

# CS480 Tower iLLuminati Design Report

Jenny Huang

December 17, 2014

## Contents

<b>1</b>	<b>Project Background</b>	<b>2</b>
<b>2</b>	<b>Problem Definition</b>	<b>3</b>
2.1	Goals . . . . .	3
2.2	Scope . . . . .	3
2.3	Deliverables . . . . .	3
2.4	Constraints . . . . .	3
2.5	Reference Material . . . . .	3
<b>3</b>	<b>Project Plan</b>	<b>3</b>
3.1	Tasks and schedule . . . . .	3
3.2	Team responsibilities . . . . .	4
<b>4</b>	<b>Concepts Considered</b>	<b>4</b>
4.1	Original ideas and those derived from other sources . . . . .	4
4.2	Quantitative Data or Measurements . . . . .	5
<b>5</b>	<b>Concept Selection</b>	<b>5</b>
5.1	Design Choices . . . . .	5
5.2	Concept Selection Criteria . . . . .	5
<b>6</b>	<b>System Architecture</b>	<b>5</b>
6.1	Architecture Design . . . . .	5
6.2	Decomposition Description . . . . .	6
<b>7</b>	<b>Data Design</b>	<b>7</b>
7.1	Data Description . . . . .	7
7.2	Data Dictionary . . . . .	7
<b>8</b>	<b>Component Design</b>	<b>7</b>
8.1	GUI . . . . .	7
8.2	Software Manager . . . . .	7
8.3	Event Manager . . . . .	8
8.4	Files Manager . . . . .	8
8.5	Command Center . . . . .	8
<b>9</b>	<b>Human Interface Design</b>	<b>8</b>
<b>10</b>	<b>Future Work</b>	<b>8</b>
10.1	Necessary Work . . . . .	8
10.2	Features not in current design . . . . .	8
10.3	Estimated size and duration of future work . . . . .	9
<b>11</b>	<b>Appendices</b>	<b>10</b>
11.1	Calculations & drawings . . . . .	10
11.2	Large tables & figures . . . . .	10
11.3	Vendor data sheets . . . . .	10
11.4	Source Codes . . . . .	10

# 1 Project Background

- Motivation for the work: Our client Dr. Rinker is very musical fan and popular star on campus, and he leads the ACM team for the department, and he enjoys offering music-related fun activities for the campus like tower light show for homecoming events. For tower light show, the current tower animator C#-programming based software plays the .tan light file type separately from the .wav audio file, which is not convenient for him and his ACM parties to use.
- Identify the need: Since University of Idaho Computer Science department is mainly C++-based programming, and the c# language does produce language barrier for a certain amount of ACM users, and the software is quite some distance from being user-friendly and functional complete, there exist the need for potential re-factor, reimplementation, or update on this software.
- Describe the expected benefits: Reimplement and update the software in C++ language will promote ACM user attendance on product software updates and also, we, as the senior design team would be able to get good practise and benefit a long way from various aspects like programming technical practice, software engineering, project design and handling, as well as problem-solving skills.

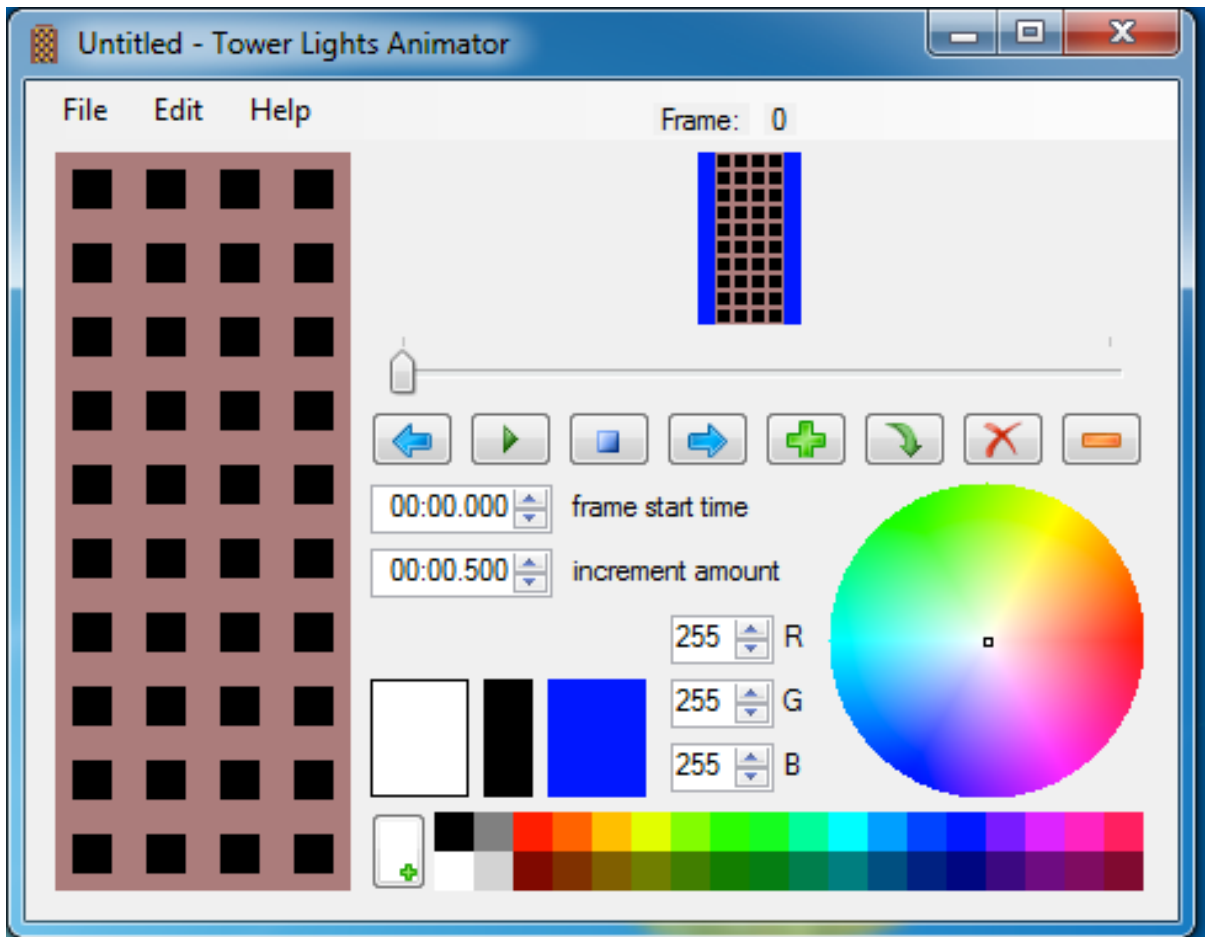


Figure 1: Base software Interface

## 2 Problem Definition

### 2.1 Goals

This design document describes the design of the software for tower light animation synchronized with audio music for the University of Idaho, Senior Design course. It is intended for the development team.

### 2.2 Scope

This application must satisfy the requirements as defined by the customer, Dr. Rinker.

- Must link the light animator .tan file with corresponding audio file
- Must redesign and reimplement the software in C++ and Qt Creator library for Graphical User Interface
- Must be a non-trivial tower light animator software
- Should make the software user-friendly and easy to use
  - When design each frame, the user promote less clicks for picking color for each grid;
  - Should have commands for automated movements
  - Should design pre-generated shapes and patterns
  - Could have pre-generated animations
  - The software could import from source files satisfies the specified requirements
  - Since audio file will be provided already, the software promotes user-friendly feature, the design should include a parser to parse audio file for tower light frame intervals.
  - The software should be real-time, which de-promotes slow response, and promote well-design for source program execution efficiency.

### 2.3 Deliverables

- Software which is well-designed and fully satisfies the client's requirements, and the application should also be user-friendly and executes smooth and efficiently.
- A Software Requirement Specification document and a user-manual which is easy to read and understand, as well as fully functional to explain all the necessary manipulations.

### 2.4 Constraints

As a senior team, we design well-design and implementation for the software, but there are critical issues that we need take into consideration for, for example, 80% of the team-mates are currently in the most famous and difficult course with extra overloads, and they can barely spend many time on this project for the fall semester. As a result, when we design the project, we keep the client's requirements in mind, and try to design the software so that the software meets the minimum requirement to be a software while satisfying the clearly-specified scope, while at the same time that the software is very expendable to promote more features, efficiency and user-experience.

### 2.5 Reference Material

- Qt Project Documentation: <https://qt-project.org/doc/>

## 3 Project Plan

### 3.1 Tasks and schedule

- Develop, review, and finish the design by the end of the first semester
- Finish the design documents so that after the Christmas break, we will still be able to pick up and start from where we left off from the fall semester

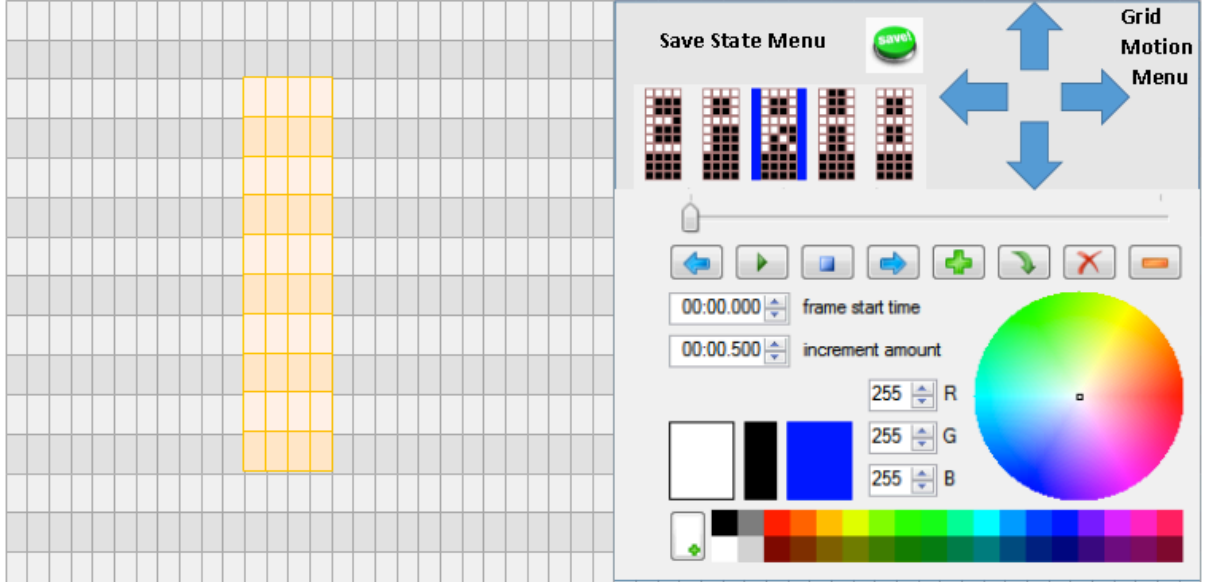


Figure 2: imaged-based GUI for Snapshot #1

### 3.2 Team responsibilities

- One team member created a image GUI design for our Snapshot #1, which is included in Figure 2
- One team member created a state diagram, and configured Qt Creator environment for the team
- One team member finished a wiki-page
- One team member gave a most basic C# code of base version software during one team member with Professor, and gave the presentation for design review
- The team has one effective team meeting for menu bar design on Nov 16th
- I was asked to type the team contract, and was the team member to design and implement a basic click-able GUI as the project implementation starting point
- I am right now asked to write the Design Report proposal individually for this course

## 4 Concepts Considered

### 4.1 Original ideas and those derived from other sources

- Original ideas were mainly derived from our current base version of C#-programming based software.
- Implementation language C++ was selected because our university is applied the C++-based data structure and programming teaching style; And Qt Creator was selected and required by course instructor Professor Bolden.

Table 1: System Functions compressed into Menubar

File	Edit	Play	Select	Mtool	Help
Open	Copy (C-c)	play from start	Row (SPC-mse)	Insert Shapes	Documentation
New	Cut (C-x)	play from current	Col (Sft-SPC-mse)	Define Pattern*	About
Save	Paste (C-v)	Pause	All (C-a)	Color Gradient*	
Save as	Insert After/Before	Stop	Invert Slt (C-i)		
Export	Clear	Move Forward	Slt Shift-C		
Close	Delete	Move Backward	Slt C-mouse		
Exit	Undo (C-z)				
	Redo (C-y)				

## 4.2 Quantitative Data or Measurements

- According to requirement that the software needs to be user-friendly, and must conduct less clicks when picking color and design color frames;
- According to requirement that the software needs to be smart, so that it can NOT be slow for source program execution.
- References: Literature/Catalogue Review

# 5 Concept Selection

## 5.1 Design Choices

We have mainly three major concept selections for this project design, and they are GUI main window layout design, pre-defined color design, and a potential wave parser insertion into the design and GUI.

- GUI Layout Selection: We have been proposed two GUI designs, snapshot when we discussed about them, which are listed as followed.
- My design for user pre-defined colors, Qt Quick dynamic JS creation or regular QPushButton one as I am doing now? Maybe I should implement the project piece by piece, from easy to difficult updated implementation
- In my currently partially implemented GUI, I don't have the .wav source file involved except for playing manipulations on this type of file, for example, play, pause and stop. But we could include the .wav type file parser engine and perform graphics generation. This project is an interesting project with lots of potentials. I will come back to work on this one later when I solved my priorities

## 5.2 Concept Selection Criteria

The criteria that we applied for the selection of concepts include:

- Scientific naming conventions;
- Concepts comparison among team members; Take the very first GUI main window design as the example, after we having implemented a form-based main window layout (included in Figure 3 & Figure 4), we discussed among team members, for the user-friendly software design requirement, we came to agreement that we should pick the traditional wide-screen one. And similarly came to other selection results.

# 6 System Architecture

## 6.1 Architecture Design

The architectural design was developed by following the software requirements according to Software Requirement Specification (SRS). Here, the Tower Light Animator will be described in terms of basic modules which will give the reader a first understanding of the functionalities of the software.

The Tower Light Animator can be described in five different modules, as drawing listed in Figure 5 - System Architecture High Level Model:

- **GUI:** It is in charge of giving access to our clients and any software user to interact with the program;
- **Software Manager:** This module is the management center of the several modules. It controls the status of software together with several other minor functionalities
- **Event Manager:** This module is responsible for handling mouse click events and keyboard events.
- **File Manager:** It is responsible for writing the software generated product contents into output specific types of files, together with some potential functionalities
- **Command Center:** This module is the center for specifying and defining the majority kinds of GUI-related activity body.

## 6.2 Decomposition Description

In this section, each module inside the Tower Light Animator application described previously is broken into smaller specific tasks which will allow the reader to identify what is the expected behaviour of each module. The detailed contents are also drawing shown in Figure 6: System Architecture Block Diagram of the animator.

- **GUI**
  - User interaction: this component is in charge of capturing the decisions made by the our clients and software user (buttons, sliders, etc.).
  - Software Feedback: this component shows all the alerts and current status of the in-progress tower light frame-making that are sent by the Software Manager module in order to inform the clients what is happening at every moment of the progress.
  - Entity Representation: this component contains the rest of the graphic elements that are part of the Tower Light Animator and the user can apply of (i.e. color sources, including predefined colors, picking from color wheel etc.).
  - It is in charge of giving access to our clients and any software user to interact with the program; The GUI is a "MUST" for the software.
- **Software Manager**
  - Control Flow: it controls the status of the software (i.e, player turn and time taken by the computer players)
  - This module is the management center of the several modules. It decides which other module is necessary to call to get source feed or to execute commands. Also, it is in charged of controlling the software module-to-module control flow, supervising the commands rules and handling excepts and errors as well.
  - This module functions and promotes software efficiency.
- **Event Manager**
  - This module is responsible for handling mouse click events and keyboard events.
  - It is responsible for receiving and handling these two device events signals, sorting, filtering, identifying the source signals, and links, triggers the corresponding response to command center;
  - This module functions for reacting to user-input events and promotes user-friendly features.
- **Files Manager**
  - It is responsible for writing the software generated product contents into output specific types of files.
  - And it is also responsible for importing and parsing the source input file and stores the source information into efficiency-concentrated data structures, and parse data into visible graphics as well if necessary.
  - This is the valued I added to the project, and will be fully conducted when the project is on track and in good shape.
  - The module is a "MUST"-accomplish feature for user-software interaction production – write into file, and also promotes user-friendly and software-smart features.
- **Command Center**

- This module is the center for specifying and defining the majority kinds of GUI-related activity body, like what should the software do if the use clicks a specific button with a mouse click on the left, the corresponding response the software would conduct will be defined in one command or more. And this is the center for all these commands.
- This module is the most basic execution element, and software cannot function without this module.

## 7 Data Design

### 7.1 Data Description

The system architecture is implemented by using C++ and Qt Creator library specific classes. Accordingly, the system entities are stored, processed and organized in these classes via different data structures, including 2D array of QWidget pointers, array list, containers and maps.

### 7.2 Data Dictionary

Component	SubComponent	Classes/Interfaces
GUI	User Interaction	Gid
GUI	User Interaction	AGrid
GUI	User Interaction	GridList
GUI	Software Feedback	Animation, QList
GUI	Entity Representation	Loc
GUI	Entity Representation	MyPushButton
GUI	Entity Representation	MyGridLayout
GUI	Entity Representation	MyDoubleSpinBox
Event Handler	Filter Handler	KeyEvent
Event Handler	Filter Handler	KeySequence
Event Handler	Filter Handler	MouseEvent
File Manager	Read/Write	QFile, QFileDevice

## 8 Component Design

### 8.1 GUI

- User interaction
- QMenu, QAction
- QHBoxLayout, QVBoxLayout, QGridLayout
- QPushButton, MyPushButton
- QSpinBox, MyDoubleSpinBox
- QSlider
- QLabel
- ColorWheel
- Qt Quick for JS dynamic user-custom colors

### 8.2 Software Manager

- QThread
- QRunnable
- QtConcurrent
- QTimerEvent
- QChildEvent

### 8.3 Event Manager

- Q\_OBJECT Signals & Slots
- EventHandler
- EventFilter
- QKeyEvent
- QKeySequence
- QMouseEvent
- QWidget::paintEvent

### 8.4 Files Manager

- QFile
- QTemporaryFile
- QFileDevice
- QBuffer
- QProcess

### 8.5 Command Center

- This module contains mainly functions.

## 9 Human Interface Design

- Main Window
- Pop-up Window
- Partially Implemented detailed Class Diagram

## 10 Future Work

### 10.1 Necessary Work

- Implement functionality to GUI
- Implement multiple pixel movement
- Figure out modifications to .tan file layout
- Next Semester
  - Implement more advanced animations/movements
  - Add more GUI features

### 10.2 Features not in current design

- Threads implementation
- Software IO manager and manipulations
- Audio file parse to generate consistence for timestamps incremental amount, a basic intuitive layout is listed as below presented in Figure 6, which has a timestamps double spin-box to show the detailed time and parsed wave drawing picture:

### 10.3 Estimated size and duration of future work

- This is a interesting project, and for the practise propose, it is better designed and implemented within a team environment, and in that case, it will be fast, because there are more contributors to the project on ideas of design and implementation
- for Qt Signals and Slots So far I got three ideas how to implement the connection between signals and slots
- Inheritance to rewrite signals and redefine slots
- Use event-filter to find signal sender(), and conduct the connection by scanning the widgets
- Most basic method that I implemented already, by connecting one by one, and repeat

The very first method I tried a little bit, but failed to get my push-button painted. But I can spend some time to debug this one, and I could also try the second method as well. There must be some way to facilitate the linking process.

# 11 Appendices

The supporting documents to long or detailed for main body includes the following several sections.

## 11.1 Calculations & drawings

## 11.2 Large tables & figures

Table 2: Commands and Quick keys		
MainMenu	Commands	Fast Keyset
File	File	C-f
	New	
	Open	C-o
	Save	
	Save as	
	Export	
	Close	
Edit	Exit	
	Edit	C-e
	Cut	C-x
	Copy	C-c
	Paste	C-v
	Insert After/Before	
	Delete	
Play	Clear Frame	
	Undo	C-z
	Redo	C-y
	Play	C-p
	From Beginning	C-b
	From Current	C-n (now)
	Pause	
Select	Stop	
	Move Forward	
	Move Backward	
	Preview Mode	C-r (review)
	Select	C-s
	Row	SPC-mse
	Col	Sft-SPC-mse
Mtool	All	C-a
	Invert	C-i
	Deselect	C-d
	Mtool	C-m (movie)
	Insert Shapes	
	Define Pattern*	
	Color Gradient*	
Help	Help	C-h
	Documentation	
	About	

Visualization (sketches, drawings, diagrams)

- Classes and Prototypes Modeling and/or Experimentation

## 11.3 Vendor data sheets

## 11.4 Source Codes

- Coding

- Based on Design
- Testing

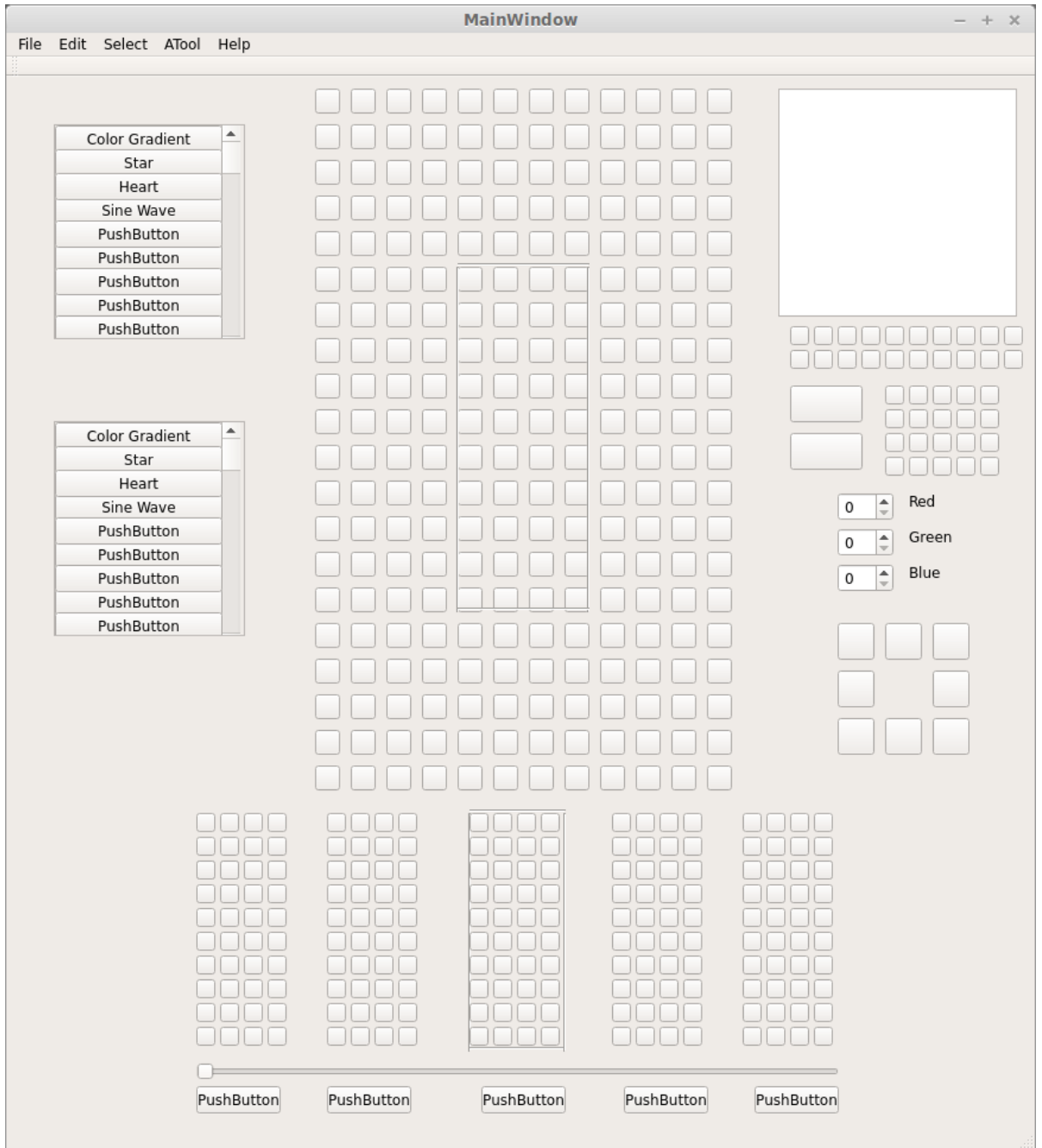


Figure 3: Team manager proposed new design

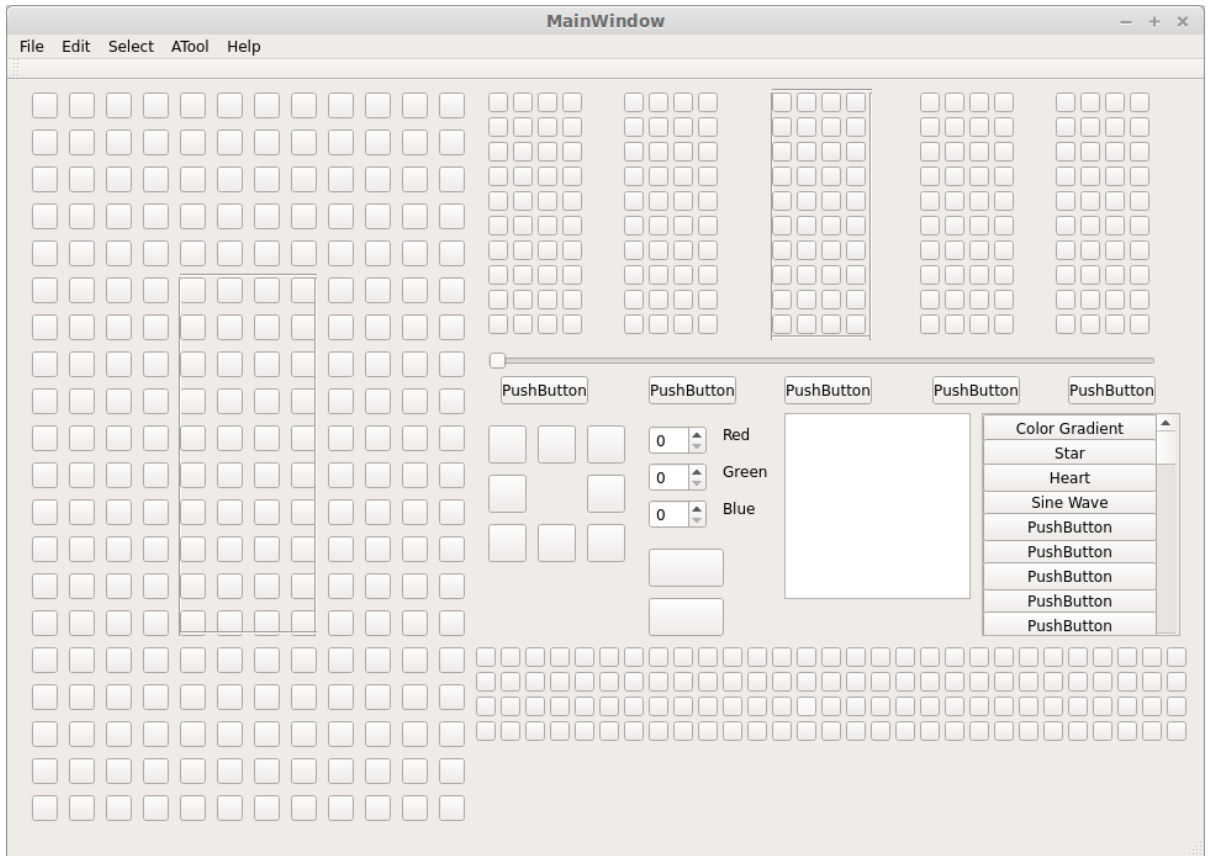


Figure 4: Original traditional design

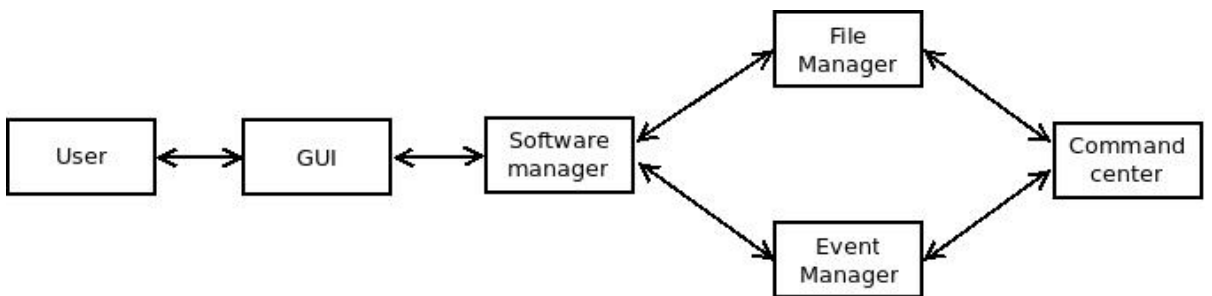


Figure 5: System Architecture High Level Model

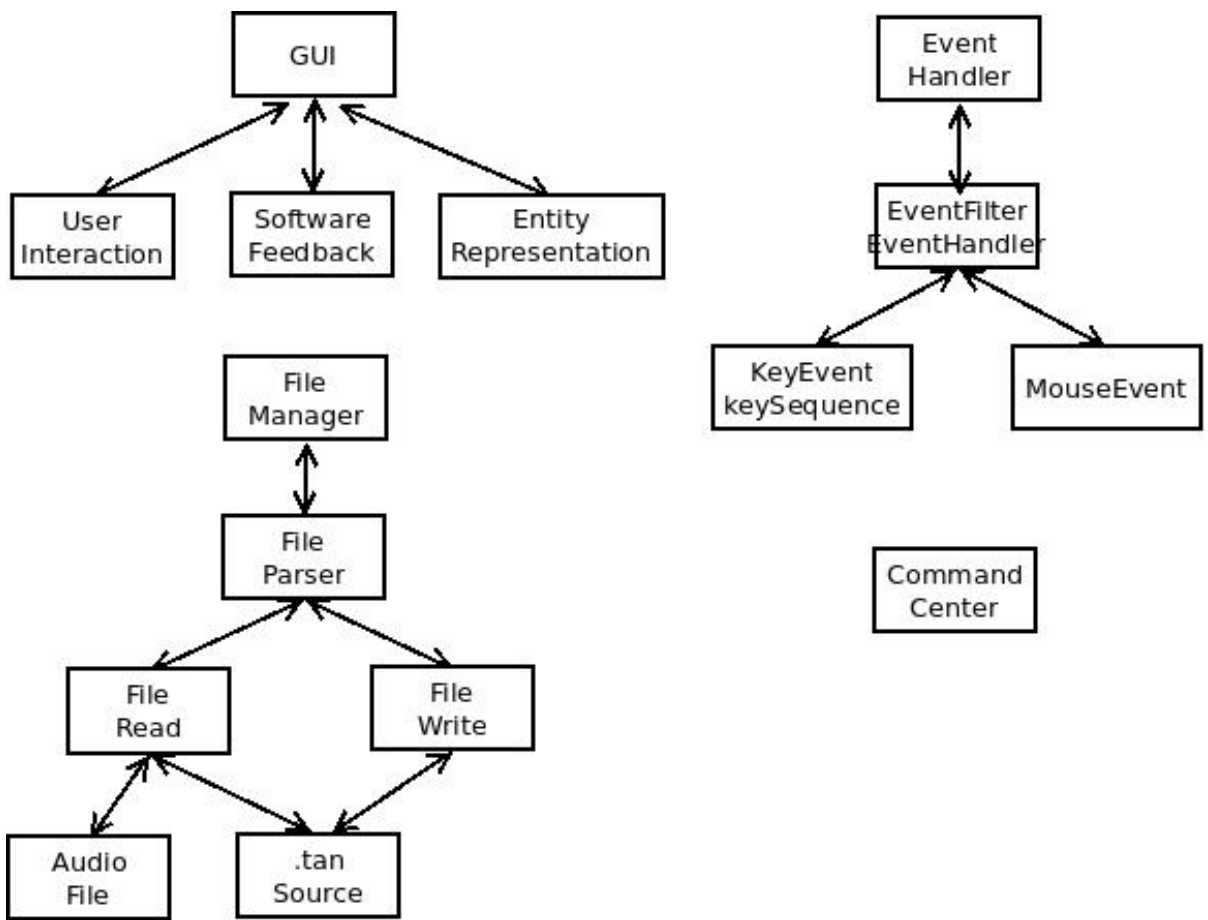


Figure 6: System Architecture Block Diagram for Tower Light Animator

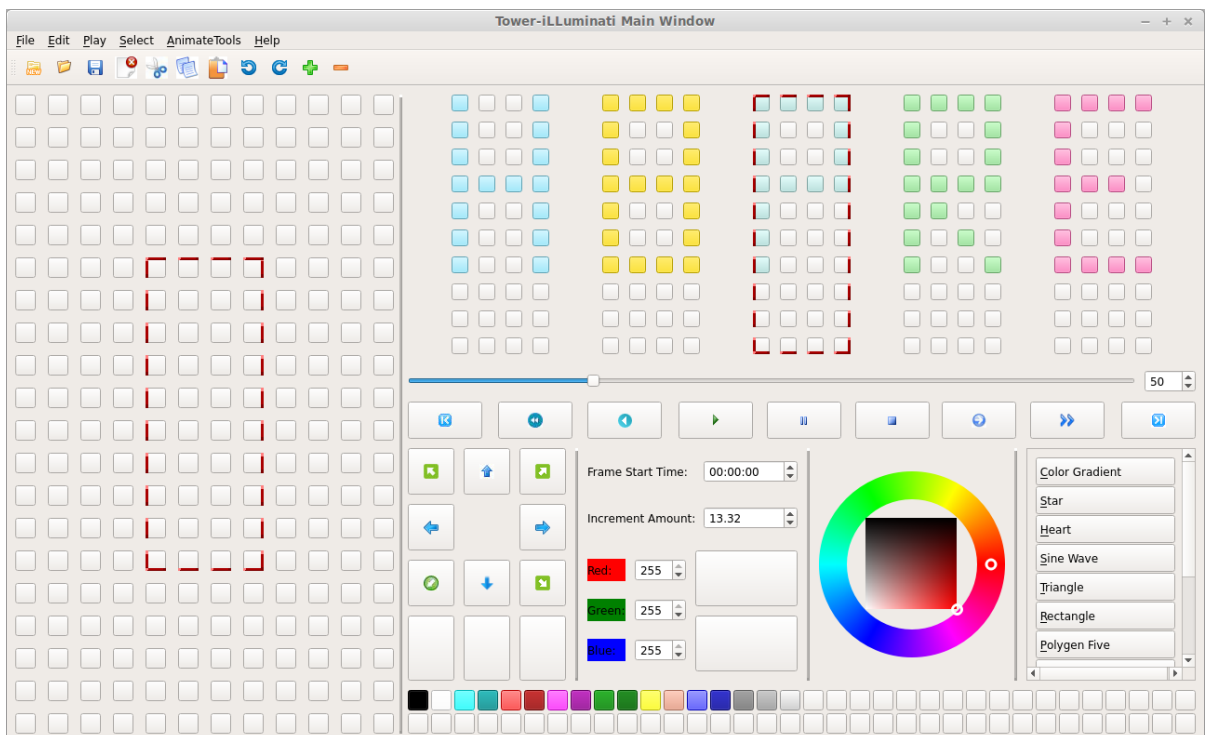


Figure 7: Main Windows

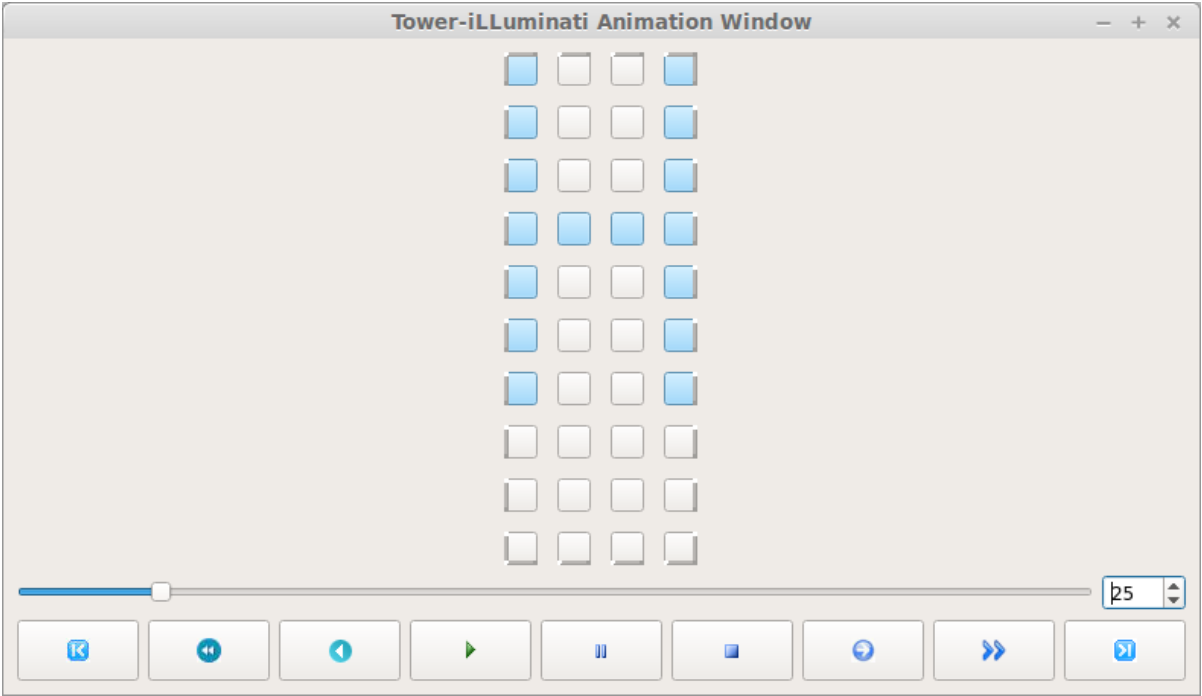


Figure 8: Pop-up Windows

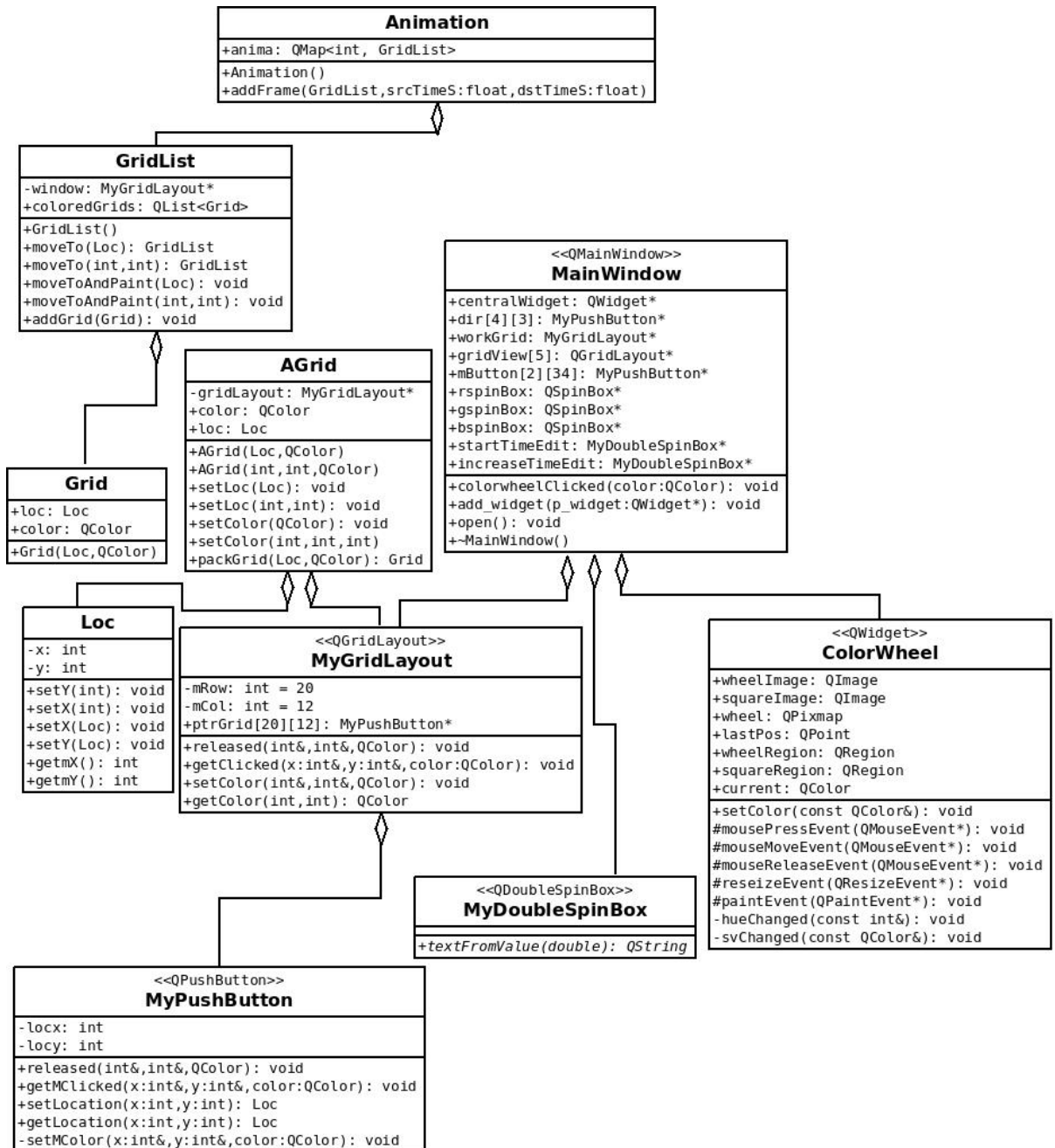


Figure 9: Partially implemented detailed class diagram

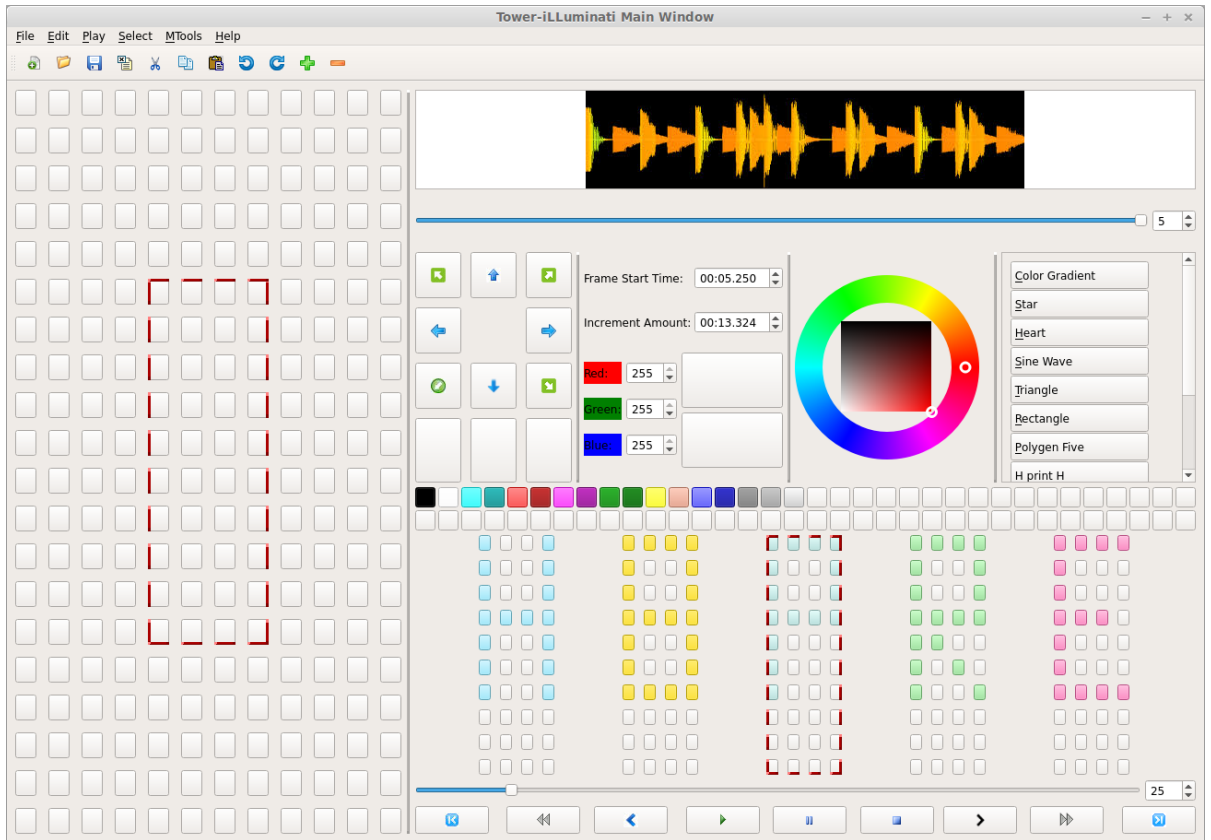


Figure 10: GUI with audio file parsed for timestamps

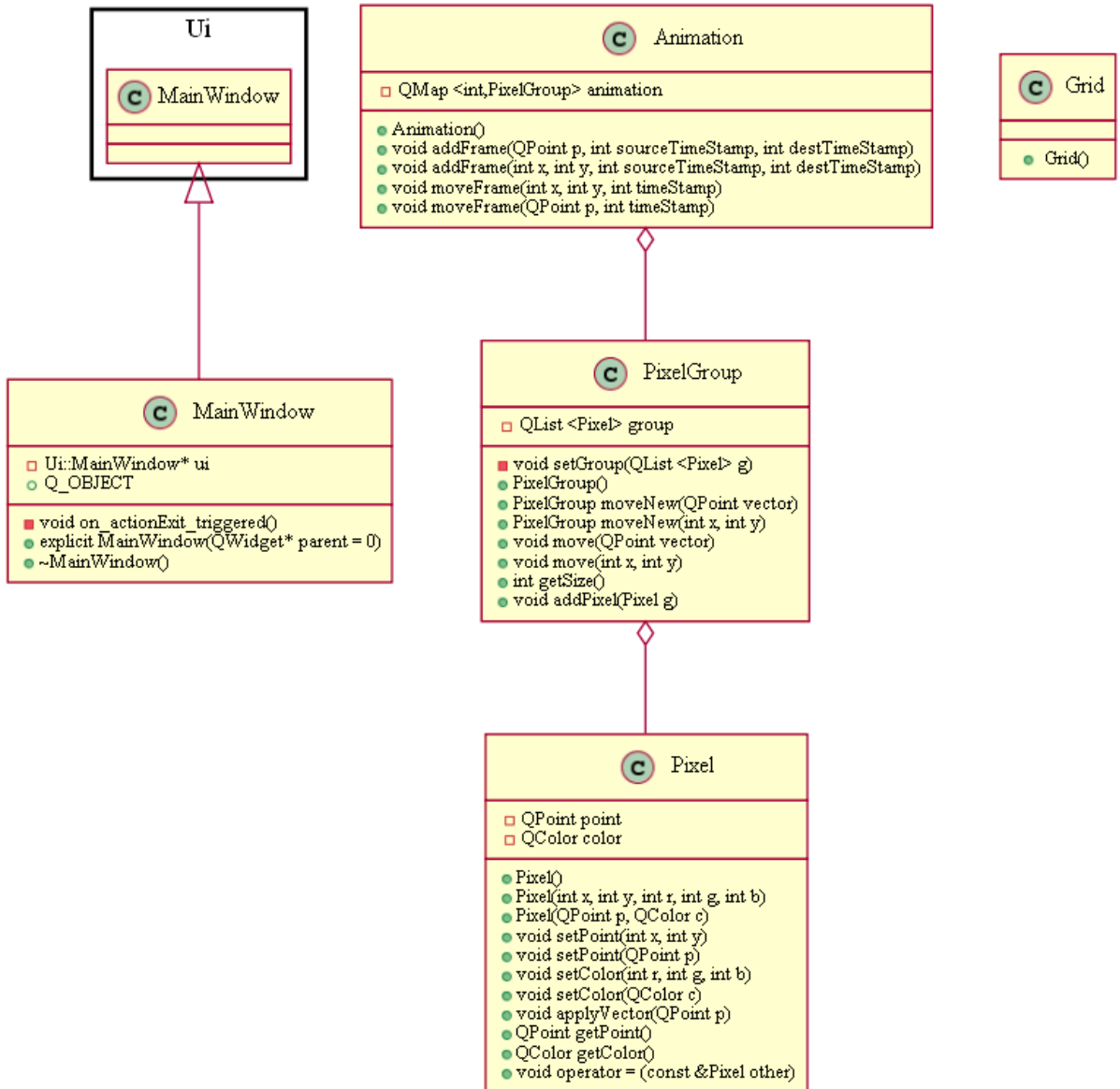


Figure 11: System Architecture Class Diagram

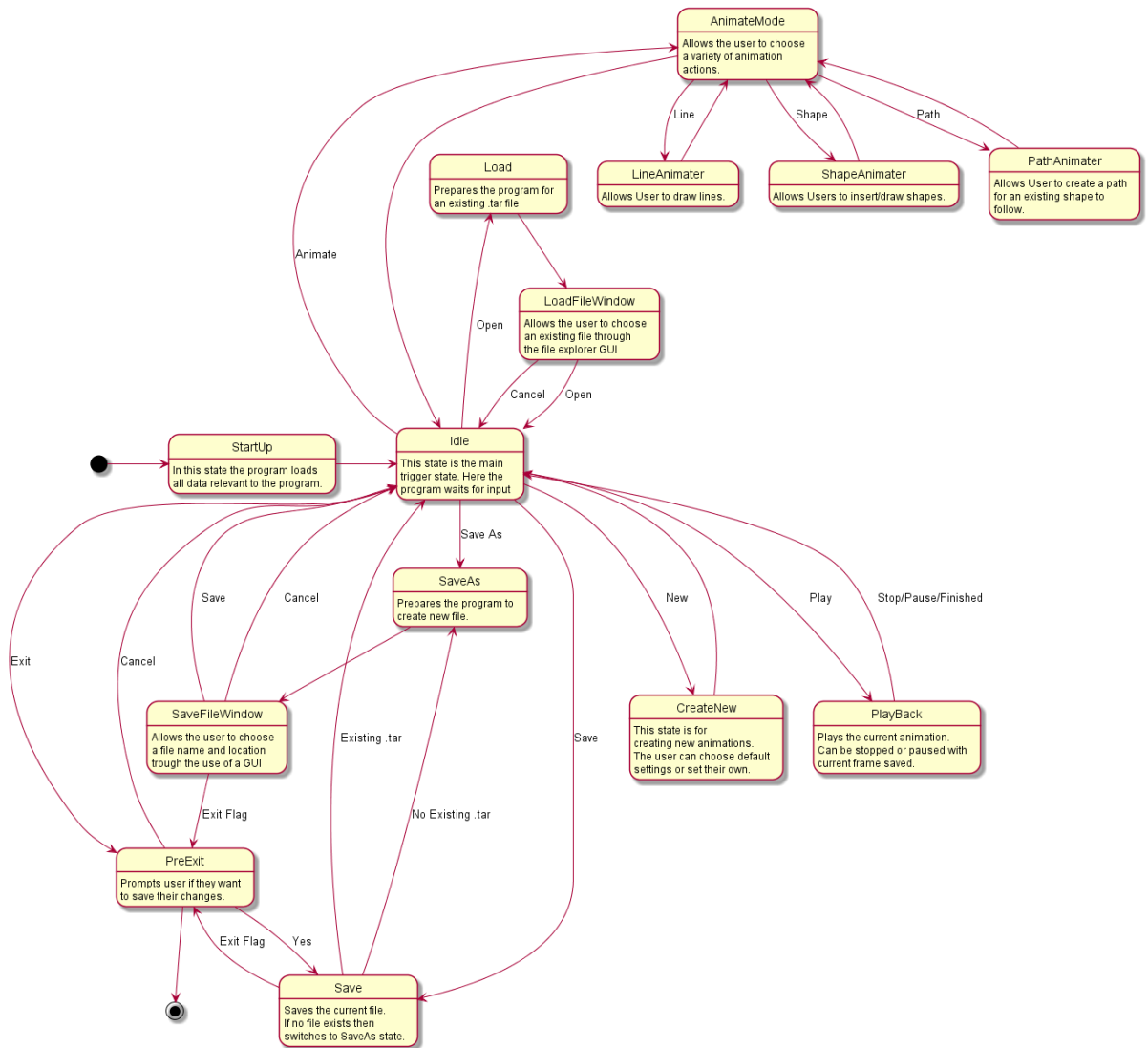


Figure 12: Software State Diagram