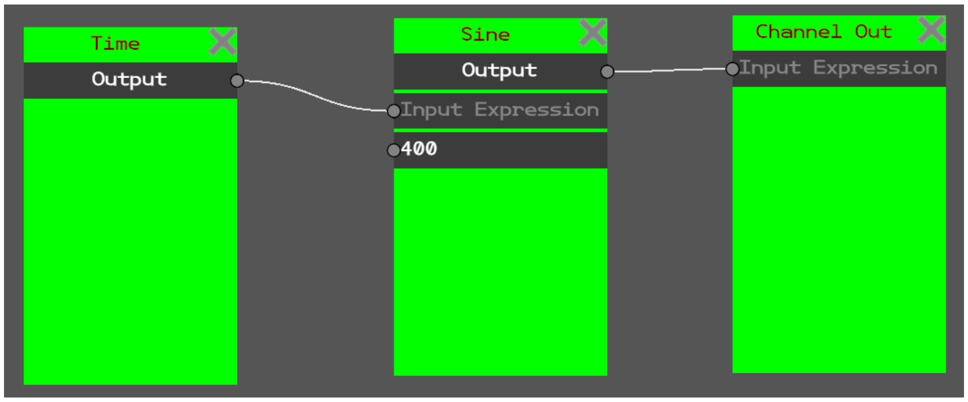
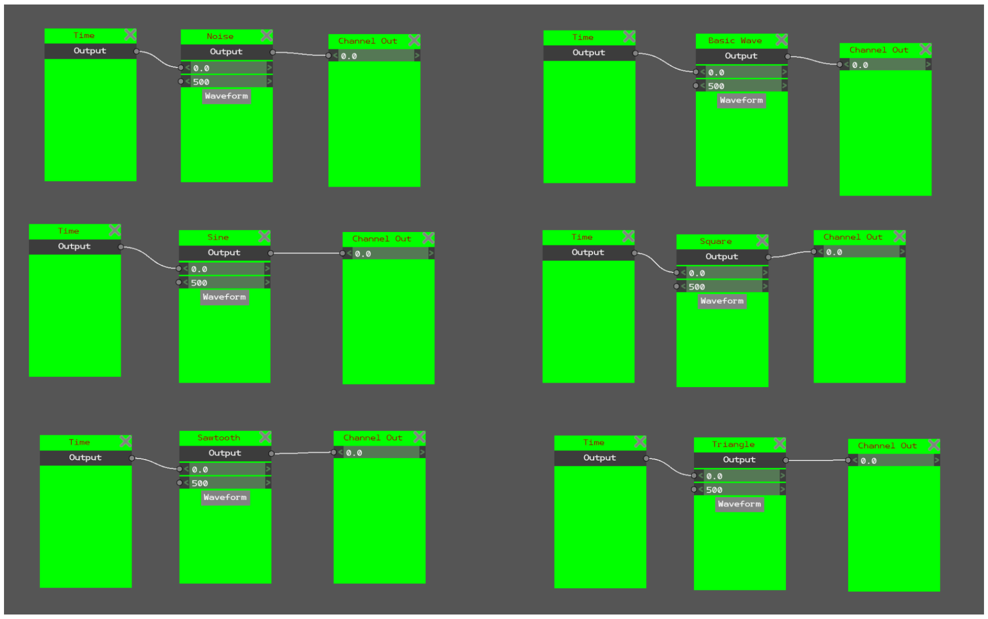
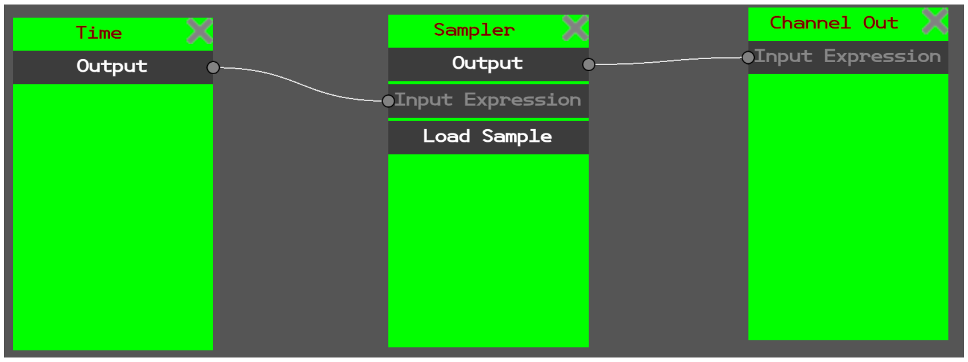
1. **Signal Routing:**
    a. **Scenario**: The user wants to route audio from one node to another.
    b. **Steps**:
        i. Set up a time node to generate a basic audio signal.
        ii. Connect the output of the time node to a processing node.
        iii. Connect the output of the processing node to the channel-out node.
        iv. Verify that the audio signal is correctly routed and processed
    c. **Results**: The following graph depicts a 400 Hz frequency. When playing it side by side with another 400 Hz sound, the 2 sound identical, overlapping each other. Additional tests led up to 1000 Hz, finding the same conclusion. That concludes this test as a success with no issues.



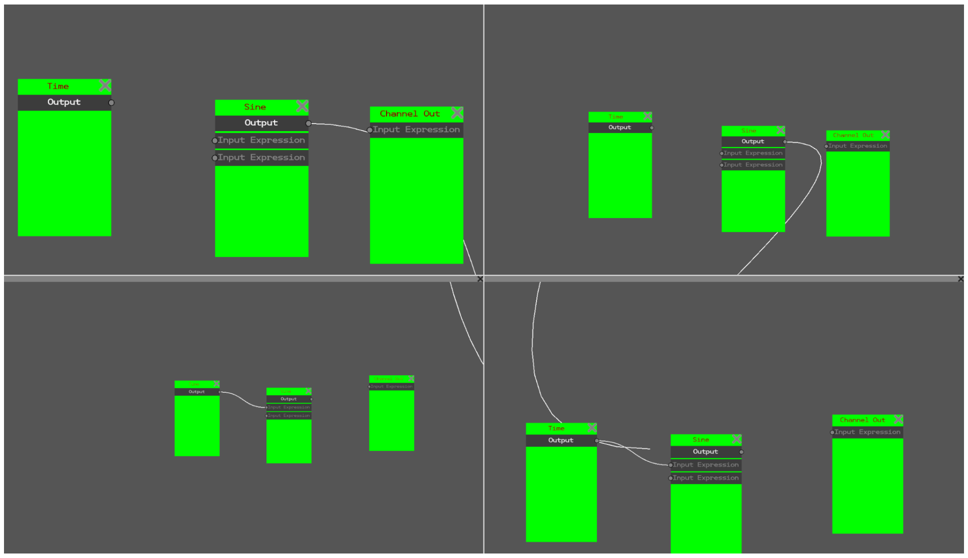
2. **Sine Wave Testing**
    a. **Scenario**: The user wants to route audio from one node to another.
    b. **Steps**:
        i. Set up a time node to generate a basic audio signal.
        ii. Connect the output of the time node to a processing node.
        iii. Connect the output of the processing node to the channel-out node.
        iv. Verify that the audio signal is correctly routed and processed
    c. **Results**: The following graph depicts a 500 Hz frequency being played in every wave form possible. When playing it side by side with another 500 Hz sound of equivalent type, the 2 sound identical, overlapping each other. That concludes this test as a success with no issues.



3. **Basic Audio Playback:**
   a. **Scenario**: The user wants to play a simple audio file.
   b. **Steps**:
      i. Load a short audio file into the sampler.
      ii. Connect the sampler node to the channel out node.
      iii. Play the audio file and verify that it plays back correctly.
   c. **Results**: The following graph below is playing a 139 octave up long loop 120 bpm. When comparing DAWdle's audio performance to the provided audio, most devices will produce the correct output. However, it would seem that some devices would create a slightly different sound as if you were playing the audio in an enclosed space that would produce an echo chamber. One of the possible reasons for this would be that the sample rate of the device would not match the sample rate of the sound file, resulting in that echo chamber sound.
   d. 

4. **User Interface Layout:**
   a. **Scenario**: The user wants to customize the layout of the app.
   b. **Steps**:

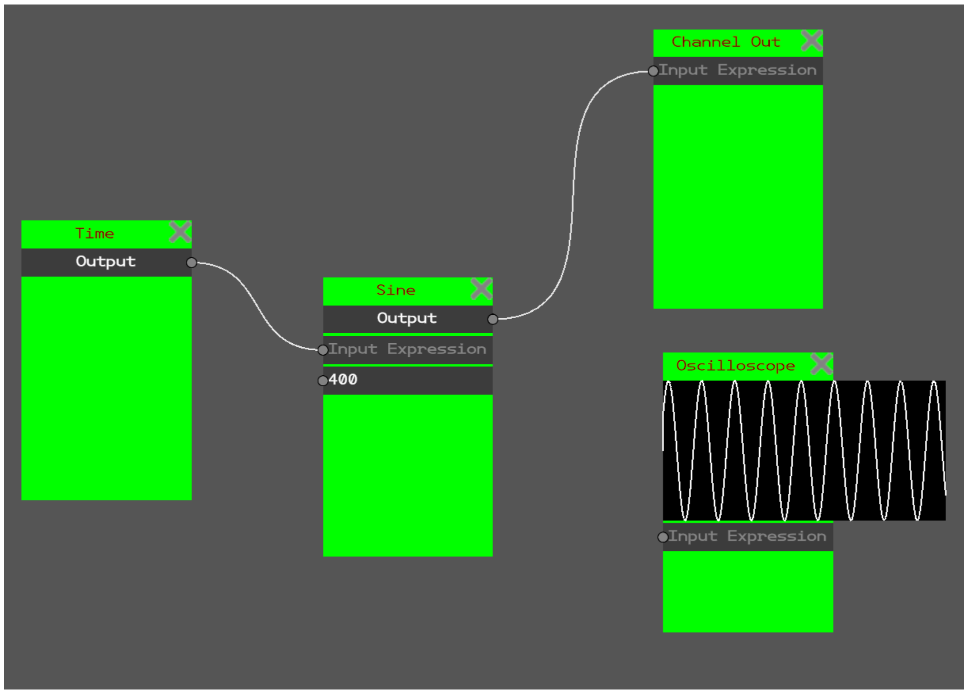
        i. Choose between horizontal and vertical split for the screen layout.
        ii. Adjust the size and position of nodes on the screen.
        iii. Verify that the layout changes are applied as expected and do not obstruct usability.
    c. **Results**: Horizontal and Vertical splits on the screen work fine when connecting the audio as if there was one screen. However, visually, lines are drawn when connecting nodes that may be seen on other screens as the program attempts to draw the lines for the nodes. Testing has revealed that there are some issues with the UI that may need to be worked out.



5. **Real-Time Parameter Adjustment:**
    a. **Scenario**: The user wants to adjust settings while the audio is playing.
    b. **Steps**:
        i. Play an audio file using the sampler node.
        ii. While the audio is playing, adjust the parameters of the file playing
        iii. Verify that the changes to the parameters take effect immediately and affect the audio output in real time.
    c. **Results**: Using the same setup as test 1 with the basic audio playback, we can determine that the sampler node can switch between the audio files given. Using 139 octave up long loop 120 bpm and 141 octave up long loop 120 bpm. Additionally, the audio does appear to start over from the which indicates a successful switch, though there does seem to be a minor overlap in the initial milliseconds after switching files. In conclusion, being able to adjust the sampler node in real time seems like a success.
6. **Visual Representation Accuracy:**
    a. **Scenario**: The user wants to compare visual waveforms with audio output.
    b. **Steps**:

i. Play an audio file using the sampler node.
ii. Observe the waveform displayed on the oscilloscope.
iii. Listen to the audio output and compare it with the visual representation
iv. Verify that the waveform accurately reflects the audio signal

c. **Results**: The following graph depicts a 400 Hz frequency. By pausing the oscilloscope we were able to see the wave formed by the frequency. When comparing it to how others have it looking, it's relatively accurate. Without numbers and definitions of where we are, it is difficult to determine 100% accuracy.



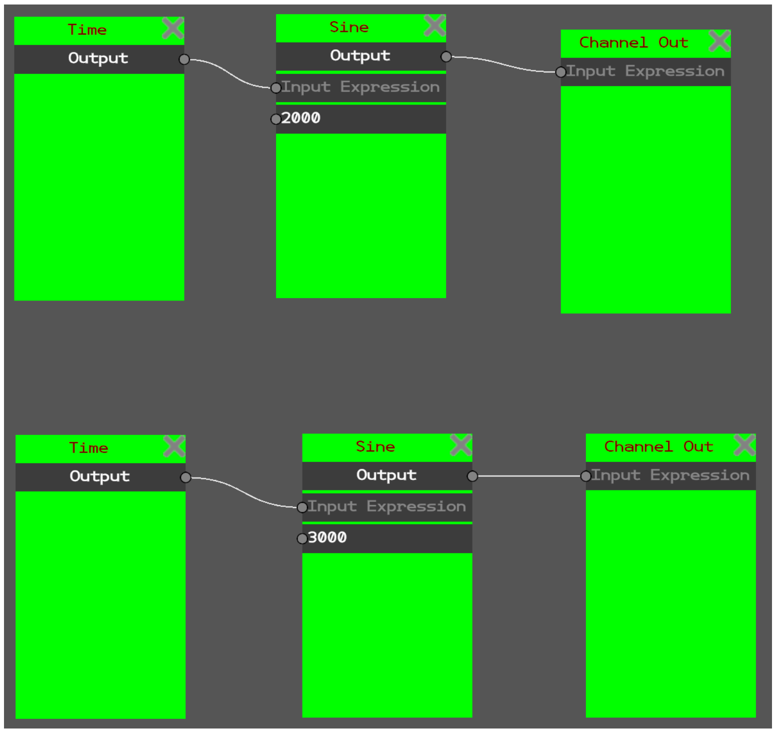
7. **Math Node Works Correctly**

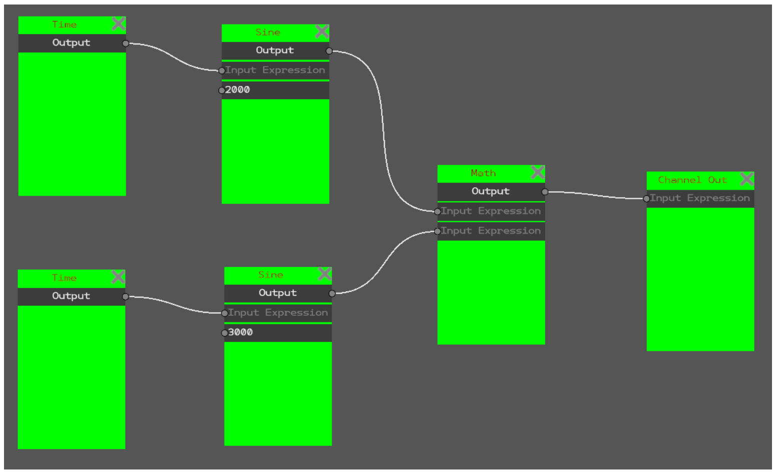
a. **Scenario**: The user wants to determine if the math nodes work correctly.

b. **Steps**:

i. Create two graphs using the time, sine, and channel-out nodes.
ii. Add the 2 sine waves on one graph
iii. On the other graph, send both sine waves to a channel out node.
iv. Compare the audio of both graphs to ensure correct output.

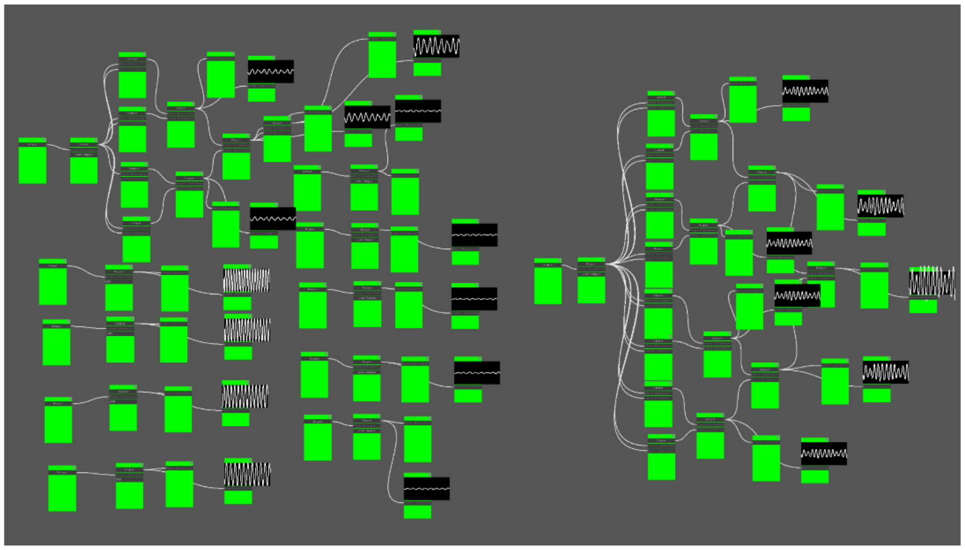
c. **Results**: The following graph depicts a 2000 Hz Sine wave and a 3000 Hz Sine wave being added together in 2 different manners. On the left, they are added as 2 separate waves after reaching channel-out nodes. They are added and sent to the channel out node as 1 wave. When analyzing the sound, they produce identical audio.



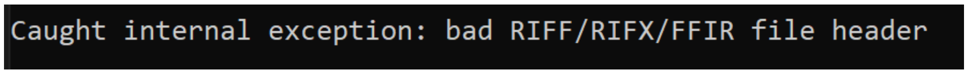


8. **Performance Testing:**
    a. **Scenario**: The user wants to test the app's performance with multiple nodes and complex signal processing.
    b. **Steps**:
        i. Create a chain of multiple nodes, including generators, transformations, and output nodes.
        ii. Load a complex audio file or generate a complex signal.
        iii. Play the audio and observe for lag, glitches, or performance issues. Verify that the app maintains smooth operation even with heavy processing loads.

c. **Results**: The following graph depicts a complex node system utilizing every type of available node. While the audio runs fine, some issues can be seen. First, the oscillations may clip out of the oscilloscope due to size. Next is how some audio becomes muted as other audio types become more prevalent. Lastly, if the graph becomes too big, leaving the application and leaving it idle makes it prone to crashing.



9. **Correctly Processing Nodes to Audio:**
    a. **Scenario**: The user wants to determine if the audio will be only when something valid is created and will not produce anything that will break stuff.
    b. **Steps**:
        i. Choose the processing nodes you want to test.
        ii. Connect input nodes to the processing nodes. These input nodes could be sources of audio signals such as oscillators or channel-out nodes.
        iii. Intentionally create a scenario that will now break the audio and determine if the program correctly stops playing audio in that situation.
        iv. Optionally, if an error occurs during processing, observe how the app handles it.
    c. **Results**: When playing a sampler node with a corrupted audio file, the following internal exception is caught. Though the application does freeze and hang, this test has revealed that playing invalid audio files will just cause an internal exception and loss of any work currently done.



```
Caught internal exception: bad RIFF/RIFX/FFIR file header
```

**10. Error Handling and Recovery:**

a. **Scenario**: The user encounters errors or unexpected behavior while using the app and wants to understand how the app handles these situations.

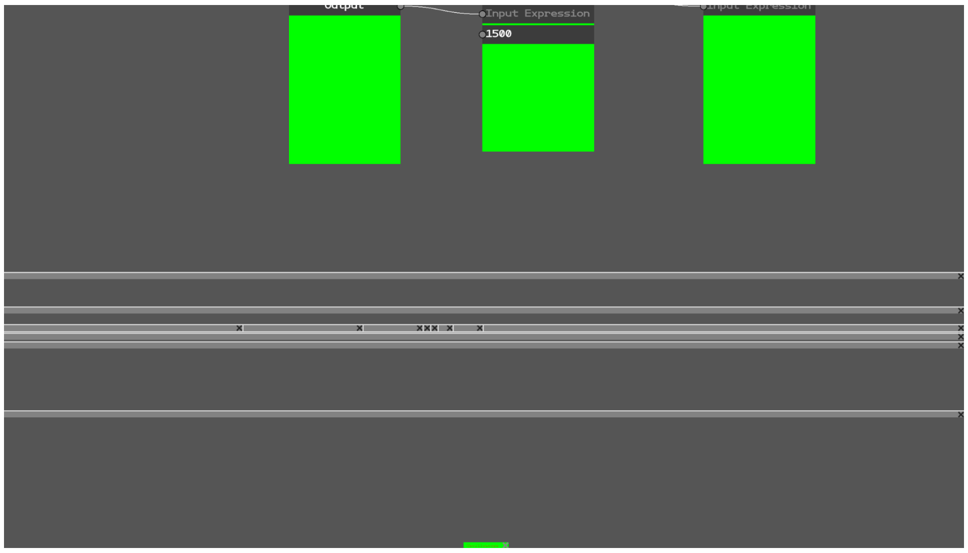
b. **Steps**:

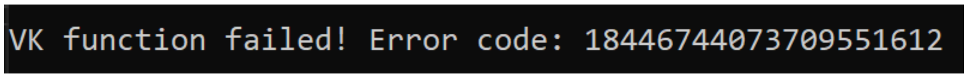
   i. Introduce deliberate errors or unexpected inputs (e.g., disconnecting nodes while audio is playing, loading corrupted audio files). Observe the app's response to these errors (e.g., error messages, and automatic recovery attempts).

   ii. Observe if recovery is attempted and how the application responds to the error.

   iii. Detail any resulting errors given from the application in the end.

c. **Results**: Early on in development, the following graph will cause a crash when too many splits in the screen are created and not enough space to create them. As a result, the application will crash and a message will display saying the function has failed with a specific error. This result shows that some failures will give proper documentation on why the code potentially failed. Recovery was not possible as the application crashed.





```
VK function failed! Error code: 18446744073709551612
```

**11. Limit Testing:**

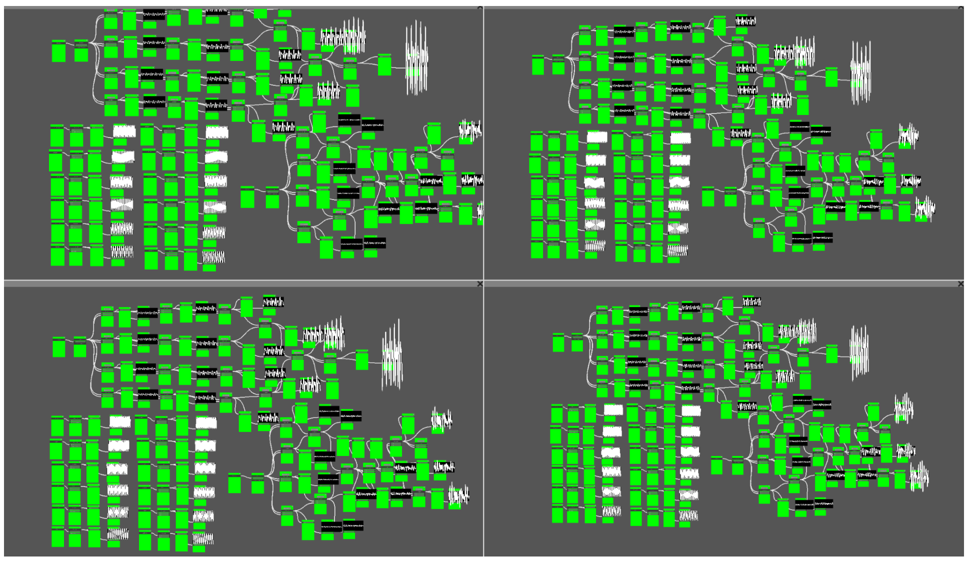
a. **Scenario**: The user wants to determine if the application can handle a large graph that utilizes everything.

b. **Steps**:

   i. Create a graph that utilizes sine waves that pipe into a channel and an oscilloscope node.

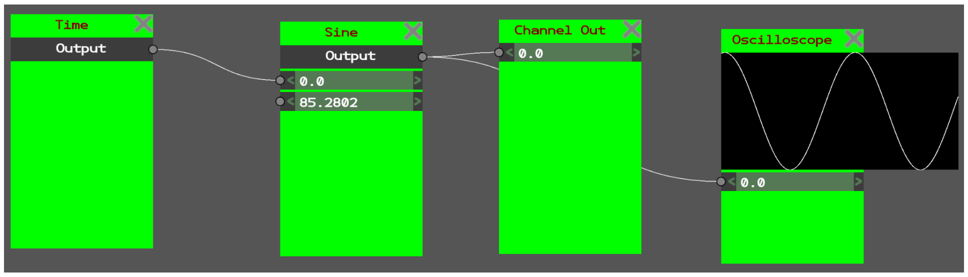
ii. Combine it with sampler nodes that pipe into a channel out and an oscilloscope node.
iii. Combine it with math nodes that pipe into a channel and oscilloscope node.
iv. Combine it with math nodes that pipe into more math nodes that pipe into a channel out and an oscilloscope node.
v. Run it all and ensure the program can run it.
vi. Optional: Create vertical and horizontal splits to ensure that it can run even with the splits.

c. **Results**: The following depicts a graph with 10+ time nodes, 20+ math nodes, 5+ sine waves, 20+ channel out nodes, 20+ oscilloscopes, and 5+ sampler nodes. Even with the addition of vertical and horizontal splits, the audio still plays fine. The only downside is that the application will get laggy with multiple split screens when moving a node around, but fine without them. There have also been noticeable cases when the graph may crash when being left idle.

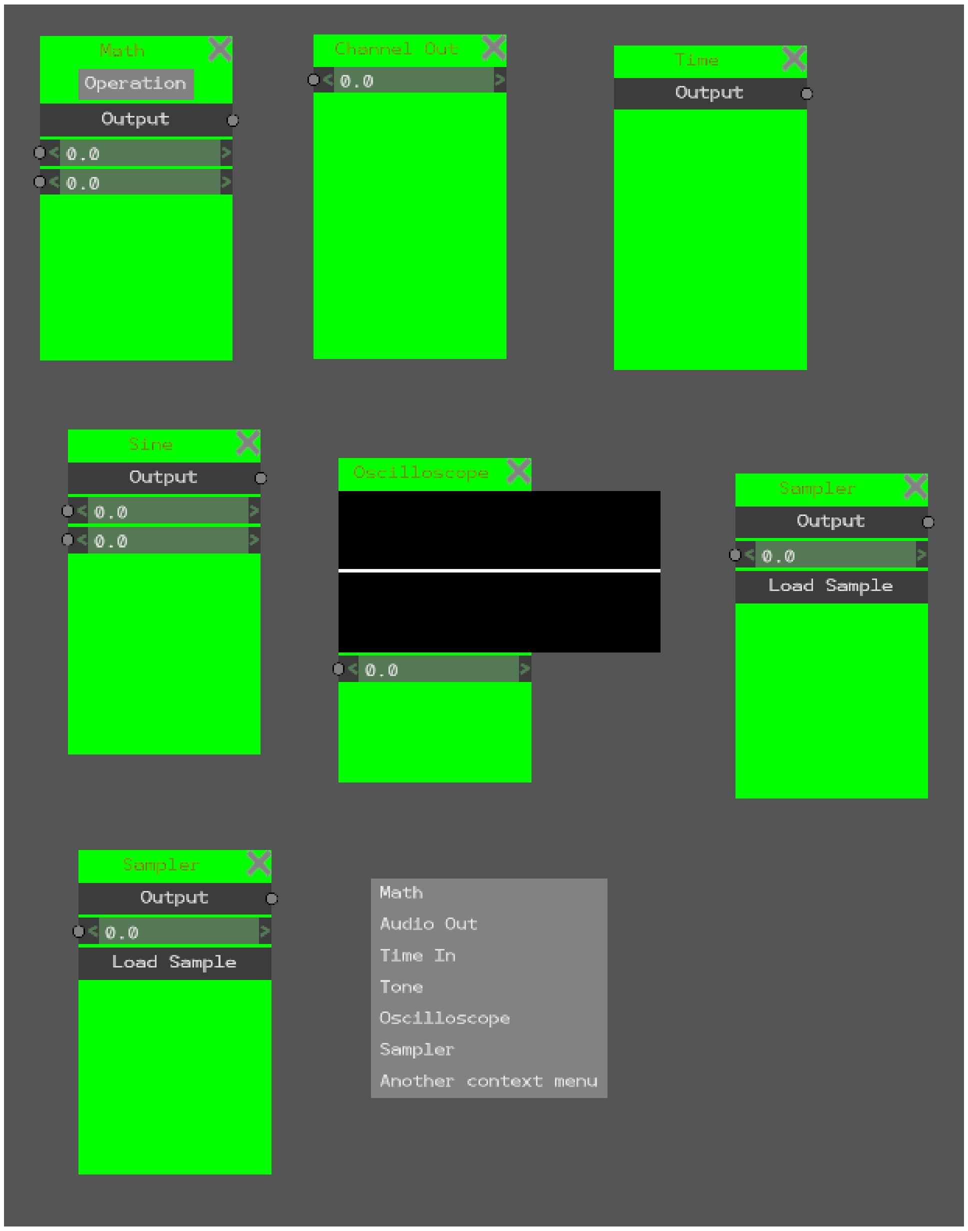


**12. Node sliders:**

a. **Scenario**: The user wants to determine if the node sliders are working correctly.
b. **Steps**:
   i. Generate a sine wave
   ii. Utilizing the sliding mechanic, generate audio for the node
   iii. Analyze the result to determine the outcome of the test.
c. **Results**: The graph below depicts a sine wave that is using the sliding mechanic to generate the lower frequency. As we can see, a wave is being produced that matches the frequency being slid. Overall, the sliding mechanic seems to be functional, though the rate at which it slides may be a bit slow, and applicable to all nodes, even the ones we don’t want it on.



13. **Right-click Menu:**
    a. **Scenario**: The user wants to determine that the correct nodes are being made with the right-click feature.
    b. **Steps**:
        i. Launch the application and right-click to select a node to create.
        ii. Generate all possible nodes to ensure that nothing goes wrong during the creation process.
    c. **Results**: The following image depicts all possible nodes that can be made from the right-click menu alongside that menu which shows a list of nodes the user can select from. This test shows that the right-click menu can work with no clear issues.



**14. Math Operations:**

a. **Scenario**: The user wants to ensure that the math nodes can alternate between the different types and work with the correct name.

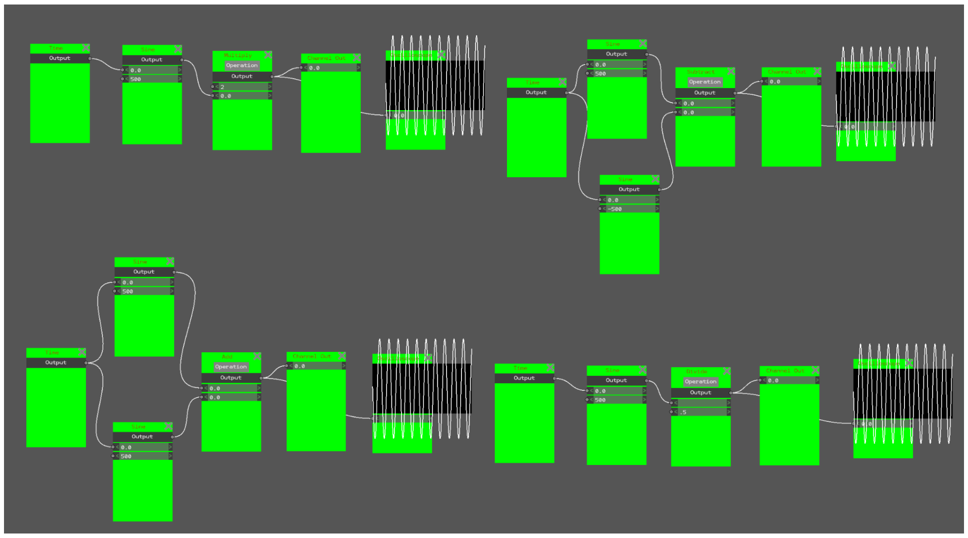
b. **Steps**:

   i. Generate one frequency of audio using the math node, selecting an initial mode.

   ii. Generate that same frequency of using the math node with different modes.

   iii. Compare the audio and waves to ensure they are working correctly.

c. **Results**: The following image depicts 4 different ways to generate 2 combined 500 Hz graphs. The bottom left is addition, the top left is multiplication, the top right is subtraction, and the bottom right is division. All 4 produce the same audio and graph. This test concludes that switching between math node types is working correctly.



**15. Expression Parsing:**

a. **Scenario**: The user wants to determine that the nodes can parse expressions correctly, which will result in the correct audio being played.

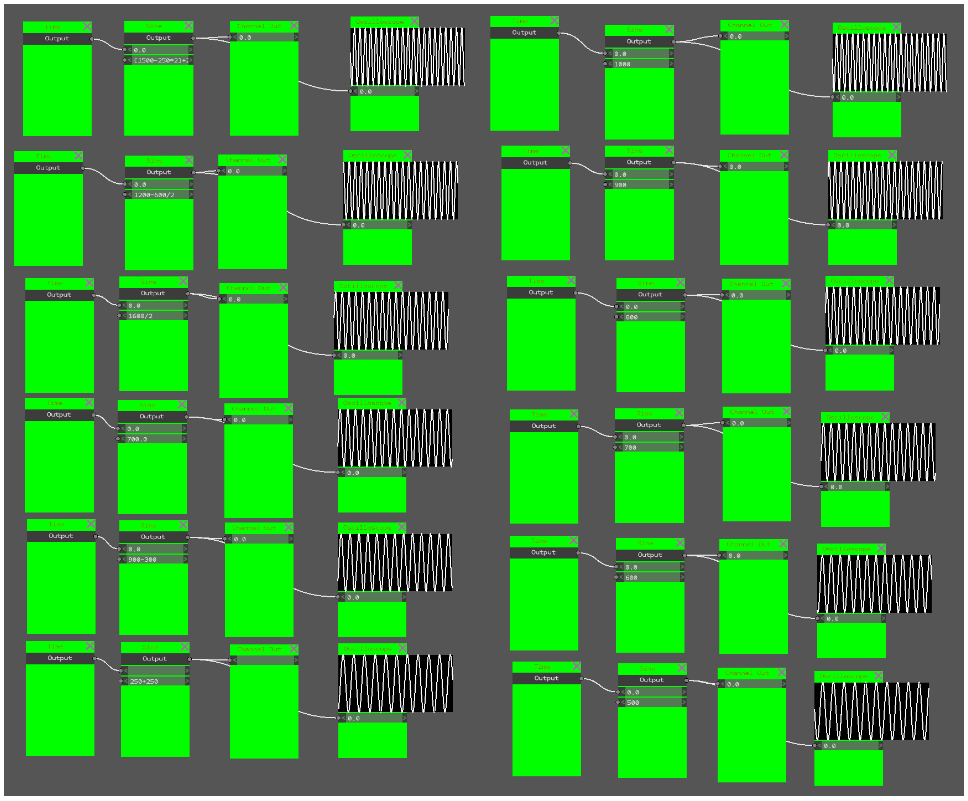
b. **Steps**:

   i. Create a normal audio graph with different sine waves

   ii. Create a second audio graph with sine waves whose values include expressions that result in the same value as the first.

   iii. Compare the audio, and determine the result

c. **Results**: The following graph below depicts 2 sets of sine waves, ranging from 500 to 1000 Hz. On the right, we just enter the value itself. On the left-hand side, we enter an expression that will result in the value equivalent on the right. When comparing the audio, the same audio and waves can be heard and seen. This concludes this test as a success with no issues for current parsing.



**16. Serialization:**

a. **Scenario**: The user wants to determine whether or not the save and load features of the application are working correctly.

b. **Steps**:

   i. Generate a graph and confirm the audio is playing correctly.

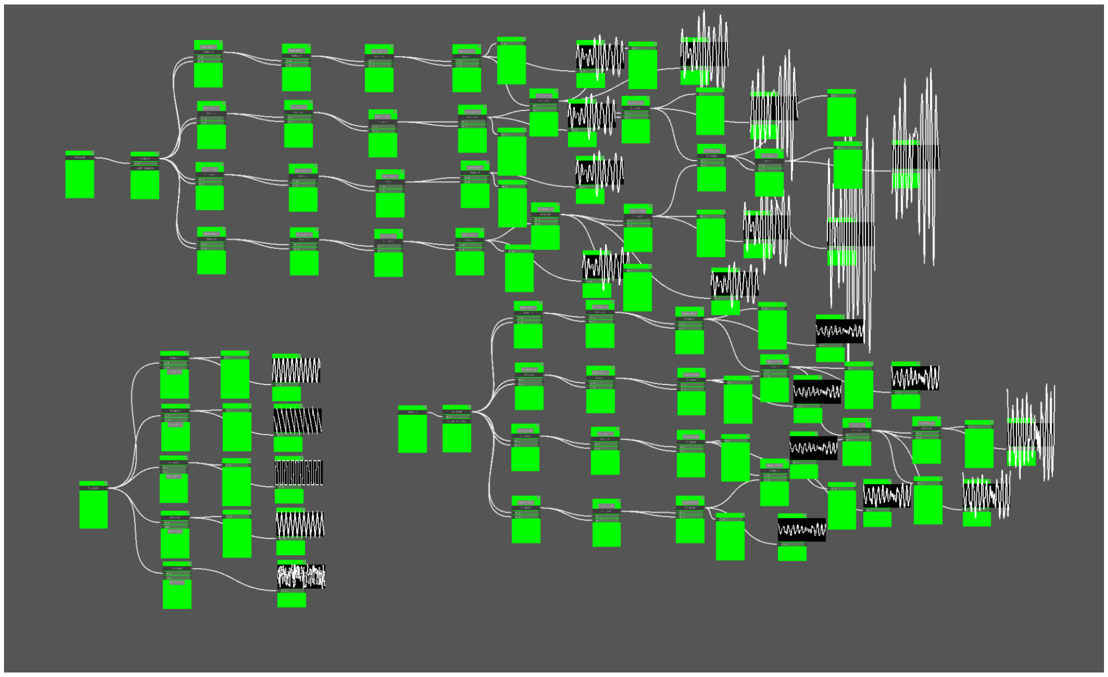
   ii. Save the graph

   iii. Delete everything or restart the application

   iv. Load the same graph that was saved in step 2

   v. Confirm the audio is working correctly.

c. **Results**: The following graph depicts an extremely large graph that is being saved and loaded into a file called example.dawdle. When the file is saved, there are no issues and the audio resumes to the moment the button was pressed to save. When loading, though there is a delay all nodes and connections are loaded in correctly, leading to the same sound being produced. That concludes this test as a success.



**17. Processing All Nodes in One Sequence:**

a. **Scenario**: The user wants to determine whether or not the application can handle a singular sequence of nodes that contain every node in the application.

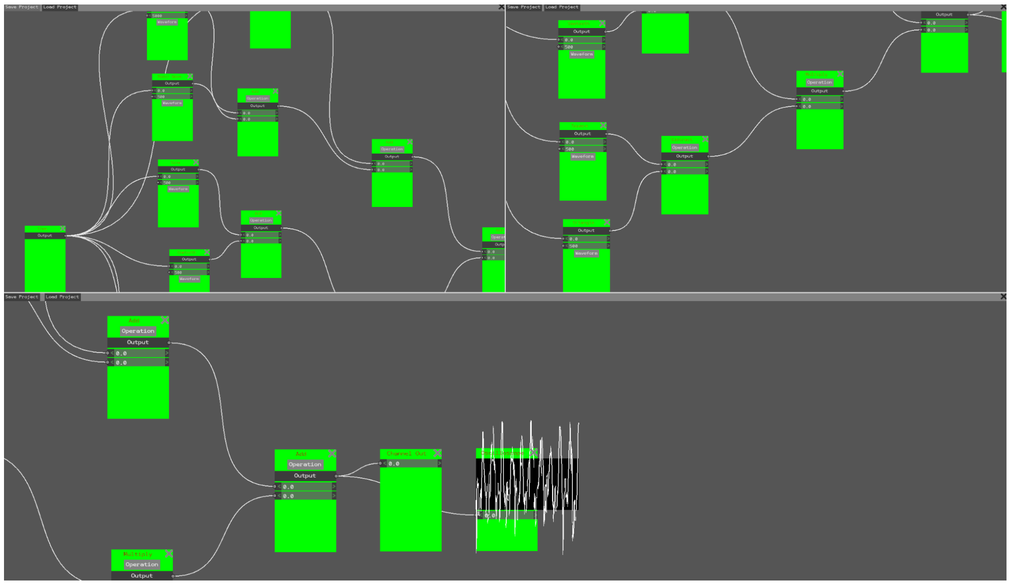
b. **Steps**:

   i. Create a graph that utilizes every node type.

   ii. Ensure that everything is connected in one sequence so that there is only one audio output node

   iii. Confirm that the application can handle this process correctly.

c. **Results**: The following graph depicts every type of node being used in the graph. There are no errors or crashes, and the audio is playing correctly. As such, the application can be concluded to handle all nodes in one sequence which indicates this test as a success.



**18. Latency Testing:**

a. **Scenario**: The user wants to determine if there is any delay when it comes to playing audio form sampler nodes.

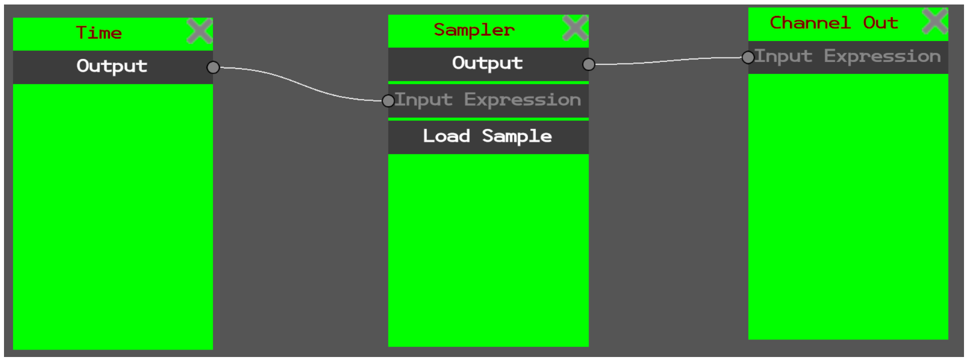
b. **Steps**:

   i. Generate a graph with the sampler node

   ii. Switch between audio files to determine if there is any delay with the play speed when switching.

   iii. Analyze the results to come to a conclusion

c. **Results**: When switching between audio files, there does appear to be a small gap where the previous audio file will produce an echo before switching over the selected audio file. Outside of this, the audio will play correctly. That concludes this test as a relative success.

d. 

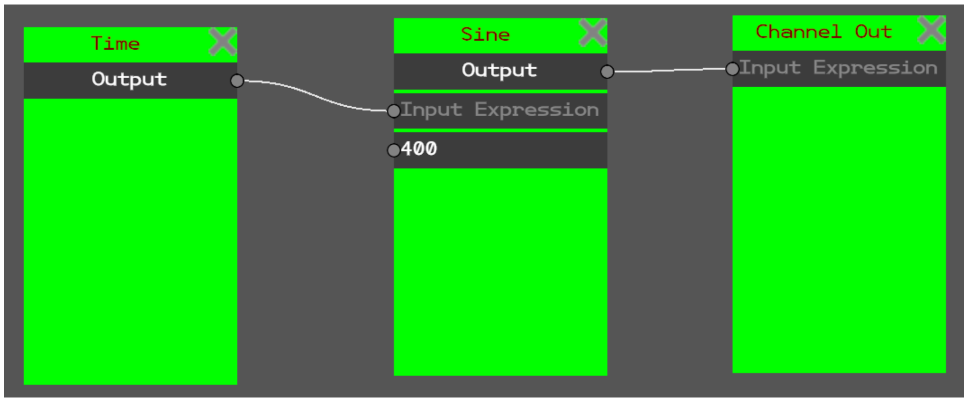
**19. Dynamic Graph Modification:**

a. Scenario: The user wants to determine if the audio will still play if the values and nodes are changed in real-time.

b. **Steps**:

   i. Generate a graph that is playing audio
   ii. Delete a node or change the values to ensure the audio is still playing
   iii. Move the nodes around while audio is playing to ensure no loss of audio quality
   iv. Analyze the results to come to a conclusion

c. **Results**: When adding or changing nodes on a graph, the audio will still play if it is connected to a channel node. This means that any dynamic changes made to the values or shifting the nodes around will cause the audio to shift, though moving the nodes themselves will not change any audio. That concludes this test as a success.

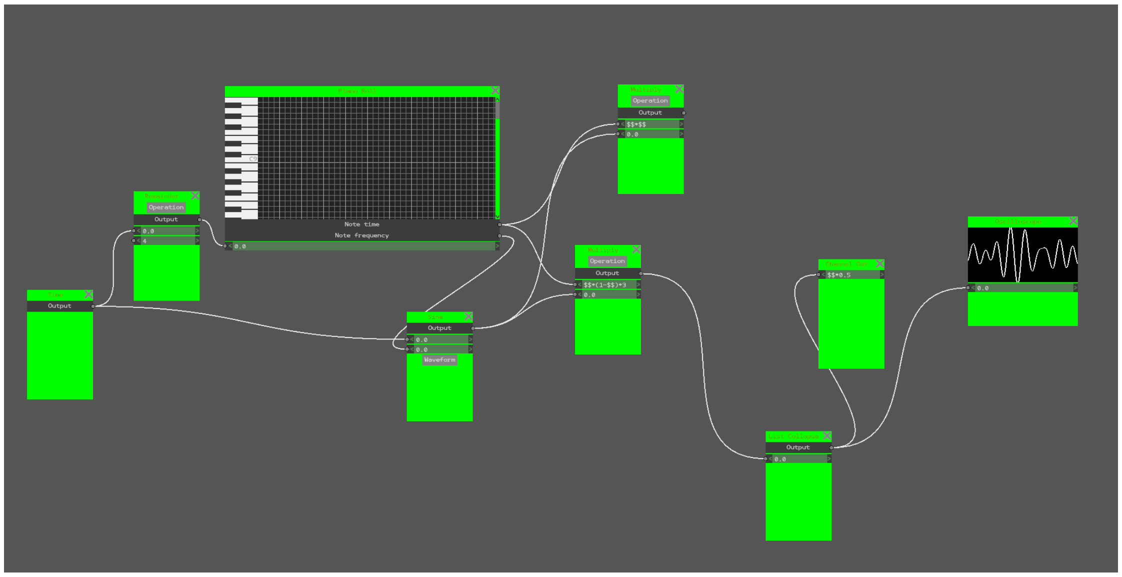


**20. Stop Button:**

a. **Scenario**: The user wants to determine if they are able to stop the audio being played without much loss of frames when clicking the button to the actual audio stopping.

b. **Steps**:
   i. Produce a graph
   ii. Pause the audio being played by hitting the stop button
   iii. Analyze whether or not there is any loss of audio and come to a conclusion.

c. **Results**: The following graphs depicts an audio being played through a multitude of different nodes. When pressing the stop button, there is immediate halt to the audio being played. As such, the test can be concluded to be a success and that the stop button works as intended.



**21. Play/Stop Button:**

a. **Scenario**: The user wants to determine if they are able to stop the audio being produced and play it again without any loss of audio quality or time.

b. **Steps**:

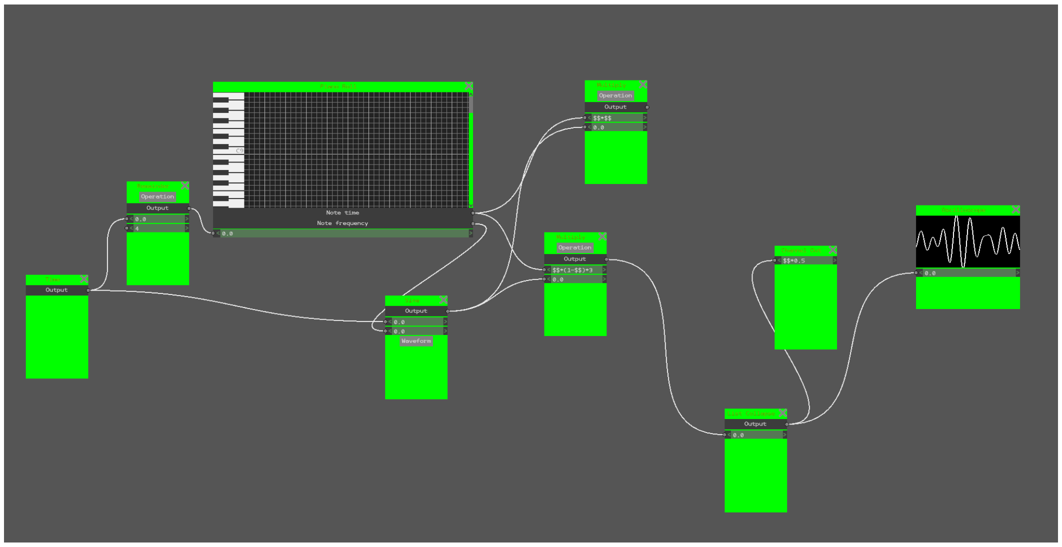
   i. Produce a graph

   ii. Pause the audio being played by hitting the stop button

   iii. Play the audio that has been paused

   iv. Repeat steps 2 and 3 multiple times to come to a conclusion as to whether or not there is any loss of audio or quality.

c. **Results**: The following graphs depicts an audio being played through a multitude of different nodes. When pressing the play and stop button, the audio stops and resumes correctly. This can be seen through the oscilloscope, stopping and resuming with little to no loss of audio. As such, this test can be concluded as a success and works as intended.



**22. Piano Roll #1:**

a. **Scenario:** The user wants to determine if the audio will play correctly when using the piano roll.

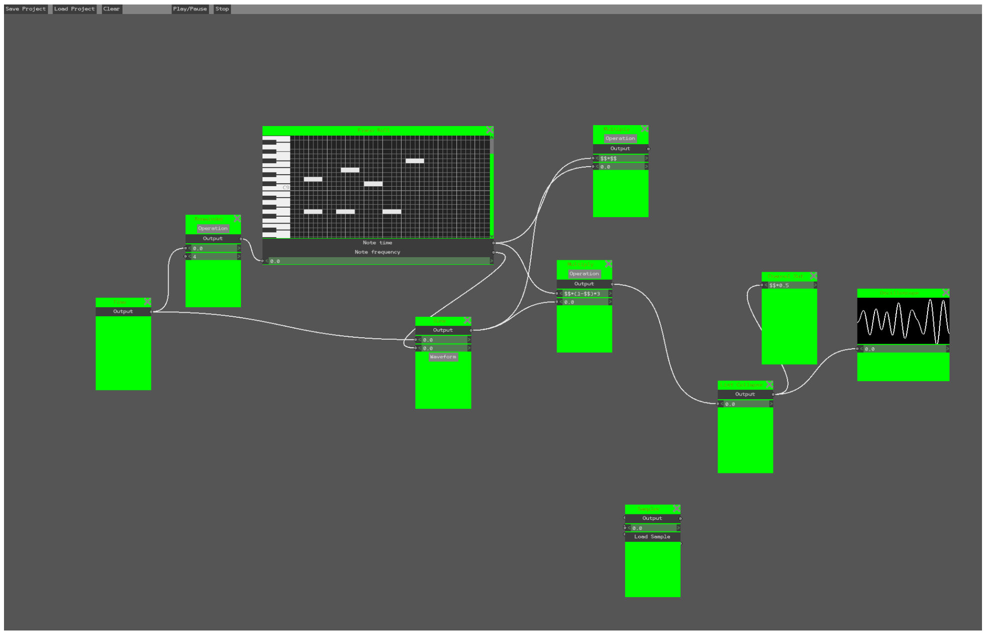
b. **Steps:**

   i. Generate a graph

   ii. Using the piano roll, ensure that the audio produced is working properly.

   iii. Analyze the audio being produced by the piano roll to ensure that it's working properly.

c. **Results:** The following graph depicts a piano roll graph. It is being used to produce audio that has been verified as correct. Additionally, the piano roll has been tested to work with other nodes that morph the output, being able to speed it up or slow it down depending on what the user wants. As such, this test can be concluded as a success and works as intended.



**23. Piano Roll #2:**

a. **Scenario:** The user wants to determine if the piano roll ui will allow for the creation of longer piano rolls.

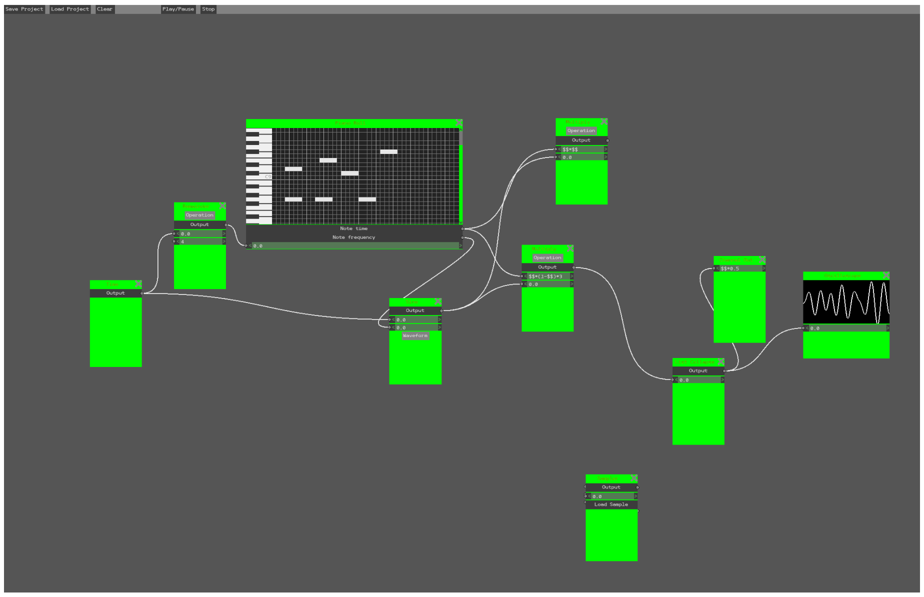
**b. Steps:**

i. Generate a graph using the piano roll

ii. Test whether or not the piano roll UI will work correctly

1. Test how the piano roll scales and if there is a limit to how far you can scroll right
2. Test if it will save sections that are separated by a large distance
3. Test if the piano roll condenses properly and doesn't play audio with the correct sequence.

c. **Results:** The following graph depicts a piano roll graph. The piano roll can play audio that is very separated. On that note, it seems that we can scroll right for as long as we need with no issues in sight. The piano roll condenses properly with the list collapse node and will not play audio without it, indicating it is being processed correctly. As such, that concludes this test as a success, with the piano roll UI working properly.

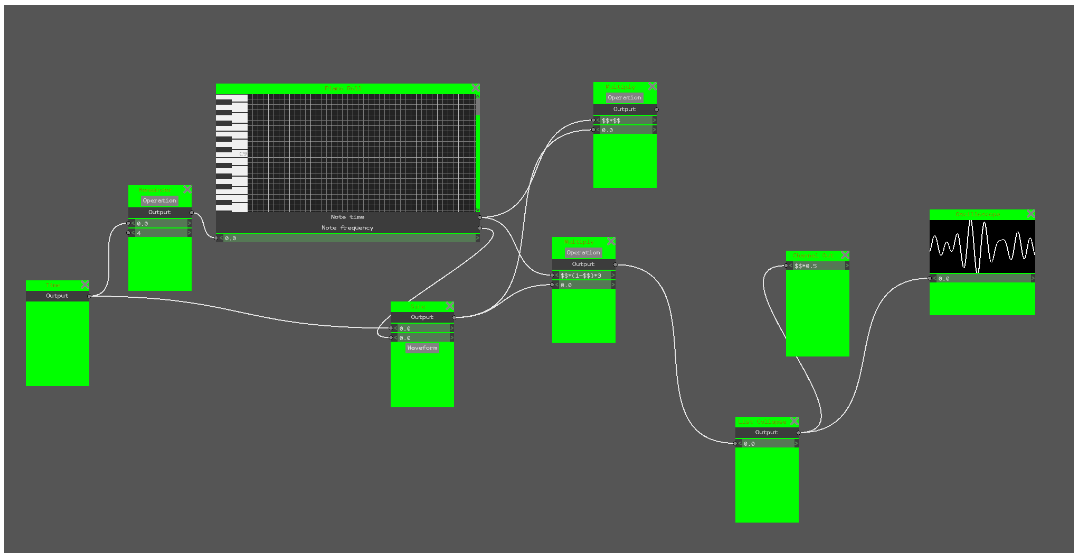


**24. List Collapse:**

a. **Scenario**: The user wants to determine if the list collapses if working correctly on the piano roll.

b. **Steps**:
   i. Generate a graph that utilizes the piano roll
   ii. Link the piano roll to a list collapse
   iii. Mess around by linking the list collapse node to an audio output node and a piano roll node directly to an audio output node.

c. **Results**: The following graph depicts a piano roll and a list collapse node. When linking the list collapse node to an audio output node, the audio will play correctly. However, when the piano roll is linked directly to the audio output node or to a sequence without a list collapse, the audio will not play. As such, this means the list collapse node works as required in conjunction with the piano roll node, being required to play audio.



**25. Clear Button:**

a. **Scenario**: The user wants to determine whether or not the clear button successfully clears the graph and if there are any issues with it.

b. **Steps**:

   i. Generate a graph

   ii. Hit the clear button

   iii. Determine whether or not the graph successfully clear

   iv. Mess around with the clear button to see if anything breaks

c. **Results**: The following graph depicts audio being played through a multitude of different nodes. When pressing the clear button, there is an immediate clear of the graph. Additionally, spamming the clear button on an empty graph does not break anything. As such, this test can be concluded as a success and works as intended.