EASY MAC

Team 4

Team Members
Brian Boggs
Melissa Burel
Liam Kilpatrick
Logan Price
Ben Spurlock

Team 4 Final Documentation

Team number: 4

Project name: Easy Mac

1. **Team members**:
    - Brian Boggs
        - Computer Science major, mathematics minor
        - Wrote InputNode and [unused] LoopNode. Designed file interface for save/load/new and basic logic to implement file generator and reader classes in GUI.
    - Melissa Burel
        - Computer Science major (second bachelor's student)
        - Wrote the KeyListener and ColorBrand class, co-wrote the main class for command-line entry (now deleted), wrote the code for GUI buttons left arrow, right arrow, record, the functions that make the arrows disappear and reappear, helped design the gui, created the [unused] splash screen, wrote Javadoc comments, readme file, and the final report.
    - Liam Kilpatrick
        - Computer Science major with Entertainment Computing concentration, and Mathematics minor
        - Wrote KeyInputNode, MouseInputNode, and IActionCanceller. Helped to design the GUI and gave advice on code for other parts of the program.
    - Logan Price,
        - Computer Science major
        - Made the fileGenerator and fileReader classes.
    - Ben Spurlock
        - Computer Science and Music major
        - Wrote the Node class and designed the structure and behavior of the timeline class. Helped format the GUI as well as add functionality for the add/remove node buttons.
2. **Description:** EasyMac allows users to create macros through keystrokes and mouse actions. The user controls the timings and how often the repetition of inputs. The interface facilitates the user to make a timeline of inputs with an adjustable time delay between them. The software will also allow the user to record inputs and generate a timeline based on them. The software will also allow for simple loops. The user can define a key or combination of keys to activate or deactivate the program. These timelines can be saved on the user's computer and loaded from the file.
3. **Analysis:**

    a. Exemplary use case[MB1]: Creating, editing and saving a macro

    1.

    The user opens the software.

2.

User clicks New and Record.

3.

System prompts the user to add keyboard inputs.

The user chooses keyboard inputs to add to the timeline.

4.

The user clicks Stop when they are done entering keystrokes.

The system assembles the timeline from nodes created by the user.

5.

The user chooses to execute the macro.

The system repeats the input(s) and their timings as specified in the user-designed timeline.

6.

The user decides to edit the macro and clicks the edit tab.

The system provides a view of each portion of the timeline and functionality to click through it.

7.

The user selects a node to delete.

The system removes that node and reconnects the other nodes in the timeline.

8.

The user clicks Start Macro.

The system repeats the input(s) and their timings as specified in the edited timeline.

9.

The user clicks the file tab and click save to save the macro.

The system opens a file saver window to enable this action and stores the timeline contents into a file.

10.

The user quits the program.

b. Essential and Enhancement Items

Essential items:

- User creates and saves a macro of key inputs through a graphical user interface.
- The user can edit the macro through a graphical user interface.

Enhancement items:

- Adding looping in the editing process.
- The ability to copy and add nodes in the editor.
- Tracking of mouse clicks and drags.

4. Design

a. Basic Parts

i. The software is composed of Node, Input Node, MouseInputNode, KeyInputNode, LoopNode, Timeline, FileReader, FileGenerator, KeyListener, ColorBrand, MainGUI, and IActionCanceller. These items allow the software to record keystrokes, save them to nodes, compile them together in a timeline, and play those back to the user who can then edit and save them.

b. The Domain Model

i. Node is the superclass of all the node classes and provides the basics of creating nodes, getting, and setting common values within the nodes. The public functions and their purposes are:

Public Methods (verbs):

- Node() – constructor for a blank node.
- Node(int delayDuration) – constructor for a node with delay duration.
- Node(Node prev, Node next, int delayDuration) - constructor for a node that indicates previous and next node along with the delay duration.
- getPrevNode() – returns the previous node.
- getButton() – returns the button value.
- setPrevNode(Node prevNode) – sets the previous node.
- setNextNode(Node nextNode)- sets the next node.
- setDelayDuration(int delayDuration) – sets the delay duration.
- getNextNode()- returns the next node.
- getDelayDuration()- returns the delay duration.
- getPressRelease() – get the Boolean for press or releases (true = pressed, false = released).
- runNode()- runs the contents of the node.

Relationships (nouns)

- Super class of InputNode, KeyInputNode, and MouseInputNode.

ii. InputNode extends the Node class creates input nodes from entered information.

Public methods (verbs):

- InputNode(Node prev, Node next, int delayDuration, boolean pressRelease, int button)- Constructs an InputNode object specifying previous node, next node, the delay duration, the press release, and the button.
- InputNode(int delayDuration, boolean pressRelease, int button)- Constructs an InputNode object specifying the delay duration, the press release, and the button.
- InputNode(Node prev, int button)- Constructs an InputNode object specifying the previous node and the button.
- getPressRelease()- returns the Boolean press or release value.
- setPressRelease(boolean pressRelease)- sets the Boolean press or release value.
- getButton() – returns the value of the key/button pressed.
- setButton(int button)- sets the key/button pressed.

Relationships (nouns)

- Subclass of Node.
- Superclass of KeyInputNode and MouseInputNode.

iii. MouseInputNode extends InputNode and creates objects that store information about mouse clicks, including if it is left, right, or middle as well as the x and y coordinates of the action.

Public methods (verbs):

- MouseInputNode(Node prev, int button)- Constructs a MouseInputNode object specifying the previous node and the next node.
- MouseInputNode(Node prev, int delayDuration, boolean pressRelease, int button)- Constructs a MouseInputNode object specifying the previous node, the delay time until the next node is called, the mouse clicks, and if the action is to click or release.
- MouseInputNode(Node prev, int delayDuration, boolean pressRelease, int button, int x, int y)- Constructs a MouseInputNode object specifying the previous node, the delay time until the next node is called, the mouse clicks, and if the action is to click or release.
- runNode()- runs the nodes contents.

Relationships (nouns)

- Subclass of InputNode.
- Used in the Timeline class.

iv. KeyInputNode extends InputNode and stores and return input information including the key that was pressed, the delay, if the key was pressed or released.

Public methods (verbs):

- KeyInputNode(Node prev, int button)- Constructs a KeyInputNode object specifying the previous node and the button identifier.
- KeyInputNode(Node prev, int delayDuration, boolean pressRelease, int button) - Constructs a KeyInputNode object specifying the previous node, the delay duration, if the key is pressed or released, and the button identifier.
- runNode()- runs the contents of the node.
- setInput(int delayDuration, int button, boolean pressRelease)- Sets the values within the node in terms of delay duration, the button, and if the button is being pressed or released.

v. LoopNode extends Node and stores the information related to looping in the timeline. It stores information about where the looping begins and ends and how many times the loop will occur.

Public methods (verbs):

- getRepeatCount()- returns the number of times to execute a loop.
- setRepeatCount(int repeatCount)- sets the number of times to execute a loop.
- runNode()- runs the content of the node.

Relationships (nouns):

- Subclass of Node.
- Used by Timeline to edit the timeline.

vi. Timeline class creates timeline objects that store, retrieve, and execute nodes.

Public methods (verbs):

- Timeline()- constructs a blank Timeline object.
- addNode(Node newNode)- adds a node to the timeline.
- removeEndNode()- removes the last node in the timeline.
- removeNode(Node node)- removes a node from the timeline and connects the surrounding nodes.
- removeCurrentNode()- removes the current node from the timeline.
- insertBeforeNode(Node currentNode, Node newNode, int delay)- inserts a node before the designated node.
- printTimeline()- prints the timeline to the console.
- runNodeDelay(Node cur)- runs the delay time using another object of Robot.
- runTimeline(IActionCanceller actionRunner)- executes the timeline.
- getStartNode()- returns the start node of the timeline.
- getEndNode()- returns the last node in the timeline.

- getCurrentNode()- returns the current node in the timeline.
- setCurrentNode(Node node)- sets the current node in the timeline.

Relationships (nouns):

- Stores MouseInputNode and KeyInputNode objects.
- Collaborates with LoopNode for looping functionality.
- Collaborates with FileGenerator and FileReader classes.

vii. FileReader reads a timeline from a file

Public Methods (verbs):

- readTimeline(String filePath)- returns a timeline found in the file path.

Relationships (nouns):

- Collaborates with the Timeline class

viii. FileGenerator constructs a file generator object that creates a file path for storing a user's macro

Public Methods (verbs):

- fileGeneration(Timeline tLine, String filePath)- creates a file path and stores the timeline object.

Relationships (nouns):

- Collaborates with FileReader and Timeline classes

ix. KeyListener implements NativeKeyListener and is used to listen for keystrokes and adds them to the timeline.

Public Methods (verbs):

- getCreatedTimeline()- returns the timeline created by the macro.
- nativeKeyPressed(NativeKeyEvent e)- Takes key presses, converts them to their ASCII value and adds them to the timeline.
- nativeKeyTyped(NativeKeyEvent e)- takes the native key typed event ID.
- nativeKeyReleased(NativeKeyEvent e)- takes the information from the key that is released.

Relationships (nouns):

- Collaborates with Timeline and KeyInputNode

c. Design Elements for Each of the Technical Items

   a. Graphical User Interface- The record button uses classes KeyListener which uses the Timeline class and the KeyInputNode class and brings together one of the big goals of this project, which is to record keystrokes outside of the console area.

b. Text Formatting or Processing- There are try/catch statements in the KeyInputNode class and MouseInputNode class, that will trigger for any incorrect key input.

c. Graphics- while our project does not use graphics to achieve the goal of creating macros, our resources folder (located under the file tab) has multiple team-created images. The team is especially proud of the easymacButton.png which indicates that an input is a keystroke.

d. Persistence – the classes FileGenerator and FileReader classes work together to save and load an instance of a timeline, using fileGeneration(Timeline tLine, String filePath) and readTimeline(String filePath).

5. Implementation

a. API Overview

1. The implementation of the domain model is divided into a few sections. Package cs321.team4.easymac contains the FileGenerator, FileReader, and Timeline classes. The APIs for the packages are as follows: FileGenerator has fileGeneration(Timeline tLine, String filePath); FileReader has readTimeline(); and Timeline has Timeline(), addNode(Node newNode), removeEndNode(), removeNode(Node node), removeCurrentNode(), insertBeforeNode(Node currentNode, Node newNode, int delay), printTimeline(), runNodeDelay(Node cur), runTimeline(IActionCanceller actionRunner), getStartNode(), getEndNode(), getCurrentNode(), setCurrentNode().

   Package cs321.team4.easymac.nodes also relates to the domain model and contains the classes related to the nodes, so Node, InputNode, KeyInputNode, MouseInputNode, and LoopNode. The APIs are as follows: Node has Node(), Node(int delayDuration), Node(Node prev, Node next, int delayDuration), getPrevNode(), getButton(), setPrevNode(Node prevNode), setNextNode(Node nextNode), setDelayDuration(int delayDuration), getNextNode(), getDelayDuration(), getPressRelease(), runNode(); Input Node has InputNode(Node prev, Node next, int delayDuration, boolean pressRelease, int button), InputNode(int delayDuration, boolean pressRelease, int button), InputNode(Node prev, int button), getPressRelease(), setPressRelease(boolean pressRelease), getButton(), setButton(int button); KeyInputNode has KeyInputNode(Node prev, int button), KeyInputNode(Node prev, int delayDuration, boolean pressRelease, int button), runNode(), setInput(int delayDuration, int button, boolean pressRelease); MouseInputNode has MouseInputNode(Node prev, int button), MouseInputNode(Node prev, int delayDuration, boolean pressRelease, int button), public MouseInputNode(Node prev, int delayDuration, boolean pressRelease, int button, int x, int y), runNode(); LoopNode has int getRepeatCount(), setRepeatCount(int repeatCount), runNode().

   In the utilities package, is the enum class ColorBrand that contains the color scheme for the macro and the KeyListener. The APIs for ColorBrand are getRGB(), getRed(), getGreen(), getBlue(), getColor(), getARGB(); KeyListener's APIs are nativeKeyPressed(NativeKeyEvent e), nativeKeyTyped(NativeKeyEvent e), and nativeKeyReleased(NativeKeyEvent e)

Finally, we have the cs321.team4.easymac.interfaces package which has the IActionCanceller class. Its APIs are cancelAction() and actionCancelled().

2. The implementation of the GUI is housed in the cs321.team4.easymac.gui package in the MainGUI class and its only API is MainGUI().

b. Public Elements

1. Node Class
   - The field prevNode is the previous node in the timeline. This in part satisfies the responsibility of traversing the nodes in the timeline.
   - The field nextNode is the next node in the timeline. This in part satisfies the responsibility of traversing the nodes in the timeline.
   - The field, delayDuration, is the delay in milliseconds between timeline events. This in part satisfies the responsibility editing the execution of the timeline.
   - The field, pressRelease, is a Boolean value representing if a key is pressed (true) or released (false). This in part satisfies the responsibility of capturing the action of the keystroke or mouse click.
   - The field, button, is a value that indicates which key is being pressed. This in part satisfies the responsibility of capturing the action of the keystroke or mouse click.
   - The constructor, Node(), constructs a blank node. This in part satisfies the responsibility editing and adding nodes to the timeline.
   - The constructor, Node(int delayDuration), constructs a node with delay duration. This in part satisfies the responsibility editing and adding nodes to the timeline.
   - The constructor, Node(Node prev, Node next, int delayDuration) constructs a node that indicates previous and next node along with the delay duration. This in part satisfies the responsibility editing and adding nodes to the timeline.
   - The method, getPrevNode(), returns the previous node. This in part satisfies the responsibility of traversing the nodes in the timeline.
   - The method, getButton() returns the button value. This in part satisfies the responsibility of viewing the nodes in the timeline.
   - The method, setPrevNode(Node prevNode), sets the previous node. This in part satisfies the responsibility of traversing the nodes in the timeline.
   - The method, setNextNode(Node nextNode) sets the next node in the timelint. This in part satisfies the responsibility of traversing the nodes in the timeline.
   - The method, setDelayDuration(int delayDuration), sets the delay value. This in part satisfies the responsibility editing the execution of the timeline.

- The method, getNextNode(), returns the next node in the timeline. This in part satisfies the responsibility of traversing the nodes in the timeline.
- The method, getDelayDuration(), returns the delay duration. This in part satisfies the responsibility of viewing the nodes in the timeline for possible editing.
- The method, getPressRelease(), returns the Boolean for press or releases (true = pressed, false = released). This in part satisfies the responsibility of viewing the nodes in the timeline for possible editing.
- The method, runNode(), runs the contents of the node. This in part satisfies the responsibility of executing the timeline.

2. InputNode Class
    - The field, pressRelease, is a Boolean value representing if a key is pressed (true) or released (false). This in part satisfies the responsibility of capturing the action of the keystroke or mouse click.
    - The field, button, is a value that indicates which key is being pressed. This in part satisfies the responsibility of capturing the action of the keystroke or mouse click.
    - The constructor, InputNode(Node prev, Node next, int delayDuration, boolean pressRelease, int button), constructs an InputNode object specifying previous node, next node, the delay duration, the press release, and the button. This in part satisfies the responsibility of editing and adding nodes to the timeline.
    - The constructor, InputNode(int delayDuration, boolean pressRelease, int button), constructs an InputNode object specifying the delay duration, the press release, and the button. This in part satisfies the responsibility of editing and adding nodes to the timeline.
    - The constructor, InputNode(Node prev, int button), constructs an InputNode object specifying the previous node and the button. This in part satisfies the responsibility editing and adding nodes to the timeline.
    - The method, getPressRelease(), returns the Boolean press or release value.
    - The method, setPressRelease(boolean pressRelease), sets the Boolean press or release value. This in part satisfies the responsibility of viewing the nodes in the timeline for possible editing.
    - The method, getButton(), returns the value of the key/button pressed. This in part satisfies the responsibility of viewing the nodes in the timeline.
    - The method, setButton(int button), sets the value for the key/button that is pressed. This in part satisfies the responsibility of setting and editing the actions in the timeline.
3. MouseInputNode Class
    - The field, x, is the x coordinate for the mouse press. This in part satisfies the responsibility of tracking where mouse clicks occur so that it can be recreated when the timeline is executed.

- The field, y, is the y coordinate for the mouse press. This in part satisfies the responsibility of tracking where mouse clicks occur so that it can be recreated when the timeline is executed.
- The constructor, MouseInputNode(Node prev, int button), constructs a MouseInputNode object specifying the previous node and the next node. This in part satisfies the responsibility of including mouse actions in the timeline.
- The constructor, MouseInputNode(Node prev, int delayDuration, boolean pressRelease, int button), onstructs a MouseInputNode object specifying the previous node, the delay time until the next node is called, the mouse clicks, and if the action is to click or release. This in part satisfies the responsibility of including mouse actions in the timeline.
- The constructor, MouseInputNode(Node prev, int delayDuration, boolean pressRelease, int button, int x, int y), constructs a MouseInputNode object specifying the previous node, the delay time until the next node is called, the mouse clicks, and if the action is to click or release. This in part satisfies the responsibility of including mouse actions in the timeline.
- The method, runNode(), executes the nodes contents. This in part satisfies the responsibility of executing the timeline.

4. KeyInputNode Class
    - The constructor, KeyInputNode(Node prev, int button), constructs a KeyInputNode object specifying the previous node and the button identifier. This in part satisfies the responsibility of including key actions in the timeline.
    - The constructor, KeyInputNode(Node prev, int delayDuration, boolean pressRelease, int button), constructs a KeyInputNode object specifying the previous node, the delay duration, if the key is pressed or released, and the button identifier. This in part satisfies the responsibility of including key actions in the timeline and allows for specific customization.
    - The method, runNode(), executes the contents of the node. This in part satisfies the responsibility of executing the timeline.
    - The method, setInput(int delayDuration, int button, boolean pressRelease), sets the values within the node in terms of delay duration, the button, and if the button is being pressed or released. This in part satisfies the responsibility of including nodes with specific customization.
5. LoopNode Class
    - The field, repeatCount, is the number if times a loop executes. This in part satisfies the responsibility of editing the timeline.
    - The field, actualNext, is the next node after the loop. This in part satisfies the responsibility of editing the timeline.
    - The method, getRepeatCount(), returns the number of times to execute a loop. This in part satisfies the responsibility of editing the timeline.

- The method, setRepeatCount(int repeatCount), sets the number of times to execute a loop. This in part satisfies the responsibility of editing and customizing the timeline.
- The method, runNode(), executes the nodes contents. This in part satisfies the responsibility of executing the timeline.

6. Timeline Class
    - The field, MAX_SIZE, indicates the largest size that the timeline can be. This in part satisfies the responsibility of executing the timeline within runnable boundaries.
    - The field, numOfNodes, is the number of nodes in the timeline. This in part satisfies the responsibility of editing and organizing the timeline.
    - The field, startNode, is the first node in the timeline. This in part satisfies the responsibility of executing the timeline.
    - The field, endNode is the last node in the timeline. This in part satisfies the responsibility of executing the timeline.
    - The field, currentNode, is the node that is currently being viewed or run. This in part satisfies the responsibility of executing and effectively editing the timeline.
    - The constructor, Timeline(), constructs a blank Timeline object. This in part satisfies the responsibility of building and executing the timeline.
    - The method, addNode(Node newNode), adds a node to the timeline. This in part satisfies the responsibility of building and editing the timeline.
    - The method, removeEndNode(), removes the last node in the timeline. This in part satisfies the responsibility of editing the timeline.
    - The method, removeNode(Node node), removes a node from the timeline and connects the surrounding nodes. This in part satisfies the responsibility of editing the timeline.
    - The method, removeCurrentNode(), removes the current node from the timeline. This in part satisfies the responsibility of editing the timeline.
    - The method, insertBeforeNode(Node currentNode, Node newNode, int delay), inserts a node before the designated node. This in part satisfies the responsibility of editing the timeline.
    - The node, printTimeline()- prints the timeline to the console. This in part satisfies the responsibility of viewing and editing the timeline.
    - The method, runNodeDelay(Node cur), runs the delay time using another object of Robot. This in part satisfies the responsibility of executing the timeline.
    - The method, runTimeline(IActionCanceller actionRunner), executes the timeline. This in part satisfies the responsibility of playing back the timeline.
    - The method, getStartNode(), returns the start node of the timeline. This in part satisfies the responsibility of executing the timeline.
    - The method, getEndNode(), returns the last node in the timeline. This in part satisfies the responsibility of editing and executing the timeline.

- The method, getCurrentNode(), returns the current node in the timeline. This in part satisfies the responsibility of executing and effectively editing the timeline.
- The method, setCurrentNode(Node node), sets the current node in the timeline. This in part satisfies the responsibility of editing the timeline.

7. FileReader Class
    - The method, readTimeline(String filePath), returns a timeline found in the file path. This in part satisfies the responsibility of persistence for a user to refer to saved work.
8. FileGenerator Class
    - fileGeneration(Timeline tLine, String filePath) creates a file path and stores the timeline object. This in part satisfies the responsibility of persistence for a user to save their work.
9. KeyListener Class
    - The field, userCreatedTimeline, is the timeline that is built by the KeyListener class. This in part satisfies the responsibility of building a timeline from key or mouse inputs.
    - The method, getCreatedTimeline(), returns the timeline that was created in KeyListener. This in part satisfies the responsibility of running and editing a timeline.
    - The method, nativeKeyTyped(NativeKeyEvent e), takes the "native key typed" event ID. This function was not used in the project, however it needed to be included in order to implement NativeKeyListener.
    - The method, nativeKeyPressed(NativeKeyEvent e), takes the key press value, converts it to ASCII, and adds it to a node, which is then added to the timeline. This in part satisfies the responsibility of building a timeline from key or mouse inputs.
    - The method, nativeKeyReleased(NativeKeyEvent e), takes the information from the key that is being released. This function was not used in the project, however it needed to be included to implement the NativeKeyListener class.
10. ColorBrand Class
    - The field, BRAND_BROWN, contains the code for the brown color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
    - The field, BRAND_DARKBLUE, contains the code for the dark blue color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
    - The field, BRAND_YELLOW, contains the code for the yellow color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
    - The field, BRAND_LIGHTBLUE, contains the code for the light blue color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.

- The field, BRAND_WHITE, contains the code for the white color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
- The method, getRGB(), returns the String value for a color's RGB. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
- The method, getRed(), returns the R of the RGB for a color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
- The method, getGreen(), returns the G of the RGB for a color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
- The method, getBlue(), returns the B of the RGB for a color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
- The method, getColor(), returns the red, green, and blue values for a color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.
- The method, getARGB(), returns the alpha and RGB for a color. This in part satisfies the responsibility of building a cohesive and pleasant user interface.

11. IActionCanceller
    - The method, cancelAction(), is used to trigger the cancellation process, synchronized with actionCancelled when implemented. This in part satisfies the responsibility of executing the timeline.
    - The method, actionCancelled(), returns a Boolean value to indicate if the macro is queued for cancellation. This in part satisfies the responsibility of executing the timeline.
12. MainGUI
    - The constructor, MainGUI(), creates the graphic user interface. This in part satisfies the responsibility of creating, editing, and executing a macro in a format that is easy and pleasing to the eye.

6. Exercise and Testing

a. Testing of each component was conducted as it was being created by each programmer. Once the GUI came together and classes started interacting, testing was conducted in terms of sequences of actions to ensure that the different classes and functions worked together as planned.

b. Walkthrough

The user runs the software. The GUI constructor is called with the set visible to true, so the GUI appears on the screen. The user clicks "New" and the system prompts the user on their next steps. It also calls the timeline constructor to make a new timeline, sets the current node to the head node, refreshes the view and updates the view of the left and right arrows in the editor.

The user clicks "Record" and enters desired keystrokes. The system puts a message to the screen informing the user that it is recording. It also calls KeyListener which logs the key inputs of the user. As the user enters keystrokes, the listener determines their type, assigns them to the correct node type, and adds them to the user-created timeline. The user clicks "Stop", and the system removes the key listener, gets the user-created timeline, and sets the current node to head node. It also refreshes the view and updates the view of the left and right arrows.

The user clicks the “Playback” button. The system plays the input(s) and their timings as specified in the user-designed timeline. The user decides to edit the macro and clicks the edit tab. The system provides a view of each portion of the timeline and functionality to click through it, refreshing the view and arrows with each click. The user determines a node to delete and clicks “Remove Node”. The system determines which node in the timeline has been selected, removes it, and adjusts the pointers of the surrounding nodes. It refreshed the view in terms of the node itself and the arrows visibility. The user clicks Start Macro. The system repeats the input(s) and their timings as specified in the edited timeline. The user clicks file and saves the macro. The system opens a file saver window to enable this action and stores the timeline contents into a file. The user quits the program.

7. Evaluation

a. The team’s original vision for the macro was ambitious and there were some items that ultimately were not implemented. Tracking mouse inputs was not realized as well as the looping functionality. Also, while the structure is there for editing a node, the implementation is still buggy. We originally budgeted 135 hours for this project. As of this report, the team spent 122 hours working on this project.

b. The portion of the application that the team is most excited about is the “Stop Macro” button, which is a forced button that can be used in the middle of the running timeline. While this item may not involve a complicated implementation, it is a feature that is frustratingly lacking in other macro programs.

c. Possible Software Improvements: The software could be improved with more extensive testing and more work on the appearance of the GUI.

d. Possible Software Extensions: Possible extensions include tracking mouse clicks and drags, accepting special key inputs, as well as node copying, and loop editing. The InputNode class is available as a superclass for mouse inputs and key inputs for future additions if there are other input methods that involve clicks and mouse movements, perhaps some kind of joystick.

8. Submission Notice

[illegible] [illegible] [illegible]

Ben Spurlock 4/21/2023 Liam Kilpatrick 4/21/2023 Logan Price 4/21/2023

[illegible] Melissa M. Burel

Brian Boggs 4/21/2023 Melissa Burel 4/21/2023