C# WPF Video Player

Zachary Rose, Logan Walsh

Abstract

Our project is a custom video player as a C# WPF application. It will allow you to browse to a file, load up a video, and control the playback as needed. AVI, MWV, and MP4 will be supported file types. Those who want more control over their video playback beyond what Windows comes with will find this application useful.

1. Introduction

For this project, we plan to make a media player program that offers some features that are unavailable in Windows Media Player. It will include a file browser to select a media file, and it will include the controls for playback such as: play/pause, fast forward/slow down, and skip to different parts of the video, as well as volume controls.

This project is intended for those who want some finer control over the playback of their video files. Windows Media Player does have the basic features for controlling playback of media, but it leaves much to be desired in customization and more niche functionality.

Windows Media Player lacks features such as moving forward and backwards a frame at a time, enlarging the video at a key press, and muting the video with a key press. In fact, Windows Media Player lacks keyboard controls entirely. VLC Media Player has these features and more, but it can sometimes feel daunting and over bloated for an average use case. This media player will fit comfortably in the middle between the most casual of users and super users.

1.1. Background

Some useful terminology:

- media: in this case, a file containing video and/or sound
- media player: a program that can playback different media, and it usually allows for some control over different aspects of the playback
- seeker: something that enables a user to skip to an arbitrary part of a media's duration with a single click, often in the form of a horizontal bar
- playback: media is in playback when it is currently being shown to the user
- file browser: a user interface to make selecting a file from the system convenient
- Windows Media Player: a media player that comes with Windows OS by default. Very bare-bones
- VLC Media Player: A popular media player for power users. Comes with a huge amount of customization options.
- Fullscreen: refers to when the video is scaled up to cover an entire monitor
- Aspect Ratio: refers to the ratio between the width and height in pixels

We decided to make a media player because we thought it would be interesting to make a tool that we might actually use, rather than some project for the sake of displaying some unusual programming concept. It will be good practice towards developing a user interface for a user-oriented application.

1.2. Impacts

Our hope is that this project will become a useful little tool for those who wish to have a light-weight and customizable media player. For those who want a middle ground between the simplicity of Windows Media Player and the overwhelming power of VLC.

1.3. Challenges

Before we started this project, we predicted that the hardest thing would be designing an interface to dynamically change keybindings. It would need a file to store them, the ability to parse and edit said file, and a graphical interface with which the user will change the bindings from within the app.

However, the biggest challenge ended up being designing the UI and keeping it balanced well, even through changing the size of the window. While WPF provided an easy way to quickly prototype a user interface, it was difficult to design something that was appealing.

Another big challenge was getting the reverse playback to work correctly. The MediaElement in the standard library doesn't come with negative playback speeds, so it had to be implemented by hand. The end product isn't quite satisfactory, and we wish we could have come up with a more robust solution.

2. Scope

The project will be done once the program runs without any issues and has enough features to differentiate it from Windows Media Player. In other words, the ability to control the video playback with the keyboard and to create custom keybindings.

We think that this is a worthwhile project because it has practical use, and it will allow us to make further use of the WPF framework. It will require an attractive interface and different form elements working together behind the scenes.

Basic Goals:

- Can load a video file to prepare for playback.
- Can control playback through both mouse and keyboard.
- Supports AVI, MKV, and MP4 filetypes.

Stretch Goals:

- Permanent customization for key bindings, with the ability to reset to defaults.
- Combining multiple files into a playlist with associated skip forward/backwards buttons.
- WPF window can be resized, with all the elements keeping the same size ratio.

We were able to meet all these goals besides creating a playlist. The interface WPF supplies for selecting multiple files at once was undesirable, and due to time constraints we couldn't find or develop an alternative.

2.1. Requirements

The functional requirements for a media player are straight forward for anyone who has used one before. We considered what we want to be able to do when watching a video, as well as what is absolutely necessary to do so. Loading a file is the absolute base requirement, but we've come to expect things like pausing and fast forwarding.

2.1.1. Functional.

- User can play video files
- User can use video timeline to skip to desired part of video directly
- Application supports fullscreen playback
- User can pause and unpause the playback at will
- Application supports different playback speeds, between -2x and 2x
- User can change the volume as desired

2.1.2. Non-Functional.

- Application must be able to edit and reload keybindings between different sessions.
- Application must be able to load and play several videos sequentially without performance loss.

2.2. Use Cases

Use cases quick reference:

Use Case #:	Short description
1	Loading up a video
2	Editing keybindings
3	Controlling playback

Use Case Number: 1
Use Case Name: Loading up a video
Description: The user wants to load a video from their hard drive to prepare for playback.

1) User opens up file browsing menu by left-clicking "File," followed by "Load File" (See Figure 1).
2) User navigates to desired video and selects it.

Termination Outcome: Video is loaded and begins playback.

Note: File selected doesn't exist.

1) User selects a file that doesn't exist, doesn't have permissions to play, or isn't a valid video format.

Termination Outcome: Application requests a valid file and allows for re-entry.

Use Case Number: 2

Use Case Name: Editing keybindings

Description: The user wants to change the default keyboard bindings for different aspects of playback control and expects these changes to persist through different sessions of using the application.

1) User opens up the options by left-clicking the “Options” tab with the mouse See 3.
2) User selects “Set Hotkeys” by left-clicking with the mouse.
3) User clicks the button corresponding to the binding they want to change, and presses the desired keyboard key. See /refKeybindingsInterface.

Termination Outcome: New keybindings are set for any available functionality, and the new bindings will be saved to a file for automatic loading upon opening the application, should the user choose to save them.

Use Case Number: 3

Use Case Name: Controlling playback

Description: During playback, the user wants to control different aspects of the playback. See Figure 2 for the buttons available to left-click.

1) User pauses and unpauses playback by left-clicking the pause button.
2) User increases and decreases playback speed using the corresponding buttons.
3) Alternative: User enters a valid playback speed directly into the text box (between -2 and 2).
4) User increases and decreases volume by dragging the volume slider using the mouse.
5) User enters fullscreen mode by left-clicking the fullscreen button.

Termination Outcome: Video is in playback in the way desired by the user.

Alternative: User wishes to use keybindings to control video playback. See Figure 4.

1) User pauses and unpauses playback by pressing the bound button on the keyboard.
2) User increases and decreases playback speed by pressing the bound buttons on the keyboard.
3) User increases and decreases volume by pressing the bound buttons on the keyboard.
4) User enters fullscreen mode by pressing the bound button on the keyboard.

Termination Outcome: Video is in playback in the way desired by the user.

2.3. Interface

Upon opening the media player, the user sees the player awaiting input (See 2). From here they may wish to load a media file, which they do through the Load File interface (See 1). This opens the Windows file search utility and makes selecting a media file simple.

Next, the user may wish to choose from several available customization options under the “Options” menu (See 3). There are four themes to choose from, which change the color and overall appearance of the media player window. The user can also toggle between stretching the media to fit the window and the media maintaining its aspect ratio with the “Stretch Aspect Ratio” option. Finally, the user can choose to edit their keybindings under “Set Hotkeys”, which opens a new window.

In the Keybindings window (See 4), the user can set their desired keyboard buttons to control the media player in addition to using the mouse. See Use Case 2 for a more detailed explanation.

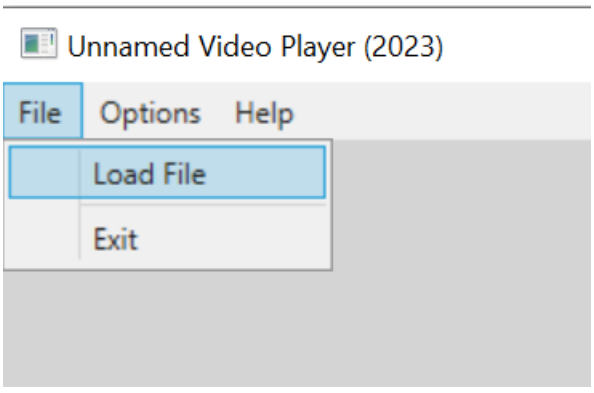

Figure 1. Sample of interface for selecting video file

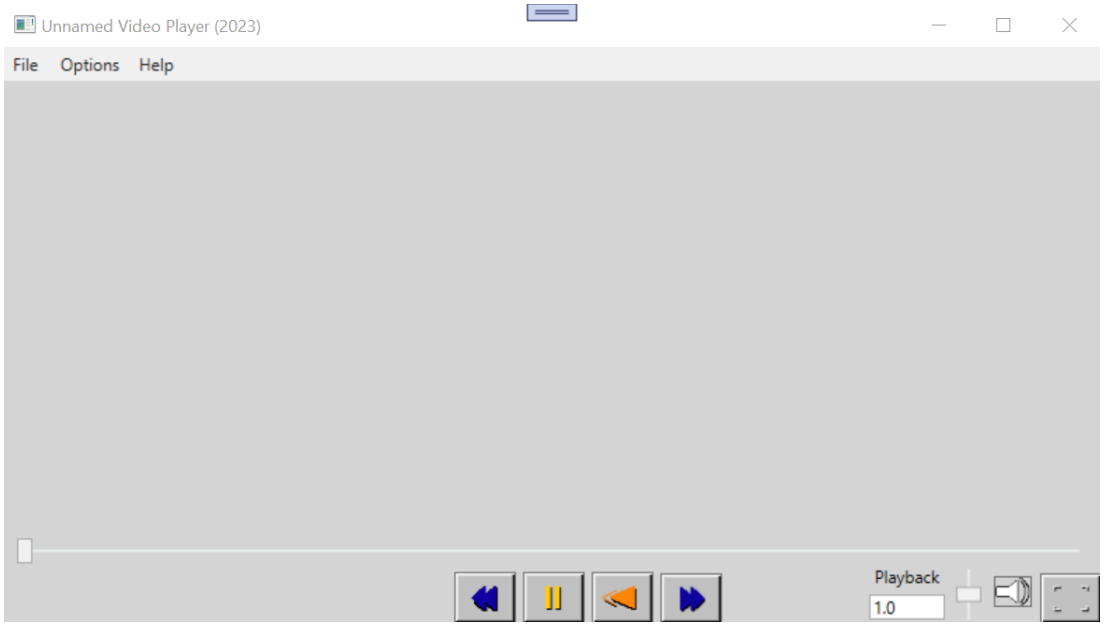

Figure 2. Sample of interface before a video is in playback

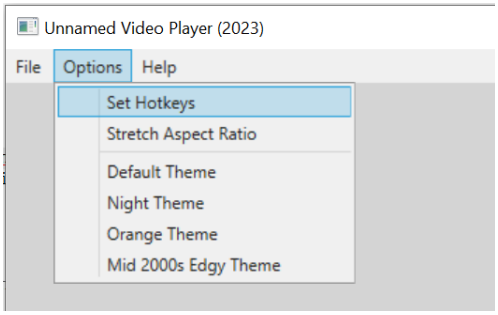

Figure 3. Several options are available in the options menu

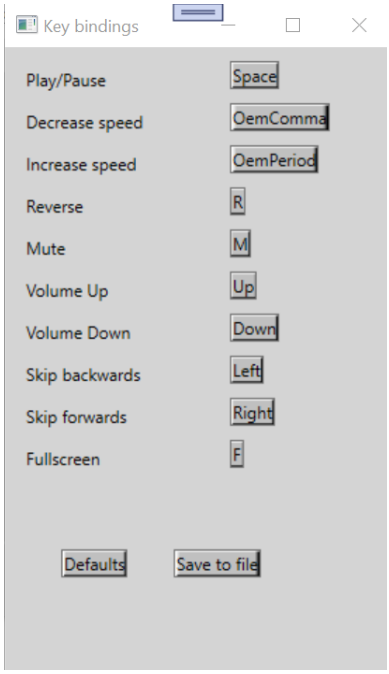

Figure 4. Window for changing keybindings

3. Project Timeline

This timeline has the dates for every major checkpoint during the development of the media player. The colored bars represent the phases of waterfall development (sans maintenance), and every group work (GW) deliverable is marked as well. See Figure 5.

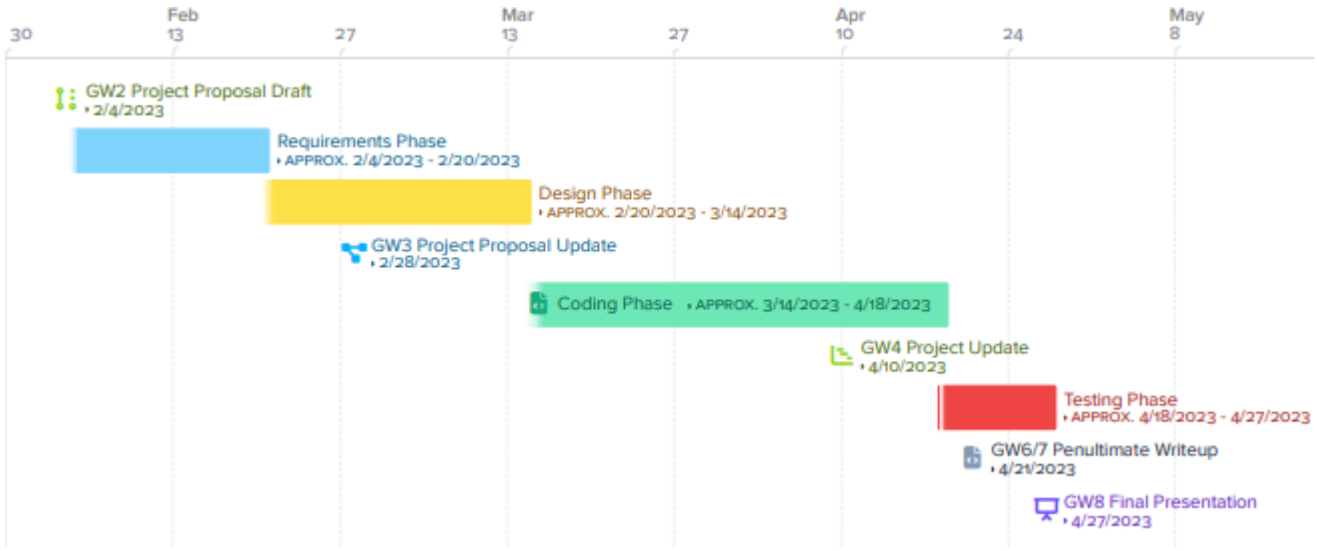

Figure 5. Proposed project timeline

4. Project Structure

The Media Player works by updating a Media Element when the user gives input. It is implemented as a MainWindow class containing a MediaElement object, as well as several helping classes to control the appearance.

Our MainWindow class contains several important features that work together to update the Media Element. A seeker is updated on a timer to remain updated with video playback, and in the same way the video playback is updated when the seeker is moved. A timer is also used to implement playing in reverse; every tick of the timer pushes the media backwards by an amount with respect to the video playback speed.

A Factory class is used to change the appearance of the application during runtime. This was chosen in order to simplify adding more themes by keeping them unbound from the main class.

The Keybindings were delegated to their own class, and an associated KeybindingsWindow class to provide the interface with which to change them.

4.1. UML Outline

The MainWindow class holds the WPF form. This form holds the button controls, and most importantly it holds the MediaElement object which is used to enable playback. Two DispatcherTimers are also used to keep the seeker updated with the video playback. These structures are located at the top of Figure 6.

The ThemeFactory is an abstract factory interface used to change the theme of the media player. When choosing a theme a new factory is constructed, and it is used to change the appearance of certain elements like background colors and button design. These structures are located at the bottom of Figure 6.

The Button Decorator is a wrapper for the Windows.Controls.Button with added state and behavior in relation to button appearance. This structure is located at the top of Figure 6.

The Keybinder class is used to dynamically change keybindings during runtime. The MainWindow has a Keybinder, which it used to correctly map the desired keyboard key to a command. The KeybindingsWindow is a separate window that has access to the MainWindows Keybinder object, allowing it to edit it during runtime or write the bindings to a text file.

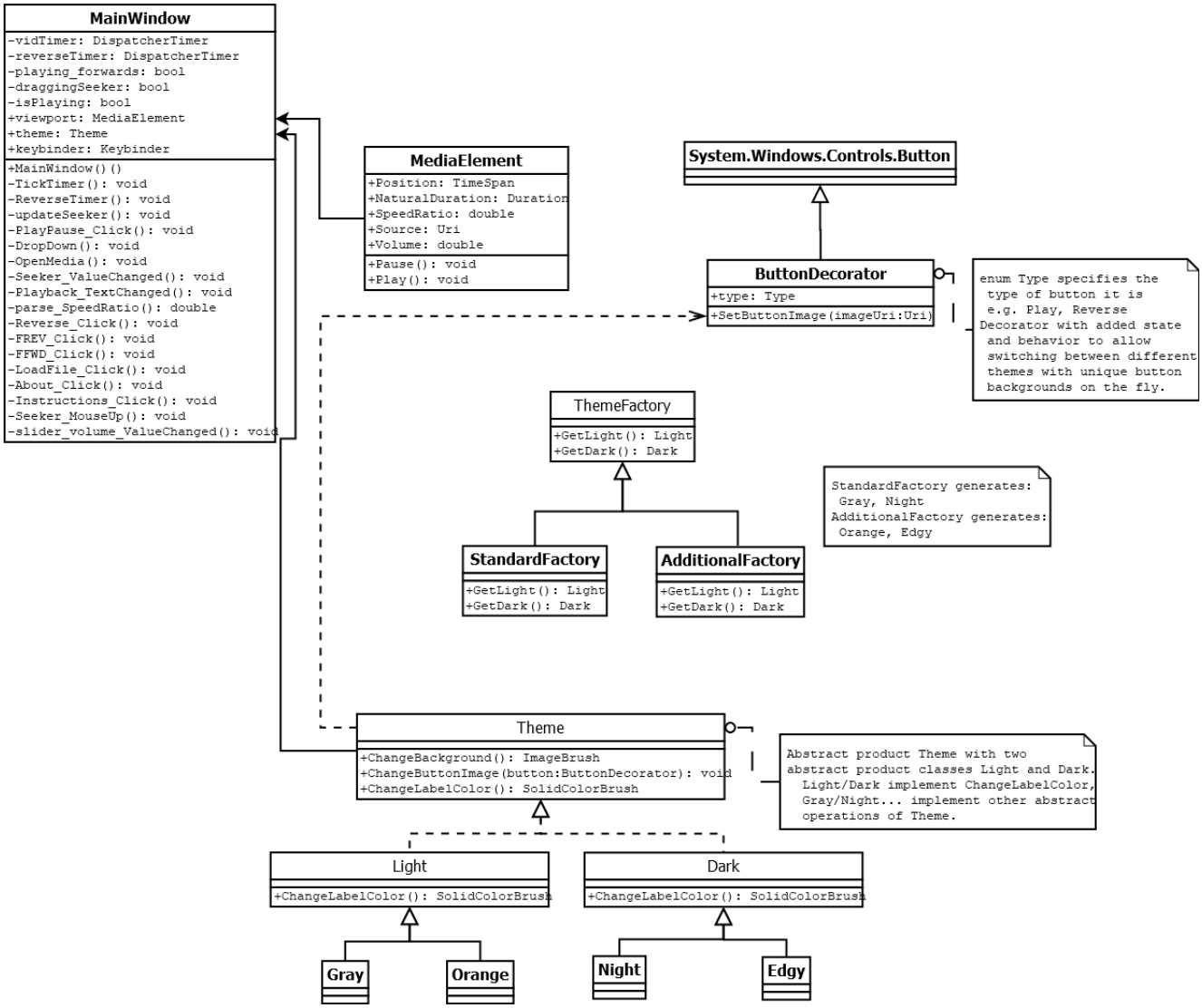

Figure 6. UML of class structure

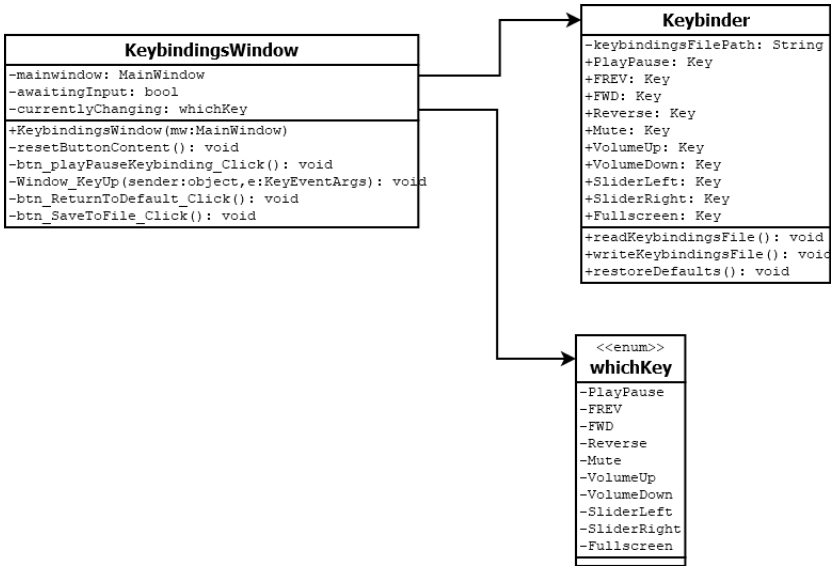

Figure 7. UML Keybinding class structures

4.2. Design Patterns Used

We used an abstract factory to provide different look-feel themes for the user to choose from (See Figure 6). The abstract factory ThemeFactory has two concrete factories: StandardFactory and AdditionalFactory. These concrete factories generate a Light or Dark. Theme is an abstract product which is subclassed by the more specific abstract products Light and Dark. The concrete products Gray, Orange, Night, and Edgy are capable of changing the color of the background and the image/color of the buttons.

We used a decorator for the Windows.Control.Button to add extra behavior and states to our form elements. The ButtonDecorator has a Type which specifies which button appearance it should have. For instance, a play button vs. a pause button, or a fast forward button. It also has a function SetButtonImage, which is provided a Uri to the correct display image by the concrete product (see the abstract factory above) that calls it. Only the button knows what kind of button it is, so this information is used by the abstract factory to provide the correct file.

5. Results

Over the course of working on this project, we've learned many different things. We successfully used waterfall development to plan our work, and we learned the difficulties of developing a larger, longer term project.

We developed a media player that competently plays video and audio in an unfamiliar environment. We were able to implement design patterns to help organize class structure and promote decoupling.

We weren't able to meet all of our goals. We haven't implemented playing a queue of videos, and we hoped to have far more customization, such as changing the amount of time skipped when jumping forwards and backwards (See 4).

5.1. Future Work

This project didn't become anything impressive. We didn't find the time to implement all that we wanted to, so the final project lacks what usefulness we hoped it would have. We don't plan to continue development on this project any further, but we'll take what we learned during development and apply it in the future. If we were to develop something similar in the future, we would use WPF again. It provided built in features that greatly simplified making the user interface, and the built in debugging tools in Visual Studio was useful.