

Programable Turing Machine with Python

Ashley Brindle | Thesis | 16022599



March 22, 2019

mCOMP COMPUTER SCEINCE

CS3 D661

Appendix I

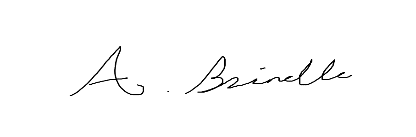
**STATEMENT OF ORIGINALITY**

**CS3D661 Individual Project**

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerized sources have been fully referenced, and I fully realize the consequences of plagiarizing any of these sources.

Student Name (Printed) \_ASHLEY BRINDLE\_\_\_\_\_\_\_\_

Student Signature 

Registered Course of Study \_MComp Computer Science\_

Date of Signing \_22nd March 2019\_\_\_\_\_\_\_\_\_

# Abstract

This project will act to inform on Alan Turing, his work, and implementation of a programmable Turing machine. Alan Turing was a British mathematician who made great advancements in computer science, One of main achievements was the Turing machine. The Turing machine is essentially a computer and is considered by most as the first computer, it uses a tape and instructions to determine results depending on what was provided. The machine will be provided with a tape which is seen as infinitely long, when provided with this tape the machine will run through instructions and change each cell on the tape depending on the current state and symbol the machine is in. This will eventually provide a completed tape which will essentially give an answer to the provided tape. The machine will do this by using a tape head which will first check the state of the machine and the symbol read on the tape, look at the instructions, change the tape to the desired symbol and change the current state of the machine and then move in the desired direction, this process will then repeat until a final tape is produced. The implementation of a Turing Machine was done through using a state pattern and making several different finite Turing Machines, these machines will be programmable by allowing the user to select a tape to run. Another machine is implemented showing a Dynamic machine allowing the user to write their own tape, instructions, starting state, and tape position, to make the machine fully dynamic.

Contents

[Abstract 2](#_Toc4182524)

[Project Aims and Objectives 2](#_Toc4182525)

[Background Research 5](#_Toc4182526)

[Sources 5](#_Toc4182527)

[Python 5](#_Toc4182528)

[Python Libraries 5](#_Toc4182529)

[Why Python? 6](#_Toc4182530)

[Alan Turing 6](#_Toc4182531)

[The Turing Machine 6](#_Toc4182532)

[Types of Turing Machine 8](#_Toc4182533)

[Universal Turing Machine 9](#_Toc4182534)

[The Invention of the Computer 9](#_Toc4182535)

[Methodology 10](#_Toc4182536)

[Design 10](#_Toc4182537)

[Design 1 11](#_Toc4182538)

[Design 2 13](#_Toc4182539)

[Pseudo Code v1: 14](#_Toc4182540)

[Pseudo Code v2: 15](#_Toc4182541)

[Pseudo code v3: 16](#_Toc4182542)

[Pseudo Code v4: 17](#_Toc4182543)

[Implementation 18](#_Toc4182544)

[Prototype 1 18](#_Toc4182545)

[Prototype 2 21](#_Toc4182546)

[Prototype 3 24](#_Toc4182547)

[Libraries and GUI 26](#_Toc4182548)

[Next Steps 27](#_Toc4182549)

[My Plan 27](#_Toc4182550)

[Timeline 28](#_Toc4182551)

[Appendix 30](#_Toc4182552)

[References 30](#_Toc4182553)

[Prototype 1 32](#_Toc4182554)

[Prototype 2 46](#_Toc4182555)

[Prototype 3 63](#_Toc4182556)

[State.py 63](#_Toc4182557)

[TuringMachine.py 63](#_Toc4182558)

[Ethics Checklist 67](#_Toc4182559)

# Introduction

This project will aim to discuss the advancements made by Alan Turing in computer science and how the Turing machine could be implemented using Python. This is of interest as Alan Turing is considered one of the founding fathers of computer science and it is interesting how his advancements shaped modern computing. This impact of this project would be to give the reader a in depth knowledge of Alan Turing and his work and will also aim to discuss how a programmable model can be made. One of the main challenges which will be discussed in this report would be what exactly makes the machine programmable and how a programmable version can or may be implemented

## Project Aims and Objectives

The aim of this project is to provide a programmable Turing Machine using the python programming language. This project should conclude with a programmable Turing machine with extra features to make mine unique, as well as have a deep understanding of python programming language and its existing libraries

Main objectives for this project are as follows:

* Background research on Alan Turing’s 1936 paper *on Computable Numbers, with an Application to the Entscheidungsproblem*
* A demonstration on how the Turing machine operates and how each of the components interact with each other
* Understand and describe Alan Turing’s contribution to Computer Science
* Research various types of Turing machines and how they work
* Add extra features to the Turing machine to allow my product to stand out
* To produce a dynamic Turing machine allowing the user to fully customise the machine
* Adding advanced features to the Turing machine to make it more informative to the user

# Background Research

## Sources

The background research for this project will cover both Python and the Turing machine, explaining in detail how the Turing machine works, the different types of machine and the python programming language, with associated useful libraries. The main source of research would be the *Annotated Turing* book by Petzold (2008), which gives a deep analysis of the Turing machine and the paper by Turing (1936) named *On Computable Numbers with an Application to the Entscheidungsproblem.* Other sources will also be used ranging from academic papers, books and websites to support any points given.

## Python

### Python Libraries

There are many types of Python libraries, ranging from libraries to help with data management to GUI implementation; GUI libraries will be the main focus as the libraries will only act as a foundation to make the product more presentable. Python has a wide range of libraries to chose from when it comes to GUI programming, some libraries come preinstalled with my python install and some will be installed manually

#### Tkinter

One python library to review would Tkinter; Tkinter is a GUI library which is pre-installed with the language, this will not require any installation from an external source, making it easier to use. Tkinter is considered one of the most popular python GUI libraries due to this point and its simplicity, this makes Tkinter a good introduction to GUI programming in Python. Tkinter is a very easy language to learn, although there is no visual designer to help with the implementation, it is however very easy to get a grasp on the techniques used to manage the window space.

Tkinter also has very detailed documentation which is easy to follow, allowing anyone to get a grasp on anything they need to know about the library and how to use it (Python 2018). Although Tkinter may be the easiest GUI library to access, it is not the best GUI tool to as said by Chun (2006). This may make Tkinter the “go-to” python GUI library for beginners by default, giving a perfect introduction to GUI programming. As stated by Lutz (2014) Tkinter is a very simple and popular language with a lot of benefits such as documentation and its portability.

#### WxPython

WxPython is another Python GUI library and is a common alternative to Tkinter which gives the ability to have a python GUI native on common operating systems such as Windows (Dunn, 2018). wxPython is an adaption of the wxWidgets cross platform C++ library. Similar to Tkinter, wxPython is well documented by providing all of the relevant information on how to use the various widgets. You can use the wxWidgets documentation to help create a GUI, however most of this documentation is written in C++, therefore requiring a broad knowledge of python to convert the syntax. As suggested by (Dunn, 2017) wxPython uses many different classes which will mirror most wxWidget classes, this allows the use of the wxWidgets documentation to be adapted to python as needed. As well as detailed documentation, wxPython provides a wide variety of classes to choose from, giving heavy customisability in your GUI application.

#### Kivy

Another Python GUI library to consider would be Kivy; Kivy is an open source Python GUI library with a wider range of customisability in comparison to other libraries discussed already. Kivy can be used to create apps from many different platforms, including different mobile operating systems such as IOS and android, allowing you to write one piece of code with compatibility with many different device operating systems (Kivy, 2018). Kivy also provide a well-documented guide on how to install and use Kivy, there is also a Python Programming guide included for users which have a vague understanding on the language. The library provides many different features such as using Widgets, drawing, managing layouts and graphics; this makes this library an excellent choice to consider when picking a Python GUI library for my project. This library is not installed with python, the various steps on their installation guide will help anyone install the library. Kivy provides the option to design your program with forms not limited to the windows-based form giving strong customisability when it comes to design.

#### PyQt

One other Python GUI library would be PyQt. PyQt is a popular python library which has been adapted from a C++ framework. PyQt also covers a wide range of classes which can be used to create GUI’s, handle networking, databases and more (Python, 2018). PyQt has a wide range of documentation for both versions PyQt4 and PyQt5, as well as books to help create GUI’s (Summerfield, 2008). The Qt library provides a Designer which can be used to help design and manage my GUI application, this is an easier method than using any of the other libraries mentioned thus far as there is the option to not code the positioning of each of the widgets and graphics on the window. However, this will still require management of receiving and sending of data by each widget (Qt Company Ltd, 2018).

### Why Python?

As part of my requirements for the final year project, python is a very easy language to use which some people may consider as simple as pseudo code. As well as being one of simplest programming languages to learn, python can be considered as one of the most useful languages with heavy use in computer networking. Python is one of the top 10 programming languages used and has many different uses; as stated by the TIOBE (2018) index, such as development in GUI applications, web development and data analysis according to Mindfire Solutions (2017). Python also has a wide range of libraries and a popular website such as PyPi provides easy access to a variety of libraries. Python can also perform high level programming tasks in significantly less lines of code compared to languages such as C++. Although python may be an easy language to use, it may not be your best choice depending on your task; however, in this case the language is efficient for the task of building a programmable Turing machine.

## Alan Turing

Alan Turing was an English mathematician who studied at Princeton University. One of Alan Turing’s major contributions to computer science was his paper named “*On Computable Numbers, with an application to the Entscheidungsproblem*” which is considered fundamental to modern computer science. This paper would cover the famous Turing machine and his Universal Machine which would in theory solve any computable problem by following a set of instructions (Watson, 2012).

### Church-Turing Thesis

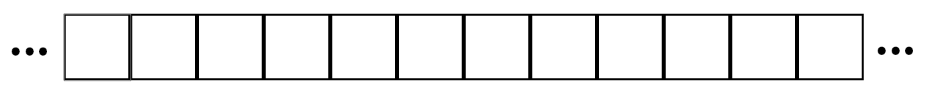
Both Alonzo Church and Alan Turing had very similar assumptions when it came to a model of computability. Alonzo Church used Lambda calculus while Alan Turing used the Turing machine; what was special about this was that any problem present to Lambda calculus, can also be solved by the Turing machine making both models equivalent.

### The Turing Machine

As Stated above the Turing machine is a machine which will be able to follow any set of instructions (Also known as an Algorithm) to perform various computable tasks. The standard Turing machine is capable of writing down a series of 1s and 0s onto a tape with squares, depending on the instructions based on each symbol the machine reads. The machine will also have a “head” to read through each square on the tape to gather its symbol, the machine would then use an instruction table to refer to before performing a task then advancing to the next square on the tape, (Mullins, 2012).

The Turing machine would work by being provided with a tape with a series of symbols (this tape could be infinite), the machine would also be provided with a series of instructions with different states and tasks (Instruction Table). The machine would then have to flow through the tape, read the symbols, then complete the task provided by the instruction table (depending on the symbol read on the tape). An example of a Turing Tape and instruction table will be provided below:

|  |  |  |  |
| --- | --- | --- | --- |
| **CONFIGURATION** | | **BEHAVIOUR** | |
| ***m-configuration*** | ***scanned symbol*** | ***operation*** | ***final m-configuration*** |
| A | NONE | P0, R | B |
| B | NONE | R | C |
| C | NONE | P1, R | D |
| D | NONE | R | A |



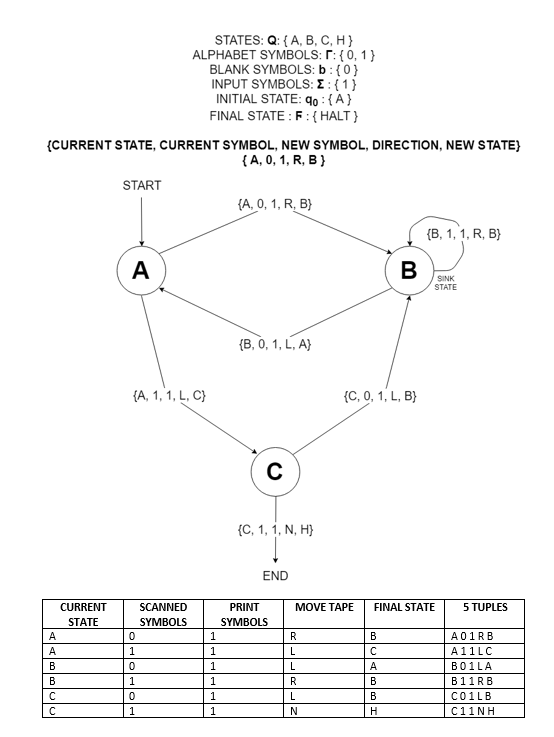
The Instruction table above provides an easy and simple to follow instruction set which will print an endless tape of 0s and 1s on the Turing tape. The machine would first check what m-configuration (state) the machine is in and then check if the scanned symbol matches the one presented on the current tape square, the machine will then perform the operation associated with the scanned symbol and the current state, then change to another m-configuration and perform the operation. The following symbols can be used in operation to tell the machine what to do next; the machine can erase the current square (**E**), Print a symbol (**P** followed by the symbol you wish to print (**P***n*)), and change direction (either Left or Right (**L** , **R**)). The machine can also take multiple instructions in its operation such as moving in any direction or print symbols more than once; this can be shown in the following simplified instruction table, Petzold (2008).

In Summary:

1. Check the m-configuration of the machine
2. Checks the scanned symbol on the tape
3. Reads the instruction table and acts accordingly to the scanned symbol and the current m-configuration
4. The machine performs the action on the given tape
   1. In this case either print a symbol or erase a symbol
5. The machine will then move the head in the given direction
   1. In this case; Left or Right
6. Repeat from (1)

|  |  |  |  |
| --- | --- | --- | --- |
| **CONFIGURATION** | | **BEHAVIOUR** | |
| ***m-configuration*** | ***scanned symbol*** | ***operation*** | ***final m-configuration*** |
| A | NONE | P0 | A |
| 0 | R, R, P1 | A |
| 1 | R, R, P0 | A |

The head on the machine will only be able to scan one square at a time and will only be mindful of that one square at any given moment. The machine will then act appropriately depending on the current m-configuration of the machine and the scanned symbol. The table will now include 6 columns, with details on: the current state, symbol scanned, printed symbol, tape direction, final state and its 5-tuples representation as shown below with a state diagram for the machine, Davis (1965).



### Types of Turing Machine

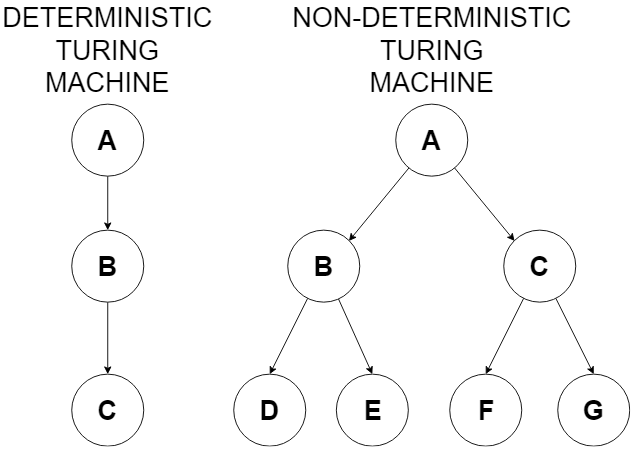
#### Definitions

* States:
* Alphabet Symbols:
* Operations:
* Transition Function:

#### Overview

Here are a few examples of different types of the Turing machine; all of which can be simulated on a single sided infinite Turing tape. One other variation of the Turing machine would be the multi-tape Turing machine, this machine is a standard Turing machine with several tapes. Each tape will have its own head which can operate individually to each other, initially the square on the first tape will have an input, whilst the other tapes are left blank as specified by Sipser (1997). Secondly, the multi-track Turing machine is a type of the multi-tape Turing machine as mentioned by Sudkamp (2006), while the machine has only one tape, the tape can hold multiple tracks. Each of these tracks are written to simultaneously, this does not make each track independent as the head will read and write to all tracks at the same time (Schafer, 2016).

Another type of Turing machine would be the non-deterministic Turing machine, Sipser (1997) suggests this machine will process in different ways depending on different possibilities. Imagine a tree of states, with each branch having multiple options in where the machine can go next. Each branch can either go into an accepting or rejecting state. If any branch enters a rejecting state at any given time, the rest of the machine will still operate; however, if a branch enters the accepting state, the entire machine will accept. The difference between the two types of machine are shown below, clearly showing how a machine what makes a Turing machine non-deterministic.



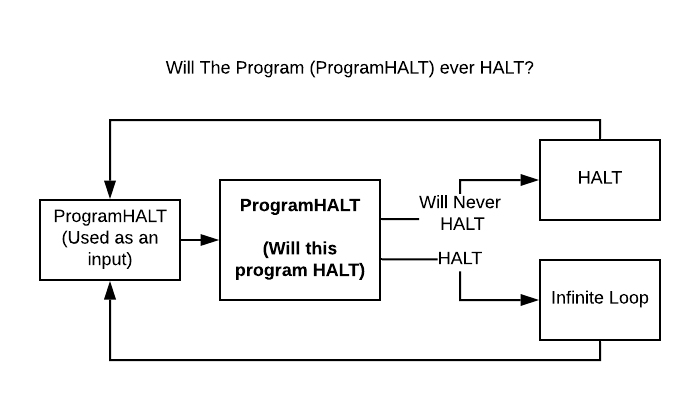
### Universal Turing Machine

After covering the basic Turing machine and the different types, the next step would be to move onto the Universal Turing machine. Petzold (2008) mentions The Universal Turing machine is the only Turing machine that should be needed, it can simulate other Turing machines once given their SD (Standard Description), this then defines the Universal Turing machine as programmable. Turing (1936) suggests in his paper o*n Computable Numbers, with an Application to the Entscheidungsproblem*, that it is possible to create a Turing machine to compute any computable sequence. The Universal Turing machine will have to be supplied with any input to operate, such as a tape containing the SD of the Turing machine the Universal Turing machine will simulate. Petzold (2008) also suggests that U (Universal Machine) cannot perfectly simulate any M (Turing Machine); this also suggests that Universal machine is not infinite, due to the SD having to be defined at the “beginning” of the tape (an infinite tape should not have a beginning).

### The Invention of the Computer

Turing’s Universal machine is quite similar to the modern computer in many ways, mainly that the machine can in theory run any algorithm, therefore being able to do all possible computations; in short, the machine will be able to do any task, given it is provided with an algorithm or program (Hom, 2013). The idea of a computable machine was originally formulated by Charles Babbage and his Analytical Engine, this became the first interpretation of a programmable computer which could be interpreted as Turing Complete suggested by (Graham-Cumming, 2018) (A machine which can be used to emulate a Turing machine (Flury, 2017)). However, Turing was first to incorporate this into his work with the Turing machine, which became a basis on what computation and programming is (Jackson, 2012). As Stated by Petzold (2008) the Turing machine is capable of any computations which can be done by the modern computer and that Turing provided an understanding on the limitations on what a computer can and cannot do. The Turing machine can be represented as the modern computer by thinking of the tape as a block of memory and the instructions being a program or algorithm for the computer to follow. Due to this, Turing gave a base understanding on what a programmable computer could be, which became the stepping stone to the modern-day computers; in summary the Universal machine could be interpreted as a modern-day computer.

### Entscheidungsproblem

The Entscheidungsproblem is a decision problem theorised by Hilbert Entscheidungsproblem which asks if there is an algorithm that can take a formal logic input and produce a true or false answer which is accurate stated by ScienceDirect (2013). Alan Turing stated the Entscheidungsproblem is unsolvable, Turing’s application of this was called the Halting problem which can be associated with the Turing machine by stating if the machine will either HALT or endlessly run. This problem was proved by Turing to be unsolvable by given a program which supposedly solves the Halting problem which will either infinitely loop if the program can HALT or to HALT if the program will not HALT and feeding the program into itself to effectively loop forever. This means that if the program can HALT, it will never HALT and if the program will never HALT, it will then HALT (Moore, Koswara and Williams, no date), This results in a paradox as shown in the image below.

# Methodology

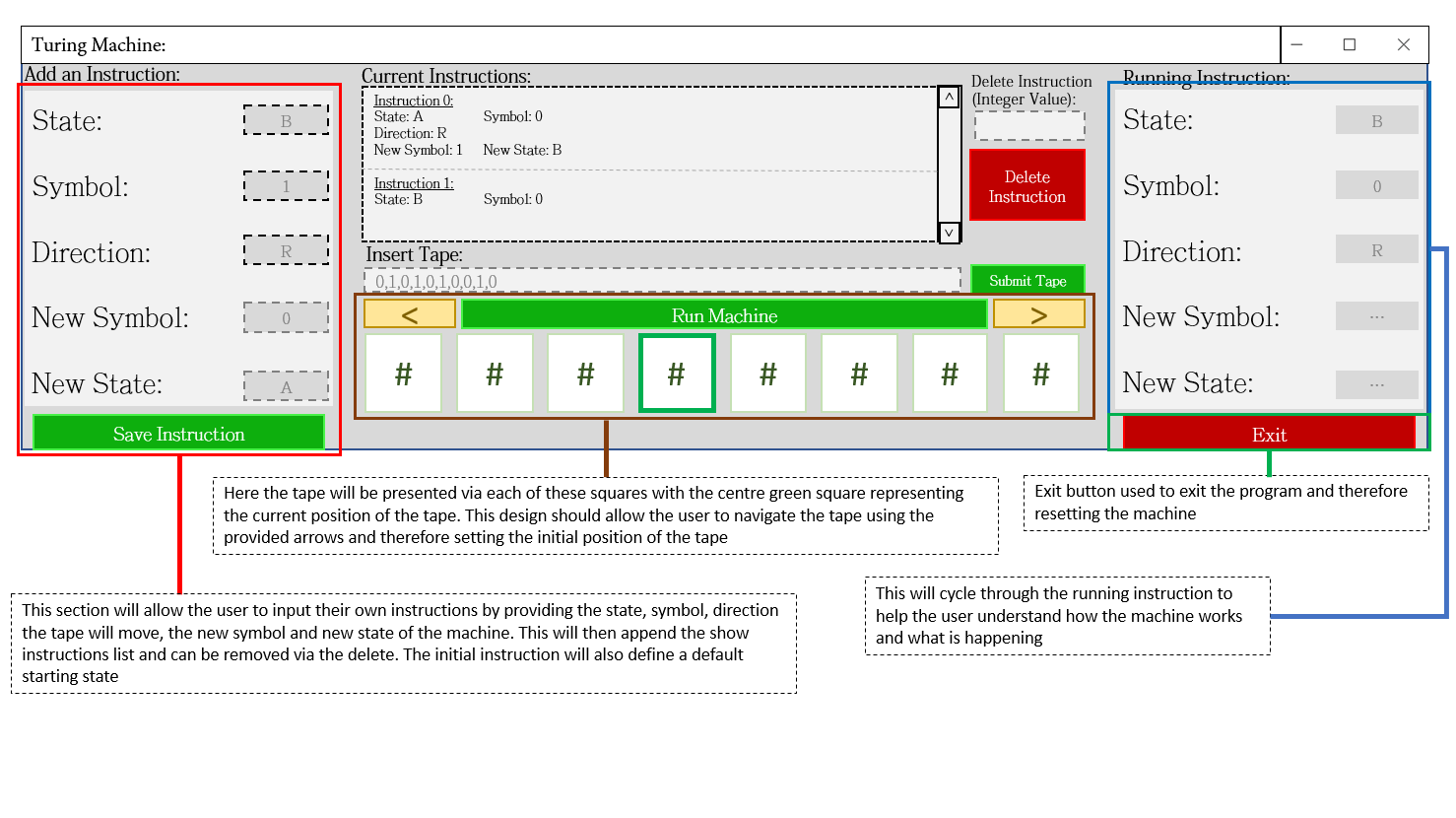
## Delivery and Model

The aim of deliverance for this thesis is to deliver a working programable Turing machine. The aim would be to achieve a dynamic model and various other finite models. The dynamic model will allow the user to fully customise the machine and teach the user on how the Turing machine works. The Turing machine can be modelled in many different ways such as using basic functions to handle every action of the tape, to a more complex model using various states to represent the machines states. Models representing a Finite State machine could be used to handle the various states that the Turing machine could be in e.g. state A, State B, etc, or each state could represent the actions the Turing machine is currently taking, e.g. reading from the tape, writing to the tape, moving the head, etc…

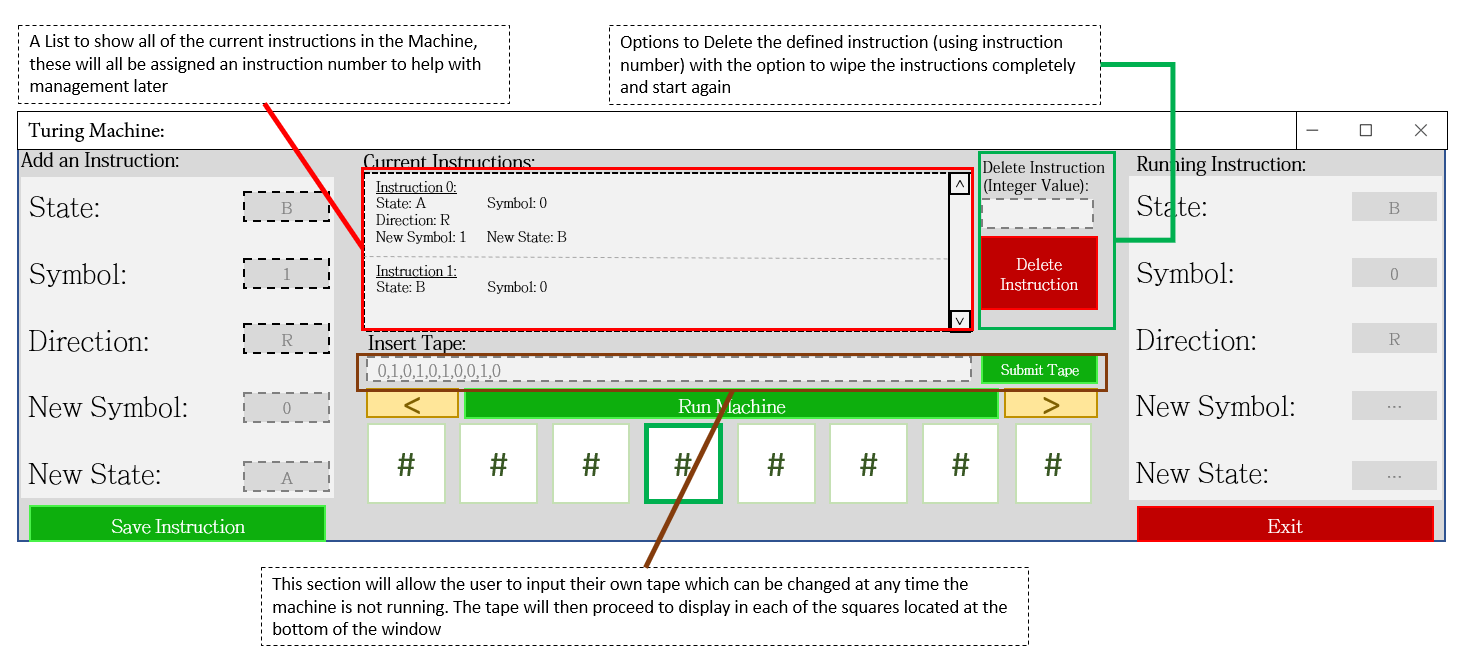
## Design

The implementation of the actual Turing Machine was produced from pseudo code which will be listed below. Several GUI designs were made to show how the program could be represented and help show the user how the machine works. The GUI designs also show how the user can implement their own instructions, tape and states; this is especially useful as it can lead to the Turing Machine being fully dynamic. A total of 2 designs were produced which will later be implemented through the use of a python GUI library which will be discussed later in this report. The initial design was developed with the ability to fully customise the machine, however there were no options for extra help on how to use the machine; this would later be changed in another design. The best way to show a Turing machine on a GUI is through displaying the tape and instructions running at run time, which will be implemented with each of these designs, allowing for the program to also teach the user how the machine works while its running and what is happening with the information presented to it. The machine will also show the final tape once the machine has finished. (The initial design shown below with the second design following):

### Design 1

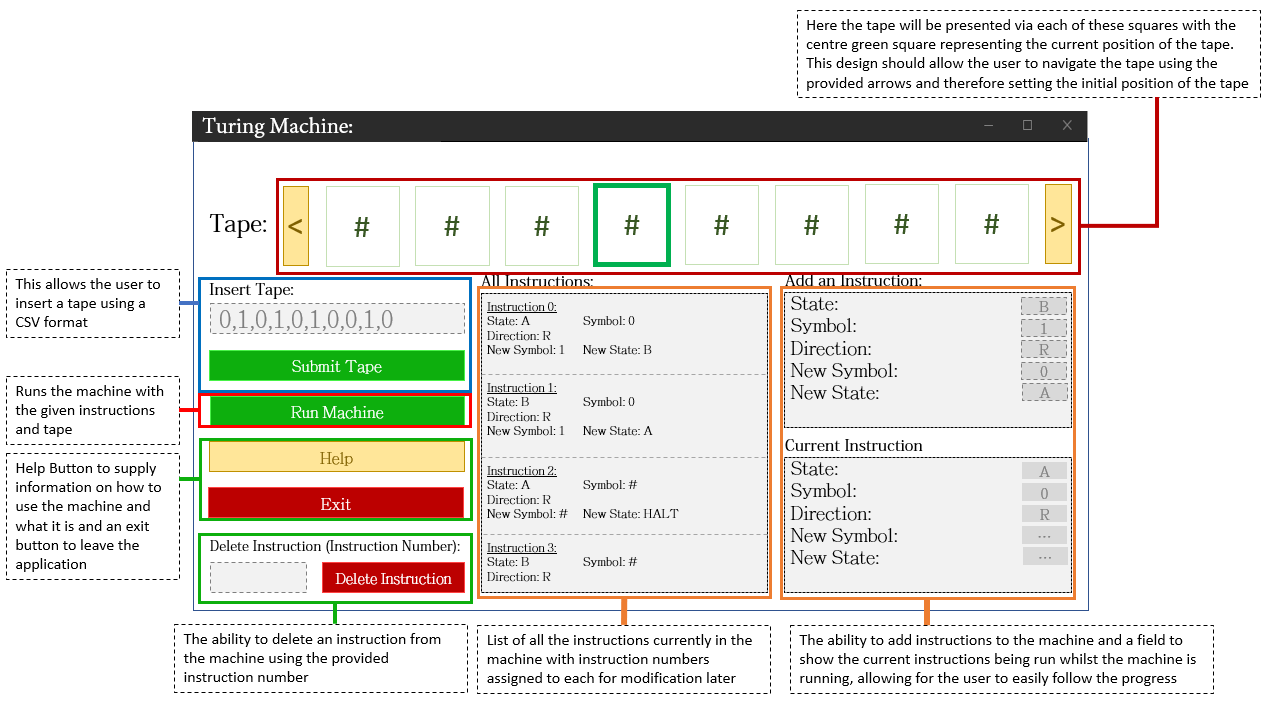


Although missing some functionality such as adding a starting state and the option to delete all instructions, this design shows all the relevant functions of a Turing machine such as the ability to see the tape run, with options to either move left or right along the tape (with the ability to select the starting position of the tape) which would show a seemingly infinite tape. The machine can also add instructions to the machine and see them running in run time through the GUI refreshing and a delay on the information updating



The machine can also view all the instructions currently loaded into the machine, while giving the option to delete any instructions present. A textbox is given to allow the user to manually add their own tape using comma separated values which would be presented on the machine tape below, with the first symbol being represented on the tape square with the solid green outline

### Design 2



This design for the Turing machine shows a lot of the same aspects as the previous design, with repositioning to better manage the space given. The “All Instructions” Listbox is now bigger to show more of the instructions at a time and a useful help button was added to inform the user how the machine works and how to use it. However, in this design an option is missing to add the starting state, this will be added when the design is implemented

### Pseudo Code v1:

RUNNING MACHINE FUNCTION:

Instructions: list = [*pre\_defined*]

Current\_instruction\_set: list = []

Tape: list = [*pre\_defined*]

Tape\_Position: int = *0*

For (index) loop cycling through the set of instructions:

If an instruction matches the current state of the machine and symbol on the tape:

Current\_instruction\_set = Instructions[index]

Disable GUI buttons/labels

Show the running instruction on the GUI

Tape[Tape\_position] = new\_symbol

Current\_state = new\_state

If direction is Right:

Tape\_position += 1

Else if direction is Left:

Tape position -= 1

If Current\_state = HALT:

End Machine

Print final\_tape

Enable GUI buttons/labels

The first prototype for the Turing machine was a very simple representation of how the machine should work. Most of the attributes such as the machines tape and instructions would be pre-defined to avoid a confusion. The machine would first check to see if there are any instructions loaded into the machine which match the current state and the current symbol being read in from the tape; once an instruction has been found the machine would begin to make the changes to the machine and tape. Firstly, the machine would change the symbol on the tape at that current position, then the machine would change the state to the next state specified from the instruction. After this the machine would then move in the specified direction, if no direction is supplied the machine would not move. Finally, after every change has been made to the tape and the machine, the machine would perform a break and proceed to find if there is a new instruction which can match the new attributes on the machine. When the machine reaches a “HALT” state, or the machine cannot find an instruction that matches the state and symbol read, a final tape would be produced for the user to see, whilst resetting the machines attributes. However, when the machine is running, if the machine has met the end of the tape or wishes to proceed past the start of the tape, a blank symbol (#) would be added to simulate a somewhat infinite tape on the machine.

### Pseudo Code v2:

STATE CLASS:

Instructions: list = []  
Current\_context: object = None

executeInstruction Function:

For (index) loop cycling through the set of instructions associated with the

current state:

If an instruction matches the symbol on the tape:

Tape[Tape\_position] = new\_symbol

Current\_state = new\_state

If direction is Right:

Tape\_position += 1

Else if direction is Left:

Tape position -= 1

Else if Current\_state = HALT:

End Machine

Print final\_tape

Enable GUI buttons/labels

INSTRUCTION CLASS:

symbol = ''

direction = 'None'

new\_symbol = ''

next\_state = ''

TURING MACHINE CLASS:

Current\_state: object = None

State\_name: string = “”

Available\_states: dictionary = {}

Tape: list = []

Tape\_position: int = 0

Temporary\_instructions: list = []

Starting\_state = None

runMachine Function:

Starting\_state = Set the starting state to one provided

Tape = Set tape to one provided

For loop going through Temporary\_instructions:

If state isn’t in Available\_states:

Add the state to the dictionary

Add the instruction to the Available\_states dictionary with asociated state

While the state isn’t HALT:

State.executeInstruction Function

The next pseudo code attempts to utilise the state class from the state design pattern. The machine will create a state object for each of the states on the machine when filling the available states. Each state will hold its own set of instructions which will represent the instructions associated with the current state. This allows for the machine to dynamically add states depending on the instructions provided. Each instruction added will add all of the states associated that instruction to available states. When the instruction is added to the machine an instruction object will be created holding all of the relevant attributes for the instruction. Disregarding all of these differences, the machine will run fairly similar to the previous pseudo code, by checking for the symbol on the tape and then performing the relevant actions to the machine.

### Pseudo code v3:

STATE CLASS: Current\_context: object = None

TRANSITION CLASS:

Is0 Function:print “ERROR”

Is1 Function:print “ERROR”

IsHALT Function:print “ERROR”

TURING STATE A CLASS:

Instructions: dictionary = 0: [R, 1, B], HALT: [R, HALT, HALT]

Is0 Function:  
 change tape[tape\_position] to new\_symbol  
 if instruction = “R”:  
 tape\_position += 1

elif instruction = “L”:  
 tape\_position -= 1  
 change state to next\_state  
 return True

IsHALT Function:  
 change tape[tape\_position] to new\_symbol  
 if instruction = “R”:  
 tape\_position += 1

elif instruction = “L”:  
 tape\_position -= 1  
 change state to HALT   
 return True

TURING STATE B CLASS:

Instructions: dictionary = 1: [R, 0, A], HALT: [R, HALT, HALT]

Is1 Function:  
 change tape[tape\_position] to new\_symbol  
 if instruction = “R”:  
 tape\_position += 1

elif instruction = “L”:  
 tape\_position -= 1  
 change state to next\_state  
 return True

IsHALT Function:  
 change tape[tape\_position] to new\_symbol  
 if instruction = “R”:  
 tape\_position += 1

elif instruction = “L”:  
 tape\_position -= 1  
 change state to HALT   
 return True

The third iteration of the machine’s pseudo code produces a static Turing machine with predefined states, instructions and a tape, this is to help get a better grasp on how the machine can be made into a finite state machine as the Turing machine could be defined as a finite state machine with many extra features. The transition class is used in this instance to help transition between each of the machine’s states, as each of the states in the machine are predefined and are not dynamically created on initialisation. Each state has its own set of functions which are associated with what that state can and cannot do, this is determined by the instructions associated with the state and only allows certain methods to be called within each state.

### Pseudo Code v4:

STATE CLASS -

STATE CONTEXT CLASS -

TURING STATE CLASS:  
 description: string = *statename* instructions: dictionary = [symbol][newsymbol, direction, newstate]

exectuteInstruction Fuction:

current\_symbol\_on\_tape = TURING MACHINE TAPE[position on tape]

TURING MACHINE TAPE[position on tape] =

instructions[current\_symbol\_on\_tape][NEWSYMBOL]

if instructions[current\_symbol\_on\_tape][DIRECTION] is Right:

tape\_position += 1

else if instructions[current\_symbol\_on\_tape][DIRECTION] is Left:

tape\_position -= 1  
 else:

TURING MACHINE FINISHED = True

return False

Current Context

SetState(instructions[current\_symbol\_on\_tape][NEWSYMBOL])

return True

TURING STATE HALT CLASS -

TURING MACHINE CLASS:

initialise Function (pass in: tape, tape\_pos, instructions, starting\_state):

tape = tape

tape\_position = tape\_pos

temp\_instructions = instructions

starting\_state = starting\_state

for loop through temp\_instructions length:

if instruction state isn’t in available states:

available states[state].instructions[symbol] =

instructions for that state and symbol

This is a very similar prototype to the pre-defined Turing machine; however, this version of the machine is dynamic and will allow any amount of instructions and a tape before the machine runs, which will be dynamically created upon initialisation. When the Turing machine is initialised, it will take 4 parameters including the tape being used, the starting position on the tape, 2D list of instructions and a starting state, which will all be initialised when the machine first runs.

Although a State and StateContext class is being used, the Transition class is not present, this is due to the fact which it is not needed in this prototype as there is only one function in the state which can run, and this will dynamically work depending on the given state and instructions it is presented with. Each state will show up in available states through the state name and an object associated with the Turing State Class. Each object will hold the associated instructions for that state, each instruction is represented as a dictionary with the symbol being the key and the contents being the actions the machine must take. The executeInstruction function will then act appropriately and make the changes to the tape and the machine.

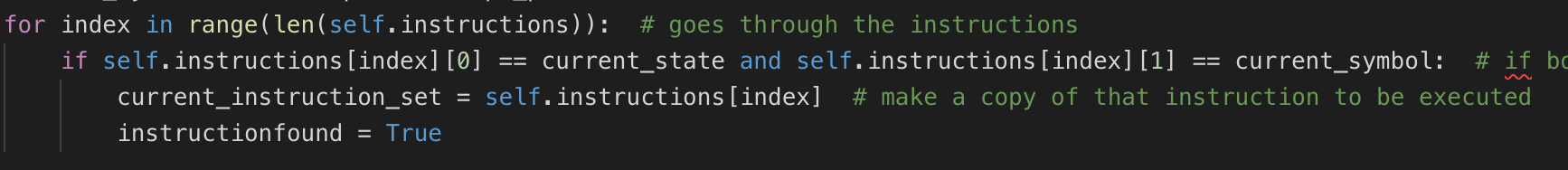
## Implementation

PyQt was originally used for the GUI, however due to limitations causing the window to fail to refresh, Tkinter was considered. Tkinter allowed for a GUI which looked similar to the design, whilst still allowing for the window to refresh whilst the Turing Machine was running. A few prototypes were made regarding the designs and pseudo code that was prepared. However, these will not represent the final product for this project and will still need to have some improvements in the future.

### Prototype 1

The first prototype was a very basic version of the Turing machine using mainly if statements to perform the actions required from the machine, the majority of the machine was also run in a single function.

This will initially cycle through the current instructions in the machine and then a check was made for if the correct instruction was on the machine depending on the current state and symbol on the tape.



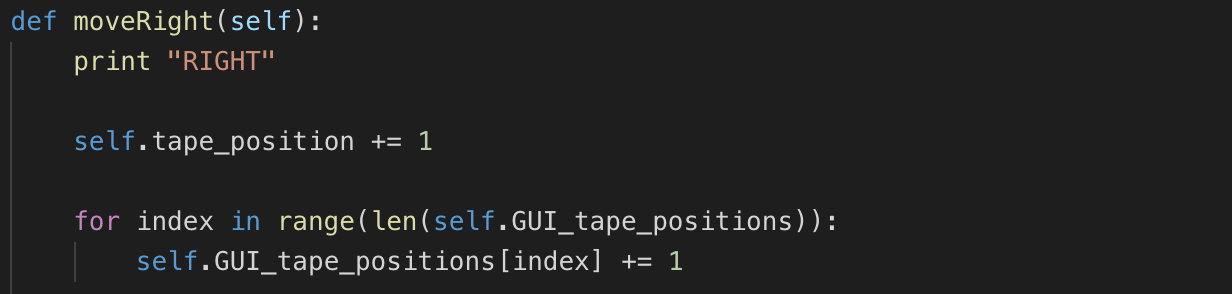
Once this check was made the machine would then proceed to make the relevant changes to the machine and the tape including the state change, tape change and the movement on the tape.



Relevant functions were made to allow the tape to move left and right. This was also needed to allow for the change to the tape on the GUI, the following screenshot will show how the positions were changed on for the tape, however the GUI tape positions would also need further checks in case the machine tape would be out of range.



The same checks were made to the moveRight() method, however the changes to the tape position were slightly different.

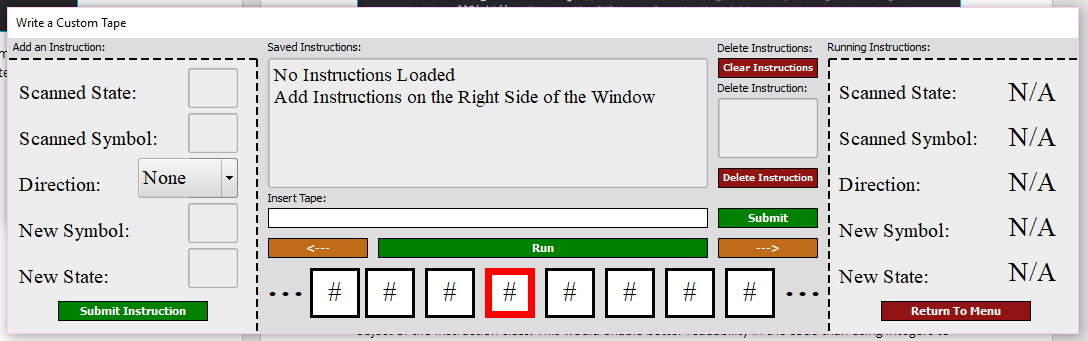


Once the machine had finished running due to the state either transitioning to a HALT state or if no instruction was found, this was handled through using a messagebox to show the final tape when finished. When the machine does end the GUI will reset to allow for another instance to run. This will also remove any additional blank symbols on either side of the tape which were added to ensure an infinite tape could be simulated



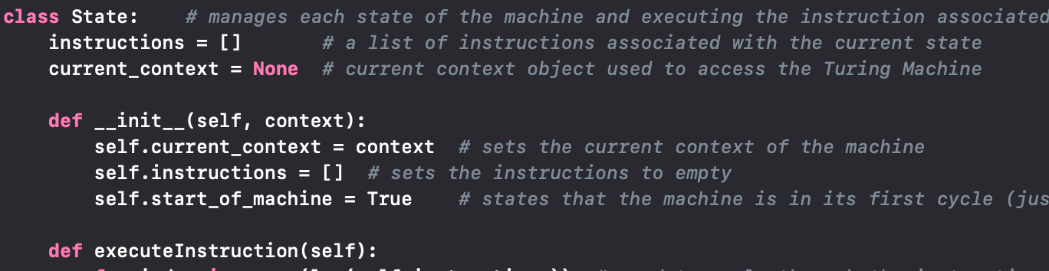
This version of the GUI was programmed using PyQt, however due to some limitations with PyQt other options were considered in future prototypes (which will be covered later)



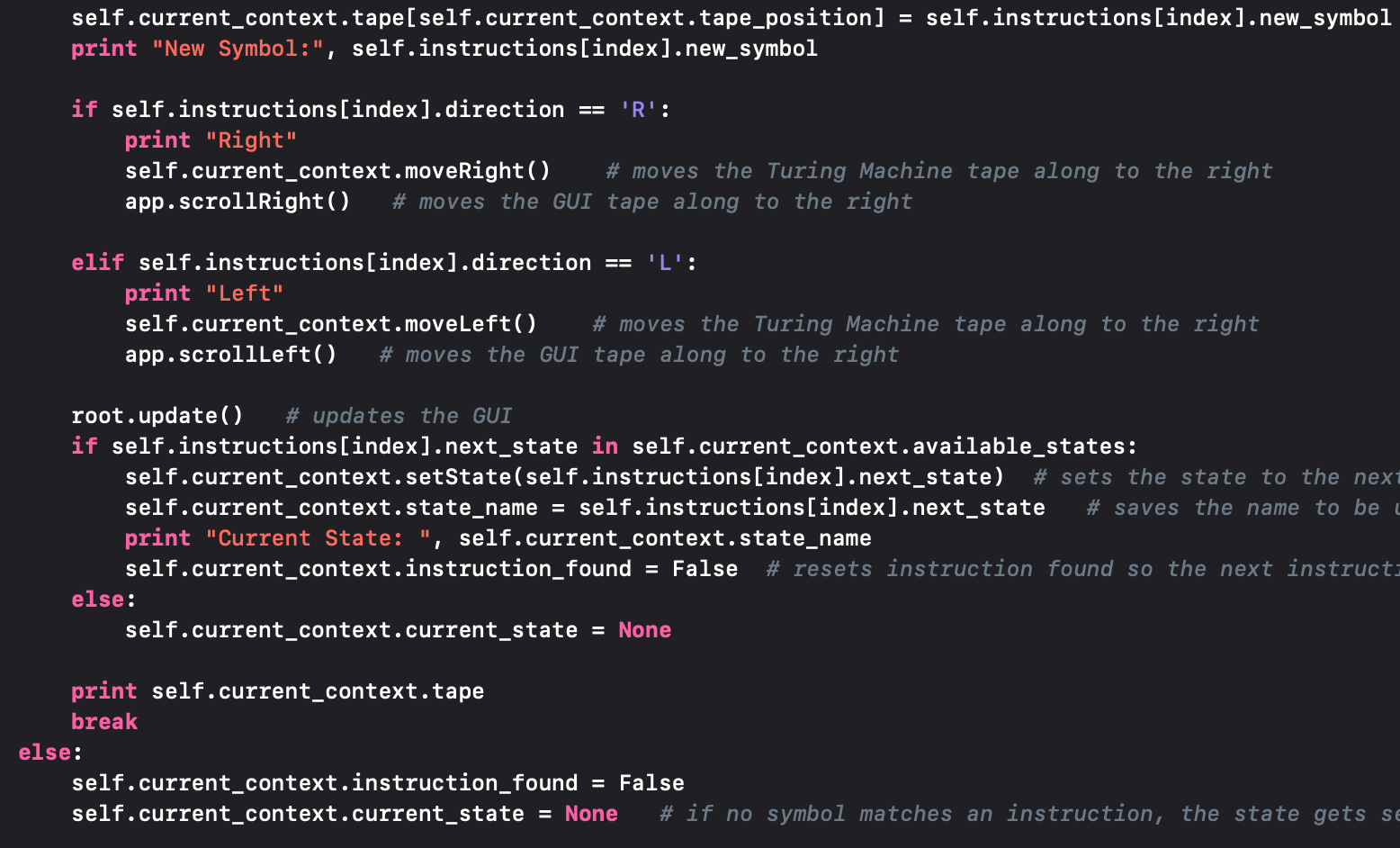


### Prototype 2

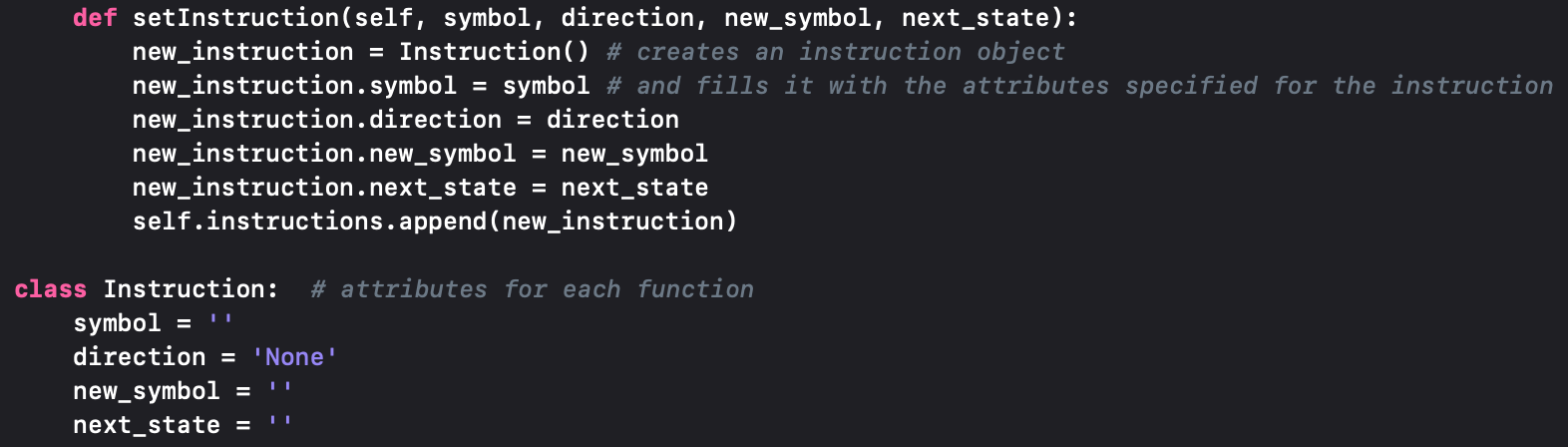
The next version of the Turing Machine was an attempt to use the state design pattern as a basis for each state the machine goes into. There were several classes within this prototype including a State class which would handle each state the machine would go into, as well as the instructions being carried out in that state. Each of the instructions associated with the current state would be stored in the instructions list within that state, meaning that it can only be accessed when in the correct state. This was an attempt to also make the machine dynamic as the instructions would be added to the state when initialising the Turing Machine.



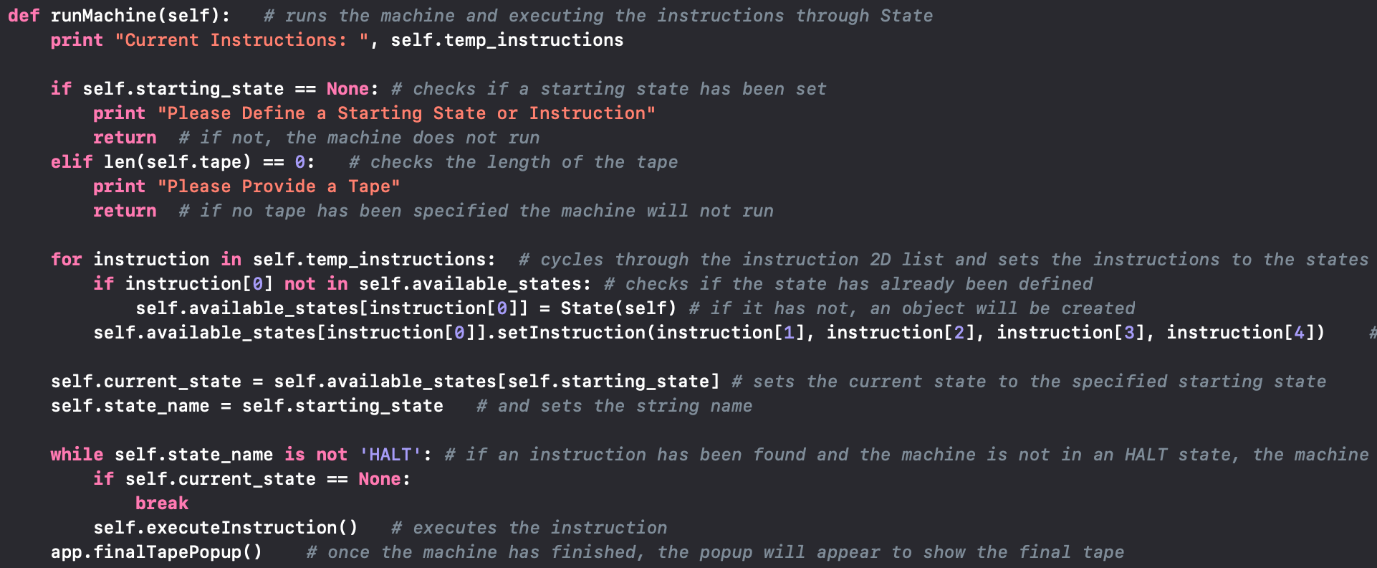
The actual running of the machine would work fairly similar to the previous prototype; however, the machine would only check the symbol on the tape as it would already know the state it is in. This would then read from the instructions list and perform the relevant instructions, e.g. change symbol, move the tape and change state. This would run off of a Boolean to state if an instruction has been found or not, therefore only running when it can.



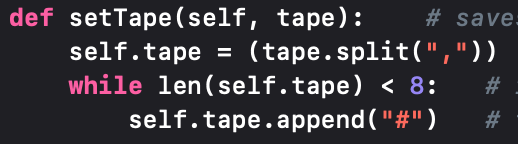
The instructions are however stored differently than before, with each instruction being stored as an object of the Instruction class. This would enable better readability in the code than using integers to represent each element of the instructions



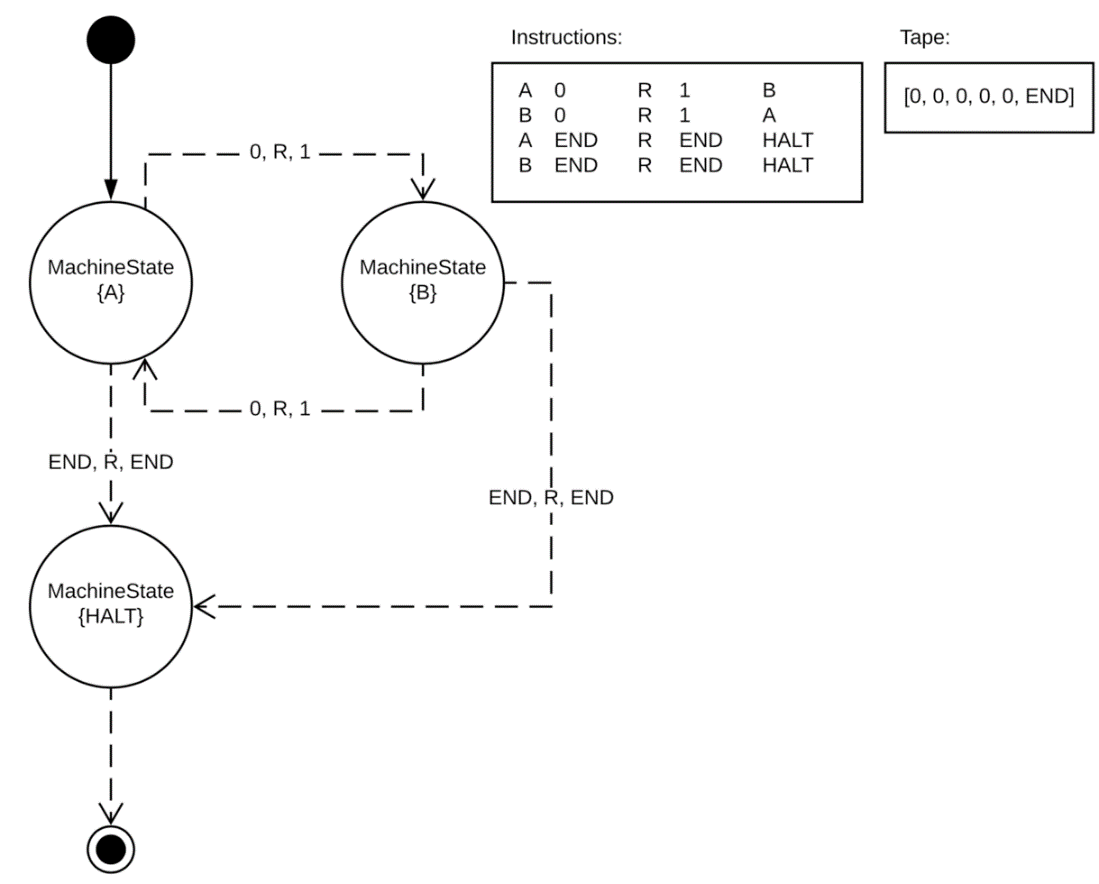
The actual machine itself would run from the TuringMachine class, which would also hold the other attributes of the machine including the tape itself and the available states the machine could transition into. The runMachine function would initially save all of the instructions to each state, then loop until the machine reaches a HALT state. Causing the final tape to be presented in a messagebox.

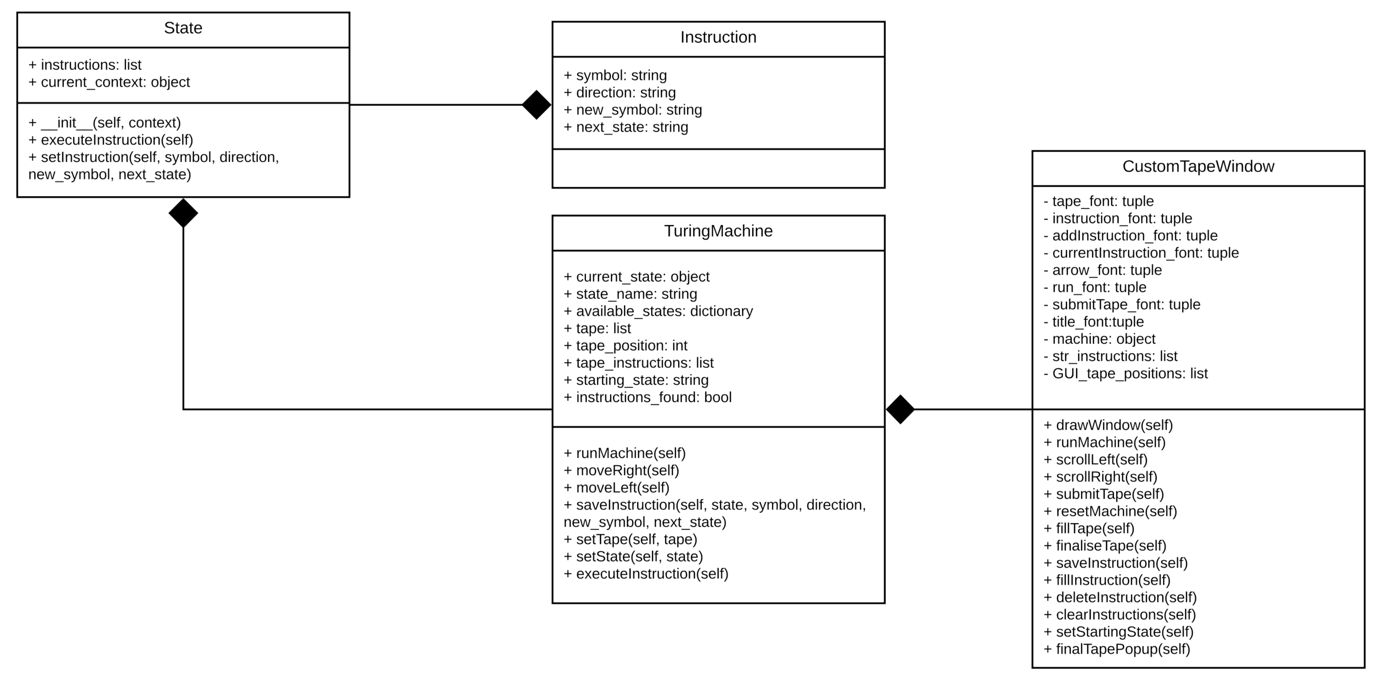


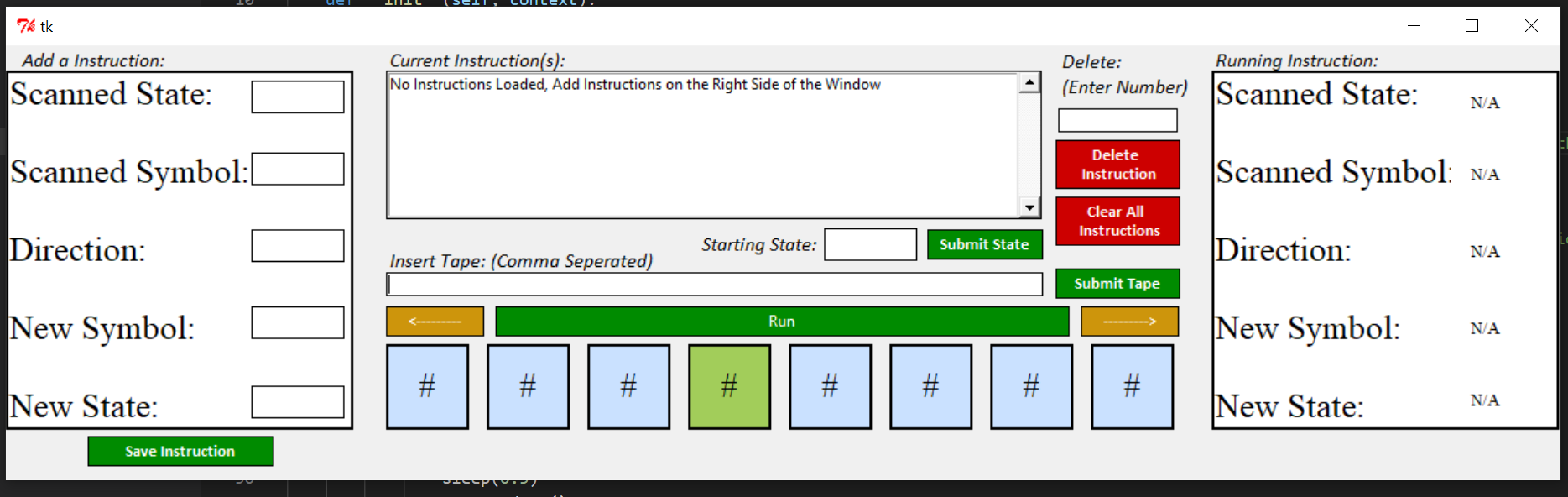
The available states would initially hold a HALT state as the machine should always be able to transition to it, to stop the machine from running:

The tape is taken in by using CSV, this allows for multiple values to be presented for an input. This would also make sure the tape has enough values to show on the GUI by automatically populating it with blank symbols if the tape is too short.

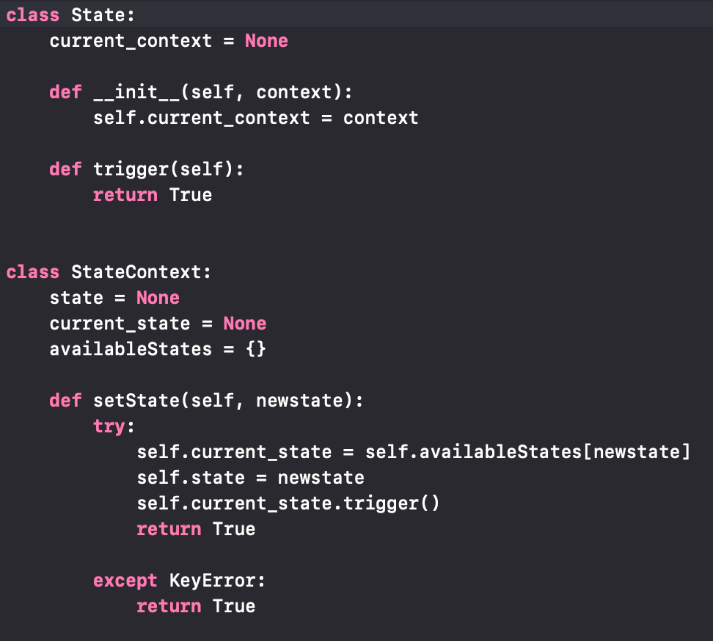
Below is a State Diagram with custom instructions inserted into the machine as well as a UML to describe how the classes communicate with each other. There is also a screenshot of the GUI using tkinter instead of PyQt, which when implemented, was close to the original design of the machine.

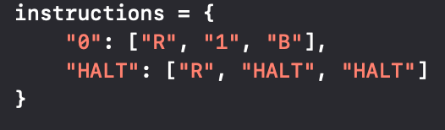


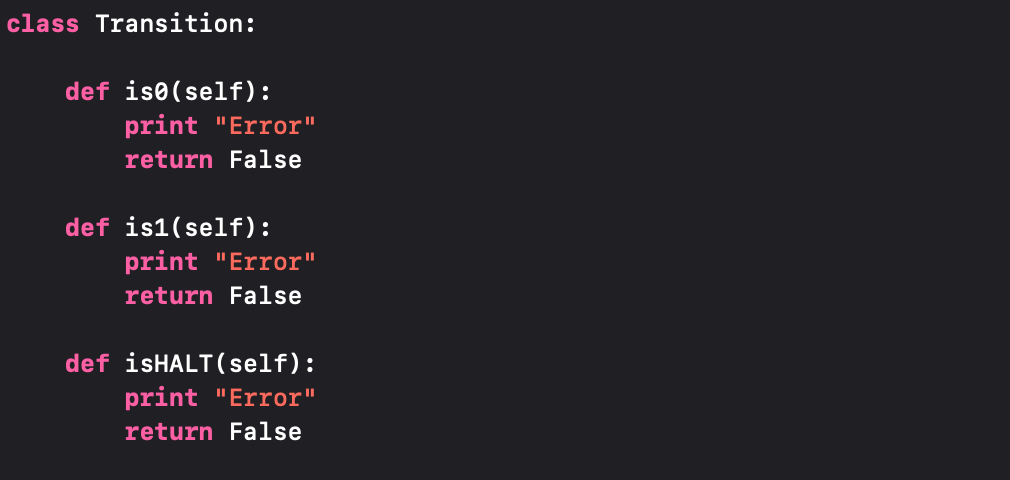




### Prototype 3

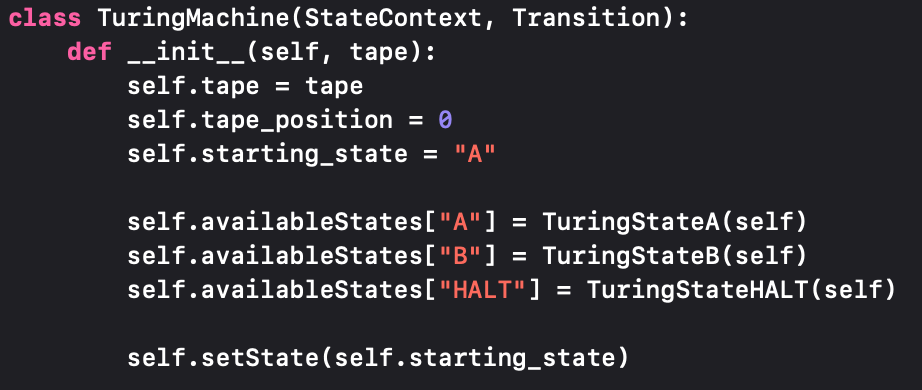
The third prototype created is a pre-defined Turing machine which has separate states for each state of the machine, with methods in each which can only run whilst in the state. Due to this interpretation being a state machine, the State and StateContext classes have been used to hold each state of the machine as well as giving context to each state and the States which the machine can transition to.

Each of the states in the machine hold an instruction dictionary which will represent the symbol as a key and the contents being the instructions carried out. This will be bound to each state and each state has a separate set of instructions which can only be carried out in that state

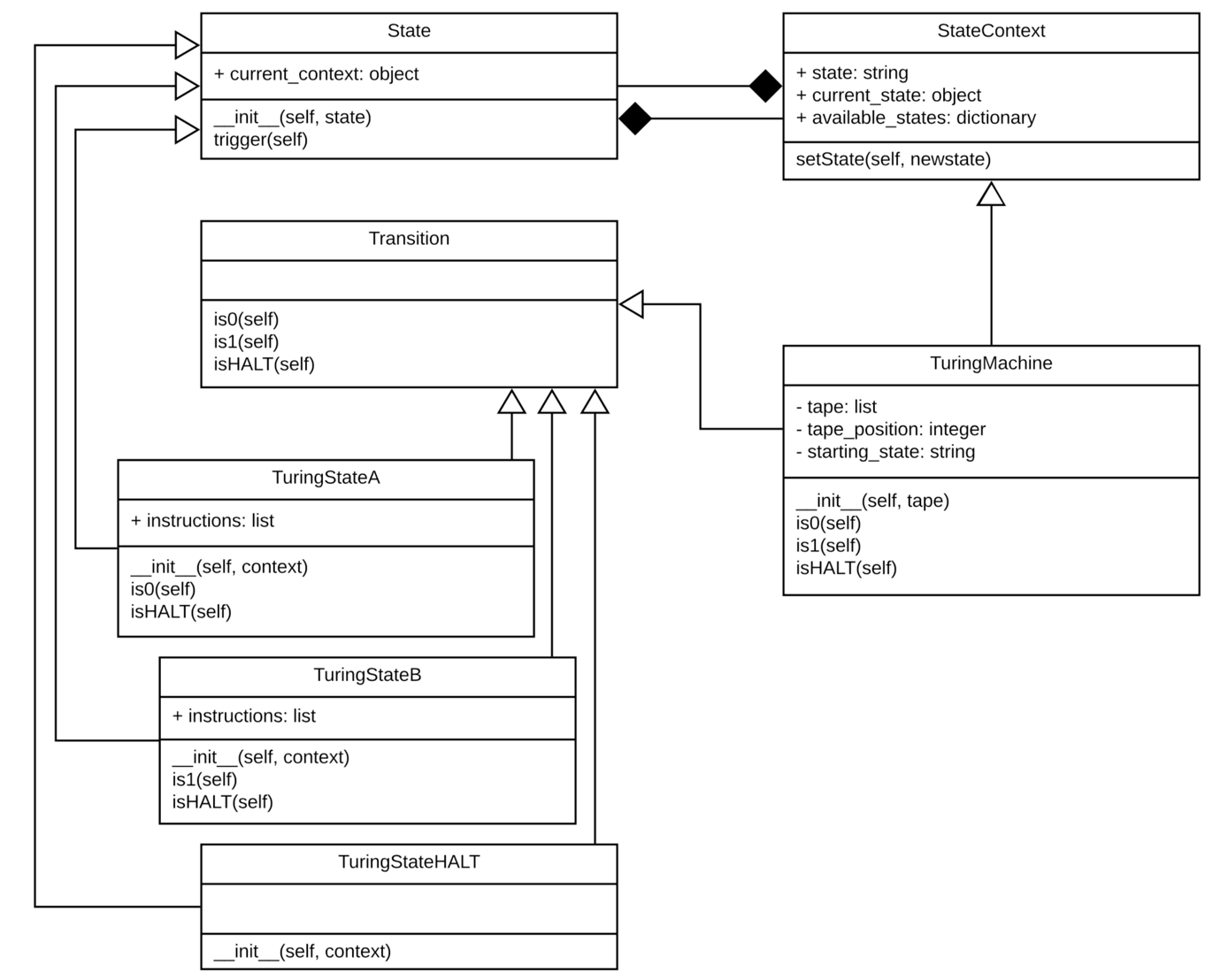
Each state will have its own set of instructions representing what the state can do given it is provided with the correct instruction. For example; the next screenshot shows how if the tape reads in a 0, and is in State A, the following would occur to change the tape and machine then move in the given direction

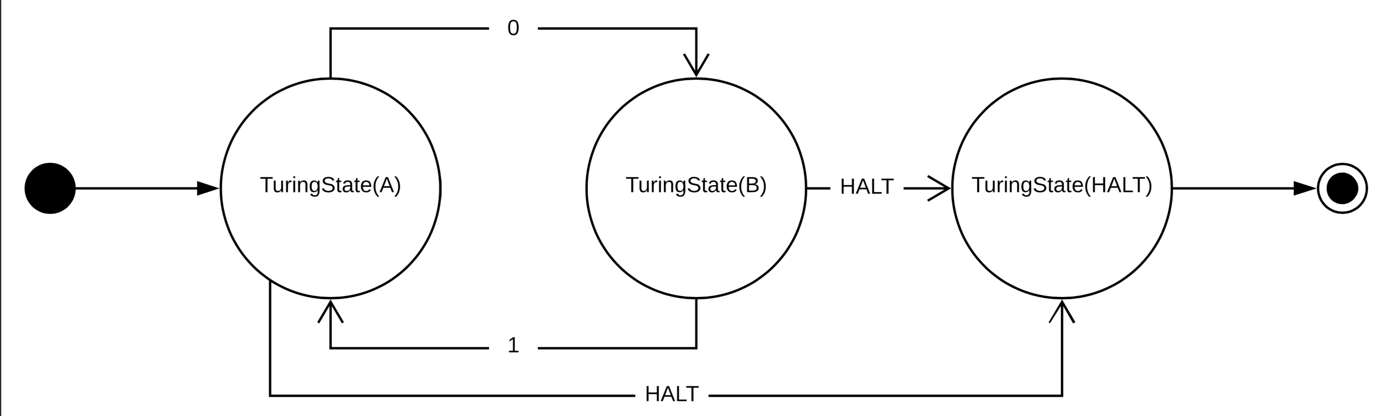
The Transition class has also been added to transition between each state, as seen here it is set with 3 methods to each represent an action the machine could take in a given state. These will represent error messages if the machine cannot transition in its current state

The TuringMachine class will hold all of the relevant attributes to the machine including the tape, this will handle the contents of the tape and all of the methods to change the tape will be called through here using current\_state, this allows the machine to dynamically call certain methods in other states depending on the state the machine is in and will initialise each state when the Turing Machine is created



Below is a given state diagram and a UML to show how the classes interact, for this given example the machine only accepts values of 0, 1 and HALT. However, in the future, the machine should be fully dynamic allowing the user to use a tape and any instructions/symbols or states of their choosing. As there was no GUI for this interpretation, no GUI screenshot will be shown:





### Final Product

The final product of the developed Turing machines will use the state design pattern to produce both a finite and dynamic set of Turing machines. There were several finite models developed which are all accessible from a menu, these include the various Turing machine “programs” accompanied by a set of tapes which will work with the given instructions. Each machine is including a state diagram so the user can easily see what is supposed to be happening on the tape. A dynamic model was also made which allows the user to add a tape, starting state, tape position, instructions and the speed in which the machine will run at, to allow a totally user orientated dynamic experience with programming the Turing machine. The dynamic model also uses the state design pattern; however, this is done slightly differently to the finite model which will be discussed in a further heading

#### Finite Models

##### Hello World

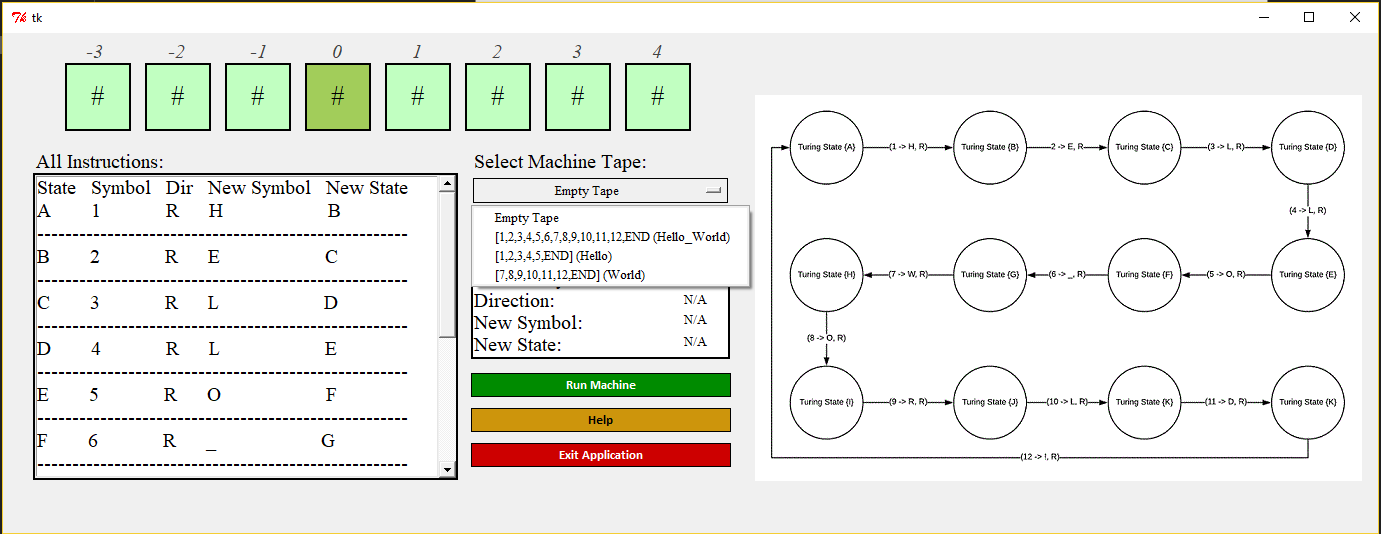
The hello world Turing machine was one of the first that were made for this final product and uses a basic state diagram to give a basic description on how the Turing states can change depending on the options on the tape, this model was necessary as it gives a basic understanding initially on how the machine will work.

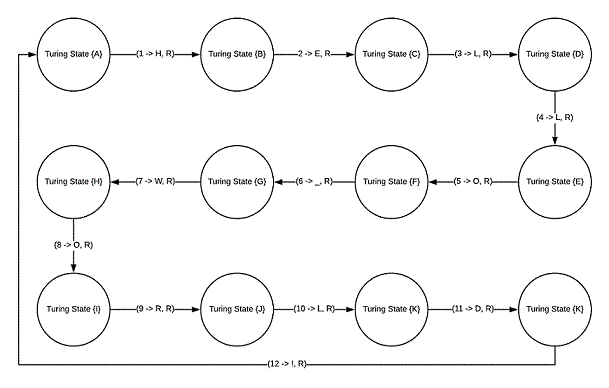
The machine uses the state design pattern, by using the state and the state context classes as shown in the above prototypes. This will have a series of classes representing each state which will transition between each other depending on the context of the machine. The code in this screenshot shows an example of how a state will work.

Each of the finite models including this one come with a set of tapes which can be selected by the user. This was done using a dropdown menu which can be used to access 3 tapes in this machine.

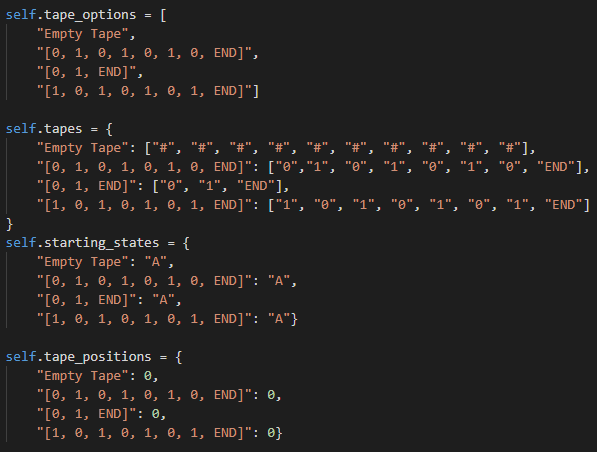
1. [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, END] 🡪 [H, E, L, L, O, \_, W, O, R, L, D, !]
2. [1, 2, 3, 4, 5, END] 🡪 [H, E, L, L, O]
3. [7, 8, 9, 10, 11, 12, END] 🡪 [W, O, R, L, D, !]

Each of the models will be accompied by a instruction set (see left) that will state what will happen in each instance of the machine, aswell as a state diagram (see right) to show how the machine will operate. The below screenshot will show a capture of the GUI used and how the user can select a tape:

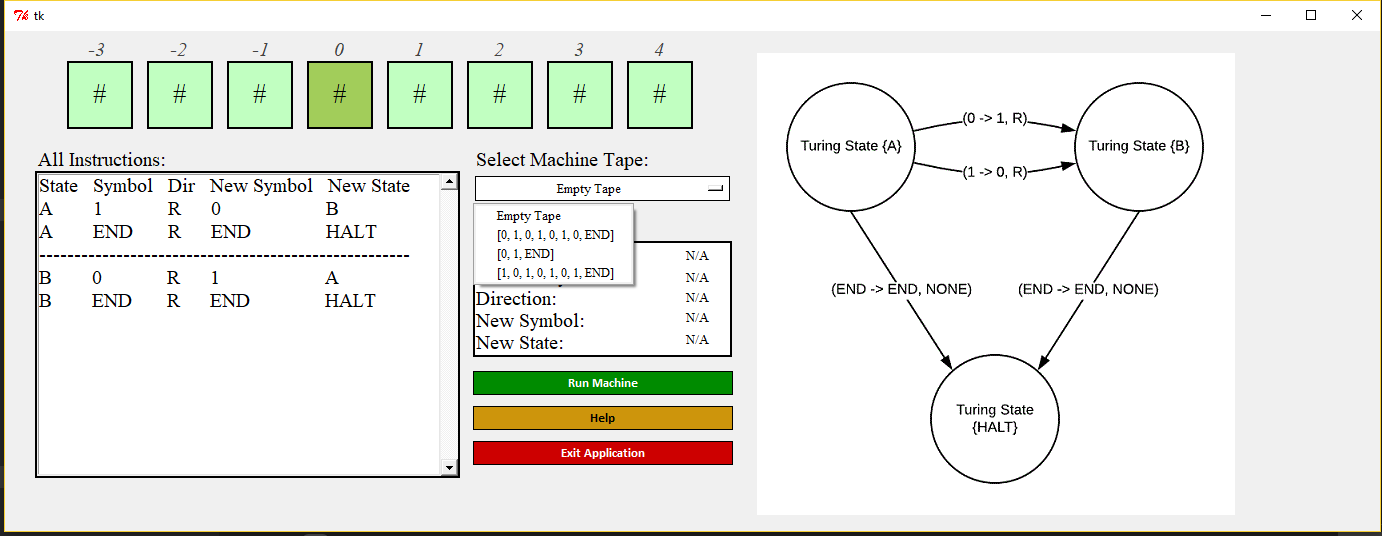


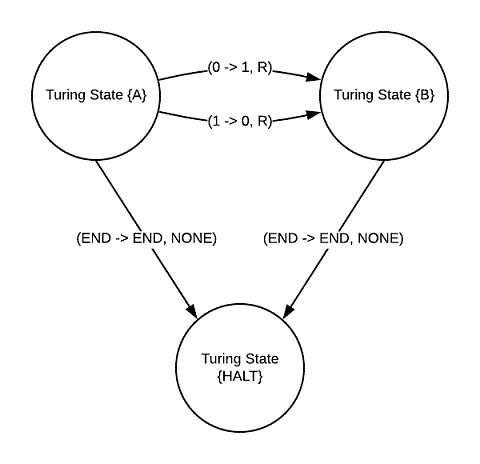


##### Flip 1/0

The next model made was a simple machine used to flip 1’s to 0’s at any length tape given. This model also used a dropdown menu to access different tapes and shows a simple state diagram explaining how the machine will operate. The coding aspect of this model was down very similarly to Hello World program as it has been modelled as a Finite State Turing machine. Each of the programs also had a starting position and starting state tied to each tape meaning that when a tape was selected the machine would configure itself to the correct context. This is an important feature for some of the further models as they require a more complex approach to how you start the tape.

1. [0, 1, 0, 1, 0, 1, 0, END] 🡪 [1, 0, 1, 0, 1, 0, 1, END]
2. [0, 1, END] 🡪 [1, 0, END]

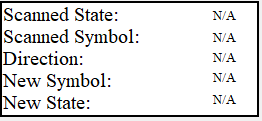
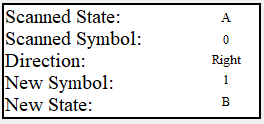


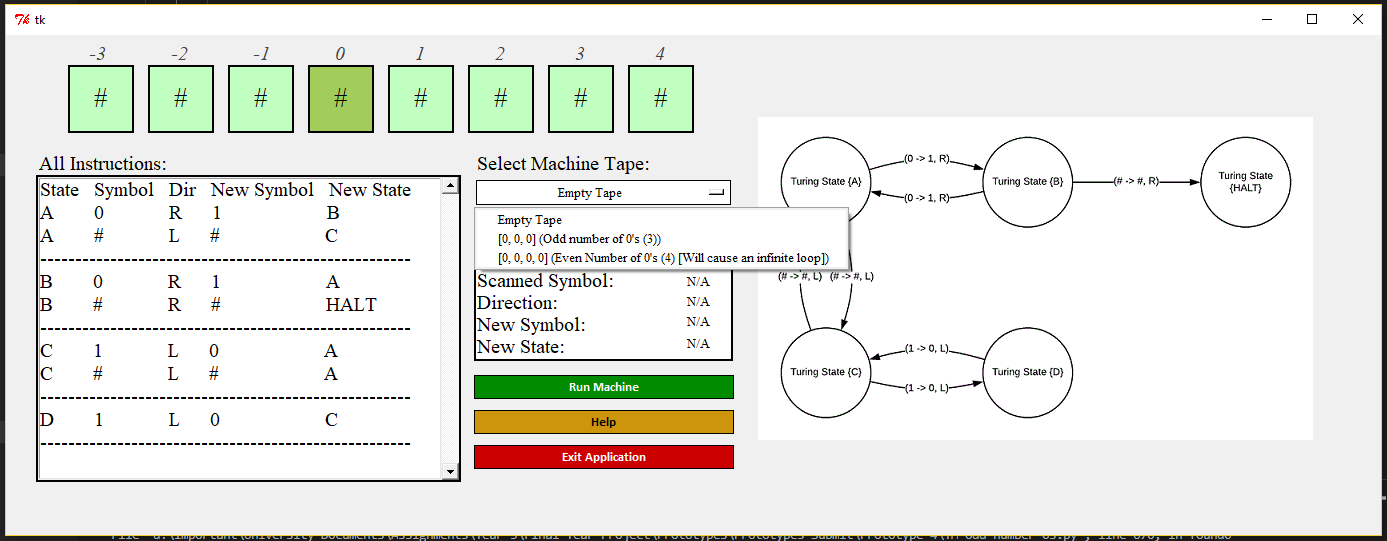


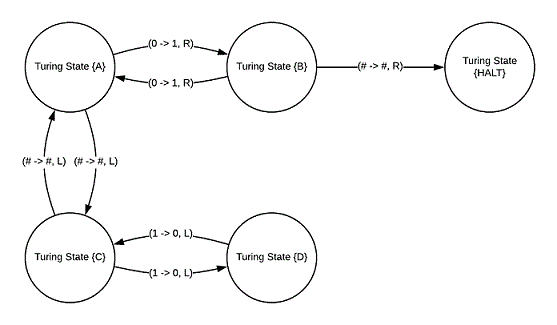
##### Odd Numbers

This model will check a tape for an odd number of 0’s on the tape this will then infinitely loop if an even number of 0’s are presented this makes the model slightly more complex than the previous as each there were more complex transitions between each state of the machine and there was a wider meaning to why the machine was in each state. This is because depending on which state the machine is in, it was either an even number of 0’s or an odd number of 0’s. The machine will only come to a HALT if an odd number of 0’s is found, the machine can be cancelled at any time if the python script is closed. This model also used a finite state model to predefine each of the states the machine can go in, it was also accompanied with a state diagram, instructions and a set of example tapes to test the machine with. In each of the finite state models there is also a small part of the GUI which describes what is actually currently happening in the state, showing the current state, the symbol read, the direction the tape will go next, the new symbol and new instruction on the tape.

1. [0, 0, 0] 🡪 [1, 1, 1]
2. [0, 0, 0, 0] 🡪 [1, 1, 1] [0, 0, 0] (infinite loop)



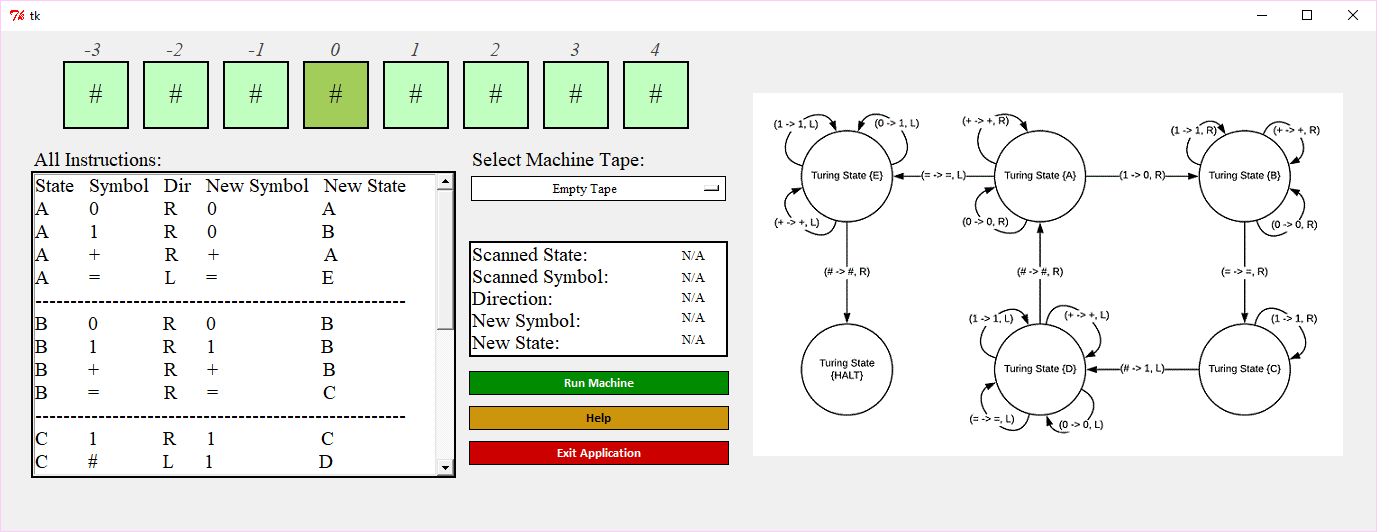


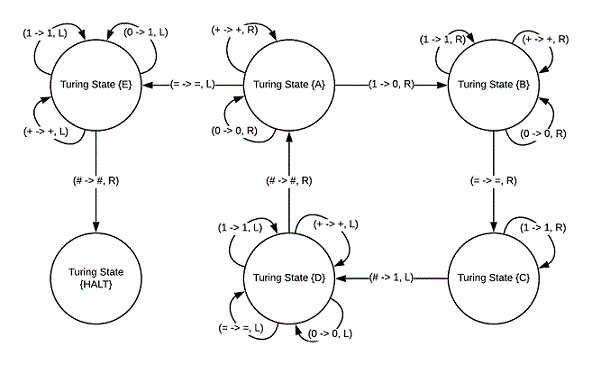
##### Unary Addition

The next program made was a unary addition Turing machine. This would use a set of sequential 1’s to determine what numbers are being summed, for example; [1] [1] [1] [+] [1] [1] [=] would represent (3 + 2 =). This model also uses a state pattern to handle each of the instruction states the machine can be in. transition methods are also used to switch between these states. The tapes used for this model were the following:

1. [1, 1, 1, +, 1, 1, =] 🡪 [1, 1, 1, +, 1, 1, =, 1, 1, 1, 1, 1] (3 + 2 = 5)
2. [1, 1, +, 1, 1, 1, 1, =] 🡪 [1, 1, x, 1, 1, 1, 1, =, 1, 1, 1, 1, 1, 1] (2 + 4 = 6)

This machine will cycle through each of the 1’s on the tape and change them to 0’s and add them to the end of the tape. Once the machine has finished running it will then change the 0’s on the tape back into 1’s. The machine will show instructions on the left of the window accompanied by a state diagram to the right. The following screenshots show the GUI used and the state diagram associated with this model:



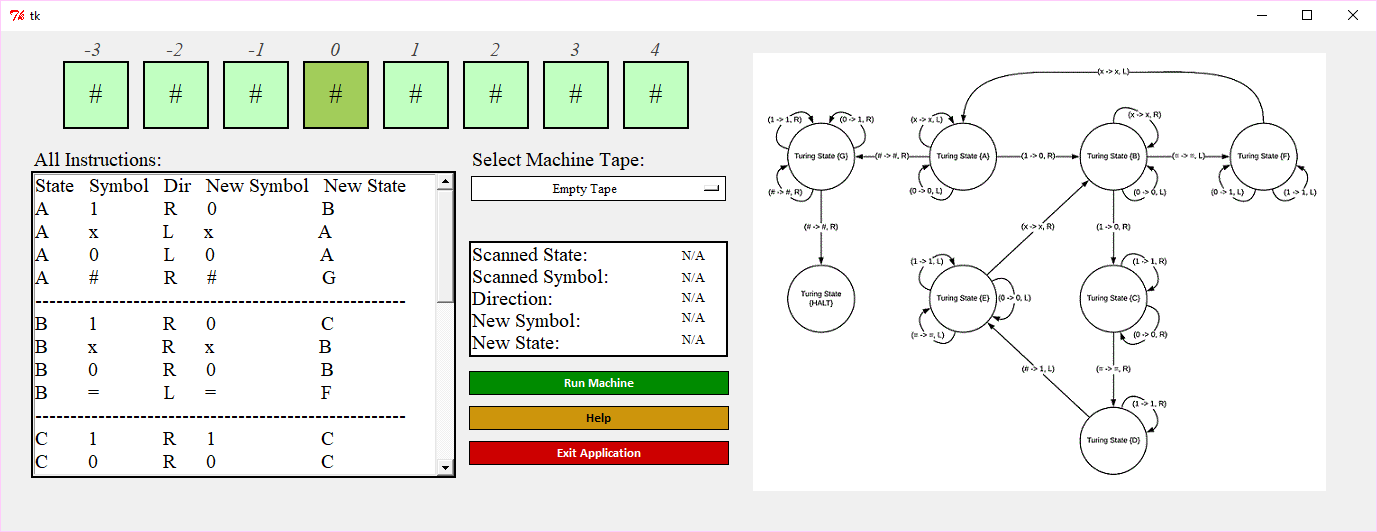


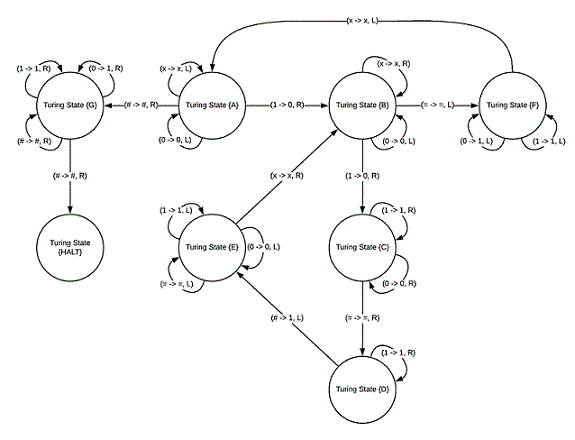
##### Unary Multiplication

The unary multiplication program was very similar to the addition model, for example on the tape the format would be represented as so: [1] [1] [1] [1] [x] [1] [1] [=], this would represent (4 x 2 =). This will cycle through the 1’s on the left of the tape and for each of those 1’s it will print the 1’s on the right, that many times. This was an interesting model to make and is the most complex out of all the Turing machines, due to its more elaborate state diagram. The machine will run through the tape and perform those actions, when the machine has appended all of the 1’s on the tape, it will then go back through the tape and set all the 0’s to a 1, this will then show a dialog box displaying the final tape for the program and reset the machine completely. The tapes used for this machine were the following:

1. [1, 1, x, 1, 1, 1, =] 🡪 [1, 1, x, 1, 1, 1, =, 1, 1, 1, 1, 1, 1] (2 x 3 = 6)
2. [1, x, 1, =] 🡪 [1, x, 1, =, 1] (1 x 1 = 1)
3. [1, 1, 1, 1, 1, x, 1, 1, =] 🡪 [1, 1, 1, 1, 1, x, 1, 1, =, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] (5 x 2 = 10)

A screenshot of the GUI and state diagram are shown below:



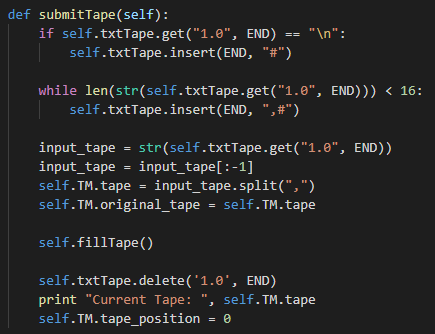


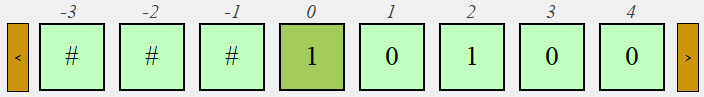
#### Dynamic Model

The dynamic model works differently to the finite state models as the states are not predefined before starting the machine. This model will allow the user to write their own states in the machine, meaning that there could be custom instructions allowing the machine to be fully dynamic. The tape is also allowed to be changed by the user and will be presented on the GUI tape at the top of the window. This is also accompanied by arrows allowing navigation through the tape, the arrows can move either left or right through that tape and there are functions to manage how the tape will be displayed on the GUI. The starting state and tape position can also be changed before the machine starts to run. There is an option to allow for reading a machine from a file provided the correct format is saved in the text file.

##### The Tape:

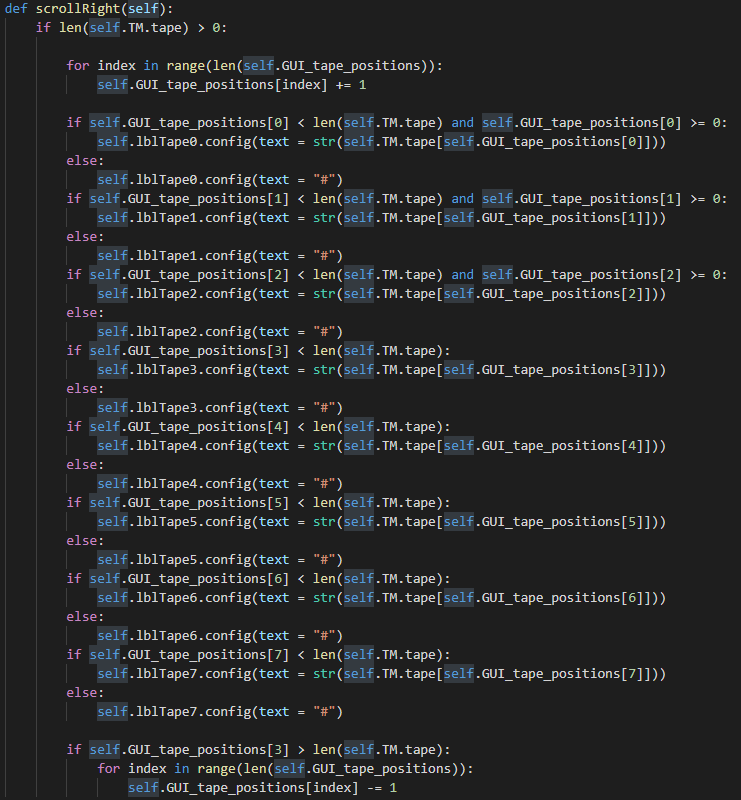
The tape can be saved to the machine in 2 ways, either by writing in the provided textbox or by loading the machine from a file. When saving a tape to the machine it will be writing using comma separated values between each value (square) on the tape, for example the format would be the following (1,0,1,0,0,1,0) 🡪 [1][0][1][0][0][1][0]. This works the same on the text file as it does the textbox. Once a tape has been added the GUI will update on the tape at the top of the window showing all of the values on the tape. When the machine has finished running it will then use a method to finalise the tape, this will remove all of the excess blank symbols on either side of the tape and format it correctly for presentation. The following screenshots show the code for receiving the tape input from the textbox and finalising the tape.





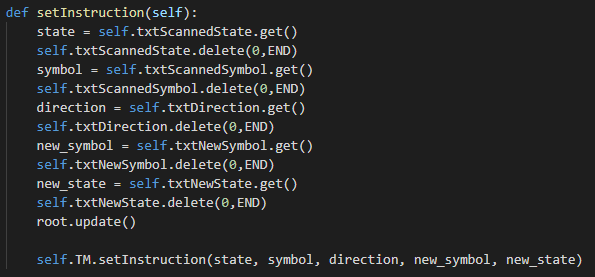
##### Directions:

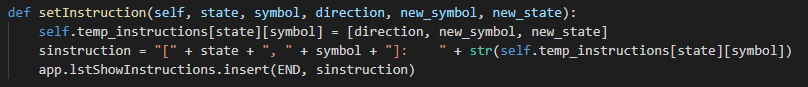
Next there are 2 arrows on either side of the tape which allow for navigation throughout the tape. One going left 1 square and one going right 1 square on the tape. This uses methods to scroll either left right on the tape whilst updating each square to the correct values. These methods also get called when the head moves left or right when the machine is running. There is list used to determine the positioning of the elements on the tape which will initially be [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7] (this will either increment or decrement depending if the machine were to move left or right. Here is a sample of code from scrolling left and right. The full code can be found in the appendix of the thesis, in brief there are several if statements which will check the length of the tape to ensure that the tape can fit on GUI and otherwise fill the empty squares with blank symbols. There is also methods to handle moving the tape left and right on the tape list in the Turing machine class, this simply increments or decrements the tape position.



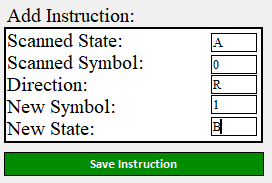
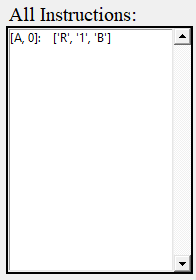


##### Saving Instructions

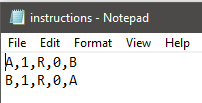
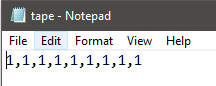
The instructions can be inserted into the machine using the provided textboxes on the GUI, This will allow the insertion of the current state, the current symbol, the direction to move the head, the new symbol and the new state. Whenever adding an instruction before starting the machine, all of the instructions will be saved into a temporary list which will allow the user to delete the instructions if they are not satisfied, this will then be dynamically added to available states when the machine start which will be covered in a further heading. When adding an instruction to the machine it will show in a listbox view in the centre to allow the user to see their progress. A save instruction button will be used once the user has finished editing the textboxes. The method to save an instruction will get all of the inputs from the user and then call a method in the Turing machine to add them to the available states; this will then create a single string to use in the listbox view, as shown below:

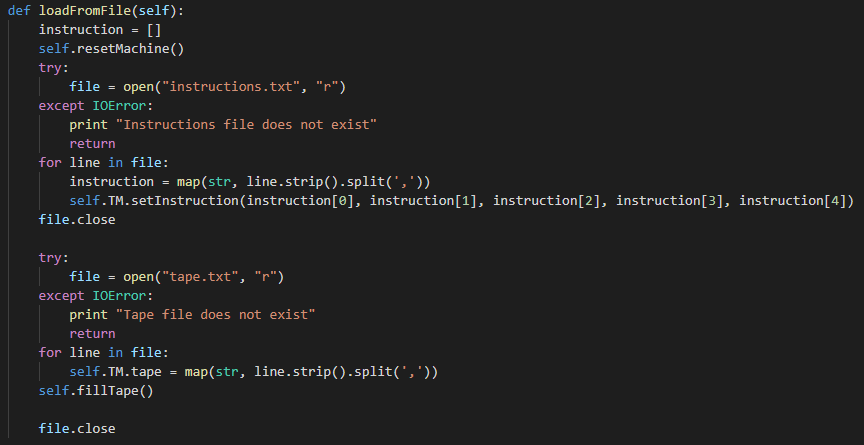


The format shown in the list box will show as:  
“[STATE, SYMBOL]: [DIRECTION, NEWSYMBOL, NEWSTATE]”   
The following screenshots show how the GUI will operate on saving the instructions to the machine:



##### Loading from a file

The Turing machine also has the ability to load a set of instructions and a tape from a text file. This is done using 2 files and requires the correct format of both the tape and instructions to load correctly. The instructions file will be formatted as the following with each instruction being separated line by line: STATE,SYMBOL,DIRECTION,NEW\_SYMBOL,NEW\_STATE. However, the tape will be written on a single line and will use a CSV format to separate the symbols on the tape. When loading from a file the machine will initially be reset to allow for the new configuration, meaning that the starting state and tape position will need to be set again if there was a custom value set. The files used would be “tape.txt” and “instructions.txt”, the following screenshots will show the format of the files and how the data is loaded into the machine.



##### Starting the machine

To start the machine the user will first have to make sure that the machine has a tape and a set of instructions. Once the user presses the “Run Machine” button the machine will reset the GUI tape positions and start the machine and initialise the machine. Initially the machine will check the tape position set for the machine and scroll right until the position is met on the GUI. This will then create a list of keys which will represent each of the states in the machine and will then set the starting state to on associated with the first instruction, this will only happen if there is no starting state in the machine already.



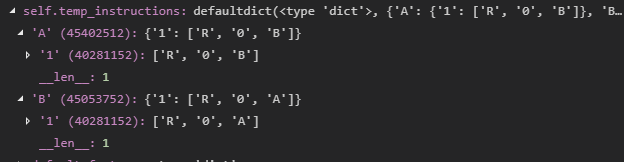


The machine will then cycle through all of the states in the machine and dynamically create the state in available states using the TuringState class. This means depending on the instructions added to the machine, it will dynamically create the needed states for the machine to transition to.

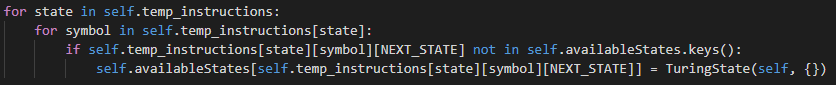


Using the TuringState class the machine is able to create as many different states as needed to allow the machine to run and each of these states will be saved to their own key in available states. The TuringState class will act as a template which will hold the different instructions associated with each state, for example; State A will have an instructions dictionary different to State B, meaning that each State will have its own unique set of instructions.

The temporary instructions uses a nested dictionary, This means that there is a dictionary key for each state in the machine, as well as a dictionary key for each symbol on the states of the machine. This allows for all of the instructions associated with each state to be passed through to each TuringState when saving to available states.



Next the machine will then look at what states the instructions are transitioning too and make sure that the state exists before starting the machine, to allow for the machine to transition correctly. This however will pass in an empty instruction as there are no instructions associated with the state.



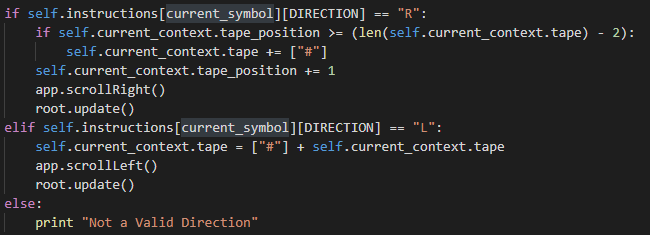
##### Running the machine

When the machine is running it will run through a while loop and attempt to run the executeInstruction() method which will handle each step the machine takes. This while loop will also handle if the machine ever enters a HALT state; if either a HALT state occurs or the executeInstruction() method returns false the machine will come to a halt and display a popup menu showing the final tape of the machine. This will then proceed to reset the machines attributes to use again. In the executeInstruction() method the machine will first check what the current symbol is on the tape and proceed to check if that symbol is associated with the current state (if there is an instruction that matches); this will then proceed to make actions on the tape. Firstly, the machine will display the current state, symbol and direction of the tape and then proceed to change the symbol on the current position on the tape. Once the tape has been changed, the machine will change to the next state associated with the instruction and move in the indicated direction.

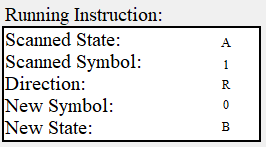
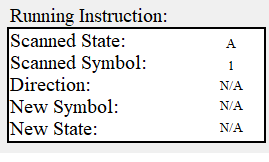




Once this has been executed the while loop will continue to run until the machine reaches a halt. To move a direction on the tape that machine change the tape position and the GUI tape positions, for example if the machine is moving right it will first check if there is enough space on the tape too move right and move accordingly, if there is no space to move right a blank symbol will be added to the end of the tape allowing the transition to happen. This simulates the tape as infinitely long. However, if the tape were to move left, a blank symbol will be added to the front of the tape and the tape position will not change as all of the values have been shifted along. Once the direction as changed the method for scrollRight() or scrollLeft() will trigger, therefore changing the GUI positions on the tape. If no direction is specified, the tape will not move.

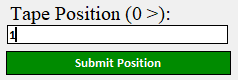


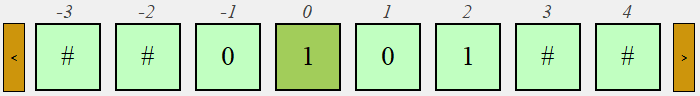
A series of sleeps are used throughout this method to allow for readability for the user. Between each sleep the GUI is updated to show the user what is currently happening in the machine, such as the direction moving, changing state, changing symbol, etc. This will allow for a user-friendly experience and help teach the user how the machine works and what is happening at each stage, the screenshot below shows the GUI interface being updated

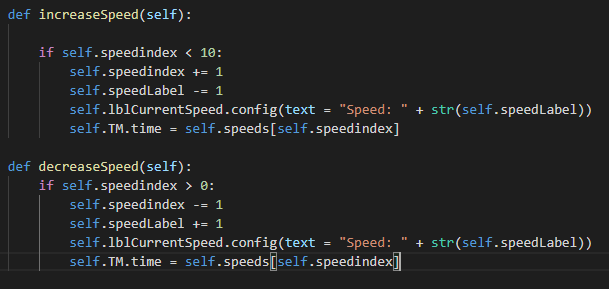


##### Extra features (position, state, speed, popup final, help)

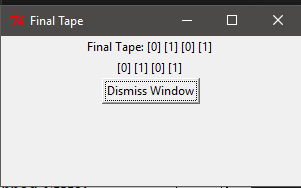
Including the works of the machine above, there were a few extra features added to allow for the best customisability in the machine. The first extra feature(s) was the ability to change the state and the position of the machine before starting. These were handled through textboxes allowing the user to write either a tape position or starting state for the machine. 2 labels at the bottom of the window were also used to show the user what the current attributes were set to. When setting a starting position on the tape, the machine would navigate the GUI tape until the tape head is under that position.



The next extra feature added was to change the speed of the machine, this uses 2 buttons; one to increment the speed and one to decrement the speed. This would change the sleep intervals between each of the updates to the GUI. There was a cap made between 0 and 10, to not allow for the speed to exceed below the minus’. This was handled through 2 methods and when the speed is incremented, the sleep interval will be reduced (all speeds were held in a list).



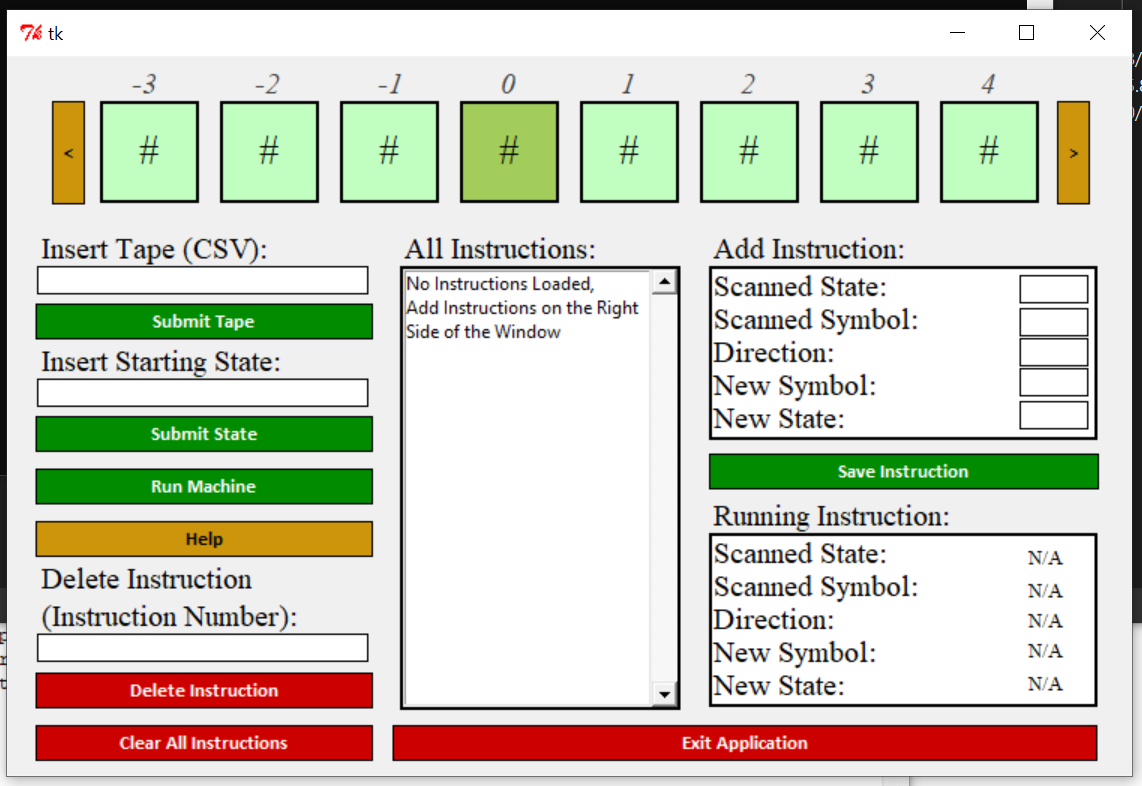
A final popup window was also added to the machine which will (on completion) display a final tape. This would then reset the machine to its initial attributes to allow for the user to try any configuration.

Finally, a help window was produced, which would give examples and explain to the user how to use the machine. This would show directions on what everything means on the machine such as the instructions, tape, states, positioning, GUI aspects, etc.



### Libraries and GUI

Only a few select libraries were used in the creation of these prototypes, one of which being the “time” library; this was used to handle how the instructions and tape would be presented on either the GUI or the CLI, using the sleep method it was possible to add delays between each action of the machine allowing GUI to be more readable. Regarding the GUI, Tkinter has been used for the majority of the project, as they’re was a major limitation with using PyQt; the interface was having trouble refreshing whilst the Turing machine was running, as the methods needed to move the tape on the GUI were being called from a different class. However due to the way Tkinter is handled, the window can be manged from anywhere in the program, allowing the screen to refresh without any issues. An extension of Tkinter was also used to allow a message box to show on the screen to display the final tape. The next prototype of the machine will use a different and friendlier GUI than the previous interpretations which has already been programmed using Tkinter ready for implementation, as shown below:



# Critical evaluation

## Testing

### Finite Models

When testing the programs, all of the finite models used very similar code and the testing is the same for each. The first test made on each of the finite models was for the tapes selected showing the correct final tape when the machine has terminated. All of the tapes showed the correct outputs for the tapes given. The screenshots below show the final tape popup window for each of the tapes in each machine (The odd number Turing Machine will not produce a final tape for the second tape as it will infinitely loop due to the even number of 0’s):

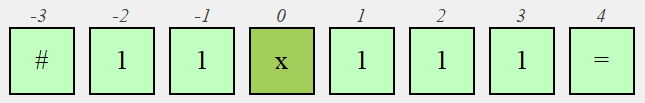
 

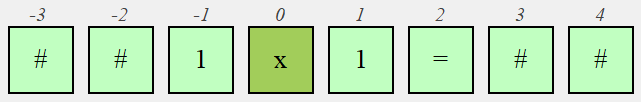


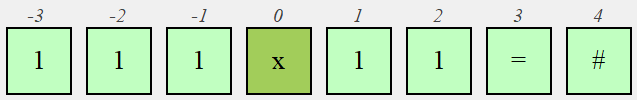
 

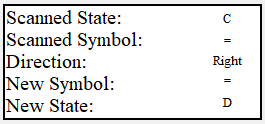
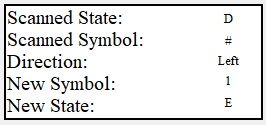
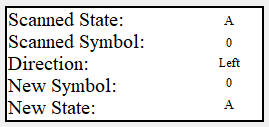
The next test that was made was on the Unary Multiplication program as this changes the tape position when selecting a tape, this would be tested to see if the GUI tape is shown correctly. This worked correctly and on each tape the position was changed to where the (x) was on the tape.



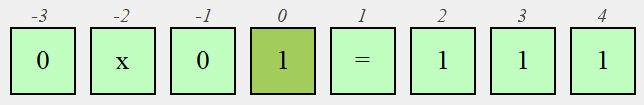


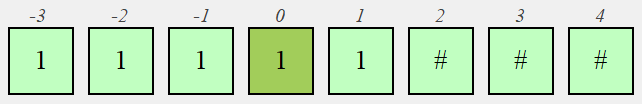


The next test was to check if in each of the machines the GUI instructions were updated correctly for while the machine was running. This worked correctly and on each transition between state the GUI Instructions were slowly updated to allow for readability.

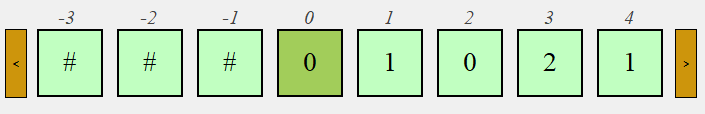
Another test made was to check if the tape correctly moved either left or right and updated the values accordingly. On each of the machines this worked correctly, moving left was not used on some machines but worked correctly when it was used.

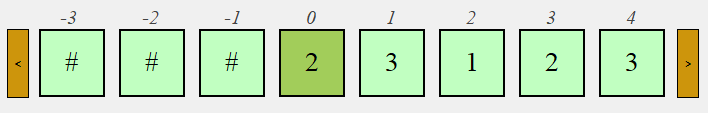




### Dynamic Machine

Similar tests were made for the dynamic machine, however there were a few extra tests in place for the extra features of the machine to make it dynamic such as having a custom tape, instructions, etc. The first test carried out was for the insert tape option, this would allow the user to have a custom tape for the machine using CSV. This will be tested through inputting several values into the machine separated by commas and should appear on the GUI tape correctly. The results below show that the tape was updated correctly.

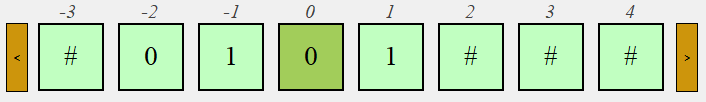
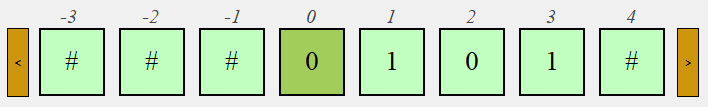
 

The next test was for setting that starting state of the machine. This would update a label at the bottom of the window which is set to display the current state of the machine, the results show that the starting state was correctly updated.

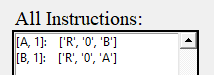
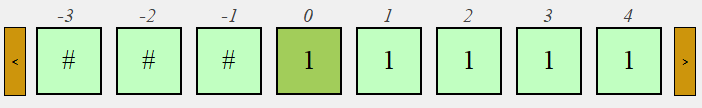
Next the tape position option was tested allowing the user to state a custom position on the tape. This will scroll right until the position is reached and will start the machine in this position. A Label at the bottom of the window will also show the current tape position. The results of this test show that the machine correctly moved to the designated position on the tape.

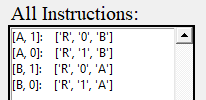
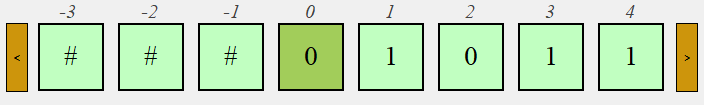


Another test made was for if the speed actually altered how fast the machine ran and should have a cap of between 0 and 10. The results of this test showed that the machine would run instantly on max speed and would vary depending on the setting. The GUI label also updated correctly.

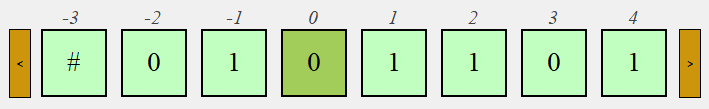
  

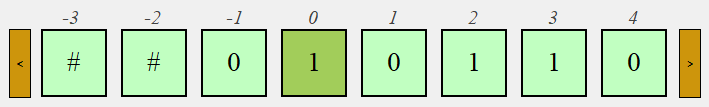
The loading from file functionality was also tested, when the instructions are loaded it should appear correctly in the listbox for instructions, as well as load the correct tape into the GUI at the top of the window. This will also reset the machine, including tape position and starting state (if set before). 2 sets of instructions were tested, and both were added to the machine correctly. The tap also showed correctly on the GUI once loaded.

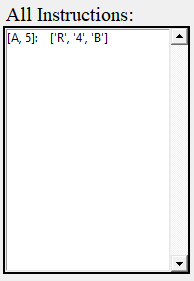
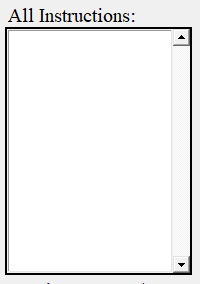
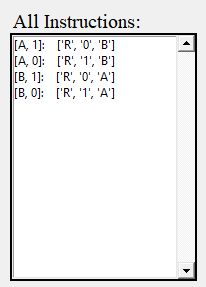
 

The next test was for the directions on either side of the tape, this should allow the user to navigate through the tape left or right, and should reset once the machine is running and not mess with the synchronisation of moving the tape.

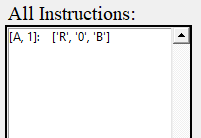
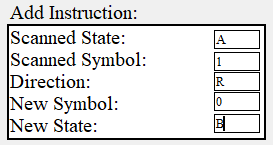


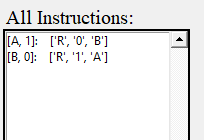
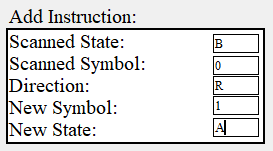


Clearing the instructions should also remove all of the instructions in the listbox and the machine, adding new instructions should load a new set of instructions.

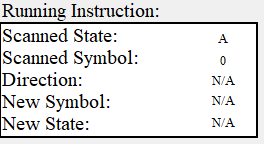
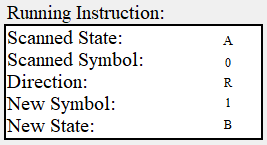


Add instructions should allow the user to set a state, symbol, direction, new symbol, and a new state. This should be appended to the all instructions listbox. If some of the textboxes are empty an empty field will be added as an instruction, this allows for the direction to be blank if needed (machine will not move).





Running the machine should run through the set of instructions and each symbol on the tape and change it accordingly, the instructions should show one by one on the running instruction part of the GUI. The tape should also change accordingly and move in the correct directions depending on instructions.

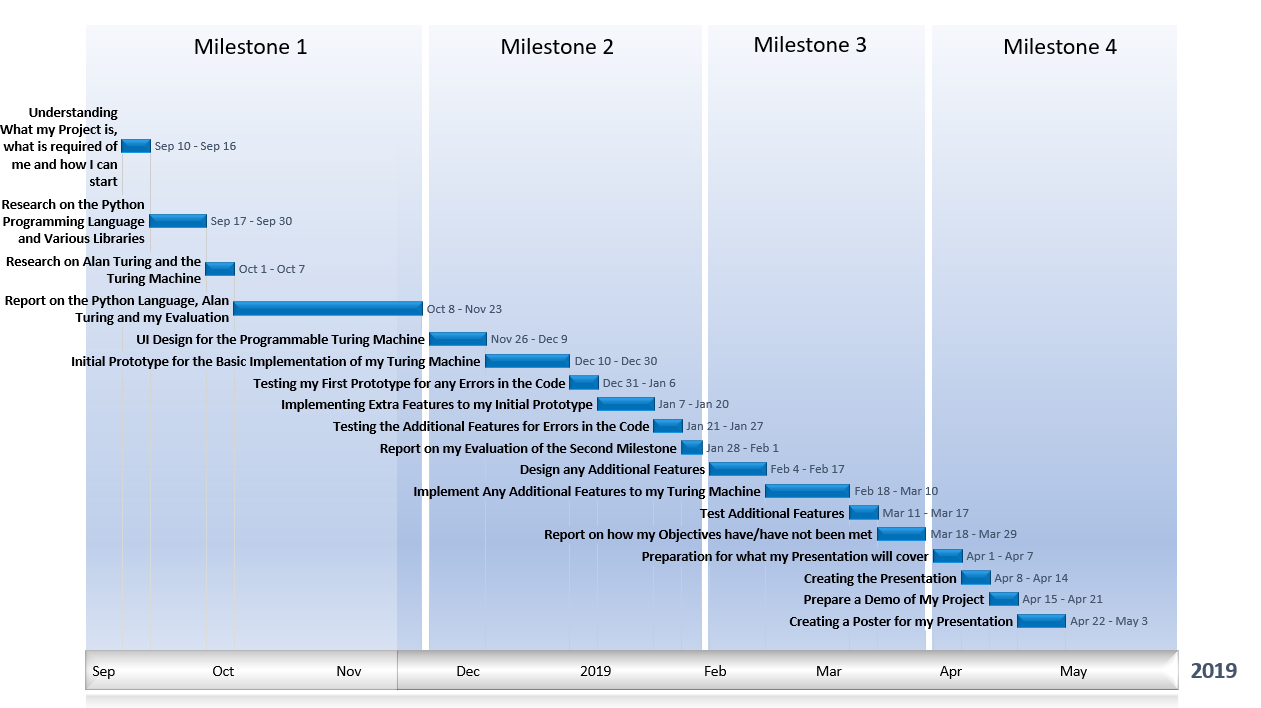
 

# Conclusion and Future Work

Alan Turing was one of the major contributors to modern computer science, with his invention of the Turing machine which can in theory compute anything given the right amount of tape and time no matter the complexity, the machine will run when supplied with instructions and a tape (memory). One of the main objectives for this thesis was to produce a dynamic Turing machine which will have full customisability of the machine. This was produced by using the state design pattern to handle which Turing states the machine can transition to. Extra features were also produced giving more customisability to the machine such as the ability to change the speed of the machine, state diagrams for the finite models and customisability to the state and tape positions. The machine is also informative, in teaching the user the basics of the Turing machine and how to use the program. In conclusion the Turing machine has been proven to be a powerful tool in computation as depending on what instructions and tape is provided, the machine could in theory do anything and the dynamic model produced allows the user to do so.

Future work for this project would be to perhaps produce the machine in another language where the GUI can be improved to a better standard, by showing a state diagram for each of the machines and allowing the user to see what is happening in the machine by following the diagram. The machine may show different parts of the state diagram glow to depending on where the machine is in terms of reading instructions. More finite machines could also be implemented to help the user learn how the Turing machine works with more complex state diagrams.

## Timeline



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Milestone(s)** | **Start Week** | **End Week** | **Description** | **Duration (Weeks)** |
| *Milestone 1* | 10/09/2018 | 16/09/2018 | Understanding What my Project is, what is required of me and how I can start | 2 |
| 17/09/2018 | 30/09/2018 | Research on the Python Programming Language and Various Libraries | 2 |
| 01/10/2018 | 07/10/2018 | Research on Alan Turing and the Turing Machine | 3 |
| 08/10/2018 | 23/11/2018 | Report on the Python Language, Alan Turing and my Evaluation | 4 |
| *Milestone 2* | 26/11/2018 | 09/12/2018 | UI Design for the Programmable Turing Machine | 2 |
| 10/12/2018 | 30/12/2018 | Initial Prototype for the Basic Implementation of my Turing Machine | 3 |
| 31/12/2018 | 06/01/2019 | Testing my First Prototype for any Errors in the Code | 1 |
| 07/01/2019 | 20/01/2019 | Implementing Extra Features to my Initial Prototype | 2 |
| 21/01/2019 | 27/01/2019 | Testing the Additional Features for Errors in the Code | 1 |
| 28/01/2019 | 01/02/2019 | Report on my Evaluation of the Second Milestone | 1 |
| *Milestone 3* | 04/02/2019 | 17/02/2019 | Design any Additional Features | 2 |
| 18/02/2019 | 10/03/2019 | Implement Any Additional Features to my Turing Machine | 3 |
| 11/03/2019 | 17/03/2019 | Test Additional Features | 1 |
| 18/03/2019 | 29/03/2019 | Report on how my Objectives have/have not been met | 2 |
| *Milestone 4* | 01/04/2019 | 07/04/2019 | Preparation for what my Presentation will cover | 1 |
| 08/04/2019 | 14/04/2019 | Creating the Presentation | 1 |
| 15/04/2019 | 21/04/2019 | Prepare a Demo of My Project | 1 |
| 22/04/2019 | 03/05/2019 | Creating a Poster for my Presentation | 1 |

# Appendix

## References

Python (2018) *24.1. Tkinter – Python interface to Tcl/Tk*. Available at: <https://docs.python.org/2/library/tkinter.html> (Accessed 27 October 2018).

Chun, W, J. (2006) *Core Python Programming: Second Edition.* USA, Massachusetts: Pearson Education, Inc.

Lutz, Mark. (2014) *Python Pocket Reference: Fifth Edition.* United States of America: O’Reilly Media, Inc.

Dunn, R. (2018) *wxPython 4.0.3*. Available at: <https://pypi.org/project/wxPython/#description> (Accessed: 27 October 2018).

Dunn, R. (2017) *wxPython Overview*. Available at: <https://docs.wxwidgets.org/3.1/overview_python.html> (Accessed: 27 October 2018).

Kivy (2018) *Getting Started* .Available at: <https://kivy.org/doc/stable/gettingstarted/intro.html> (Accessed: 27 October 2018).

Python (2018) *About PyQt* .Available at: <https://wiki.python.org/moin/PyQt> (Accessed: 27 October 2018).

Summerfield, M. (2008) *Rapid GUI Programming with Python and Qt.* United States of America: Pearson Education, Inc.

Qt Company Ltd (2018*) Qt Documentation*. Available at: <http://doc.qt.io/qt-5/qtdesigner-manual.html> (Accessed: 27 October 2018).

TIOBE (2018). *TIOBE Index for October 2018.* Available at: <https://www.tiobe.com/tiobe-index/> (Accessed: 03 November 2018).

Mindfire Solutions (2017). *Python: 7 Important Reasons Why You Should Use Python*. Available at: <https://medium.com/@mindfiresolutions.usa/python-7-important-reasons-why-you-should-use-python-5801a98a0d0b> (Accessed: 03 November 2018).

Watson, I. (2012) *How Alan Turing Invented the Computer Age*. Available at: <https://blogs.scientificamerican.com/guest-blog/how-alan-turing-invented-the-computer-age/> (Accessed: 04 November 2018)

Mullins, R. (2012) *What is a Turing Machine*. Available at: <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/turing-machine/one.html> (Accessed: 04 November 2018)

Petzold, C. (2008) *The Annotated Turing.* Indianapolis, Indiana. Wiley Publishing, Inc.

Davis, M (1965). *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions,* New York. Raven Press Books, Ltd.

Sipser, M. (2013) *Introduction of the Theory of Computation, Third Edition.* United States of America. PWS Publishing Inc

Schafer, S. (2016*) What is a Multitrack Turing machine*? Available at: <https://www.quora.com/What-is-a-Multitrack-Turing-machine> (Accessed: 10 November 2018)

Sudkamp, T. (2006) *Languages and machines: an introduction to the theory of computer science, Third Edition.* Boston, Pearson Addison-Wesley

Hom, E (2013) *Alan Turing Biography: Computer Pioneer, Gay Icon.* Available at: <https://www.livescience.com/29483-alan-turing.html> (Accessed: 13 November 2018)

Graham-Cumming, J. (2010) *Lets Build Babbage’s Ultimate Mechanical Computer.* Available at: <https://www.newscientist.com/article/mg20827915-500-lets-build-babbages-ultimate-mechanical-computer/> (Accessed: 13 November 2018)

Flury, T. (2017) *What is a Turing Complete Machine?* Available at: <https://www.quora.com/What-is-a-Turing-complete-machine> (Accessed: 13 November 2018)

Jackson, J. (2012) *How Alan Turing Set the Rules for Computing.* Available at: <https://www.pcworld.com/article/258232/how_alan_turing_set_the_rules_for_computing.html> (Accessed: 13 November 2018)

James, M. (2017) *What is a Turing Machine?* Available at: <https://www.i-programmer.info/babbages-bag/23-turing-machines.html> (Accessed: 23 January 2019)

ScienceDirect (2013), *Entscheidungsproblem.* Available at: <https://www.sciencedirect.com/topics/mathematics/entscheidungsproblem> (Accessed: 23 March 2019).

Moore, Koswara and Williams (*no date*) *Halting Problem.* Available at: <https://brilliant.org/wiki/halting-problem/> (Accessed: 23 March 2019)

## Prototype 1

from PyQt4 import QtGui

from PyQt4.QtCore import Qt

import sys

from time import sleep

class TuringMachine():

def \_\_init\_\_(self):

self.tape = []

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7] # positions of the tape, which will change depending on left or right (GUI)

self.str\_instructions = []

self.instructions = []

self.tape\_position = 0 # tape position (machine)

def saveInstructions(self):

templist = []

templist.append(str(self.txtScannedState.text()))

templist.append(str(self.txtScannedSymbol.text()))

templist.append(str(self.cbDirection.currentText()))

templist.append(str(self.txtNewSymbol.text()))

templist.append(str(self.txtNewState.text()))

self.instructions.append(templist)

self.str\_instructions.append("Scanned State: " + str(self.txtScannedState.text()) + " " +

"Scanned Symbol: " + str(self.txtScannedSymbol.text()) + " ")

self.str\_instructions.append("Direction: " + str(self.cbDirection.currentText()))

self.str\_instructions.append("New Symbol: " + str(self.txtNewSymbol.text()) + " " +

"New State: " + str(self.txtNewState.text()))

self.txtScannedState.setText("")

self.txtScannedSymbol.setText("")

self.txtDirection.setText("")

self.txtNewSymbol.setText("")

self.txtNewState.setText("")

print "Current Instructions: " + str(self.instructions)

self.fillInstructions()

def runMachine(self):

self.btnRight.setEnabled(False)

self.btnLeft.setEnabled(False)

self.btnSubmit.setEnabled(False)

self.btnExit.setEnabled(False)

self.txtScannedState.setEnabled(False)

self.txtScannedSymbol.setEnabled(False)

self.txtDirection.setEnabled(False)

self.txtNewSymbol.setEnabled(False)

self.txtNewState.setEnabled(False)

self.txtDeleteInstruction.setEnabled(False)

app.processEvents()

current\_instruction\_set = []

instructionfound = False

current\_state = "A" # start state

while current\_state != "H" and len(self.tape) > 0:

app.processEvents()

current\_symbol = self.tape[self.tape\_position]

for index in range(len(self.instructions)): # goes through the instructions

if self.instructions[index][0] == current\_state and self.instructions[index][1] == current\_symbol: # if both the symbol and state match the instruction

current\_instruction\_set = self.instructions[index] # make a copy of that instruction to be executed

instructionfound = True

if instructionfound == True:

self.btnRight.setEnabled(False)

self.btnLeft.setEnabled(False)

self.btnSubmit.setEnabled(False)

self.btnExit.setEnabled(False)

self.txtScannedState.setEnabled(False)

self.txtScannedSymbol.setEnabled(False)

self.txtDirection.setEnabled(False)

self.txtNewSymbol.setEnabled(False)

self.txtNewState.setEnabled(False)

self.txtDeleteInstruction.setEnabled(False)

print "Current Tape: " + str(self.tape)

print "Scanned Symbol: " + str(current\_symbol)

print "Scanned State: " + str(current\_state)

self.lblScannedState.setText(str(current\_instruction\_set[0]))

sleep(0.5)

app.processEvents()

self.lblScannedSymbol.setText(str(current\_instruction\_set[1]))

sleep(0.5)

app.processEvents()

self.lblDirection.setText(str(current\_instruction\_set[2]))

sleep(0.5)

app.processEvents()

self.lblNewSymbol.setText(str(current\_instruction\_set[3]))

sleep(0.5)

app.processEvents()

self.lblNewState.setText(str(current\_instruction\_set[4]))

sleep(0.5)

app.processEvents()

self.tape[self.tape\_position] = current\_instruction\_set[3] # assigns new value to the tape

print "New Value on Square: " + str(self.tape[self.tape\_position])

current\_state = current\_instruction\_set[4] # assigns new state to machine

print "New State of Machine: " + str(current\_state)

if current\_instruction\_set[2] == "R" and current\_instruction\_set[4] != "H":

if self.tape\_position == len(self.tape):

self.tape.append('#')

self.moveRight()

sleep(1)

print "Moving Right"

print "--------------------------------------"

print ""

instructionfound = False

elif current\_instruction\_set[2] == "L" and current\_instruction\_set[4] != "H":

if self.tape\_position == -1:

self.tape\_position = 0

self.moveLeft()

sleep(1)

print "Moving Left"

print "--------------------------------------"

print ""

instructionfound = False

if self.tape\_position < 0 or self.tape\_position > len(self.tape):

self.final\_tape()

print ""

print "<--- End of Tape --->"

print "Final Tape: " + str(self.tape)

else:

self.final\_tape()

print ""

print "<--- End of Tape --->"

print "Final Tape: " + str(self.tape)

break

else:

self.final\_tape()

print ""

print "<--- End of Tape --->"

print "Final Tape: " + str(self.tape)

self.reset\_tape()

self.btnRight.setEnabled(True)

self.btnLeft.setEnabled(True)

self.btnSubmit.setEnabled(True)

self.btnExit.setEnabled(True)

self.txtScannedState.setEnabled(True)

self.txtScannedSymbol.setEnabled(True)

self.txtDirection.setEnabled(True)

self.txtNewSymbol.setEnabled(True)

self.txtNewState.setEnabled(True)

self.txtDeleteInstruction.setEnabled(True)

app.processEvents()

class CustomTapeWindow(QtGui.QMainWindow, TuringMachine):

def \_\_init\_\_(self, parent = None):

super(CustomTapeWindow, self).\_\_init\_\_(parent)

TuringMachine.\_\_init\_\_(self)

QtGui.QApplication.setStyle(QtGui.QStyleFactory.create("Plastique"))

self.setGeometry(700, 400, 1070, 295)

self.setFixedSize(1070, 295)

self.setWindowTitle("Write a Custom Tape")

self.setWindowFlags(Qt.Window | Qt.WindowTitleHint | Qt.CustomizeWindowHint)

self.setStyleSheet("background-color: #dedddd;")

self.scroll\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7] # positions of the tape, which will change depending on left or right

self.drawWindow()

def drawWindow(self):

self.lblInstructionTitle = QtGui.QLabel("Running Instructions:", self)

self.lblInstructionTitle.move(820, 2)

self.lblInstructionTitle.resize(self.lblInstructionTitle.sizeHint())

self.lblInstructionBorder = QtGui.QLabel(self)

self.lblInstructionBorder.setWordWrap(True)

self.lblInstructionBorder.setText(" Scanned State: \n\n"

" Scanned Symbol: \n\n"

" Direction: \n\n"

" New Symbol: \n\n"

" New State: \n\n")

self.lblInstructionBorder.setStyleSheet(

"font-size: 20px; "

"border: 2px solid #000000;"

"border-style: dashed hidden hidden dashed;"

"background-color: #ededed;")

self.lblInstructionBorder.setFont(QtGui.QFont('Times', 40))

self.lblInstructionBorder.resize(250, 300)

self.lblInstructionBorder.move(820, 20)

self.lblScannedState = QtGui.QLabel("N/A", self)

self.lblScannedState.resize(50, 50)

self.lblScannedState.move(1000, 30)

self.lblScannedState.setAlignment(Qt.AlignCenter)

self.lblScannedState.setStyleSheet("background-color: #ededed;")

self.lblScannedState.setFont(QtGui.QFont('Times', 20))

self.lblScannedSymbol = QtGui.QLabel("N/A", self)

self.lblScannedSymbol.resize(50, 50)

self.lblScannedSymbol.move(1000, 75)

self.lblScannedSymbol.setAlignment(Qt.AlignCenter)

self.lblScannedSymbol.setStyleSheet("background-color: #ededed;")

self.lblScannedSymbol.setFont(QtGui.QFont('Times', 20))

self.lblDirection = QtGui.QLabel("N/A", self)

self.lblDirection.resize(50, 50)

self.lblDirection.move(1000, 120)

self.lblDirection.setAlignment(Qt.AlignCenter)

self.lblDirection.setStyleSheet("background-color: #ededed;")

self.lblDirection.setFont(QtGui.QFont('Times', 20))

self.lblNewSymbol = QtGui.QLabel("N/A", self)

self.lblNewSymbol.resize(50, 50)

self.lblNewSymbol.move(1000, 165)

self.lblNewSymbol.setAlignment(Qt.AlignCenter)

self.lblNewSymbol.setStyleSheet("background-color: #ededed;")

self.lblNewSymbol.setFont(QtGui.QFont('Times', 20))

self.lblNewState = QtGui.QLabel("N/A", self)

self.lblNewState.resize(50, 50)

self.lblNewState.move(1000, 210)

self.lblNewState.setAlignment(Qt.AlignCenter)

self.lblNewState.setStyleSheet("background-color: #ededed;")

self.lblNewState.setFont(QtGui.QFont('Times', 20))

self.lblInfinite0 = QtGui.QLabel("...", self)

self.lblInfinite0.resize(50, 50)

self.lblInfinite0.move(253, 215)

self.lblInfinite0.setAlignment(Qt.AlignCenter)

self.lblInfinite0.setFont(QtGui.QFont('Times', 40))

self.lblTape0 = QtGui.QLabel("#", self)

self.lblTape0.resize(50, 50)

self.lblTape0.setStyleSheet(

"border:3px solid #000000;"

"background-color: #ffffff;") # css sheet

self.lblTape0.move(302, 230)

self.lblTape0.setAlignment(Qt.AlignCenter)

self.lblTape0.setFont(QtGui.QFont('Times', 20))

self.lblTape1 = QtGui.QLabel("#", self)

self.lblTape1.resize(50, 50)

self.lblTape1.setStyleSheet(

"border:3px solid #000000;"

"background-color: #ffffff;") # css sheet

self.lblTape1.move(357, 230)

self.lblTape1.setAlignment(Qt.AlignCenter)

self.lblTape1.setFont(QtGui.QFont('Times', 20))

self.lblTape2 = QtGui.QLabel("#", self)

self.lblTape2.resize(50, 50)

self.lblTape2.setStyleSheet(

"border:3px solid #000000;"

"background-color: #ffffff;") # css sheet

self.lblTape2.move(417, 230)

self.lblTape2.setAlignment(Qt.AlignCenter)

self.lblTape2.setFont(QtGui.QFont('Times', 20))

self.lblTape3 = QtGui.QLabel("#", self)

self.lblTape3.resize(50, 50)

self.lblTape3.setStyleSheet(

"border: 7px solid #ff0000;"

"background-color: #ffffff;") # css sheet

self.lblTape3.move(477, 230)

self.lblTape3.setAlignment(Qt.AlignCenter)

self.lblTape3.setFont(QtGui.QFont('Times', 20))

self.lblTape4 = QtGui.QLabel("#", self)

self.lblTape4.resize(50, 50)

self.lblTape4.setStyleSheet(

"border:3px solid #000000;"

"background-color: #ffffff;") # css sheet

self.lblTape4.move(537, 230)

self.lblTape4.setAlignment(Qt.AlignCenter)

self.lblTape4.setFont(QtGui.QFont('Times', 20))

self.lblTape5 = QtGui.QLabel("#", self)

self.lblTape5.resize(50, 50)

self.lblTape5.setStyleSheet(

"border:3px solid #000000;"

"background-color: #ffffff;") # css sheet

self.lblTape5.move(597, 230)

self.lblTape5.setAlignment(Qt.AlignCenter)

self.lblTape5.setFont(QtGui.QFont('Times', 20))

self.lblTape6 = QtGui.QLabel("#", self)

self.lblTape6.resize(50, 50)

self.lblTape6.setStyleSheet(

"border:3px solid #000000;"

"background-color: #ffffff;") # css sheet

self.lblTape6.move(657, 230)

self.lblTape6.setAlignment(Qt.AlignCenter)

self.lblTape6.setFont(QtGui.QFont('Times', 20))

self.lblTape7 = QtGui.QLabel("#", self)

self.lblTape7.resize(50, 50)

self.lblTape7.setStyleSheet(

"border:3px solid #000000;"

"background-color: #ffffff;") # css sheet

self.lblTape7.move(717, 230)

self.lblTape7.setAlignment(Qt.AlignCenter)

self.lblTape7.setFont(QtGui.QFont('Times', 20))

self.lblInfinite1 = QtGui.QLabel("...", self)

self.lblInfinite1.resize(50, 50)

self.lblInfinite1.move(770, 215)

self.lblInfinite1.setAlignment(Qt.AlignCenter)

self.lblInfinite1.setFont(QtGui.QFont('Times', 40))

self.btnRun = QtGui.QPushButton("Run", self)

self.btnRun.clicked.connect(self.runMachine)

self.btnRun.resize(330, 20)

self.btnRun.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: green; ")

self.btnRun.move(370, 200)

self.txtTape = QtGui.QLineEdit(self)

self.txtTape.setStyleSheet(

"border: 1px solid #000000;"

"background-color: #ffffff;")

self.txtTape.resize(440, 20)

self.txtTape.move(260, 170)

self.lblTape = QtGui.QLabel("Insert Tape: ", self)

self.lblTape.move(260, 145)

self.lblTape.setStyleSheet("background-color: rgba(0,0,0,0%);") #transparent

self.btnSubmit = QtGui.QPushButton("Submit", self)

self.btnSubmit.clicked.connect(self.submit)

self.btnSubmit.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: green; ")

self.btnSubmit.resize(100, 20)

self.btnSubmit.move(710, 170)

self.btnExit = QtGui.QPushButton("Return To Menu", self)

self.btnExit.clicked.connect(self.close)

self.btnExit.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: #901414; ")

self.btnExit.resize(150, 20)

self.btnExit.move(873, 263)

self.btnRight = QtGui.QPushButton("--->", self)

self.btnRight.clicked.connect(self.moveRight)

self.btnRight.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: #bf6b18; ")

self.btnRight.resize(100, 20)

self.btnRight.move(710,200)

self.btnLeft = QtGui.QPushButton("<---", self)

self.btnLeft.clicked.connect(self.moveLeft)

self.btnLeft.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: #bf6b18; ")

self.btnLeft.resize(100,20)

self.btnLeft.move(260,200)

self.lblAddInstructions = QtGui.QLabel("Add an Instruction:", self)

self.lblAddInstructions.move(5, 2)

self.lblAddInstructions.resize(self.lblAddInstructions.sizeHint())

self.lblAddInstructionBorder = QtGui.QLabel(self)

self.lblAddInstructionBorder.setWordWrap(True)

self.lblAddInstructionBorder.setText(" Scanned State: \n\n"

" Scanned Symbol: \n\n"

" Direction: \n\n"

" New Symbol: \n\n"

" New State: \n\n")

self.lblAddInstructionBorder.setStyleSheet(

"font-size: 20px; "

"border: 2px solid #000000;"

"border-style: dashed dashed hidden hidden;"

"background-color: #ededed;")

self.lblAddInstructionBorder.setFont(QtGui.QFont('Times', 40))

self.lblAddInstructionBorder.resize(250, 300)

self.lblAddInstructionBorder.move(0, 20)

self.txtScannedState = QtGui.QLineEdit(self)

self.txtScannedState.resize(50, 40)

self.txtScannedState.move(180, 30)

self.txtScannedState.setAlignment(Qt.AlignCenter)

self.txtScannedState.setStyleSheet("background-color: #ededed;")

self.txtScannedState.setFont(QtGui.QFont('Times', 20))

self.txtScannedSymbol = QtGui.QLineEdit(self)

self.txtScannedSymbol.resize(50, 40)

self.txtScannedSymbol.move(180, 75)

self.txtScannedSymbol.setAlignment(Qt.AlignCenter)

self.txtScannedSymbol.setStyleSheet("background-color: #ededed;")

self.txtScannedSymbol.setFont(QtGui.QFont('Times', 20))

self.txtDirection = QtGui.QLineEdit(self)

self.txtDirection.resize(50, 40)

self.txtDirection.move(180, 120)

self.txtDirection.setAlignment(Qt.AlignCenter)

self.txtDirection.setStyleSheet("background-color: #ededed;")

self.txtDirection.setFont(QtGui.QFont('Times', 20))

self.cbDirection = QtGui.QComboBox(self)

self.cbDirection.addItems(["None", "R", "L"])

self.cbDirection.move(130, 120)

self.cbDirection.resize(100, 40)

self.cbDirection.setStyleSheet("background-color: #ededed;")

self.cbDirection.setFont(QtGui.QFont('Times', 15))

self.txtNewSymbol = QtGui.QLineEdit(self)

self.txtNewSymbol.resize(50, 40)

self.txtNewSymbol.move(180, 165)

self.txtNewSymbol.setAlignment(Qt.AlignCenter)

self.txtNewSymbol.setStyleSheet("background-color: #ededed;")

self.txtNewSymbol.setFont(QtGui.QFont('Times', 20))

self.txtNewState = QtGui.QLineEdit(self)

self.txtNewState.resize(50, 40)

self.txtNewState.move(180, 210)

self.txtNewState.setAlignment(Qt.AlignCenter)

self.txtNewState.setStyleSheet("background-color: #ededed;")

self.txtNewState.setFont(QtGui.QFont('Times', 20))

self.btnSubmitInstruction = QtGui.QPushButton("Submit Instruction", self)

self.btnSubmitInstruction.clicked.connect(self.saveInstructions)

self.btnSubmitInstruction.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: green; ")

self.btnSubmitInstruction.resize(150, 20)

self.btnSubmitInstruction.move(50, 263)

self.lblViewInstructions = QtGui.QLabel("Saved Instructions:", self)

self.lblViewInstructions.move(260, 2)

self.lblViewInstructions.resize(self.lblViewInstructions.sizeHint())

self.txtViewInstructions = QtGui.QTextEdit(self)

self.txtViewInstructions.resize(440, 130)

self.txtViewInstructions.move(260, 20)

self.txtViewInstructions.setStyleSheet("background-color: #ededed;")

self.txtViewInstructions.setFont(QtGui.QFont('Times', 14))

self.txtViewInstructions.setReadOnly(True)

self.txtViewInstructions.append("No Instructions Loaded")

self.txtViewInstructions.append("Add Instructions on the Right Side of the Window")

self.lblDeleteAllInstructions = QtGui.QLabel("Delete Instructions:", self)

self.lblDeleteAllInstructions.move(710, -5)

self.lblDeleteAllInstructions.setStyleSheet("background-color: rgba(0,0,0,0%);") # transparent

self.btnClearInstructions = QtGui.QPushButton("Clear Instructions", self)

self.btnClearInstructions.clicked.connect(self.clearInstructions)

self.btnClearInstructions.resize(100, 20)

self.btnClearInstructions.move(710, 20)

self.btnClearInstructions.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: #901414;")

self.btnClearInstructions.setFont(QtGui.QFont('SansSerif', 7.5))

self.lblDeleteInstruction = QtGui.QLabel("Delete Instruction:", self)

self.lblDeleteInstruction.move(710, 35)

self.lblDeleteInstruction.setStyleSheet("background-color: rgba(0,0,0,0%);") # transparent

self.txtDeleteInstruction = QtGui.QLineEdit(self)

self.txtDeleteInstruction.move(710, 60)

self.txtDeleteInstruction.resize(100, 60)

self.txtDeleteInstruction.setStyleSheet("background-color: #ededed;")

self.txtDeleteInstruction.setFont(QtGui.QFont('Times', 16))

self.txtDeleteInstruction.setAlignment(Qt.AlignCenter)

self.btnDeleteInstruction = QtGui.QPushButton("Delete Instruction", self)

self.btnDeleteInstruction.clicked.connect(self.deleteInstruction)

self.btnDeleteInstruction.resize(100, 20)

self.btnDeleteInstruction.move(710, 130)

self.btnDeleteInstruction.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: #901414;")

self.btnDeleteInstruction.setFont(QtGui.QFont('SansSerif', 7.5))

self.fillTape()

def fillTape(self):

self.lblTape0.setText("#")

self.lblTape1.setText("#")

self.lblTape2.setText("#")

self.lblTape3.setText("#")

self.lblTape4.setText("#")

self.lblTape5.setText("#")

self.lblTape6.setText("#")

self.lblTape7.setText("#")

if len(self.tape) >= 5:

self.lblTape3.setText(str(self.tape[0]))

self.lblTape4.setText(str(self.tape[1]))

self.lblTape5.setText(str(self.tape[2]))

self.lblTape6.setText(str(self.tape[3]))

self.lblTape7.setText(str(self.tape[4]))

elif len(self.tape) >= 4:

self.lblTape3.setText(str(self.tape[0]))

self.lblTape4.setText(str(self.tape[1]))

self.lblTape5.setText(str(self.tape[2]))

self.lblTape6.setText(str(self.tape[3]))

elif len(self.tape) >= 3:

self.lblTape3.setText(str(self.tape[0]))

self.lblTape4.setText(str(self.tape[1]))

self.lblTape5.setText(str(self.tape[2]))

elif len(self.tape) >= 2:

self.lblTape3.setText(str(self.tape[0]))

self.lblTape4.setText(str(self.tape[1]))

elif len(self.tape) >= 1:

self.lblTape3.setText(str(self.tape[0]))

app.processEvents()

def fillInstructions(self):

self.txtViewInstructions.setPlainText("")

index = 0

instruction\_number = 0

while index != len(self.str\_instructions):

self.txtViewInstructions.append("INSTRUCTION: " + str(instruction\_number) + ":")

self.txtViewInstructions.append(str(self.str\_instructions[index]))

self.txtViewInstructions.append(str(self.str\_instructions[index + 1]))

self.txtViewInstructions.append(str(self.str\_instructions[index + 2]))

self.txtViewInstructions.append("---------------------------------------------------------")

index += 3

instruction\_number += 1

app.processEvents()

def deleteInstruction(self):

if self.txtDeleteInstruction.text() != "":

instruction = int(self.txtDeleteInstruction.text())

index = instruction \* 3

if index + 2 <= len(self.str\_instructions) and len(self.instructions) > 0:

print "Deleting Instruction: (" + str(instruction) + ") " + str(self.instructions[instruction])

self.instructions.pop(instruction)

self.txtViewInstructions.setPlainText("")

self.str\_instructions.pop(index)

self.str\_instructions.pop(index)

self.str\_instructions.pop(index)

self.fillInstructions()

if self.txtViewInstructions.toPlainText() == "":

self.txtViewInstructions.setPlainText("No Instructions Loaded, \n"

"Add Instructions on the Right Side of the Window")

self.txtDeleteInstruction.setText("")

def clearInstructions(self):

self.instructions = []

app.processEvents()

self.txtViewInstructions.setPlainText("No Instructions Loaded, \n"

"Add Instructions on the Right Side of the Window")

def moveLeft(self):

print "LEFT"

if self.tape\_position > 0:

self.tape\_position -= 1

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] -= 1

if self.GUI\_tape\_positions[3] == -1:

self.tape = ["#"] + self.tape

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] += 1

if self.GUI\_tape\_positions[3] > -1:

if self.GUI\_tape\_positions[0] > -1:

self.lblTape0.setText(str(self.tape[self.GUI\_tape\_positions[0]]))

else:

self.lblTape0.setText("#")

if self.GUI\_tape\_positions[1] > -1:

self.lblTape1.setText(str(self.tape[self.GUI\_tape\_positions[1]]))

else:

self.lblTape1.setText("#")

if self.GUI\_tape\_positions[2] > -1:

self.lblTape2.setText(str(self.tape[self.GUI\_tape\_positions[2]]))

else:

self.lblTape2.setText("#")

self.lblTape3.setText(str(self.tape[self.GUI\_tape\_positions[3]]))

self.lblTape4.setText(str(self.tape[self.GUI\_tape\_positions[4]]))

self.lblTape5.setText(str(self.tape[self.GUI\_tape\_positions[5]]))

self.lblTape6.setText(str(self.tape[self.GUI\_tape\_positions[6]]))

self.lblTape7.setText(str(self.tape[self.GUI\_tape\_positions[7]]))

def moveRight(self):

print "RIGHT"

self.tape\_position += 1

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] += 1

self.tape.append("#")

if self.GUI\_tape\_positions[0] > -1:

self.lblTape0.setText(str(self.tape[self.GUI\_tape\_positions[0]]))

else:

self.lblTape0.setText("#")

if self.GUI\_tape\_positions[1] > -1:

self.lblTape1.setText(str(self.tape[self.GUI\_tape\_positions[1]]))

else:

self.lblTape1.setText("#")

if self.GUI\_tape\_positions[2] > -1:

self.lblTape2.setText(str(self.tape[self.GUI\_tape\_positions[2]]))

else:

self.lblTape2.setText("#")

self.lblTape3.setText(str(self.tape[self.GUI\_tape\_positions[3]]))

self.lblTape4.setText(str(self.tape[self.GUI\_tape\_positions[4]]))

self.lblTape5.setText(str(self.tape[self.GUI\_tape\_positions[5]]))

self.lblTape6.setText(str(self.tape[self.GUI\_tape\_positions[6]]))

self.lblTape7.setText(str(self.tape[self.GUI\_tape\_positions[7]]))

def submit(self):

self.scroll\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7] # resets scroll positions

if self.txtTape.text() == "":

self.txtTape.setText("#")

inputtape = str(self.txtTape.text())

self.tape = inputtape.split(",") # splits each value by comma, allowing for multiple entries

while len(self.tape) < 8:

self.tape.append("#")

self.fillTape()

app.processEvents()

self.txtTape.setText("")

def reset\_tape(self):

self.lblTape0.setText("#")

self.lblTape1.setText("#")

self.lblTape2.setText("#")

self.lblTape3.setText("#")

self.lblTape4.setText("#")

self.lblTape5.setText("#")

self.lblTape6.setText("#")

self.lblTape7.setText("#")

self.tape\_position = 0

self.tape = []

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6,7] # positions of the tape, which will change depending on left or right

self.lblScannedState.setText("N/A")

self.lblScannedSymbol.setText("N/A")

self.lblDirection.setText("N/A")

self.lblNewSymbol.setText("N/A")

self.lblNewState.setText("N/A")

def final\_tape(self):

finaltapestr = ""

while self.tape[0] == "#":

self.tape.pop(0)

if self.tape[0] != "#":

break

while self.tape[-1] == "#":

self.tape.pop(-1)

if self.tape[-1] != "#":

break

for index in range(len(self.tape)):

finaltapestr += "[" + str(self.tape[index]) + "] "

msgTape = QtGui.QMessageBox()

msgTape.setWindowTitle("Final Tape")

msgTape.setText("<b>" + finaltapestr + "</b>")

msgTape.setStandardButtons(QtGui.QMessageBox.Ok)

msgTape.exec\_()

class ExamplesWindow(QtGui.QMainWindow, TuringMachine):

pass

class HomeWindow(QtGui.QMainWindow):

def \_\_init\_\_(self):

QtGui.QApplication.setStyle(QtGui.QStyleFactory.create("Plastique"))

super(HomeWindow, self).\_\_init\_\_()

self.setGeometry(700, 400, 350, 185)

self.setFixedSize(350, 185)

self.setWindowFlags(Qt.Window | Qt.WindowTitleHint | Qt.CustomizeWindowHint)

self.draw\_Window()

def draw\_Window(self):

self.btnStartCustomTape = QtGui.QPushButton("Start Machine With Custom Tape", self)

self.btnStartCustomTape.clicked.connect(self.showCustomTape)

self.btnStartCustomTape.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: green; "

"font-size: 17px")

self.btnStartCustomTape.resize(300, 40)

self.btnStartCustomTape.move(25, 25)

self.btnStartExampleTapes = QtGui.QPushButton("Example Machines", self)

self.btnStartExampleTapes.clicked.connect(self.showExamples)

self.btnStartExampleTapes.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: green; "

"font-size: 17px")

self.btnStartExampleTapes.resize(300, 40)

self.btnStartExampleTapes.move(25, 75)

self.btnQuit = QtGui.QPushButton("Quit", self)

self.btnQuit.clicked.connect(self.close\_application)

self.btnQuit.setStyleSheet(

"border: 1px solid #000000;"

"font: bold; "

"color: white;"

"background-color: #901414; "

"font-size: 20px")

self.btnQuit.resize(300, 40)

self.btnQuit.move(25, 125)

self.customtape\_dialog = CustomTapeWindow(self)

self.examples\_dialog = ExamplesWindow(self)

self.show()

def close\_application(self):

userinput = QtGui.QMessageBox.question(self, 'Quit', "Are you sure you wish to exit?",

QtGui.QMessageBox.Yes | QtGui.QMessageBox.No)

if userinput == QtGui.QMessageBox.Yes:

print "EXITING"

sys.exit()

else:

pass

def showCustomTape(self):

self.customtape\_dialog.show()

def showExamples(self):

self.examples\_dialog.show()

if \_\_name\_\_ == '\_\_main\_\_':

app = QtGui.QApplication(sys.argv) # argv allows you to pass arguments through the cmd when running

# defined app here

GUI = HomeWindow()

sys.exit(app.exec\_())

## Prototype 2

from Tkinter import \*

import tkMessageBox

import sys

from time import sleep

class State: # manages each state of the machine and executing the instruction associated with the state

instructions = [] # a list of instructions associated with the current state

current\_context = None # current context object used to access the Turing Machine

def \_\_init\_\_(self, context):

self.current\_context = context # sets the current context of the machine

self.instructions = [] # sets the instructions to empty

self.start\_of\_machine = True # states that the machine is in its first cycle (just started)

def executeInstruction(self):

for index in range(len(self.instructions)): # used to cycle through the instructions until one is found which matches both the current symbol and state

if self.current\_context.tape\_position == len(self.current\_context.tape):

break

if self.instructions[index].symbol == self.current\_context.tape[self.current\_context.tape\_position]:

app.lblScannedState.config(text = str(self.current\_context.state\_name)) # these lines will slowly show the instruction to help with readability when running

sleep(0.5) # 0.5s sleep

root.update() # updates the GUI

app.lblScannedSymbol.config(text = str(self.instructions[index].symbol))

sleep(0.5)

root.update()

app.lblDirection.config(text = str(self.instructions[index].direction))

sleep(0.5)

root.update()

app.lblNewSymbol.config(text = str(self.instructions[index].new\_symbol))

sleep(0.5)

root.update()

app.lblNewState.config(text = str(self.instructions[index].next\_state))

sleep(0.5)

root.update()

sleep(1)

self.current\_context.tape[self.current\_context.tape\_position] = self.instructions[index].new\_symbol # changes the symbol on the tape to the one specified from the instruction

print "New Symbol:", self.instructions[index].new\_symbol

if self.instructions[index].direction == 'R':

print "Right"

self.current\_context.moveRight() # moves the Turing Machine tape along to the right

app.scrollRight() # moves the GUI tape along to the right

elif self.instructions[index].direction == 'L':

print "Left"

self.current\_context.moveLeft() # moves the Turing Machine tape along to the right

app.scrollLeft() # moves the GUI tape along to the right

root.update() # updates the GUI

if self.instructions[index].next\_state in self.current\_context.available\_states:

self.current\_context.setState(self.instructions[index].next\_state) # sets the state to the next state specified in the instruction

self.current\_context.state\_name = self.instructions[index].next\_state # saves the name to be used in TuringMachine too

print "Current State: ", self.current\_context.state\_name

self.current\_context.instruction\_found = False # resets instruction found so the next instruction can be found

else:

self.current\_context.current\_state = None

print self.current\_context.tape

break

else:

self.current\_context.instruction\_found = False

self.current\_context.current\_state = None # if no symbol matches an instruction, the state gets set to None, therefore ending the machine

def setInstruction(self, symbol, direction, new\_symbol, next\_state):

new\_instruction = Instruction() # creates an instruction object

new\_instruction.symbol = symbol # and fills it with the attributes specified for the instruction

new\_instruction.direction = direction

new\_instruction.new\_symbol = new\_symbol

new\_instruction.next\_state = next\_state

self.instructions.append(new\_instruction)

class Instruction: # attributes for each function

symbol = ''

direction = 'None'

new\_symbol = ''

next\_state = ''

class TuringMachine: # The main Turing Machine which will handle the attributes of the machine, such as the tape and states (The machine will run from here)

current\_state = None # current state object

state\_name = "" # name of the current state

available\_states = {"HALT": None} # dict of available state objects

tape = [] # tape list

tape\_position = 0 # current position on the tape

temp\_instructions = [] # holds the instructions in a 2D list before they are set

starting\_state = None # defines the current state

instruction\_found = True # states if the instruction has been found or not

def runMachine(self): # runs the machine and executing the instructions through State

print "Current Instructions: ", self.temp\_instructions

if self.starting\_state == None: # checks if a starting state has been set

print "Please Define a Starting State or Instruction"

return # if not, the machine does not run

elif len(self.tape) == 0: # checks the length of the tape

print "Please Provide a Tape"

return # if no tape has been specified the machine will not run

for instruction in self.temp\_instructions: # cycles through the instruction 2D list and sets the instructions to the states

if instruction[0] not in self.available\_states: # checks if the state has already been defined

self.available\_states[instruction[0]] = State(self) # if it has not, an object will be created

self.available\_states[instruction[0]].setInstruction(instruction[1], instruction[2], instruction[3], instruction[4]) # then the instruction will be added to that state

self.current\_state = self.available\_states[self.starting\_state] # sets the current state to the specified starting state

self.state\_name = self.starting\_state # and sets the string name

while self.state\_name is not 'HALT': # if an instruction has been found and the machine is not in an HALT state, the machine will continue to run

if self.current\_state == None:

break

self.executeInstruction() # executes the instruction

app.finalTapePopup() # once the machine has finished, the popup will appear to show the final tape

def moveRight(self): # handles moving right on the tape

self.tape\_position += 1 # moves along once to the right of the tape

self.tape.append("#") # a # will be added to the end to stop any out of range errors by the GUI

def moveLeft(self): # handles moving left on the tape

self.tape\_position -= 1 # moves along once to the left of the tape

self.tape = ["#"] + self.tape # the machine will add a # to the start of the tape to stop any out of rang errors by the GUI

def saveInstruction(self, state, symbol, direction, new\_symbol, next\_state): # saves an instruction to the 2D list

current\_instruction = [state, symbol, direction, new\_symbol, next\_state] # creates a temp list to save to the 2D list of instructions

self.temp\_instructions.append(current\_instruction) # adds this list to the list of instructions

def setTape(self, tape): # saves the specified tape

self.tape = (tape.split(",")) # using comma seperation, the tape will be split from the symbols inbetween each comma

while len(self.tape) < 8: # if the machine does nto meet the required length

self.tape.append("#") # the machine will fill the tape with blank symbols (all blank symbols will be removed at the end of the machine)

def setState(self, state): # sets the current state

self.current\_state = self.available\_states[state] # sets the current state to the state passed in

def executeInstruction(self): # executes the State instruction

self.current\_state.executeInstruction() # executes the State executeInstruction() depending on the specified state

class CustomTapeWindow:

def \_\_init\_\_(self, window):

self.tape\_font = ("times", 20)

self.instruction\_font = ("times", 20)

self.addInstruction\_font = ("times", 15)

self.currentInstruction\_font = ("times", 10)

self.arrow\_font = ("calibri", 10, "bold")

self.run\_font = ("bold")

self.submitTape\_font = ("calibri", 10, "bold")

self.title\_font = ("calibri", 12, "italic")

self.machine = TuringMachine()

self.str\_instructions = []

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]

self.drawWindow()

def drawWindow(self):

self.lblTapeTitle = Label(

root,

text = "Insert Tape: (Comma Seperated)",

font = self.title\_font)

self.lblTapeTitle.pack()

self.lblTapeTitle.place(x = 302, y = 158)

self.lblTape0 = Label(

root,

text= "#",

bg = "LightSteelBlue1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape0.pack()

self.lblTape0.place(x = 302, y = 237)

self.lblTape1 = Label(

root,

text="#",

bg = "LightSteelBlue1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape1.pack()

self.lblTape1.place(x = 382, y = 237)

self.lblTape2 = Label(

root,

text="#",

bg = "LightSteelBlue1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape2.pack()

self.lblTape2.place(x = 462, y = 237)

self.lblTape3 = Label(

root,

text="#",

bg = "DarkOliveGreen3",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape3.pack()

self.lblTape3.place(x = 542, y = 237)

self.lblTape4 = Label(

root,

text="#",

bg = "LightSteelBlue1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape4.pack()

self.lblTape4.place(x = 622, y = 237)

self.lblTape5 = Label(

root,

text="#",

bg = "LightSteelBlue1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape5.pack()

self.lblTape5.place(x = 702, y = 237)

self.lblTape6 = Label(

root,

text="#",

bg = "LightSteelBlue1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape6.pack()

self.lblTape6.place(x = 782, y = 237)

self.lblTape7 = Label(

root,

text="#",

bg = "LightSteelBlue1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape7.pack()

self.lblTape7.place(x = 862, y = 237)

self.lblAddInstructionTitle = Label(

root,

text = "Add a Instruction:",

font = self.title\_font)

self.lblAddInstructionTitle.pack()

self.lblAddInstructionTitle.place(x = 10, y = -1)

self.lblInstructionsTitle = Label(

root,

text = "Running Instruction:",

font = self.title\_font)

self.lblInstructionsTitle.pack()

self.lblInstructionsTitle.place(x = 958, y = -1)

self.lblInstructionsBorder = Label(

root,

text= "Scanned State: \n\n"

"Scanned Symbol: \n\n"

"Direction: \n\n"

"New Symbol: \n\n"

"New State:",

bg = "white",

fg = "black",

font = self.instruction\_font,

height = 9,

width = 18,

borderwidth = 2,

relief = "solid",

justify = LEFT,

anchor = "w")

self.lblInstructionsBorder.pack()

self.lblInstructionsBorder.place(x = 958, y = 20)

self.lblScannedState = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.currentInstruction\_font,

height = 2,

width = 7)

self.lblScannedState.pack()

self.lblScannedState.place(x = 1148, y = 28)

self.lblScannedSymbol = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.currentInstruction\_font,

height = 2,

width = 7)

self.lblScannedSymbol.pack()

self.lblScannedSymbol.place(x = 1148, y = 85)

self.lblDirection = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.currentInstruction\_font,

height = 2,

width = 7)

self.lblDirection.pack()

self.lblDirection.place(x = 1148, y = 146)

self.lblNewSymbol = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.currentInstruction\_font,

height = 2,

width = 7)

self.lblNewSymbol.pack()

self.lblNewSymbol.place(x = 1148, y = 207)

self.lblNewState = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.currentInstruction\_font,

height = 2,

width = 7)

self.lblNewState.pack()

self.lblNewState.place(x = 1148, y = 264)

self.lblAddInstructionsBorder = Label(

root,

text= "Scanned State: \n\n"

"Scanned Symbol: \n\n"

"Direction: \n\n"

"New Symbol: \n\n"

"New State:",

bg = "white",

fg = "black",

font = self.instruction\_font,

height = 9,

width = 18,

borderwidth = 2,

relief = "solid",

justify = LEFT,

anchor = "w")

self.lblAddInstructionsBorder.pack()

self.lblAddInstructionsBorder.place(x = 0, y = 20)

self.txtScannedState = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtScannedState.pack()

self.txtScannedState.place(x = 195, y = 28)

self.txtScannedSymbol = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtScannedSymbol.pack()

self.txtScannedSymbol.place(x = 195, y = 85)

self.txtDirection = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtDirection.pack()

self.txtDirection.place(x = 195, y = 146)

self.txtNewSymbol = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtNewSymbol.pack()

self.txtNewSymbol.place(x = 195, y = 207)

self.txtNewState = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtNewState.pack()

self.txtNewState.place(x = 195, y = 270)

self.btnSaveInstruction = Button(

root,

text = "Save Instruction",

command = self.saveInstruction,

bg = "green4",

fg = "white",

font = self.arrow\_font,

height = 1,

width = 20,

borderwidth = 1,

relief = "solid")

self.btnSaveInstruction.pack()

self.btnSaveInstruction.place(x = 65, y = 310)

self.btnRight = Button(

root,

text = "--------->",

command = self.scrollRight,

bg = "DarkGoldenrod3",

fg = "white",

font = self.arrow\_font,

height = 1,

width = 10,

borderwidth = 1,

relief = "solid")

self.btnRight.pack()

self.btnRight.place(x = 854, y = 207)

self.btnLeft = Button(

root,

text = "<---------",

command = self.scrollLeft,

bg = "DarkGoldenrod3",

fg = "white",

font = self.arrow\_font,

height = 1,

width = 10,

borderwidth = 1,

relief = "solid")

self.btnLeft.pack()

self.btnLeft.place(x = 302, y = 207)

self.txtStartingStateTitle = Label(

root,

text = "Starting State:",

font = self.title\_font)

self.txtStartingStateTitle.pack()

self.txtStartingStateTitle.place(x = 550, y = 145)

self.txtStartingState = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtStartingState.pack()

self.txtStartingState.place(x = 650, y = 145)

self.btnStartingState = Button(

root,

text = "Submit State",

command = self.setStartingState,

bg = "green4",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 12,

borderwidth = 1,

relief = "solid")

self.btnStartingState.pack()

self.btnStartingState.place(x = 732, y = 146)

self.btnRun = Button(

root,

text = "Run",

command = self.runMachine,

bg = "green4",

fg = "white",

height = 1,

width = 64,

borderwidth = 1,

relief = "solid")

self.btnRun.pack()

self.btnRun.place(x = 389, y = 207)

self.txtTape = Text(

root,

bg = "white",

fg = "black",

font = self.submitTape\_font,

height = 1,

width = 74,

borderwidth = 1,

relief = "solid")

self.txtTape.pack()

self.txtTape.place(x = 302, y = 180)

self.btnTape = Button(

root,

text = "Submit Tape",

command = self.submitTape,

bg = "green4",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 13,

borderwidth = 1,

relief = "solid")

self.btnTape.pack()

self.btnTape.place(x = 834, y = 177)

self.lblAddInstructionTitle = Label(

root,

text = "Current Instruction(s):",

font = self.title\_font)

self.lblAddInstructionTitle.pack()

self.lblAddInstructionTitle.place(x = 302, y = -1)

self.instructionsFrame = Frame(

root,

borderwidth = 1,

relief = "solid")

self.instructionsFrame.pack()

self.instructionsFrame.place(x = 302, y = 20)

self.scrlInstructions = Scrollbar(self.instructionsFrame)

self.scrlInstructions.pack(side = "right", fill = "y")

self.lstShowInstructions = Listbox(

self.instructionsFrame,

bg = "white",

yscrollcommand = self.scrlInstructions.set,

fg = "black",

height = 7,

width = 83)

self.lstShowInstructions.pack(side = "left", fill = "y")

self.scrlInstructions.config(command = self.lstShowInstructions.yview)

self.lstShowInstructions.insert(0, "No Instructions Loaded, \n"

"Add Instructions on the Right Side of the Window")

self.lblDeleteTitle = Label(

root,

text = "Delete:",

font = self.title\_font)

self.lblDeleteTitle.pack()

self.lblDeleteTitle.place(x = 836, y = 0)

self.lblDeleteTitle2 = Label(

root,

text = "(Enter Number)",

font = self.title\_font)

self.lblDeleteTitle2.pack()

self.lblDeleteTitle2.place(x = 836, y = 20)

self.txtDeleteInstruction = Entry(

root,

bg = "white",

fg = "black",

font = self.submitTape\_font,

width = 13,

borderwidth = 1,

relief = "solid")

self.txtDeleteInstruction.pack()

self.txtDeleteInstruction.place(x = 836, y = 50)

self.btnDeleteInstruction = Button(

root,

text = "Delete \n Instruction",

command = self.deleteInstruction,

bg = "red3",

fg = "white",

font = self.submitTape\_font,

height = 2,

width = 13,

borderwidth = 1,

relief = "solid")

self.btnDeleteInstruction.pack()

self.btnDeleteInstruction.place(x = 834, y = 75)

self.btnDeleteAllInstructions = Button(

root,

text = "Clear All \n Instructions",

command = self.clearInstructions,

bg = "red3",

fg = "white",

font = self.submitTape\_font,

height = 2,

width = 13,

borderwidth = 1,

relief = "solid")

self.btnDeleteAllInstructions.pack()

self.btnDeleteAllInstructions.place(x = 834, y = 120)

self.txtTape.focus()

def runMachine(self):

if self.machine.starting\_state == None and len(self.machine.temp\_instructions) > 0:

self.machine.starting\_state = self.machine.temp\_instructions[0][0]

print "Starting State: ", self.machine.starting\_state

print "Starting Tape: ", self.machine.tape

self.machine.runMachine()

def scrollLeft(self):

if len(self.machine.tape) > 0:

print "LEFT"

if self.machine.tape\_position > 0:

self.machine.tape\_position -= 1

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] -= 1

if self.GUI\_tape\_positions[3] == -1:

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] += 1

if self.GUI\_tape\_positions[3] > -1:

if self.GUI\_tape\_positions[0] > -1:

self.lblTape0.config(text = str(self.machine.tape[self.GUI\_tape\_positions[0]]))

else:

self.lblTape0.config(text = "#")

if self.GUI\_tape\_positions[1] > -1:

self.lblTape1.config(text = str(self.machine.tape[self.GUI\_tape\_positions[1]]))

else:

self.lblTape1.config(text = "#")

if self.GUI\_tape\_positions[2] > -1:

self.lblTape2.config(text = str(self.machine.tape[self.GUI\_tape\_positions[2]]))

else:

self.lblTape2.config(text = "#")

if self.GUI\_tape\_positions[3] < len(self.machine.tape):

self.lblTape3.config(text = str(self.machine.tape[self.GUI\_tape\_positions[3]]))

else:

self.lblTape3.config(text = "#")

if self.GUI\_tape\_positions[4] < len(self.machine.tape):

self.lblTape4.config(text = str(self.machine.tape[self.GUI\_tape\_positions[4]]))

else:

self.lblTape4.config(text = "#")

if self.GUI\_tape\_positions[5] < len(self.machine.tape):

self.lblTape5.config(text = str(self.machine.tape[self.GUI\_tape\_positions[5]]))

else:

self.lblTape5.config(text = "#")

if self.GUI\_tape\_positions[6] < len(self.machine.tape):

self.lblTape6.config(text = str(self.machine.tape[self.GUI\_tape\_positions[6]]))

else:

self.lblTape6.config(text = "#")

if self.GUI\_tape\_positions[7] < len(self.machine.tape):

self.lblTape7.config(text = str(self.machine.tape[self.GUI\_tape\_positions[7]]))

else:

self.lblTape7.config(text = "#")

def scrollRight(self):

if len(self.machine.tape) > 0:

print "RIGHT"

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] += 1

if self.GUI\_tape\_positions[0] < len(self.machine.tape):

self.lblTape0.config(text = str(self.machine.tape[self.GUI\_tape\_positions[0]]))

else:

self.lblTape0.config(text = "#")

if self.GUI\_tape\_positions[1] < len(self.machine.tape):

self.lblTape1.config(text = str(self.machine.tape[self.GUI\_tape\_positions[1]]))

else:

self.lblTape1.config(text = "#")

if self.GUI\_tape\_positions[2] < len(self.machine.tape):

self.lblTape2.config(text = str(self.machine.tape[self.GUI\_tape\_positions[2]]))

else:

self.lblTape2.config(text = "#")

if self.GUI\_tape\_positions[3] < len(self.machine.tape):

self.lblTape3.config(text = str(self.machine.tape[self.GUI\_tape\_positions[3]]))

else:

self.lblTape3.config(text = "#")

if self.GUI\_tape\_positions[4] < len(self.machine.tape):

self.lblTape4.config(text = str(self.machine.tape[self.GUI\_tape\_positions[4]]))

else:

self.lblTape4.config(text = "#")

if self.GUI\_tape\_positions[5] < len(self.machine.tape):

self.lblTape5.config(text = str(self.machine.tape[self.GUI\_tape\_positions[5]]))

else:

self.lblTape5.config(text = "#")

if self.GUI\_tape\_positions[6] < len(self.machine.tape):

self.lblTape6.config(text = str(self.machine.tape[self.GUI\_tape\_positions[6]]))

else:

self.lblTape6.config(text = "#")

if self.GUI\_tape\_positions[7] < len(self.machine.tape):

self.lblTape7.config(text = str(self.machine.tape[self.GUI\_tape\_positions[7]]))

else:

self.lblTape7.config(text = "#")

if self.GUI\_tape\_positions[3] > len(self.machine.tape):

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] -= 1

def submitTape(self):

if self.txtTape.get("1.0", END) == "":

self.txtTape.insert(END, "#")

input\_tape = str(self.txtTape.get("1.0", END))

input\_tape = input\_tape[:-1]

self.machine.setTape(input\_tape)

self.fillTape()

self.txtTape.delete('1.0', END)

print "Current Tape: ", self.machine.tape

def resetMachine(self):

self.lblTape0.config(text = "#")

self.lblTape1.config(text = "#")

self.lblTape2.config(text = "#")

self.lblTape3.config(text = "#")

self.lblTape4.config(text = "#")

self.lblTape5.config(text = "#")

self.lblTape6.config(text = "#")

self.lblTape7.config(text = "#")

self.machine.tape\_position = 0

self.machine.tape = []

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7] # positions of the tape, which will change depending on left or right

self.machine.available\_states = {}

self.machine.temp\_instructions = []

self.machine.state\_name = ""

self.machine.starting\_state = None

self.machine.current\_state = None

self.lblScannedState.config(text = "N/A")

self.lblScannedSymbol.config(text = "N/A")

self.lblDirection.config(text = "N/A")

self.lblNewSymbol.config(text = "N/A")

self.lblNewState.config(text = "N/A")

self.clearInstructions()

root.update()

def fillTape(self):

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]

self.lblTape0.config(text = "#")

self.lblTape1.config(text = "#")

self.lblTape2.config(text = "#")

self.lblTape3.config(text = "#")

self.lblTape4.config(text = "#")

self.lblTape5.config(text = "#")

self.lblTape6.config(text = "#")

self.lblTape7.config(text = "#")

if len(self.machine.tape) >= 5:

self.lblTape3.config(text = str(self.machine.tape[0]))

self.lblTape4.config(text = str(self.machine.tape[1]))

self.lblTape5.config(text = str(self.machine.tape[2]))

self.lblTape6.config(text = str(self.machine.tape[3]))

self.lblTape7.config(text = str(self.machine.tape[4]))

elif len(self.machine.tape) >= 4:

self.lblTape3.config(text = str(self.machine.tape[0]))

self.lblTape4.config(text = str(self.machine.tape[1]))

self.lblTape5.config(text = str(self.machine.tape[2]))

self.lblTape6.config(text = str(self.machine.tape[3]))

elif len(self.machine.tape) >= 3:

self.lblTape3.config(text = str(self.machine.tape[0]))

self.lblTape4.config(text = str(self.machine.tape[1]))

self.lblTape5.config(text = str(self.machine.tape[2]))

elif len(self.machine.tape) >= 2:

self.lblTape3.config(text = str(self.machine.tape[0]))

self.lblTape4.config(text = str(self.machine.tape[1]))

elif len(self.machine.tape) >= 1:

self.lblTape3.config(text = str(self.machine.tape[0]))

def finaliseTape(self):

finaltapestr = ""

while self.machine.tape == "#":

self.machine.tape.pop(0)

if self.machine.tape[0] != "#":

break

if len(self.machine.tape) > 1:

while self.machine.tape[-1] == "#":

self.machine.tape.pop(-1)

if self.machine.tape[-1] != "#":

break

for index in range (len(self.machine.tape)):

finaltapestr += "[" + str(self.machine.tape[index]) + "] "

return finaltapestr

def saveInstruction(self):

temp\_list = []

self.machine.saveInstruction(

str(self.txtScannedState.get()),

str(self.txtScannedSymbol.get()),

str(self.txtDirection.get()),

str(self.txtNewSymbol.get()),

str(self.txtNewState.get()))

self.str\_instructions.append("Scanned State: " + str(self.txtScannedState.get()))

self.str\_instructions.append("Scanned Synbol: " + str(self.txtScannedSymbol.get()))

self.str\_instructions.append("Direction: " + str(self.txtDirection.get()))

self.str\_instructions.append("New Symbol: " + str(self.txtNewSymbol.get()))

self.str\_instructions.append("New State: " + str(self.txtNewState.get()))

self.txtScannedState.delete("0", END)

self.txtScannedSymbol.delete("0", END)

self.txtDirection.delete("0", END)

self.txtNewSymbol.delete("0", END)

self.txtNewState.delete("0", END)

self.fillInstructions()

self.txtScannedState.focus()

def fillInstructions(self):

if self.lstShowInstructions.size() != 0:

self.lstShowInstructions.delete(0, END)

index = 0

instruction\_number = 0

while index != len(self.str\_instructions):

self.lstShowInstructions.insert(END, "\n INSTRUCTION: " + str(instruction\_number) + ": \n")

self.lstShowInstructions.insert(END, str(self.str\_instructions[index]))

self.lstShowInstructions.insert(END, str(self.str\_instructions[index + 1]))

self.lstShowInstructions.insert(END, str(self.str\_instructions[index + 2]))

self.lstShowInstructions.insert(END, str(self.str\_instructions[index + 3]))

self.lstShowInstructions.insert(END, str(self.str\_instructions[index + 4]))

self.lstShowInstructions.insert(END, "---------------------------------------------------------")

index += 5

instruction\_number += 1

def deleteInstruction(self):

instruction = self.txtDeleteInstruction.get()

if len(instruction) != 0:

if isinstance(instruction, int) == False:

print "Please Enter an Integer"

self.txtDeleteInstruction.delete("0", END)

return

index = instruction \* 5

if index + 4 <= len(self.str\_instructions) and len(self.machine.temp\_instructions) > 0:

print "Deleteing Instruction: (" + str(instruction) + ") " + str (self.machine.temp\_instructions[instruction])

self.machine.temp\_instructions.pop(instruction)

self.lstShowInstructions.delete(0, END)

print "Current Instructions:", self.machine.temp\_instructions

self.str\_instructions.pop(index)

self.str\_instructions.pop(index)

self.str\_instructions.pop(index)

self.str\_instructions.pop(index)

self.str\_instructions.pop(index)

if len(self.str\_instructions) >= 5:

self.lstShowInstructions.delete(0, END)

strindex = 0

instruction\_number = 0

while strindex != len(self.str\_instructions):

self.lstShowInstructions.insert(END, "\n INSTRUCTION: " + str(instruction\_number) + ": \n")

self.lstShowInstructions.insert(END, str(self.str\_instructions[strindex]))

self.lstShowInstructions.insert(END, str(self.str\_instructions[strindex + 1]))

self.lstShowInstructions.insert(END, str(self.str\_instructions[strindex + 2]))

self.lstShowInstructions.insert(END, str(self.str\_instructions[strindex + 3]))

self.lstShowInstructions.insert(END, str(self.str\_instructions[strindex + 4]))

self.lstShowInstructions.insert(END, "---------------------------------------------------------")

strindex += 5

instruction\_number += 1

if self.lstShowInstructions.size() == 0:

self.lstShowInstructions.insert(

0, "No Instructions Loaded, \n Add Instructions on the Left Side of the Window")

self.txtDeleteInstruction.delete("0", END)

def clearInstructions(self):

self.lstShowInstructions.delete(0, END)

self.machine.available\_states = {}

self.machine.temp\_instructions = []

root.update()

self.lstShowInstructions.insert(0, "No Instructions Loaded, \n"

"Add Instructions on the Right Side of the Window")

print self.machine.temp\_instructions

def setStartingState(self):

if str(self.txtStartingState.get()) != "":

self.machine.starting\_state = str(self.txtStartingState.get())

print "starting State set to: ", self.machine.starting\_state

def finalTapePopup(self):

finaltape\_str = self.finaliseTape()

popup\_finaltape = Toplevel()

popup\_finaltape.title("Final Tape")

popup\_finaltape.geometry("300x150+750+250")

if self.machine.instruction\_found == False:

lblInstructionFound = Label(popup\_finaltape, text = "Machine Halt \n Because No Instruction Present \n for the Current Context of the Turing Machine")

lblInstructionFound.pack()

elif self.machine.instruction\_found == True:

lblInstructionFound = Label(popup\_finaltape, text = "Machine Halt \n Because of Transition to HALT State")

lblInstructionFound.pack()

lblFinalTape = Label(popup\_finaltape, text = finaltape\_str)

lblFinalTape.pack()

btnDismiss = Button(popup\_finaltape, text="Dismiss Window", command = popup\_finaltape.destroy)

btnDismiss.pack()

btnDismiss.focus()

self.resetMachine()

if \_\_name\_\_ == "\_\_main\_\_":

root = Tk()

app = CustomTapeWindow(root)

root.geometry("1235x345+250+250")

root.mainloop()

## Prototype 3

from State import State, StateContext

from time import sleep

DIRECTION = 0

NEWSYMBOL = 1

NEXTSTATE = 2

class Transition:

def is0(self):

print "Error"

return False

def is1(self):

print "Error"

return False

def isHALT(self):

print "Error"

return False

class TuringStateA(State, Transition):

instructions = {

"0": ["R", "1", "B"],

"HALT": ["R", "HALT", "HALT"]

}

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def is0(self):

print "Current State: A"

sleep(0.5)

print "Current Symbol: 0"

sleep(0.5)

self.current\_context.tape[self.current\_context.tape\_position] = self.instructions["0"][NEWSYMBOL]

self.current\_context.setState(self.instructions["0"][NEXTSTATE]) # changing to next machine state

print "Moving Direction: ", self.instructions["0"][DIRECTION]

sleep(0.5)

if self.instructions["0"][DIRECTION] == "R":

self.current\_context.tape\_position += 1 # moving right

elif self.instructions["0"][DIRECTION] == "L":

self.current\_context.tape\_position -= 1 # moving left

print "New Symbol: ", self.instructions["0"][NEWSYMBOL]

sleep(0.5)

print "Next State: ", self.instructions["0"][NEXTSTATE]

sleep(0.5)

print "Tape: ", self.current\_context.tape

sleep(0.5)

return True

def isHALT(self):

print "Current State: ", self.current\_context.state

sleep(0.5)

print "Current Symbol: HALT"

sleep(0.5)

self.current\_context.tape[self.current\_context.tape\_position] = self.instructions["HALT"][NEWSYMBOL]

self.current\_context.setState(self.instructions["HALT"][NEXTSTATE]) # changing to next machine state

print "Moving Direction: ", self.instructions["HALT"][DIRECTION]

sleep(0.5)

if self.instructions["HALT"][DIRECTION] == "R":

self.current\_context.tape\_position += 1 # moving right

elif self.instructions["HALT"][DIRECTION] == "L":

self.current\_context.tape\_position -= 1 # moving left

print "New Symbol: ", self.instructions["HALT"][NEWSYMBOL]

sleep(0.5)

print "Next State: ", self.instructions["HALT"][NEXTSTATE]

sleep(0.5)

print "Tape: ", self.current\_context.tape

sleep(0.5)

return True

class TuringStateB(State, Transition):

instructions = {

"1": ["R", "0", "A"],

"HALT": ["R", "HALT", "HALT"]

}

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def is1(self):

print "Current State: ", self.current\_context.state

sleep(0.5)

print "Current Symbol: 1"

sleep(0.5)

self.current\_context.tape[self.current\_context.tape\_position] = self.instructions["1"][NEWSYMBOL]

self.current\_context.setState(self.instructions["1"][NEXTSTATE]) # changing to next machine state

print "Moving Direction: ", self.instructions["1"][DIRECTION]

sleep(0.5)

if self.instructions["1"][DIRECTION] == "R":

self.current\_context.tape\_position += 1 # moving right

elif self.instructions["1"][DIRECTION] == "L":

self.current\_context.tape\_position -= 1 # moving left

print "New Symbol: ", self.instructions["1"][NEWSYMBOL]

sleep(0.5)

print "Next State: ", self.current\_context.state

sleep(0.5)

print "Tape: ", self.current\_context.tape

sleep(0.5)

def isHALT(self):

print "Current State: ", self.current\_context.state

sleep(0.5)

print "Current Symbol: HALT"

sleep(0.5)

self.current\_context.tape[self.current\_context.tape\_position] = self.instructions["HALT"][NEWSYMBOL]

self.current\_context.setState(self.instructions["HALT"][NEXTSTATE]) # changing to next machine state

print "Moving Direction: ", self.instructions["HALT"][DIRECTION]

sleep(0.5)

if self.instructions["HALT"][DIRECTION] == "R":

self.current\_context.tape\_position += 1 # moving right

elif self.instructions["HALT"][DIRECTION] == "L":

self.current\_context.tape\_position -= 1 # moving left

print "New Symbol: ", self.instructions["HALT"][NEWSYMBOL]

sleep(0.5)

print "Next State: ", self.instructions["HALT"][NEXTSTATE]

sleep(0.5)

print "Tape: ", self.current\_context.tape

sleep(0.5)

return True

class TuringStateHALT(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

class TuringMachine(StateContext, Transition):

def \_\_init\_\_(self, tape):

self.tape = tape

self.tape\_position = 0

self.starting\_state = "A"

self.availableStates["A"] = TuringStateA(self)

self.availableStates["B"] = TuringStateB(self)

self.availableStates["HALT"] = TuringStateHALT(self)

self.setState(self.starting\_state)

def is0(self):

return self.current\_state.is0()

def is1(self):

return self.current\_state.is1()

def isHALT(self):

return self.current\_state.isHALT()

if \_\_name\_\_ == "\_\_main\_\_":

TM = TuringMachine(["0", "1", "0", "1", "0", "1", "0", "1", "HALT"])

while TM.state != "HALT":

if TM.tape[TM.tape\_position] == "0":

TM.is0()

elif TM.tape[TM.tape\_position] == "1":

TM.is1()

elif TM.tape[TM.tape\_position] == "HALT":

TM.isHALT()

print TM.tape

## Prototype 4

*GUI code is not included in the appendix, but will be included in the actual python files*

### TM\_addition\_unary

class Transition:

def foundBlank(self):

print "Error, [#] or Blank Symbol instruction not found in current state"

return False

def found0(self):

print "Error, [0] instruction not found in current state"

def found1(self):

print "Error, [1] instruction not found in current state"

return False

def foundPlus(self):

print "Error, [+] instruction not found in current state"

return False

def foundEquals(self):

print "Error [=] instruction not found in current state"

return False

# one of the states of the machine which will handle changing the tape and the machines state depending on the method called

class TuringStateA(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

# if the machine finds a 0 on the tape it will

def found0(self):

# display the current state on the GUI

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

# pause for readability

sleep(0.2)

# display the current symbol read on the tape

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

# pause for readability

root.update()

sleep(0.2)

# display the direction of the head

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

# pause for readability

sleep(0.2)

# using current context changes the symbol on the turing machine tape

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

# using current context changes to the next state of the machine

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

# proceeds to move the tape to the right on the list

self.current\_context.moveRight()

# and moves the tape right on the GUI

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundPlus(self):

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

print "(+) Symbol on Tape"

app.lblScannedSymbol.config(text = "+")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "+"

print "(+) New Symbol on Tape"

app.lblNewSymbol.config(text = "+")

root.update()

sleep(0.2)

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEquals(self):

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

print "(=) Symbol on Tape"

app.lblScannedSymbol.config(text = "=")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "="

print "(=) New Symbol on Tape"

app.lblNewSymbol.config(text = "=")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateB(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found0(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundPlus(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(+) Symbol on Tape"

app.lblScannedSymbol.config(text = "+")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "+"

print "(+) New Symbol on Tape"

app.lblNewSymbol.config(text = "+")

root.update()

sleep(0.2)

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEquals(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(=) Symbol on Tape"

app.lblScannedSymbol.config(text = "=")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "="

print "(=) New Symbol on Tape"

app.lblNewSymbol.config(text = "E")

root.update()

sleep(0.2)

self.current\_context.setState("C")

print "(C) New State"

app.lblNewState.config(text = "C")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateC(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found1(self):

print "(C) Current State"

app.lblScannedState.config(text = "C")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("C")

print "(C) New State"

app.lblNewState.config(text = "C")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundBlank(self):

print "(C) Current State"

app.lblScannedState.config(text = "C")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateD(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found0(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundPlus(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(+) Symbol on Tape"

app.lblScannedSymbol.config(text = "+")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "+"

print "(+) New Symbol on Tape"

app.lblNewSymbol.config(text = "+")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEquals(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(=) Symbol on Tape"

app.lblScannedSymbol.config(text = "=")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "="

print "(=) New Symbol on Tape"

app.lblNewSymbol.config(text = "=")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundBlank(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "#"

print "(#) New Symbol on Tape"

app.lblNewSymbol.config(text = "#")

root.update()

sleep(0.2)

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateE(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found0(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundPlus(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(+) Symbol on Tape"

app.lblScannedSymbol.config(text = "+")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "+"

print "(+) New Symbol on Tape"

app.lblNewSymbol.config(text = "+")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundBlank(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "#"

print "(#) New Symbol on Tape"

app.lblNewSymbol.config(text = "#")

root.update()

sleep(0.2)

self.current\_context.setState("HALT")

print "(HALT) New State"

app.lblNewState.config(text = "HALT")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

self.current\_context.end\_of\_machine = True

sleep(1)

return True

class TuringStateHALT(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

class TuringMachine(StateContext, Transition):

# when the machine is initialised the state, tape and position will be set

def \_\_init\_\_(self, starting\_state, tape, starting\_position):

self.tape = tape

self.tape\_position = starting\_position

self.starting\_state = starting\_state

# a bool to determine if the machine has completed or not

self.end\_of\_machine = False

# in the available states dict the machine creates an instance of each of the TuringStates under keys used to transition between states

self.availableStates["A"] = TuringStateA(self)

self.availableStates["B"] = TuringStateB(self)

self.availableStates["C"] = TuringStateC(self)

self.availableStates["D"] = TuringStateD(self)

self.availableStates["E"] = TuringStateE(self)

self.availableStates["HALT"] = TuringStateHALT(self)

# sets the starting state to the one specified

self.setState(self.starting\_state)

def foundBlank(self):

return self.current\_state.foundBlank()

def found0(self):

return self.current\_state.found0()

def found1(self):

return self.current\_state.found1()

def foundPlus(self):

return self.current\_state.foundPlus()

def foundEquals(self):

return self.current\_state.foundEquals()

# a dictionary used to handle what symbols are read on the tape and run the corrisponding method depending on what was read

# e.g. if (1) was read on the tape, the machine will run self.found1 form the dictionary

def runMachine(self):

symbols = {

"#": self.foundBlank,

"1": self.found1,

"0": self.found0,

"+": self.foundPlus,

"=": self.foundEquals

}

print "\nStarting Tape: ", self.tape

print "\n - - - - - - - - - - - - - - \n"

# for loop which will run through the machine and call the correct methods until the machine reaches a Halting state

while self.end\_of\_machine is False:

if symbols[self.tape[self.tape\_position]]() is False:

break

# resets the GUI instructions when new instructions are read

app.lblScannedState.config(text = "N/A")

app.lblScannedSymbol.config(text = "N/A")

app.lblDirection.config(text = "N/A")

app.lblNewSymbol.config(text = "N/A")

app.lblNewState.config(text = "N/A")

# when finished the machines tape is finalised

app.finalTapePopup()

# handles moving right on the tape

def moveRight(self):

if self.tape\_position >= len(self.tape) -2:

# a (#) will be added to the end to stop any out of range errors by the GUI

self.tape.append("#")

self.tape\_position += 1 # moves along once to the right of the tape

# handles moving left on the tape

def moveLeft(self):

# the machine will add a # to the start of the tape to stop any out of rang errors by the GUI

self.tape = ["#"] + self.tape

if \_\_name\_\_ == "\_\_main\_\_":

root = Tk()

app = TuringMachineGUI(root)

root.geometry("1375x500+250+250")

root.title("Unary Addition")

root.mainloop()

### TM\_flip\_1\_0

class Transition:

def found0(self):

print "Error, [0] instruction not found in current state"

return False

def found1(self):

print "Error, [1] instruction not found in current state"

return False

def foundEND(self):

print "Error, [#] or Blank Symbol instruction not found in current state"

return False

# one of the states of the machine which will handle changing the tape and the machines state depending on the method called

class TuringStateA(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

# if the machine finds a 0 on the tape it will

def found0(self):

# display the current state on the GUI

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

# display the current symbol read on the tape

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

# display the direction of the head

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

# using current context changes the symbol on the turing machine tape

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

# using current context changes to the next state of the machine

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

# proceeds to move the tape to the right on the list

self.current\_context.moveRight()

# and moves the tape right on the GUI

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateB(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found1(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringMachine(StateContext, Transition):

# when the machine is initialised the state, tape and position will be set

def \_\_init\_\_(self, starting\_state, tape, starting\_position):

self.tape = tape

self.tape\_position = starting\_position

self.starting\_state = starting\_state

# a bool to determine if the machine has completed or not

self.end\_of\_machine = False

# in the available states dict the machine creates an instance of each of the TuringStates under keys used to transition between states

self.availableStates["A"] = TuringStateA(self)

self.availableStates["B"] = TuringStateB(self)

# sets the starting state to the one specified

self.setState(self.starting\_state)

def found0(self):

return self.current\_state.found0()

def found1(self):

return self.current\_state.found1()

def foundEND(self):

return self.current\_state.foundEND()

# a dictionary used to handle what symbols are read on the tape and run the corrisponding method depending on what was read

# e.g. if (1) was read on the tape, the machine will run self.found1 form the dictionary

def runMachine(self):

symbols = {

"0": self.found0,

"1": self.found1,

"END": self.foundEND

}

print "\nStarting Tape: ", self.tape

print "\n - - - - - - - - - - - - - - \n"

# for loop which will run through the machine and call the correct methods until the machine reaches a Halting state

while self.end\_of\_machine is False:

if symbols[self.tape[self.tape\_position]]() is False:

break

# resets the GUI instructions when new instructions are read

app.lblScannedState.config(text = "N/A")

app.lblScannedSymbol.config(text = "N/A")

app.lblDirection.config(text = "N/A")

app.lblNewSymbol.config(text = "N/A")

app.lblNewState.config(text = "N/A")

# when finished the machines tape is finalised

app.finalTapePopup()

# handles moving right on the tape

def moveRight(self):

if self.tape\_position >= len(self.tape) -2:

# a (#) will be added to the end to stop any out of range errors by the GUI

self.tape.append("#")

# moves along once to the right of the tape

self.tape\_position += 1

# handles moving left on the tape

def moveLeft(self):

# the machine will add a # to the start of the tape to stop any out of rang errors by the GUI

self.tape = ["#"] + self.tape

if \_\_name\_\_ == "\_\_main\_\_":

root = Tk()

app = TuringMachineGUI(root)

root.geometry("1375x500+250+250")

root.title("Flip 1's and 0's")

root.mainloop()

### TM\_Hello\_World

class Transition:

def foundBlank(self):

print "Error, [#] or Blank Symbol instruction not found in current state"

return False

def found1(self):

print "Error, [1] instruction not found in current state"

return False

def found2(self):

print "Error, [2] instruction not found in current state"

return False

def found3(self):

print "Error, [3] instruction not found in current state"

return False

def found4(self):

print "Error, [4] instruction not found in current state"

return False

def found5(self):

print "Error, [5] instruction not found in current state"

return False

def found6(self):

print "Error, [6] instruction not found in current state"

return False

def found7(self):

print "Error, [7] instruction not found in current state"

return False

def found8(self):

print "Error, [8] instruction not found in current state"

return False

def found9(self):

print "Error, [9] instruction not found in current state"

return False

def found10(self):

print "Error, [10] instruction not found in current state"

return False

def found11(self):

print "Error, [11] instruction not found in current state"

return False

def found12(self):

print "Error, [12] instruction not found in current state"

return False

def foundEND(self):

print "Error, END instruction not found in current state"

return False

# one of the states of the machine which will handle changing the tape and the machines state depending on the method called

class TuringStateA(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

# if the machine finds a 0 on the tape it will

def found1(self):

# display the current state on the GUI

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

# display the current symbol read on the tape

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

# display the direction of the head

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

# using current context changes the symbol on the turing machine tape

self.current\_context.tape[self.current\_context.tape\_position] = "H"

print "(H) New Symbol on Tape"

app.lblNewSymbol.config(text = "H")

root.update()

sleep(0.2)

# using current context changes to the next state of the machine

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

# proceeds to move the tape to the right on the list

self.current\_context.moveRight()

# and moves the tape right on the GUI

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateB(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found2(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(2) Symbol on Tape"

app.lblScannedSymbol.config(text = "2")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "E"

print "(E) New Symbol on Tape"

app.lblNewSymbol.config(text = "E")

root.update()

sleep(0.2)

self.current\_context.setState("C")

print "(C) New State"

app.lblNewState.config(text = "C")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateC(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found3(self):

print "(C) Current State"

app.lblScannedState.config(text = "C")

root.update()

sleep(0.2)

print "(3) Symbol on Tape"

app.lblScannedSymbol.config(text = "3")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "L"

print "(L) New Symbol on Tape"

app.lblNewSymbol.config(text = "L")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateD(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found4(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(4) Symbol on Tape"

app.lblScannedSymbol.config(text = "4")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "L"

print "(L) New Symbol on Tape"

app.lblNewSymbol.config(text = "L")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateE(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found5(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(5) Symbol on Tape"

app.lblScannedSymbol.config(text = "5")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "O"

print "(O) New Symbol on Tape"

app.lblNewSymbol.config(text = "H")

root.update()

sleep(0.2)

self.current\_context.setState("F")

print "(F) New State"

app.lblNewState.config(text = "F")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateF(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found6(self):

print "(F) Current State"

app.lblScannedState.config(text = "F")

root.update()

sleep(0.2)

print "(6) Symbol on Tape"

app.lblScannedSymbol.config(text = "6")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "\_"

print "(\_) New Symbol on Tape"

app.lblNewSymbol.config(text = "\_")

root.update()

sleep(0.2)

self.current\_context.setState("G")

print "(G) New State"

app.lblNewState.config(text = "G")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateG(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found7(self):

print "(G) Current State"

app.lblScannedState.config(text = "G")

root.update()

sleep(0.2)

print "(6) Symbol on Tape"

app.lblScannedSymbol.config(text = "6")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "W"

print "(W) New Symbol on Tape"

app.lblNewSymbol.config(text = "W")

root.update()

sleep(0.2)

self.current\_context.setState("H")

print "(H) New State"

app.lblNewState.config(text = "H")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateH(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found8(self):

print "(H) Current State"

app.lblScannedState.config(text = "H")

root.update()

sleep(0.2)

print "(8) Symbol on Tape"

app.lblScannedSymbol.config(text = "8")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "O"

print "(O) New Symbol on Tape"

app.lblNewSymbol.config(text = "O")

root.update()

sleep(0.2)

self.current\_context.setState("I")

print "(I) New State"

app.lblNewState.config(text = "I")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateI(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found9(self):

print "(I) Current State"

app.lblScannedState.config(text = "I")

root.update()

sleep(0.2)

print "(9) Symbol on Tape"

app.lblScannedSymbol.config(text = "9")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "R"

print "(R) New Symbol on Tape"

app.lblNewSymbol.config(text = "R")

root.update()

sleep(0.2)

self.current\_context.setState("J")

print "(J) New State"

app.lblNewState.config(text = "J")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateJ(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found10(self):

print "(J) Current State"

app.lblScannedState.config(text = "J")

root.update()

sleep(0.2)

print "(10) Symbol on Tape"

app.lblScannedSymbol.config(text = "10")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "L"

print "(L) New Symbol on Tape"

app.lblNewSymbol.config(text = "L")

root.update()

sleep(0.2)

self.current\_context.setState("K")

print "(K) New State"

app.lblNewState.config(text = "K")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateK(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found11(self):

print "(K) Current State"

app.lblScannedState.config(text = "K")

root.update()

sleep(0.2)

print "(11) Symbol on Tape"

app.lblScannedSymbol.config(text = "11")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "D"

print "(D) New Symbol on Tape"

app.lblNewSymbol.config(text = "D")

root.update()

sleep(0.2)

self.current\_context.setState("L")

print "(L) New State"

app.lblNewState.config(text = "L")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringStateL(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found12(self):

print "(L) Current State"

app.lblScannedState.config(text = "L")

root.update()

sleep(0.2)

print "(12) Symbol on Tape"

app.lblScannedSymbol.config(text = "12")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "!"

print "(!) New Symbol on Tape"

app.lblNewSymbol.config(text = "!")

root.update()

sleep(0.2)

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEND(self):

self.current\_context.tape[self.current\_context.tape\_position] = "#"

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

return True

# display final tape and end machine

class TuringMachine(StateContext, Transition):

# when the machine is initialised the state, tape and position will be set

def \_\_init\_\_(self, starting\_state, tape, starting\_position):

self.tape = tape

self.tape\_position = starting\_position

self.starting\_state = starting\_state

# a bool to determine if the machine has completed or not

self.end\_of\_machine = False

# in the available states dict the machine creates an instance of each of the TuringStates under keys used to transition between states

self.availableStates["A"] = TuringStateA(self)

self.availableStates["B"] = TuringStateB(self)

self.availableStates["C"] = TuringStateC(self)

self.availableStates["D"] = TuringStateD(self)

self.availableStates["E"] = TuringStateE(self)

self.availableStates["F"] = TuringStateF(self)

self.availableStates["G"] = TuringStateG(self)

self.availableStates["H"] = TuringStateH(self)

self.availableStates["I"] = TuringStateI(self)

self.availableStates["J"] = TuringStateJ(self)

self.availableStates["K"] = TuringStateK(self)

self.availableStates["L"] = TuringStateL(self)

# sets the starting state to the one specified

self.setState(self.starting\_state)

def foundBlank(self):

return self.current\_state.foundBlank()

def found1(self):

return self.current\_state.found1()

def found2(self):

return self.current\_state.found2()

def found3(self):

return self.current\_state.found3()

def found4(self):

return self.current\_state.found4()

def found5(self):

return self.current\_state.found5()

def found6(self):

return self.current\_state.found6()

def found7(self):

return self.current\_state.found7()

def found8(self):

return self.current\_state.found8()

def found9(self):

return self.current\_state.found9()

def found10(self):

return self.current\_state.found10()

def found11(self):

return self.current\_state.found11()

def found12(self):

return self.current\_state.found12()

def foundEND(self):

return self.current\_state.foundEND()

# a dictionary used to handle what symbols are read on the tape and run the corrisponding method depending on what was read

# e.g. if (1) was read on the tape, the machine will run self.found1 form the dictionary

def runMachine(self):

symbols = {

"#": self.foundBlank,

"1": self.found1,

"2": self.found2,

"3": self.found3,

"4": self.found4,

"5": self.found5,

"6": self.found6,

"7": self.found7,

"8": self.found8,

"9": self.found9,

"10": self.found10,

"11": self.found11,

"12": self.found12,

"END": self.foundEND

}

print "\nStarting Tape: ", self.tape

print "\n - - - - - - - - - - - - - - \n"

# for loop which will run through the machine and call the correct methods until the machine reaches a Halting state

while self.end\_of\_machine is False:

if symbols[self.tape[self.tape\_position]]() is False:

break

# resets the GUI instructions when new instructions are read

app.lblScannedState.config(text = "N/A")

app.lblScannedSymbol.config(text = "N/A")

app.lblDirection.config(text = "N/A")

app.lblNewSymbol.config(text = "N/A")

app.lblNewState.config(text = "N/A")

# when finished the machines tape is finalised

app.finalTapePopup()

# handles moving right on the tape

def moveRight(self):

if self.tape\_position >= len(self.tape) -2:

# a # will be added to the end to stop any out of range errors by the GUI

self.tape.append("#")

# moves along once to the right of the tape

self.tape\_position += 1

# handles moving left on the tape

def moveLeft(self):

# the machine will add a # to the start of the tape to stop any out of rang errors by the GUI

self.tape = ["#"] + self.tape

if \_\_name\_\_ == "\_\_main\_\_":

root = Tk()

app = TuringMachineGUI(root)

root.geometry("1375x500+250+250")

root.title("Hello World")

root.mainloop()

### TM\_multiply\_unary

class Transition:

def foundBlank(self):

print "Error, [#] or Blank Symbol instruction not found in current state"

return False

def found1(self):

print "Error, [1] instruction not found in current state"

return False

def found0(self):

print "Error, [0] instruction not found in current state"

return False

def foundx(self):

print "Error, [x] instruction not found in current state"

return False

def foundEquals(self):

print "Error, [=] instruction not found in current state"

return False

# one of the states of the machine which will handle changing the tape and the machines state depending on the method called

class TuringStateA(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

# if the machine finds a 0 on the tape it will

def foundx(self):

# display the current state on the GUI

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

# display the current symbol read on the tape

print "(x) Symbol on Tape"

app.lblScannedSymbol.config(text = "x")

root.update()

sleep(0.2)

# display the direction of the head

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

# using current context changes the symbol on the turing machine tape

self.current\_context.tape[self.current\_context.tape\_position] = "x"

print "(x) New Symbol on Tape"

app.lblNewSymbol.config(text = "x")

root.update()

sleep(0.2)

# using current context changes to the next state of the machine

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

# proceeds to move the tape to the left on the list

self.current\_context.moveLeft()

# and moves the tape right on the GUI

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundBlank(self):

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "#"

print "(#) New Symbol on Tape"

app.lblNewSymbol.config(text = "#")

root.update()

sleep(0.2)

self.current\_context.setState("G")

print "(G) New State"

app.lblNewState.config(text = "G")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found0(self):

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateB(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def foundx(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(x) Symbol on Tape"

app.lblScannedSymbol.config(text = "x")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "x"

print "(x) New Symbol on Tape"

app.lblNewSymbol.config(text = "x")

root.update()

sleep(0.2)

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEquals(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(=) Symbol on Tape"

app.lblScannedSymbol.config(text = "=")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "="

print "(=) New Symbol on Tape"

app.lblNewSymbol.config(text = "=")

root.update()

sleep(0.2)

self.current\_context.setState("F")

print "(F) New State"

app.lblNewState.config(text = "F")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found0(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("C")

print "(C) New State"

app.lblNewState.config(text = "C")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateC(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def foundEquals(self):

print "(C) Current State"

app.lblScannedState.config(text = "C")

root.update()

sleep(0.2)

print "(=) Symbol on Tape"

app.lblScannedSymbol.config(text = "=")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "="

print "(=) New Symbol on Tape"

app.lblNewSymbol.config(text = "=")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found0(self):

print "(C) Current State"

app.lblScannedState.config(text = "C")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("C")

print "(C) New State"

app.lblNewState.config(text = "C")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(C) Current State"

app.lblScannedState.config(text = "C")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("C")

print "(C) New State"

app.lblNewState.config(text = "C")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateD(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found1(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundBlank(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateE(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def foundx(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(x) Symbol on Tape"

app.lblScannedSymbol.config(text = "x")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "x"

print "(x) New Symbol on Tape"

app.lblNewSymbol.config(text = "x")

root.update()

sleep(0.2)

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundEquals(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(=) Symbol on Tape"

app.lblScannedSymbol.config(text = "=")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "="

print "(=) New Symbol on Tape"

app.lblNewSymbol.config(text = "=")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found0(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(E) Current State"

app.lblScannedState.config(text = "E")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("E")

print "(E) New State"

app.lblNewState.config(text = "E")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateF(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def foundx(self):

print "(F) Current State"

app.lblScannedState.config(text = "F")

root.update()

sleep(0.2)

print "(x) Symbol on Tape"

app.lblScannedSymbol.config(text = "x")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "x"

print "(x) New Symbol on Tape"

app.lblNewSymbol.config(text = "x")

root.update()

sleep(0.2)

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found0(self):

print "(F) Current State"

app.lblScannedState.config(text = "F")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("F")

print "(F) New State"

app.lblNewState.config(text = "F")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(F) Current State"

app.lblScannedState.config(text = "F")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("F")

print "(F) New State"

app.lblNewState.config(text = "F")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateG(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def foundx(self):

print "(G) Current State"

app.lblScannedState.config(text = "G")

root.update()

sleep(0.2)

print "(x) Symbol on Tape"

app.lblScannedSymbol.config(text = "x")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "x"

print "(x) New Symbol on Tape"

app.lblNewSymbol.config(text = "x")

root.update()

sleep(0.2)

self.current\_context.setState("HALT")

print "(HALT) New State"

app.lblNewState.config(text = "HALT")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

self.current\_context.end\_of\_machine = True

sleep(1)

return True

def foundBlank(self):

print "(G) Current State"

app.lblScannedState.config(text = "G")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "#"

print "(#) New Symbol on Tape"

app.lblNewSymbol.config(text = "#")

root.update()

sleep(0.2)

self.current\_context.setState("G")

print "(G) New State"

app.lblNewState.config(text = "G")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found0(self):

print "(G) Current State"

app.lblScannedState.config(text = "G")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("G")

print "(G) New State"

app.lblNewState.config(text = "G")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def found1(self):

print "(G) Current State"

app.lblScannedState.config(text = "G")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("G")

print "(G) New State"

app.lblNewState.config(text = "G")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateHALT(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

class TuringMachine(StateContext, Transition):

# when the machine is initialised the state, tape and position will be set

def \_\_init\_\_(self, starting\_state, tape, starting\_position):

self.tape = tape

self.tape\_position = starting\_position

self.starting\_state = starting\_state

# a bool to determine if the machine has completed or not

self.end\_of\_machine = False

# in the available states dict the machine creates an instance of each of the TuringStates under keys used to transition between states

self.availableStates["A"] = TuringStateA(self)

self.availableStates["B"] = TuringStateB(self)

self.availableStates["C"] = TuringStateC(self)

self.availableStates["D"] = TuringStateD(self)

self.availableStates["E"] = TuringStateE(self)

self.availableStates["F"] = TuringStateF(self)

self.availableStates["G"] = TuringStateG(self)

self.availableStates["HALT"] = TuringStateHALT(self)

# sets the starting state to the one specified

self.setState(self.starting\_state)

def foundBlank(self):

return self.current\_state.foundBlank()

def found0(self):

return self.current\_state.found0()

def found1(self):

return self.current\_state.found1()

def foundx(self):

return self.current\_state.foundx()

def foundEquals(self):

return self.current\_state.foundEquals()

# a dictionary used to handle what symbols are read on the tape and run the corrisponding method depending on what was read

# e.g. if (1) was read on the tape, the machine will run self.found1 form the dictionary

def runMachine(self):

symbols = {

"#": self.foundBlank,

"1": self.found1,

"0": self.found0,

"x": self.foundx,

"=": self.foundEquals

}

print "\nStarting Tape: ", self.tape

print "\n - - - - - - - - - - - - - - \n"

# for loop which will run through the machine and call the correct methods until the machine reaches a Halting state

while self.end\_of\_machine is False:

if symbols[self.tape[self.tape\_position]]() is False:

break

# resets the GUI instructions when new instructions are read

app.lblScannedState.config(text = "N/A")

app.lblScannedSymbol.config(text = "N/A")

app.lblDirection.config(text = "N/A")

app.lblNewSymbol.config(text = "N/A")

app.lblNewState.config(text = "N/A")

# when finished the machines tape is finalised

app.finalTapePopup()

# handles moving right on the tape

def moveRight(self):

if self.tape\_position >= len(self.tape) -2:

# a # will be added to the end to stop any out of range errors by the GUI

self.tape.append("#")

# moves along once to the right of the tape

self.tape\_position += 1

# handles moving left on the tape

def moveLeft(self):

# the machine will add a # to the start of the tape to stop any out of rang errors by the GUI

self.tape = ["#"] + self.tape

if \_\_name\_\_ == "\_\_main\_\_":

root = Tk()

app = TuringMachineGUI(root)

root.geometry("1375x500+250+250")

root.title("Unary Multiplication")

root.mainloop()

### TM\_odd\_number\_0s

class Transition:

def foundBlank(self):

print "Error, [#] or Blank Symbol instruction not found in current state"

return False

def found1(self):

print "Error, [1] instruction not found in current state"

return False

def found0(self):

print "Error, [0] instruction not found in current state"

return False

# one of the states of the machine which will handle changing the tape and the machines state depending on the method called

class TuringStateA(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

# if the machine finds a 0 on the tape it will

def found0(self):

# display the current state on the GUI

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

# display the current symbol read on the tape

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

# display the direction of the head

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

# using current context changes the symbol on the turing machine tape

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

# using current context changes to the next state of the machine

self.current\_context.setState("B")

print "(B) New State"

app.lblNewState.config(text = "B")

root.update()

# proceeds to move the tape to the right on the list

self.current\_context.moveRight()

# and moves the tape right on the GUI

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundBlank(self):

print "(A) Current State"

app.lblScannedState.config(text = "A")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "#"

print "(#) New Symbol on Tape"

app.lblNewSymbol.config(text = "#")

root.update()

sleep(0.2)

self.current\_context.setState("C")

print "(C) New State"

app.lblNewState.config(text = "C")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateB(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found0(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(0) Symbol on Tape"

app.lblScannedSymbol.config(text = "0")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "Right")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "1"

print "(1) New Symbol on Tape"

app.lblNewSymbol.config(text = "1")

root.update()

sleep(0.2)

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundBlank(self):

print "(B) Current State"

app.lblScannedState.config(text = "B")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "R")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "#"

print "(#) New Symbol on Tape"

app.lblNewSymbol.config(text = "#")

root.update()

sleep(0.2)

self.current\_context.setState("HALT")

print "(HALT) New State"

app.lblNewState.config(text = "HALT")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "End of Machine"

print "\n - - - - - - - - - - - - - - \n"

app.finaliseTape()

self.current\_context.end\_of\_machine = True

sleep(1)

return True

class TuringStateC(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found1(self):

print "(C) Current State"

app.lblScannedState.config(text = "C")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("D")

print "(D) New State"

app.lblNewState.config(text = "D")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

def foundBlank(self):

print "(C) Current State"

app.lblScannedState.config(text = "C")

root.update()

sleep(0.2)

print "(#) Symbol on Tape"

app.lblScannedSymbol.config(text = "#")

root.update()

sleep(0.2)

print "(R) Direction"

app.lblDirection.config(text = "R")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "#"

print "(#) New Symbol on Tape"

app.lblNewSymbol.config(text = "#")

root.update()

sleep(0.2)

self.current\_context.setState("A")

print "(A) New State"

app.lblNewState.config(text = "A")

root.update()

self.current\_context.moveRight()

app.scrollRight(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateD(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

def found1(self):

print "(D) Current State"

app.lblScannedState.config(text = "D")

root.update()

sleep(0.2)

print "(1) Symbol on Tape"

app.lblScannedSymbol.config(text = "1")

root.update()

sleep(0.2)

print "(L) Direction"

app.lblDirection.config(text = "Left")

root.update()

sleep(0.2)

self.current\_context.tape[self.current\_context.tape\_position] = "0"

print "(0) New Symbol on Tape"

app.lblNewSymbol.config(text = "0")

root.update()

sleep(0.2)

self.current\_context.setState("C")

print "(C) New State"

app.lblNewState.config(text = "C")

root.update()

self.current\_context.moveLeft()

app.scrollLeft(self.current\_context.tape)

root.update()

print "Tape: ", self.current\_context.tape

print "\n - - - - - - - - - - - - - - \n"

sleep(1)

return True

class TuringStateHALT(State, Transition):

def \_\_init\_\_(self, context):

State.\_\_init\_\_(self, context)

class TuringMachine(StateContext, Transition):

# when the machine is initialised the state, tape and position will be set

def \_\_init\_\_(self, starting\_state, tape, starting\_position):

self.tape = tape

self.tape\_position = starting\_position

self.starting\_state = starting\_state

# a bool to determine if the machine has completed or not

self.end\_of\_machine = False

# in the available states dict the machine creates an instance of each of the TuringStates under keys used to transition between states

self.availableStates["A"] = TuringStateA(self)

self.availableStates["B"] = TuringStateB(self)

self.availableStates["C"] = TuringStateC(self)

self.availableStates["D"] = TuringStateD(self)

self.availableStates["HALT"] = TuringStateHALT(self)

# sets the starting state to the one specified

self.setState(self.starting\_state)

def foundBlank(self):

return self.current\_state.foundBlank()

def found0(self):

return self.current\_state.found0()

def found1(self):

return self.current\_state.found1()

# a dictionary used to handle what symbols are read on the tape and run the corrisponding method depending on what was read

# e.g. if (1) was read on the tape, the machine will run self.found1 form the dictionary

def runMachine(self):

symbols = {

"#": self.foundBlank,

"1": self.found1,

"0": self.found0

}

print "\nStarting Tape: ", self.tape

print "\n - - - - - - - - - - - - - - \n"

# for loop which will run through the machine and call the correct methods until the machine reaches a Halting state

while self.end\_of\_machine is False:

if symbols[self.tape[self.tape\_position]]() is False:

break

# resets the GUI instructions when new instructions are read

app.lblScannedState.config(text = "N/A")

app.lblScannedSymbol.config(text = "N/A")

app.lblDirection.config(text = "N/A")

app.lblNewSymbol.config(text = "N/A")

app.lblNewState.config(text = "N/A")

# when finished the machines tape is finalised

app.finalTapePopup()

# handles moving right on the tape

def moveRight(self):

# moves along once to the right of the tape

self.tape\_position += 1

# a # will be added to the end to stop any out of range errors by the GUI

self.tape.append("#")

# handles moving left on the tape

def moveLeft(self):

# the machine will add a # to the start of the tape to stop any out of rang errors by the GUI

self.tape = ["#"] + self.tape

if \_\_name\_\_ == "\_\_main\_\_":

root = Tk()

app = TuringMachineGUI(root)

root.geometry("1375x500+250+250")

root.title("Odd Number of 0's")

root.mainloop()

### Dynamic Turing Machine

from State import State, StateContext

from time import sleep

from collections import defaultdict

from Tkinter import \*

import tkMessageBox

import sys

DIRECTION = 0

NEW\_SYMBOL = 1

NEXT\_STATE = 2

class TuringMachineGUI():

'''Handles all of the GUI aspects of the program, such as updating the interface

and resetting the machine'''

def \_\_init\_\_(self, window):

# defines all of the fonts used in the GUI

self.tape\_font = ("times", 20)

self.position\_font = ("times", 15, "italic")

self.instruction\_font = ("times", 15)

self.addInstruction\_font = ("times", 10)

self.currentInstruction\_font = ("times", 10)

self.arrow\_font = ("calibri", 10, "bold")

self.run\_font = ("bold")

self.submitTape\_font = ("calibri", 10, "bold")

self.title\_font = ("times", 15)

self.title2\_font = ("times", 11)

# creates an instance of the turing machine to allow for adding the tape, state, position, etc. To the machine

self.TM = TuringMachine()

# attributes to handle the speed the machine will run (sleep)

self.speedLabel = 5

self.speeds = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

self.speedindex = 5

# the GUI tape positions to handle where each value on the tape will show on the GUI

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]

# draws the GUI aspects on the window

self.drawWindow()

def drawWindow(self):

# shows the tape positions at the top of the window

self.lblPositions = Label(

root,

text= "-3 -2 -1 0 1 2 3 4",

fg = "gray26",

font = self.position\_font)

self.lblPositions.pack()

self.lblPositions.place(x = 80, y = 5)

# the following represents each square of the tape

self.lblTape0 = Label(

root,

text= "#",

bg = "DarkSeaGreen1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape0.pack()

self.lblTape0.place(x = 62, y = 30)

self.lblTape1 = Label(

root,

text="#",

bg = "DarkSeaGreen1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape1.pack()

self.lblTape1.place(x = 142, y = 30)

self.lblTape2 = Label(

root,

text="#",

bg = "DarkSeaGreen1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape2.pack()

self.lblTape2.place(x = 222, y = 30)

self.lblTape3 = Label(

root,

text="#",

bg = "DarkOliveGreen3",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape3.pack()

self.lblTape3.place(x = 302, y = 30)

self.lblTape4 = Label(

root,

text="#",

bg = "DarkSeaGreen1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape4.pack()

self.lblTape4.place(x = 382, y = 30)

self.lblTape5 = Label(

root,

text="#",

bg = "DarkSeaGreen1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape5.pack()

self.lblTape5.place(x = 462, y = 30)

self.lblTape6 = Label(

root,

text="#",

bg = "DarkSeaGreen1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape6.pack()

self.lblTape6.place(x = 542, y = 30)

self.lblTape7 = Label(

root,

text="#",

bg = "DarkSeaGreen1",

fg = "black",

font = self.tape\_font,

height = 2,

width = 4,

borderwidth = 2,

relief = "solid")

self.lblTape7.pack()

self.lblTape7.place(x = 622, y = 30)

# allows for moving right

self.btnRight = Button(

root,

text = ">",

bg = "DarkGoldenrod3",

fg = "black",

font = self.arrow\_font,

height = 4,

width = 2,

borderwidth = 1,

relief = "solid",

command = self.scrollRight

)

self.btnRight.pack()

self.btnRight.place(x = 700, y = 30)

# allows for moving left

self.btnLeft = Button(

root,

text = "<",

bg = "DarkGoldenrod3",

fg = "black",

font = self.arrow\_font,

height = 4,

width = 2,

borderwidth = 1,

relief = "solid",

command = self.scrollLeft

)

self.btnLeft.pack()

self.btnLeft.place(x = 30, y = 30)

# title for the add instruction section

self.lblInstructionsTitle = Label(

root,

text = "Add Instruction:",

font = self.title\_font)

self.lblInstructionsTitle.pack()

self.lblInstructionsTitle.place(x = 468, y = 115)

# border for adding instructions

self.lblAddInstructionsBorder = Label(

root,

text= "Scanned State: \n"

"Scanned Symbol: \n"

"Direction: \n"

"New Symbol: \n"

"New State:",

bg = "white",

fg = "black",

font = self.instruction\_font,

height = 5,

width = 23,

borderwidth = 2,

relief = "solid",

justify = LEFT,

anchor = "w")

self.lblAddInstructionsBorder.pack()

self.lblAddInstructionsBorder.place(x = 468, y = 140)

# the following allow the user to add instructions

self.txtScannedState = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtScannedState.pack()

self.txtScannedState.place(x = 675, y = 146)

self.txtScannedSymbol = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtScannedSymbol.pack()

self.txtScannedSymbol.place(x = 675, y = 168)

self.txtDirection = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtDirection.pack()

self.txtDirection.place(x = 675, y = 188)

self.txtNewSymbol = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtNewSymbol.pack()

self.txtNewSymbol.place(x = 675, y = 208)

self.txtNewState = Entry(

root,

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7,

borderwidth = 1,

relief = "solid")

self.txtNewState.pack()

self.txtNewState.place(x = 675, y = 230)

# saves the instructions to the machine

self.btnSaveInstruction = Button(

root,

text = "Save Instruction",

bg = "green4",

fg = "white",

font = self.arrow\_font,

height = 1,

width = 36,

borderwidth = 1,

relief = "solid",

command = self.setInstruction)

self.btnSaveInstruction.pack()

self.btnSaveInstruction.place(x = 468, y = 265)

# title for the running instructions

self.lblInstructionsTitle = Label(

root,

text = "Running Instruction:",

font = self.title\_font)

self.lblInstructionsTitle.pack()

self.lblInstructionsTitle.place(x = 468, y = 293)

# border for the running instructions

self.lblInstructionsBorder = Label(

root,

text= "Scanned State: \n"

"Scanned Symbol: \n"

"Direction: \n"

"New Symbol: \n"

"New State:",

bg = "white",

fg = "black",

font = self.instruction\_font,

height = 5,

width = 23,

borderwidth = 2,

relief = "solid",

justify = LEFT,

anchor = "w")

self.lblInstructionsBorder.pack()

self.lblInstructionsBorder.place(x = 468, y = 318)

# the following shows the currently running instructions

self.lblScannedState = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7)

self.lblScannedState.pack()

self.lblScannedState.place(x = 665, y = 324)

self.lblScannedSymbol = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7)

self.lblScannedSymbol.pack()

self.lblScannedSymbol.place(x = 665, y = 346)

self.lblDirection = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7)

self.lblDirection.pack()

self.lblDirection.place(x = 665, y = 366)

self.lblNewSymbol = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7)

self.lblNewSymbol.pack()

self.lblNewSymbol.place(x = 665, y = 386)

self.lblNewState = Label(

root,

text = "N/A",

bg = "white",

fg = "black",

font = self.addInstruction\_font,

width = 7)

self.lblNewState.pack()

self.lblNewState.place(x = 665, y = 408)

# title for showing the listbox of instructions

self.lblAllInstructionsTitle = Label(

root,

text = "All Instructions:",

font = self.title\_font)

self.lblAllInstructionsTitle.pack()

self.lblAllInstructionsTitle.place(x = 262, y = 115)

# the following handles the list box for isntructions

self.instructionsFrame = Frame(

root,

borderwidth = 2,

relief = "solid")

self.instructionsFrame.pack()

self.instructionsFrame.place(x = 262, y = 140)

# with a scroll bar

self.scrlInstructions = Scrollbar(self.instructionsFrame)

self.scrlInstructions.pack(side = "right", fill = "y")

self.lstShowInstructions = Listbox(

self.instructionsFrame,

bg = "white",

yscrollcommand = self.scrlInstructions.set,

fg = "black",

height = 15,

width = 27)

self.lstShowInstructions.pack(side = "left", fill = "y")

self.scrlInstructions.config(command = self.lstShowInstructions.yview)

# the following handles adding a tape to the machine

self.lblInsertTapeTitle = Label(

root,

text = "Insert Tape (CSV):",

font = self.title\_font)

self.lblInsertTapeTitle.pack()

self.lblInsertTapeTitle.place(x = 20, y = 115)

self.txtTape = Text(

root,

bg = "white",

fg = "black",

font = self.submitTape\_font,

height = 1,

width = 31,

borderwidth = 1,

relief = "solid")

self.txtTape.pack()

self.txtTape.place(x = 20, y = 140)

self.btnTape = Button(

root,

text = "Submit Tape",

bg = "green4",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 31,

borderwidth = 1,

relief = "solid",

command = self.submitTape)

self.btnTape.pack()

self.btnTape.place(x = 19, y = 165)

# the following handles adding a starting state to the machine

self.lblStartingTitle = Label(

root,

text = "Insert Starting State:",

font = self.title\_font)

self.lblStartingTitle.pack()

self.lblStartingTitle.place(x = 20, y = 190)

self.txtStartState = Text(

root,

bg = "white",

fg = "black",

font = self.submitTape\_font,

height = 1,

width = 31,

borderwidth = 1,

relief = "solid")

self.txtStartState.pack()

self.txtStartState.place(x = 20, y = 215)

self.btnStartState = Button(

root,

text = "Submit State",

bg = "green4",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 31,

borderwidth = 1,

relief = "solid",

command = self.setStartingState)

self.btnStartState.pack()

self.btnStartState.place(x = 19, y = 240)

# runs the machine

self.btnRun = Button(

root,

text = "Run Machine",

bg = "green4",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 31,

borderwidth = 1,

relief = "solid",

command = self.runMachine)

self.btnRun.pack()

self.btnRun.place(x = 19, y = 275)

# handles changing the speed of the machine

self.lblCurrentSpeed = Label(

root,

text = "Speed: 5",

font = self.title\_font)

self.lblCurrentSpeed.pack()

self.lblCurrentSpeed.place(x = 20, y = 302)

self.btnIncreaseSpeed = Button(

root,

text = "-",

bg = "red3",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 5,

borderwidth = 1,

relief = "solid",

command = self.increaseSpeed)

self.btnIncreaseSpeed.pack()

self.btnIncreaseSpeed.place(x = 150, y = 305)

self.btnDecreaseSpeed = Button(

root,

text = "+",

bg = "green4",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 5,

borderwidth = 1,

relief = "solid",

command = self.decreaseSpeed)

self.btnDecreaseSpeed.pack()

self.btnDecreaseSpeed.place(x = 200, y = 305)

# shows help for the machine

self.btnHelp = Button(

root,

text = "Help",

bg = "DarkGoldenRod3",

fg = "black",

font = self.submitTape\_font,

height = 1,

width = 31,

borderwidth = 1,

relief = "solid",

command = self.helpPopup)

self.btnHelp.pack()

self.btnHelp.place(x = 19, y = 333)

# handles displaying the starting state and position of the machine

self.lblStartingState = Label(

root,

text = "Starting State: N/A",

font = self.title\_font)

self.lblStartingState.pack()

self.lblStartingState.place(x = 262, y = 390)

self.lblTapePosition = Label(

root,

text = "Tape Position: 0",

font = self.title\_font)

self.lblTapePosition.pack()

self.lblTapePosition.place(x = 262, y = 415)

# handles changing the tape position

self.lblTapePositionTitle = Label(

root,

text = "Tape Position (0 >):",

font = self.title\_font)

self.lblTapePositionTitle.pack()

self.lblTapePositionTitle.place(x = 20, y = 360)

self.txtTapePosition = Entry(

root,

bg = "white",

fg = "black",

font = self.submitTape\_font,

width = 31,

borderwidth = 1,

relief = "solid")

self.txtTapePosition.pack()

self.txtTapePosition.place(x = 20, y = 385)

self.btnTapePosition = Button(

root,

text = "Submit Position",

bg = "green4",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 31,

borderwidth = 1,

relief = "solid",

command = self.setTapePosition)

self.btnTapePosition.pack()

self.btnTapePosition.place(x = 19, y = 411)

# loads data from file

self.btnLoadDataFromFile = Button(

root,

text = "Reset and Load Data From File",

bg = "DarkGoldenRod3",

fg = "black",

font = self.submitTape\_font,

height = 1,

width = 31,

borderwidth = 1,

relief = "solid",

command = self.loadFromFile)

self.btnLoadDataFromFile.pack()

self.btnLoadDataFromFile.place(x = 19, y = 446)

# deletes all instructions in the machine

self.btnDeleteAllInstructions = Button(

root,

text = "Clear All Instructions",

bg = "red3",

fg = "white",

font = self.submitTape\_font,

height = 1,

width = 26,

borderwidth = 1,

relief = "solid",

command = self.clearInstructions)

self.btnDeleteAllInstructions.pack()

self.btnDeleteAllInstructions.place(x = 260, y = 446)

self.btnExit = Button(

root,

text = "Exit Application",

bg = "red3",

fg = "white",

font = self.submitTape\_font,

command = root.destroy,

height = 1,

width = 36,

borderwidth = 1,

relief = "solid")

self.btnExit.pack()

self.btnExit.place(x = 468, y = 446)

# used to scroll left on the GUI tape

def scrollLeft(self):

if len(self.TM.tape) > 0:

# moves the GUI tape positions to the left, checks for original tape as this will determine if the machine is scrolling while running or before

if str(self.TM.tape) == str(self.TM.original\_tape):

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] -= 1

if self.GUI\_tape\_positions[3] == -1:

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] += 1

# this will handle what values are presented on the GUI tape

if self.GUI\_tape\_positions[3] > -1:

if self.GUI\_tape\_positions[0] > -1:

self.lblTape0.config(text = str(self.TM.tape[self.GUI\_tape\_positions[0]]))

else:

self.lblTape0.config(text = "#")

if self.GUI\_tape\_positions[1] > -1:

self.lblTape1.config(text = str(self.TM.tape[self.GUI\_tape\_positions[1]]))

else:

self.lblTape1.config(text = "#")

if self.GUI\_tape\_positions[2] > -1:

self.lblTape2.config(text = str(self.TM.tape[self.GUI\_tape\_positions[2]]))

else:

self.lblTape2.config(text = "#")

if self.GUI\_tape\_positions[3] < len(self.TM.tape):

self.lblTape3.config(text = str(self.TM.tape[self.GUI\_tape\_positions[3]]))

else:

self.lblTape3.config(text = "#")

if self.GUI\_tape\_positions[4] < len(self.TM.tape):

self.lblTape4.config(text = str(self.TM.tape[self.GUI\_tape\_positions[4]]))

else:

self.lblTape4.config(text = "#")

if self.GUI\_tape\_positions[5] < len(self.TM.tape):

self.lblTape5.config(text = str(self.TM.tape[self.GUI\_tape\_positions[5]]))

else:

self.lblTape5.config(text = "#")

if self.GUI\_tape\_positions[6] < len(self.TM.tape):

self.lblTape6.config(text = str(self.TM.tape[self.GUI\_tape\_positions[6]]))

else:

self.lblTape6.config(text = "#")

if self.GUI\_tape\_positions[7] < len(self.TM.tape):

self.lblTape7.config(text = str(self.TM.tape[self.GUI\_tape\_positions[7]]))

else:

self.lblTape7.config(text = "#")

def scrollRight(self):

if len(self.TM.tape) > 0:

# moves the GUI tape positions to the right

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] += 1

# this will handle what values are presented on the GUI tape

if self.GUI\_tape\_positions[0] < len(self.TM.tape) and self.GUI\_tape\_positions[0] >= 0:

self.lblTape0.config(text = str(self.TM.tape[self.GUI\_tape\_positions[0]]))

else:

self.lblTape0.config(text = "#")

if self.GUI\_tape\_positions[1] < len(self.TM.tape) and self.GUI\_tape\_positions[1] >= 0:

self.lblTape1.config(text = str(self.TM.tape[self.GUI\_tape\_positions[1]]))

else:

self.lblTape1.config(text = "#")

if self.GUI\_tape\_positions[2] < len(self.TM.tape) and self.GUI\_tape\_positions[2] >= 0:

self.lblTape2.config(text = str(self.TM.tape[self.GUI\_tape\_positions[2]]))

else:

self.lblTape2.config(text = "#")

if self.GUI\_tape\_positions[3] < len(self.TM.tape):

self.lblTape3.config(text = str(self.TM.tape[self.GUI\_tape\_positions[3]]))

else:

self.lblTape3.config(text = "#")

if self.GUI\_tape\_positions[4] < len(self.TM.tape):

self.lblTape4.config(text = str(self.TM.tape[self.GUI\_tape\_positions[4]]))

else:

self.lblTape4.config(text = "#")

if self.GUI\_tape\_positions[5] < len(self.TM.tape):

self.lblTape5.config(text = str(self.TM.tape[self.GUI\_tape\_positions[5]]))

else:

self.lblTape5.config(text = "#")

if self.GUI\_tape\_positions[6] < len(self.TM.tape):

self.lblTape6.config(text = str(self.TM.tape[self.GUI\_tape\_positions[6]]))

else:

self.lblTape6.config(text = "#")

if self.GUI\_tape\_positions[7] < len(self.TM.tape):

self.lblTape7.config(text = str(self.TM.tape[self.GUI\_tape\_positions[7]]))

else:

self.lblTape7.config(text = "#")

# this moves the tape back if the tape is too short for the GUI

if self.GUI\_tape\_positions[3] > len(self.TM.tape):

for index in range(len(self.GUI\_tape\_positions)):

self.GUI\_tape\_positions[index] -= 1

# method to handle increasing the speed of the machine

def increaseSpeed(self):

# handles if the speed goes too high

if self.speedindex < 10:

# changes index to be used in speeds list

self.speedindex += 1

self.speedLabel -= 1

self.lblCurrentSpeed.config(text = "Speed: " + str(self.speedLabel))

# sets the speed using the speeds list defined when initialised, this list holds all the different values the sleep can use in the TuringState

self.TM.time = self.speeds[self.speedindex]

else:

print "Too Slow"

print "Interval: ", self.TM.time

def decreaseSpeed(self):

# handles if the speed goes too low

if self.speedindex > 0:

# changes index to be used in speeds list

self.speedindex -= 1

self.speedLabel += 1

self.lblCurrentSpeed.config(text = "Speed: " + str(self.speedLabel))

# sets the speed form the speeds list

self.TM.time = self.speeds[self.speedindex]

else:

print "Too Fast"

print "Interval: ", self.TM.time

def setInstruction(self):

# gets all of the instructions from the textboxes

state = self.txtScannedState.get()

self.txtScannedState.delete(0,END)

symbol = self.txtScannedSymbol.get()

self.txtScannedSymbol.delete(0,END)

direction = self.txtDirection.get()

self.txtDirection.delete(0,END)

new\_symbol = self.txtNewSymbol.get()

self.txtNewSymbol.delete(0,END)

new\_state = self.txtNewState.get()

self.txtNewState.delete(0,END)

root.update()

# and adds them to the temp\_instructions using setInstructions

self.TM.setInstruction(state, symbol, direction, new\_symbol, new\_state)

def submitTape(self):

# gets the tape from the textbox and if the textbox is empty the tape is filled with blank symbols

if self.txtTape.get("1.0", END) == "\n":

self.txtTape.insert(END, "#")

# fills the tape until it reaches a length of 8 (16 is used due to the commas in the string)

while len(str(self.txtTape.get("1.0", END))) < 16:

self.txtTape.insert(END, ",#")

# gets the tape

input\_tape = str(self.txtTape.get("1.0", END))

# removes the new line

input\_tape = input\_tape[:-1]

# using CSV, seperate the values on the tape

self.TM.tape = input\_tape.split(",")

# sets the original tape used for navigation when the machine is not running

self.TM.original\_tape = self.TM.tape

## fills the tape with the new tape values

self.fillTape()

# emptys the testbox

self.txtTape.delete('1.0', END)

print "Current Tape: ", self.TM.tape

# resets the tape position

self.TM.tape\_position = 0

def fillTape(self):

# initially resets the GUI tape

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]

self.lblTape0.config(text = "#")

self.lblTape1.config(text = "#")

self.lblTape2.config(text = "#")

self.lblTape3.config(text = "#")

self.lblTape4.config(text = "#")

self.lblTape5.config(text = "#")

self.lblTape6.config(text = "#")

self.lblTape7.config(text = "#")

# sets the values on the tape depending on the the length

if len(self.TM.tape) >= 5:

self.lblTape3.config(text = str(self.TM.tape[0]))

self.lblTape4.config(text = str(self.TM.tape[1]))

self.lblTape5.config(text = str(self.TM.tape[2]))

self.lblTape6.config(text = str(self.TM.tape[3]))

self.lblTape7.config(text = str(self.TM.tape[4]))

elif len(self.TM.tape) >= 4:

self.lblTape3.config(text = str(self.TM.tape[0]))

self.lblTape4.config(text = str(self.TM.tape[1]))

self.lblTape5.config(text = str(self.TM.tape[2]))

self.lblTape6.config(text = str(self.TM.tape[3]))

elif len(self.TM.tape) >= 3:

self.lblTape3.config(text = str(self.TM.tape[0]))

self.lblTape4.config(text = str(self.TM.tape[1]))

self.lblTape5.config(text = str(self.TM.tape[2]))

elif len(self.TM.tape) >= 2:

self.lblTape3.config(text = str(self.TM.tape[0]))

self.lblTape4.config(text = str(self.TM.tape[1]))

elif len(self.TM.tape) >= 1:

self.lblTape3.config(text = str(self.TM.tape[0]))

def setStartingState(self):

# if the textbox has a value

if str(self.txtStartState.get("1.0", END)) != "":

# it will get the value

self.TM.starting\_state = str(self.txtStartState.get("1.0", END))

# and set that as the next starting state

self.TM.starting\_state = self.TM.starting\_state[:-1]

print "starting State set to: ", self.TM.starting\_state

self.lblStartingState.config(text = "Starting State: " + self.TM.starting\_state)

# emptys the textbox

self.txtStartState.delete(1.0, END)

def setTapePosition(self):

# if the textbox has a value

if str(self.txtTapePosition.get()) != "":

# and if the int value in is more than 0 (avoids setting position in the minus')

if int(str(self.txtTapePosition.get())) >= 0:

# sets the new tape position

self.TM.tape\_position = int(self.txtTapePosition.get())

print "Tape Position set to: ", str(self.TM.tape\_position)

self.lblTapePosition.config(text = "Tape Position: " + str(self.TM.tape\_position))

# deletes the textbox contents

self.txtTapePosition.delete(0, END)

# and resets the tape positions before moving the tape to the right

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]

# moves the tape to the right on the tape position number

for x in range(self.TM.tape\_position):

app.scrollRight()

def finaliseTape(self):

finaltapestr = ""

# removes the blank symbols on the front of the tape

while self.TM.tape[0] == "#":

self.TM.tape.pop(0)

if self.TM.tape[0] != "#":

break

# removes the blank symbols on the end of the tape

if len(self.TM.tape) > 1:

while self.TM.tape[-1] == "#":

self.TM.tape.pop(-1)

if self.TM.tape[-1] != "#":

break

# finialises the tape and makes it readable for the user

for index in range (len(self.TM.tape)):

finaltapestr += "[" + str(self.TM.tape[index]) + "] "

return finaltapestr

def runMachine(self):

# resets the GUI tape positions beack to its initial state

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]

# re-fills the tape

self.fillTape()

root.update()

# starts the machine using the Turing machine instance

self.TM.startMachine()

# displays a popup window showing the final tape of the machine

def finalTapePopup(self):

# calls finaliseTape to get the string to print to the popup window

finaltape\_str = self.finaliseTape()

popup\_finaltape = Toplevel()

popup\_finaltape.title("Final Tape")

popup\_finaltape.geometry("300x150+750+250")

# sets the final tape on the pop up

lblInstructionFound = Label(popup\_finaltape, text = "Final Tape: " + finaltape\_str)

lblInstructionFound.pack()

lblFinalTape = Label(popup\_finaltape, text = finaltape\_str)

lblFinalTape.pack()

btnDismiss = Button(popup\_finaltape, text="Dismiss Window", command = popup\_finaltape.destroy)

btnDismiss.pack()

btnDismiss.focus()

# when the popup runs, the machine will reset allowing for another instance of the Turing Machine to run

self.resetMachine()

def helpPopup(self):

# string for the help window, explaining how to use the machine

help\_string = """A turing machine works by a set of instructions and acting accordingly depending on what state the machine is and what is present on the tape

An example instruction would be, if in state A and symbol is 0, change symbol to 1 and move to state B, then move right.

<--- All INSTRUCTIONS LIST --->

the instructions can be read as follows

[STATE, SYMBOL, DIRECTION, NEW SYMBOL, NEW STATE]

<--- ADD INSTRUCTIONS --->

instructions can be added to the machine on the right side of the window and will be appeneded to the All Instructions window in the center

<--- INSERT TAPE --->

to insert a tape use comma seperated values such as [1, 2, 1, 2, 3, 4, 1, 2]

this will then show on the tape at the top of the window

<--- INSERT STARTING STATE --->

allows you to change teh starting state of the machine and will be displayed at the bottom of the window

<--- SPEED --->

allows for change of speed for a max of 10 (will run instantly) and min of 0

<--- TAPE POSITION --->

allows the setting of changing the position on the tape, to any value above 0

<--- RESET AND LOAD DATA FROM FILE --->

allows the use of pre loaded data into the machine of instructions and a tape. State and position will be set seperatly

use files instructions.txt and tape.txt

instructions are written in format: state,symbol,direction,new\_symbol,new\_state A,1,R,0,B (each line will represent a new instruction)

tape is written in format: 1,2,3,1,2,3,4,1

<--- TAPE DIRECTION --->

use the arrows on either side of the tape to navigate through the tape (viewing experience only whilst the machine is not running)

<--- RUNNING INSTRUCTIONS --->

will show the currently executed instructions

<--- CLEAR ALL INSTRUCTIONS --->

clears all of the instructions from the machine """

popup\_finaltape = Toplevel()

popup\_finaltape.title("Help")

popup\_finaltape.geometry("1000x650+750+250")

lblInstructionFound = Label(

popup\_finaltape,

text = help\_string)

lblInstructionFound.pack()

btnDismiss = Button(popup\_finaltape, text="Dismiss Window", command = popup\_finaltape.destroy)

btnDismiss.pack()

btnDismiss.focus()

def clearInstructions(self):

# resets the instructions on the machine

self.TM.temp\_instructions = {}

# this avoids key errors, and creates the key if it does not exist

self.TM.temp\_instructions = defaultdict(dict)

self.lstShowInstructions.delete(0, END)

def resetMachine(self):

# resets everything in the machine to its default attributes

self.lblTape0.config(text = "#")

self.lblTape1.config(text = "#")

self.lblTape2.config(text = "#")

self.lblTape3.config(text = "#")

self.lblTape4.config(text = "#")

self.lblTape5.config(text = "#")

self.lblTape6.config(text = "#")

self.lblTape7.config(text = "#")

self.TM.tape\_position = 0

self.TM.tape = []

self.GUI\_tape\_positions = [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7] # positions of the tape, which will change depending on left or right

self.TM.available\_states = {}

self.TM.temp\_instructions = {}

self.TM.temp\_instructions = defaultdict(dict)

self.TM.state = None

self.TM.starting\_state = None

self.TM.original\_tape = []

self.TM.current\_state = None

self.lstShowInstructions.delete(0,END)

self.lblStartingState.config(text = "Starting State: N/A")

self.lblTapePosition.config(text = "Tape Position: 0")

self.lblScannedState.config(text = "N/A")

self.lblScannedSymbol.config(text = "N/A")

self.lblDirection.config(text = "N/A")

self.lblNewSymbol.config(text = "N/A")

self.lblNewState.config(text = "N/A")

root.update()

def loadFromFile(self):

instruction = []

# resets the machine before loading from a file

self.resetMachine()

try:

# sets the file to the instructions.txt for reading

file = open("instructions.txt", "r")

except IOError: # IOError handles if the file does not exist

print "Instructions file does not exist"

return

for line in file:

# for each line in the file, it is stripped and split using commas, and appaended to the instructions 2D list

instruction = map(str, line.strip().split(','))

self.TM.setInstruction(instruction[0], instruction[1], instruction[2], instruction[3], instruction[4])

file.close

try:

# sets teh file to tape.txt for reading

file = open("tape.txt", "r")

except IOError: # IOError for if the file does not exist

print "Tape file does not exist"

return

for line in file:

# reads the tape from the file, using CSV

self.TM.tape = map(str, line.strip().split(','))

self.fillTape()

self.TM.original\_tape = self.TM.tape

file.close

class Transition:

# if the state machine cannot run the transition between each state this will serve as a default message

def executeInstruction(self):

print "Cannot Execute Instruction"

return False

class TuringState(State, Transition):

''' this will serve as the main state in the state machine and will be used to

dynamically create the states needed in the machine this is done through looking

through the instructions provided and adding the states which have been speicifed

as a TuringState() object. This means that there will be multiple of this state in

the state machine, each representing the different states in the State and Turing Machine'''

# depending on the state the machine is in, a dictionary will be used to contain all

# instructions which can be executed in the current state. This will be different with each state in the machine

instructions = {}

# when the state is created the instructions which are relevent to the state will be passed in through a

# parameter and then saved in the current states instructions dictionary

def \_\_init\_\_(self, context, new\_instructions):

State.\_\_init\_\_(self, context)

self.instructions = new\_instructions

def executeInstruction(self):

print "(" + str(self.instructions) + ") Instructions Asociated with State: (" + self.current\_context.state

# the machine gets the current symbol on the tape and saves it to a variable to be used

current\_symbol = self.current\_context.tape[self.current\_context.tape\_position]

# if the symbol on the tape exists in the instructions, the instructions relevent to

# the symbol will be carried out

if current\_symbol in self.instructions:

# prints out the current state of the machine and has a sleep for readability

print "(" + self.current\_context.state + ") Current State"

app.lblScannedState.config(text = self.current\_context.state)

sleep(self.current\_context.time)

root.update()

# prints the current symbol read from the tape

app.lblScannedSymbol.config(text = current\_symbol)

sleep(self.current\_context.time)

root.update()

# prints the direction the machine will take next

print "(" + self.instructions[current\_symbol][DIRECTION] + ") Direction"

app.lblDirection.config(text = self.instructions[current\_symbol][DIRECTION])

sleep(self.current\_context.time)

root.update()

# the machine will then change the current symbol on the tape to the new symbol provided in the instructions

self.current\_context.tape[self.current\_context.tape\_position] = self.instructions[current\_symbol][NEW\_SYMBOL]

print "(" + self.instructions[current\_symbol][NEW\_SYMBOL] + ") New Symbol"

app.lblNewSymbol.config(text = self.instructions[current\_symbol][NEW\_SYMBOL])

sleep(self.current\_context.time)

root.update()

# then the machine will transition to the new state in the machine

self.current\_context.setState(self.instructions[current\_symbol][NEXT\_STATE])

print "(" + self.instructions[current\_symbol][NEXT\_STATE] + ") New State"

app.lblNewState.config(text = self.instructions[current\_symbol][NEXT\_STATE])

sleep(self.current\_context.time)

root.update()

# the machine will move either left or right depending on the instructions,

# if an invalid direction is provided, the heads position on the tape will not move

if self.instructions[current\_symbol][DIRECTION] == "R":

if self.current\_context.tape\_position >= (len(self.current\_context.tape) - 2):

self.current\_context.tape += ["#"]

self.current\_context.tape\_position += 1

app.scrollRight()

root.update()

elif self.instructions[current\_symbol][DIRECTION] == "L":

self.current\_context.tape = ["#"] + self.current\_context.tape

app.scrollLeft()

root.update()

else:

print "Not a Valid Direction"

print "New Tape: ", self.current\_context.tape

else:

return False

class TuringMachine(StateContext, Transition):

''' the main TuringMachine class will be the main way to access the state machine

and set all of the correct attributes for defining the condition of the Turing Machine.

This class will also include methods for adding an instruction to the temp\_instructions

dictionary which will handle all of the states in the machine and their associated instructions,

as well as handle adding all of the states to the machine'''

def \_\_init\_\_(self):

# default attributes for the Turing Machine

self.tape = []

self.original\_tape = []

self.starting\_state = None

self.tape\_position = 0

self.time = 0.2

# this will hold all of the instructions before they are initalised in the machine,

# this is a nested dictionary containing 2 sets of keys,

# the state keys and the symbol keys

self.temp\_instructions = {}

# this is used through the collections library, giving the ability to create a key

# in the nested dictionary if none is present

self.temp\_instructions = defaultdict(dict)

# calls the correct method with the current state of the machine

def executeInstruction(self):

return self.current\_state.executeInstruction()

# this method is used to save an instruction to the machines temporary dictionary.

# This will be called whenever a single instruction is submitted and adds the

# instruction or creates a new key if needed in the nested dictionary

def setInstruction(self, state, symbol, direction, new\_symbol, new\_state):

# saves the instruction to the nested dictionary using the state and the symbol provided,

# while saving a list of the actions the machine will take

self.temp\_instructions[state][symbol] = [direction, new\_symbol, new\_state]

sinstruction = "[" + state + ", " + symbol + "]: " + str(self.temp\_instructions[state][symbol])

app.lstShowInstructions.insert(END, sinstruction)

# this method will initalise the machine when all of the instructions are present.

# The machine will create the states dynamically depending on what instructions are saved

def startMachine(self):

# depending on the tape position, the machine will scroll right that many times to appear correctly on the GUI

for x in range(self.tape\_position):

app.scrollRight()

# firstly the machine will get all of the keys (states) that are used in the instructions

states = self.temp\_instructions.keys()

# if there is not starting state, the first state in the instructions will be set as so

if self.starting\_state is None:

self.starting\_state = states[0]

print "Starting State set to: " + self.starting\_state

app.lblStartingState.config(text = "Starting State: " + self.starting\_state)

# and will then cycle through the states and add them to the machine with the instructions

# associated with each state, passed in via a method parameter

for state in states:

self.availableStates[state] = TuringState(self, self.temp\_instructions[state])

# the machine will then go through all of the states that the machine should be able to transition to

# and if it does not exist it will create it by looking into the nested dictionaries contents.

# If the state does not exist it will be create with an empty set of instructions, these instructions

# are set to be empty as the state will not transition anywhere

for state in self.temp\_instructions:

for symbol in self.temp\_instructions[state]:

if self.temp\_instructions[state][symbol][NEXT\_STATE] not in self.availableStates.keys():

self.availableStates[self.temp\_instructions[state][symbol][NEXT\_STATE]] = TuringState(self, {})

# the starting state will then be set

self.setState(self.starting\_state)

# runs the machine while the machine does not reach a HALT or finds instructions

while (True):

if self.state == "HALT":

app.finalTapePopup()

break

if (self.executeInstruction() == False):

app.finalTapePopup()

break

if \_\_name\_\_ == '\_\_main\_\_':

root = Tk()

app = TuringMachineGUI(root)

root.geometry("750x480+250+250")

root.title("Dynamic Custom Turing Machine")

root.mainloop()

## Ethics Checklist

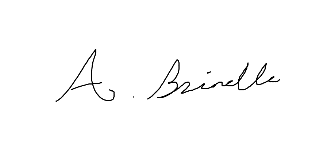
This form is only applicable for assessed exercises that use other people (‘participants’) for the collection of information, typically in getting comments about a system or a system design, or getting information about how a system could be used, or evaluating a working system.

If your proposed activity does not comply with any one or more of the points below then please contact your project supervisor and/or project coordinator for advice. If your evaluation does comply with all the points below, please sign this form and submit it with your assessed work.

1. Participants were not exposed to any risks greater than those encountered in their normal working 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. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback.*
2. The experimental materials were paper-based, or comprised software running on standard hardware. *Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones and PDAs.*
3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project. 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. Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.
4. No incentives were offered 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.
5. No information about the evaluation or materials was intentionally withheld from the participants. Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.
6. No participant was under the age of 16. Parental consent is required for participants under the age of 16.
7. No participant has an impairment that may limit their understanding or communication. Additional consent is required for participants with impairments.
8. Neither I nor my supervisor is in a position of authority or influence over any of the 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.
9. All participants were 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.
10. All participants have been informed of my contact details. All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.
11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions. The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation.
12. All the data collected from the participants is stored in an anonymous form. All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Student Name: Ashley Brindle

Student ID: 16022599

Student’s Signature: 

Date: 23 November 2018