Specification Document - PancakeMediaPlayer

LIFAPCD - Spring 2024

Alyce THÉOBALD, Killian POULETTE, Fabien JOHNSON-DE-BISSCHOP

Project rapporteur: Nicolas LOUVET

Université Claude Bernard Lyon 1
Licence 2 - Portail Informatique

Contents

1 Project Overview **2**

2 Project Description **2**

3 Constraints **2**
3.1 Time constraints 2
3.2 Development constraints 2

4 Specifications **3**
4.1 Core 3
4.2 User Interface 4
4.3 Networking 6

5 Annexes **7**

1 Project Overview

The goal of this project is to create a homebrewed low-level audio player for Unix/Linux System. This project's idea came up because one of us always wanted to make an audio player. The PancakeMediaPlayer will be a good way for us all to acquire even more advanced skills in C/C++ programming.

This project is directly linked to our Curiculum as part of LIFAPCD classes.

2 Project Description

PancakeMediaPlayer is a simple audio player for Unix system. This application is able to parse WAV files using a custom-made algorithm and can load other audio file formats using a FOSS library.

This media player can also be used as a client to receive music from a remote machine through a custom-made broadcast system.

3 Constraints

3.1 Time constraints

- Development timeline: 8 weeks
- Deadline: 16 April 2024

3.2 Development constraints

- The code must include at least one advanced algorithm.
- The code must respect the View-Controller model.

4 Specifications

4.1 Core

ID	Feature	Deliverable	Completion Conditions
C1	Read sound data from a file	A class: Loader	When the Loader is able to read metadata and raw sound data.
C1.1.1	A magic number identifier system. The goal of this system is to read the first bytes of the file and identify its type.	Protected method: loadMetadata	When the method identify correctly the type.
C1.1.2	Retrieve the file length and store it.	Same	When the method identify the length correctly.
C1.2	Retrieve the correct pulse code modulations and store them into a Dynamic Array. Also reads the Metadata.	Public method: load. It will use the loadMetadata method.	When the method loads the data correctly.
C2	Format sound data into a playable format for ALSA and apply sound filters.	A class: Formatter	When the Formatter is able to correctly format the data.
C2.1	An algorithm capable of correctly filtering the unnecessary frequencies or artifacts in the sound.	A public method: apply_filter	When the filtered sound is audible and sounds good.
C2.2	An algorithm to perfectly format the audio data into PCM (Pulse Code Modulation).	A public method: format_data	When the sound is playable by ALSA.
C3	A simple API to read and format the sound data from a file.	An aggregation class: Parser	When the Loader and the Formatter are ready for use.
C4	A player that uses the ALSA API to play sounds from a file.	A class: Music.	When the Music can play, pause, unpause, and jump to a specific timecode.
C4.1	An algorithm to jump correctly to a specific timecode in the PCM array.	A public method: jump	When it correctly jumps.

4.2 User Interface

ID	Feature	Deliverable	Completion Conditions
U1	Create a superclass Application that will serve as a base for the text and graphic User Interfaces.	A superclass: Application	When the TextApplication class is fully implemented: can display a full-fledged user interface and handle user inputs.
U1.1	Opens a file and adds it to the playlist.	A protected method : openFile	When the class is able to verify if the given path leads to an actual file and add said path to the playlist.
U1.2	Plays the current song if it is paused.	A protected method: play	When the song is successfully played.
U1.3	Pauses the current song if it is playing.	A protected method: pause	When the playing song is successfully paused.
U1.4	Starts the previous song in the playlist.	A protected method: previousSong	When the application can switch between this song and the previous one and handle the case when there isn't one.
U1.5	Starts the next song in the playlist.	A protected method: nextSong	When the application can switch between this song and the next one and handle the case when there isn't one.
U2	Create a text based application to serve as an interface for the media player.	A class: TextApplication, subclass of Application	When TextApplication can display a full-fledged user interface and handle user inputs.
U2.1	Handles what happens when a SIGINT signal is received.	A static private method : sigHandler and a static bool : stop	When whenever a SIGINT signal is received the 'stop' boolean is toggled to true which will end the main loop and correctly free memory.
U2.2	The main loop of the application, will handle all the necessary calls to other methods and serve as a core to the application.	A public method: run	When the event loop is properly handled. This method is a necessary building block but it will be built upon continuously.
U2.3	Draws a window using ncurses.	A private method : draw	When the window is properly drawn in the terminal.
U2.4	Handles user inputs.	A private method : handleEvents	When all necessary user inputs are handled : 'q' to quit, proper resizing of the window if the terminal is resized, press arrow to move in the menu...
U2.5	Handles the menu.	A private method : handleMenu	When all the menu's options call the right methods when they are selected.
U2.6	Draws the menu	A private method : drawMenu	When the menu options are displayed and you can see which option is currently selected.

ID	Feature	Deliverable	Completion Conditions
U2.7	Draws the "play/pause", "previous" and "next" buttons.	A private method : drawMediaButtons	When the media buttons are displayed.
U2.8	Draws the progression bar.	A private method : drawProgressionBar	When the progression bar is displayed and there is a visible indicator of how far into the track we are.
U2.9	Draws the spectrogram of the current song.	A private method : drawSpectrogram	When the spectrogram of the current song can be properly displayed with the ability to zoom in or out.
U3	Create a graphic application to serve as an interface for the media player.	A class: GraphicApplication, subclass of Application	When GraphicApplication can display a full-fledged user interface and handle user inputs.
U3.1	The main loop of the application will handle all the necessary calls to other methods and serve as a core to the application.	A public method: run	When the event loop is properly handled. This method is a necessary building block but it will be built upon continuously.
U3.2	Draws a window using SDL2.	A private method : draw	When SDL2 can bring up a working window.
U3.3	Handles user input.	A private method : handleEvents	When all necessary user inputs are handled : press ESCAPE to close the window, all the mouse events for the different buttons, properly handling the window being resized...
U3.4	Handles the menu.	A private method : handleMenu	When all the menu's options call the right methods when they are selected.
U3.5	Draws the menu	A private method : drawMenu	When the menu options are displayed and you can see which option is currently selected.
U3.6	Draws the "play/pause", "previous" and "next" buttons.	A private method : drawMediaButtons	When the media buttons are displayed.
U3.7	Draws the progression bar.	A private method : drawProgressionBar	When the progression bar is displayed and there is a visible indicator of how far into the music we are.
U3.8	Draws the spectrogram of the current song.	A private method : drawSpectrogram	When the spectrogram of the current song can be properly displayed with the ability to zoom in or out.

4.3 Networking

ID	Feature	Deliverable	Completion Conditions
N1	Generate and broadcast packets to a bunch of clients.	A class: Server.	When the Server is able to send TCP packets to every clients connected.
N1.1	Generate a blob of binary code that correspond to 3 PCM chunks and a byte of flags (new music, jump, pause, unpause, ...).	A public method: generate_packet	When the blob is correctly formatted.
N1.2	Broadcast a packet to every connected client using TCP. We're using TCP because we will not have time to customize UDP for reliability purposes.	A public method: broadcast_packet	When the blob could be send over the network without any issues.
N1.3	Poll for new clients or client disconnections.	A public method: poll_clients	When the server can handle connections.
N1.4	A way to start and stop the server.	Two public methods: run_daemon, stop_daemon	When the server can start, stop and restart without any issues.
N2	Read data from a packet and choose the correct function to run based on the flags.	A class: Handler	When the handler is able to read packets, extract data and choose the correct function based on the flags.
N2.1.1	Write to a cache file that is destroyed when the destructor is called.	A public method: write_to_cache	When the cache is correctly written.
N2.1.2	Read from a cache file that is destroyed when the destructor is called.	A public method: read_from_cache	When the data is correctly read.
N2.2.1	Connect to a broadcast server using its IPv4 address and a port.	A public method: connect_to_broadcast	When the handler correctly connects to the server.
N2.2.2	Disconnect from a broadcast server.	A public method: disconnect_from_ broadcast	When the handler correctly disconnects from the server.
N2.3	Read the data from a packet using binary masks and then choose (based on the flags) the correct action to apply to the Music class.	A public method	When the data is correctly read.
N2.4	A way to start and stop the server.	Two public methods: run_daemon, stop_daemon	When the handler can start, stop and restart without any issue.

5 Annexes

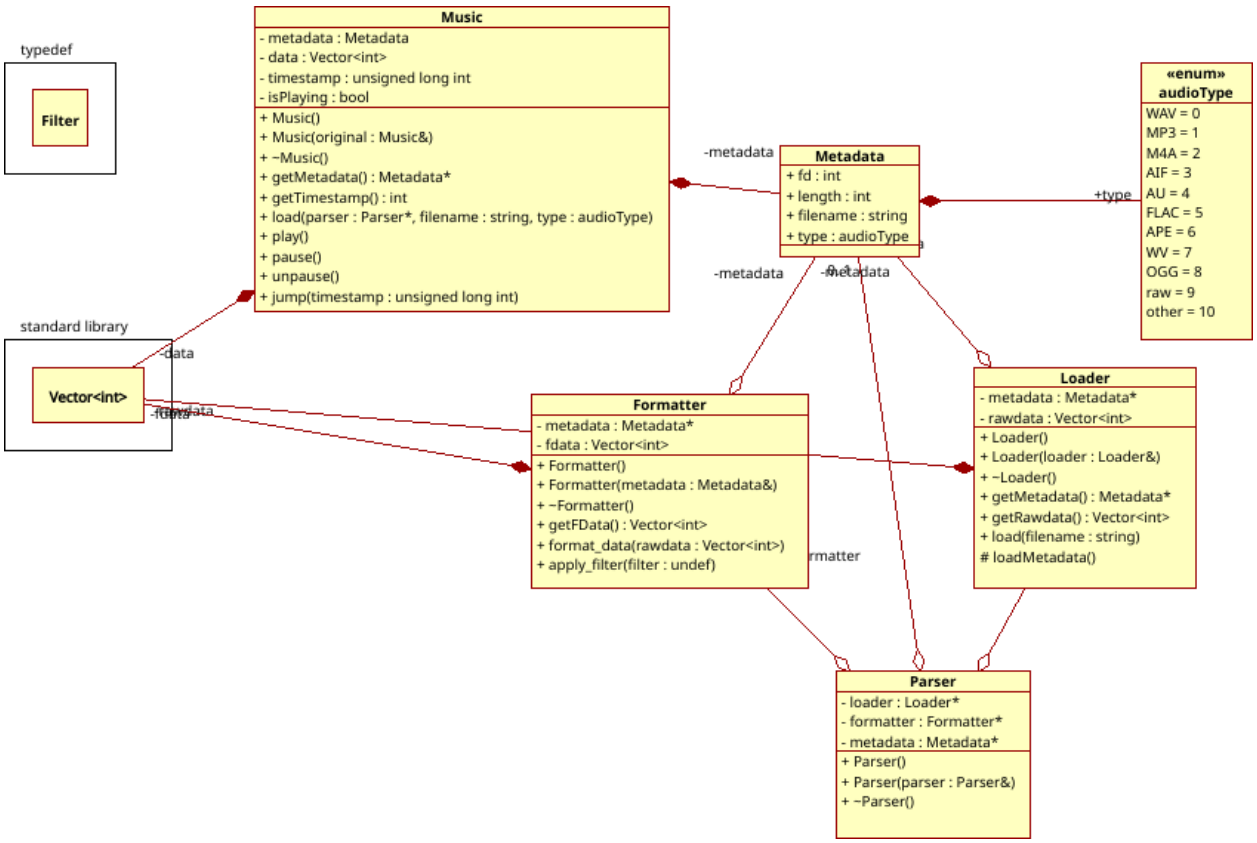

Figure 2: Core class diagram

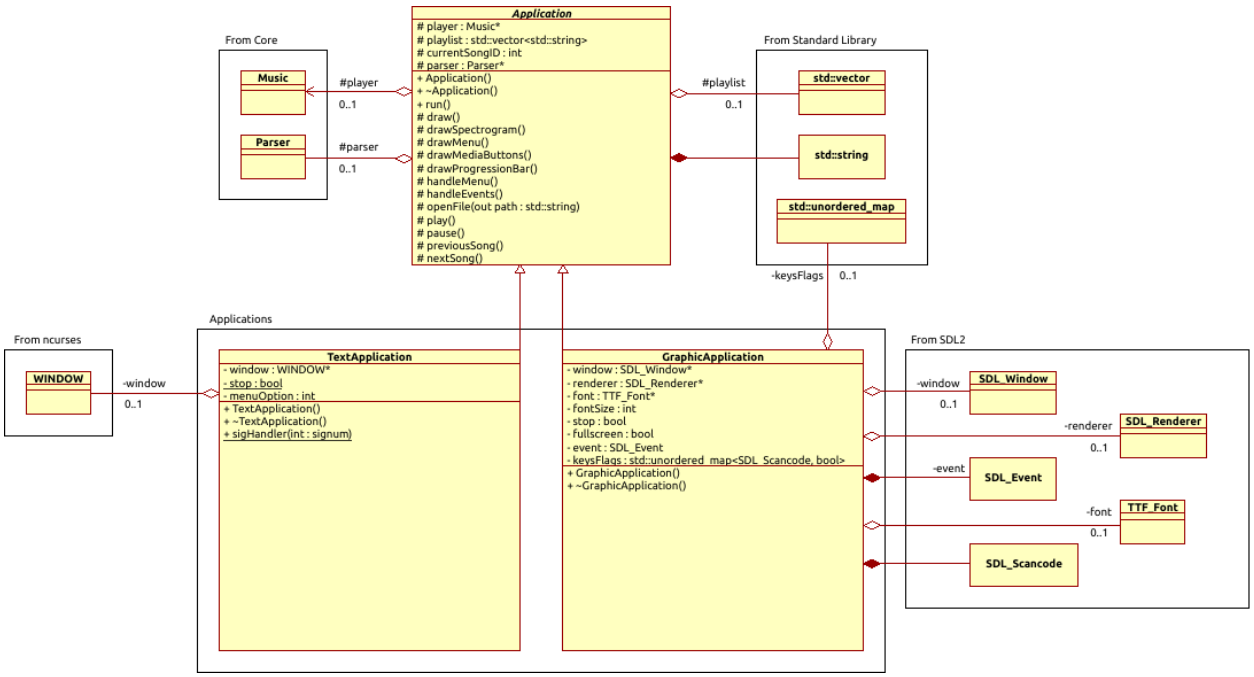

Figure 3: User Interface class diagram

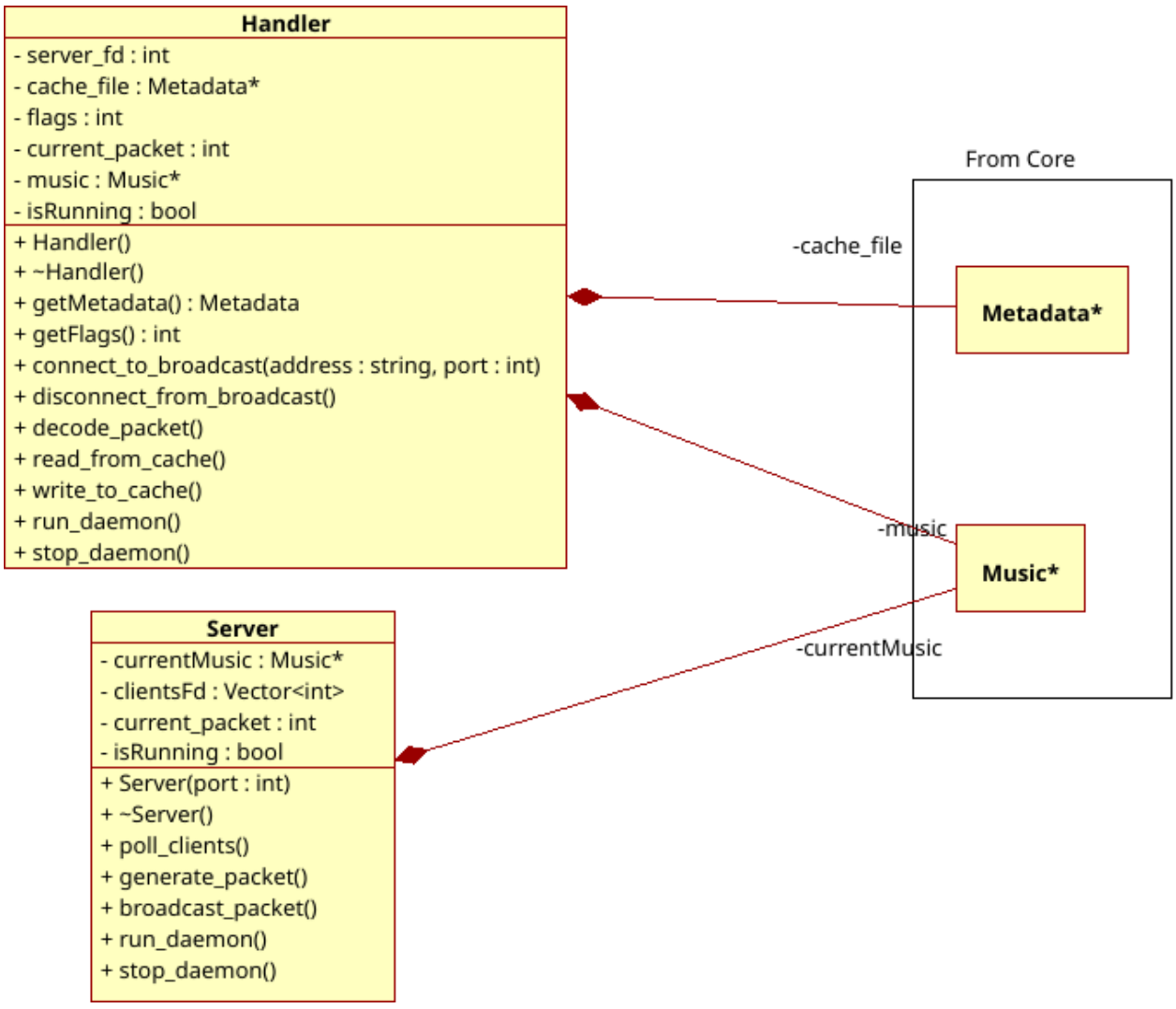

Figure 4: Networking class diagram

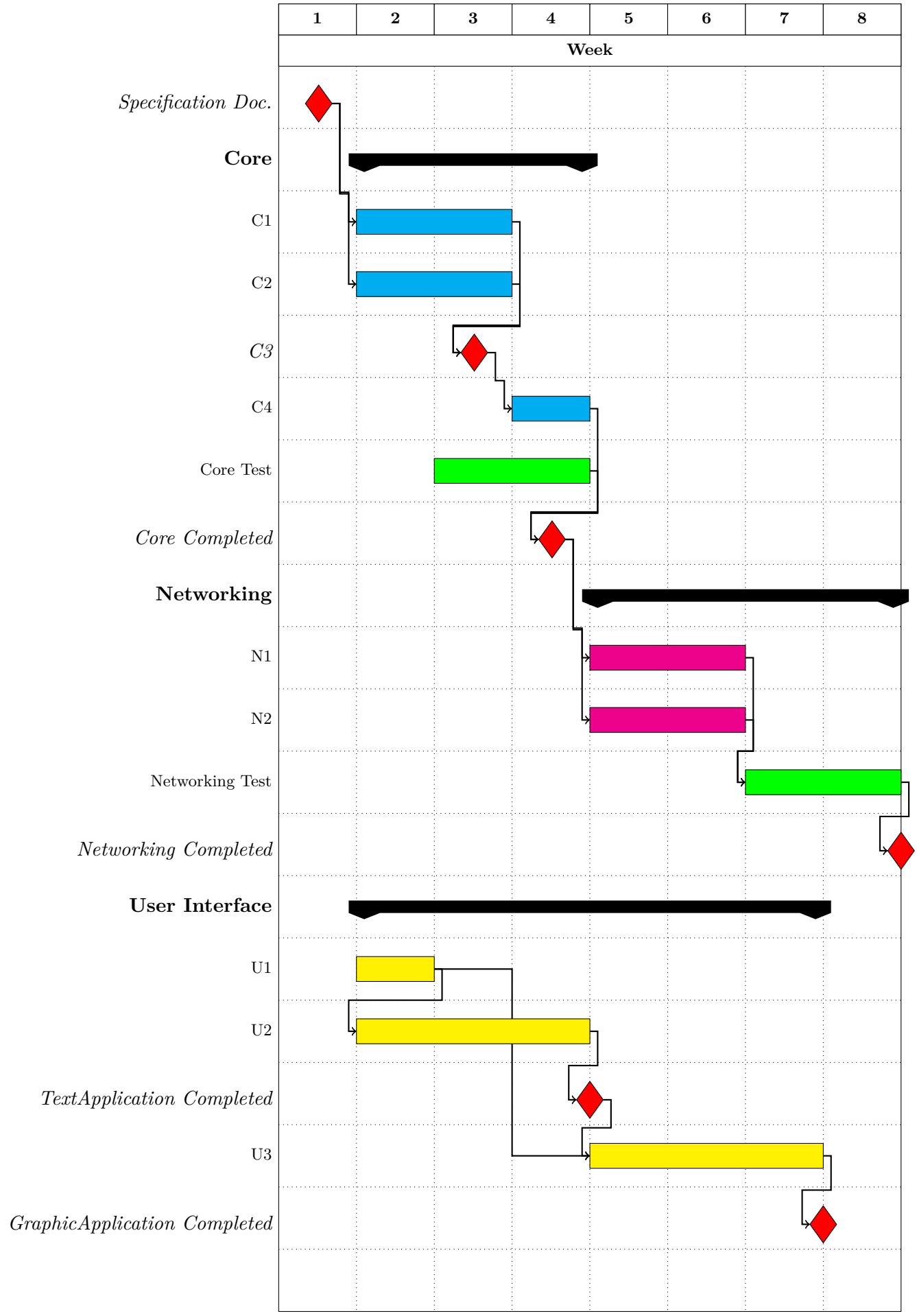

Figure 5: PancakeMediaPlayer Gantt Chart