Video Player Using Python

Arjit Singh
21bec010@nirmauni.ac.in

Arya Patel
21bec013@nirmauni.ac.in

Sujal Bhojani
21bec016@nirmauni.ac.in

Abstract: **Video playback applications are widely used for various purposes, including entertainment, education, and professional presentations. This paper introduces a versatile video player application implemented in Python using the Kivy framework. Kivy is an open-source Python library for developing multitouch applications, making it an excellent choice for creating interactive and cross-platform video player interfaces.**

1. Introduction:

Video playback applications are essential for providing immersive and interesting experiences in the constantly changing world of digital multimedia consumption. With the growing need for flexible and adaptable video players, developers are looking for strong frameworks that make it easier to create apps with plenty of features. The work presents a Python programme that uses the Kivy framework to create a sophisticated video player.

Python has been more and more popular as the language of choice for developing applications in recent years because of its extensive library ecosystem, readability, and versatility. An open-source Python module called Kivy expands Python's applicability to multitouch applications. Kivy is a cross-platform framework that is perfect for creating interactive user interfaces for video playback because of its flexibility and user-friendliness.

Video players are useful for more than just watching films or TV shows; they are essential for professional presentations, learning, and interactive information delivery. The video player application that is being shown acknowledges the many needs of its users and provides a comprehensive solution that prioritises usability, extensibility, and cross-platform compatibility.

Objectives:

The primary objectives of this paper are to:

- Introduce a Python-based video player application utilizing the Kivy framework.
- Showcase the application's key features, including file selection, video playback controls, and fullscreen toggling.
- Demonstrate the extensibility of the application, encouraging developers to build upon the provided foundation.
- Highlight the cross-platform nature of the Kivy framework, enabling the deployment of the video player across diverse operating systems.

We can learn more about the possibilities of Python and Kivy for developing dynamic multimedia apps, encouraging creativity, and expanding the possibilities for video playback through this study.

2. Problem Statement:

The existing video playback programmes have numerous limitations in their

functionality, such as a lack of extensibility, dependence on specific platforms, poor user interface design, restricted interaction capabilities, difficulties in ensuring accessibility, and a fragmented ecosystem. Conventional video players frequently have a predetermined range of capabilities that offer limited opportunities for customization, hence restricting the introduction of new features or improvements. Platform reliance is a significant concern due to the requirement of video players to function seamlessly across a wide range of operating systems. The immersive potential of video playback experiences is further constrained by deficiencies in user interface design, restricted interaction capabilities, and difficulties in achieving accessibility. The objective of the proposed solution is to offer a video player that is expandable, compatible with multiple platforms, and user-friendly. This contribution seeks to enhance the quality of video playback experiences and promote a more inclusive multimedia environment within the Python development community..

3. Implementation:

The main goal of this project involves the development of a Python-based video player utilising the Kivy framework. The primary objective is to design an adaptable and simple application that effectively tackles the observed challenges experienced in current video players. The key components and characteristics of the implementation includes

- **File Chooser Widget:**

The application includes a FileChooserListView widget, which facilitates users in navigating their file system with smoothness and efficiency. The default directory is designated as 'C:\', offering users a point of origin for navigating through files.

- **Video Player Widget:**

The central component of the application is the VideoPlayer widget, which is equipped with customizable settings that enable the video to be stretched to fit its dimensions. The provided widget offers a range of conventional video controls, such as play, pause, stop, and volume adjustment..

- **Fullscreen Toggle:**

The application includes a feature that allows users to switch between fullscreen and regular mode, thereby improving the quality of their visual experience. The 'F' key can be utilised by users to seamlessly transition between fullscreen and normal modes, hence offering adaptability in accommodating many display scenarios.

- **Additional Controls:**

The video player responds to various key presses to provide a comprehensive user experience.

Tab Key (KeyCode: 9): Exiting fullscreen mode.

'P' Key (KeyCode: 112): Toggling play/pause functionality.

'M' Key (KeyCode: 109): Muting/unmuting audio with dynamic volume adjustments.

'S' Key (KeyCode: 115): Stopping video playback.

- **Event Handling:**

The programme utilises event handling mechanisms in order to effectively respond to user inputs. The on_selection function is associated with the file chooser's selection event, guaranteeing that chosen video files are loaded into the player. Furthermore, the on_press function is responsible for

managing the button press event that enables the switching of fullscreen mode.

- **Key Down Event Handling:**

The on_key_down function is associated with the 'on_key_down' event of the Window, enabling the implementation of prompt handling of keystrokes. This feature enhances the usability of the system, allowing users to easily perform tasks such as toggling fullscreen mode, controlling playback (play/pause), adjusting the volume, and pausing the video.

- **Extensibility:**

The programme has been intentionally intended to prioritise extensibility, hence providing a development environment that encourages developers to enhance and expand upon the existing foundation. The introduction of extra features is facilitated by the modular construction, rendering it a suitable foundation for those aiming to tailor the video player in accordance with their personal needs.

- **Cross-Platform Compatibility:**

The utilisation of the Kivy framework facilitates the achievement of cross-platform compatibility, hence allowing for the deployment of the video player application on a wide range of operating systems. This facilitates a uniform user experience irrespective of the underlying platform.

- **User Interface Design:**

The layout has been designed utilising a vertical BoxLayout to effectively fit the file chooser, video player, and any potential future expansions. The design of the user interface aims to achieve an optimal balance of simplicity and functionality, hence facilitating use for a diverse range of users.

4. Conclusion:

The Python-based video player application, built using the Kivy framework, is a significant step towards addressing the challenges in existing video playback solutions. Its versatility and extensibility allow developers to tailor the application to specific use cases, such as educational, entertainment, or professional presentations. The application exhibits robust cross-platform compatibility, ensuring a consistent user experience across various operating systems. It also incorporates intuitive keyboard shortcuts, responsive touch interactions, dynamic volume control, fullscreen toggling, and customizable key bindings for a more immersive video playback experience. The user interface design balances simplicity with functionality, promoting ease of use for a wide audience. The application contributes to the Python multimedia ecosystem by providing a versatile solution for developers to build upon, encouraging collaboration and innovation.

5. References

Leong, D., & Bahl, V. (2015). *Creating a Python-based Portable Media Player with Enhanced Parental.* Singapore: The Python Papers 11: 4.

Virbel, M. (2014, October 21). kivy. *kivy*. Github.

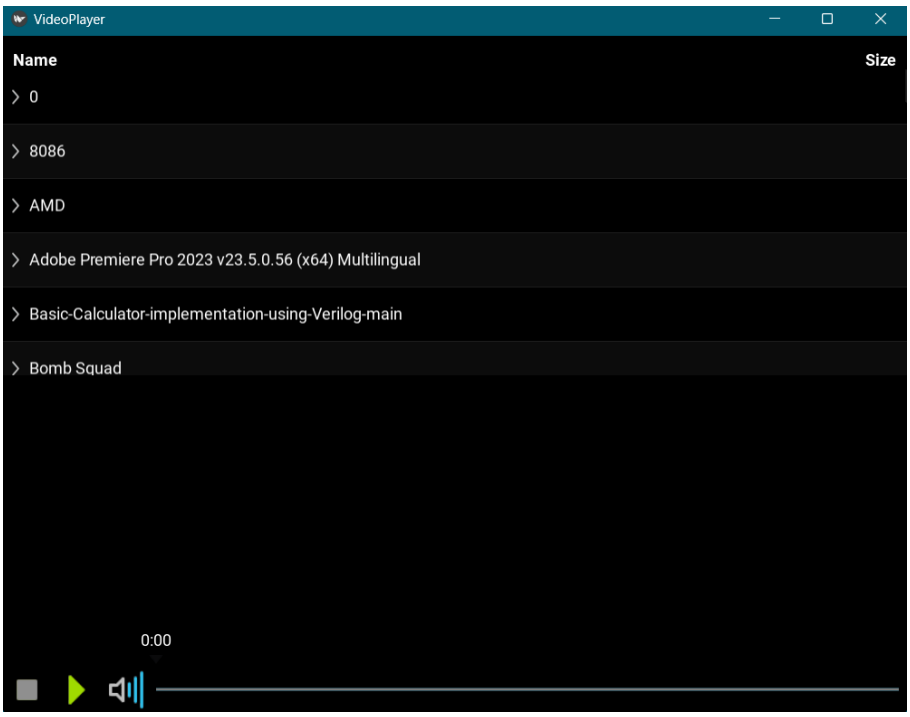
Virbel, M., Pettier, G., Arora, A., Taylor, A., Einhorn, M., Larkin, R., & Galimberti, M. (2014, October 21). *About Us: Kivy*. Retrieved from Kivy: https://kivy.org/about.html

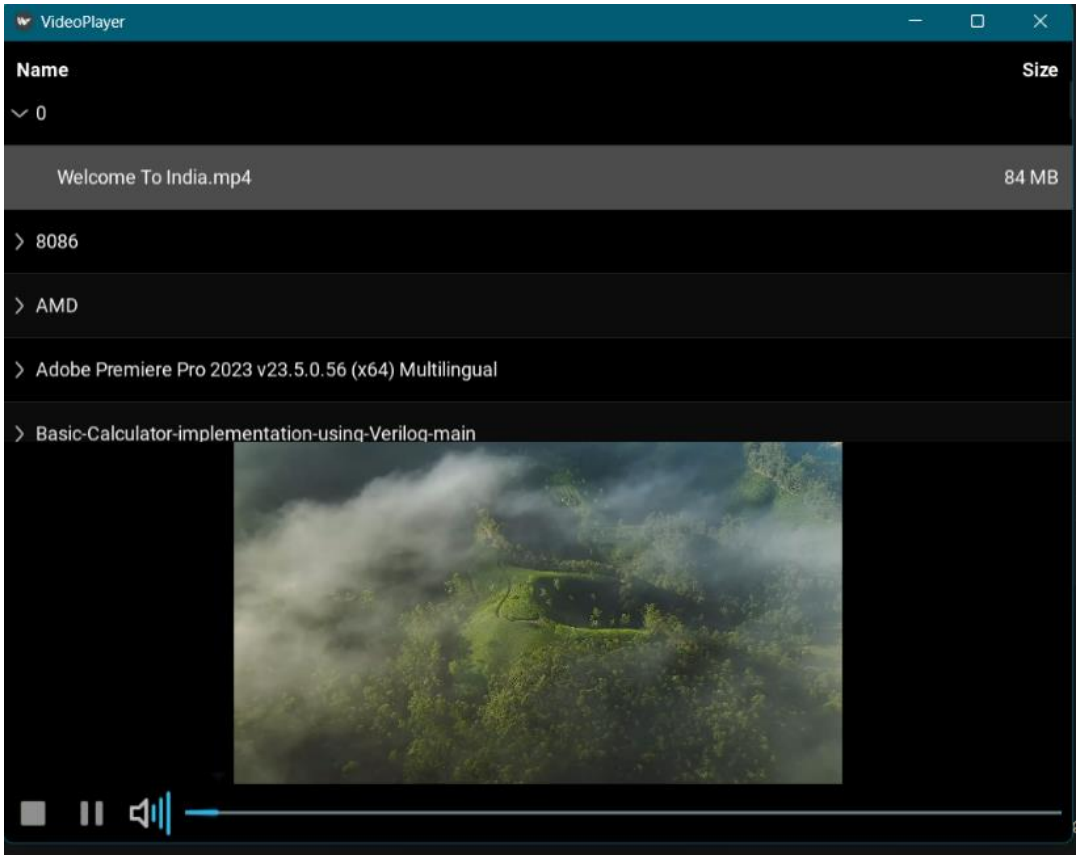
CODE:

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.videoplayer import VideoPlayer
from kivy.uix.filechooser import FileChooserListView
from kivy.core.window import Window

class VideoPlayerApp(App):
    def build(self):
        # Create a vertical layout
        layout = BoxLayout(orientation='vertical')

        # Create a file chooser widget and set the initial path to 'C:\\'
        filechooser = FileChooserListView(path='C:\\')

        # Create a video player widget and allow it to stretch the video to fill
its size
        video = VideoPlayer(options={'allow_stretch': True})

        # Add the file chooser and video player to the layout
        layout.add_widget(filechooser)
        layout.add_widget(video)

        # Define a function that will be called when the selection of the file
chooser changes
        def on_selection(*args):
            # Check if a file is selected
            if filechooser.selection:
                # Set the source of the video player to the selected file
                video.source = filechooser.selection[0]
                # Start playing the video
                video.state = 'play'
            else:
                print("No file selected.")

        # Bind the on_selection function to the selection event of the file
chooser
        filechooser.bind(selection=on_selection)
        # Define different functions that will be called when specific keys are
pressed
        def on_key_down(window, key, *args):
            if key == 102:  # 102 is the keycode for 'F'
                video.fullscreen = not video.fullscreen
            elif key == 9:  # 9 is the keycode for 'Tab'
                video.fullscreen = False
            elif key == 112:  # 112 is the keycode for 'P'
```

```
                if video.state == 'play':
                    video.state = 'pause'
                else:
                    video.state = 'play'
            elif key == 109:  # 109 is the keycode for 'M'
                if video.volume > 0:
                    video.volume = 0
                else:
                    video.volume = 1
            elif key == 115:  # 115 is the keycode for 'S'
                video.state = 'stop'

        # Bind the on_key_down function to the 'on_key_down' event of Window
        Window.bind(on_key_down=on_key_down)

        # Return the layout
        return layout

    def on_stop(self):
        # Release the video player resources when the application is closed
        self.root.children[1].state = 'stop'

if __name__ == '__main__':
    # Create an instance of VideoPlayerApp and start it
    VideoPlayerApp().run()
    video = VideoPlayer(options={'allow_stretch': True, 'audio': 'avbin',
'audio_buffer': 512})
```

Default View:





Fullscreen Toggle: