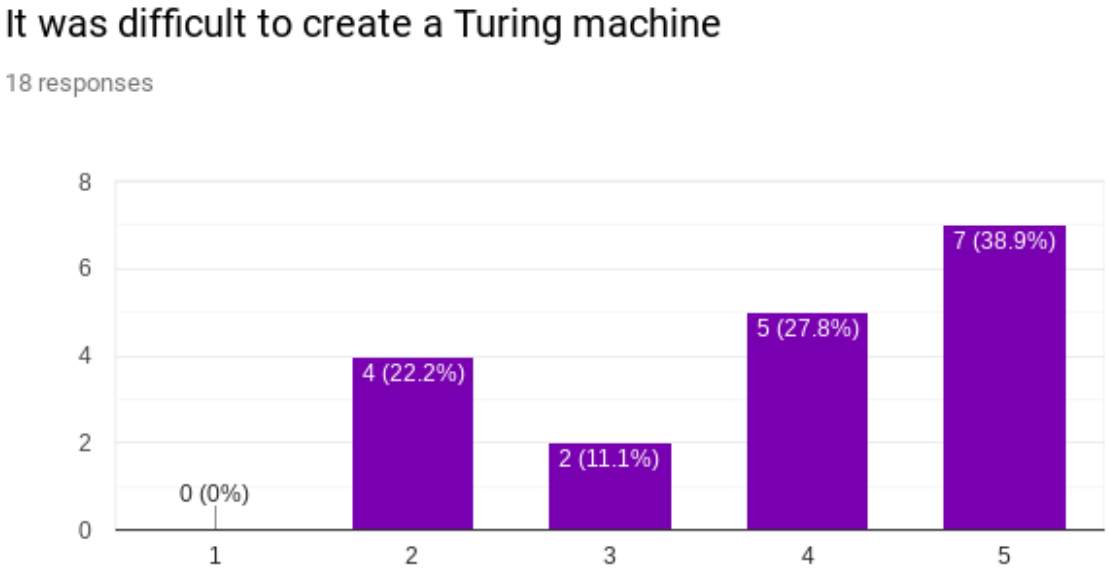

Figure D.4: Question 4

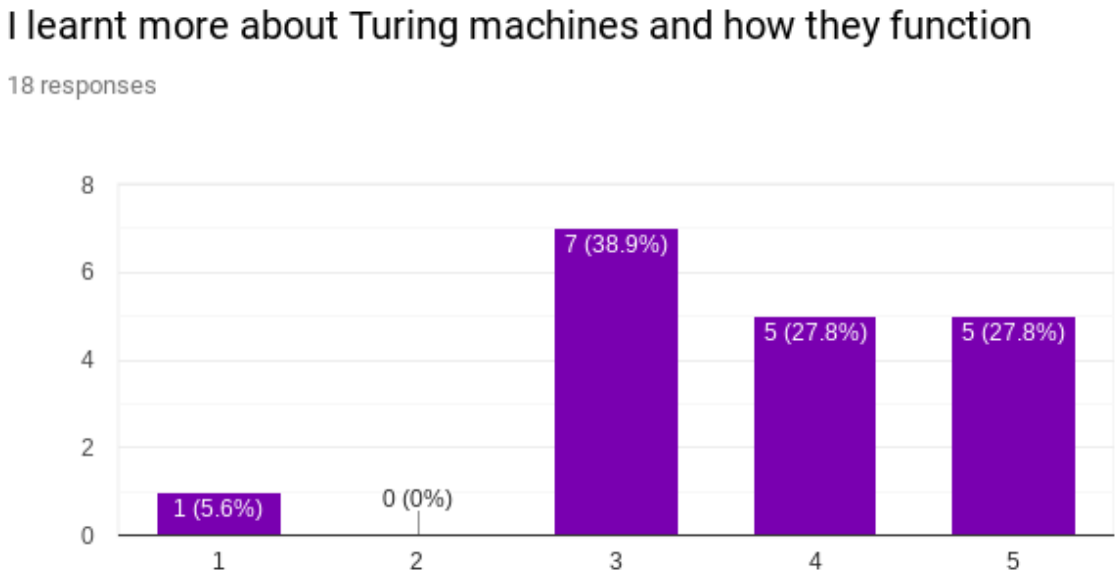

Figure D.5: Question 5

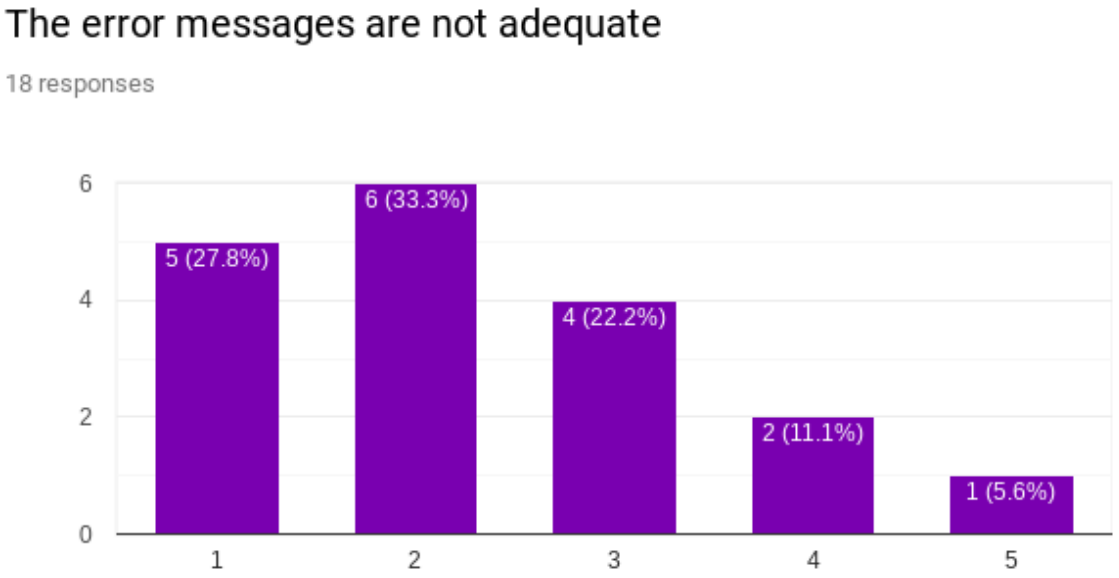

Figure D.6: Question 6

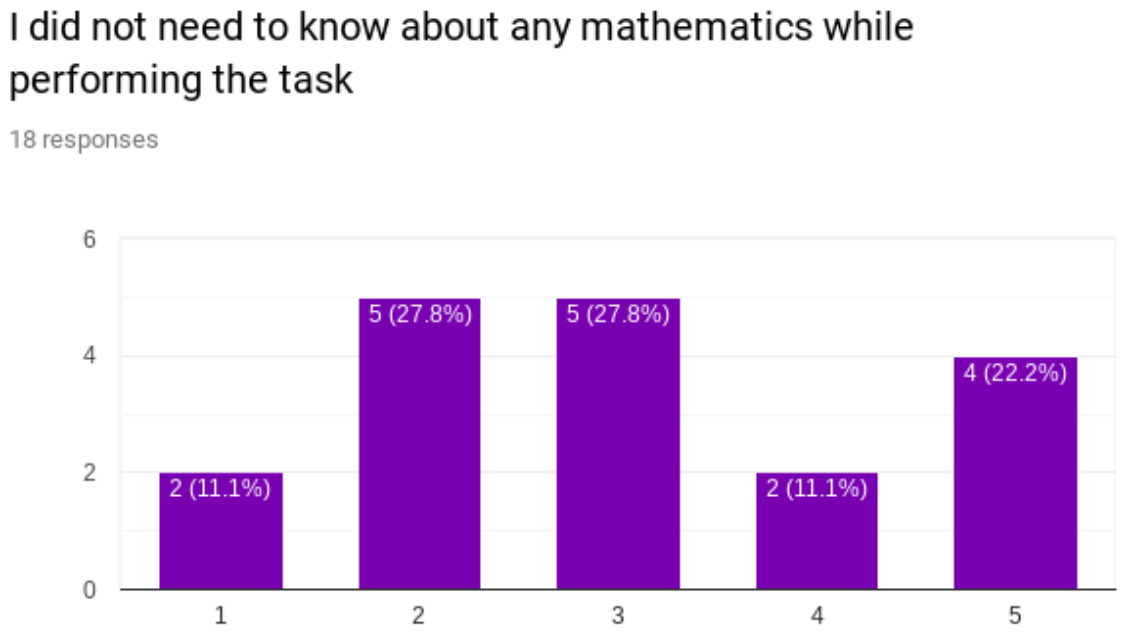

Figure D.7: Question 7

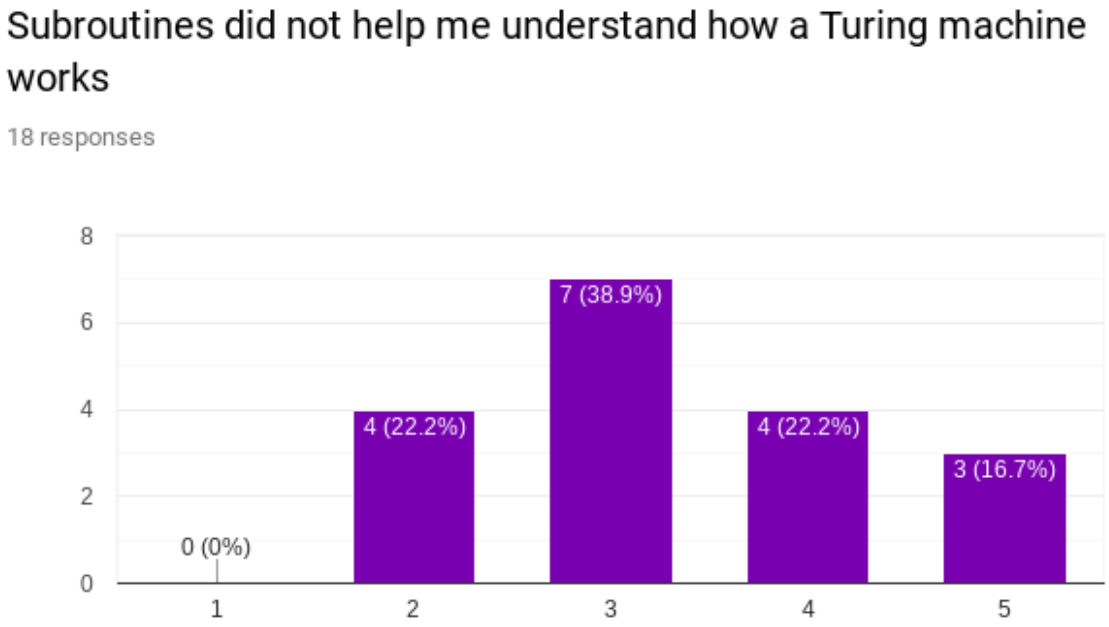

Figure D.8: Question 8

D.2 Stage three results

D.2.1 What aspects of the game did you like?

- Coding is nice, and it had a solid visual element.
- Easy see what happens
- Instructions were on the page to refer to
- I liked creating the Turing machine once i grasped the concept of how they work
- Learning about Turing Machines
- The ease of instructions given.
- Pressing the buttons
- cool gramphics
- Very clear layout, clean design.
- how to connect the arrows and put it all together
- Looked nice
- The flow chart layout of how the created machine 'thinks' is really nice. Once a machine has been made you can easily follow the steps it will take when it follows the tape.
- I liked the error messages as they were helpful
- Playing around with the tape and the inputs

D.2.2 What aspects of the game did you dislike?

- Hard to use the interface, and the subroutines felt like they should have been personally programmable, or explained that they weren't.
- The letters were unclear without prior prompting
- Couldn't play the game as didn't know what binary is
- The explanations about how they worked
- Assigning transitions was difficult
- Trying to make the Turing machine.
- Not understanding what I was trying to do
- the dragging stuff is confusing
- Complicated, and unsure of the purpose.
- the labels on the arrows weren't clear on what they did
- Didn't know what any of the buttons meant
- It took some guessing to figure out how to remove an arrow once it was placed. Also some things were not explained very well, like what 'B' means in a read:write direction arrow. It was also not clear what to expect from the subroutines and how to include them, so that also required some trial and error before it clicked.
- For me personally it wasn't entirely clear how to make a transition
- The UI

D.2.3 What would you do to improve the game?

- Work on the description of things and the available blurb when playing. eg "You must finish on an end state", "RHS moves the Turing machine to the right hand side of the tape."
- Add some description to what each letter means and what some of the levels mean e.g saying level 2 adds a 1 at the end
- Simpler language or better description on what a binary is
- "Explain a little more clearly how to put together a turing machine, maybe include a guided example of creating one, even just using like images.
- I dont get the subroutines, youve put what seem to be acronyms but havent stated what they are or do"
- States/transitions to highlight when selected
- No idea, I'm not smart enough to understand how to create one.
- Make a simple beginners tutorial saying 'click this to do this'
- improve the dragging

- Not sure, as I wouldn't know what the objective is.
- better explanation on what every hing does, or a tutorial level
- A tutorial would go a long way towards making it all really clear for somebody who doesn't know much about bitwise computing, perhaps a guided tour of the game designer solving one of the levels with some text to explain each step.
- Add color
- Make the error messages tell you what is wrong rather than just saying Fail and try another input

D.2.4 Any extra comments?

- My arrows to their desricptions didn't align...
- Forced tutorial due to laborious instructions on how to play
- Impressive work. Well done!
- the dragging of the arrows needs improvement
- cool stuff
- n/a
- Explain what the letters in the diamonds stand for and what a subroutine is