Senior Design: Technology Report

Dr. Taylor

Team Sisyphus

Fall 2020

Technology Report Overview

The team plans on extending the Sisyphus table's lighting features to change colors based on the mood of music. This raises several technological issues that need to be investigated. A machine learning algorithm needs to be selected that will give the best results for the lowest processing power, as the Raspberry Pi is small, and resources are limited. In addition to selecting a performant algorithm, external methods of processing the audio will be researched. A method of recording sound needs to be added for the table to listen to music. Last, the Sisyphus app will need to be modified for a user to be able to turn the mood lighting mode on.

After a series of tests and spikes the team came to the conclusion that using an off-the-shelf algorithm (with slight modifications) would be best for the current product iteration. The team is using this repo as a starting point https://github.com/tyiannak/color_your_music_mood?ref=hackernoon.com which uses Support Vector Machines and data from the Spotify API to create a generic model that classifies audio's Mood and Valence. The team also considered alternatives of both writing a classifier from scratch or using non-ML strategies (simple feature extraction and mapping) but found these to have high risk and lacking functionality respectively.

Processing the machine learning algorithm is very intensive, as it involves audio processing, the machine learning algorithm, and changing light colors. Doing all these operations solely on the Raspberry Pi in extremely CPU intensive, so different ways of processing the algorithm were researched. One method that was tested was recording the audio through a phone's microphone, processing the audio, and sending the colors to the table. This method will not work, as audio processing is difficult on phones, and iPhones and Androids process audio differently, so it would take too much effort to implement. Currently, the team is researching running the processing on an external web server. Sending the audio to a web server for processing and having it return the color needed may have a substantial impact on performance.

The current system does not have a way of recording sound, so for this project to be usable, a way of recording audio must be added to the system. Initially, the team thought that using a phone's microphone may work, but during testing for processing the algorithm on a phone, it was found that this will not work. Another method that could be used to record sound would be to sell an "upgrade package" for the table. This would include either a USB microphone or a USB sound card. In general, a USB microphone will provide better performance, but be more expensive. The team hasn't reached a decision on what method will be used to record sound as of now.

Some modifications will need to be made to the app in order for the user to turn the mood lighting mode on. The mobile app runs on Cordova, a framework that essentially converts web app code into iOS and Android code. Because of this, changes only need to be made to the web app code, and all platforms will be able to use these changes. In the app, a button will

be added that allows the user to turn the mood lighting mode on. In addition to this, the team is planning on adding a feature that allows the user to change the color axis to colors they prefer, and change the axes based on which feature they would like to be displayed. Some usability testing was done on an initial mockup of the app. The mockup can be found here, and the results of testing can be found here.

Functional Requirements

F1: The LED strip shall turn red (as a default) when the audio corresponds to anger.

- Description: When mood lighting is enabled for the first time by a user we should provide an initial configuration of the feature so that users can get an idea of how the feature works.
- Rationale:Red is a color that people commonly correlate to anger so red correlating to anger will make a good default to the Mood Lighting feature.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled for the first time via the Sisyphus mobile app one quadrant of the grid shall show as red with the emotion anger.
- Customer Satisfaction: 7
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: None
- History: Last modified 2/8/2021

F2: The LED strip shall turn blue (as a default) when the audio corresponds to sadness.

- Description: When mood lighting is enabled for the first time by a user we should provide an initial configuration of the feature so that users can get an idea of how the feature works.
- Rationale:Blue is a color that people commonly correlate to sadness so blue correlating to sadness will make a good default to the Mood Lighting feature.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled for the first time via the Sisyphus mobile app one quadrant of the grid shall show as blue with the emotion sadness.
- Customer Satisfaction: 7
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: None
- History: Last modified 2/8/2021

F3: The LED strip shall turn green (as a default) when the audio corresponds to calmness.

- Description: When mood lighting is enabled for the first time by a user we should provide an initial configuration of the feature so that users can get an idea of how the feature works.
- Rationale:Green is a color that people commonly correlate to sadness so green correlating to sadness will make a good default to the Mood Lighting feature.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled for the first time via the Sisyphus mobile app one quadrant of the grid shall show as green with the emotion calmness.
- Customer Satisfaction: 7
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: None
- History: Last modified 2/8/2021

F4: The LED strip shall turn yellow (as a default) when the audio corresponds to happiness.

- Description: When mood lighting is enabled for the first time by a user we should provide an initial configuration of the feature so that users can get an idea of how the feature works.
- Rationale:Yellow is a color that people commonly correlate to happiness so yellow correlating to happiness will make a good default to the Mood Lighting feature.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled for the first time via the Sisyphus mobile app one quadrant of the grid shall show as yellow with the emotion happiness.
- Customer Satisfaction: 7
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: None
- History: Last modified 2/8/2021

F5: The mood lighting mode shall be enabled via the Sisyphus mobile app.

- Description: An option needs to be available for the user to enable the mood lighting feature.
- Rationale:All options for the Sisyphus table are controlled via the Sisyphus mobile app so we should put the option to enable mood lighting where all of the other options for the table are. Also, since there are other light patterns users shall be able to enable/disable the feature on demand in order to select other light patterns.
- Originator: Andrew Wojciechowski
- Fit Criterion: When a user navigates to the settings page in the Sisyphus mobile app an option to enable mood lighting shall be available to the user as a checkbox.
- Customer Satisfaction: 8
- Customer Dissatisfaction: 3
- Priority: High
- Dependencies: None

- History: Last modified 2/8/2021

F6: The table shall collect audio samples when the mood lighting mode is enabled.

- Description: When mood lighting is enabled samples shall be collected in order to determine what color the LEDs on the table should be turned to.
- Rationale: The only option that requires audio input is mood lighting so audio samples should only be collected when mood lighting is enabled.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled the table shall collect 5 second samples to use for the mood lighting feature.
- Customer Satisfaction: 8
- Customer Dissatisfaction: 10
- Priority: High
- Dependencies: None
- History: Last modified 2/8/2021

F7: The color shall be changed through already existing externally facing APIs

- Description: The table has APIs for setting the colors already and the mood lighting feature should take advantage of those existing APIs.
- Rationale:It will be easier to use the existing APIs to change the color instead of writing our own APIs.
- Originator: Andrew Wojciechowski
- Fit Criterion: When the mood lighting algorithm has calculated a color to set on the table. The feature shall use the existing light APIs on the table to set the primary color of the lights on the table.
- Customer Satisfaction: 3
- Customer Dissatisfaction: 3
- Priority: Low
- Dependencies: None
- History: Last modified 2/8/2021

F8: The user shall be able to select custom colors that map to different moods

- Description: Customization shall be available in order for users to change which colors coordinate to which moods.
- Rationale: People may correlate moods to color differently so a feature shall be provided in order to select custom color to correlate to different moods.
- Originator: Andrew Wojciechowski
- Fit Criterion: When mood lighting is enabled in the Sisyphus mobile app the color axises shall be displayed and the colors of each of the quadrants shall be customizable in the Sisyphus mobile app.

- Customer Satisfaction: 8
- Customer Dissatisfaction: 5
- Priority: High
- Dependencies: None
- History: Last modified 2/8/2021

Nonfunctional Requirements

NF1: The audio system shall be installable within 20 minutes, by a user who has minimal experience with technology and should require similar technical skills to that necessary for installing the Sisyphus LED Kit.

- Description: The modification necessary to add audio recording and processing to the table should be minimal so that Sisyphus industries can market this addition as an upgrade kit to their users
- Rationale: Keeping install time below 20 minutes means that the user can use the new functionality quickly and does not require more of their time then necessary (since we assume most Sisyphus table owners have minimal technical experience)
- Originator: R. George Burkhardt
- Fit Criterion: The installation of the audio module (i.e. microphone, Pi processor) should take no more than 20 minutes. The upgrade kit shall be of a similar technical level to the LED install kit.
- Customer Satisfaction: 8
- Customer Dissatisfaction: 10
- Priority: High
- Dependencies: N/A
- History: Last Modified 2/9/2021

NF2: The system shall alert the user that any and all audio recordings are not saved.

- Description: A disclaimer should be available to the user saying than no audio will be recorded and stored on external servers if any web services are used
- Rationale: In keeping with ethical software practice, the system should ensure the user is aware of how the audio is used.
- Originator: R. George Burkhardt
- Fit Criterion: A disclaimer is immediately available to the user upon installing the audio upgrade kit
- Customer Satisfaction: 7
- Customer Dissatisfaction: 3
- Priority: High
- Dependencies: F5
- History: Last Modified 2/9/2021

NF3: Any new additions to the UI will abide by the current UX design patterns and standards

- Description: Any new UI components (e.g. dropdowns, icons, etc.) should be styled in similar ways and any interactions should match how the app currently operates (e.g. checkboxes for toggling options)
- Rationale: The app should provide a cohesive user experience between old features and new features. Changes in styles and patterns can be jarring to users, thus it should be avoided or kept to a minimum
- Originator: R. George Burkhardt
- Fit Criterion: New UI elements will be styled with the same colors, fonts, and icon packs that are currently in the app. New UI elements will be kept to a minimum.
- Customer Satisfaction: 5
- Customer Dissatisfaction: 5
- Priority: Medium
- Dependencies: F6, F8
- History: Last Modified 2/9/2021

NF4: The audio feature shall not negatively impact any existing functionality

- Description: Any changes made should be done in such a way that they do not regress the system
- Rationale: Any changes will need to exist side by side tables that do not have audio/lighting functionality thus any changes must be atomic as to not ruin the product experience for other users
- Originator: R. George Burkhardt
- Fit Criterion: The table will be able to function normally with or without audio input
- Customer Satisfaction: 8
- Customer Dissatisfaction: 10
- Priority: High
- Dependencies: N/A
- History: Last Modified 2/9/2021

Technology Evaluations

Goal

We are hoping to create a new mode of interaction with the table in which users play music and the table calculates the emotion of the piece and displays an appropriate color to enhance the emotional impact of the piece. In doing so, we need to make a number of fundamental changes to the existing codebase and potentially to the hardware setup. In doing so, we created a list of qualities that we seek to uphold with the additions. The qualities are as follows:

- Ensure the Pi's resource usage does not increase to unsustainable amounts (i.e. CPU > 50% average utilization)
- Ensure that the code running on the table is easily maintained, and that code we add does not contribute to existing complexity

- Ensure that the new functionality does not negatively interfere with the current functionality
- Ensure that any potential changes will be affordable to Sisyphus Industries and its customers
- Ensure that the color displayed on the table is accurate to the mood of the music being played

Technology Changes

The existing technology stack of the project centers almost entirely around the Raspberry Pi 3 model A+ used by the table. This module is responsible for running 3 separate software repositories (a web server to process requests to alter the table, a web application that serves as a user interface, and a 'master project' that manages the table's state and starts up both of the previous repositories). These projects are mostly written in javascript, although code that directly interacts with the lights on the table is written in python and utilizes the rpi_ws281x library to control the Adafruit Neopixel ® RGBW LED's.

Changes to this architecture to support the addition of a "mood coloring based on audio" option are centered around the implementation of the mood lighting algorithm and hardware used to collect audio.

The implementation for using raw audio to control and manipulate the table's lights has the following options, and the general module will be referred to as "audio processing":

- Writing a machine-learning algorithm from scratch to process music and output a valence/energy score from a raw audio signal, which would in turn be mapped to a color by way of a 2d-grid. A dataset would have to be created to train the algorithm, ostensibly from repeatedly calling the Spotify API.
- Using an off-the-shelf algorithm. Only one (working) algorithm was found with licensing permitting free modification and distribution, published by senior researcher Theodoros Giannakopoulos as part of a mood-lighting demo.
- Utilizing a library to simply pulse a table's lights with audio and beat detection

Even with a suitable implementation, the architecture of the implementation had many options as well:

- Extending the Sisyphus table to include a microphone, then performing audio processing code on the Sisyphus table directly. The lights could be directly manipulated from the table via the python libraries available.
- Extending the Sisyphus table to include a microphone, then streaming audio data to a server that would perform audio processing. Another web request would be made to the sisyphus API to change the table's lights accordingly.
- Using a mobile phone to collect audio and perform audio processing, again using web requests to the sisyphus API to change the table's lights
- Using a mobile phone to collect audio, then streaming audio data to a server that would perform audio processing. A web request would (again) need to be made to the Sisyphus API to change the table's lights.

- Using a desktop application to collect audio and perform audio processing. A web request would need to be made to the Sisyphus API to change the table's lights.

In addition to audio processing and architecture options, applicable libraries for collecting audio on each platform mentioned above must be considered:

- PortAudio vs. ALSA
    - These libraries are for operating-system audio capture on *nix systems, including the Raspberry pi, which is being considered as a recording platform.
- PyAudio vs. Sounddevice vs. Other python libraries (if python is the chosen language)
    - These are Python interfaces to the above libraries, important for recording audio on either *nix systems, or in the case of PyAudio, Windows 10.

Finally, hardware to support extending the Sisyphus table, if chosen, would also present several options.

- USB microphones vary in price range and are readily available at many retailers. The Raspberry Pi 3A+ has 4 USB ports which are not currently used. USB microphones typically have better sound quality which means better input and output for the machine learning algorithm. However, the two trade offs are cost and aesthetics (USB microphones are typically more bulky)
- USB sound cards allow for a Mic-In line (which currently doesn't exist on the Raspberry Pi) which would allow for 3.5mm (or similar) microphone. This combo is typically cheaper but can run into issues with quality (USB Sound Cards are usually limited to a mono channel). Also, smaller microphones are easier to blend in with the table so aesthetics are less of a concern.
- Dedicated Raspberry Pi HATs for sound recording / processing, which are another option, however a hat is already used on the Pi for communicating with the Arduino module (https://respeaker.io/).
- Modern smartphones that have microphones and APIs that allow for audio streaming. One option is to enable the phone to be the source of audio and stream it back to the table for processing. This has the advantage in that most (if not all) Sisyphus Table owners have a smartphone (probably the same that was used to set it up) so this option would reduce the overall cost of the upgrade

Decision Criteria

The final architecture decision was made by utilizing the following criteria:

- Is the option well-supported by an active community? Is there a guarantee of long term support?
- Does the option integrate well with the existing infrastructure?
- Can the architecture be replicated and deployed to customers easily?
    - How seamless is integration?
    - How many steps are needed from start (getting the new kit) to finish (having audio affect the lights)?
    - How much time is needed for setup?

    - Specialized equipment (screwdriver, soldering gun, etc.)
    - Potential damages from screwed up?
    - How expensive is the replication?
- If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?
- If run on an external device and utilizing web requests and/or audio streaming, would performance constraints permit its practical usage and/or ruin the experience for the user?
- Is the option feasible? Can a prototype be created to demonstrate this?
- Does the option provide a potentially valuable and interesting new feature for users to appreciate?

In cases where the decision criteria for two alternatives is roughly equivalent (both appear to be good options), a pros vs. cons analysis will be performed between the two libraries.

Audio Processing Options & Evaluation

- Writing a machine-learning algorithm from scratch to process music and output a valence/energy score from a raw audio signal, which would in turn be mapped to a color by way of a 2d-grid. A dataset would have to be created to train the algorithm, ostensibly from repeatedly calling the Spotify API.

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	No. There is significant pain involved in maintaining an algorithm written from scratch--creating, maintaining the datasource, testing the algorithm, etc.	5
Can the architecture be replicated and deployed to customers easily?	Yes - dependent on how the algorithm is built	5
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	Yes or no, dependent on how the algorithm is built	5
Is the option feasible?	Not in the current timeframe, unfortunately. While this option might be an option if there existed a full year's worth of time, ⅓ of a year is insufficient. See prototype **####.**	5

If run on an external device and utilizing web requests and/or audio streaming, would performance constraints permit its practical usage and/or ruin the experience for the user?	Yes or no, dependent on how the algorithm is built	5

- Using an off-the-shelf algorithm. Only one (working) algorithm was found with licensing permitting free modification and distribution, published by senior researcher Theodoros Giannakopoulos as part of a mood-lighting demo.

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	Yes and no. There is not an active community, but the researcher in charge of the algorithm is extremely responsive, responding to both Github issues filed and providing information about dataset acquisition over email.	5
Can the architecture be replicated and deployed to customers easily?	Yes with caveats. The algorithm in question is written in Python, so can realistically be deployed anywhere that Python (with supporting C binaries) can be run.	5
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	Yes, performance tests indicate 75% CPU usage.	2
Is the option feasible?	Yes.	2, 4, 5
If run on an external device and utilizing web requests and/or audio streaming, would performance constraints permit its practical usage and/or ruin the experience for the user?	Most likely not. Audio streaming to and from a server can be done relatively quickly.	4

- Utilizing a library to simply pulse a table's lights with audio and beat detection (*future work; largely unevaluated*).

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	Yes, in Python, librosa has widespread support. Additionally, PyAudioAnalysis appears to have this ability.	n/a
Can the architecture be replicated and deployed to customers easily?	Yes with caveats. The libraries mentioned are written in Python, so can realistically be ported anywhere where Python (with C) is permitted.	n/a
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	Undetermined as of 2021/2/10.	n/a
Is the option feasible?	Undetermined as of 2021/2/10.	n/a
If run on an external device and utilizing web requests and/or audio streaming, would performance constraints permit its practical usage and/or ruin the experience for the user?	Undetermined as of 2021/2/10.	n/a

Audio Processing Options & Evaluation

- Extending the Sisyphus table to include a microphone, then performing audio processing code on the Sisyphus table directly. The lights could be directly manipulated from the table via the python libraries available.

Criteria	**Evaluation**	**Applicable Prototypes**

Is the option well-supported by an active community? Is there a guarantee of long term support?	N/A - never been done before	2
Can the architecture be replicated and deployed to customers easily?	This would require customers to purchase an add-on kit and assemble the microphone on the table themselves.	2
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	Yes. Current tests indicate CPU usage as high as 75%.	2
Is the option feasible?	Yes, although not performant.	2
If run on an external device and utilizing web requests and/or audio streaming, would performance constraints permit its practical usage and/or ruin the experience for the user?	N/A for this -- no external device	n/a

- Extending the Sisyphus table to include a microphone, then streaming audio data to a server that would perform audio processing. Another web request would be made to the sisyphus API to change the table's lights accordingly.

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	N/A - never been done before	4
Can the architecture be replicated and deployed to customers easily?	Yes with caveats. The libraries mentioned are written in Python, so can realistically be ported anywhere where Python (with C) is permitted.	4

If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	n/a	n/a
Is the option feasible?	Yes, see prototype	4
If run on an external device and utilizing web requests and/or audio streaming, would performance constraints permit its practical usage and/or ruin the experience for the user?	Undetermined as of 2021/2/10.	4

- Using a mobile phone to collect audio and perform audio processing, again using web requests to the sisyphus API to change the table's lights

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	No. There is not much of a community at all for raw audio processing on android; similar with Swift. Closest match on Java was a library that had been ported from Python (JLibrosa) for which no active community exists.	3
Can the architecture be replicated and deployed to customers easily?	With an app, yes.	3
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	n/a	n/a
Is the option feasible?	Yes, but not within the existing timeframe. See prototype for in-depth explanation.	3

If run on an external device and utilizing web requests and/or audio streaming, would performance constraints permit its practical usage and/or ruin the experience for the user?	n/a	n/a

- Using a mobile phone to collect audio, then streaming audio data to a server that would perform audio processing. A web request would (again) need to be made to the Sisyphus API to change the table's lights.
- Using a desktop application to collect audio and perform audio processing. A web request would need to be made to the Sisyphus API to change the table's lights.

  Due to similarity the above two are paired together.

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	No. There is not much of a community at all for raw audio processing on android; similar with Swift. Closest match on Java was a library that had been ported from Python (JLibrosa).	4
Can the architecture be replicated and deployed to customers easily?	Will require research into cloud resources, but for now, yes. A free instance of Heroku provides all of the availability our web server needs. After rolling this out to customers, the backend will likely require some scaling.	4
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	n/a	n/a
Is the option feasible?	Yes.	4
If run on an external device and utilizing web requests and/or audio streaming, would performance constraints permit its practical	Yes.	4

usage and/or ruin the experience for the user?		

Audio Collection Library Options & Evaluation

- PortAudio vs. ALSA
    - These libraries are for operating-system audio capture on *nix systems, including the Raspberry pi, which is being considered as a recording platform.
- Ossaudiodev vs. PyAudio vs. Sounddevice vs. Other python libraries (if python is the chosen language)

Evaluation for ossaudiodev:

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	Yes, there is support available, however, the library has been deprecated for modern linux distributions.	1
Can the architecture be replicated and deployed to customers easily?	Yes, but although there is native support for the library in Python, use of the library has been deprecated for modern linux distributions.	1
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	No, not for pure audio recording.	1
Is the option feasible?	Yes, but not recommended.	1

Evaluation for PyAudio (interfaces with PortAudio):

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	Yes.	1
Can the architecture be replicated and deployed to customers easily?	Yes; officially supported python packages available.	1
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of	No, not for pure audio recording.	1

(caused missed deadlines) or cause overheating?		
Is the option feasible?	Yes	1

Evaluation for sounddevice (interfaces with PortAudio & uses numpy):

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	Yes.	1
Can the architecture be replicated and deployed to customers easily?	Yes; officially supported python packages available.	1
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of (caused missed deadlines) or cause overheating?	No, not for pure audio recording.	1
Is the option feasible?	Yes	1

Evaluation for pyalsaaudio

Criteria	**Evaluation**	**Applicable Prototypes**
Is the option well-supported by an active community? Is there a guarantee of long term support?	Yes.	1
Can the architecture be replicated and deployed to customers easily?	It depends. The library force the use of PCM, which is a pro or a con depending on where and on what architecture it is used.	1
If run on a Raspberry Pi (Sisypshus table), would the solution impede the CPU of	No, not for pure audio recording.	1

(caused missed deadlines) or cause overheating?		
Is the option feasible?	Yes, although extremely low-level. May leave too much power in the developer's hands.	1

Because the options above are in many ways similar, and divided by a series of nuances, the following pros and cons evaluation was performed, except for ossaudio, which was determined in the initial investigation to be a poor choice due to its deprecation:

pyAudio (interfaces with PortAudio):

Pros	**Cons**
Support blocking and callback streaming	Requires PortAudio
Used in spikes with success	
Supports both OSS and ALSA	

sounddevice (interfaces with PortAudio, uses numpy to handle data):

Pros	**Cons**
Exceptional level of support available	Requires PortAudio
Smaller API footprint than pyAudio	
Support for both callbacks and streaming	

Pyalsaaudio (very low level wrapper for ALSA interface)

Pros	**Cons**
Specific to Linux environment without the need for a Sound API	Extremely low level API leaves too much control to the developer
Forces use of the PCM	

Conclusion: pyAudio with PortAudio is the sound API of choice. pyAudio was chosen over the sounddevice interface because pyAudio has a larger, more customizable and extensible API as well as successfully being used in sample code and spikes. pyAudio is also a more popular package than sounddevice (according to the latest pip download statistics), hopefully meaning better and longer term support.

Hardware Options & Evaluation

For each hardware option, the decision criteria defined above was considered to be met or not applicable, since hardware does not have the same properties as a software product (e.g., running hardware 'in the cloud' is not logical). For this reason, only a pros and cons analysis was performed.

- USB microphones vary in price range and are readily available at many retailers. The Raspberry Pi 3A+ has 1 USB port, but can be extended via a hub. USB microphones typically have better sound quality which means better input and output for the machine learning algorithm. However, the two tradeoffs are cost and aesthetics (USB microphones are typically more bulky).
- USB sound cards allow for a Mic-In line (which currently doesn't exist on the Raspberry Pi) which would allow for 3.5mm (or similar) microphone. This combo is typically cheaper but can run into issues with quality (USB Sound Cards are usually limited to a mono channel). Also, smaller microphones are easier to blend in with the table so aesthetics are less of a concern.
- Dedicated Raspberry Pi HATs for sound recording / processing, which are another option, however a hat is already used on the Pi for communicating with the Arduino module (https://respeaker.io/).
- Modern smartphones that have microphones and APIs that allow for audio streaming. One option is to enable the phone to be the source of audio and stream it back to the table for processing. This has the advantage in that most (if not all) Sisyphus Table owners have a smartphone (probably the same that was used to set it up) so this option would reduce the overall cost of the upgrade

Decision

We ultimately decided to proceed with using the existing model found here: https://github.com/tyiannak/color_your_music_mood?ref=hackernoon.com. This model would be fed the audio features of the recording device, then process the valence and arousal of the segment, which would then be correlated to color by a user-adjustable conversion in the hopes of accounting for the variance in other people. The team briefly explored performing the ML analysis and audio recording on a phone but this was found to be risky and untenable.

In terms of audio input the team settled on using PortAudio which is a cross-platform API, with Python bindings via the pyaudio library, that sits on top of ALSA (Advanced Linux Sound Architecture). It was also discovered through testing that USB microphones are the better choice since the yield higher quality audio streams and are more configurable (i.e. sampling rate, frequency range, etc.) which is needed to match the expected input type for our chosen ML algorithms (which was trained on 8KHz audio clips).

Rationale: Summary

Choosing an out-of-the-box machine learning model meets the criteria of a relatively well-supported option. It also meets the criteria of bringing maximum value to the user as it is more likely to produce a robust, well-featured end result.

Choosing PortAudio, Python, and pyaudio meets the baseline criteria of preserving current functionality of the Raspberry Pi while also allowing for easy integration with the lighting module (also controlled by Python libraries) and the chosen ML model which was trained and built in Python.

Choosing USB microphones over other alternatives meets the criteria of replicating the setup/architecture on the customer's side. There would be less parts to manage and less steps. Although USB microphones are more expensive, the overall ease of use is a justified trade off (especially considering requirement NF1)

Prototypes

Several different prototypes and spikes were performed to acquaint the team with the ML algorithms and test out different architectures and setups so that subsequent technology decisions could be justified. Goals, scope, and knowledge acquired for each of these major spikes (or prototypes) are described below.

1. Choosing Audio Setup

https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/lights-audio-input

With this prototype the team wanted to test the optimal audio setup and explore different low level and high level audio interface libraries. One of the first things done in this prototype was to explore how audio may be recorded on a Raspberry Pi via the Raspbian Linux distribution. The team discovered that both OSS (deprecated) and ALSA are available on the Raspberry Pi. There are two Python libraries that wrap these interfaces, ossaudiodev (a native Python module) and pyalsa. Ultimately, it was decided that these libraries were too "low-level" and would require a lot of development time to produce useful code. Next, the team evaluated higher level C/C++ libraries like PortAudio and PulseAudio which allow for cross platform development and the use of "higher level" Python libraries like pyaudio and sounddevice. PusleAudio was found to be too complex and overkill for the requirements of the new feature but PortAudio and pyaudio were found to be appropriate in terms of API footprint and feature set (see above for additional details). Finally, the team tested different audio hardware devices - primarily USB microphones and USB sound cards with 3.5mm microphones. Via a series of tests with passing audio to the machine learning model, it was decided that USB microphones were the better choice since they yielded better audio quality on average at a cost that wasn't too expensive (versus the wide range of audio cards and microphones that would yield poor quality audio/data on the lower end side). So, for the purposes of

developing the feature, the team decided on using USB microphones with PortAudio and pyaudio to record and process audio.

2. Exploring ML on Pi

https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/sound-output-to-color-spike
https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/sisbot-lights-from-audio-input
https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/mood-lighting-performance
With this prototype the team wanted to learn if we could run the machine learning algorithm to convert audio data into a color on a Raspberry PI. One of the first things we did with this prototype is that we investigated running a machine learning algorithm to convert audio data from a wav file to a color and then updating a light strip. We found that this approach was very feasible but this was causing the Raspberry PI to overheat. We then converted the prototype to get the audio data from a microphone and make a web request to the lab table which was also very feasible but it still caused the Raspberry PI to overheat. We did a CPU analysis of this algorithm on the PI and we found that it was impeding the CPU of the Raspberry PI. We then got this down to about 60% once we removed the library OpenCV from the algorithm but we wanted to get that down further. Finally, one of the last things we investigated having the PI only listen to the audio data and stream that to a Python server running elsewhere which had a very minimal impact on the Raspberry PI. So, even though running the ML on a Raspberry PI was feasible we decided that running the ML elsewhere was a better option in order to not impede the CPU on the table.

3. Exploring ML on Phone

https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/phone-ml-demo
With this prototype the team wanted to learn if we could run a machine learning algorithm to convert audio data into a color on a phone. One of the first things that we looked into if it was possible to leave the existing code in Python but this was not feasible since there are very few mobile app libraries out there that support Python and there were none that support installing pip dependencies. The next thing we tried is extracting all of the audio features on the phone and streaming them to a Python server to do the classification. This was not feasible as well since there's a lack of libraries for feature extraction in Java which was the language we used. Also, it was very difficult to get the features similar to what pyAudioAnalysis was expecting and linear algebra in Java was more difficult than Python. This technically is possible and this would have not impeded the CPU on the table's Raspberry PI but given the difficult feasibility we determined that the risk was too high for this option compared to the alternatives such as having the ML completely be on a web server.

4. Exploring ML on Server (ML as a Service)

https://gitlab.com/msoe.edu/sdl/sd21/sisyphus/mood-lighting-server

With this prototype, the team wanted to test if the machine learning audio processing could be moved from a local environment to an external server without degrading performance. The team was able to split existing spikes into a client and server script. The server script was uploaded to Heroku while the client script remained on a microphone equipped Pi. After sorting out VPN issues (due to remote development policies in lieu of COVID-19) the team was able to take audio from a Pi, send it to the server, receive a RGBA value, and then forward that to the team's table to update it's color. The latency sat at about 2 second which isn't optimal for "real-time" audio processing but there are still many optimizations available (such as a local server rather than one in the cloud) and the goal for the spike was met, proving that a server architecture is feasible.

5. Mood Lighting & Alternative Algorithms

- Attempted to investigate an algorithm for correlating music to color, with the goal of delivering a feasible algorithm to turn an audio into a color.
- Addressed writing a home-grown solution
- Addressed writing a solution that does not utilize any "machine learning" techniques but rather raw audio feature extraction / fourier transformation.
- Ultimately decided that the risk and cost of maintaining our own algorithm was quite high: the development team lacks skills and time to acquire sufficient skills to author an algorithm effectively, especially without a readily available dataset.
- Using a non-machine learning technique was found to be unreliable due to difficulty quantizing the relation between common audio features and a mood output.