Software Requirements Specification (SRS)
Macro Buddy

Team: **Macro Buddy-3**
Authors: **Roniel Abreu, Dan Hemphill, Nathan Lamberson, Seamus Rioux, Gregory Smelkov**
Customer: **Users who prefer a simplified approach to writing shell scripts**
Instructor: **Professor James Daly**

1 Introduction

This Software Requirements Specification for MacroBuddy includes detailed information on the application. It will cover information about this document, a high-level overview of what the application is, the specific requirements for the system, and functionality of the application through a variety of diagrams, as well as a description of and instructions to install the prototype.

1.1 Purpose

The purpose of this document is to provide a technical overview of the requirements of the Macro Buddy application. It is intended as a formal agreement between the client and the system designers about all aspects of this application. This document is intended for the team developing the application, though it may also benefit clients and stakeholders in understanding how the software was developed.

1.2 Scope

Macro Buddy will allow users to record commands in the emulated terminal built directly into the application. They will be able to save and name the group of commands as a macro and use them again repeatedly. Macro Buddy also allows for users to import any of their own shell files and use them in the application. It will also allow them to edit any of their recorded macros.

The goal of this product is to help programmers work efficiently. Macro Buddy will help users save time by recording their most used commands and being able to use them with the click of a button. This will help users run their commands in a shell as well as use macros all in one application.

Macro Buddy's scope is encompassed by a Python file and the Python libraries Tkinter and Pynput, as well as the terminal emulator Xterm. This application saves shell files in a single folder located in the same directory it is run in. It also has a built in text editor which will allow users to edit their shell files.

1.3 Definitions, acronyms, and abbreviations

1. Shell: Command line interface to operating system services
2. Script: Short executable commonly run by an interpreter
3. Emulator: Software that allows you to mimic features from different software
4. Tkinter: Python graphic library for GUIs
5. Pynput: Python I/O library
6. Xterm: Terminal emulator for X Window System
7. Interface: Communication bridge between components(including the user)

1.4 Organization

This SRS is organized into several sections. Section 1 includes the introduction, high-level descriptions of the software, and the organization of this document. Section 2 describes the project and covers the functions, users, constraints, and dependencies of the software. Section 3 features the specific requirements for MacroBuddy. Section 4 contains the modeling requirements, including the use case and class diagrams, sequence diagrams, and state diagrams. Section 5 has information about the prototype, namely how to run it and a sample scenario. Section 6 includes the references, sources, and a link to the MacroBuddy site.

2 Overall Description

This section will describe how Macro Buddy will be used to simplify the process of executing and writing shell commands for programmers. A full description of Macro Buddy in this section will be provided, as well as constraints for programmers using Macro Buddy.

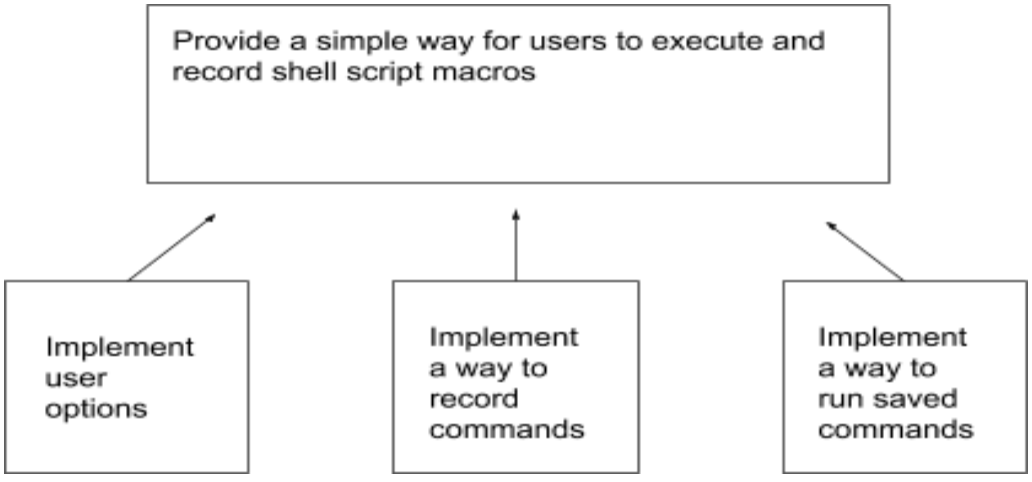
2.1 Product Perspective

In day-to-day work, all software developers will occasionally find themselves repeating the same sequences of terminal commands over and over again. Often our quality of life can be improved by writing a shell script to automate the process; however, the task of writing and debugging shell scripts as well as keeping those scripts organized can sometimes feel as time consuming or more than the task would have taken in the first place. Macro Buddy is an application that writes and organizes shell scripts for you. This project's scope includes a UNIX terminal and a system to run those scripts in a UI.

2.2 Product Functions

The major functions of Macro Buddy are:

- Allowing users to record shell scripts as macros and name them
- Displaying options in the user interface to execute the users recorded macros
- Integrated terminal for users to enter shell commands and run them
- Providing users with the ability to import their existing shell scripts



2.3 User Characteristics

A Macro Buddy user would likely be a programmer who frequently uses shell scripts. They are expected to have a thorough understanding of programming as well as writing and using shell scripts. In addition, the user is expected to be able to follow directions to download and install and run software. This tool is not meant for new programmers and will likely not benefit them as much as an experienced programmer.

2.4 Constraints

Macro Buddy scripts are constrained to their appropriate shell configuration in the terminal. In Linux, this can be changed by editing the SHELL environment variable. All of the features are limited to what is supported by Xterm. This will allow users to have full access to their file system but display features may be limited to Xterm's capabilities. The user is also limited to the text editor if they would like to edit shell files in the application.

The user is expected to use a unix based terminal to run their shell script commands. This constrains the user to only commands that are supported by UNIX terminals. Shell script files used in this application for macros need to be saved in a local folder within the directory in order to be used. Any shell script files that are imported will be saved in this folder as well in order to keep track of the name given for the macro.

The user is also required to have python installed on their computer to run the application. Macro Buddy requires that Tkinter, Pynput, and Xterm are also installed on the computer as well.

2.5 Assumptions and Dependencies

The user is required to have Python installed on their computer and depends on the terminal emulator program Xterm, as well as the Python libraries Pynput and Tkinter. The user must have a familiarity with Unix based shell scripts as Xterm emulates this within the application. The user must have an operating system compatible with the X Window System, which includes all Linux distributions and MacOS. In addition, our prototype assumes that the user does not wish to start a blank script in the text editor by omitting the option to start a new file, as this would defeat the purpose of the application.

2.6 Apportioning of Requirements

MacroBuddy is meant to assist a programmer and decrease the time it takes to write a shell script. At launch, it will be a simple UI with a terminal and buttons to run and manage existing scripts. In future versions, MacroBuddy can be expanded to support error checking and script validation. In addition, a future version would include the ability to pause and resume recording a macro script.

3 Specific Requirements

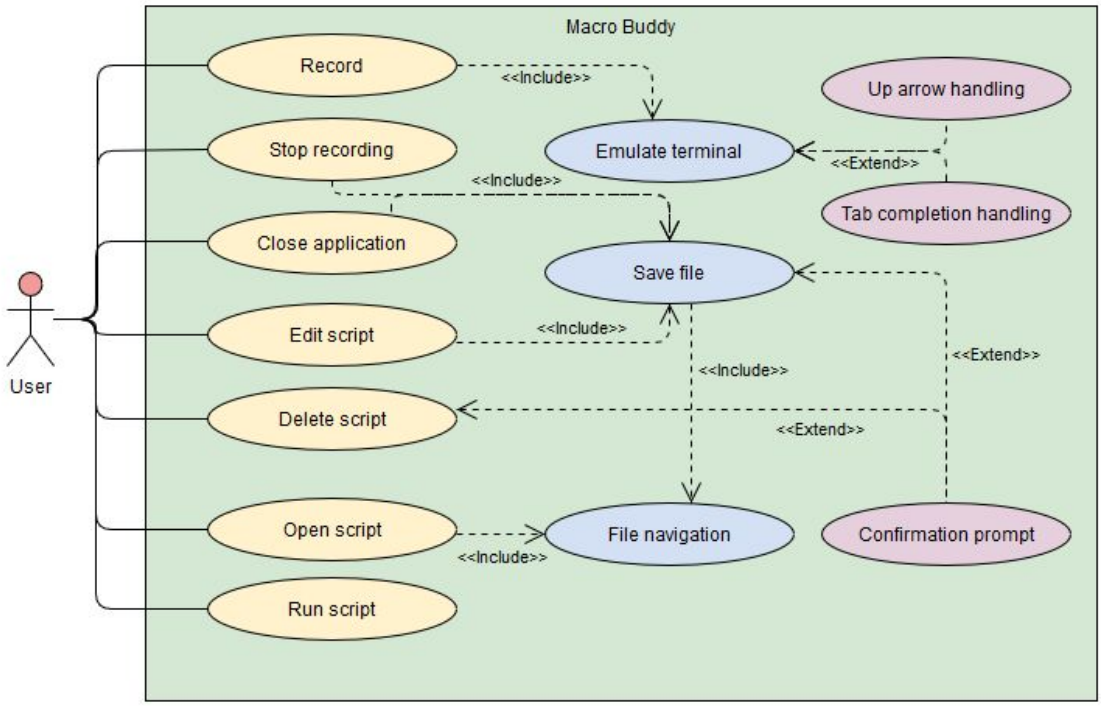
1. A terminal emulator will serve as the front end of the application.
    a. On UNIX-based systems the application will emulate the terminal directly from the operating system.
    b. On Windows, the application will capture an emulated UNIX-based terminal.
    c. Must have complete functionality like the built in terminal for the Operating System.
2. Shell scripts can be created through the application by recording a terminal session or loaded from existing shell script files.
    a. Once scripts are recorded, they will be stored within Macro Buddy's UI
    b. The user will have the option to stop a recording which will only be possible once a recording has been started.
        i. Once recording has started, everything will be saved to a file.
        ii. Once recording has been stopped, the current steps will be output to a file.
        iii. The user will then have a chance to name the file created which will appear in the UI of Macro Buddy.
    c. The user can specify the name of the script upon creation.
    d. Shell scripts can be saved into a directory specified by the user.
    e. The working directory of the script will be determined by the location at the start of recording.
    f. The user will be able to load and run pre existing scripts.
3. The user will also be able to delete macros, in order to not clutter up their space and file system.
    a. Before a macro is deleted, the user will be warned and have the chance to export it, preventing scripts from being tied to the application or being lost.
    b. Once a macro is deleted it will be removed from the user interface as well as the file storage for the macro.
4. The application should allow the user to select what command interpreter the application will use.
    a. The user can configure the preferred interpreter compatible with their system for use within Macro Buddy.
    b. The application will choose the appropriate script format based on the command interpreter the user chooses.
5. Command line completion and history will be dealt with to ensure the output files format matches the input of commands.
    a. Have an algorithm that removes excess characters from the script to ensure only commands being run are saved within the macro, or have the user be able to edit the script to clean it up.

    b. Errors in commands running will be removed from the script to eliminate double runs of commands that aren't meant to run twice.
    c. Stop commands (like Ctrl+C) will be trimmed to eliminate the faulty running of a macro.
        i. This is on a case by case basis, predefined before recording starts.
6. The macro recorder records all input, even mistakes.
    a. If a mistake is made then the user will have to re-record the sequence, or edit the previously recorded one.
7. Macro Buddy will remember previously made scripts/macros.
    a. Reopening Macro Buddy will still provide all previously made scripts.
    b. The terminal emulator will start fresh with no knowledge of the previous session.

4 Modeling Requirements

Use Case Diagram:

The diagram below contains the possible use cases for Macro Buddy. Each use case contains a description and other important information in the use case description section following the diagram.



Use Case Descriptions:

The following tables contain information on the Use Cases used in the above diagram, and explain their implementation in further detail.

Use Case:	Record
Actors:	User (initiator)
Type:	Primary
Description:	The user chooses to start recording shell commands
Includes:	Emulate terminal
Extends:	None

Cross-Refs:	None
Use-Cases:	None

Use Case:	Stop recording
Actors:	User (initiator)
Type:	Primary
Description:	The user chooses to stop recording shell commands and is prompted to save the file
Includes:	Save file
Extends:	None
Cross-Refs:	None
Use-Cases:	None

Use Case:	Close application
Actors:	User (initiator)
Type:	Primary
Description:	The user chooses to close the application and is prompted to save the file
Includes:	Save file
Extends:	None
Cross-Refs:	None
Use-Cases:	None

Use Case:	Edit script
Actors:	User (initiator)

Type:	Primary
Description:	The user is able to edit saved scripts via text editor or similar
Includes:	Save file
Extends:	None
Cross-Refs:	None
Use-Cases:	None

Use Case:	Delete script
Actors:	User (initiator)
Type:	Primary
Description:	The user chooses to delete a script and is prompted to ask if they are sure
Includes:	None
Extends:	None
Cross-Refs:	None
Use-Cases:	None

Use Case:	Open script
Actors:	User (initiator)
Type:	Primary
Description:	The user chooses an existing script to open
Includes:	File navigation
Extends:	None
Cross-Refs:	None

Use-Cases:	None

Use Case:	Run script
Actors:	User (initiator)
Type:	Primary
Description:	The user chooses to run an existing script
Includes:	None
Extends:	None
Cross-Refs:	None
Use-Cases:	None

Use Case:	Emulate terminal
Actors:	User (initiator)
Type:	Primary
Description:	The user chooses to start recording the shell. After they choose this option anything they type will be saved
Includes:	None
Extends:	None
Cross-Refs:	None
Use-Cases:	Record

Use Case:	Save file
Actors:	User (initiator)
Type:	Primary
Description:	User saves an edited or recorded script

Includes:	File navigation
Extends:	None
Cross-Refs:	None
Use-Cases:	Stop recording, Close application, Edit

Use Case:	File Navigation
Actors:	User (initiator)
Type:	Primary
Description:	User navigates to appropriate directory for opening and saving scripts
Includes:	None
Extends:	None
Cross-Refs:	None
Use-Cases:	Open script, Save file

Use Case:	Up arrow handling
Type:	Secondary
Description:	Handles input of the up arrow operations
Includes:	None
Extends:	Emulate terminal
Cross-Refs:	None
Use-Cases:	None

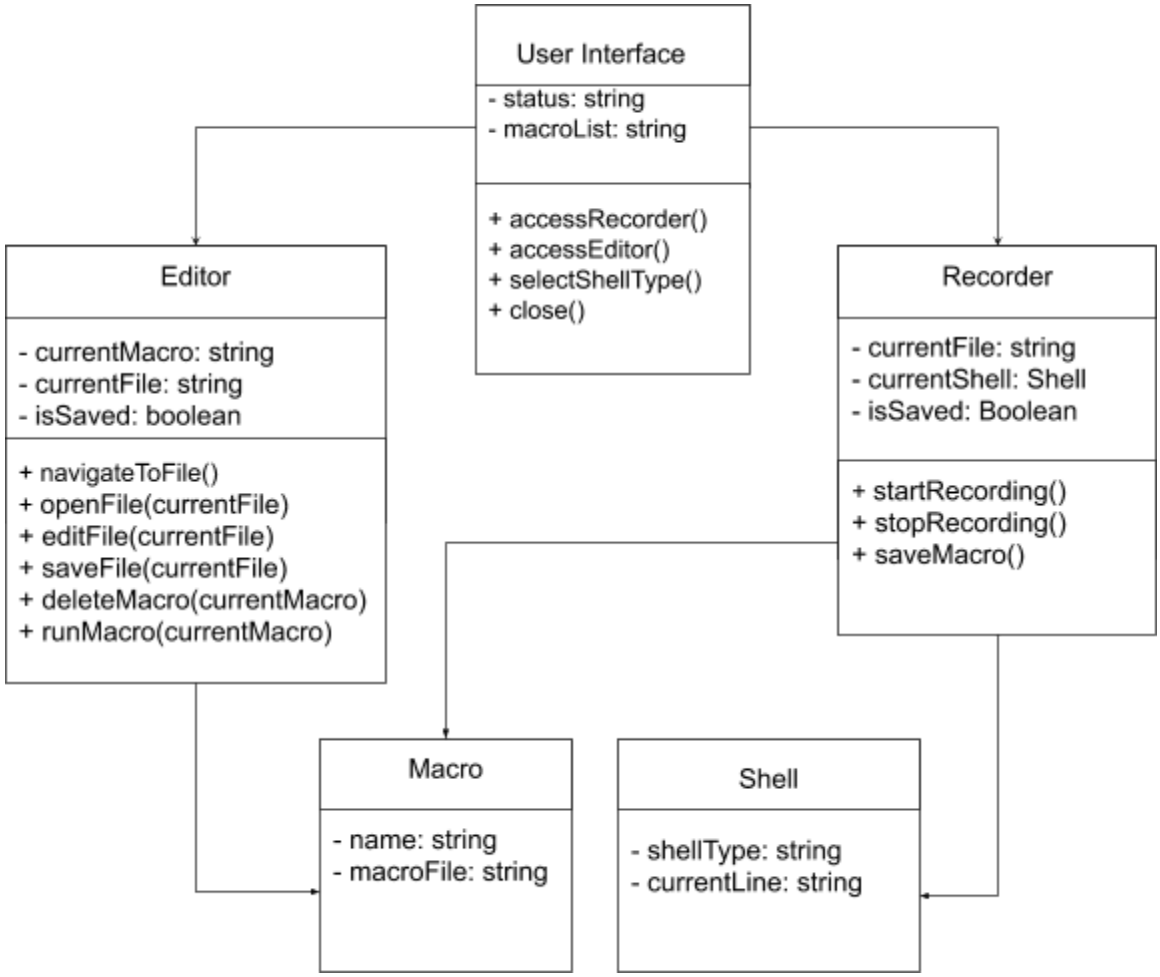
Use Case:	Tab completion handling
Type:	Secondary

Description:	Handles input of the tab key properly within the terminal and shell script
Includes:	None
Extends:	Emulate terminal
Cross-Refs:	None
Use-Cases:	None

Use Case:	Confirmation prompt
Actors:	User (initiator)
Type:	Secondary
Description:	Requests user to confirm deletion or overwriting of the script
Includes:	None
Extends:	Delete script, Save file
Cross-Refs:	None
Use-Cases:	None

Class Diagram:

The diagram below contains the classes for Macro Buddy. Each class contains attributes and methods tied to them which are explained in the descriptions following the diagram.



Class Descriptions:

The following tables contain information on the classes used in the above diagram, and explain their implementation in further detail.

Element Name		**Description**
Editor		The main editor for macros within the Macro Buddy application. This allows the user to open, edit, and run macros.
Attributes		
	currentMacro: string	A string containing the name of the current Macro.

	currentFile: string	A string containing the name of the file for the current Macro.
	isSaved: Boolean	A boolean representation of if the Macro is saved or not.
Operations		
	NavigateToFile()	Allows the user to navigate through their file directory and open a file to open, or choose a location so save it. This is similar to the operation on other applications.
	openFile(currentFile)	Open the file at a specified location.
	editFile(currentFile)	Begin editing the file at a specified location.
	saveFile(currentFile)	Save the file at the specified location, updating all changes made to it since its last save.
	deleteMacro(currentMacro)	Delete a Macro of a specified name, including the files for the macro.
	runMacro(currentMacro)	Run the Macro specified.
Relationships	This class gets information from elements in the Macro class and is accessible from the User Interface.	
UML Extensions	N / A	

Element Name		**Description**
Macro		This class contains the information for the Macros, allowings other classes to use them.
Attributes		
	name: string	A string containing the name Macro
	macroFile: string	A string containing the location of the file the Macro is associated with
Operations		
	N / A	N / A
Relationships	This class is accessed from the Editor and Recorder classes.	
UML Extensions	N / A	

Element Name	**Description**

Recorder		This class allows the user to record and save Macros using a specified Shell.
Attributes		
	currentFile: string	A string that constraints the current file, unsaved or not, that the macro is being recorded to.
	currentShell: Shell	A enum type that contains the information of the current shell Macro Buddy is emulating.
	isSaved: Boolean	A boolean representation of if the Macro is saved or not.
Operations		
	startRecording()	Start recording a sequence of actions in the emulated shell and begin writing it to the macro file.
	stopRecording()	Stop recording the shell and stop writing to the macro file.
	saveMacro()	Save the macro file by switching to the editor.
Relationships	This class gets information from the Macro and Shell classes and is accessible from the User Interface.	
UML Extensions	N / A	

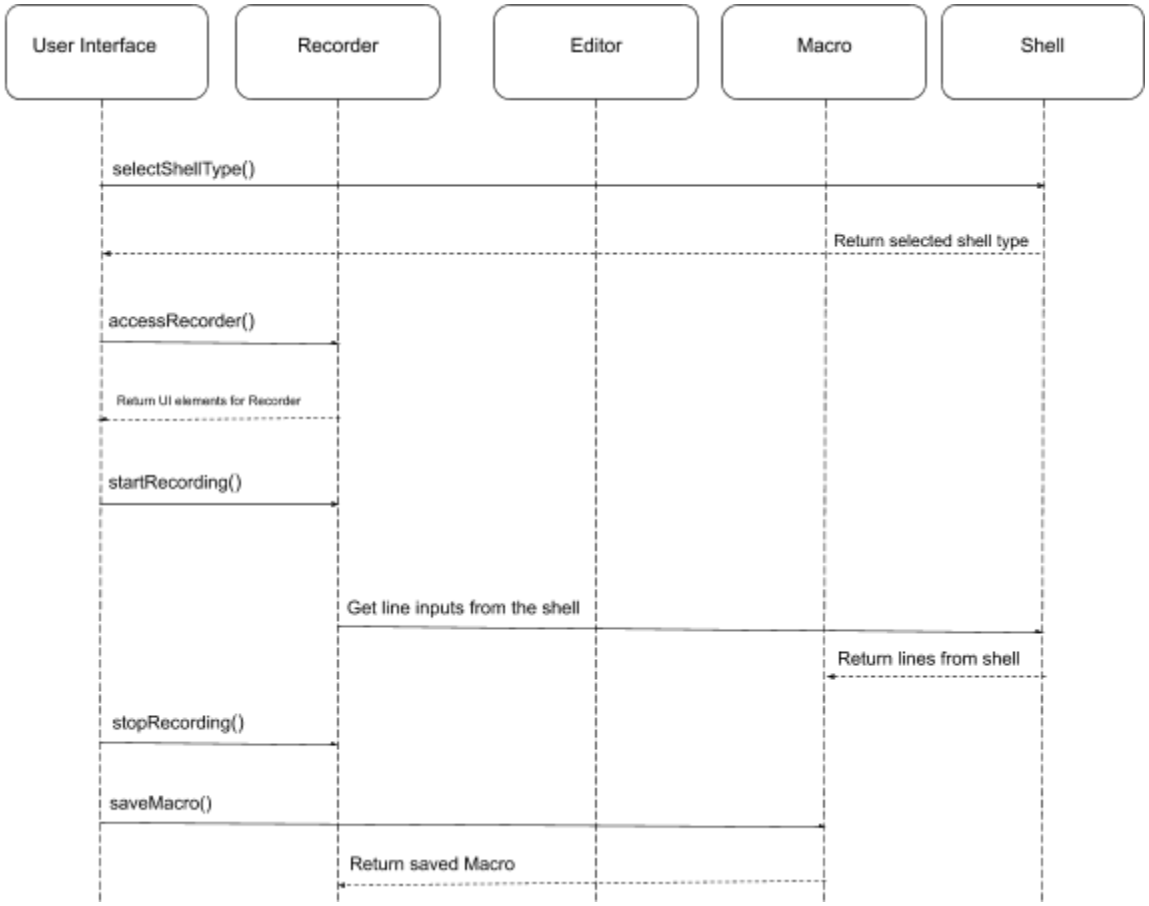
Element Name		**Description**
Shell		This class contains the information for the shell being used in Macro Buddy.
Attributes		
	shellType: string	A string containing the current type of shell being emulated.
	currentLine: string	A string containing the information being displayed on the current line within the shell.
Operations		
	N / A	N / A
Relationships	This class is accessed from the Recorder class	
UML Extensions	N / A	

Element Name	**Description**

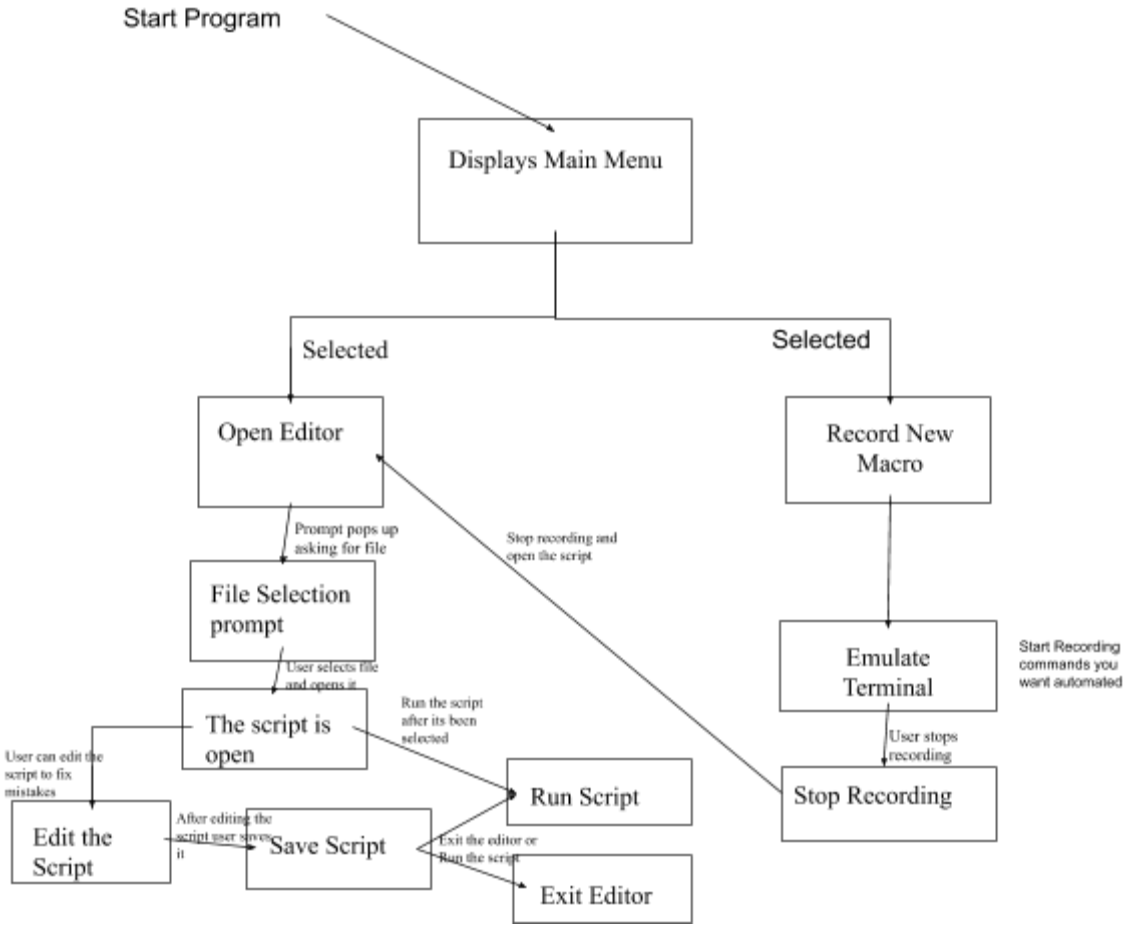
User Interface		This is the class that the user has direct access to and interacts with to use Macro Buddy. It allows for access to the Recorder and Editor, which supplies the main functionality of the Macro Buddy application.
Attributes		
	status: string	The current status of the Macro Buddy application (ex. Running, Frozen, etc.)
	macroList: string	A list of the current macros in Macro Buddy.
Operations		
	accessRecorder()	Gives the user access to the Recorder class.
	accessEditor()	Gives the user access to the Editor class.
	selectShellType()	Lets the user change the current shell type being emulated.
	close()	Close out of the Macro Buddy application.
Relationships	This class reads information from Editor and Recorder classes.	
UML Extensions	N / A	

Sequence Diagrams:

The user can choose the shell type they want to use from the user interface. After selecting a shell type, they can access the recorder from the User Interface. After accessing the recorder they can start recording commands in the shell. As commands are typed into the shell, they are written into a file created by the recorder. At any point the user can stop recording the commands they are typing into the shell. After they have stopped recording they can choose to save the macro file, which will allow for the user to run the macro at a later time when they wish to.



State Diagram: State Diagram to represent the events in the order that they will take place and the expected behavior of each event.



5 Prototype

Our prototype includes:

1. a terminal interface for recording macros
2. a text editor interface for loading, editing and executing saved macros

The terminal interface offers an embedded Xterm console that is configurable to any UNIX-based command line shell the user wants. It includes two buttons: an “Open Script” button, which allows the user to open an existing script in the text editor interface, and a “Record New Macro” button (that toggles to a “Stop Recording” button when pressed), which will begin recording terminal commands and continue until the user chooses to stop(Fig 1).

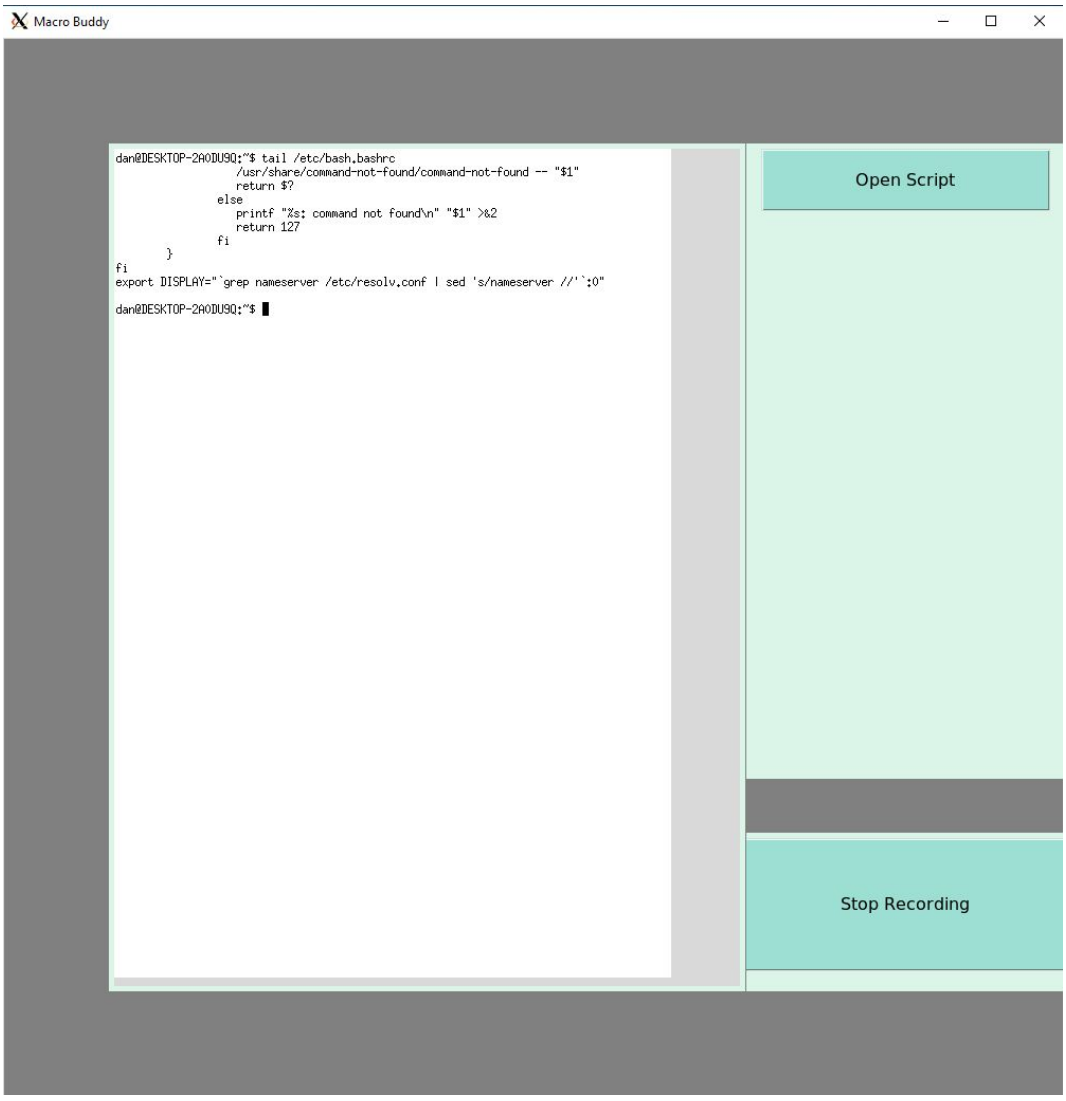


Fig 1

The text editor interface is accessible by selecting “Open Script”, and offers the user buttons to run the selected script, or exit the editor and return to the previous terminal. It also includes a menu bar with basic text editor options, including the options to delete a script, open a different script or start a new one from scratch, if the user so chooses(Fig 2).

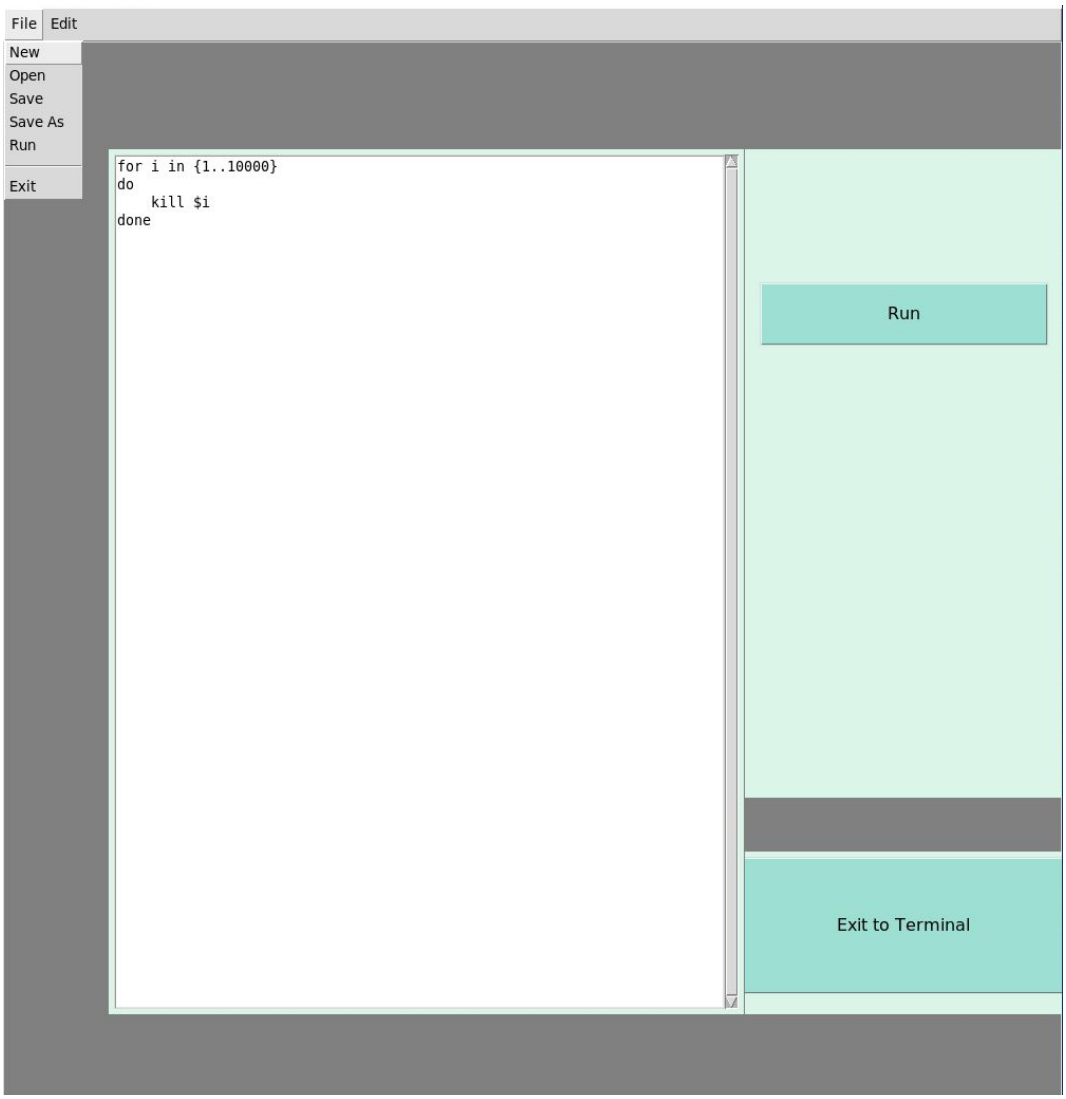


Fig 2

5.1 How to Run Prototype

- This is a Python 3 application. In UNIX-based systems, it can be executed by typing `python3 macroBuddy.py` from the command line.
- It has the following dependencies:
    1. tkinter
    2. pynput
    3. xterm
- These can be installed with the following command line inputs, respectively:
    1. `sudo apt-get install python3-tk`
    2. `pip3 install pynput` (requires python3-pip)
    3. `sudo apt-get install xterm`
- It works with any OS compatible with the X Window System.
- Source code and step-by-step instructions can be found here: https://github.com/IsItGreg/MacroBuddy

5.2 Sample Scenarios

A simple use scenario for the application is when a user needs to write a quick script to automate a task for work. The user opens the application, hits the record button, types the desired sequence of commands in the application terminal, and hits stop recording(see Fig 1). Note that the terminal is fully functional, and all typed commands will be executed.

A prompt asks the user if they want to save and what to name the file. The user then presses the “open script” button, navigates to the named script, and opens it. The text editor interface appears with the loaded script, where the user may review and edit the script if necessary(Fig 2); once satisfied, the user may select “Run” to repeat the typed commands whenever they wish.

6 References

[1]"Graphical User Interfaces with Tk — Python 3.9.0 documentation", *docs.python.org*, 2020. [Online]. Available: https://docs.python.org/3/library/tk.html. [Accessed: 14- Nov- 2020].

[2]"Qt for Python — Qt for Python", *doc.qt.io*, 2020. [Online]. Available: https://doc.qt.io/qtforpython/. [Accessed: 14- Nov- 2020].

[3]"PyQt5 Reference Guide — PyQt v5.15.1 Reference Guide", *Riverbankcomputing.com*, 2020. [Online]. Available: https://www.riverbankcomputing.com/static/Docs/PyQt5/. [Accessed: 14- Nov- 2020].

[4]V. Bernat, "Writing your own terminal emulator", *MTU Ninja*, 2017. .

[5]G. Szabo, "Create your own interactive shell with cmd in Python", *Code Maven*, 2018. .

[6]D. Yoo, "getch()-like unbuffered character reading from stdin on both Windows and Unix « Python recipes « ActiveState Code", *code.activestate.com*, 2002. [Online]. Available: https://code.activestate.com/recipes/134892-getch-like-unbuffered-character-reading-from-stdin/. [Accessed: 15- Nov- 2020].

[7]"pynput 1.7.1", *PyPI*, 2020. [Online]. Available: https://pypi.org/project/pynput/. [Accessed: 15- Nov- 2020].

[8]R. Abreu, D. Hemphill, N. Lamberson, S. Rioux and G. Smelkov, "Macro Buddy", *Isitgreg.github.io*, 2020. [Online]. Available: https://isitgreg.github.io/MacroBuddy/. [Accessed: 15- Nov- 2020].

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.