Calvin University

Department of Computer Science

Final Senior Design Report

Nathan Herder

05/17/2020

Advisor: Joel Adams

Project vision and overview:

The goal of my senior design project has been to create a library that joins together the functionality of two other c++ libraries that have been created by Calvin Students over the years. Those two libraries being TSGL (Thread Safe Graphics Library) and TSAL (Thread Safe Audio Library). The purpose of this wasn't only to make a higher level general purpose library that combines the functionality of both libraries, but also to create and explore some pedagogical tools to potentially help computer science students to understand various sorting algorithms through visualizations and audializations. My Audio Visual library serves to give others the ability to create programs that utilize both the visual elements of TSGL combined with audio from TSAL. The other main goal that the library fulfills is allowing audialization, visualizations, or a combination of the two to be played.

Audializations refer to sound representations of sorting algorithms using different frequencies to convey the patterns that appear audibly in sorting algorithms when sorting numbers. I wanted to try and discover what patterns appear and whether or not different algorithms can be distinguished by ear rather than only visibly. This gave me some leeway to experiment with how these audializations are represented.

This library can hopefully be used as a building block for future development for summer research or future design projects. I also hope that it can be used as a teaching tool to engage students and learn about sorting algorithms, and potentially be extended

in the future to combine the new versions and features of TSGL and TSAL to be used together.

Background, including research review:

Beginning the fall semester I had no prior knowledge of TSAL and some slight exposure to TSGL in previous classes, mainly CS 112. This required that I familiarize myself more with both libraries, understand their APIs, dependencies, and how they work so that I could begin to think about design decisions when it came time to combine them into my Audio Visual library.

The first bit of research required that I better understand how to build and obtain packages of these libraries to be able to use on my system, which is running Ubuntu 18.04 LTS. Chris Wieringa had packaged Ian Adams most recent summer work on TSGL for the Calvin lab machines, which run the same version of Ubuntu, and was able to provide some help and guidance on how to build a debian package of TSGL that I was able to use. TSAL uses a the GNU Autotools build system that I was not familiar with which required a little bit of research to get more comfortable with. Autotools, Autoconf, and Automake seemed daunting at first because of the amount of files that are automatically generated for you, but a little research and building TSAL's instructions made building TSAL rather trivial.

TSGL had some very good examples and documentation to get me familiar with the API and examples on how to use a canvas, start, and stop drawing animations. Ian

implemented two sorting algorithms this past summer, a parallel merge sort and a shaker sort to demonstrate TSGL's capabilities, that served as a very good starting point for combining TSAL's functionality later on.

TSAL took some getting used to, to fully understand the purpose of a mixer and synthesizer objects. I do not have much musical background, fortunately examining TSAL's examples and having Mark, the creator of TSAL, at my disposal was needed to be able to thoughtfully work on combining the two libraries.

System design and Implementation:

TSAL was created and designed by Joel Adams and Mark Wissink to be a companion library to TSGL. These similar design choices made in advance such as both libraries being designed to be thread safe, header only, and cross platform compatible helped in guiding many of my decisions. The goal for my audial visual library was to also be a C++ header only library by including both of the TSGL and TSAL headers in the base class. Header only libraries have some drawbacks and benefits depending on the scenario. In the case of Calvin's libraries they are easy to integrate and simplify the build process for users so that they don't have to specify the library to the linker. All the code is pulled in by the preprocessor when the library is included. They still required that dependencies be downloaded manually by the user, unless a configuration script is made, but c++ does not have a native package manager anyways so that would be a problem to deal with even if it was a static or shared library. On the

other hand being header only makes for larger object files and can increase compilation time as a project grows if your library begins being included multiple times it can unnecessarily be compiled multiple times. In the case of this library that's not really an issue since anyone that uses it should only be including it once.

When beginning to design the Audio Visual library professor Adams and I wanted to make sure to allow for any programs to be able to make use of the TSGL canvas and TSAL synthesizers. We also wanted to create a class specifically for sorting algorithms to inherit functionality and characteristics specific to them. These decisions led to the layout of the class structure that was implemented. The base class, AudialVisualization, handles all command line parameters, provides data and functions that anyone would need to be able to use the functionality of both classes such as: a TSGL Canvas pointer, TSAL Mixer pointer, methods to create a canvas or mixer, getters and setters for Canvas parameters, and additional information.

The handling of command line parameters is handled through the use of cxxopts, a lightweight header only command line parsing library. Cxxopts provides a very nice framework to create positional, named, or flag style command line arguments. It also provides a default help option to display all possible arguments with a description for users. There are seven parameters I have made available which can be used to specify whether or not you want a canvas, a mixer for playing sound, canvas height, canvas width, the number of data elements to be sorted, the number of threads to be used, and flags to specify which sorting algorithm to run.

The SortingAudialVisualization class inherits from AudialVisualization and is designed to be inherited from in order to implement sorting algorithms or to play the existing sorting algorithms which have been implemented. The inheritance and polymorphism allowed similar features of all sorting algorithms to be encapsulated in attempts to reduce unnecessarily repeating code. For example, the SortingAudialVisualization class overrides the method to create a canvas so that it can ensure the canvas height is always half of the width. There are also some common getter and setter methods used to compute the width of the data element rectangles onto the canvas if you prefer to sort less data elements than the width of the canvas.

Each implementation of the sorting algorithms inherit form SortingAudialVisualization and are encapsulated into their own respective class containing the algorithm and data specific to them. They have a simple run function to determine whether or not a canvas and mixer need to be created based on the command line arguments passed and a function that sorts the data and plays the audialization and visualizations.

One area of the design that had to be altered during the project was the implementation of the sorting algorithm classes. I first created three methods in each sorting algorithm class that accepted a pass by reference canvas and mixer: one that is passed only a canvas object, another that is passed only a mixer object, and a third that is passed both. After examination it became clear that this required a lot of repeated code and could be implemented using pointers that could be passed as null if an object

doesn’t exist. This required only implement one method as opposed to three. This helped to simplify the classes but then required that the pointers be checked carefully when running the algorithms to avoid any null pointer references. This was a valuable adaptation to the implementation that was changed as I encountered the problem.

An additional design implementation that I changed was the object required to be created by the user. When first designed I was creating the objects for the sorting that I wanted to run in my programs main function and calling their run function. This was a poor design decision because it required recompiling every time the user wanted to run a different sorting algorithm. To fix this poor decision I refactored the code so that to run any of the sorting algorithms the user must only instantiate a SortingAudialVisualization object. Then each sorting algorithm object is created and played based on the -s flags and their arguments given at runtime.

Some additional tweaking and experimentation was required when running the audializations with TSAL. I began by starting with the method used in TSAL’s examples to compute the MIDI notes to play in the audializations. This method started at a C3 MIDI note and added a constant multiplied by the current data value being sorted divided by the max data element. This method of computing the MIDI note to play didn’t seem to make the most sense and was inconsistent when running different sorting algorithms. Some of the sorting algorithms would begin mapping sounds into a very low or high range that was difficult to hear. I worked with Mark to create a new helper method in TSAL that can be given a range of MIDI notes, a range from zero to the

height of the canvas, and the current data value. These parameters are then used to map the current data value to an appropriate note in the range of notes specified so that the audializations are more consistent no matter the number of items that are being sorted and stay in a more pleasant range for listening to.

The decision to be able to change the number of data elements being sorted was made in order to be able to speed up the demos for my final presentation and for others that may use my library in the future. The sorting algorithms originally showed the data being sorted by using TSGL's drawLine() function representing each data element being sorted as only one pixel wide. Each sort only sorts as many numbers as the canvas is wide. For slower algorithms such as bubble, insertion, and selection it would take a very long time to complete. You could use the -w flag to run the algorithms with a smaller canvas width which would reduce the number of items being sorted to speed up the visualization, but then the canvas would be very small and not optimal for viewing. This dilemma is why I chose to add the -d flag to be able to have a larger canvas to see and allow for sorting fewer items so that it doesn't take too much time to watch the algorithm complete. Rather than using the tsgl::drawLine() method it now uses tsgl::drawRectangle() method to draw the data with a larger than one pixel width.

Results and discussion:

The final result of my project is a functioning library that combines the features of TSGL completed at the end of Ian's summer research and TSAL's synthesizers and

thread synthesizers. There are currently five implemented sorting algorithms including bubble, insertion, selection, shaker, and parallel merge sorts. There is also a linear merge sort that demonstrates the library's ability to support non sorting algorithm use cases.

Before college classes were moved online Mark Wissink, professor Adams, and I were able to conduct an experiment to try and see whether or not learning sorting algorithms via audializations had any apparent effect on the learning outcomes of students. We had a total of thirty-seven CS 112 students who participated and who we split into control and treatment groups. CS 112 students were chosen as test subjects because they have not yet formally covered sorting algorithms in CS 212 but have still been exposed to Big O notation and time complexity in 112. The control group remained in the gold lab and the treatment group was placed into the maroon lab.

The experiment lasted for fifty minutes and began by having both the control and treatment group take the same Google Forms pre-quiz dealing with bubble, insertion, quick, and merge sorting algorithms. The pre-quiz asked various questions about the runtime, Big O time complexity, and to identify sorting algorithms from pseudocode. The control group spent ten minutes reading online articles about each sorting algorithm, its time complexity and how it is implemented. The treatment group read the same articles as the control group for five minutes and then interacted with each sorting algorithm audibly in TSAL. Finally, both groups ended the experiment by taking a post-quiz of a

very similar structure to the pre-quiz. The goal of the experiment being to see whether or not interacting with sorting algorithms audibly has any effect on learning outcome.

A grading scale of one point for each question was used. Any additional selected check marks for any of the questions beyond that of the answer resulted in the subtraction of a point. The null hypothesis formed was that there is no significant learning outcome difference between the control and treatment groups. The alternative hypothesis being that there is a significant learning outcome difference between the control and treatment groups. The degrees of freedom was one less than the number of participants, so 36. The average post score from the control group was 11.88 with a variance of 8.57, while the average post score from the experiment group was 11.33 with a variance of 9.743. This resulted in a t-statistic of 0.18 and a t-critical of 2.03 for a two-tailed test. The t-statistic is less than the cutoff t-critical value and we received a p-value of 0.858, leading me to fail to reject the null hypothesis and state that there was no real significant difference in learning ourcomes between the two groups.

We identified some parts of the experiment that could have gone more smoothly. Some of the CS 112 students were not as familiar with the command line and our instructions could have been more clear causing less time to be spent interacting with the audializations. There were also some unnecessary ALSA audio driver messages produced by TSAL that raised questions by those in the treatment group. Those TSAL message should be silenced in any future runs of the experiment. There was also some confusion about which audio jack to use on the lab computers, some machines required

using the green audio output on the back of the machine rather than headphone jack on the top. Finally, the aforementioned issues caused slight delays in sticking to the experiment schedule and getting the treatment group participants up and running with the experiment and remaining in sync with one another.

Overall it was a good first experiment that I think would be interesting to run on future CS 112 students with some of the kinks worked out. It would also potentially be neat to try with visually impaired people who can't see a traditional sort. The experiment group expressed that the audializations were a more engaging means of learning over the control group. I personally can tell apart the various sorting algorithms based on sound alone. There are very distinct audible patterns that arise with each algorithm.

Conclusions:

This was a fun and challenging project to project to work on. It was neat to see the progress and work of past and current Calvin students combined together for the first time into the audial visualization library that I designed and created. I would say that this was a successful project that was able to meet a majority of the intended goals along the way. There is now an object oriented header only command line library that combines the functionality of both TSGL and TSAL. There are also a handful of sorting algorithms allowing visualizations and audializations to play. There is no shortage of future work for Calvin students to step in, learn, experiment, and make improvements

along the way. I'm looking forward to seeing future growth and development in this library in the future

Future work:

In the future there are many areas for the improvement of the library. One feature that I would like to see added is the ability to run multiple sorting algorithms simultaneously using multiple canvases and mixers. Near the end of the semester I began working on this but did not quite complete it. I was anticipating being able to pass multiple -s to the command line to create multiple omp threads for each sorting algorithm being run. If the user wants to run multiple sorts at once and one is a parallel sorting algorithm, the thread created for that algorithm can then split to create the number specified via the -t command line argument. This added functionality would make a better learning tool to teach sorting algorithms and show visualizations.

In hindsight another thing that could be changed, is to have one function in the SortingAudialVisualization class that can generate the array of numbers that need to be sorted and passed to the sorting algorithms as an argument. This would further reduce repeated code in each of the sorting algorithm classes and place it into a single location available to all.

I would be great if there could be more analysis on the learning outcomes that audializations provide to students. I would like to see another experiment conducted on

CS 112 students with more of the experiment issues that we found worked out beforehand. I would also like to see and experiment designed with visually impaired people to see how they find this method of learning algorithms and their ability to distinguish them.

A final additional thing that would help make this library more usable on all platforms would be to create a consistent build environment across TSGL, TSAL and my library. GNU Autotools or CMake may be a good solution to download the required library dependencies for you depending on the users platform. It would also be nice to set up a continuous integration pipeline so that each library is built for each platform when a push into master is made. As the libraries grow and increasingly depend on one another, this would make the experience more enjoyable for more users on more platforms. I would like to see Ian's new 3D additions to TSGL able to be used in my library.

Acknowledgments:

Throughout the course of this senior design project and my time at Calvin I have received a lot of help, advice, and knowledge from my professors, peers, family, and others. I appreciate and am thankful for all the support that I have received. First, I want to thank Professor Adams for being my advisor throughout this project. I've enjoyed working with you and am thankful for the guidance that you have provided. I also wanted to thank Chris Wieringa for his help early on in my project as well as Ian Adams

and Mark Wissink for being willing to help answer questions and resolve problems related to TSGL and TSGL along the way. Finally, thank you to the Calvin computer science department and all the professors who have helped to shape me and my education throughout my time at Calvin.

References:

TSGL: https://github.com/Calvin-CS/TSGL

TSAL: https://github.com/Calvin-CS/TSGL

CXXOPTS: https://github.com/jarro2783/cxxopts

Appendixes:

```
/**
 * AudialVisualization.h declares a class that can be inherited from
 * to wrap TSGL and TSAL programs.
 *
 * Who: Nate Herder
 * When: 11/06/2019
 * Where: Calvin University
 *
 */
#pragma once

#include <tsgl.h>
#include <tsal.hpp>
#include <cxxopts.hpp>
#include <omp.h>
#include <iostream>
#include <memory>
#include <string>

using namespace tsgl;
using namespace tsal;

namespace avlib {

class AudialVisualization {
protected:
  int num_threads;
  bool show_visualizaiton;
  bool play_audialization;
  int canvas_height;
  int canvas_width;
  int num_algorithms_to_run;
  std::vector<std::string> sorting_algorithms;
  Mixer *mixer = nullptr;
  Canvas *canvas = nullptr;
  std::vector<ThreadSynth> voices;
  int data_amount;

public:
  AudialVisualization(int argc, char **argv);
  virtual Canvas *createCanvas(std::string canvas_name);
  virtual Mixer *createMixer();
  void setVisualization(const bool b);
  void setAudialization(const bool b);
```

```
  void setNumThreads(const int n);
  void setCanvasHeight(const int h);
  void setCanvasWidth(const int w);
  void setSortingAlgorithm(const std::vector<std::string>& a);
  void setDataAmount(const int a);
  bool showVisualization() const;
  bool playAudialization() const;
  int getCanvasHeight() const;
  int getCanvasWidth() const;
  int getDataAmount() const;
  int getNumThreads() const;
  std::vector<std::string> getSortingAlgorithms() const;
  ~AudialVisualization();
};

}


/**
 * AudialVisualization.cpp defines the functionality that is common
 * to wanting running TSAL and TSGL applictions together.
 *
 * Who: Nate Herder
 * When: 11/06/2019
 * Where: Calvin University
 *
 */

#include "AudialVisualization.h"

namespace avlib {

cxxopts::ParseResult parse(int argc, char** argv) {
  try {
        cxxopts::Options options("AudialVisualization", "- command line options");

        options.positional_help("[optional args]").show_positional_help();

        options.allow_unrecognised_options().add_options()("help", "Print help")(
        "a, audial", "play audialization using TSAL",
cxxopts::value<bool>()->default_value("false"))(
        "v, visual", "show visualization using TSGL",
cxxopts::value<bool>()->default_value("false"))(
        "h, canvas-height", "TSGL canvas height",
cxxopts::value<int>()->default_value("1024"))(
        "w, canvas-width", "TSGL canvas width",
cxxopts::value<int>()->default_value("1024"))(
```

```
        "d, data-amount" , "Number of data elements to sort. Only used when inheriting
from SortingAudialViaulization. data-amount must be less than canvas-width.",
cxxopts::value<int>()->default_value("1024"))(
        "t, threads", "number of threads to use",
cxxopts::value<int>()->default_value("1"))(
        "s, sorting-algorithm", "decide which sorting algorithm to run",
cxxopts::value<std::vector<std::string>>());

        auto results = options.parse(argc, argv);

        if (results.count("help")) {
        std::cout << options.help({"", "Group"}) << std::endl;
        exit(0);
        }

        return results;

  } catch (const cxxopts::OptionSpecException& e) {
        std::cout << "error parsing options: " << e.what() << std::endl;
        exit(1);
  }
}

AudialVisualization::AudialVisualization(int argc, char** argv) {
  auto result = parse(argc, argv);

  if (result["canvas-height"].as<int>() > 0) {
        setCanvasHeight(result["canvas-height"].as<int>());
  }
  if (result["canvas-width"].as<int>() > 0) {
        if (result["canvas-width"].as<int>() < 100) {
        setCanvasWidth(100);
        }
        setCanvasWidth(result["canvas-width"].as<int>());
  }
  if (result["data-amount"].as<int>() > getCanvasWidth()) {
        std::cout << "data-amount can't be larger than canvas-width" << std::endl;
        std::exit(1);
  } else {
        setDataAmount(result["data-amount"].as<int>());
  }

  if( result.count("sorting-algorithm") > 0 ) {
        setSortingAlgorithm(result["sorting-algorithm"].as<std::vector<std::string>>());
  }

  if (result["threads"].as<int>() > omp_get_num_procs()) {
        setNumThreads(omp_get_num_procs());
```

```
  } else {
        setNumThreads(result["threads"].as<int>());
  }

  setAudialization(result["audial"].as<bool>());
  setVisualization(result["visual"].as<bool>());

}

Canvas* AudialVisualization::createCanvas(std::string canvas_name) {
  if (showVisualization()) {
        canvas = new Canvas(0, 0, getCanvasWidth(), getCanvasHeight(), canvas_name);
  }
  return canvas;
}

Mixer* AudialVisualization::createMixer() {
  if (playAudialization()) {
        mixer = new Mixer();
  }
  return mixer;
}

void AudialVisualization::setVisualization(const bool b) {
  show_visualizaiton = b;
}

void AudialVisualization::setAudialization(const bool b) {
  play_audialization = b;
}

void AudialVisualization::setNumThreads(const int n) {
  num_threads = n;
}

void AudialVisualization::setCanvasHeight(const int h) {
  canvas_height = h;
}

void AudialVisualization::setCanvasWidth(const int w) {
  canvas_width = w;
}

void AudialVisualization::setSortingAlgorithm(const std::vector<std::string>& a) {
  sorting_algorithms = a;
}

void AudialVisualization::setDataAmount(const int a) {
```

```
  data_amount = a;
}

bool AudialVisualization::showVisualization() const {
  return show_visualizaiton;
}

bool AudialVisualization::playAudialization() const {
  return play_audialization;
}

int AudialVisualization::getNumThreads() const {
  return num_threads;
}

int AudialVisualization::getCanvasHeight() const {
  return canvas_height;
}

int AudialVisualization::getCanvasWidth() const {
  return canvas_width;
}

int AudialVisualization::getDataAmount() const {
  return data_amount;
}

std::vector<std::string> AudialVisualization::getSortingAlgorithms() const {
  return sorting_algorithms;
}

AudialVisualization::~AudialVisualization() {
  delete canvas;
  delete mixer;
}


}


/**
 * SortingAudialVisualization.h declares the class that all audial
 * visualizations will inherit from containing all methods, command line
 * preferences, and other data that is common across all sorting algorithms.
 *
 * Who: Nate Herder
 * When: 11/06/2019
 * Where: Calvin University
```

```
 *
 */

#pragma once

#include <string>
#include "AudialVisualization.h"
#include "tsgl.h"

using namespace tsgl;
using namespace tsal;

namespace avlib {

class SortingAudialVisualization : public AudialVisualization {
private:
  bool even_data_chunks;
  bool main_thread;
  int block_size;
  int number_normal_block_size;

public:
  SortingAudialVisualization(int argc, char** argv, bool value = true);
  Canvas* createCanvas(std::string canvas_name);
  Mixer* createMixer();
  void setEvenDataChunks(const int v);
  bool getEvenDataChunks() const;
  void setBlockSize(const int bs);
  int getBlockSize() const;
  void setNumberNormalBlockSize(const int n);
  int getNumberNormalBlockSize() const;
  bool getMainThread();
  ~SortingAudialVisualization();
};

}


/**
 * SortingAudialVisualizations.cpp holds defines the common methods and data
 * across all sorting algorithms.
 *
 * Who: Nate Herder
 * When: 11/06/2019
 * Where: Calvin University
 *
 */
```

```
#include "SortingAudialVisualization.h"
#include "BubbleSorter.h"
#include "InsertionSorter.h"
#include "MergeSorter.h"
#include "SelectionSorter.h"
#include "ShakerSorter.h"
#include <cmath>

namespace avlib {

SortingAudialVisualization::SortingAudialVisualization(int argc, char** argv, bool value) :
AudialVisualization(argc, argv), main_thread{value} {
 if( (getCanvasWidth() % getDataAmount()) == 0 ) {
        setEvenDataChunks(true);
        setBlockSize( (getCanvasWidth()/getDataAmount()) );
        setNumberNormalBlockSize( getDataAmount() );
 } else {
        setEvenDataChunks(false);
        setBlockSize( floor( (getCanvasWidth()/getDataAmount()) ) );
        setNumberNormalBlockSize( (getDataAmount() - (getCanvasWidth() %
getDataAmount())) );
 }

 std::vector<std::string> sort_run_vector = getSortingAlgorithms();

 if( getMainThread() == true ) {
        #pragma omp parallel num_threads( sort_run_vector.size() )
        {
        int tid = omp_get_thread_num();

        if(sort_run_vector.at(tid) == "bubble") {
        BubbleSorter b(argc, argv);
        b.run();
        } else if(sort_run_vector.at(tid) == "insertion") {
        InsertionSorter i(argc, argv);
        i.run();
        } else if(sort_run_vector.at(tid) == "merge") {
        MergeSorter m(argc, argv);
        m.run();
        } else if(sort_run_vector.at(tid) == "selection") {
        SelectionSorter s(argc, argv);
        s.run();
        } else if(sort_run_vector.at(tid) == "shaker") {
        ShakerSorter sh(argc, argv);
        sh.run();
        }
        }
 }
```

```
}

Canvas* SortingAudialVisualization::createCanvas(std::string canvas_name) {
  setCanvasWidth(getCanvasWidth() );
  setCanvasHeight((getCanvasWidth() / 2));
  canvas = new Canvas(0, 0, getCanvasWidth(), (getCanvasWidth() / 2), canvas_name);
  return canvas;
}

Mixer* SortingAudialVisualization::createMixer() {
  mixer = new Mixer();
  return mixer;
}

void SortingAudialVisualization::setEvenDataChunks(const int v) {
  even_data_chunks = v;
}

bool SortingAudialVisualization::getEvenDataChunks() const {
  return even_data_chunks;
}

void SortingAudialVisualization::setBlockSize(const int bs) {
  block_size = bs;
}

int SortingAudialVisualization::getBlockSize() const {
  return block_size;
}

void SortingAudialVisualization::setNumberNormalBlockSize(const int n) {
  number_normal_block_size = n;
}

int SortingAudialVisualization::getNumberNormalBlockSize() const {
  return number_normal_block_size;
}

bool SortingAudialVisualization::getMainThread() {
  return main_thread;
}


SortingAudialVisualization::~SortingAudialVisualization() {
  delete canvas;
  delete mixer;
```

```
}

}


/**
 * BubbleSorter.h declares declares, overrides, and implements the algorithm
 * necessary to implement a bubble sort audialization and visualization. It
 * inherits from SortingAudialVisualizations.h
 *
 * Who: Nate Herder
 * When: 02/27/2020
 * Where: Calvin University
 *
 */

#pragma once

#include "SortingAudialVisualization.h"

namespace avlib {

class BubbleSorter : public SortingAudialVisualization {
 public:
  BubbleSorter(int argc, char **argv, bool value = false);
  void run();
  void BubbleSort(Canvas *can, std::vector<ThreadSynth> &voices, int data_elements);
};

}


/**
 * BubbleSorter.cpp defines the methods and algorithm required to make the
 * audial/visualizaiton for bubble sort
 *
 * Who: Nate Herder
 * When: 02/27/2020
 * Where: Calvin University
 *
 */

#include "BubbleSorter.h"

namespace avlib {

BubbleSorter::BubbleSorter(int argc, char **argv, bool value) :
SortingAudialVisualization(argc, argv, value) {
```

```
}


void BubbleSorter::BubbleSort(Canvas *can, std::vector<ThreadSynth> &voices, int
data_elements) {
  int cwh = getCanvasHeight();  // canvas window height
  ColorFloat color = RED;
  ColorFloat bg = BLACK;
  ColorFloat sort_done_color = WHITE;

  int block_size = getBlockSize();
  int number_normal_block_size = getNumberNormalBlockSize();
  int block_size_plus_one = block_size + 1;
  if (showVisualization()) {
        can->start();
  }

  // generate the data to sort
  int *numbers = new int[data_elements];  // Array to store the data
  for (int i = 0; i < data_elements; i++) {
        numbers[i] = rand() % getCanvasHeight();
  }

  // draw the original random data
  if (showVisualization()) {
        for (int i = 0; i < data_elements; i++) {
        if( i < number_normal_block_size ) {
        can->drawRectangle((i*block_size), (cwh-numbers[i]), block_size, numbers[i],
color);
        } else {

can->drawRectangle(((number_normal_block_size*block_size)+(((i-number_normal_bloc
k_size)*block_size_plus_one)) ), (cwh-numbers[i]), block_size_plus_one, numbers[i],
color);
        }
        }
  }

  int temp;
  // begin sorting
  for (int i = 0; i < data_elements; i++) {
        for (int j = 1; j < data_elements - i; j++) {
        if (numbers[j] < numbers[j-1]) {
        temp = numbers[j];
        numbers[j] = numbers[j-1];
        numbers[j-1] = temp;
```

```
        if (playAudialization()) {
        MidiNote note = Util::scaleToNote(numbers[j], std::make_pair(0,
getCanvasHeight()), std::make_pair(C3, C7));
        voices.at(0).play(note, Timing::MICROSECOND, 50);
        }

        if (showVisualization()) {
        can->sleep();  // recommended to sleep before drawing
        if(j <= number_normal_block_size) {
        can->drawRectangle((j*block_size), 0, block_size, cwh, bg);
        can->drawRectangle((j*block_size), (cwh-numbers[j]), block_size, numbers[j],
color);
        can->drawRectangle(((j-1)*block_size), 0, block_size, cwh, bg);
        can->drawRectangle(((j-1)*block_size), (cwh-numbers[j-1]), block_size,
numbers[j-1], color);
        } else {

can->drawRectangle(((number_normal_block_size*block_size)+(((j)-number_normal_bloc
k_size)*block_size_plus_one)), 0, block_size_plus_one, cwh, bg);

can->drawRectangle(((number_normal_block_size*block_size)+(((j)-number_normal_bloc
k_size)*block_size_plus_one)), (cwh-numbers[j]), block_size_plus_one, numbers[j], color
);

can->drawRectangle(((number_normal_block_size*block_size)+(((j-1)-number_normal_bl
ock_size)*block_size_plus_one)), 0, block_size_plus_one, cwh, bg);

can->drawRectangle(((number_normal_block_size*block_size)+(((j-1)-number_normal_bl
ock_size)*block_size_plus_one)), (cwh-numbers[j-1]), block_size_plus_one, numbers[j-1],
color);
        }
        }
        }
        }
 }

 if(playAudialization()) {
        voices.at(0).stop();
 }

 //after sorting turn data white
 if( showVisualization()) {
        for (int i = 0; i < data_elements; i++) {
        if(i < number_normal_block_size) {
        can->drawRectangle((i*block_size), (cwh-numbers[i]), block_size, numbers[i],
sort_done_color);
        } else {
```

```
can->drawRectangle(((number_normal_block_size*block_size)+((i-number_normal_block
_size)*block_size_plus_one)), (cwh-numbers[i]), block_size_plus_one, numbers[i],
sort_done_color);
        }
        }

        can->wait();
 }

 delete[] numbers;
}

void BubbleSorter::run() {
 if (showVisualization()) {
        createCanvas("Bubble Sort");
        canvas->setBackgroundColor(BLACK);
 }

 if (playAudialization()) {
        createMixer();

        voices = std::vector<ThreadSynth>(1, ThreadSynth(mixer));
        mixer->add(voices[0]);
        voices[0].setEnvelopeActive(false);
 }

 if (showVisualization() || playAudialization()) {
        BubbleSort(canvas, voices, getDataAmount());
 } else {
        std::cout << "neither -v or -a flags set" << std::endl;
        std::exit(0);
 }
}


}


/**
 * InsertionSorter.h declares declares, overrides, and implements the algorithm
 * necessary to implement a insertion sort audialization and visualization. It
 * inherits from SortingAudialVisualizations.h
 *
 * Who: Nate Herder
 * When: 02/27/2020
 * Where: Calvin University
 *
```

```
 */

#pragma once

#include "SortingAudialVisualization.h"

namespace avlib {

class InsertionSorter : public SortingAudialVisualization {
 public:
  InsertionSorter(int argc, char **argv, bool value = false);
  void InsertionSort(Canvas *can, std::vector<ThreadSynth> &voices, int data_elements);
  void run();
};

}


/**
 * InsertionSorter.cpp defines the methods and algorithm required to make the
 * audial/visualizaiton for insertion sort
 *
 * Who: Nate Herder
 * When: 02/27/2020
 * Where: Calvin University
 *
 */

#include "InsertionSorter.h"

namespace avlib {

InsertionSorter::InsertionSorter(int argc, char** argv, bool value) :
SortingAudialVisualization(argc, argv, value) {

}


void InsertionSorter::InsertionSort(Canvas* can, std::vector<ThreadSynth>& voices, int
data_elements) {
  int cwh = getCanvasHeight();
  ColorFloat color = RED;
  ColorFloat bg = BLACK;
  ColorFloat sort_done_color = WHITE;

  int block_size = getBlockSize();
  int number_normal_block_size = getNumberNormalBlockSize();
  int block_size_plus_one = block_size + 1;
```

```
 if (showVisualization()) {
        can->start();
 }

 // Generate data
 int* numbers = new int[data_elements];
 for (int i = 0; i < data_elements; i++) {
        numbers[i] = rand() % (getCanvasHeight());
 }

 // draw the original random data
 if (showVisualization()) {
        for (int i = 0; i < data_elements; i++) {
        if( i < number_normal_block_size ) {
        can->drawRectangle((i*block_size), (cwh-numbers[i]), block_size, numbers[i],
color);
        } else {

can->drawRectangle(((number_normal_block_size*block_size)+(((i-number_normal_bloc
k_size)*block_size_plus_one)) ), (cwh-numbers[i]), block_size_plus_one, numbers[i],
color);
        }
        }
 }

 int insertValue;
 int j;
 int temp;
 // begin sorting
 for (int i = 1; i < data_elements; i++) {
        insertValue = numbers[i];
        j = i;
        while (j > 0 && numbers[j - 1] > insertValue) {

        if(showVisualization()) {
        if(j < number_normal_block_size) {
        can->drawRectangle( (j*block_size), 0, block_size, cwh, bg );
        can->drawRectangle( ((j-1)*block_size), 0, block_size, cwh, bg );
        } else {
        can->drawRectangle(
((j*number_normal_block_size)+((j-number_normal_block_size)*block_size_plus_one)),
0, block_size_plus_one, cwh, bg);
        can->drawRectangle(
(((j-1)*number_normal_block_size)+((j-1)-number_normal_block_size)*block_size_plus_o
ne), 0, block_size_plus_one, cwh, bg);
        }
        }
```

```
        if (playAudialization()) {
        MidiNote note = Util::scaleToNote(numbers[j], std::make_pair(0,
getCanvasHeight()), std::make_pair(C3, C7));
        voices.at(0).play(note, Timing::MILLISECOND, 30);
        }

        numbers[j] = numbers[j - 1];
        if (showVisualization()) {
        can->sleep();
        if(j < number_normal_block_size) {
        can->drawRectangle( (j*block_size), (cwh-numbers[j]), block_size, numbers[j],
color );
        can->drawRectangle( ((j-1)*block_size), (cwh-insertValue), block_size, insertValue,
color );
        } else {
        can->drawRectangle(
((j*number_normal_block_size)+((j-number_normal_block_size)*block_size_plus_one)),
(cwh-numbers[j]), block_size_plus_one, numbers[j], color );
        can->drawRectangle(
(((j-1)*number_normal_block_size)+((j-1)-number_normal_block_size)*block_size_plus_o
ne), (cwh-numbers[j-1]), block_size_plus_one, numbers[j-1], color );
        }
        }
        j--;
        }
        numbers[j] = insertValue;
 }

 //make sure that the ThreadSynth stops playing
 if(playAudialization()) {
        voices.at(0).stop();
 }

 // after sorting turn data white
 if (showVisualization()) {
        for (int i = 0; i < data_elements; i++) {
        if( i < number_normal_block_size ) {
        can->drawRectangle((i*block_size), (cwh-numbers[i]), block_size, numbers[i],
sort_done_color);
        } else {

can->drawRectangle(((number_normal_block_size*block_size)+(((i-number_normal_bloc
k_size)*block_size_plus_one)) ), (cwh-numbers[i]), block_size_plus_one, numbers[i],
sort_done_color);
        }
        }

        can->wait();
```

```
  }

  delete[] numbers;
}


void InsertionSorter::run() {
  if (showVisualization()) {
        createCanvas("Insertion Sort");
  }
  if (playAudialization()) {
        createMixer();
  }

  if (playAudialization()) {
        voices = std::vector<ThreadSynth>(1, ThreadSynth(mixer));
        mixer->add(voices[0]);
        voices[0].setEnvelopeActive(false);
  }

  if (showVisualization()) {
        canvas->setBackgroundColor(BLACK);
  }

  if (showVisualization() || playAudialization()) {
        InsertionSort(canvas, voices, getDataAmount());
  } else {
        std::cout << "neither -v or -a flags set" << std::endl;
        std::exit(0);
  }

}


}


/**
 * MergeSorter.h declares declares, overrides, and implements the algorithm
 * necessary to implement a merge sort audialization and visualization. It
 * inherits from SortingAudialVisualizations.h
 *
 * Who: Nate Herder
 * When: 11/06/2019
 * Where: Calvin University
 *
 */
```

```
#pragma once

#include "SortingAudialVisualization.h"

namespace avlib {

class MergeSorter : public SortingAudialVisualization {
 public:
  MergeSorter(int argc, char **argv, bool value = false);
  void MergeSort(Canvas *can, std::vector<ThreadSynth> &voices, int threads, int size);
  void run();
};

}


/**
 * MergeSorter.cpp defines the methods and algorithm required to make the
 * audial/visualizaiton for merge sort
 *
 * Who: Nate Herder
 * When: 11/06/2019
 * Where: Calvin University
 *
 */

#include "MergeSorter.h"

namespace avlib {

enum MergeState {
  S_MERGE = 1,
  S_SHIFT = 2,
  S_WAIT = 3,
  S_DONE = 4,
  S_HIDE = 5
};

struct sortData {
  ColorFloat color;  // Color of the thread
  MergeState state;  // Current state of the threads
  int first, last,   // Start and end of our block
          left, right,   // Indices of two numbers to compare
          fi, hi, li,          // Indices of first middle and last numbers in a set
          depth;           // Current depth of the merge
  int* a;          // Array of numbers to sort
  int seg, segs;         // Current / total segments
  int size;
```

```
sortData(int* arr, int f, int l, ColorFloat c) {
        fi = hi = li = 0;  // Initialize indices
        left = right = 0;  // Initialize bounds
        color = c;       // Set the color
        a = arr;         // Get a pointer to the array we'll be sorting
        first = f;       // Set the first element we need to worry about
        last = l;        // Set the last element we need to worry about
        depth = 0;       // We start at depth 0
        seg = 0;
        segs = 1;  // We start on segment -1, with a total of 1 segment
        while (segs < (l - f)) {  // If the current number of segments is more than
                         // the # of elements, we're done
        ++depth;                 // Otherwise, increment the depth...
        segs *= 2;               //...and double the number of segments
        }
        state = S_SHIFT;  // Start Merging
        size = 2;
}

void restart(int l) {
        depth = 0;
        hi = last;
        right = hi + 1;
        last = li = l;
        fi = left = first;
        state = S_MERGE;
        size *= 2;
}

void sortStep() {
        int tmp, pivot, jump;
        switch (state) {
        case S_SHIFT:
        pivot = jump = segs / 2;
        fi = first;
        li = last;
        hi = (fi + li) /
        2;  // Set our half index to the median of our first and last
        for (tmp = depth; tmp > 0; --tmp) {
        jump /= 2;
        if (seg < pivot) {
        pivot -= jump;
        li = hi;  // Set out last index to our old half index
        } else {
        pivot += jump;
        fi = hi + 1;  // Set out first index to our old half index plus one
        }
```

```
        hi = (fi + li) /
        2;  // Set our new half index to the median of our first and last
        }
        left = fi;
        right = hi + 1;
        state = S_MERGE;  // We're ready to start Merging
        break;
        case S_MERGE:
        if (left > right || right > last) {
        seg = 0;        // Reset our segment(s)
        segs /= 2;  // We're now using half as many segments
        state = (depth-- == 0) ? S_WAIT : S_SHIFT;
        } else if (right > li) {
        ++seg;
        state = S_SHIFT;  // Move on to the next segment and recalculate our
                        // first and last indices
        } else if (left <= hi && a[left] < a[right]) {
        ++left;
        } else {
        tmp = a[right];
        for (int x = right; x > left; --x) a[x] = a[x - 1];
        a[left] = tmp;
        ++left;
        ++right;
        ++hi;
        }
        break;
        default:
        break;
        }
  }
};

MergeSorter::MergeSorter(int argc, char** argv, bool value) :
SortingAudialVisualization(argc, argv, value) {

}

void MergeSorter::MergeSort(Canvas* can, std::vector<ThreadSynth>& voices, int
threads, int size) {
  if (showVisualization()) {
        can->start();
  }
  const int IPF = 1;              // Iterations per frame

  // generate the data to sort
  int* numbers = new int[size];  // Array to store the data
  for (int i = 0; i < size; i++) {
```

```
        numbers[i] = rand() % (getCanvasHeight());
}

int bs = size / threads;
int ex = size % threads;
sortData** sd = new sortData*[threads];
int f = 0;
int l = (ex == 0) ? bs - 1 : bs;
for (int i = 0; i < threads; ++i) {
        sd[i] = new sortData(numbers, f, l, Colors::highContrastColor(i));
        f = l + 1;
        if (i < ex - 1) {
        l += (bs + 1);
        } else {
        l += bs;
        }
}

// begin sorting
#pragma omp parallel num_threads(threads)
{
        int tid = omp_get_thread_num();

        while (true) {
        if (showVisualization()) {
        can->sleep();
        }
        if (sd[tid]->state == S_WAIT) {  // Merge waiting threads
        if (playAudialization()) {
        voices.at(tid).stop();
        }
        if ((tid % sd[tid]->size) > 0)
        sd[tid]->state = S_DONE;
        else {
        int next = tid + sd[tid]->size / 2;
        if (next < threads && sd[next]->state == S_DONE) {
        sd[next]->state = S_HIDE;
        sd[tid]->restart(sd[next]->last);
        }
        }
        }
        for (int i = 0; i < IPF; i++) {
        sd[tid]->sortStep();
        }
        if (showVisualization()) {
        can->pauseDrawing();  // Tell the Canvas to stop updating the screen temporarily
        }
        int start = sd[tid]->first, height;
```

```
        int cwh = getCanvasHeight();
        ColorFloat color;
        ColorFloat bg = BLACK;
        if (sd[tid]->state != S_HIDE) {
        // Draw a black rectangle over our portion of the screen to cover up the old
drawing
        if (showVisualization()) {
        can->drawRectangle(start, 0, sd[tid]->last - sd[tid]->first, cwh, bg);
        }
        for (int i = sd[tid]->first; i < sd[tid]->last; ++i, ++start) {
        height = numbers[i];
        if (i == sd[tid]->left) {
        if(playAudialization()) {
        MidiNote note = Util::scaleToNote(numbers[i], std::make_pair(0,
getCanvasHeight()), std::make_pair(C3, C7));
        voices.at(tid).play(note, Timing::MICROSECOND, 50);
        }
        }
        if (sd[tid]->state == S_WAIT || sd[tid]->state == S_DONE)
        color = WHITE;
        else {
        if (i == sd[tid]->right || i == sd[tid]->left)
        color = WHITE;
        else if (i < sd[tid]->left)
        color = sd[tid]->color;
        else if (i >= sd[tid]->fi && i <= sd[tid]->li)
        color = Colors::blend(sd[tid]->color, WHITE, 0.5f);
        else
        color = Colors::blend(sd[tid]->color, BLACK, 0.5f);
        }
        if (showVisualization()) {
        can->drawLine(start, cwh - height, start, cwh, color);
        }
        }
        }
        if (showVisualization()) {
        can->resumeDrawing();  // Tell the Canvas it can resume updating
        }
        }

        if(playAudialization()) {
        voices.at(tid).stop();
        }
  }
}

void MergeSorter::run() {
  int num_threads = getNumThreads();
```

```
  if (showVisualization()) {
        createCanvas("Merge Sort");
        canvas->setBackgroundColor(BLACK);
  }
  if (playAudialization()) {
        createMixer();

        voices = std::vector<ThreadSynth>(num_threads, ThreadSynth(mixer));
        for (unsigned i = 0; i < voices.size(); i++) {
        mixer->add(voices[i]);
        voices[i].setVolume(0.5);
        voices[i].setEnvelopeActive(false);
        }
  }

  if (showVisualization() || playAudialization()) {
        MergeSort(canvas, voices, num_threads, getCanvasWidth());
  } else {
        std::cout << "neither -v or -a flags set" << std::endl;
        std::exit(0);
  }
}


}


/**
 * SelectionSorter.h declares declares, overrides, and implements the algorithm
 * necessary to implement a selection sort audialization and visualization. It
 * inherits from SortingAudialVisualizations.h
 *
 * Who: Nate Herder
 * When: 04/08/2020
 * Where: Calvin University
 *
 */

#pragma once

#include "SortingAudialVisualization.h"

namespace avlib {

class SelectionSorter : public SortingAudialVisualization {
 public:
  SelectionSorter(int argc, char **argv, bool value = false);
```

```
  void run();
  void SelectionSort(Canvas *can, std::vector<ThreadSynth> &voices, int data_elements);
};

}


/**
 * SelectionSorter.cpp defines the methods and algorithm required to make the
 * audial/visualizaiton for selection sort
 *
 * Who: Nate Herder
 * When: 02/27/2020
 * Where: Calvin University
 *
 */

#include "SelectionSorter.h"

namespace avlib {


SelectionSorter::SelectionSorter(int argc, char **argv, bool value):
SortingAudialVisualization(argc, argv, value) {

}

void SelectionSorter::SelectionSort(Canvas *can, std::vector<ThreadSynth> &voices, int
data_elements) {
  int cwh = getCanvasHeight();  // canvas window height
  ColorFloat color = RED;
  ColorFloat bg = BLACK;
  ColorFloat sort_done_color = WHITE;

  int block_size = getBlockSize();
  int number_normal_block_size = getNumberNormalBlockSize();
  int block_size_plus_one = block_size + 1;
  if (showVisualization()) {
          can->start();
  }

  // generate the data to sort
  int *numbers = new int[data_elements];  // Array to store the data
  for (int i = 0; i < data_elements; i++) {
          numbers[i] = rand() % getCanvasHeight();
  }

  // draw the original random data
```

```
 if (showVisualization()) {
        for (int i = 0; i < data_elements; i++) {
        if( i < number_normal_block_size ) {
        can->drawRectangle((i*block_size), (cwh-numbers[i]), block_size, numbers[i],
color);
        } else {

can->drawRectangle(((number_normal_block_size*block_size)+(((i-number_normal_bloc
k_size)*block_size_plus_one)) ), (cwh-numbers[i]), block_size_plus_one, numbers[i],
color);
        }
        }
 }

 // begin sorting
 int min;
 int temp;
 for (int i = 0; i < data_elements; i++) {
        min = i;
        for (int j = i; j < data_elements; j++) {
        min = numbers[j] < numbers[min] ? j : min;
        }

        if(showVisualization()) {
        can->sleep();  // recommended to sleep before drawing
        if(min < number_normal_block_size) { //this isnt going to work well work well for
odd data elemtns that don't divide evenly
        can->drawRectangle( (i*block_size), 0, block_size, cwh, bg );
        can->drawRectangle( (min*block_size), 0, block_size, cwh, bg  );
        }
        }

        temp = numbers[i];
        numbers[i] = numbers[min];
        numbers[min] = temp;

        if (playAudialization()) {
        MidiNote note = Util::scaleToNote(numbers[min], std::make_pair(0,
getCanvasHeight()), std::make_pair(C3, C7));
        voices.at(0).play(note, Timing::MICROSECOND, 30);
        }

        if (showVisualization()) {
        if(min < number_normal_block_size) {
        can->drawRectangle((i*block_size), (cwh-numbers[i]), block_size, numbers[i],
color);
        can->drawRectangle((min*block_size), (cwh-numbers[min]), block_size,
numbers[min], color);
```

```
        }
        }

  }

  if(playAudialization()) {
        voices.at(0).stop();
  }

  //after sorting turn data white
  if( showVisualization()) {
        for (int i = 0; i < data_elements; i++) {
        if(i < number_normal_block_size) {
        can->drawRectangle((i*block_size), (cwh-numbers[i]), block_size, numbers[i],
sort_done_color);
        }
        }

        can->wait();
  }

  delete[] numbers;

}

void SelectionSorter::run() {

  if (showVisualization()) {
        createCanvas("Selection Sort");
        canvas->setBackgroundColor(BLACK);
  }

  if (playAudialization()) {
        createMixer();

        voices = std::vector<ThreadSynth>(1, ThreadSynth(mixer));
        mixer->add(voices[0]);
        voices[0].setEnvelopeActive(false);
  }

  if (showVisualization() || playAudialization()) {
        SelectionSort(canvas, voices, getDataAmount());
  } else {
        std::cout << "neither -v or -a flags set" << std::endl;
        std::exit(0);
  }

}
```

```
}


/**
 * ShakerSorter.h declares declares, overrides, and implements the algorithm
 * necessary to implement a shaker sort audialization and visualization. It
 * inherits from SortingAudialVisualizations.h
 *
 * Who: Nate Herder
 * When: 04/08/2020
 * Where: Calvin University
 *
 */

#pragma once

#include "SortingAudialVisualization.h"

namespace avlib {

class ShakerSorter : public SortingAudialVisualization {
 public:
  ShakerSorter(int argc, char **argv, bool value = false);
  void run();
  void ShakerSort(Canvas *can, std::vector<ThreadSynth> &voices, int data_elements);
};

}


/**
 * SortingAudialVisualizations.cpp holds defines the common methods and data
 * across all sorting algorithms.
 *
 * Who: Nate Herder
 * When: 11/06/2019
 * Where: Calvin University
 *
 */

#include "SortingAudialVisualization.h"
#include "BubbleSorter.h"
#include "InsertionSorter.h"
#include "MergeSorter.h"
#include "SelectionSorter.h"
#include "ShakerSorter.h"
```

```
#include <cmath>

namespace avlib {

SortingAudialVisualization::SortingAudialVisualization(int argc, char** argv, bool value) :
AudialVisualization(argc, argv), main_thread{value} {
  if( (getCanvasWidth() % getDataAmount()) == 0 ) {
        setEvenDataChunks(true);
        setBlockSize( (getCanvasWidth()/getDataAmount()) );
        setNumberNormalBlockSize( getDataAmount() );
  } else {
        setEvenDataChunks(false);
        setBlockSize( floor( (getCanvasWidth()/getDataAmount()) ) );
        setNumberNormalBlockSize( (getDataAmount() - (getCanvasWidth() %
getDataAmount())) );
  }

  std::vector<std::string> sort_run_vector = getSortingAlgorithms();

  if( getMainThread() == true ) {
        #pragma omp parallel num_threads( sort_run_vector.size() )
        {
        int tid = omp_get_thread_num();

        if(sort_run_vector.at(tid) == "bubble") {
        BubbleSorter b(argc, argv);
        b.run();
        } else if(sort_run_vector.at(tid) == "insertion") {
        InsertionSorter i(argc, argv);
        i.run();
        } else if(sort_run_vector.at(tid) == "merge") {
        MergeSorter m(argc, argv);
        m.run();
        } else if(sort_run_vector.at(tid) == "selection") {
        SelectionSorter s(argc, argv);
        s.run();
        } else if(sort_run_vector.at(tid) == "shaker") {
        ShakerSorter sh(argc, argv);
        sh.run();
        }
        }
  }


}

Canvas* SortingAudialVisualization::createCanvas(std::string canvas_name) {
  setCanvasWidth(getCanvasWidth() );
```

```
  setCanvasHeight((getCanvasWidth() / 2));
  canvas = new Canvas(0, 0, getCanvasWidth(), (getCanvasWidth() / 2), canvas_name);
  return canvas;
}

Mixer* SortingAudialVisualization::createMixer() {
  mixer = new Mixer();
  return mixer;
}

void SortingAudialVisualization::setEvenDataChunks(const int v) {
  even_data_chunks = v;
}

bool SortingAudialVisualization::getEvenDataChunks() const {
  return even_data_chunks;
}

void SortingAudialVisualization::setBlockSize(const int bs) {
  block_size = bs;
}

int SortingAudialVisualization::getBlockSize() const {
  return block_size;
}

void SortingAudialVisualization::setNumberNormalBlockSize(const int n) {
  number_normal_block_size = n;
}

int SortingAudialVisualization::getNumberNormalBlockSize() const {
  return number_normal_block_size;
}

bool SortingAudialVisualization::getMainThread() {
  return main_thread;
}


SortingAudialVisualization::~SortingAudialVisualization() {
  delete canvas;
  delete mixer;
}

}


/**
```

```
 * linearSearch.h declares declares, overrides, and implements the algorithm
 * necessary to implement a linear search audialization and visualization. It
 * inherits from AudialVisualization.h
 *
 * Who: Nate Herder
 * When: 04/23/2020
 * Where: Calvin University
 *
 */

#pragma once

#include "AudialVisualization.h"

namespace avlib {

  class LinearSearch : public AudialVisualization {
  public:
        LinearSearch(int argc, char **argv);
        void makeSearch(Canvas *can, std::vector<ThreadSynth> &voices, int
data_elements, int search);
        void run();
  };

}


/**
 * LinearSearch.cpp defines the methods and algorithm required to make the
 * audial/visualizaiton for a linear search
 *
 * Who: Nate Herder
 * When: 04/23/2020
 * Where: Calvin University
 *
 */

#include "LinearSearch.h"
#include <cmath>

namespace avlib {

  LinearSearch::LinearSearch(int argc, char **argv) : AudialVisualization(argc, argv) {

  }

  void LinearSearch::makeSearch(Canvas *can, std::vector<ThreadSynth> &voices, int
data_elements, int search) {
```

```
        int cwh = getCanvasHeight();  // canvas window height
        ColorFloat color = RED;
        ColorFloat bg = BLACK;
        ColorFloat sort_done_color = WHITE;
        if (showVisualization()) {
        can->start();
        }

        int block_size;
        int number_normal_block_size;
        if( (getCanvasWidth() % getDataAmount()) == 0 ) {
        block_size = (getCanvasWidth()/getDataAmount());
        number_normal_block_size = getDataAmount();
        } else {
        block_size = floor( (getCanvasWidth()/getDataAmount()) );
        number_normal_block_size = (getDataAmount() - (getCanvasWidth() %
getDataAmount()));
        }
        int block_size_plus_one = block_size + 1;

        // generate the data to sort
        int *numbers = new int[data_elements];  // Array to store the data
        for (int i = 0; i < data_elements; i++) {
        numbers[i] = rand() % getCanvasHeight();
        }

        if(showVisualization()) {
        std::string display_text = "Searching for: " + std::to_string(search);
        can->drawText( display_text, 10, 20, 20, color );
        }

        // draw the original random data
        if (showVisualization()) {
        for (int i = 0; i < data_elements; i++) {
        if( i < number_normal_block_size ) {
        can->drawRectangle((i*block_size), (cwh-numbers[i]), block_size, numbers[i],
color);
        } else {

can->drawRectangle(((number_normal_block_size*block_size)+(((i-number_normal_bloc
k_size)*block_size_plus_one)) ), (cwh-numbers[i]), block_size_plus_one, numbers[i],
color);
        }

        }
        }

        int result_location = NULL;
```

```
        bool search_found = false;
        //begin searching
        for(int i = 0; i < data_elements; i++) {
        if(numbers[i] == search) {
        result_location = i;
        search_found = true;
        break;

        }else {

        if (playAudialization()) {
        MidiNote note = Util::scaleToNote(numbers[i], std::make_pair(0,
getCanvasHeight()), std::make_pair(C3, C7));
        voices.at(0).play(note, Timing::MICROSECOND, 50);
        }

        if(showVisualization()) {
        can->sleep();
        if( i < number_normal_block_size ) {
        can->drawRectangle( i*block_size, (cwh-numbers[i]), block_size, numbers[i],
sort_done_color );
        } else {
        can->drawRectangle(
((number_normal_block_size*block_size)+(((i-number_normal_block_size)*block_size_pl
us_one)) ), (cwh-numbers[i]), block_size_plus_one, numbers[i], sort_done_color );
        }
        }

        }
        }
        if(search_found) {
        std::string display_text = "Found at index: " + std::to_string(result_location);
        can->drawText( display_text, 10, 45, 20, color );
        } else {
        std::string display_text = "No matches found";
        can->drawText( display_text, 10, 45, 20, color );
        }


        if(playAudialization()) {
        voices.at(0).stop();
        }

        if(showVisualization()) {
        can->wait();
        }

        delete[] numbers;
```

```
 }

 void LinearSearch::run() {

        if (showVisualization()) {
        createCanvas("Linear Search");
        canvas->setBackgroundColor(BLACK);
        }

        if (playAudialization()) {
        createMixer();

        voices = std::vector<ThreadSynth>(1, ThreadSynth(mixer));
        mixer->add(voices[0]);
        voices[0].setEnvelopeActive(false);
        }

        int search = rand() % getCanvasHeight();

        if (showVisualization() || playAudialization()) {
        std::cout << "Searching for: " << search << std::endl;
        makeSearch(canvas, voices, getDataAmount(), search);
        } else {
        std::cout << "neither -v or -a flags set" << std::endl;
        std::exit(0);
        }

 }

}


/**
 * main.cpp is the driver
 *
 * Who: Nate Herder
 * When: 11/06/2019
 * Where: Calvin University
 *
 */

#include "SortingAudialVisualization.h"
#include "LinearSearch.h"

using namespace avlib;

int main(int argc, char **argv) {
```

```
  SortingAudialVisualization av(argc, argv);

  // LinearSearch ls(argc, argv);
  // ls.run();

  return 0;
}


#Target to make and objects needed
TARGET = main
OBJS = $(TARGET).o MergeSorter.o BubbleSorter.o InsertionSorter.o
SortingAudialVisualization.o AudialVisualization.o SelectionSorter.o ShakerSorter.o
LinearSearch.o

#Compiler, remove command, and OS we're working on
CC = g++
RM = rm -f
UNAME := $(shell uname)

#Check if we're on a Mac or a Linux machine

#Linux
ifeq ($(UNAME), Linux)
   OS_LFLAGS :=
   OS_INCLUDE :=
   OS_LDIRS := -L/usr/lib -L/usr/local/lib
   OS_EXTRA_LIB :=
   OS_GLFW := glfw
   OS_GL := -lGL
endif

#Mac
ifeq ($(UNAME), Darwin)
   OS_LFLAGS := -framework Cocoa -framework OpenGl -framework IOKit -framework
Corevideo
   OS_INCLUDE :=
   OS_LDIRS :=
   OS_EXTRA_LIB :=
   OS_GLFW := glfw3
   OS_GL :=
   OS_EXTRA_LINK :=
endif

#Compiler flags
CXXFLAGS=-c -O3 -g3 \
         -Wall -Wextra -pedantic-errors \
         -I/usr/include/ \
```

```
        -I/usr/include/TSGL/ \
        ${OS_INCLUDE} \
        -I/usr/include/freetype2/ \
        -I/usr/include/freetype2/freetype  \
        -I/usr/local/include/tsal/ \
        -std=c++11 -fopenmp \
        -Wno-unused-function #Supress warnings

#Linking flags
LFLAGS=-o $(TARGET) \
        ${OS_EXTRA_LIB} \
   ${OS_LDIRS} \
        ${OS_EXTRA_LINK} \
        -ltsgl -lfreetype \
        -lGLEW -l${OS_GLFW} \
        -lportaudio \
        -ltsal \
        -lX11 ${OS_GL} -lXrandr \
   -L/usr/lib \
   -L/usr/local/lib \
        -fopenmp \
        ${OS_LFLAGS}

.SUFFIXES: .cpp .o

#all command
all: $(TARGET)

#Linking
$(TARGET): $(OBJS)
   @echo "\nLinking $(TARGET)..."
   $(CC) $(OBJS) $(LFLAGS)
   @echo

#Compiling
.cpp.o:
   @echo "\nCompiling $<..."
   $(CC) $(CXXFLAGS) $<

#Clean command
clean:
   $(RM) $(TARGET) $(OBJS) *~
```