VAMIX

Software Engineering | University of Auckland

Video and Audio Editing Software

Chang Kon Han
chan743

Table of Contents

Table of Figures

VAMIX

VAMIX is a video and audio editing software which runs on the Linux operating system and needs avconv and VLC to work. VAMIX provides common features in video and audio editing.

VAMIX includes a media player which provides standard functionalities such as playing/pausing, fast forwarding or rewinding time etc. VAMIX also allows fullscreen support. Fullscreen allows users to enjoy VAMIX as a standalone media player or to watch their completed video edits. During fullscreen, the playback panel is visible if users move the mouse, disappearing after a few seconds if mouse is not moved. Time and volume sliders are present so users can change the values manually. Changing the slider can be done by dragging or clicking to desired position.

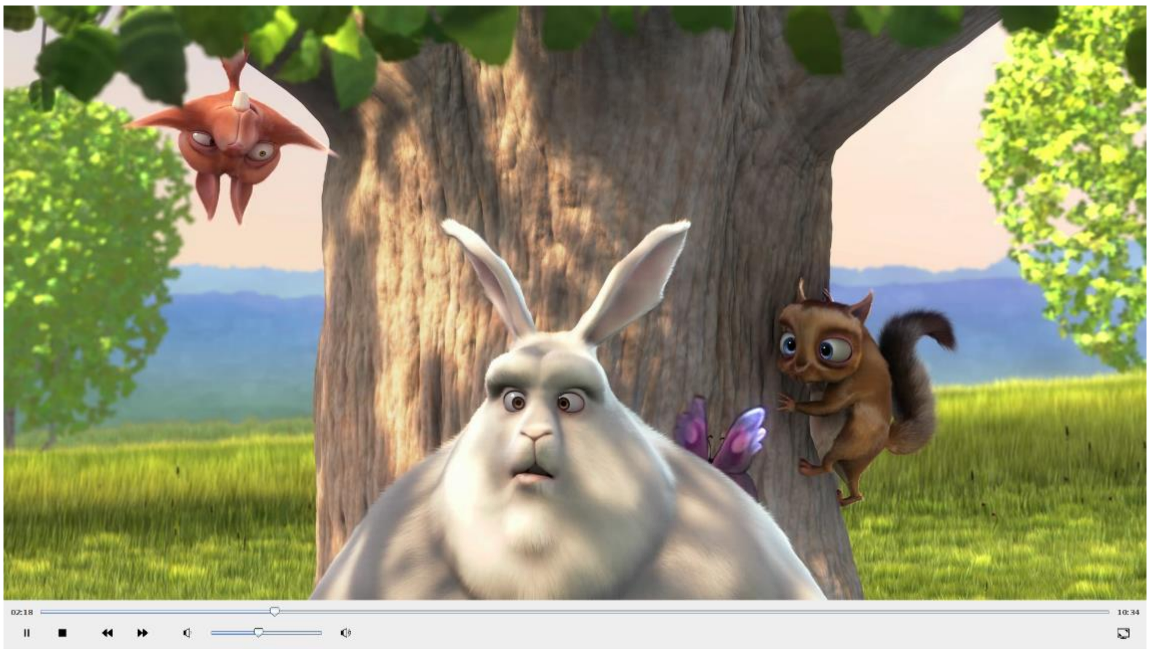


Figure 1 Media Player in fullscreen with playback panel shown

The software supports extraction of audio from media files. Either extracting the entire audio or extracting portions.

Other audio editing options include replacing audio in video files, overlaying a video file with another audio file, change volumes in a media file or add another audio track to a video file.

Video editing include encoding text to a video, adding fade effects or subtitle support.

For text editing and adding fade effects, users have the option to save the current work to a text file if they want to continue working on it at a later date. Users can load work from previous sessions and continue working on their video filters. VAMIX will only load valid save files and ignore invalid files.

VAMIX also provides preview functionality. Before outputting a video file, users can preview the video effects to see what it looks like. Previewing opens a new window which displays the video output. For fade filters, it plays the entire video but for text edits, it only previews the selected time.

For subtitle support, VAMIX can create subtitles or edit existing ones and add subtitles to a video file.

Planning and Analysis

Approach

VAMIX is a complicated software project which involves the integration of several libraries. Hence it was important to identify the target audience and project specifications. Knowing that changes later in the software development cycle is costly, I put emphasis on understanding the software specifications. Furthermore, using software design patterns and minimising code re use was a key goal. Early GUI designs revolved around creating User Interface models.

Targeted Audience

VAMIX is targeted for young adults, particularly for ages 18 – 25 years old. Nowadays, with the popularity of video sharing websites such as Youtube, more people are using video and audio editing software, however, it is mainly used by young adults. Therefore VAMIX is intended for this age group as they will be our primary users. Furthermore, VAMIX concentrates primarily for casual users but provides sophisticated features so advanced users can also use the software. Although VAMIX is concentrated for young adults, this does not mean it cannot be used by other age groups, VAMIX will be made so that its interface is simple enough to be used by everyone.

In particular, several considerations must be considered to appease the target audience:

- VAMIX look and feel should not deviate from standard video and audio editors. Compared to other age groups, software is used much more by young adults hence users will become familiarised with particular look and feel.
- User interface should be simple and easy to understand for casual users.

Choice of Programming Language

VAMIX is programmed in the Java programming language. The reason being that Java is a common and popular programming language with support for many of the required functionalities. In addition, VAMIX uses bash commands as some of the required code, in particular, calling avconv commands must be implemented through the terminal. VAMIX uses avconv and vlcj to function properly. Avconv uses libav, an open source audio and video processing tool to manipulate multimedia formats. Vlcj is an open source project which provides functionality to use the VLC media player using the Java programming language.

Implementation

Design of GUI

Colour Choice

To not overwhelm the users, not many colours were used in the GUI. For a simplistic look, blue was chosen for some of the buttons with the text coloured white. This colour combination was chosen to provide good contrast and make the buttons easy to view without straining the eyes. To accompany the blue and white button, the other buttons are coloured light grey with black coloured text. The decision to choose these colours are that blue and grey complement each other. Furthermore, both button colour choices is easily visible and with the grey background of the software, looks appealing. Another reason for choosing the grey and black coloured buttons is that it still looks good when the button is not enabled. For example, during editing of subtitles, pressing the edit button will disable some of the buttons from being clicked. I noticed during software construction that other colours may look good when buttons are not disabled but when disabled, does not look good.

Layout and Presentation

The layout for VAMIX is to have all the video and audio editing features in a single window. Only the download feature is not in the same window and this design decision was made because download is not linked to the media player therefore it does not need to be together. But for the other features, an emphasis was made to keep all the features in a single window so that users can see where the features are. Having all the features in the same window make finding features easier which is important particularly for casual users. If features are hard to find, this will make the software frustrating to use. Furthermore, the media player is shown because users want to see what they are editing so it is important to keep the functionality features and the media player in the same place. Navigating through the other features are done by pressing the arrow buttons.

All features have a title or label explaining the feature functionality. The text for the button was chosen to be descriptive and concise. Many of the button text contain the same wording. This design decision was made to avoid confusion for users using the software and provide consistency.

Instead of text for media player buttons, appropriate icons were used. The icons used are standard for many media players so using the media player will be intuitive. If users do not understand what the icons stand for, tooltips are included to inform user what its functionality is.

For video editing, all information is displayed through a table as it is a common form of showing information. Tables show the information added by users and can be easily navigated by scrolling. Tables support editing and deleting of information.

System Feedback and Error Handling

System feedback is an important focus when constructing the GUI. It is important to notify users of what is going on in the software and the progress of editing. All messages or progress are shown through a pop up window. The messages are descriptive and inform users the situation. If an error message is shown, it informs the user what went wrong. Progress monitors show the progress of the process and the percentage it has completed. The window for progress monitors show a progress bar along with a cancel button. Users can cancel the process if they press the button.

For Text Editing, text can be customised by specifying the size, colour and font style. To show users what text will be placed into the video, I made the text area be responsive. This means that the text area will show a preview of the text on the text box so users can see what the text would look like.

Developer Documentation

Version Control

In a complex software project which involves many changes in code, version control systems are necessary to document code changes. I used Git and GitHub to manage the changes in the software. Git is a distributed version control system which has advantages compared to other version control systems because I can create a local repository and work at any place. When ready to share the changes, I can push the files to GitHub, an online repository. This was particularly useful as I am able to work at home or at university or on my desktop or laptop.

Comments

To improve maintainability, the code was documented with normal comments as well as Javadoc comments. Commenting helps improve readability and makes the code easier to understand for myself and others.

Code Structure

Java files was put into appropriate named packages. Putting code into packages is important to manage code files during software development, especially when the number of java files become large. Packages are named to describe what the java files are supposed to do. For example, one package is named worker and this contains all the SwingWorker classes.

Java files were also named with a naming convention. To improve the identification of java classes and its purpose, I appended keywords to the end of the java file. This groups common java classes so I can identify which java classes will share common functionality. For example, all classes which extends JFrame was to have Frame at the end of the filename. Hence, in this project, the java files which extends JFrame are:

- VamixFrame
- FullScreenMediaPlayerFrame
- ReadmeFrame.

Code Implementation

The aim was to minimise code re use and use appropriate design patterns to improve the code quality.

The most used design pattern was the Singleton design pattern. The decision to use singleton design pattern is because I believed some of the components only need one instance. The most commonly used area I used singleton design pattern was creating JPanels. Many of the JPanels only require one instance. For example, a JPanel which contains the features for subtitle functionality. For VAMIX, only one instance is needed because having multiple panels for subtitle functionality is not needed. Only one is needed hence the singleton design pattern is used. To improve singleton design pattern, I used lazy instantiation, which creates the Object only when it is needed.

Observer design pattern was used. For example, to accurately listen for changes in the media player, I created an extra java class which implements the MediaPlayerEventListener. This new java class listens for appropriate changes to the media file and then update the media player components such as the JSliders for the time, JLabels for the start time and end time etc.

To minimise code reuse, I used abstract classes. Abstract classes cannot be instantiated themselves and must be extended by other java classes. The advantages of abstract classes is that it can provide code for child classes to use, hence reducing code duplication. When manipulating media files, an avconv command must be made. During avconv commands, output messages are shown on standard output, which include the time completed and other important information. This output message is the same for most of the avconv commands VAMIX will be using. To inform users of the progress of the process, I take the information from the avconv output and get the time completed information. From this information, I find the percentage complete and display the result to the user through a JProgressMonitor. As I display the progress for all the processes, all the processes will need the same code to display the progress hence I moved all the relevant code to the abstract class. Child classes do not need duplicate code.

Implementing the design of the GUI was done through MigLayout, a Java Layout Manager. This is a layout manager which provides more freedom in moving java components which makes designing the GUI easier.

Deterministic finite automata was used to implement the visibility of playback panel during fullscreen.

Development Issues and Solutions

Updating correct rows in JTable for downloads

Issue

Whenever a download is started, a new row is inserted to the table. The row shows the download information and shows the progress of the download and time remaining. By design choice, it was determined that when a download is finished or the download has been cancelled, the row which is representing the download will be removed from the table. The issue is that when a file is being downloaded, it updates a particular row in the table. However, when multiple downloads are occurring, when one download is completed or cancelled, its row is removed from the table hence the other rows position in the table is adjusted. As a result, the JTable may update the information for the wrong row or update a row in the table which does not exist, causing an exception to occur.

Solution

Create a table listener which implements TableModelListener. This child class is notified whenever the table is changed. Whenever the table has deleted a row, update the row positions the JProgressBar is supposed to update. Hence, the rows in the table is showing information for the correct download and will not update an invalid row causing an exception to occur.

Updating the Time Slider without setting time

Issue

This was an issue because I linked the JSlider to set the time of the media player. As a result, moving the JSlider will change the time of the media player. This is desired, however, as the JSlider sets the time for the media player, if the media is playing and is updating the position of the JSlider, it is also setting the media player’s time. This is an issue because while the media is being played, the time is also being set at the same time, causing the media to not play smoothly. Hence it is desired to change the time of the media player when the user moves the JSlider manually but when the JSlider is updating from the media player, do not set the time.

Solution

Every JSlider has a model. During a change in the JSlider position, the model fires a change and notifies all observers of the change. The observer receives notification of this change and in this example, will set the time for the media player. Instead of changing the media player time everytime, I created a new model, which extends DefaultBoundedRangeModel. This child class contains an additional Boolean variable which keeps track when it should be active or not. When the media file is being played, it will call the JSlider to update position. Before updating the position, set the Boolean variable to false so the model should not be active. When the observers are notified, it checks the Boolean variable and decides if it should set the time or not.

Fullscreen support for Media Player

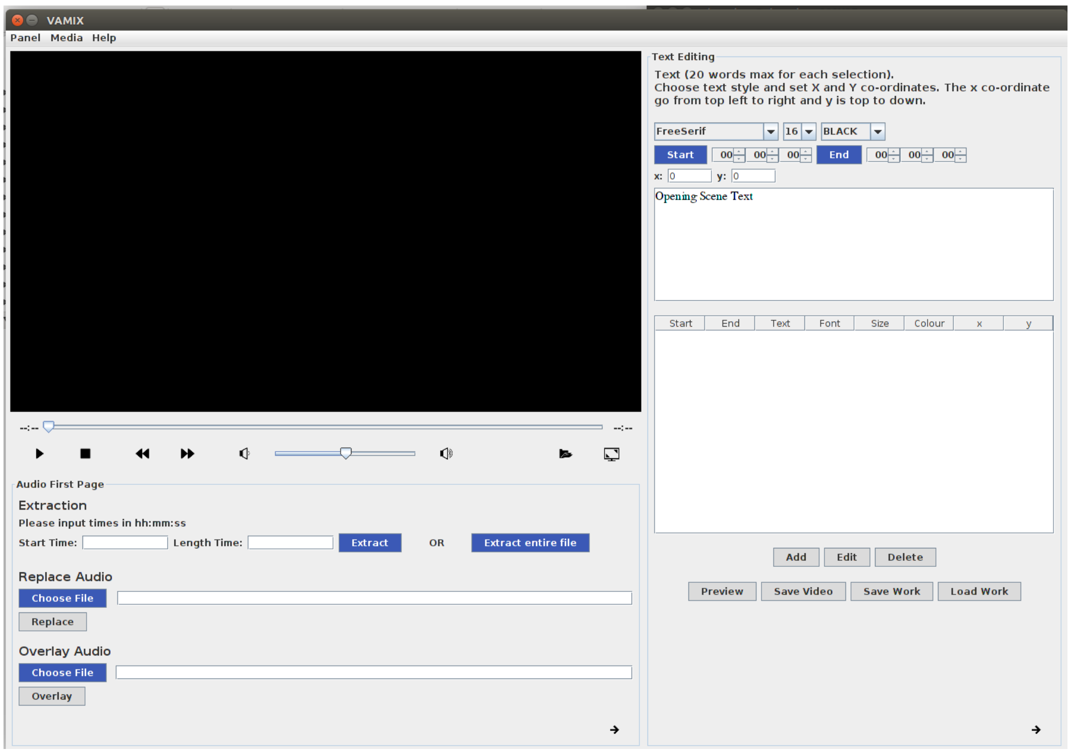
Issue

How to have get the media to be played in fullscreen while having the ability to show the playback panel.

Solution

The biggest issue was finding the right approach. Firstly, I tried to implement an overlay so that the media player is playing in the background and the front panel is transparent with the playback panel at the bottom. This approach was unsuccessful. My solution was to use JLayeredPane. JLayeredPane allows components to be placed on top of each other. The positions for each of the components was determined manually by getting the size of the monitor resolution and setting the position. The playback panel is not visible but with mouse movement, it shows for panel and when mouse movement stops, the playback panel is not visible after 3 seconds.

To implement the visibility of the playback panel without lagging the media player, a deterministic finite automata was used. The idea is to keep track of the current state of the mouse movement. If the mouse is moving, it should remain at the initial state and not start the counter. When the mouse has stopped moving, it will start the timer and count for 3 seconds. When the timer reaches 3 seconds, the playback panel's visibility is set to false.



Text (20 words max for each selection).
Choose text style and set X and Y co-ordinates. The x co-ordinate go from top left to right and y is top to down.

Please input times in hh:mm:ss

Opening Scene Text

Figure 2 VAMIX look during start up

Evaluation and Testing

Testing was an ongoing process during the software development lifecycle. For this software, I believed creating test scripts to automate tests was unnecessary and would require so much time that it would not be beneficial so it was decided that testing would be done manually.

Testing involves checking the code manually to identify faults. Testing inputs for functionality is important. I modelled the possible inputs into equivalence classes. Test both correct and incorrect inputs and make sure the expected outputs are returned. Finding and correcting faults is important for improving the quality of the software. Robustness for errors is important for any software application and it is ideal to identify errors and inform users the error with a message. If the software is not robust, the software may freeze and become unresponsive.

While testing for code quality is achievable manually, it is difficult for GUI. Evaluation on GUI is difficult because what I may find intuitive, others may not. Clients may have different opinions on the look of the GUI so to improve evaluation of GUI, I asked others for feedback.

During the development of VAMIX, I asked some of my friends, same age group and not doing Software Engineering for advice on GUI. Listening to their feedback, it was obvious that some features were not intuitive to others. The main advice was choice of error messages. In particular, "Please parse Media". For less experienced users with computers, they will not understand the meaning of the above statement so it was advised to make message more user friendly.

Peer Evaluation

During the software development for VAMIX, peer evaluations were completed. Some of the main feedbacks given were:

- Reduce code duplication by creating abstract classes or interfaces.
- Create jar file to make code initialisation easier.
- Use better package naming. Package naming should be for functionality rather than grouping types.
- Text Edits should be allowed for any time of the video, not just for opening and closing scenes.
- Change the colour of the buttons.

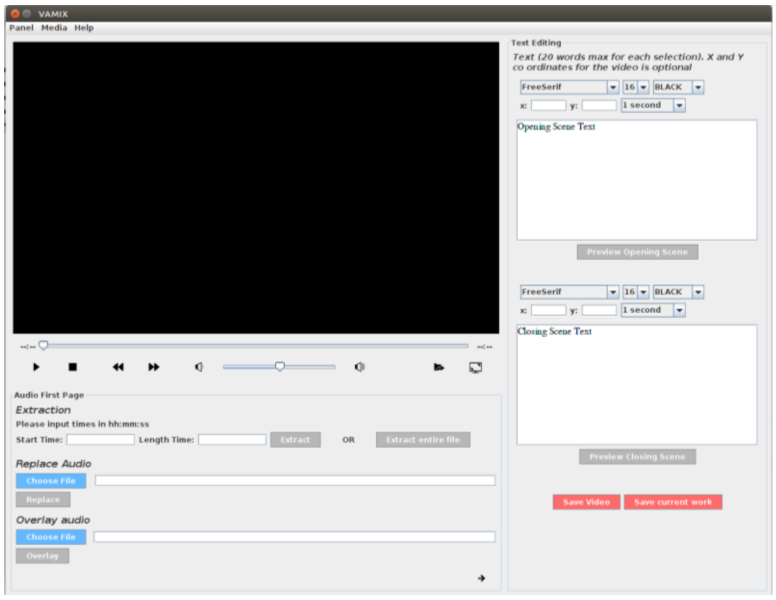


Figure 3 VAMIX look during beta submission

My peer's opinion are useful because they fit my targeted audience. The feedback will help me improve the GUI for the final product of VAMIX. From their feedback, I changed several features.

The most important being able to add text edits anywhere in the video. This was my top priority because users should not be limited to editing only opening and closing scenes. Furthermore, the length of the text edits were limited because the length was between 1 to 10 seconds. I changed VAMIX so that the final product will allow users to edit at any time of the video.

Secondly, code duplication was reduced by creating abstract classes for the SwingWorker classes. Many of the swingworker classes used duplicate code so to improve the quality of the code, abstract classes was made to reduce code repetition. Furthermore, I identified and fixed other areas where code duplication was apparent.

For the final submission, I made sure that jar file was available. During the beta submission, I had difficulty creating a jar file so only the source files were submitted for peer reviews. This is a problem because users are forced to run the software through an IDE such as eclipse. This is a problem for many people, particularly casual users who may not do programming hence do not have eclipse installed.

The button colour was changed to a different shade of blue and different shade of grey. I kept the same colour choice as I believed blue and grey complement each other nicely. However, from the feedback, it was said for colours like grey, it was too similar to the colour when JButton is disabled so users may not know if the button can be pressed or not.

The only change I did not implement for the final product was changing the package naming convention. I chose to stick with my package naming convention because for a software project of this size, I believe it is simpler to manage the packages how I did it during the beta submission. The suggestion was to name packages based on functionality hence all functionality should be in one package, everything related to audio should be packaged in another folder. Ultimately, I thought this may be too complex and will create a deeper package hierarchy, which would make it more complicated. For example, if I was to put everything related to Audio in a single package, the package may include Java files which are JPanels, Swingworker classes etc. This will create more confusion and the necessary solution is to create another folder which sorts all audio JPanel classes in one folder and all audio Swingworker classes in another folder etc. This will create a deeper package hierarchy and for this software project, I believed was too complex.

The comparison of the VAMIX GUI can be seen from the changes in Figure 2 and Figure 3.

Conclusion

VAMIX is a video and audio editing software which runs on the Linux operating system, providing basic and advanced multi media manipulation. The objective for this software project was to create a high quality, user friendly software for our targeted audience by applying knowledge learnt from class. The target audience was determined to be young adults and objectives were met by evaluating and testing the software throughout its development life cycle and keeping good documentation of code.

Future Improvements

For the future, VAMIX can be updated to have more features and support more file formats. Previewing can be shown on the same window. The GUI can be further enhanced through user feedback to make a more user friendly software.