

به نام پروردگار یکتا



دانشگاه صنعتی اصفهان
دانشکده مهندسی برق و کامپیوتر

موضوع پروژه

نرم افزار اجرای زنده موسیقی و lightshow با استفاده از دستگاه - Launchpad Mini MK2

MIDI Controller

استاد راهنما

دکتر الهام محمودزاده

گزارش پروژه کارشناسی

رسول کامکار

شهریور ۱۴۰۳

فهرست

فصل اول مقدمه	۷
۱.۱- بررسی مشکل و هدف پروژه	۷
۲.۱- آشنایی با دستگاه Launchpad Mini Mk2	۸
۳.۱- آشنایی با استاندارد MIDI	۸
۴.۱- معرفی تکنولوژی‌های استفاده شده	۹
۵.۱- هدف نهایی پروژه	۱۱
فصل دوم طراحی نرم افزار	۱۲
۱.۲- نیازمندی‌ها	۱۲
۱.۱.۲- نیازمندی‌های عملکردی	۱۲
۲.۱.۲- نیازمندی‌های کیفی	۱۵
۲.۲ معماری نرم افزار	۱۶
۱.۲.۲- انتخاب معماری مناسب	۱۶
۲.۲.۲- زمینه سیستم	۱۸
۳.۲.۲- مخازن سیستم	۱۹
۴.۲.۲- اجزاء سیستم	۲۱
۳.۲- موارد کاربری	۲۵
۱.۳.۲- پنجره اصلی	۲۵
۲.۳.۲- نوار ابزار	۲۶
۳.۳.۲- صفحه خلاصه	۲۷
۴.۳.۲- صفحه تنظیم صدا	۲۸
۵.۳.۲- صفحه انیمیشن	۲۹

۳۰	۶.۳.۲ - مدیر پروژه
۳۰	۷.۳.۲ - ماندگاری
۳۱	۸.۳.۲ - تاریخچه یادمان‌ها
۳۲	۹.۳.۲ - مبدل میدی
۳۳	۱۰.۳.۲ - مخزن صداها
۳۴	۱۱.۳.۲ - مدیر انیمیشن
۳۵	۱۲.۳.۲ - فهرست افزونه‌ها
۳۵	۴.۲ - توالی عملکرد مبدل میدی
۳۶	۱.۴.۲ - توالی پخش صدا
۳۶	۲.۴.۲ - توالی‌های پخش انیمیشن
۳۹	فصل سوم پیاده‌سازی نرم افزار
۳۹	۱.۳ - کلاس‌ها
۳۹	۱.۱.۳ - GUI
۵۰	۲.۱.۳ - Business
۵۹	۳.۱.۳ - Persistence
۶۰	۴.۱.۳ - MIDI
۶۲	۲.۳ - افزونه‌ها
۶۵	۳.۳ - آزمون نرم افزار
۶۶	۱.۳.۳ - تست‌های منطق
۶۷	۲.۳.۳ - تست‌های ادغام
۶۷	۳.۳.۳ - تست‌های میدی
۶۸	۴.۳.۳ - تست‌های افزونه‌ها
۶۸	۴.۳ - طراحی رابط کاربری

فصل چهارم جمع‌بندی و پیشنهادها.....	۷۴
۱.۴- نتیجه.....	۷۴
۲.۴- پیشنهادها.....	۷۴
۵- پیوست‌ها.....	۷۵
۶- مراجع.....	۷۸

فصل اول

مقدمه

۱.۱ - بررسی مشکل و هدف پروژه

در دنیای موسیقی، سخت افزارهایی به نام میدی کنترلر^۱ برای تسریع کار آهنگسازان وجود دارد. این دستگاه‌ها شامل طیف گسترده‌ای از لحاظ طراحی، ساختار، هدف و کاربرد هستند. در این حوزه، آهنگسازان از نرم افزارهایی تحت عنوان کارگاه صدای دیجیتال^۲ یا به اختصار دوا^۳ برای تولید موسیقی استفاده می‌کنند. این نرم افزارها یک استودیو کامل با امکاناتی از قبیل شبیه سازی انواع سازها، میکس و مستر^۴ سخت افزارهای افکت صوتی و بسیاری موارد دیگر را در اختیار کاربر قرار می‌دهند. از نمونه این نرم افزارها می‌توان به Ableton و FL Studio اشاره کرد. دستگاه‌های میدی کنترلر به صورت بومی^۵ برای اتصال به این نرم افزارها و کنترل فضای این استودیوهای مجازی طراحی شده‌اند.

دستگاه لانچپد^۶ یک نمونه میدی کنترلر است که برای کنترل قابلیت‌های مختلف دوا استفاده می‌شود. در کنار این کاربرد اصلی، علاقه‌مندان و نوازندگان موسیقی از این دستگاه برای اجرای زنده هم استفاده می‌کنند. دلیل این مسئله، طراحی خاص این دستگاه و همچنین وجود LEDهای مختلف برای هر دکمه این دستگاه است که قابلیت نمایش نور^۷ را به کاربر ارائه می‌دهد. مشکل اصلی، عدم وجود یک نرم افزار مناسب برای این هدف است. نرم افزارهای دوا موجود به دلیل محیط بسیار حرفه‌ای، نیاز به زمان زیاد برای یادگیری دارند و همینطور محیط کاربرپسندی^۸ ندارند، از طرفی، هدف اصلی این نرم افزارها تولید موسیقی است و نه اجرای زنده، به همین دلیل برای رسیدن به این هدف با یک نرم افزار دوا، کاربر مجبور به تلاش اضافه است که باعث کاهش بازدهی می‌شود. در بسیاری از موارد، یک دوا

^۱ MIDI Controller

^۲ Digital audio workstation

^۳ DAW

^۴ Mix & Master

^۵ Native

^۶ Launchpad

^۷ Lightshow

^۸ User-friendly

قابلیت تنظیم نمایش نور لانچپد را ندارد و در نتیجه کاربر مجبور به نصب افزونه‌های^۱ مختلف است. برای درک بهتر از فضای نرم‌افزارهای دآو تصاویری از Ableton و FL Studio در بخش پیوست (تصاویر ۷۰ و ۷۱) موجود است.

هدف این پروژه، طراحی و پیاده‌سازی یک نرم‌افزار با محیط کاملاً کاربرپسند است که هدف آن تنها اجرای زنده موسیقی و نمایش نور با استفاده از یک دستگاه لانچپد، در این مورد مدل Mini Mk2 است.

۲.۱ - آشنایی با دستگاه Launchpad Mini Mk2

این دستگاه یکی از مدل‌های مختلف سری لانچپد ساخت شرکت Novation [1] است. این لانچپد دارای ۱۶ دکمه کنترلی و ۶۴ دکمه اصلی است. هر دکمه دارای دو LED قرمز و سبز است که با ترکیب این دو، میتوان به رنگ‌های زرد و کهربایی^۲ نیز دست پیدا کرد. با فشار هر دکمه، یک پیام MIDI به سیستم متصل شده ارسال می‌شود و همچنین وضعیت LEDها را می‌توان با ارسال پیام از سیستم به لانچپد تنظیم کرد، جزییات پروتکل مربوطه در وبسایت دکتر سندر لیمنز^۳ موجود است. [2] (متأسفانه به دلیل قدیمی بودن مدل این دستگاه، این سند از وبسایت رسمی شرکت Novation حذف شده است)



شکل ۱ دستگاه Launchpad Mini Mk2

۳.۱ - آشنایی با استاندارد MIDI

پس از ورود سازهای دیجیتال و کمرنگ شدن سازهای آنالوگ، پروتکل رابط دیجیتال ابزار موسیقی^۴ یا به اختصار میدی^۵ برای ذخیره اطلاعات موسیقی دیجیتال و همینطور ارتباط بین دستگاه‌ها در دنیای کامپیوترها طراحی و شروع به استفاده شد [3]. این پروتکل می‌تواند تغییرات یک ساز دیجیتال را به دستگاه‌های دیگر اطلاع دهد و همینطور می‌توان از آن برای ذخیره این تغییرات به صورت فایل استفاده کرد. به طور کلی انتقال داده در این پروتکل به

^۱ Plugin

^۲ Amber

^۳ Sander Leemans

^۴ Musical Instrument Digital Interface

^۵ MIDI

صورت پیام‌های ۳ بایتی انجام می‌شود که بسته به نوع دستگاه و کاربرد می‌تواند شامل اطلاعات مختلفی باشد. به طور مثال، با فشردن دکمه A1 لانچ‌پد که نشان دهنده نوت C-1 است، پیام (144,0,127) و با رها کردن همان دکمه پیام (144,0,0) به سیستم ارسال می‌شود.

۴.۱ - معرفی تکنولوژی‌های استفاده شده

- زبان برنامه نویسی ++C: این نرم افزار باید فایل‌های صوتی را با سرعت بالا پردازش و پخش کند. همچنین انیمیشن‌های نمایش نور تعریف شده توسط کاربر باید با حداقل خطا به پیام‌های میدی تبدیل شده و به دستگاه لانچ‌پد ارسال شود. به دلیل حساسیت بر سرعت بالا و نیاز به حداقل تاخیر در بخش‌هایی از این پروژه و به طور کل نیاز به وجود خاصیت بلادرنگی^۱، زبان سی پلاس پلاس برای پیاده‌سازی این نرم افزار انتخاب شد.
- کتابخانه [4] tgui: این کتابخانه برای پیاده‌سازی رابط گرافیکی کاربری^۲ استفاده می‌شود. دلیل انتخاب این کتابخانه، پشتیبانی از سیستم‌های عامل مختلف و همچنین استفاده از پشت-پایان^۳ گرافیکی مختلف مانند SFML، RAYLIB، SDL و ... است. همچنین به دلیل استفاده از این بک‌اندها، برنامه عملیات‌های مربوط به گرافیک را می‌تواند بر روی کارت گرافیک اصلی انجام دهد که باعث افزایش سرعت کلی سیستم می‌شود. از طرفی این کتابخانه قابلیت‌های اضافه کردن ویجت^۴ را به سادگی در اختیار توسعه‌دهنده قرار می‌دهد که به دلیل نیازهای خاص این نرم افزار بسیار حیاتی است.
- کتابخانه SFML [5]: یک کتابخانه چند رسانه‌ای است که برای محیط گرافیکی برنامه و همچنین به عنوان بک‌اند tgui استفاده می‌شود. دلیل استفاده از این کتابخانه، راحتی توسعه نسبت به نمونه‌های دیگر مانند SDL و OpenGL به دلیل پشتیبانی از شیء‌گرایی ++C است.
- Boost [6]: یکی از بزرگ‌ترین و پراستفاده‌ترین مجموعه کتابخانه‌های زبان ++C است که از کتابخانه‌های مختلف آن در این پروژه استفاده شده. دلیل استفاده از این مجموعه، نه تنها طیف گسترده قابلیت‌های آن، بلکه طراحی حرفه‌ای و سرعت بالای آن نسبت به دیگر کتابخانه‌ها است. از این مجموعه، کتابخانه‌های زیر در پروژه استفاده شده‌اند:

Container ○

Realtime ^۱

GUI ^۲

Backend ^۳

Widget ^۴

- Archive ○
- Serialization ○
- Array ○
- SmartPtr ○
- Move ○
- Thread ○
- Filesystem ○
- Chrono ○

همچنین از کلاس noncopyable برای پیاده‌سازی کلاس‌های یگانه^۱ استفاده شده است.

- کتابخانه [7] RtMidi: این کتابخانه برای ارتباط بلادرنگ با دستگاه‌های میدی استفاده می‌شود. از مزایای این کتابخانه سادگی رابط برنامه‌نویسی آن و همچنین پشتیبانی از سیستم‌های عامل مختلف است. یکی از نکات مثبت این کتابخانه برای پروژه، خاصیت چندنخی^۲ آن است، به این صورت که دریافت پیام از دستگاه در یک نخ جدا انجام می‌شود.
- کتابخانه [8] SDL2_mixer: این کتابخانه برای پردازش و پخش فایل‌های صوتی است. از ویژگی‌های مهم این کتابخانه، پشتیبانی از فرمت‌های مختلف صوتی از جمله MP3, FLAC, OGG, WAV و همین‌طور قابلیت اختصاص کانال صوتی به تعداد بسیار بالا و پخش با سرعت بالا است. به دلیل نیاز SDL2_mixer به کتابخانه SDL، این تکنولوژی نیز به پروژه اضافه شده است.
- GoogleTest Framework [9]: برای آزمون^۳ این پروژه استفاده شده است.
- نرم افزار [10] loopMIDI: به منظور تست بخش‌های مربوط به میدی در این پروژه، نیاز به باز کردن درگاه‌های^۴ میدی مجازی^۵ و اتصال برنامه تست به نرم افزار اصلی به عنوان شیه ساز سخت افزار است. loopMIDI این قابلیت را در سیستم عامل ویندوز فراهم می‌کند.

^۱ Singleton

^۲ Multithread

^۳ Testing

^۴ Port

^۵ Virtual

۵.۱ - هدف نهایی پروژه

هدف نهایی این پروژه طراحی و پیاده‌سازی یک نرم افزار دسکتاپ^۱ برای کار با دستگاه لانچپد است. این نرم افزار باید این نیازهای اصلی کاربر را برطرف کند:

۱. کاربر بتواند برای هر دکمه اصلی یک فایل صدا تنظیم کند که با فشار آن دکمه، از سیستم صوتی پخش شود.

۲. کاربر بتواند برای هر دکمه اصلی یک انیمیشن نمایش نور تنظیم کند که با فشار آن دکمه، این انیمیشن بر روی دکمه‌های لانچپد نمایش داده شود. هر انیمیشن شامل تعدادی فریم^۲ و هر فریم نشان‌دهنده وضعیت LEDهای هر دکمه و دارای یک مدت زمان است.

۳. دکمه کنترلی (در کل ۱۶ دکمه) صفحه فعال برنامه را تغییر می‌دهد، به این صورت که کاربر بتواند برای هر دکمه اصلی، ۱۶ تنظیمات مختلف را اعمال کند. این عملکرد را برای ادامه گزارش تغییر صفحه^۳ می‌نامیم.

این نرم افزار را SLPP^۴ می‌نامیم و در فصل بعدی، نیازمندی‌های نرم افزار را به طور دقیق‌تر بررسی و تعریف می‌کنیم.

^۱ Desktop

^۲ Frame

^۳ Page changing

^۴ Simple Launchpad Program

فصل دوم

طراحی نرم افزار

در این فصل، ابتدا نیازمندی‌های^۱ عملکردی^۲ و کیفی^۳ مورد نیاز را بررسی می‌کنیم، سپس بر اساس این نیازمندی‌ها، معماری نرم افزار و بعد از آن، یوزکیس‌های بخش‌های مختلف برنامه را تعریف می‌کنیم.

۱.۲ - نیازمندی‌ها

۱.۱.۲ - نیازمندی‌های عملکردی

۱. عملکردهای صفحه اصلی برنامه

این صفحه به صورت GUI دکمه‌های لاینچپد را به کاربر نمایش می‌دهد، عملکرد این دکمه‌ها به صورت زیر است.

a. کاربر باید بتواند با انتخاب یک دکمه اصلی، به بخش تنظیمات صدا^۴ برای آن دکمه منتقل شود.

b. کاربر باید بتواند با انتخاب یک دکمه اصلی، به بخش مدیریت انیمیشن^۵ برای آن دکمه منتقل

شود.

c. کاربر باید بتواند با انتخاب یک دکمه کنترلی، صفحه فعال^۶ نرم افزار را تغییر دهد.

۲. عملکردهای نوار ابزار^۷

این نوار ابزار به صورت گرافیکی در بالای صفحه به صورت یک نوار فهرست به کاربر نمایش داده

می‌شود. هر کدام از عملکردهای پایین، باید یک دکمه میانبر^۸ اختصاص یافته نیز داشته باشد.

^۱ Requirements

^۲ Functional

^۳ Non-functional

^۴ Audio Config

^۵ Animation Management

^۶ Active Page

^۷ Toolbar

^۸ Hotkey, Shortcut

- a. ذخیره پروژه بر روی فایل سیستم^۱. در صورتی که برای پروژه فعال تاکنون فایلی انتخاب نشده است، کاربر با استفاده از یک صفحه انتخاب فایل^۲ مسیر مورد نظر را انتخاب می کند.
- b. بارگذاری^۳ یک پروژه ذخیره شده در فایل سیستم. در صورت انتخاب این گزینه، به کاربر باید یک پیام تایید نشان داده شود، زیرا محتوای پروژه فعال در صورت ذخیره نبودن حذف خواهد شد.
- c. شروع یک پروژه جدید. در صورت انتخاب این گزینه، به کاربر باید یک پیام تایید نشان داده شود، زیرا محتوای پروژه فعال در صورت ذخیره نبودن حذف خواهد شد.
- d. بستن برنامه. در صورت انتخاب این گزینه، به کاربر باید یک پیام تایید نشان داده شود، زیرا محتوای پروژه فعال در صورت ذخیره نبودن حذف خواهد شد.
- e. باز کردن صفحه خلاصه^۴ پروژه.
- f. باز کردن صفحه انتخاب درگاه ورودی میدی.
- g. باز کردن صفحه انتخاب درگاه خروجی میدی.
- h. لغو^۵ یا تکرار^۶ آخرین تغییرات برنامه.
۳. عملکردهای صفحه خلاصه پروژه
- در این صفحه، کاربر می تواند تمامی ۱۶ صفحه پروژه و تنظیمات اعمال شده هر دکمه را مشاهده کند.
- a. هر دکمه باید وضعیت تنظیم بودن یا نبودن صدا را نمایش دهد.
- b. هر دکمه باید تعداد فریم های انیمیشن تنظیم شده را نمایش دهد.
- c. با کلیک بر روی هر دکمه، صدای تنظیم شده پخش می شود. دلیل اضافه کردن این عملکرد، بررسی وضعیت پروژه فعال بدون نیاز به اتصال لانچپد است.
۴. عملکردهای صفحات انتخاب درگاه ورودی و خروجی میدی
- a. در این صفحات، درگاه های باز میدی به کاربر نمایش داده می شود و کاربر می تواند یکی از این درگاه ها را انتخاب کند.

^۱ File System

^۲ File Dialog

^۳ Load

^۴ Summary Page

^۵ Undo

^۶ Redo

۵. عملکردهای صفحه تنظیم صدا

- a. انتخاب یک فایل صوتی از روی فایل سیستم و بارگذاری آن فایل برای دکمه انتخاب شده.
- فایل‌های WAV, OGG, MP3, FLAC باید پشتیبانی شوند.
- b. تنظیم تکرار صدا. اگر این گزینه فعال باشد، صدا به صورت مکرر تا زمان دوباره فعال شدن دکمه پخش می‌شود.
- c. حذف صدا. با انتخاب این گزینه، اگر برای دکمه انتخاب شده صدایی تنظیم شده باشد، حذف می‌شود.

۶. عملکردهای صفحه مدیریت انیمیشن

- a. جا به جایی بین فریم‌ها. کاربر باید بتواند با استفاده از رابط گرافیک، به فریم‌های قبلی، بعدی، اولین فریم و آخرین فریم حرکت کند. همچنین باید بتواند شماره فریم مورد نظر را مستقیم وارد کند.
- b. حذف یا اضافه کردن یک فریم. اگر کاربر در حال مشاهده آخرین فریم باشد، با انتخاب گزینه فریم بعدی، به صورت خودکار یک فریم به انیمیشن‌ها اضافه می‌شود. همچنین گزینه‌هایی مختص اضافه کردن یا حذف کردن فریم در اختیار کاربر قرار دارد.
- c. تنظیم مدت زمان یک فریم. این زمان به صورت یک عدد اعشاری با واحد ثانیه تنظیم می‌شود و نشان‌دهنده طول فعال بودن این فریم بر روی لانچپد است.
- d. تنظیم نور هر دکمه در یک فریم. کاربر در این صفحه برای هر فریم می‌تواند نور هر ۶۴ دکمه لانچپد را تنظیم کند. گزینه‌هایی که کاربر می‌تواند انتخاب کند باید با قابلیت‌های لانچپد (موجود در فایل launchpad programmer's reference) هماهنگ باشد. این گزینه‌ها شامل:
- قرمز، زرد، کهربایی و سبز است. همچنین کاربر می‌تواند انتخاب کند که این نور خاموش شود یا وضعیت آن تغییری نکند. در نتیجه به طور کل ۶ گزینه مختلف در اختیار کاربر قرار می‌گیرد.
- e. نمایش و انتخاب افزونه‌های انیمیشن^۱. کاربر باید لیست افزونه‌ها را مشاهده و از این لیست گزینه مورد نظر را انتخاب کند. افزونه‌ها را در بخش‌های بعدی بررسی می‌کنیم.

۷. عملکردهای لانچپد

- a. تغییر صفحه فعال برنامه. علاوه بر قابلیت تغییر صفحه فعال با استفاده از رابط کاربری گرافیکی در صفحه اصلی، با فشار دادن دکمه‌های کنترلی لانچپد، این صفحه فعال تغییر می‌کند.

^۱ Animation Plugins

b. پخش صدا. با فشار دادن یک دکمه اصلی در لانچپد، در صورت تنظیم بودن صدا برای آن دکمه در صفحه فعال، باید از سیستم صوتی پخش شود. در صورتی که حالت تکرار فعال باشد، با فشار دوباره دکمه، صدا قطع می شود.

c. پخش انیمیشن. با فشار دادن یک دکمه اصلی در لانچپد، انیمیشن نور مربوط به آن دکمه در صفحه فعال بر روی لانچپد نمایش داده می شود.

۲.۱.۲ - نیازمندی های کیفی

این نیازمندی ها، علاوه بر مشخص کردن معیارهای کیفیت برنامه، مسیر طراحی معماری نرم افزار را نیز مشخص می کنند.

۱. عملکرد و بازدهی^۱

a. توالی مسیر پخش یک صدا باید تا حد ممکن بهینه^۲ شده و حداقل تاخیر ممکن را داشته باشد. این مسیر با دریافت پیام میدی از سمت لانچپد شروع و با ارسال داده صدا به کارت صدا و سیستم صوتی پایان میابد.

b. توالی مسیر پخش یک انیمیشن باید تا حد ممکن بهینه^۳ شده و حداقل تاخیر ممکن را داشته باشد. این مسیر با دریافت پیام میدی از سمت لانچپد شروع و با ارسال داده انیمیشن به صورت پیام میدی به لانچپد پایان میابد.

c. فاصله زمانی فریم ها باید حداقل خطا را نسبت به بازه زمانی تنظیم شده توسط کاربر داشته باشند. نکته قابل توجه درباره انیمیشن ها و صداها این است که امکان پخش چند انیمیشن و صدا به صورت همزمان با هم وجود دارد.

۲. استحکام^۴

در صورت وقوع مشکلات ورودی و خروجی داده، از سمت فایل سیستم و یا از سمت لانچپد، روند اجرای برنامه نباید مختل شود. همچنین در صورت نیاز کاربر باید از وقوع این مشکلات باخبر شود.

۳. قابلیت استفاده^۵

^۱ Performance

^۲ Optimized

^۳ Optimized

^۴ Robustness

^۵ Usability

a. کاربر باید بتواند به راحتی تمام از این نرم افزار استفاده کند. هدف اصلی این پروژه ارائه یک

رابط کاربری مناسب و حل مشکلات مربوط به پیچیدگی DAW ها است.

b. کاربر باید بتواند تغییرات مربوط به انیمیشن را به سرعت و راحتی انجام دهد.

۴. قابلیت انتقال^۱

در صورت نیاز، برنامه امکان خروجی گرفتن برای سیستم های عاملی به جز ویندوز را داشته باشد. انتخاب

کتابخانه های کراس پلفرم^۲ این امر را فراهم می کند.

۵. توسعه پذیری^۳

با توجه به نیازهای استفاده کاربر و اهداف پروژه، ابزارهای مربوط به کار با انیمیشن ها باید به سادگی و

سرعت به نرم افزار اضافه شوند.

۲.۲ معماری نرم افزار

طراحی های انجام شده برای معماری این پروژه، با توجه به نیازمندی های مختلف و با استفاده از استاندارد C4

model در سه سطح مختلف انجام و آماده شده است [11].

۱.۲.۲ - انتخاب معماری مناسب

با توجه به ویژگی ها و نیازمندی های مطرح شده در بخش قبل، معماری انتخاب شده ترکیبی از معماری میکرو کرنل^۴

و معماری لایه ای^۵ است [12] [13]. در معماری میکرو کرنل، نرم افزار از دو بخش اصلی هسته و افزونه ها تشکیل

شده است. در هسته این معماری از طراحی لایه ای استفاده شده است. دلیل استفاده از معماری میکرو کرنل، نیازمندی

توسعه پذیری است. با توجه به اینکه نیاز کاربران نسبت به کار با انیمیشن ها در این نرم افزار می تواند به سرعت تغییر

کند، ابزارهای مربوط به کار با انیمیشن به صورت افزونه هایی خارج از هسته اصلی برنامه پیاده سازی می شوند. در این

معماری، پیاده سازی افزونه ها می تواند به دو صورت مبتنی بر کامپایل^۶ یا مبتنی بر زمان اجرا^۷ انجام شود. تفاوت این

دو روش در پیچیدگی استقرار^۸ و دشواری پیاده سازی آنها است. در روش مبتنی بر کامپایل، افزونه ها در کد اصلی

Portability^۱

Cross-Platform^۲

Extensibility^۳

Microkernel Architecture^۴

Layered Architecture^۵

Compile-Based^۶

Runtime-Based^۷

Deployment^۸

برنامه در کنار هسته قرار دارند و در صورت تغییر مجموعه افزونه‌ها، تمامی برنامه نیاز به استقرار دوباره^۱ دارد، در عین حال این روش سادگی و سرعت پیاده‌سازی بسیار بالاتری نسبت به روش مبتنی بر زمان اجرا دارد. در روش زمان اجرا، در صورت تغییر افزونه، نیازی به استقرار دوباره نیست و تغییرات به راحتی در نرم افزار در حال اجرا اعمال می‌شود، اما این روش پیچیدگی پیاده‌سازی بیشتری برای توسعه‌دهنده نسبت به روش دیگر دارد. به دلیل حجم پایین و سادگی نصب این پروژه و همینطور نیاز به سرعت بالای توسعه افزونه‌ها، از روش مبتنی بر کامپایل استفاده شده است.

جدول ۱ ویژگی‌های معماری میکرو کرنل [12]

Architecture characteristic	Star rating
Partitioning type	Domain and technical
Number of quanta	1
Deployability	☆☆☆
Elasticity	☆
Evolutionary	☆☆☆
Fault tolerance	☆
Modularity	☆☆☆
Overall cost	☆☆☆☆☆
Performance	☆☆☆
Reliability	☆☆☆
Scalability	☆
Simplicity	☆☆☆☆
Testability	☆☆☆

همانطور که اشاره شد، در هسته این پروژه از معماری لایه‌ای استفاده شده است. دلیل این انتخاب، سادگی نرم افزار در حال توسعه به لحاظ تعداد اجزاء^۲ و ارتباط آن‌ها و همچنین هزینه‌های پایین مربوط به این معماری است. همچنین با قراردادن اجزاء مرتبط با هم در یک لایه معماری و به حداقل رساندن فاصله آن‌ها، ویژگی‌های کیفی بازدهی نیز فراهم می‌شوند.

^۱ Redeployed

^۲ Components

جدول ۲ ویژگی‌های معماری لایه‌ای [13]

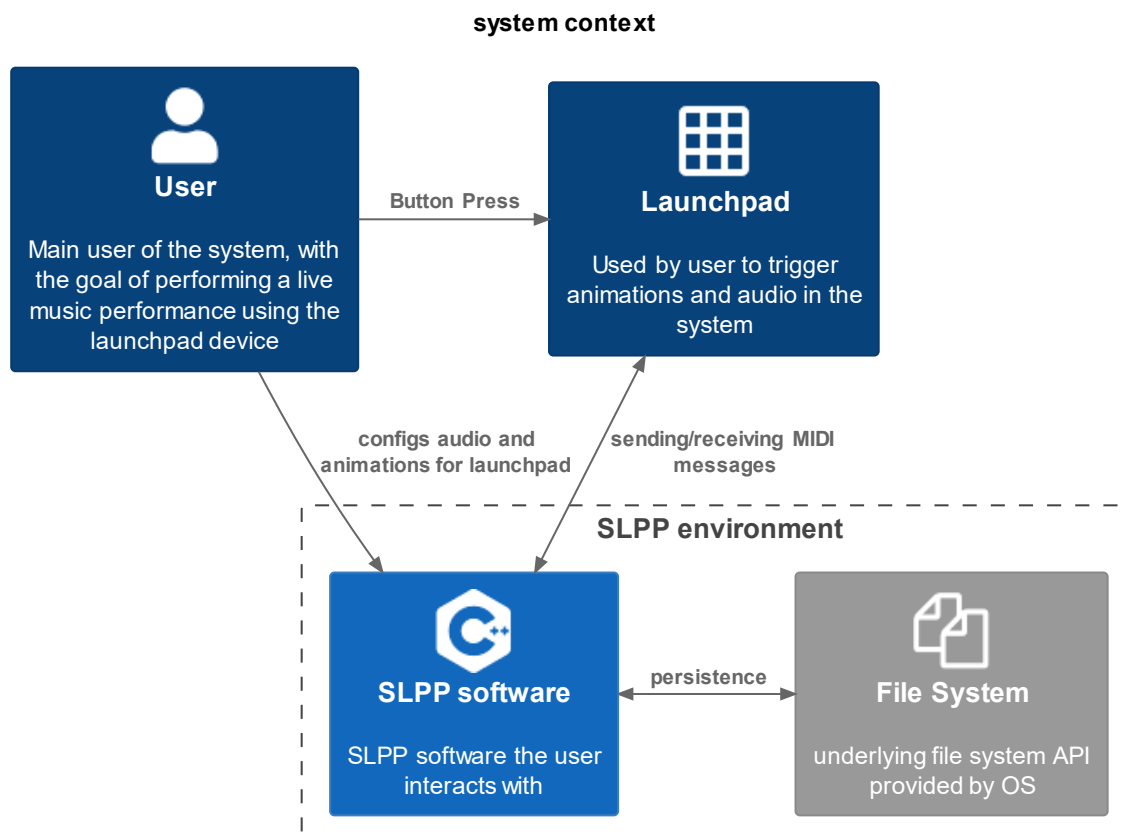
Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1
Deployability	★
Elasticity	★
Evolutionary	★
Fault tolerance	★
Modularity	★
Overall cost	★★★★★
Performance	★★
Reliability	★★★
Scalability	★
Simplicity	★★★★★
Testability	★★

۲.۲.۲ - زمینه سیستم^۱

در این سطح، یک طراحی کلی از سیستم و ارتباط آن با بازیگران^۲ و سیستم‌های خارجی انجام می‌شود.

^۱ System Context

^۲ Actors



شکل ۲ طراحی کلی سیستم

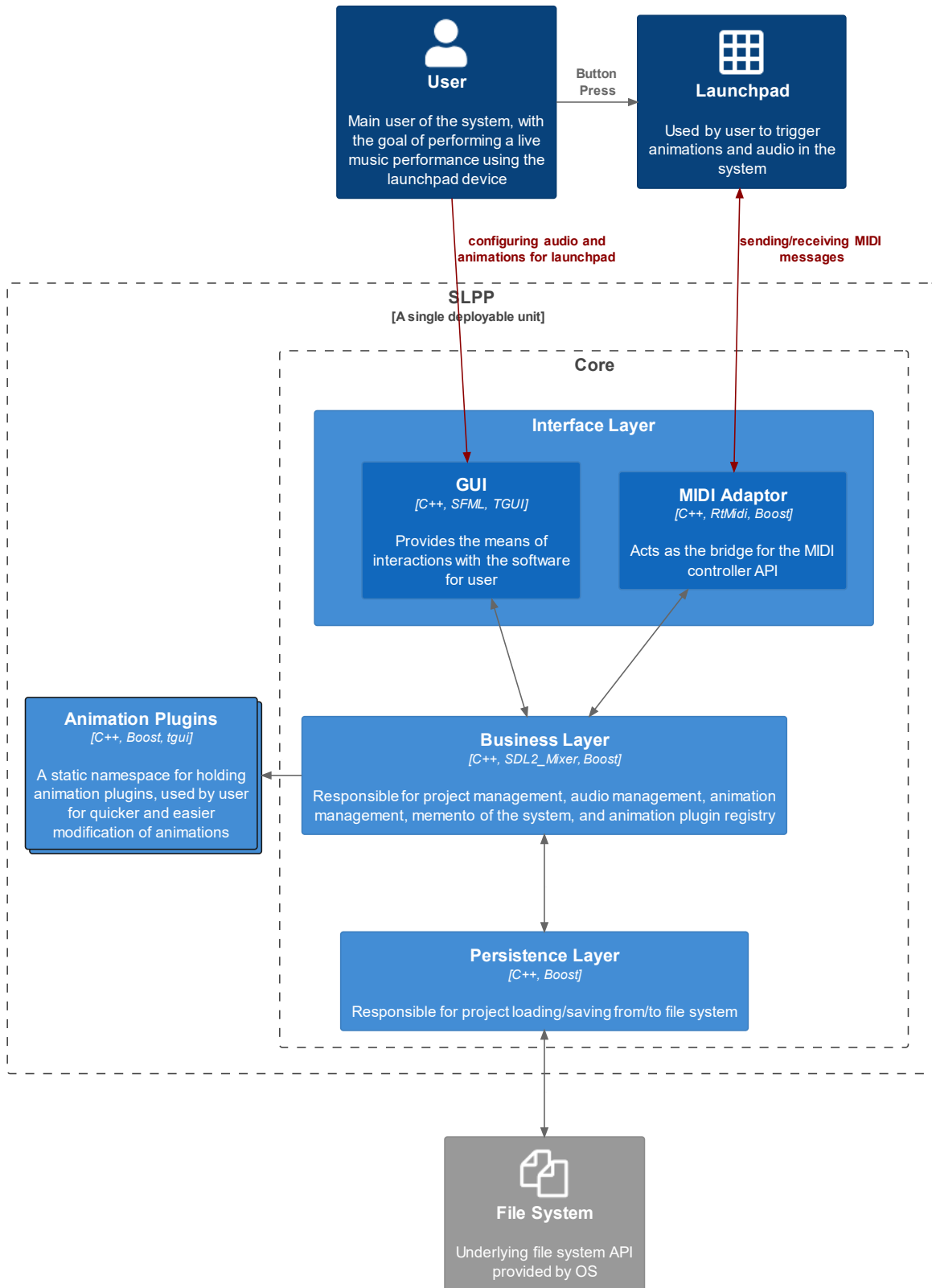
همانطور که در تصویر مشخص است، هدف اصلی این طراحی تشخیص روابط و وابستگی‌ها بین بازیگران و نرم افزار و همینطور با سیستم‌های دیگر است. نرم افزار SLPP، دو بازیگر اصلی، کاربر و لانچپد، و همینطور به دلیل نیازمندی‌های عملکردی، وابستگی به فایل سیستم دارد.

۳.۲.۲ - مخازن سیستم

در این سطح، مخازن^۱ سیستم مشخص می‌شوند. در هسته، هر مخزن نشان‌دهنده یک لایه و وظایف آن است.

^۱ Containers

Container Architecture of SLPP System



شکل ۳ مخازن سیستم SLPP

همانطور که در تصویر مشخص است، این طراحی معماری سیستم را واضح تر نشان می دهد. هسته این سیستم متشکل از ۳ لایه اصلی است که وظایف این لایه ها را بررسی می کنیم.

۱. لایه رابط کاربری^۱: وظیفه برقراری ارتباط با بازیگران سیستم و انتقال ورودی های بازیگران به لایه پایینی را دارد. به دلیل وجود دو بازیگر کاملاً متفاوت، این لایه به دو بخش اصلی تشکیل شده است. بخش GUI وظیفه برقراری ارتباط با کاربر نرم افزار و بخش MIDI وظیفه برقراری ارتباط با دستگاه لانیچد را دارد. ۲. لایه منطق یا بیزینس^۲: وظیفه نگهداری منطق اصلی برنامه و برقراری ارتباط اجزاء اصلی با یکدیگر را دارد. تمامی عملکرد اصلی نرم افزار در این لایه پیاده سازی می شود که در سطح اجزاء به بررسی آنها می پردازیم.

۳. لایه ماندگاری^۳: وظیفه ارتباط با فایل سیستم برای ذخیره یا بارگذاری پروژه های کاربر را دارد.

همچنین خارج از این هسته، یک مخزن برای نگهداری افزونه ها در نظر گرفته شده است.

۴.۲.۲ - اجزاء^۴ سیستم

در این سطح، اجزاء تشکیل دهنده هر مخزن، وظایف و ارتباط بین لایه ای آنها بررسی می شوند، همچنین درباره نحوه عملکرد این اجزاء، نحوه ارتباط آنها با دیگر اجزاء در لایه های دیگر (برای کاهش پیچیدگی تصویر) و پایین عنوان، تکنولوژی های استفاده شده مشخص است. به دلیل محتوای زیاد، تصویر این سطح به دو بخش تقسیم شده است. بخش اول، پنجره اصلی و همچنین همه اجزاء مربوط به عملیات های پروژه (مانند ذخیره، بارگذاری، Undo، redo) در همه لایه ها را نشان می دهد. بقیه اجزاء در بخش دوم نمایش داده شده اند.

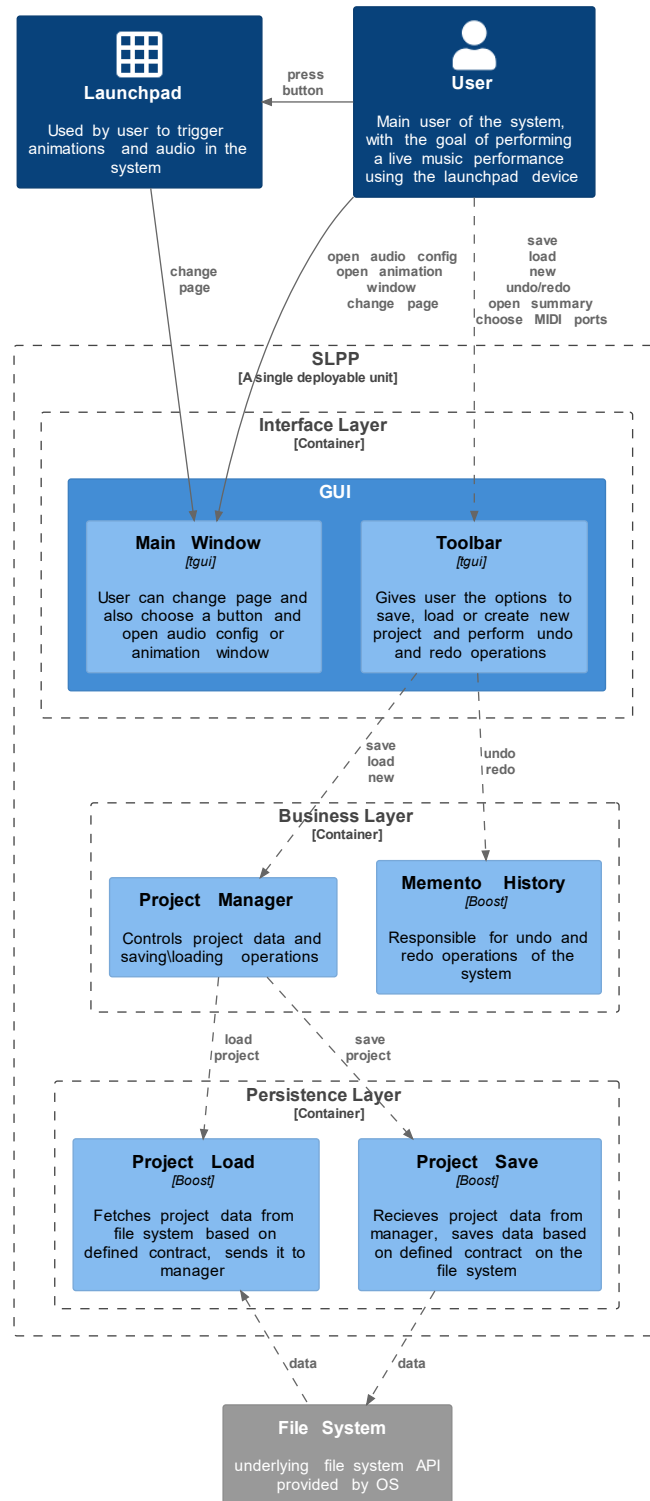
^۱ Interface

^۲ Business

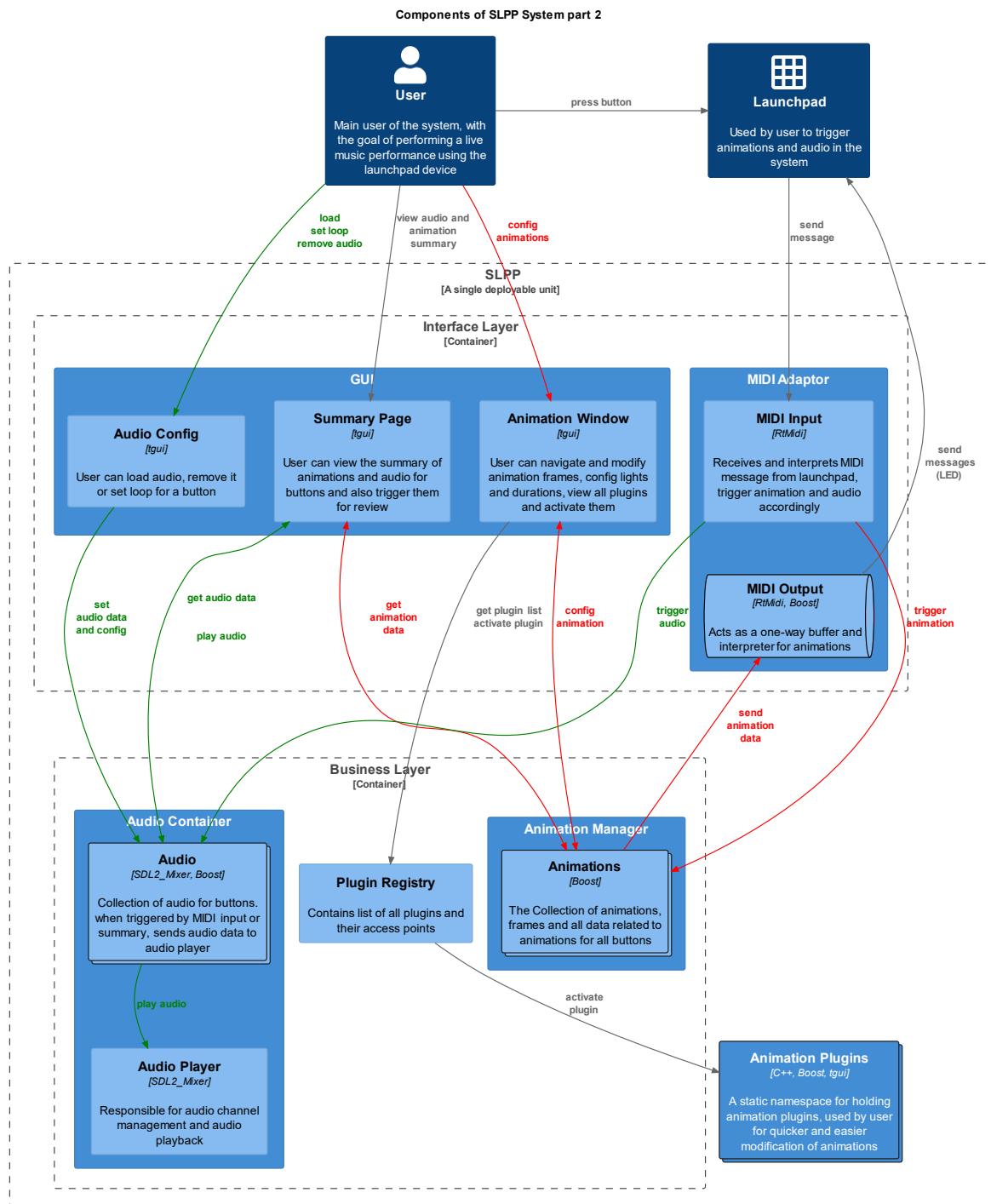
^۳ Persistence

^۴ Components

Components of SLPP System part 1



شکل ۴ اجزاء سیستم، بخش ۱



شکل ۵ اجزاء سیستم، بخش ۲

در لایه کاربری، بخش رابط کاربری گرافیکی، ۵ جزء اصلی وجود دارد

۱. پنجره اصلی: نمایش صفحه اصلی برنامه که شامل دکمه‌های تغییر صفحه لانچپد و دکمه‌های اصلی برای

باز کردن صفحه تنظیم صدا و صفحه انیمیشن است.

۲. نوار ابزار: نمایش گزینه‌های برنامه به کاربر

۳. تنظیم صدا: نمایش گزینه‌های مربوط به تنظیمات صدا
۴. صفحه خلاصه: نمایش خلاصه انیمیشن‌ها و صداها در پروژه
۵. پنجره انیمیشن: مدیریت انیمیشن و نمایش و انتخاب افزونه‌ها

همچنین در این لایه برای ارتباط با لانچپد بخش مبدل^۱ میدی قرار دارد که شامل ۲ جزء اصلی است:

۱. ورودی: دریافت پیام از لانچپد و فراخوانی دستورات صدا و انیمیشن مربوطه
۲. خروجی: دریافت اطلاعات انیمیشن و ارسال آن به لانچپد. این بخش سیستم به شکل یک بافر^۲ یک طرفه برای ارسال انیمیشن‌ها به لانچپد عمل می‌کند.

در لایه بیزینس، اجزاء اصلی منطق برنامه قرار دارند

۱. مدیر پروژه: ارتباط با لایه ماندگاری برای ذخیره و بارگذاری پروژه و همچنین ساخت یک پروژه جدید.
 ۲. تاریخچه یادمان‌ها^۳: ذخیره تاریخچه تغییرات اعمال شده و همینطور بازگردانی این تاریخچه برای لغو یا تکرار.
 ۳. مخزن صداها: شامل اطلاعات صداها و مربوط به دکمه‌های لانچپد و همینطور پخش‌کننده صدا^۴. هر دکمه اصلی لانچپد دارای یک صدا است، با وجود ۱۶ صفحه و ۶۴ دکمه، این مجموعه به ۱۰۲۴ صدا می‌رسد.
 ۴. مدیریت انیمیشن‌ها: شامل اطلاعات تمامی انیمیشن‌ها، فریم‌ها و اطلاعات مربوط به آن‌هاست. هر دکمه اصلی لانچپد انیمیشن خود را دارد، با وجود ۱۶ صفحه و ۶۴ دکمه، این مجموعه به ۱۰۲۴ انیمیشن می‌رسد.
- هر انیمیشن دارای تعدادی فریم است، این تعداد توسط کاربر تنظیم می‌شود. هر فریم اطلاعات LED تمامی ۶۴ دکمه لانچپد را مشخص می‌کند و دارای یک طول زمانی است که نشان می‌دهد این فریم چند ثانیه فعال است.

۵. فهرست افزونه‌ها^۵: شامل لیست تمامی افزونه‌ها و همچنین راه فعال‌سازی آن‌هاست.

لایه ماندگاری تنها دو جزء ذخیره و بارگذاری دارد.

^۱ Adaptor

^۲ Buffer

^۳ Memento

^۴ Audio Player

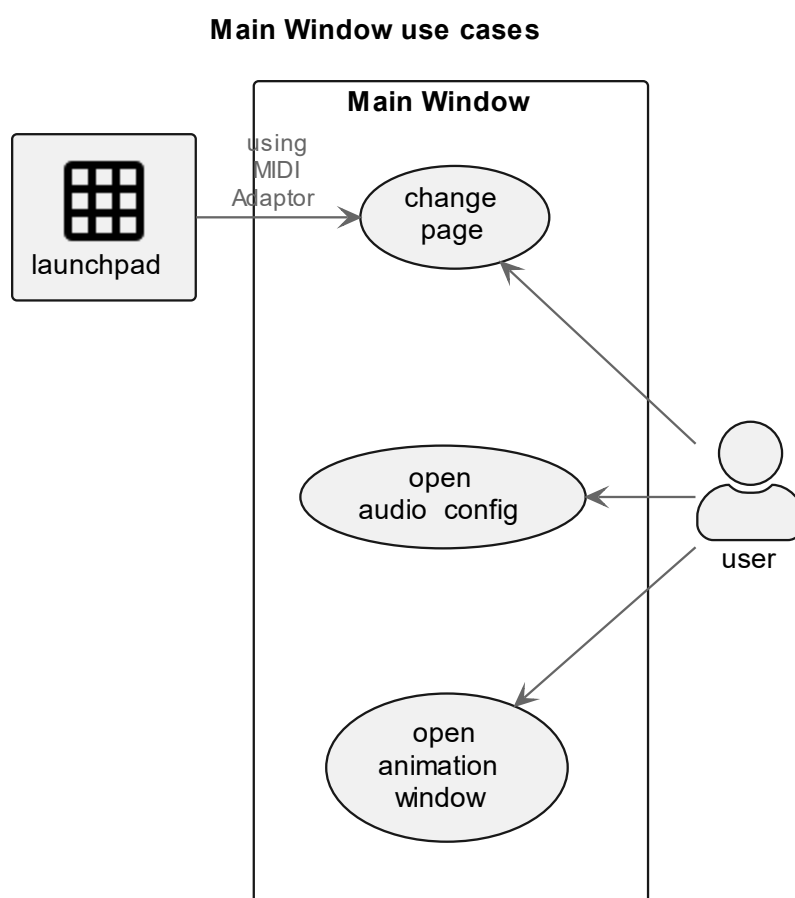
^۵ Plugin Registry

خارج از این لایه‌ها و هسته سیستم، مجموعه افزونه‌ها قرار دارد. این افزونه‌ها تا حد ممکن مستقل از خود سیستم طراحی شده‌اند. برخی از این افزونه‌ها نیاز به دریافت ورودی از کاربر برای انجام عملیات تعریف شده دارند، به همین دلیل برای کاهش جفت‌شدگی^۱ بین این مجموعه و سایر سیستم، دسترسی مستقیم به tgui به افزونه‌ها داده شده تا نیاز به ارتباط با لایه Interface نداشته باشند.

۳.۲- موارد کاربری

برای دید بهتر نسبت به نحوه پیاده‌سازی این نرم افزار، نیاز است برای هر جزء سیستم، موارد کاربری^۲ تعریف کنیم.

۱.۳.۲- پنجره اصلی



شکل ۶ موارد کاربری پنجره اصلی

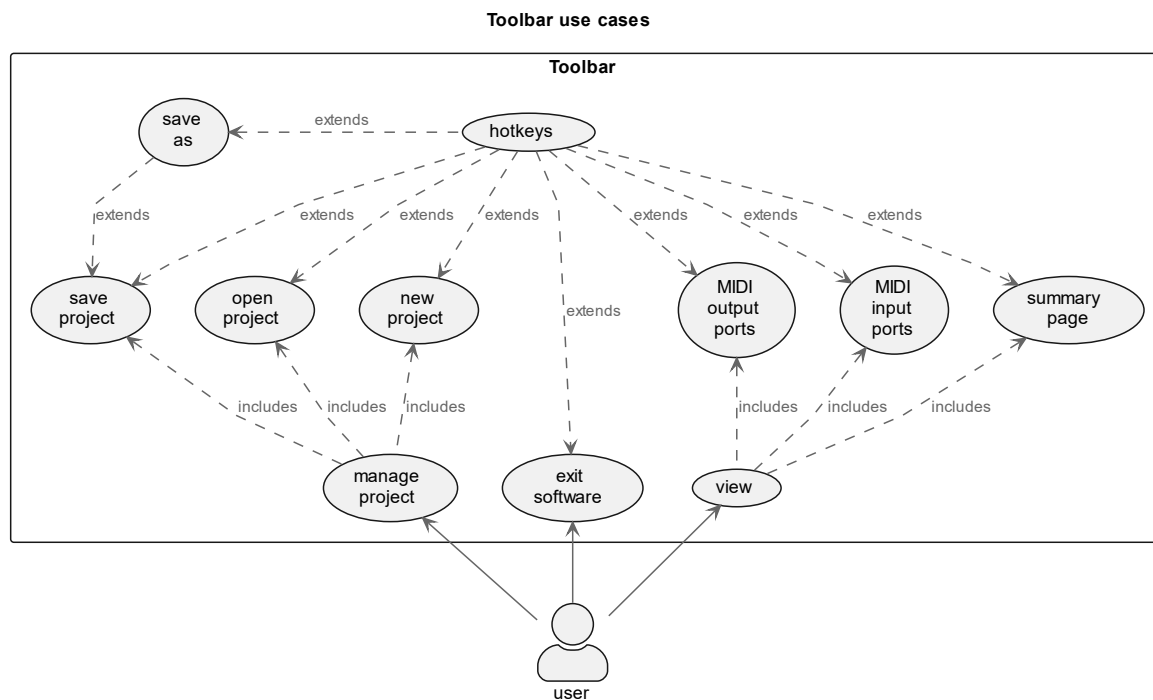
پنجره اصلی اولین بخش از رابط کاربری گرافیکی است که کاربر با آن مواجه می‌شود. این صفحه شامل ۱۶ دکمه تغییر صفحه لانچپد و ۶۴ دکمه اصلی است. با انتخاب دکمه تغییر صفحه، صفحه فعال برنامه تغییر می‌کند. با انتخاب

^۱ Coupling

^۲ Use Cases

دکمه اصلی کاربر می تواند صفحه تنظیم صدا یا صفحه انیمیشن را باز کند. همچنین لانچر از طریق مبدل میدی می تواند صفحه فعال برنامه را تغییر دهد.

۲.۳.۲ – نوار ابزار



شکل ۷ موارد کاربری نوار ابزار

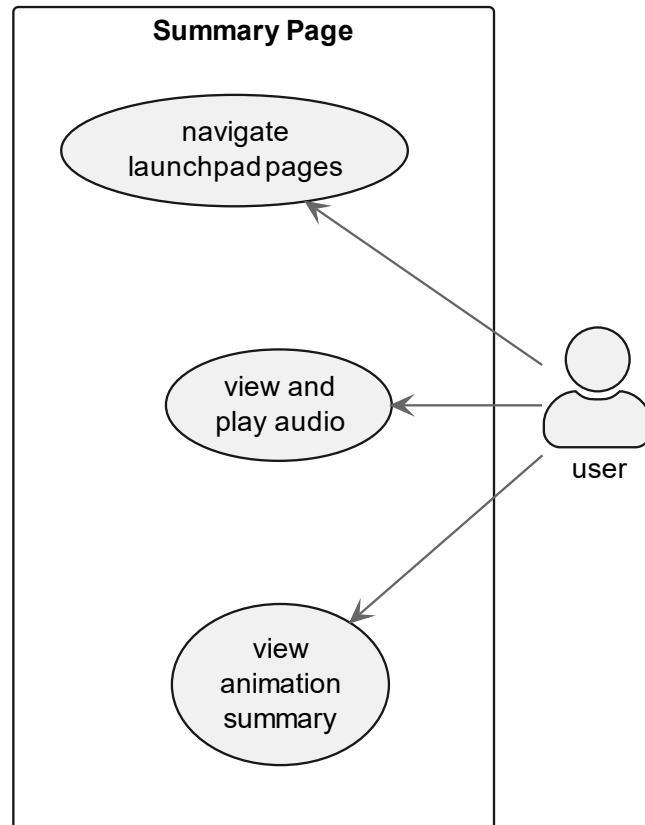
۱. مدیریت پروژه: این مورد شامل ذخیره پروژه، ساخت پروژه جدید و باز کردن یک پروژه ذخیره شده است، همچنین مورد ذخیره پروژه می تواند در یک فایل جدید باشد.

۲. خروج از نرم افزار

۳. مشاهده: این مورد شامل صفحات خلاصه، انتخاب درگاه ورودی میدی و خروجی میدی است.

نیاز است برای همه این موارد علاوه بر دسترسی به صورت گرافیکی یک دکمه میانبر نیز تعریف شود.

Summary Page use cases



شکل ۸ موارد کاربری صفحه خلاصه

این صفحه خلاصه‌ای از تمامی محتویات پروژه داخل برنامه را نشان می‌دهد، از جمله صداهای بارگذاری شده و تعداد فریم‌های انیمیشن هر دکمه. نحوه نمایش به صورت یک صفحه ۸ در ۸ از دکمه‌های گرافیکی است.

۱. جا به جایی بین صفحات لانچپد: کاربر باید بتواند همه ۱۶ صفحه لانچپد را مشاهده کند. طراحی این

عملکرد بهتر است با دکمه‌های گرافیکی صورت بگیرد

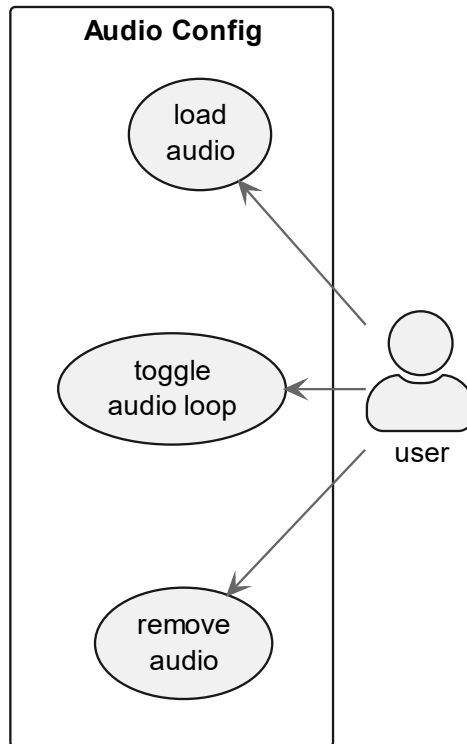
۲. مشاهده و پخش صدا: برای هر دکمه، اطلاعات مربوط به بارگذاری صدا باید نمایش داده شود، همچنین

کاربر باید بتواند با کلیک بر روی دکمه‌های این صفحه صدای بارگذاری شده را پخش کند.

۳. مشاهده اطلاعات انیمیشن: برای هر دکمه، تعداد فریم‌های انیمیشن مربوط به آن دکمه باید نمایش داده

شود.

Audio Config use cases



شکل ۹ موارد کاربری تنظیم صدا

این یک صفحه ساده است که با انتخاب یک دکمه در صفحه اصلی برنامه باز می شود تا کاربر بتواند تنظیمات صدای مربوط به آن دکمه را انجام دهد.

۱. بارگذاری صدا: کاربر می تواند یک فایل صوتی را انتخاب و بارگذاری کند، این عملکرد با استفاده از یک

صفحه انتخاب فایل^۱ صورت می گیرد. در صورتی که در بارگذاری فایل انتخاب شده خطایی رخ دهد،

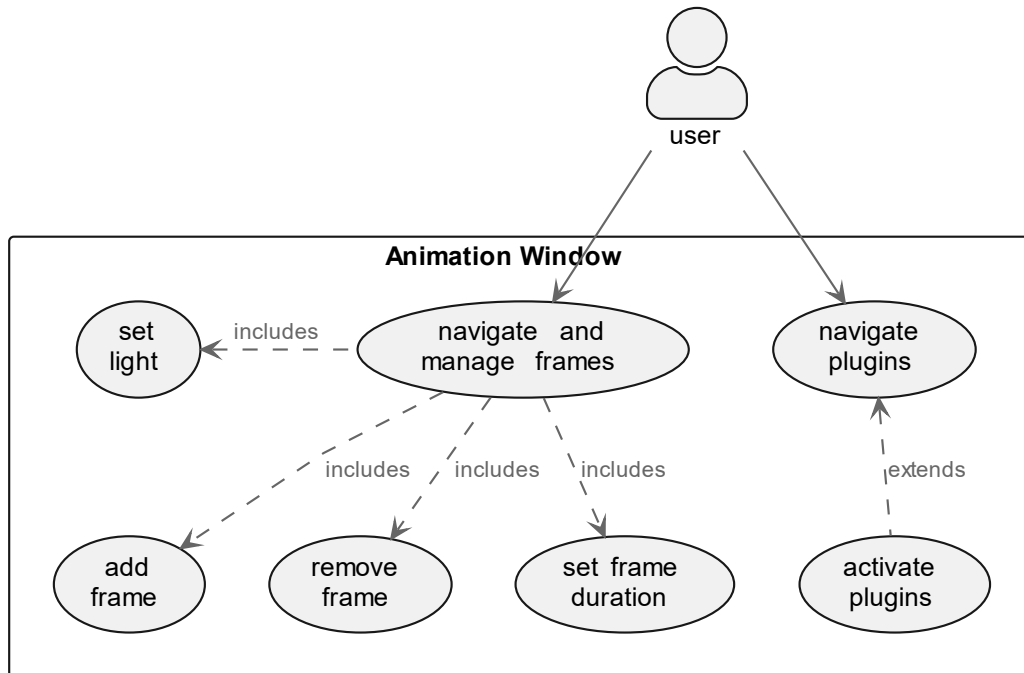
کاربر باید از این خطا مطلع شود. فایل های پشتیبانی شده عبارت اند از: MP3, FLAC, WAV, OGG

۲. تنظیم تکرار صدا: کاربر می تواند تکرار صدا را تنظیم کند.

۳. حذف صدا: کاربر می تواند یک صدای بارگذاری شده را حذف کند.

^۱ File Dialog

Animation Window use cases



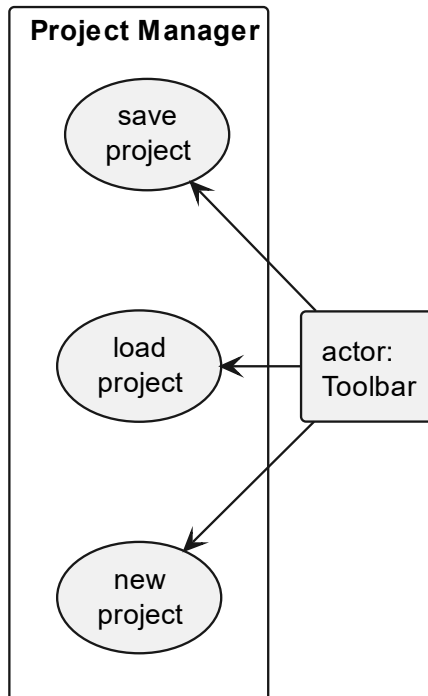
شکل ۱۰ موارد کاربری پنجره انیمیشن

کاربر می تواند با انتخاب یک دکمه در صفحه اصلی، این پنجره را باز کند.

۱. تنظیم فریم ها: کاربر باید بتواند برای هر فریم اطلاعات نور (LED) هر دکمه و طول زمانی آن فریم را تنظیم کند و در صورت نیاز یک فریم اضافه یا حذف کند.
۲. افزونه ها: کاربر باید بتواند لیست تمامی افزونه ها را مشاهده و در صورت نیاز یک افزونه را انتخاب و فعال کند.

۶.۳.۲ - مدیر پروژه

Project Manager use cases

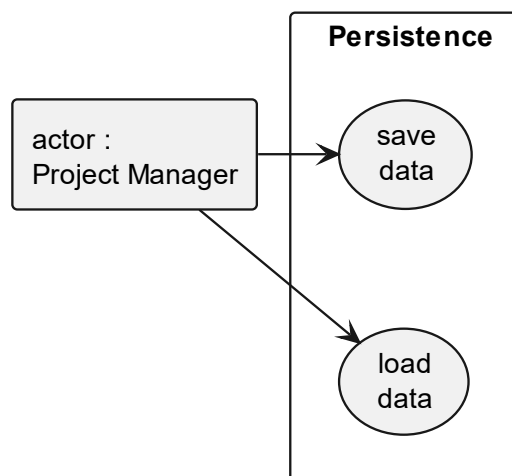


شکل ۱۱ موارد کاربری مدیر پروژه

همانطور که در تصور مشخص است، تنها بازیگر مربوط به مدیر پروژه، نوار ابزار است. تمامی عملیات‌های مربوط به یک پروژه توسط مدیر پروژه انجام می‌شود.

۷.۳.۲ - ماندگاری

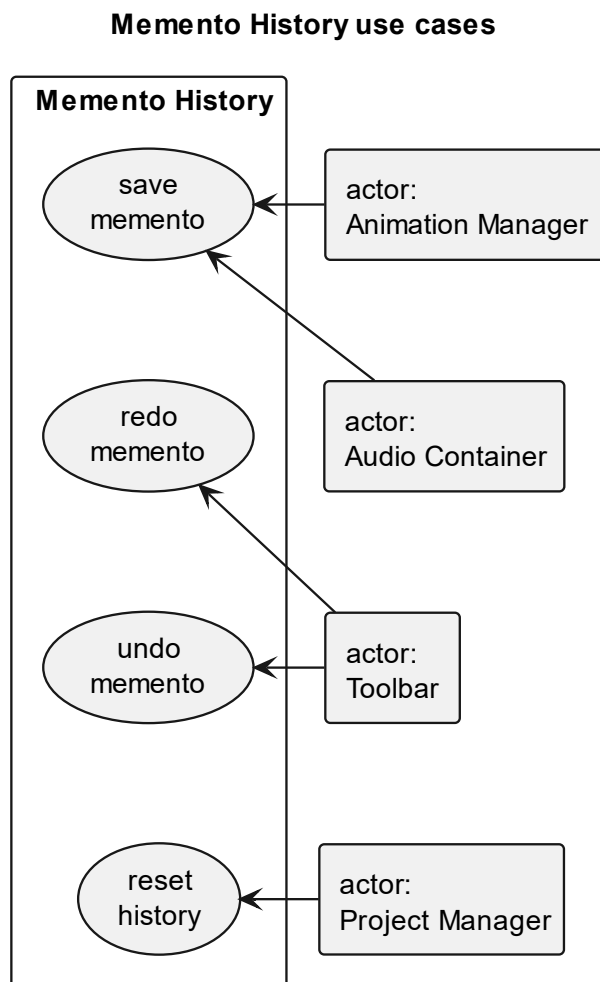
Persistence use cases



شکل ۱۲ موارد کاربری ماندگاری

پایین ترین لایه هسته که وظیفه آن ذخیره یا بارگذاری اطلاعات است که مدیر پروژه به عنوان رابط آن با لایه بالایی عمل می کند.

۸.۳.۲ - تاریخچه یادمان ها



شکل ۱۳ موارد کاربری تاریخچه یادمان ها

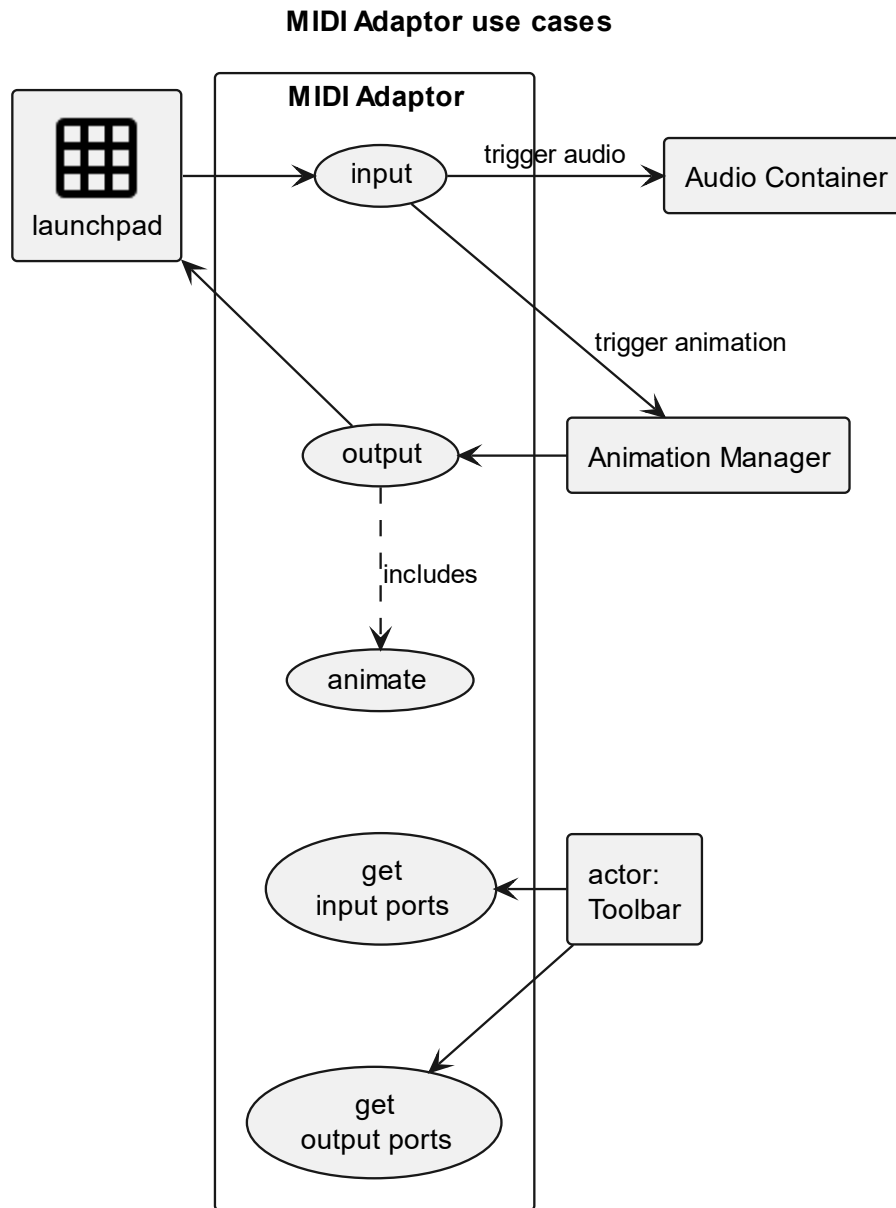
این تاریخچه قابلیت ذخیره تصاویر لحظه ای^۱ برای مدیریت انیمیشن ها و مخزن صداها را فراهم می کند. همچنین کاربر می تواند از طریق نوار ابزار، در این تاریخچه حرکت و عملیات های انجام شده را لغو^۲ یا تکرار^۳ کند. همچنین در موارد تغییر پروژه، مدیر پروژه می تواند این تاریخچه را بازنشانی^۴ کند.

^۱ Snapshots

^۲ Undo

^۳ Redo

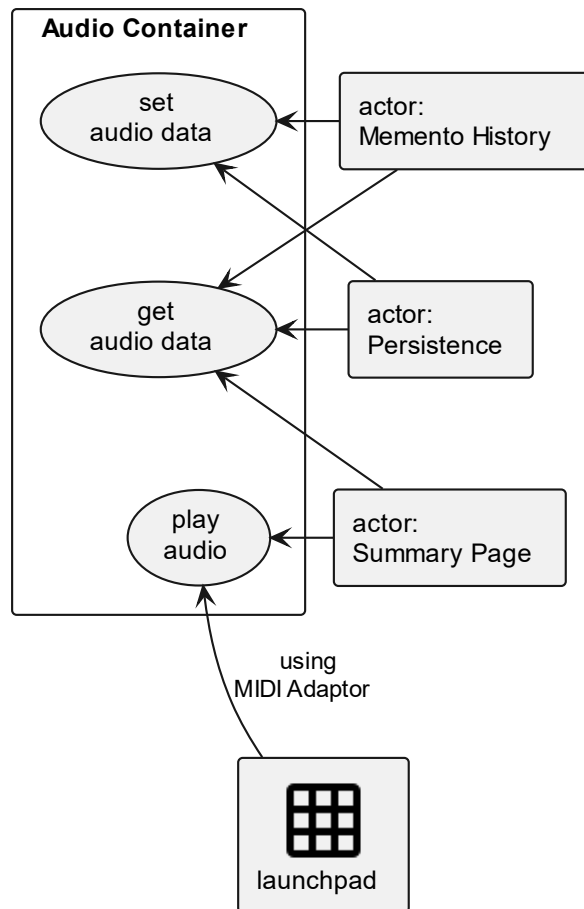
^۴ Reset



شکل ۱۴ موارد کاربری مبدل میدی

یکی از پیچیده‌ترین اجزاء سیستم از لحاظ پیاده‌سازی است. مبدل میدی پیام‌های لانچپد را دریافت و براساس پروتکل‌های لانچپد، آن‌ها را تفسیر کرده و عملیات‌های مورد نیاز را بر روی مخزن صداها یا مدیریت انیمیشن‌ها اجرا می‌کند. این مبدل اطلاعات انیمیشن‌ها را از مدیریت انیمیشن‌ها دریافت و پس از زمانبندی مناسب، به پیام‌های میدی تبدیل کرده و به لانچپد ارسال می‌کند. همچنین این مبدل لیست درگاه‌های ورودی و خروجی باز سیستم را برای نمایش به کاربر در بخش نوار ابزار فراهم می‌کند.

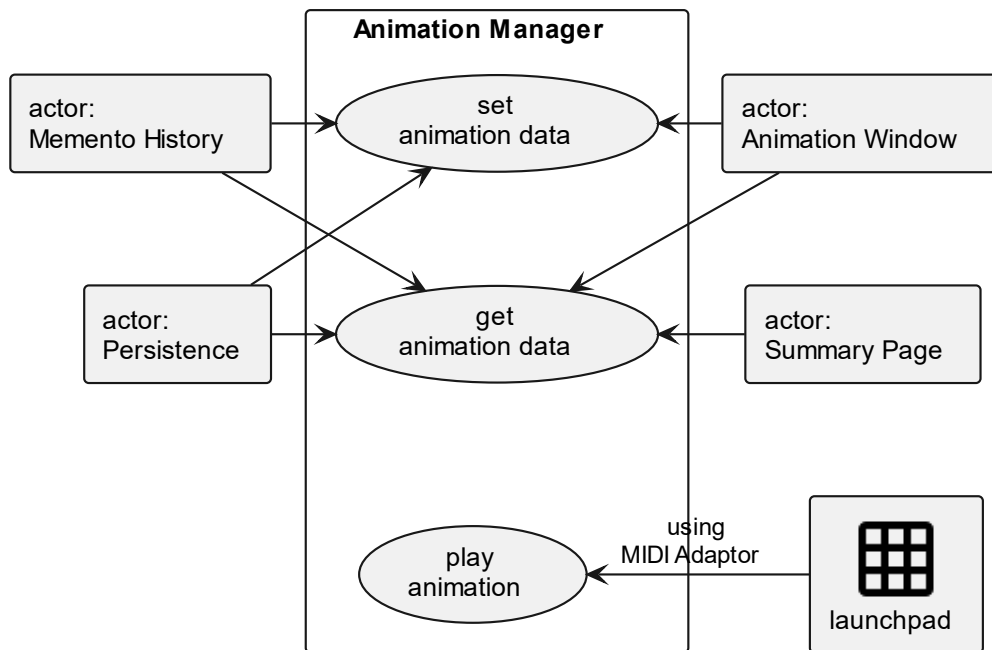
Audio Container use cases



شکل ۱۵ موارد کاربری مخزن صداها

این مخزن شامل اطلاعات تمامی صداها و نرم افزار برای تمامی ۱۶ صفحه و ۶۴ دکمه (در مجموعه ۱۰۲۴ صدا) است. بخش ماندگاری و همینطور تاریخچه یادمان‌ها برای ذخیره و بارگذاری اطلاعات از موارد تنظیم و دریافت اطلاعات این مخزن استفاده می‌کنند. دریافت اطلاعات توسط صفحه خلاصه نیز استفاده می‌شود تا اطلاعات مخزن را به کاربر نمایش دهد. قابلیت پخش صدا در این مخزن توسط لاینچپد (با واسطه مبدل میدی) و صفحه خلاصه استفاده می‌شود.

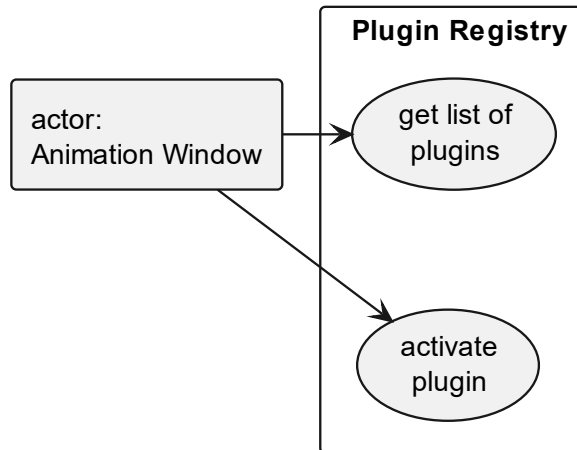
Animation Manager use cases



شکل ۱۶ موارد کاربری مدیر انیمیشن

این بخش شامل اطلاعات تمامی انیمیشن‌های نرم افزار برای تمامی ۱۶ صفحه و ۶۴ دکمه (در مجموعه ۱۰۲۴ انیمیشن) است. بخش ماندگاری و همینطور تاریخچه یادمان‌ها برای ذخیره و بارگذاری اطلاعات از موارد تنظیم و دریافت اطلاعات استفاده می‌کنند. صفحه انیمیشن برای نمایش اطلاعات به کاربر و همچنین به‌روزرسانی عملیات‌های کاربر بر روی انیمیشن‌ها، از موارد تنظیم و دریافت اطلاعات استفاده می‌کنند. لانیچپد (با واسطه مبدل میدی) می‌تواند یک انیمیشن را فعال کند.

Plugin Registry use cases



شکل ۱۷ موارد کاربری فهرست افزونه‌ها

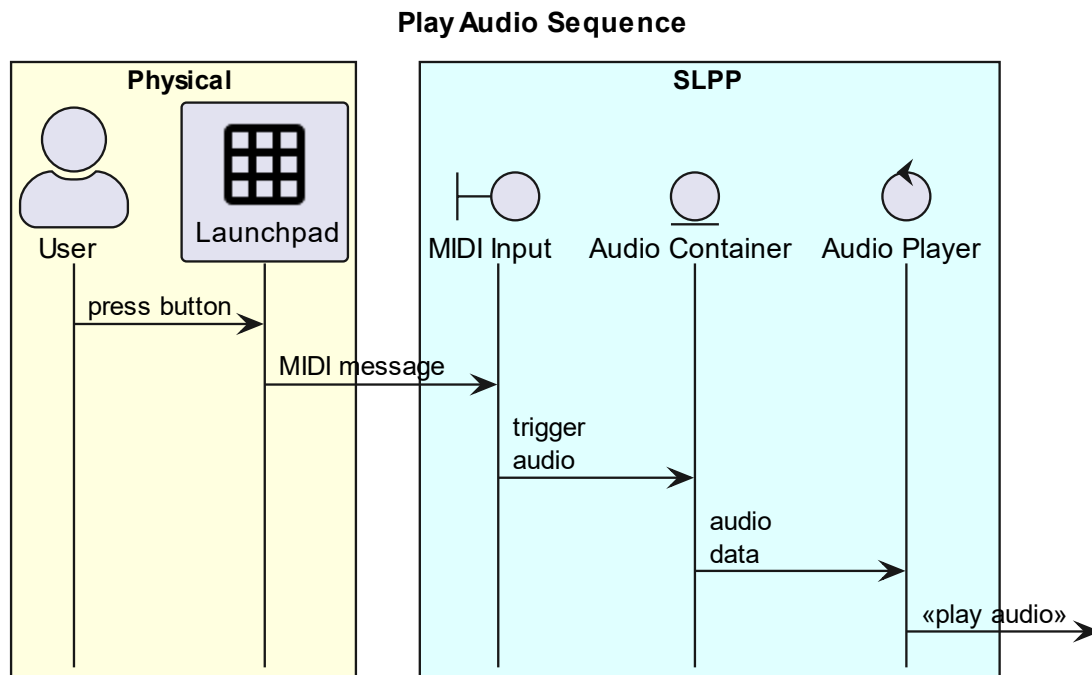
پنجره انیمیشن لیست افزونه‌ها را از این فهرست دریافت و به کاربر نمایش می‌دهد. با انتخاب کاربر، افزونه انتخاب شده فعال می‌شود.

۴.۲ - توالی عملکرد مبدل میدی

همانطور که در بخش‌های قبلی تاکید شد، مبدل میدی مهم‌ترین بخش این پروژه است، از طرفی دستیابی به برخی از نیازمندی‌های کیفی به نحوه پیاده‌سازی این مبدل بستگی دارد. به همین دلیل نیاز است نحوه عملکرد این بخش از پروژه با جزئیات بیشتری بررسی شود. به این منظور، نمودارهای توالی^۱ این مبدل را طراحی و بررسی می‌کنیم. در هنگام پیاده‌سازی، توجه به این توالی‌ها برای بهینه‌سازی کد حیاتی است.

^۱ Sequence diagram

۱.۴.۲ - توالی پخش صدا



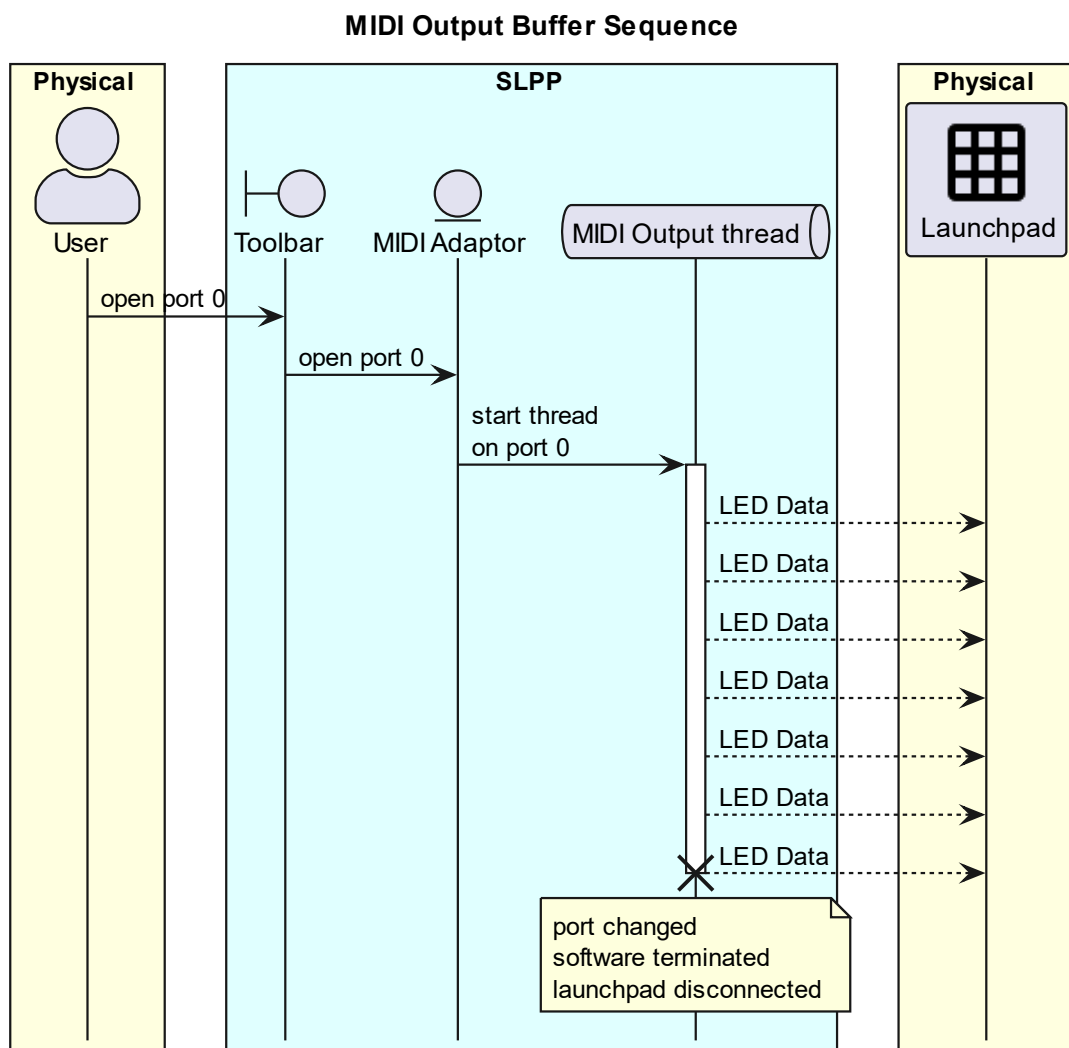
شکل ۱۸ نمودار توالی پخش صدا

این توالی با فشردن یک دکمه اصلی لانچپد توسط کاربر شروع می‌شود. پس از این واقعه، یک پیام میدی از طرف لانچپد به ورودی میدی ارسال می‌شود. این ورودی پیام را تفسیر کرده و بر اساس مختصات دکمه فشرده شده، صدای مورد نظر را در مخزن صدا فعال می‌کند. مخزن اطلاعات صدا را به پخش کننده ارسال و در نهایت صدا از سیستم صوتی پخش می‌شود.

۲.۴.۲ - توالی‌های پخش انیمیشن

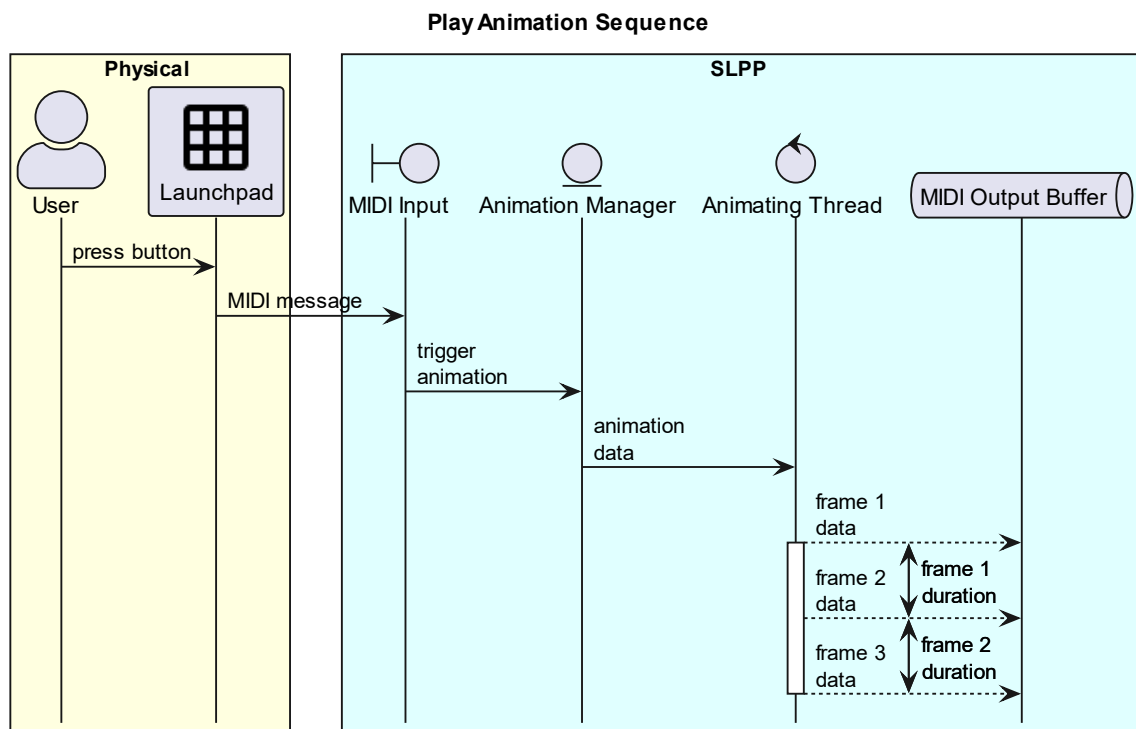
پخش انیمیشن به مراتب پیچیده‌تر از پخش صدا و به دلیل نیاز به پخش فریم‌های متعدد در فاصله‌های زمانی مشخص، نیاز به استفاده از تکنیک‌های برنامه‌نویسی چندنخی^۱ دارد.

^۱ Multithread



شکل ۱۹ نمودار توالی بافر خروجی میدی

با انتخاب پورت میدی خروجی توسط کاربر، نخ خروجی به طور نامحدود شروع به ارسال اطلاعات LEDها از بافر یک طرفه به لانچپد می کند. این بافر تنها شامل اطلاعات نور ۶۴ دکمه اصلی است که در یک توالی دیگر به روز می شوند.



شکل ۲۰ نمودار توالی پخش انیمیشن

این توالی با فشردن شدن یک دکمه اصلی لانچپد توسط کاربر شروع می شود. پس از این واقعه، یک پیام میدی از طرف لانچپد به ورودی میدی ارسال می شود. این ورودی پیام را تفسیر کرده و بر اساس مختصات دکمه فشرده شده، انیمیشن مورد نظر را پخش می کند. برای پخش یک انیمیشن، یک نخ جدید به وجود می آید که بافر خروجی را با اطلاعات هر فریم به روز می کند. این نخ بین هر به روزرسانی به اندازه طول زمانی فریم متوقف و تا فریم آخر این روند تکرار می شود.

با ترکیب این دو توالی چند نخ، اطلاعات انیمیشن ها با حداقل تاخیر و دقت بسیار بالا به لانچپد ارسال می شود.

فصل سوم

پیاده‌سازی نرم افزار

در این فصل به نحوه پیاده‌سازی نرم افزار و کلاس‌های طراحی شده با استفاده از نمودارهای کلاس^۱، آزمون نرم افزار^۲ و طراحی رابط گرافیکی می‌پردازیم.

۱.۳ - کلاس‌ها

در این بخش، به بررسی کلاس‌های هر فضای نام^۳ و ارتباط آن‌ها با یکدیگر می‌پردازیم. قبل از شروع بخش بعدی، لازم به ذکر است که تمامی کلاس‌های یگانه^۴ از کلاس noncopyable کتابخانه Boost ارث‌بری می‌کنند.

GUI - ۱.۱.۳

پیش از شروع این بخش، لازم است نحوه عملکرد ویجت‌ها^۵ در کتابخانه tgui را بررسی کنیم. در این کتابخانه تعدادی ویجت تعریف شده وجود دارد که هنگام توسعه می‌توان از آن‌ها استفاده کرد. همچنین برای گسترش قابلیت‌های یک ویجت، می‌توان با ارث‌بری از آن، علاوه بر قابلیت‌های اصلی آن ویجت، ویژگی‌های مورد نیاز را نیز اضافه کرد. برای اضافه کردن یک ویجت، پس از ارث‌بری نیاز به انجام چند مرحله است:

۱. تعریف توابع clone, copy, create: این توابع در کتابخانه tgui به عنوان توابع کارخانه^۶ طراحی شده و

تمامی ویجت‌ها باید این توابع را ارائه دهند.

۲. سازنده^۷ این ویجت‌ها باید حتما دو پارامتر مورد نیاز کلاس Widget را محیا کنند.

^۱ Class diagram

^۲ Software Testing

^۳ Namespace

^۴ Singleton

^۵ Widget

^۶ Factory

^۷ Constructor

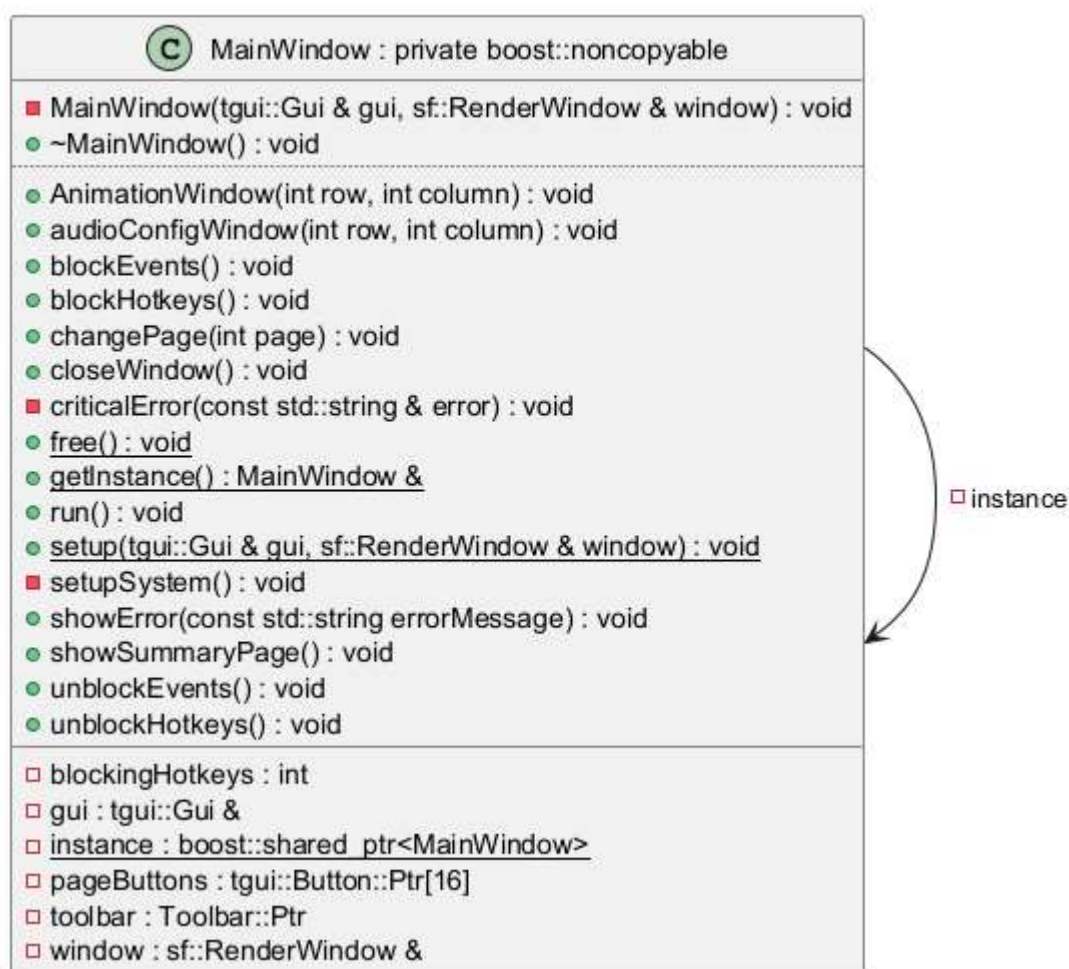
۳. تعریف اشاره گر^۱: همه ویجت‌ها باید نوعی جدید به نام Ptr از نوع `std::shared_ptr` تعریف کنند. به طور مثال:

```
typedef std::shared_ptr<MainButton> Ptr;
```

تصاویری از رابط کاربری گرافیکی در بخش پیوست قرار دارد.

اکنون به بررسی کلاس‌های مربوط به پنجره اصلی می‌پردازیم.

GUI::MainWindow class



شکل ۲۱ نمودار کلاس MainWindow

^۱ Pointer

GUI::ButtonPanel class

C ButtonPanel : public tgui::Panel
<ul style="list-style-type: none"> ● ButtonPanel(const char * typeName = "Panel", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(ButtonPanel::ConstPtr widget) : ButtonPanel::Ptr</u> ● <u>create() : ButtonPanel::Ptr</u>

شکل ۲۲ نمودار کلاس ButtonPanel

GUI::ButtonVerticalLayout class

C ButtonVerticalLayout : public tgui::VerticalLayout
<ul style="list-style-type: none"> ● ButtonVerticalLayout(const char * typeName = "ButtonVerticalLayout", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(ButtonVerticalLayout::ConstPtr widget) : ButtonVerticalLayout::Ptr</u> ● <u>create() : ButtonVerticalLayout::Ptr</u>

شکل ۲۳ نمودار کلاس ButtonVerticalLayout

GUI::ButtonHorizontalLayout class

C ButtonHorizontalLayout : public tgui::HorizontalLayout
<ul style="list-style-type: none"> ● ButtonHorizontalLayout(int rowNumber, const char * typeName = "ButtonHorizontalLayout", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(ButtonHorizontalLayout::ConstPtr widget) : ButtonHorizontalLayout::Ptr</u> ● <u>create(int rowNumber = 0) : ButtonHorizontalLayout::Ptr</u>
□ rowNumber : int

شکل ۲۴ نمودار کلاس ButtonHorizontalLayout

GUI::MainButton class

C MainButton : public tgui::Button
<ul style="list-style-type: none"> ● MainButton(int row, int column, const char * typeName = "MainButton", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(MainButton::ConstPtr widget) : MainButton::Ptr</u> ● <u>create(int row, int column) : MainButton::Ptr</u>
◆ column : int
◆ row : int

شکل ۲۵ نمودار کلاس MainButton

GUI::PageButtonHorizontalLayout class

C PageButtonHorizontalLayout : public tgui::HorizontalLayout
• PageButtonHorizontalLayout(const char * typeName = "PageButtonHorizontalLayout", bool initRenderer = true) : void
◆ clone() const : Widget::Ptr
• <u>copy(PageButtonHorizontalLayout::ConstPtr widget) : PageButtonHorizontalLayout::Ptr</u>
• <u>create() : PageButtonHorizontalLayout::Ptr</u>

شکل ۲۶ نمودار کلاس PageButtonHorizontalLayout

GUI::PageButton class

C PageButton : public tgui::Button
• PageButton(int pageNumber, const char * typeName = "PageButton", bool initRenderer = true) : void
◆ clone() const : Widget::Ptr
• <u>copy(PageButton::ConstPtr widget) : PageButton::Ptr</u>
• <u>create(int pageNumber) : PageButton::Ptr</u>
◆ pageNumber : int

شکل ۲۷ نمودار کلاس PageButton

کلاس MainWindow یک کلاس یگانه است که وظیفه مدیریت صفحه اصلی و وقایع برنامه را دارد. این کلاس در ظاهر یک سطر دکمه تغییر صفحه در بالا، و ۸ سطر اصلی که هر سطر ۸ دکمه اصلی و یک دکمه تغییر صفحه در سمت راست (کاملاً مشابه چینش دکمه‌های لانچپد) دارد. برای رسیدن به این چینش، این کلاس از یک PageButtonHorizontalLayout و یک ButtonPanel استفاده می‌کند که در ادامه به آن‌ها می‌پردازیم. توابع animationWindow و audioConfigWindow توسط دکمه‌های اصلی برای جابه‌جایی کاربر به صفحه انیمیشن و تنظیم صدا استفاده می‌شود. توابع showError و showCritical برای نمایش خطا یا خطای حیاتی (که پس از آن برنامه بسته می‌شود)، توسط هر بخشی که عملکرد آن ممکن است با خطا مواجه شود استفاده می‌شود. توابع مربوط به events و hotkeys برای فعال یا غیرفعال کردن پنجره‌های زیرین برنامه، هنگام باز کردن یک صفحه جدید (مانند صفحه انتخاب درگاه ورودی میدی) استفاده می‌شود.

کلاس ButtonPanel یک ویجت نگه‌دارنده است که یک ButtonVerticalLayout را در خود نگه می‌دارد. کلاس ButtonVerticalLayout یک ویجت چینش عمودی دارای ۸ ButtonHorizontalLayout است. کلاس ButtonHorizontalLayout یک ویجت چینش افقی است که در سازنده خود شماره سطر خود را از والد دریافت می‌کند. این ویجت شامل ۸ MainButton و یک PageButton است.

کلاس **MainButton** یک ویجت دکمه است. با کلیک راست بر روی این ویجت، کاربر به صفحه انیمیشن و با کلیک چپ، به صفحه تنظیم صدا منتقل می‌شود (با استفاده از توابع **MainWindow**). هر دکمه شماره سطر و ستون خود را از والد دریافت می‌کند. با شناور^۱ شدن اشاره گر موس بر روی این ویجت، یک راهنمای ابزار^۲ به کاربر نمایش داده می‌شود.

کلاس **PageButtonHorizontalLayout** یک ویجت چینش افقی شامل ۸ **PageButton** است.

کلاس **PageButton** یک ویجت دکمه است که در سازنده خود شماره صفحه مربوطه را دریافت می‌کند. با کلیک کاربر بر این دکمه، صفحه فعال برنامه تغییر می‌کند.

اکنون به بررسی کلاس‌های مربوط به نوار ابزار می‌پردازیم.

GUI::ToolBar class

C	ToolBar : public tgui::MenuBar
●	ToolBar(const char * typeName = "MenuBar", bool initRenderer = true) : void
◆	clone() const : Widget::Ptr
●	copy(ToolBar::ConstPtr widget) : ToolBar::Ptr
●	create() : ToolBar::Ptr
■	editRedo() : void
■	editUndo() : void
■	fileExit() : void
■	fileNew() : void
■	fileOpen() : void
■	fileSave() : void
■	fileSaveAs() : void
●	handleEvent(sf::Event e) : void
■	openOpenFileDialog() : void
■	openSaveFileDialog() : void
●	viewMIDIInputPorts() : void
●	viewMIDIOutputPorts() : void
■	viewSummaryPage() : void

شکل ۲۸ نمودار کلاس ToolBar

^۱ Hover


^۲ Tooltip

GUI::MIDIPortsChooseWindow class

	MIDIPortsChooseWindow : public tgui::ChildWindow
•	MIDIPortsChooseWindow(const boost::container::vector<std::string> & list, const char * typeName = "ChildWindow", bool initRenderer = true) : void
•	clone() const : Widget::Ptr
•	copy(MIDIPortsChooseWindow::ConstPtr widget) : MIDIPortsChooseWindow::Ptr
•	create(const boost::container::vector<std::string> & list) : MIDIPortsChooseWindow::Ptr
•	getSelectedItem() : std::string
•	listbox : tgui::ListBox::Ptr
•	selectedItem : std::string

شکل ۲۹ نمودار کلاس MIDIPortsChooseWindow

GUI::ConfirmMessageBox class

	ConfirmMessageBox : public tgui::MessageBox
•	ConfirmMessageBox(const char * typeName = "MessageBox", bool initRenderer = true) : void
•	~ConfirmMessageBox() : void
•	clone() const : Widget::Ptr
•	copy(ConfirmMessageBox::ConstPtr widget) : ConfirmMessageBox::Ptr
•	create() : ConfirmMessageBox::Ptr

شکل ۳۰ نمودار کلاس ConfirmMessageBox

کلاس Toolbar وظیفه نمایش گزینه‌های برنامه به کاربر و انتقال عملیات انتخاب شده به بخش مربوطه را دارد. این کلاس برای هر گزینه یک تابع دارد که در آن تابع عملیات مورد نظر انجام می‌شود.

کلاس MIDIPortsChooseWindow یک ویجت پنجره است که برای انتخاب درگاه ورودی یا خروجی میدی توسط Toolbar استفاده می‌شود. با انتخاب گزینه انتخاب درگاه توسط کاربر، کلاس Toolbar لیست درگاه‌ها را از مبدل مورد نظر دریافت کرده و سپس این پنجره را به کاربر نشان می‌دهد. در داخل این پنجره یک جعبه لیست انتخاب^۱ وجود دارد که با بسته شدن پنجره، گزینه انتخاب شده توسط کاربر به Toolbar و سپس به مبدل مورد نظر منتقل شده و درگاه باز می‌شود.

کلاس ConfirmMessageBox یک ویجت جعبه پیام^۲ است که توسط Toolbar برای دریافت تایید از کاربر برای بستن برنامه، پروژه یا باز کردن یک پروژه جدید که همگی منجر به حذف اطلاعات می‌شوند استفاده می‌گردد. اکنون به بررسی کلاس‌های مربوط به صفحه خلاصه می‌پردازیم.

^۱ ListBox

^۲ MessageBox

GUI::SummaryPage class

C SummaryPage : public tgui::ChildWindow
<ul style="list-style-type: none"> SummaryPage(const char * typeName = "ChildWindow", bool initRenderer = true) : void
<ul style="list-style-type: none"> clone() const : Widget::Ptr copy(SummaryPage::ConstPtr widget) : SummaryPage::Ptr create() : SummaryPage::Ptr getPage() const : int nextPage() : void previousPage() : void setupControlPanel(tgui::Panel::Ptr controlPanel) : void setupMainPanel(tgui::Panel::Ptr mainPanel) : void setupPanels() : void
<ul style="list-style-type: none"> page : int pageLabel : tgui::Label::Ptr

شکل ۳۱ نمودار کلاس SummaryPage

GUI::SummaryButton class

C SummaryButton : public tgui::Button
<ul style="list-style-type: none"> SummaryButton(SummaryPage * parent, int row, int column, const char * typeName = "AnimationButton", bool initRenderer = true) : void
<ul style="list-style-type: none"> clone() const : Widget::Ptr copy(SummaryButton::ConstPtr widget) : SummaryButton::Ptr create(SummaryPage * parent, int row, int column) : SummaryButton::Ptr updateTime(tgui::Duration elapsedTime) : bool
<ul style="list-style-type: none"> column : int parent : SummaryPage * row : int

شکل ۳۲ نمودار کلاس SummaryButton

کلاس SummaryPage یک پنجره است که توسط کاربر به واسطه Toolbar فعال می‌شود. این کلاس شامل دو بخش است که توسط توابع setupPanels, setupMainPanel, setupControlPanel تنظیم می‌شود. در بخش اصلی، ۶۴ SummaryButton با چینش ۸ در ۸ قرار دارد. در بخش کنترلی، دو دکمه برای تغییر صفحه لایچپد قرار دارد.

کلاس SummaryButton یک دکمه دارای شماره سطر و ستون است که به کاربر وضعیت بارگذاری صدا و همچنین تعداد فریم‌های انیمیشن آن دکمه را نمایش می‌دهد. با کلیک کاربر بر روی این دکمه، صدا پخش می‌شود. در این کلاس تابع updateTime والد override شده که در هر تازه سازی^۱ صفحه گرافیکی این تابع اجرا می‌شود (مربوط به کتابخانه‌های SFML و tgui). در این تابع اطلاعات مربوط به دکمه از بخش‌های مربوطه

^۱ refresh

دریافت و ظاهر دکمه به روز می شود. دلیل این پیاده سازی، امکان تغییر اطلاعات به هنگام باز بودن این پنجره است که نیاز به به روزرسانی پویا را ایجاد می کند. با شناور شدن موس بر روی این دکمه، یک راهنمای ابزار به کاربر نمایش داده می شود

اکنون به بررسی کلاس صفحه تنظیم صدا می پردازیم.

GUI::AudioConfig class

C AudioConfig : public tgui::ChildWindow
<ul style="list-style-type: none"> AudioConfig(int row, int column, const char * typeName = "ChildWindow", bool initRenderer = true) : void clone() const : Widget::Ptr copy(AudioConfig::ConstPtr widget) : AudioConfig::Ptr create(int row, int column) : AudioConfig::Ptr loadAudio() : void loadAudioPath(const std::vector<tgui::Filesystem::Path> & paths) : void removeAudio() : void setLooping() : void updateTime(tgui::Duration elapsedTime) : bool
<ul style="list-style-type: none"> column : int row : int setLoopingButton : tgui::Button::Ptr

شکل ۳۳ نمودار کلاس AudioConfig

این پنجره شامل ۳ دکمه برای بارگذاری صدا، تنظیم تکرار صدا و حذف صدا است که در سازنده شماره سطر و ستون مربوطه را دریافت تا هنگام تنظیم اطلاعات از آن ها استفاده کند. دکمه تکرار صدا علاوه بر دریافت ورودی، وضعیت تکرار (فعال یا غیرفعال) را نشان می دهد، به همین دلیل نیاز به تعریف تابع updateTime و به روزرسانی اطلاعات این دکمه به صورت پویا است.


حال کلاس های مربوط به پنجره انیمیشن را بررسی می کنیم.

GUI::AnimationWindow class

C AnimationWindow : public tgui::ChildWindow
<ul style="list-style-type: none"> AnimationWindow(int mainRow, int mainColumn, const char * typeName = "ChildWindow", bool initRenderer = true) : void clone() const : Widget::Ptr copy(AnimationWindow::ConstPtr widget) : AnimationWindow::Ptr create(int mainRow, int mainColumn) : AnimationWindow::Ptr setMainButtonText(AnimationPanel::Ptr panel) : void setPanels() : void
<ul style="list-style-type: none"> mainColumn : int mainRow : int


شکل ۳۴ نمودار کلاس AnimationWindow

GUI::AnimationPluginsPanel class

 AnimationPluginsPanel : public tgui::Panel
<ul style="list-style-type: none"> ● AnimationPluginsPanel(const char * typeName = "Panel", bool initRenderer = true) : void
<ul style="list-style-type: none"> ● activatePlugin() : void ◆ clone() const : Widget::Ptr ● <u>copy(AnimationPluginsPanel::ConstPtr widget) : AnimationPluginsPanel::Ptr</u> ● <u>create() : AnimationPluginsPanel::Ptr</u>
<ul style="list-style-type: none"> ◆ listBox : tgui::ListBox::Ptr


شکل ۳۵ نمودار کلاس AnimationPluginsPanel

GUI::AnimationPanel class

 AnimationPanel : public tgui::Panel
<ul style="list-style-type: none"> ● AnimationPanel(const char * typeName = "Panel", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(AnimationPanel::ConstPtr widget) : AnimationPanel::Ptr</u> ● <u>create() : AnimationPanel::Ptr</u>

شکل ۳۶ نمودار کلاس AnimationPanel

GUI::AnimationControlPanel class

 AnimationControlPanel : public tgui::Panel
<ul style="list-style-type: none"> ● AnimationControlPanel(const char * typeName = "Panel", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(AnimationControlPanel::ConstPtr widget) : AnimationControlPanel::Ptr</u> ● <u>create() : AnimationControlPanel::Ptr</u> ◆ firstFrame() : void ◆ frameChanged() : void ◆ frameDurationChanged() : void ◆ insertFrame() : void ◆ lastFrame() : void ◆ nextFrame() : void ◆ previousFrame() : void ◆ removeFrame() : void ● updateTime(tgui::Duration elapsedTime) : bool
<ul style="list-style-type: none"> ○ durationInput : tgui::EditBox::Ptr ○ frameCounter : tgui::Label::Ptr ○ frameInput : tgui::EditBox::Ptr

شکل ۳۷ نمودار کلاس AnimationControlPanel

GUI::AnimationVerticalLayout class

C AnimationVerticalLayout : public tgui::VerticalLayout
<ul style="list-style-type: none"> ● AnimationVerticalLayout(const char * typeName = "AnimationVerticalLayout", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(AnimationVerticalLayout::ConstPtr widget) : AnimationVerticalLayout::Ptr</u> ● <u>create() : AnimationVerticalLayout::Ptr</u>

شکل ۳۸ نمودار کلاس AnimationVerticalLayout

GUI::AnimationHorizontalLayout class

C AnimationHorizontalLayout : public tgui::HorizontalLayout
<ul style="list-style-type: none"> ● AnimationHorizontalLayout(int rowNumber, const char * typeName = "AnimationHorizontalLayout", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(AnimationHorizontalLayout::ConstPtr widget) : AnimationHorizontalLayout::Ptr</u> ● <u>create(int rowNumber = 0) : AnimationHorizontalLayout::Ptr</u>
<ul style="list-style-type: none"> □ rowNumber : int

شکل ۳۹ نمودار کلاس AnimationHorizontalLayout

GUI::AnimationButton class

C AnimationButton : public tgui::Button
<ul style="list-style-type: none"> ● AnimationButton(int row, int column, const char * typeName = "AnimationButton", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(AnimationButton::ConstPtr widget) : AnimationButton::Ptr</u> ● <u>create(int row, int column) : AnimationButton::Ptr</u> ■ setBackground(Business::Light light) : bool ● setLight(Business::Light light) : bool ● updateTime(tgui::Duration elapsedTime) : bool
<ul style="list-style-type: none"> □ column : int □ currentLight : Business::Light □ row : int

شکل ۴۰ نمودار کلاس AnimationButton

GUI::LightSelectWindow class

C LightSelectWindow : public tgui::ChildWindow
<ul style="list-style-type: none"> ● LightSelectWindow(AnimationButton * parent, const char * typeName = "ChildWindow", bool initRenderer = true) : void
<ul style="list-style-type: none"> ◆ clone() const : Widget::Ptr ● <u>copy(LightSelectWindow::ConstPtr widget) : LightSelectWindow::Ptr</u> ● <u>create(AnimationButton * parent) : LightSelectWindow::Ptr</u>
<ul style="list-style-type: none"> ◆ parent : AnimationButton *

شکل ۴۱ نمودار کلاس LightSelectWindow

کلاس AnimationWindow یک پنجره شامل ۳ بخش انیمیشن، کنترل و افزونه‌ها است. بخش انیمیشن در مرکز صفحه و برای تنظیم نور دکمه‌ها، کنترل در پایین صفحه و برای جابه‌جایی و تنظیم فریم‌های انیمیشن و بخش انتخاب افزونه‌ها در سمت چپ صفحه قرار دارد. این سه بخش توسط کلاس‌های AnimationPanel, AnimationControlPanel, AnimationPluginsPanel نمایش داده می‌شوند.

کلاس AnimationPanel وظیفه نمایش دکمه‌های تنظیم نور انیمیشن شامل ۶۴ دکمه به صورت ۸ در ۸ را دارد. این ویجت شامل یک AnimationVerticalLayout است.

کلاس AnimationVerticalLayout یک ویجت چینش عمودی و دارای ۸ AnimationHorizontalLayout است.

کلاس AnimationHorizontalLayout یک ویجت چینش افقی شامل شماره سطر و ۸ AnimationButton است.

کلاس AnimationButton یک ویجت دکمه است که رنگ دکمه مربوطه (بر اساس شماره سطر و ستون) را در فریم کنونی انیمیشن نشان می‌دهد. این ویجت به صورت پویا رنگ پس‌زمینه^۱ خود را در تابع updateTime تنظیم می‌کند. با کلیک بر روی این ویجت، صفحه LightSelectWindow باز می‌شود.

کلاس LightSelectWindow یک صفحه کوچک شامل ۶ دکمه انتخاب نور (زرد، کهربایی، قرمز، سبز، بدون تغییر و خاموش) است که کاربر با انتخاب، نور دکمه مربوطه در فریم فعال انیمیشن را تنظیم می‌کند.

کلاس AnimationControlPanel شامل چندین ویجت ورودی برای جابه‌جایی و تنظیم فریم‌های انیمیشن است.

- دکمه first frame: جابه‌جایی به اولین فریم انیمیشن
- دکمه previous frame: جابه‌جایی به فریم قبلی
- دکمه next frame: جابه‌جایی به فریم بعدی یا اضافه کردن فریم جدید (به هنگام نمایش آخرین فریم)
- دکمه last frame: جابه‌جایی به آخرین فریم انیمیشن
- دکمه insert frame: اضافه کردن یک فریم در مکان فعال
- دکمه delete frame: حذف فریم فعال

^۱ Background

- ورودی frame: دریافت شماره فریم به صورت مستقیم از کاربر و همچنین نمایش شماره فریم فعال
- برچسب total frames: نمایش تعداد کل فریم‌های انیمیشن
- ورودی duration: نمایش یا دریافت طول زمانی فریم به صورت اعشاری در واحد ثانیه

تمامی این موارد دارای یک راهنمای ابزار هستند. تابع updateTime برای به‌روزرسانی پویای اطلاعات فریم‌ها پیاده‌سازی شده است.

کلاس AnimationPluginsPanel دارای یک جعبه لیست انتخاب و دکمه فعال‌سازی افزونه است. این لیست از کلاس فهرست افزونه‌ها دریافت و به کاربر نمایش داده می‌شود. کاربر می‌تواند یک افزونه را انتخاب و با دکمه فعال‌سازی را انجام دهد.

اکنون کلاس ToolTip را بررسی می‌کنیم.

GUI::ToolTip class

C	ToolTip : public tgui::Label
•	ToolTip(tgui::String tip, const char * typeName = "ToolTip", bool initRenderer = true) : void
♦	clone() const : Widget::Ptr
•	<u>copy(ToolTip::ConstPtr widget) : ToolTip::Ptr</u>
•	<u>create(tgui::String tip) : ToolTip::Ptr</u>

شکل ۴۲ نمودار کلاس ToolTip

این کلاس برای نمایش راهنمای ابزارها در برخی صفحات رابط کاربری استفاده می‌شود.

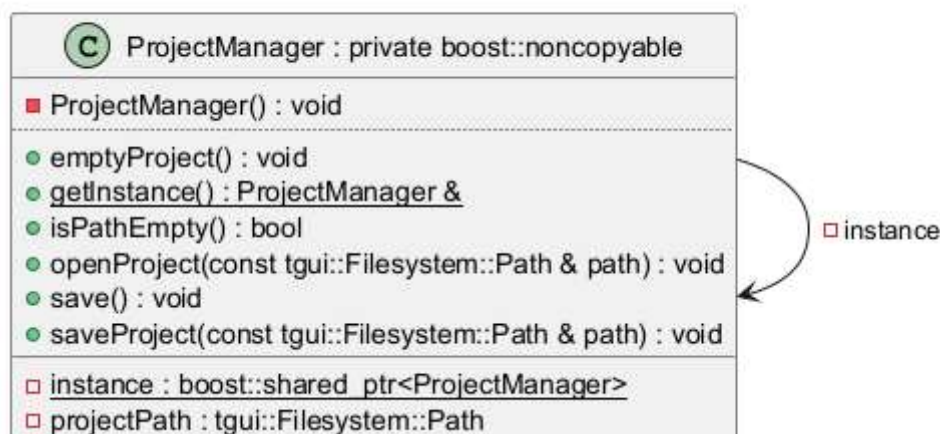
۲.۱.۳ Business

در این فضای نام، کلاس‌های مربوط به منطق برنامه از جمله مدیر پروژه، تاریخچه، کلاس‌های مربوط به انیمیشن، کلاس‌های مربوط به صدا و فهرست افزونه‌ها قرار دارد. در این فضا، کلاس‌های مربوط به انیمیشن‌ها و صداها که نیاز به ذخیره‌سازی و بارگذاری توسط لایه Persistence را دارند، توابع مربوط به سریال‌سازی^۱ را براساس آرشیو باینری^۲ کتابخانه boost پیاده‌سازی می‌کنند.

^۱ serialization

^۲ binary_archive

Business::ProjectManager class

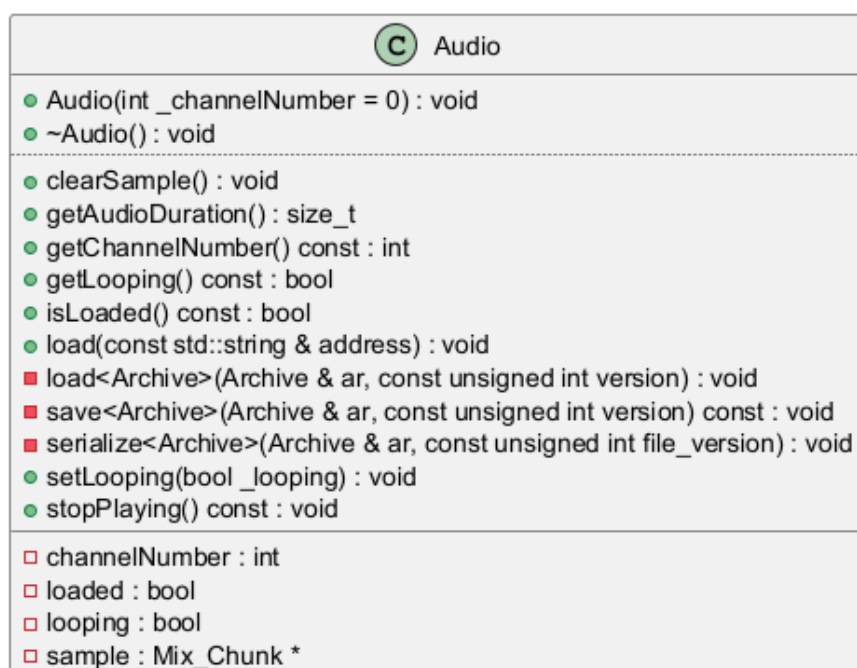


شکل ۴۳ نمودار کلاس ProjectManager

مدیر پروژه یک کلاس یگانه است که وظیفه نگهداری آدرس کنونی پروژه، ذخیره و بارگذاری پروژه‌ها و ساخت پروژه جدید که شامل پاک کردن تمامی صداها و انیمیشن‌ها است را دارد. این کلاس توسط نوار ابزار استفاده می‌شود و خود با فضای نام Persistence برای ذخیره و بارگذاری و با کلاس‌های مدیر انیمیشن و مخزن صداها در ارتباط است.

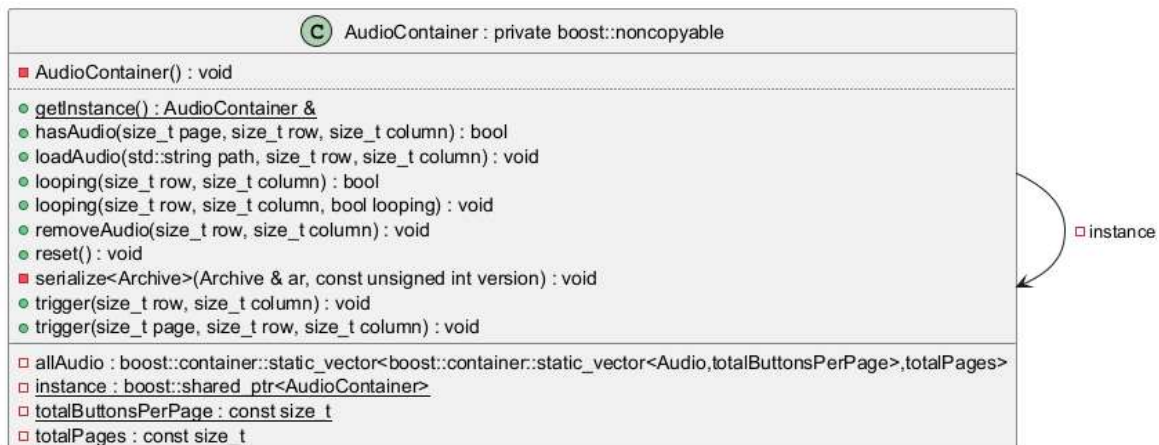
اکنون به بررسی کلاس‌های مربوط به ذخیره و پردازش صدا می‌پردازیم.

Business::Audio class



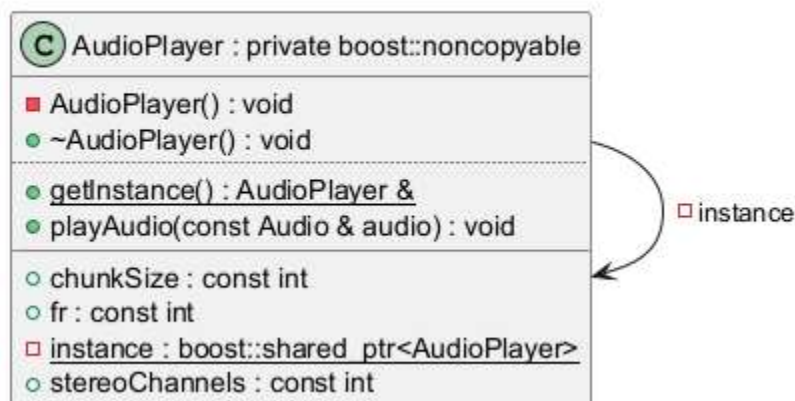
شکل ۴۴ نمودار کلاس Audio

Business::AudioContainer class



شکل ۴۵ نمودار کلاس AudioContainer

Business::AudioPlayer class



شکل ۴۶ نمودار کلاس AudioPlayer

در کتابخانه SDL2_mixer برای پخش صدا نیاز به برقرار ارتباط با یک دستگاه صوتی است. همچنین برای هر صدا نیاز به باز کردن یک کانال است. اطلاعات مربوط به صدا در این کتابخانه در یک ساختار^۱ به نام Mix_Chunk ذخیره می‌شود.

کلاس AudioPlayer وظیفه راه‌اندازی کتابخانه SDL2_mixer و پخش صداها را دارد. این کلاس در سازنده خود توابع مربوط به SDL و SDL2_mixer را فراخوانی، سیستم صوتی راه‌اندازی و به اندازه تمامی صداها مورد نیاز (۱۰۲۴) کانال صوتی باز و در مخرب^۲ خود این سیستم صوتی را غیرفعال و منابع را آزاد می‌کند.

^۱ Structure

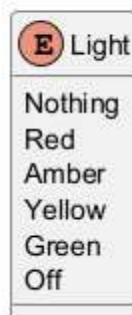
^۲ Destructor

کلاس Audio وظیفه ذخیره اطلاعات مربوط به صدای هر دکمه را دارد. این کلاس وضعیت بارگذاری صدا برای دکمه مورد نظر، وضعیت تکرار و همچنین اطلاعات صدا را با استفاده از ساختار Mix_Chunk ذخیره و مدیریت می‌کند. هر صدا در این سیستم مربوط به یک کانال صوتی است که در این کلاس این شماره نیز نگهداری می‌شود. همچنین برای ذخیره‌سازی و بارگذاری، توابع مربوط به سریال‌سازی، پیاده‌سازی شده‌اند.

AudioContainer یک کلاس یگانه و مخزن اصلی صدا در این نرم افزار است. این کلاس از static_vector کتابخانه Boost برای نگهداری صداها در ۱۶ صفحه مختلف و در هر صفحه ۶۴ صدا استفاده می‌کند. دلیل استفاده از این کتابخانه، ترکیب مزایای دو کلاس array و vector که منجر به سرعت بالا و ذخیره‌سازی بر روی heap می‌شود است. این کلاس عملیات‌های مربوط به صداها را از بخش‌های مختلف نرم افزار دریافت و به صدای مربوطه منتقل می‌کند. در عملیات پخش صدا، این مخزن شماره سطر و ستون را دریافت و اطلاعات صدای مورد نظر در صفحه فعال را به AudioPlayer منتقل می‌کند. تابع reset توسط مدیر پروژه برای بازنشانی تمامی صداها هنگام اجرای عملیات پروژه جدید استفاده می‌شود. همچنین برای ذخیره‌سازی و بارگذاری، توابع مربوط به سریال‌سازی، پیاده‌سازی شده‌اند.

حال کلاس‌های مربوط به ذخیره و پردازش انیمیشن‌ها را بررسی می‌کنیم.

Business::Light enum



شکل ۴۷ نمودار کلاس Light

یک شمارنده^۱ شامل ۶ نور مختلف که توسط لانچپد پشتیبانی می‌شود. عدد متناظر با هر کدام از این مقادیر بر اساس پروتکل لانچپد مقداردهی شده است. این اعداد برابراند با:

Nothing (no change) = 0

Red = 15

^۱ Enumeration (enum)

Amber = 63

Yellow = 62

Green = 60

Off = 12

مقدار No Change به لانچپد ارسال نمی شود، بلکه هنگام به روزرسانی بافر خروجی میدی، نادیده گرفته شده و مقدار قبلی برای LED مورد نظر باقی می ماند.

برای این شمارنده توابع کمکی از جمله تبدیل نور به رشته، تبدیل رشته به نور و تبدیل نور به نام ویجت (استفاده شده در پنجره انتخاب نور در صفحه انیمیشن) تعریف شده است.

Business::Frame class

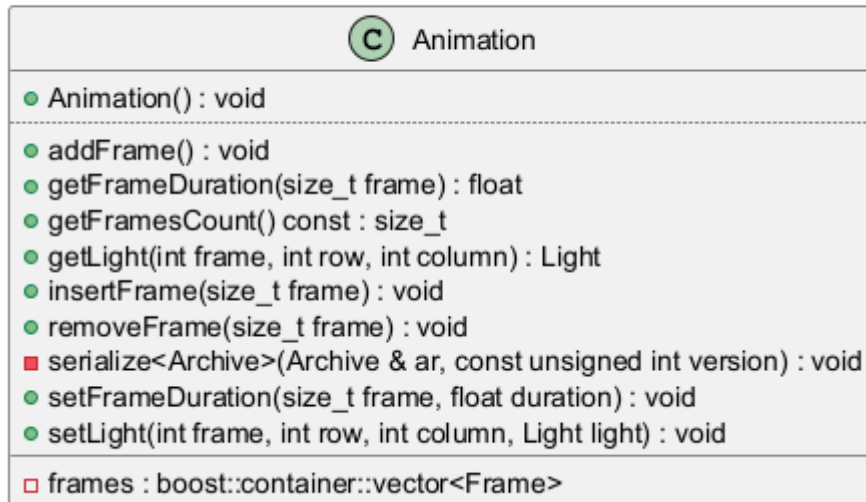
C Frame
● Frame(float duration = 0.f) : void
● getDuration() const : float
● getLight(int row, int column) : Light
■ serialize<Archive>(Archive & ar, const unsigned int version) : void
● setDuration(float duration) : void
● setLight(int row, int column, Light light) : void
□ duration : float
□ lights : boost::array<Light, totalButtonsPerPage>

شکل ۴۸ نمودار کلاس Frame

این کلاس وضعیت LEDهای لانچپد در هر لحظه که شامل ۶۴ مقدار نور برای دکمه های اصلی لانچپد است را نگهداری می کند. از کلاس array کتابخانه boost برای نگهداری این مقادیر استفاده شده که دلیل آن سرعت بالا در تغییر و دسترسی به المان ها است. هر فریم یک مقدار مدت زمان^۱ نیز دارد که نشان دهنده طول فعال بودن این فریم است. همچنین برای ذخیره سازی و بارگذاری، توابع مربوط به سریال سازی، پیاده سازی شده اند.

^۱ duration

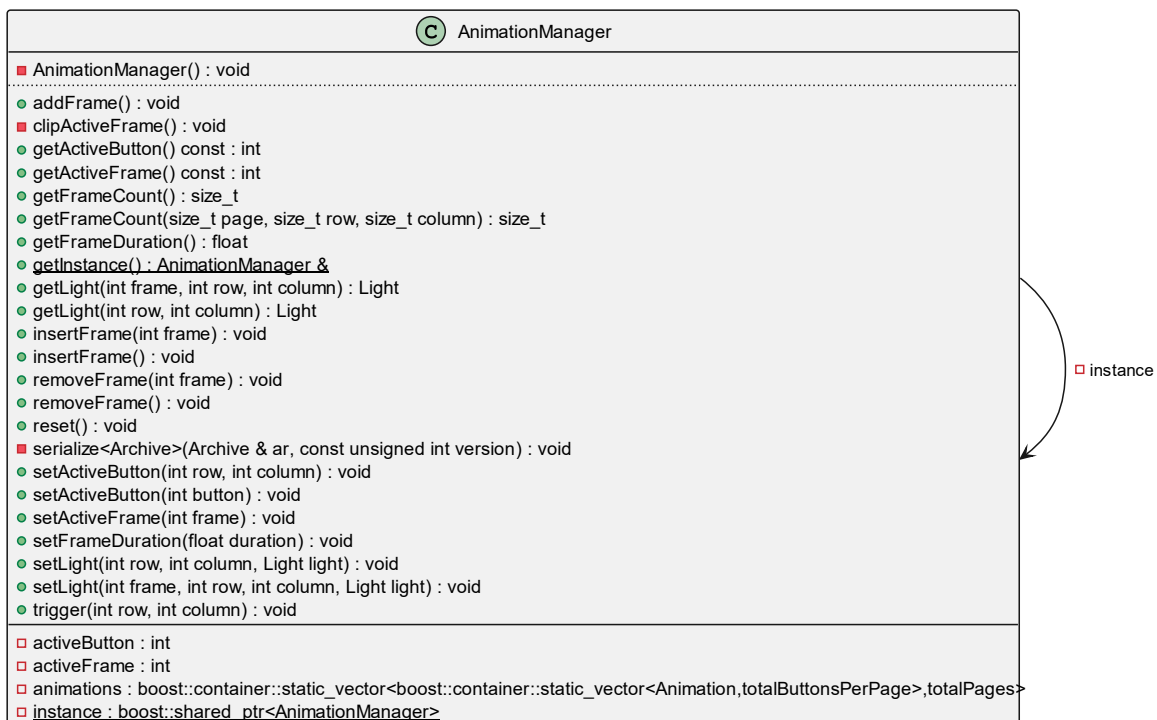
Business::Animation class



شکل ۴۹ نمودار کلاس Animation

این کلاس نشان‌دهنده یک انیمیشن در نرم افزار است. هر انیمیشن شامل تعداد متغیری فریم است که توسط کاربر کنترل می‌شود. این کلاس تمامی توابع مورد نیاز برای تغییر یا دریافت اطلاعات مربوط به فریم‌ها را فراهم می‌کند. همچنین برای ذخیره‌سازی و بارگذاری، توابع مربوط به سریال‌سازی، پیاده‌سازی شده‌اند.

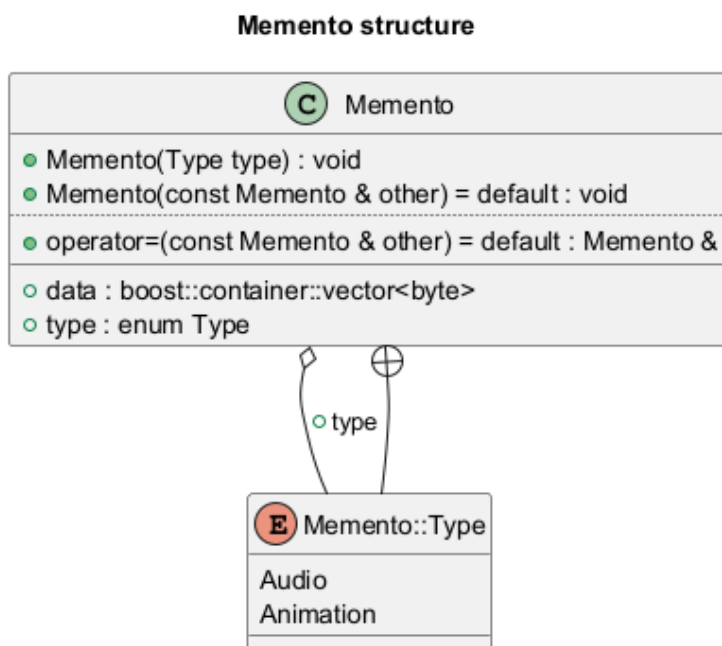
Business::AnimationManager class



شکل ۵۰ نمودار کلاس AnimationManager

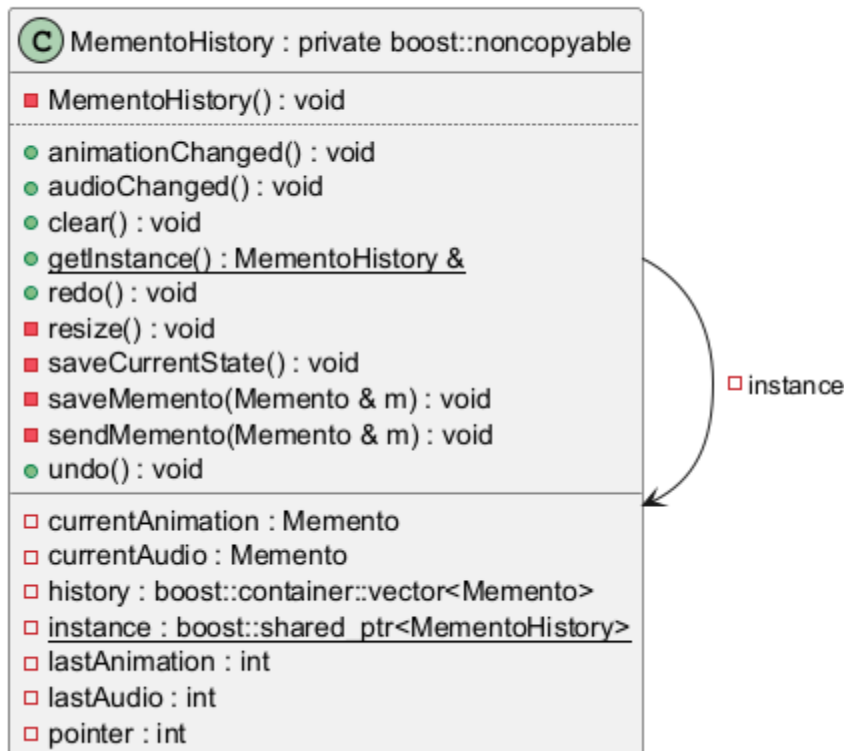
AnimationManager یک کلاس یگانه است که وظیفه مدیریت تمامی انیمیشن‌های سیستم را بر عهده دارد. این کلاس از static_vector کتابخانه Boost برای نگهداری انیمیشن‌ها در ۱۶ صفحه مختلف و در هر صفحه ۶۴ انیمیشن (به ازای هر دکمه اصلی) استفاده می‌کند. توابعی نیز برای دسترسی و تغییر اطلاعات انیمیشن‌ها، فریم‌ها و نورها پیاده‌سازی شده‌اند. علاوه بر این موارد، این کلاس وظیفه پخش یک انیمیشن در هنگام دریافت پیام توسط مبدل میدی را نیز دارد، تابع trigger شماره سطر و ستون دکمه فشرده شده را دریافت و در صفحه فعال سیستم، انیمیشن مورد نظر را به خروجی میدی ارسال می‌کند. تابع reset توسط مدیر پروژه برای بازنشانی تمامی صداها هنگام اجرای عملیات پروژه جدید استفاده می‌شود. همچنین برای ذخیره‌سازی و بارگذاری، توابع مربوط به سریال‌سازی، پیاده‌سازی شده‌اند.

اکنون کلاس‌های تاریخچه یادمان‌ها را بررسی می‌کنیم.



شکل ۵۱ نمودار ساختار Memento

Business::MementoHistory class

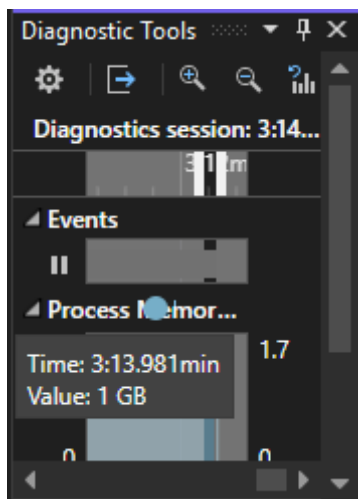


شکل ۵۲ نمودار کلاس MementoHistory

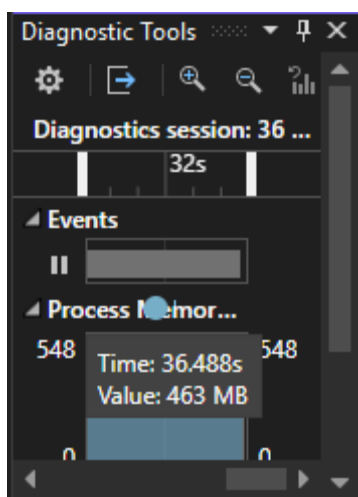
کلاس Memento یک ساختار است که وظیفه آن نگهداری اطلاعات آرشیو باینری مربوط به یک تصویر^۱ از سیستم را دارد. نوع اطلاعات ذخیره شده بر اساس Type با دو مقدار انیمیشن و صدا مشخص می‌شود.

کلاس یگانه MementoHistory وظیفه ذخیره تاریخچه تغییرات سیستم و حرکت در این تاریخچه را دارد. این تاریخچه یک مجموعه‌ای از کلاس Memento با حداکثر طول ۱۰۰ تغییر است. دلیل انتخاب این عدد، مصرف حافظه بالا در یک پروژه سنگین (۵۱۲ صدا و ۵۱۲ انیمیشن با ۱۰ فریم در هر انیمیشن) است. تصاویر ۵۳ و ۵۴ مصرف حافظه را نشان می‌دهند. این کلاس با استفاده از یک اشاره گر وضعیت کنونی سیستم، یا به عبارتی مکان کنونی سیستم در تاریخچه تغییرات را نگهداری میکند. فراخوانی توابع redo و undo این اشاره گر را در تاریخچه حرکت و وضعیت سیستم را به روز می‌کند. توابع audioChanged و animationChanged یک Memento از نوع متناظر را در تاریخچه ذخیره می‌کند. تابع clear توسط مدیر پروژه برای پاکسازی تاریخچه هنگام باز شدن یک پروژه جدید استفاده می‌شود.

^۱ snapshot



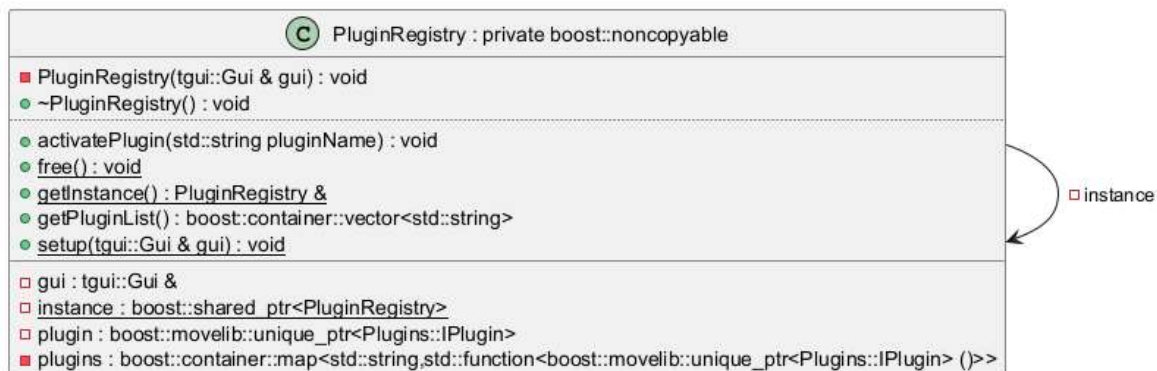
شکل ۵۳ مصرف حافظه ۱ گیگابایتی با طول تاریخچه ۲۰۰ تغییر در یک پروژه با ۵۱۲ صدا و ۵۱۲ انیمیشن ۱۰ فریمی



شکل ۵۴ کاهش چشمگیر مصرف حافظه با طول تاریخچه ۱۰۰ تغییر

اکنون آخرین کلاس این فضا، فهرست افزونه‌ها را بررسی می‌کنیم.

Business::PluginRegistry class



شکل ۵۵ نمودار کلاس PluginRegistry

کلاس یگانه PluginRegistry وظیفه نگه‌داری لیست همه افزونه‌های سیستم و فعال‌سازی افزونه انتخاب شده توسط کاربر در صفحه انیمیشن‌ها را دارد. این لیست در یک نقشه^۱ از کتابخانه boost ذخیره می‌شود. کلید این نقشه نام افزونه و مقدار متناظر، تابع سازنده آن افزونه است. افزونه فعال نیز در یک اشاره‌گر یکتا^۲ قرار دارد. پنجره انیمیشن با استفاده از تابع getPluginList، فهرست تمامی افزونه‌ها را به کاربر نمایش می‌دهد.

۳.۱.۳ Persistence

Persistence::Load class

C Load
• Load(const tgui::Filesystem::Path & path) : void
• load() : void
□ path : boost::filesystem::path

شکل ۵۶ نمودار کلاس Load

Persistence::Save class

C Save
• Save(const tgui::Filesystem::Path & path) : void
• save() : void
□ path : boost::filesystem::path

شکل ۵۷ نمودار کلاس Save

در این فضای نام، دو کلاس برای ذخیره و بارگذاری اطلاعات پروژه وجود دارد. هر دوی این کلاس‌ها از سریال‌سازی و آرشیو باینری کتابخانه boost برای جمع‌آوری اطلاعات سیستم از مخزن صداها و مدیر انیمیشن‌ها و از filesystem کتابخانه boost برای ارتباط با فایل سیستم استفاده می‌کنند.

کلاس Load بارگذاری اطلاعات از فایل سیستم به نرم‌افزار را با استفاده از binary_iarchive کتابخانه boost انجام می‌دهد. در صورت بروز خطا در بارگذاری داده‌ها به کاربر اطلاع‌رسانی می‌شود.

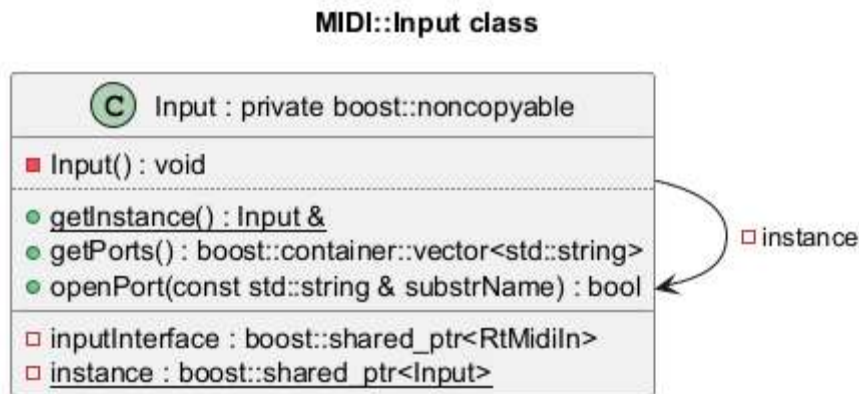
^۱ map

^۲ unique_ptr

کلاس Save ذخیره اطلاعات از نرم افزار به فایل سیستم را با استفاده از binary_oarchive کتابخانه boost انجام می دهد.

MIDI – ۴.۱.۳

در این فضا، ۲ کلاس مهم مبدل میدی، Input و Output قرار دارد که هر دو یگانه هستند.



شکل ۵۸ نمودار کلاس Input

این کلاس با استفاده از کلاس RtMidiIn با رابط کاربری برنامه نویسی^۱ مربوط به میدی سیستم عامل ارتباط برقرار می کند. برای افزایش سرعت پردازش، خارج از این کلاس، یک تابع پاسخ به تماس^۲ به صورت درخت^۳ و ایستا^۴ تعریف شده که در سازنده این کلاس، به شیئ intpuInterface با استفاده از تابع setCallback کلاس RtMidiIn منتسب^۵ می شود که به محض دریافت پیام از طرف لانچپد، این تابع اجرا و پردازش های مربوط به پیام صورت می گیرد. پس از انجام این پردازش ها، بسته به نوع دکمه فشار داده شده (کنترلی یا اصلی) عملیات مربوط در دیگر بخش های نرم افزار (تغییر صفحه فعال سیستم یا پخش انیمیشن و صدا با استفاده از توابع trigger) اجرا می شود.

توابع getPorts و openPort در نوار ابزار برای انتخاب درگاه ورودی میدی استفاده می شوند.

^۱ API

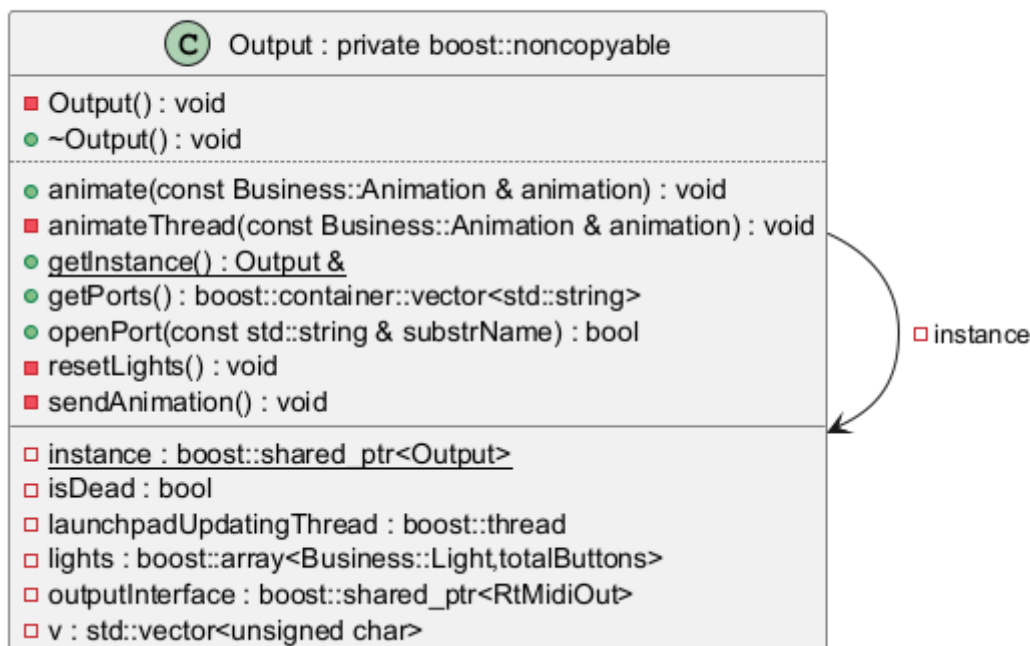
^۲ Callback

^۳ inline

^۴ static

^۵ bind

MIDI::Output class



شکل ۵۹ نمودار کلاس Output

این کلاس با استفاده از کلاس RtMidiOut با رابط کاربری برنامه‌نویسی مربوط به میدی سیستم عامل ارتباط برقرار می‌کند. توابع `getPorts` و `openPort` در نوار ابزار برای انتخاب درگاه خروجی میدی استفاده می‌شوند. در این کلاس وضعیت زنده LEDهای لانیچید در آرایه `lights` ذخیره می‌شوند. در هنگام باز شدن پورت خروجی، یک نخ جدید شروع شده که اطلاعات این آرایه را در یک چرخه بی‌نهایت به لانیچید ارسال می‌کند. این ارسال به روش `Rapid LED update [2]` تا زمان فعال شدن متغیر `isDead` ادامه پیدا می‌کند.

پس از فعال شدن یک انیمیشن برای پخش، اطلاعات این انیمیشن به تابع `animate` ارسال می‌شود. در این تابع یک نخ جدید با تابع `animateThread` باز شده و در لحظه جدا می‌شود.^۱ در تابع `animateThread`، آرایه `lights` با اطلاعات فریم‌های انیمیشن از اولین تا آخرین فریم در یک چرخه موازی [14] به‌روز می‌شود. برای تحقق فواصل زمانی بین فریم‌ها، از کتابخانه `boost chrono` استفاده شده است. برای کاهش تاخیر و افزایش سرعت اجرا، این کلاس به عنوان کلاس دوست^۲ انیمیشن و فریم تعریف شده که در نتیجه آن، بدون نیاز به فراخوانی تابع‌های اضافی به داده داخل این کلاس دسترسی دارد.


^۱ detach

^۲ friend class

۲.۳ - افزونه‌ها

تمامی افزونه‌های سیستم از کلاس انتزاعی ^۱ IPlugin ارث‌بری می‌کنند. همانطور که در فصل قبلی اشاره شد، برای کاهش نیازمندی افزونه‌ها به دیگر بخش‌های سیستم، دسترسی مستقیم به کتابخانه tgui به افزونه‌ها داده شده تا در صورت نیاز ورودی‌های خود را از کاربر دریافت کنند.

Plugins::IPlugin class

 IPlugin
<ul style="list-style-type: none"> ● IPlugin(tgui::Gui & gui, Business::Animation & animation, Business::Frame & frame, Business::Audio & audio) : void ● ~IPlugin() constexpr = default : void
<ul style="list-style-type: none"> ● activate() = 0 : void
<ul style="list-style-type: none"> ◇ activeButton : int ◇ animation : Business::Animation & ◇ audio : Business::Audio & ◇ col : int ◇ frame : Business::Frame & ◇ gui : tgui::Gui & ◇ row : int

این کلاس در سازنده خود شیئی اصلی tgui، انیمیشن فعال، فریم فعال و صدای دکمه مورد نظر را دریافت می‌کند. همچنین سطر و ستون انیمیشن فعال را مستقیماً از مدیر انیمیشن دریافت می‌کند.

پس از پیاده‌سازی، یک افزونه باید به لیست فهرست افزونه‌ها اضافه شود، اینکار در سازنده فهرست با استفاده از ماکرو ^۲ REGISTER_PLUGIN انجام می‌شود.

حال به بررسی افزونه‌های نوشته شده و عملکرد آن‌ها می‌پردازیم.

• CopyAnimation

اطلاعات انیمیشن را کپی می‌کند. این اطلاعات با استفاده از binary_oarchive کتابخانه boost به صورت یک فایل ذخیره می‌شود.

• PasteAnimation

اطلاعات انیمیشن را می‌چسباند ^۳. این اطلاعات با استفاده از binary_iarchive کتابخانه boost از فایل ذخیره شده بارگذاری می‌شود.

^۱ Abstract

^۲ Macro

^۳ paste

- **CopyFrame**
اطلاعات فریم را کپی می کند. این اطلاعات با استفاده از `binary_oarchive` کتابخانه `boost` به صورت یک فایل ذخیره می شود.
- **PasteFrame**
اطلاعات فریم را می چسباند. این اطلاعات با استفاده از `binary_iarchive` کتابخانه `boost` از فایل ذخیره شده بارگذاری می شود.
- **SaveAnimation**
اطلاعات انیمیشن را ذخیره می کند. این اطلاعات با استفاده از `binary_iarchive` کتابخانه `boost` در یک فایل با آدرس انتخابی کاربر ذخیره می شود.
- **LoadAnimation**
اطلاعات انیمیشن را بارگذاری می کند. این اطلاعات با استفاده از `binary_oarchive` کتابخانه `boost` از یک فایل با آدرس انتخابی کاربر بارگذاری می شود.
- **MoveAnimationUp**
تمامی فریم های انیمیشن را یک سطر به بالا حرکت می دهد. خانه های سطر پایینی برابر با `Nothing` قرار می گیرند.
- **MoveAnimationUpWrapping**
تمامی فریم های انیمیشن را یک سطر به بالا حرکت می دهد. خانه های سطر پایینی برابر با سطر بالایی قرار می گیرند.
- **MoveAnimationDown**
تمامی فریم های انیمیشن را یک سطر به پایین حرکت می دهد. خانه های سطر بالایی برابر با `Nothing` قرار می گیرند.
- **MoveAnimationDownWrapping**
تمامی فریم های انیمیشن را یک سطر به پایین حرکت می دهد. خانه های سطر بالایی برابر با سطر پایینی قرار می گیرند.
- **MoveAnimationRight**
تمامی فریم های انیمیشن را یک ستون به راست حرکت می دهد. خانه های ستون چپ برابر با `Nothing` قرار می گیرند.

- **MoveAnimationRightWrapping**
تمامی فریم‌های انیمیشن را یک ستون به راست حرکت می‌دهد. خانه‌های ستون چپ برابر با ستون راست قرار می‌گیرند.
- **MoveAnimationLeft**
تمامی فریم‌های انیمیشن را یک ستون به چپ حرکت می‌دهد. خانه‌های ستون راست برابر با Nothing قرار می‌گیرند.
- **MoveAnimationLeftWrapping**
تمامی فریم‌های انیمیشن را یک ستون به چپ حرکت می‌دهد. خانه‌های ستون راست برابر با ستون چپ قرار می‌گیرند.
- **ReplaceLightFrame**
دو نور را از کاربر ورودی گرفته و در فریم فعال آن‌ها را جایگزین می‌کند.
- **ReplaceLightAnimation**
دو نور را از کاربر ورودی گرفته و در تمامی انیمیشن آن‌ها را جایگزین می‌کند.
- **FillAnimationToAudio**
فاصله زمانی بین فریم‌ها را برابر با نسبت طول صدا به تعداد فریم‌ها قرار می‌دهد. در صورتی که فریم آخر خالی از نور باشد، طول زمانی آن تغییری نمی‌کند. این افزونه برای هماهنگ کردن انیمیشن با صدا استفاده می‌شود.
- **RotateAnimationLeft**
تمامی فریم‌های انیمیشن را ۹۰ درجه در جهت خلاف عقربه‌های ساعت می‌چرخاند.
- **RotateFrameLeft**
فریم فعال را ۹۰ درجه در جهت خلاف عقربه‌های ساعت می‌چرخاند.
- **RotateAnimationRight**
تمامی فریم‌های انیمیشن را ۹۰ درجه در جهت عقربه‌های ساعت می‌چرخاند.
- **RotateFrameRight**
فریم فعال را ۹۰ درجه در جهت عقربه‌های ساعت می‌چرخاند.
- **FlipAnimationVertically**
تمامی فریم‌های انیمیشن را به صورت عمودی معکوس می‌کند.

- FlipFrameVertically
فریم فعال را به صورت عمودی معکوس می کند.
- FlipAnimationHorizontally
تمامی فریم های انیمیشن را به صورت افقی معکوس می کند.
- FlipFrameHorizontally
فریم فعال را به صورت افقی معکوس می کند.
- CircleAnimatoinGenerator
شعاع یک دایره، نور، نوع انیمیشن (out, in, out in) و نوع شکل (دایره توخالی یا دایره توپر) را از کاربر دریافت کرده و یک دایره به مرکز دکمه فعال با استفاده از الگوریتم Midpoint circle [15] رسم می کند.
- VerticalLineAnimationGenerator
طول یک خطو نور و نوع انیمیشن (out, in, out in) را از کاربر دریافت کرده و یک انیمیشن خط عمودی که از دکمه انتخاب شده شروع و به بالا و پایین حرکت می کند را رسم می کند.
- HorizontalLineAnimationGenerator
طول یک خطو نور و نوع انیمیشن (out, in, out in) را از کاربر دریافت کرده و یک انیمیشن خط افقی که از دکمه انتخاب شده شروع و به چپ و راست حرکت می کند را رسم می کند.
- PlusLineAnimationGenerator
ترکیب دو افزونه قبلی است.

۳.۳ - آزمون نرم افزار

برای بررسی صحت عملکرد بخش های منطقی نرم افزار مجموعه ای از آزمون ها^۱ با استفاده از فریمورک^۲ googletest در ۴ فضای مختلف پیاده سازی شده است. قبل از کامپایل و شروع تست ها نیاز است که مقدار GTEST_ در پروژه اصلی به عنوان یک دستورالعمل پیش پردازنده^۳ تعریف^۴ شود. با اضافه کردن این دستور، متغیرهای مشترکی بین دو پروژه اصلی و تست برای اندازه گیری زمان اجرای توالی های حساس تعریف می شوند. در

^۱ tests

^۲ framework

^۳ Preprocessor directive

^۴ define

صورتی که این مقدار اضافه نشود، تست‌های میدی با مشکل پیوند^۱ مواجه می‌شوند. هر فضا از یک یا چندین دنباله^۲ تست تشکیل شده است. همچنین برای تست‌های مربوط به ذخیره یا بارگذاری انواع داده، تعدادی فایل در کنار پروژه تست قرار داده شده است. نتیجه نهایی تست صحت نرم افزار در تصویر ۶۰ قابل مشاهده است.

Test	Duration	Traits	Error Message	Run Debug
SLPP_Test (46)	6.1 min			
BusinessTest (8)	3.7 min			
AnimationUnitTest (4)	3.7 min			
AudioUnitTest (3)	1.2 sec			
ProjectManager (1)	< 1 ms			
IntegrationTests (5)	9.1 sec			
Audio (1)	5.2 sec			
MementoHistory (2)	2.6 sec			
ProjectManager (2)	1.4 sec			
MIDITests (5)	2.2 min			
MIDITests (1)	72 ms			
RepeatDelayTest/VirtualMIDITest (..)	2.2 min			
PluginTests (28)	3.2 sec			
PluginTests (28)	3.2 sec			

Group Summary

SLPP_Test

Tests in group: 46

Total Duration: 6.1 min

Outcomes

46 Passed

شکل ۶۰ نتیجه تست صحت عملکرد نرم افزار

۱.۳.۳ - تست‌های منطق

- تست‌های واحد انیمیشن (AnimationUnitTest)

- تست مدیر انیمیشن (AnimationManagerUnitTest)

- تست انیمیشن (AnimationTest)

- تست فریم (FrameTest)

- تست نور (LightTest)

- تست‌های واحد صدا (AudioUnitTest)

- تست مخزن صدا (AudioContainerTest)

- تست فرمت‌های صدا (AudioFormatsTest)

- تست صدا (AudioTest)

- تست مدیر پروژه (ProjectManager)

- تست واحد (UnitTest)

^۱ linking

^۲ suite

۲.۳.۳ - تست‌های ادغام^۱

- تست‌های صدا (Audio)
 - تست ادغام مخزن و پخش‌کننده (AudioContainer_AudioPlayer_test)
- تست‌های تاریخچه (MementoHistory)
 - تست ادغام با مدیر انیمیشن (MementoHistory_AnimationManager_test)
 - تست ادغام با مخزن صداها (MementoHistory_AudioContainer_test)
- تست‌های مدیر پروژه (ProjectManager)
 - تست ادغام با مخزن صداها و مدیر انیمیشن (ProjectManager_AudioContainer_AnimationManager_test)
 - تست ادغام با لایه ماندگاری (ProjectManager_PersistenceLayer_test)

۳.۳.۳ - تست‌های میدی

برای عملکرد صحیح این تست‌ها نیاز است نرم افزار loopMIDI فعال و ۲ پورت با نام‌های slpp to test و test to slpp باشد. این نرم افزار به عنوان یک مجازی‌ساز میدی عمل کرده و اجازه می‌دهد دو برنامه تست و SLPP به صورت مجازی با هم ارتباط میدی برقرار کنند. برای اطمینان از صحت تنظیمات نرم افزار loopMIDI می‌توانید به تصویر ۷۲ در بخش پیوست‌ها مراجعه کنید.

- تست میدی
 - تست پورت‌های ورودی خروجی (InputOutputPortsTest)
 - تست تاخیر مجازی
 - تست تاخیر انیمیشن و صدا (AnimationAndAudioDelayTest)
- با هر بار اجرا، تست تاخیر ۴ بار تکرار می‌شود. در این تست یک انیمیشن و یک صدا فعال می‌شوند. این انیمیشن دارای ۴ فریم با بازه‌های زمانی به ۰.۰۵ ثانیه، ۰.۱۵ ثانیه، ۰.۳ ثانیه و ۰ ثانیه است. نتیجه تاخیرهای ۴ اجرا به صورت زیر است.

جدول ۳ نتیجه تست تاخیر میدی

Section	Minimum delay	Maximum delay	Average delay
Audio	53.3 μ s	171.9 μ s	99.9 μ s
Animation thread	340.4 μ s	582.8 μ s	438.85 μ s

^۱ integration

Frame 1	340.6 μ s	583.1 μ s	439.02 μ s
Frame 2	774.9 μ s	2861 μ s	1731 μ s
Frame 3	1602 μ s	8178 μ s	3844 μ s
Frame 4	3096 μ s	10305 μ s	5963 μ s

دلیل وجود جهش در تاخیرات انیمیشن و فریم‌ها، استفاده از تکنیک چندنخی و نیاز به شروع یک نخ جدید است.

لاگ تست تاخیر در کد ۱ بخش پیوست‌ها موجود است.

۴.۳.۳ - تست‌های افزونه‌ها

برای تمامی افزونه‌های سیستم یک تست برای سنجش صحت عملکرد پلاگین مورد نظر پیاده‌سازی شده است.

۴.۳ - طراحی رابط کاربری^۱

کتابخانه tgui قابلیت طراحی و سفارش‌سازی تم^۲ برنامه را به سادگی فراهم می‌کند. در این کتابخانه هر ویجت دارای یک رندرکننده^۳ متناظر با نام آن ویجت است که تمامی ویژگی‌های ظاهری آن ویجت را مشخص می‌کند. برای طراحی تم برنامه تنها نیاز است تمامی رندرکننده‌های ویجت‌های استفاده شده تنظیم شوند.

طراحی تم برنامه با هدف رسیدن به حالت تاریک^۴ بر اساس Google Material 2 [16] انجام شده. به دلیل اینکه اجزای لایه‌های توضیح بهتر نور LEDها معمولاً در فضایی با شدت نور پایین انجام می‌شود، تصمیم به طراحی تاریک محیط برنامه گرفته شد.

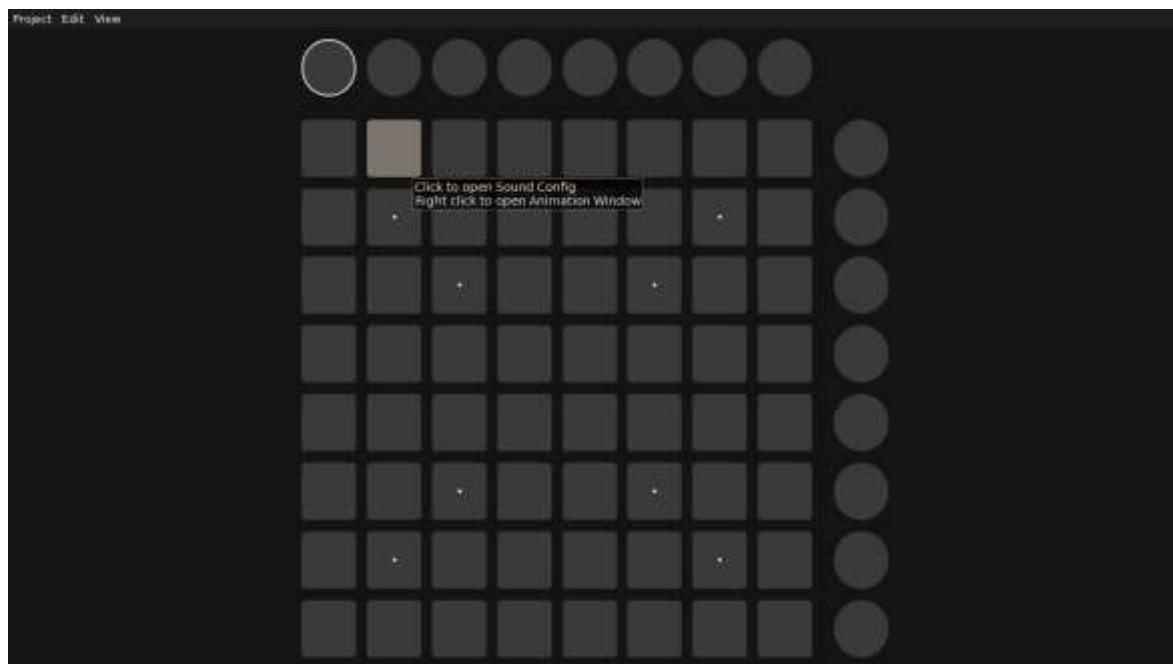
در ادامه، تصاویری از بخش‌های اصلی برنامه برای بررسی تم نهایی قرار داده شده‌اند.

^۱ UI/UX

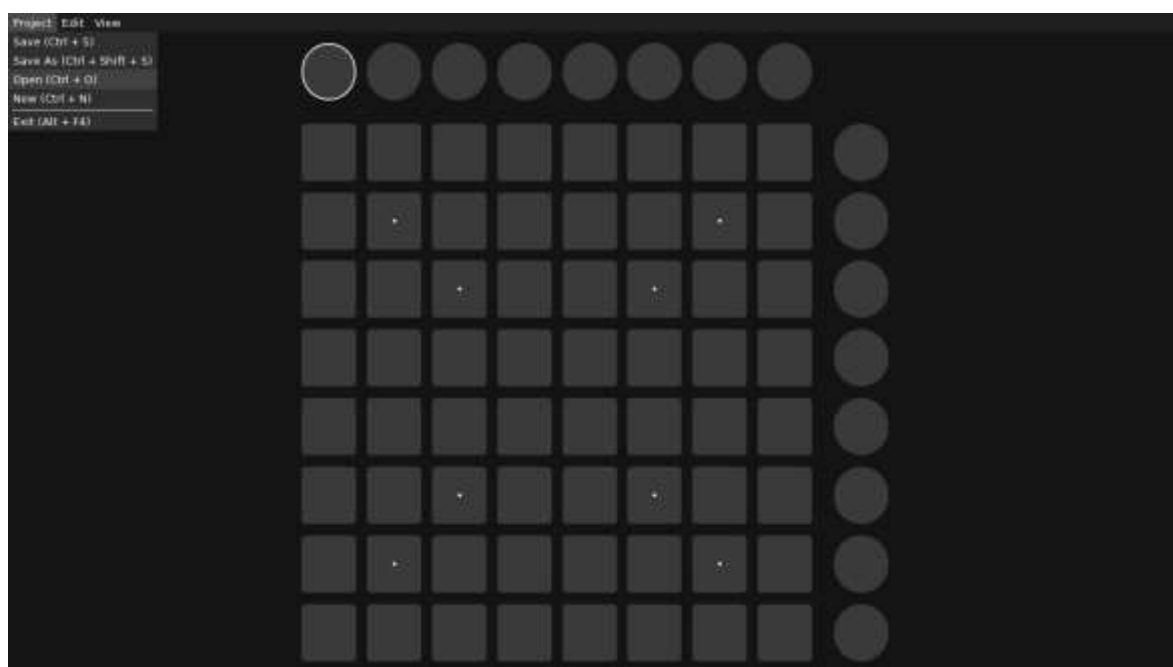
^۲ Theme

^۳ Renderer

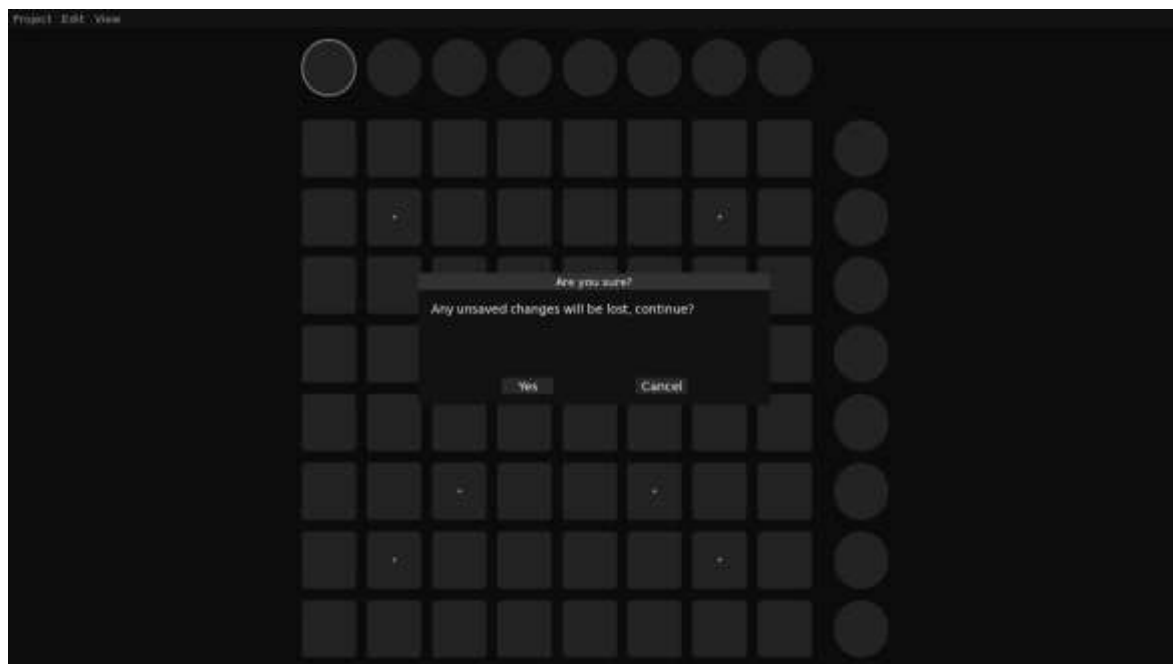
^۴ Dark mode



شکل ۶۱ صفحه اصلی



شکل ۶۲ نوار ابزار



شکل ۶۳ پیام تایید



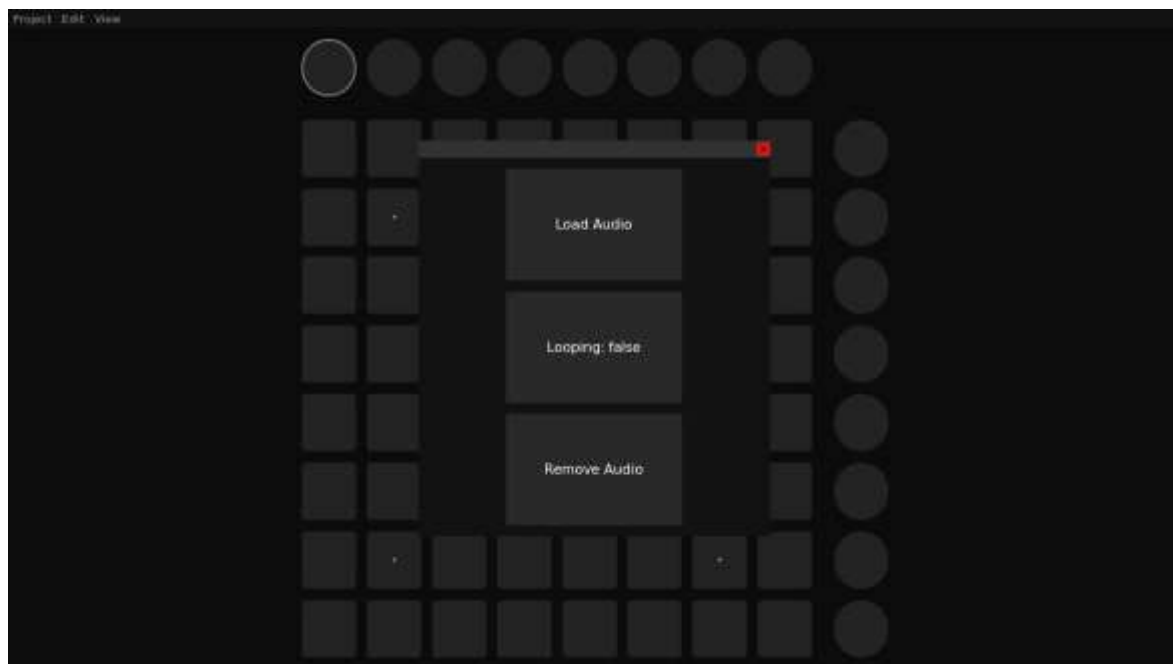
شکل ۶۴ پنجره انتخاب فایل



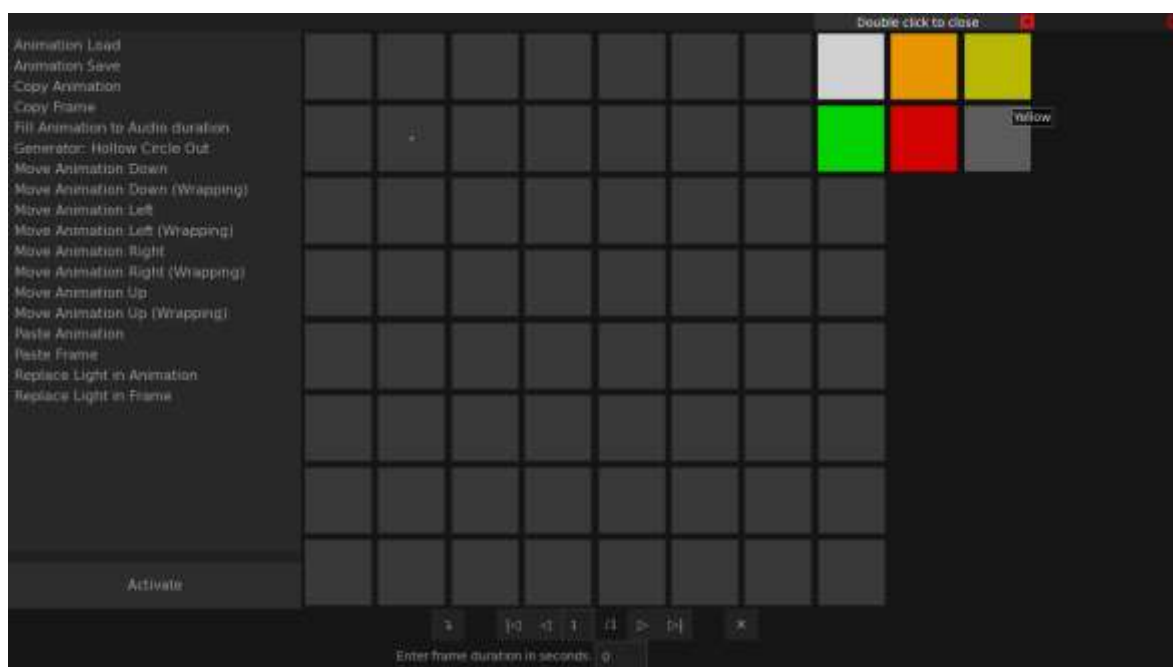
شکل ۶۵ صفحه خلاصه



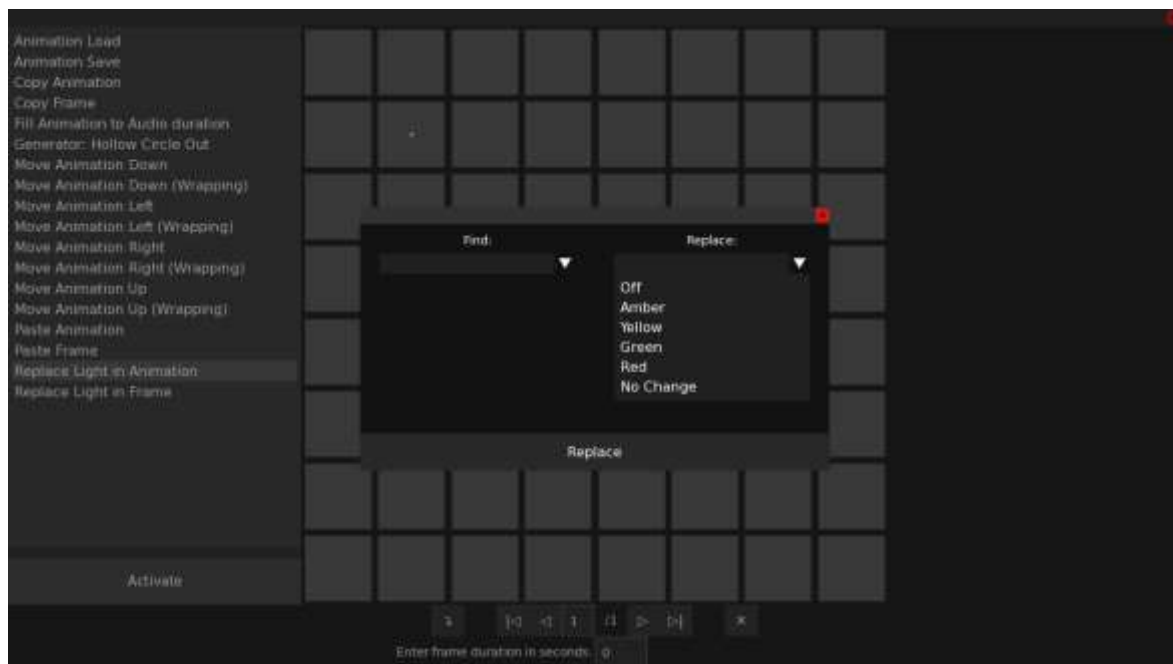
شکل ۶۶ صفحه انتخاب درگاه میدی



شکل ۶۷ صفحه تنظیم صدا



شکل ۶۸ صفحه انیمیشن، انتخاب نور LED



شکل ۶۹ صفحه انیمیشن، ورودی یکی از افزونه‌ها

فصل چهارم

جمع‌بندی و پیشنهادها

۱.۴ - نتیجه

ماحصل این پروژه، یک نرم افزار با طراحی کاربرپسند و کاملاً بهینه برای هدف اجرای زنده موسیقی و نمایش نور با دستگاه لانچپد مدل Mini Mk2 است. در طول انجام این پروژه، بازدهی عملکرد نرم افزار، ارتباط و تجربه کاربر با این برنامه مهم‌ترین اولویت بود که نتیجه آن، محیط کاربری بسیار مناسب خصوصاً نسبت به نرم افزارهای دیگر حوزه موسیقی است.

۲.۴ - پیشنهادها

با بررسی عملکرد نرم افزار در محیط واقعی و همچنین دریافت بازخورد از کاربران، قابلیت‌های بیشتری می‌توان به این نرم افزار اضافه کرد و همچنین مجموعه افزونه‌های برنامه را نیز گسترش داد. در نسخه‌های آینده نرم افزار می‌توان پشتیبانی از سایر مدل‌های دستگاه لانچپد را نیز اضافه کرد، این مدل‌ها علاوه بر طراحی ظاهری متفاوت، قابلیت‌های بیشتری نیز دارند، به طور مثال مدل‌های جدیدتر دارای LEDهایی با طیف نوری بالاتری نسبت به مدل مدنظر این پروژه هستند. در حال حاضر پروژه فقط بر روی سیستم عامل ویندوز تست و استقرار^۱ یافته، برای دسترسی کاربران بیشتر به این نرم افزار نیاز است بر روی سیستم عامل‌های دیگر نیز تست و پخش^۲ شود.

^۱ deployed

^۲ Release

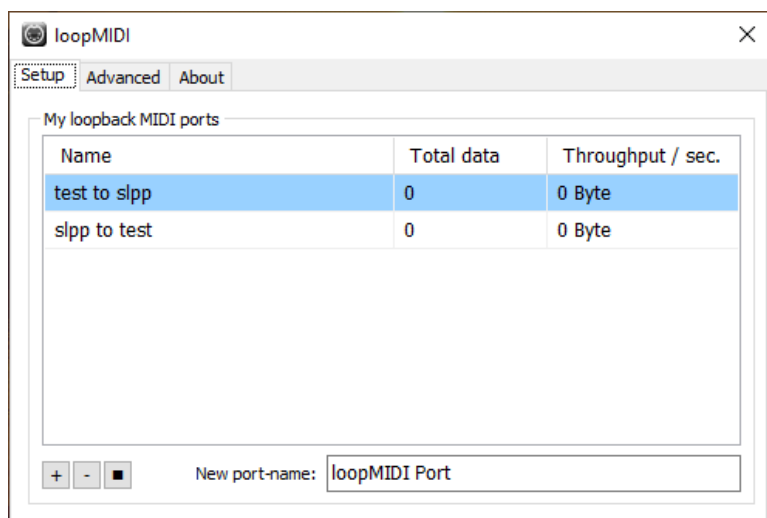
۵- پیوست‌ها



شکل ۷۰ محیط FL Studio



شکل ۷۱ محیط Ableton



شکل ۷۲ نرم افزار loopMIDI آماده برای تست

LOG : Iteration #1

LOG : Audio #0 played in 53.3 microseconds

LOG : Animation thread started in 340.4 microseconds

LOG : Animation frame #1 set in 0.3406 milliseconds

LOG : Animation frame #2 set in 50.7749 milliseconds

LOG : Animation frame #3 set in 201.602 milliseconds

LOG : Animation frame #4 set in 504.035 milliseconds

LOG : Iteration #2

LOG : Audio #0 played in 171.9 microseconds

LOG : Animation thread started in 582.8 microseconds

LOG : Animation frame #1 set in 0.5831 milliseconds

LOG : Animation frame #2 set in 51.9024 milliseconds

LOG : Animation frame #3 set in 208.178 milliseconds

LOG : Animation frame #4 set in 510.305 milliseconds

LOG : Iteration #3

LOG : Audio #0 played in 87.6 microseconds

LOG : Animation thread started in 405 microseconds

LOG : Animation frame #1 set in 0.4051 milliseconds

LOG : Animation frame #2 set in 51.386 milliseconds

LOG : Animation frame #3 set in 202.368 milliseconds

LOG : Animation frame #4 set in 503.096 milliseconds

LOG : Iteration #4

LOG : Audio #0 played in 86.8 microseconds

LOG : Animation thread started in 427.2 microseconds

LOG : Animation frame #1 set in 0.4273 milliseconds

LOG : Animation frame #2 set in 52.861 milliseconds

LOG : Animation frame #3 set in 203.23 milliseconds

LOG : Animation frame #4 set in 506.416 milliseconds

- [1] "Novation," [Online]. Available: <https://novationmusic.com>.
- [2] novation, "Launchpad programmer's reference," [Online]. Available: https://leemans.ch/latex/doc_launchpad-programmers-reference.pdf.
- [3] C. Sapp, "Essentials of the MIDI protocol," [Online]. Available: <https://ccrma.stanford.edu/~craig/articles/linuxmidi/misc/essenmidi.html>.
- [4] B. V. d. Velde, "Texus' Graphical User Interface," [Online]. Available: <https://tgui.eu/>. [Accessed 2024].
- [5] L. Gomila, "Simple and Fast Multimedia Library," [Online]. Available: <https://www.sfml-dev.org/>. [Accessed 2024].
- [6] "Boost C++ Libraries," [Online]. Available: <https://www.boost.org>.
- [7] G. Scavone, "The RtMidi Tutorial," [Online]. Available: <http://www.music.mcgill.ca/~gary/rtmidi/>.
- [8] "SDL_mixer 2.0," Simple DirectMedia Layer, [Online]. Available: https://wiki.libsdl.org/SDL_mixer.
- [9] "GoogleTest," [Online]. Available: <https://google.github.io/googletest/>.
- [10] T. Erichsen, "loopMIDI," [Online]. Available: <https://www.tobias-erichsen.de/software/loopmidi.html>.

- [11] M. Richards and N. Ford, *Fundamentals of Software Architecture An Engineering Approach*, O'Reilly, 2021.
- [12] M. Richards and N. Ford, "Microkernel Architecture Style," in *Fundamentals of Software Architecture An Engineering Approach*, O'Reilly, 2021, pp. 149-163.
- [13] M. Richards and N. Ford, "Layered Architecture Style," in *Fundamentals of Software Architecture An Engineering Approach*, O'Reilly, 2021, pp. 133-143.
- [14] "Auto-Parallelization and Auto-Vectorization," Microsoft, [Online]. Available: <https://learn.microsoft.com/en-us/cpp/parallel/auto-parallelization-and-auto-vectorization>.
- [15] D. Hearn and M. P. Baker, "Midpoint Circle Algorithm," in *Computer Graphics C Version*, Prentice Hall, pp. 98-102.
- [16] "Material Design - Dark theme," Google, [Online]. Available: <https://m2.material.io/design/color/dark-theme.html>.