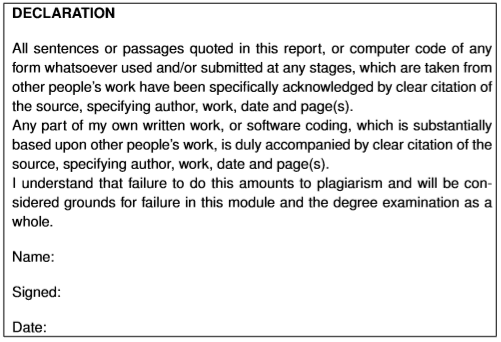
|  |
| --- |
|  |
|  |
| **Animation of Algorithms: Dissertation**  CO3015 Computer Science Project  Department of Informatics, University of Leicester  By Joel Mahon  Submitted: May, 2019 |



Joel Mahon

02/05/2019

# Abstract

This paper discusses a program for aiding understanding of algorithms. Algorithms can sometimes overwhelm computer science students and deter younger people from studying computer science. This paper outlines a way to supplement learning in a digestible way that doesn’t overwhelm or deter people using an approach informed by prior work on this topic, the amount of prior work on this topic is vast and provides a strong foundation.

A Windows executable that provides interactive 3D animations of three common or famous sorting algorithms was created: Bubble Sort, Merge Sort, and Quick Sort. The tool represents data, either submitted by the user, generated randomly, or a default set, as cubes with numbers on them. The cubes then have the position and colour animated and manipulated to visualise the actions of the algorithms, with the user controlling the animation playback. This is further supplemented by each algorithm having a code representation that highlights the code currently being used to execute the current or next action, whichever is appropriate. The program is extendable via abstraction and allows third-parties to create their own algorithms using existing actions and allows them to add their own actions.

The successful abstraction and third-party support proved to be harder than the features that supported learning of the directly incorporated three algorithms.

# Table of Contents

[1. Abstract 1](#_Toc7720583)

[2. Table of Contents 2](#_Toc7720584)

[3. Introduction 4](#_Toc7720585)

[3.1 Sorting Algorithms 4](#_Toc7720586)

[3.2 Animation 4](#_Toc7720587)

[3.3 Aims 5](#_Toc7720588)

[3.4 Objectives 5](#_Toc7720589)

[3.5 Challenges and their solutions 5](#_Toc7720590)

[3.5.1 An intuitive GUI 5](#_Toc7720591)

[3.5.2 Interactivity 5](#_Toc7720592)

[3.5.3 Understanding 6](#_Toc7720593)

[3.5.4 Third-party extension 6](#_Toc7720594)

[3.6 Originality 7](#_Toc7720595)

[4. Prior Work 7](#_Toc7720596)

[4.1 Interactivity 7](#_Toc7720597)

[4.2 3D virtual tools for teaching 8](#_Toc7720598)

[4.3 Previous Animated Algorithm(s) software 9](#_Toc7720599)

[5. Requirements 10](#_Toc7720600)

[5.1 User Interface 10](#_Toc7720601)

[5.2 3D Rendering 13](#_Toc7720602)

[5.3 Animated Algorithms 13](#_Toc7720603)

[6. Design 14](#_Toc7720604)

[6.1 Main (loop) 15](#_Toc7720605)

[6.2 GUI engine 15](#_Toc7720606)

[6.3 3D rendering engine 15](#_Toc7720607)

[6.4 Algorithms engine 16](#_Toc7720608)

[6.4.1 Main Engine 16](#_Toc7720609)

[6.4.2 Actions engine 17](#_Toc7720610)

[6.4.3 Timelines engine 18](#_Toc7720611)

[6.4.4 Paths engine 18](#_Toc7720612)

[7. Data Structures 18](#_Toc7720613)

[7.1 3D rendering engine 18](#_Toc7720614)

[7.2 Algorithms engine 18](#_Toc7720615)

[8. Algorithms 19](#_Toc7720616)

[8.1.1 Build actions 19](#_Toc7720617)

[9. Implementation 25](#_Toc7720618)

[9.1 Main (loop) 25](#_Toc7720619)

[9.1.1 Camera position control 25](#_Toc7720620)

[9.1.2 Camera angle control 27](#_Toc7720621)

[9.2 GUI engine 28](#_Toc7720622)

[9.2.1 Set camera to see scene 28](#_Toc7720623)

[9.2.2 ImGui usage 30](#_Toc7720624)

[9.3 3D rendering engine 31](#_Toc7720625)

[9.3.1 Initialise the model 32](#_Toc7720626)

[9.3.2 Shader class 35](#_Toc7720627)

[9.3.3 Pass uniforms and render the model 39](#_Toc7720628)

[9.4 Algorithms engine 40](#_Toc7720629)

[9.4.1 Main engine 40](#_Toc7720630)

[9.4.2 Actions engine 43](#_Toc7720631)

[9.4.3 Timelines engine 45](#_Toc7720632)

[9.4.4 Paths engine 46](#_Toc7720633)

[9.5 Miscellaneous 48](#_Toc7720634)

[9.5.1 Settings singleton 48](#_Toc7720635)

[10. Testing 50](#_Toc7720636)

[10.1 Requirement tests 50](#_Toc7720637)

[10.2 Stress tests 50](#_Toc7720638)

[11. Critical Appraisal 50](#_Toc7720639)

[11.1 Critical analysis 51](#_Toc7720640)

[11.1.1 An intuitive GUI 51](#_Toc7720641)

[11.1.2 Interactivity 51](#_Toc7720642)

[11.1.3 Understanding 53](#_Toc7720643)

[11.1.4 Third-party extension 55](#_Toc7720644)

[11.1.5 Software quality 57](#_Toc7720645)

[11.2 Real world context 57](#_Toc7720646)

[11.3 Personal development 57](#_Toc7720647)

[12. Conclusion 58](#_Toc7720648)

[13. Glossary 59](#_Toc7720649)

[14. Bibliography 60](#_Toc7720650)

[15. Appendix 62](#_Toc7720651)

# Introduction

For this project I have developed software that primarily aims to teach people from all age groups, provided they have at least basic computer literacy, more about some common and/or famous algorithms. The specific algorithms are bubble sort[1], merge sort[2], and quick sort[3].

A secondary aim was to provide a public API with which third-parties could create their own animated algorithms without considerable effort, this aim only exists to support the primary aim.

These aims were achieved by utilising Open Graphics Library (OpenGL).

*Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering.*[4]

And GLFW.

*GLFW is a lightweight utility library for use with OpenGL. GLFW stands for Graphics Library Framework. It provides programmers with the ability to create and manage windows and OpenGL contexts, as well as handle joystick, keyboard and mouse input.*[5]

The 3D rendering is handled in the software solution at the lowest reasonable level for a project not aiming to create its own OpenGL or DirectX substitute.

The (2D) graphical user interface (GUI) rendering is done using Dear ImGui[6], an immediate graphical user interface library, hence the name ImGui. Immediate means that the GUI is drawn every frame dynamically.

## Sorting Algorithms

Sorting algorithms, in the most abstract sense, take an ordered collection of elements (as in the elements have order, unlike a bag or set, not that the elements are in order) and produce an ordered collection of correctly ordered elements as output. The correct order of said ordered collection of ordered elements is determined by the comparison operator between two elements (e.g. integer1 < integer2 will return true if integer1 is less than integer2, but any type may have a way to be compared to another of the same type to produce a Boolean result).

## Animation

In the virtual world, because computers are binary and not analogue, there are no truly seamless changes to values. Everything has a specific translation/scale/rotation, those values are digital and not analogue, and animation in the context of this project is the illusion of seamless change in these values by changing them in such small increments that the human eye does not perceive a sudden change in value.

*To create the illusion of movement, an image is displayed on the computer monitor and repeatedly replaced by a new image that is similar to it, but advanced slightly in time (usually at a rate of 24, 25 or 30 frames/second).*[7]

## Aims

As previously mentioned, I aimed to create a program able to teach people from all age groups, provided they have at least basic computer literacy, more about some common and/or famous algorithms. And to provide an API that a third-party could use to instrument animations of their own algorithms without considerable effort.

## Objectives

To achieve the above aims there are several broad objectives that must be completed.

* To research previous examples of media that present animated algorithms.
* To research techniques for improving student learning.
* To identify successful aspects of each piece of researched media.
* To identify the best educational techniques that can be incorporated into a software solution.
* To design a software system that incorporates the successful aspects of the researched media.
* To implement the designed software system.
* To verify that the software system implements the design.

## Challenges and their solutions

To achieve the above aims there are a few broad challenges that must be solved.

### An intuitive GUI

There’s a challenge of poor user understanding of how to use the program leading to user frustration and slower program operation (which in turn would impede learning because more time is spent focusing on how to interact with the GUI instead of on learning the algorithms).

To overcome this a clear to use GUI, that's as simple as possible whilst still providing all the necessary control the user needs, must be implemented.

### Interactivity

There’s a challenge of poor user engagement, the user should be engaged, to enhance their learning.

*We estimate the learning benefit from extra doing (1 SD [standard deviation] increase) to be more than six times that of extra watching or reading* [8]*.*

*[PollEverywhere Audience Response System] promoted interactivity, focused attention, and provided feedback on comprehension. A total of 95% reported that it increased their participation and found that it clarified their thinking and helped to focus on key points. Another 81.7% mentioned that it increased their motivation to learn. Students regarded it as a useful method for giving real-time feedback, which stimulated their performance and participation. Data from CS students echoed the findings from the dental students. Reports from focus groups demonstrated that this strategy was helpful in focusing students’ attention and in clarifying information.*[9]

(My supervisor also has made good use of a similar technology to increase student interaction, and that the interaction was good for student learning.)

To overcome this the user must have ways to influence the algorithms, including but not limited to deciding the values to be used and in what order, and being able to move forwards and backwards through the algorithm.

### Understanding

There’s a challenge of poor user understanding of what is happening in the algorithms, which naturally impedes learning,

To overcome this the user must have the ability to view explanations of what just occurred, and what is about to occur, and the abstract way in each algorithm works.

I have also made the animations in 3D using 3D models, this has been shown to aid learning in this[10] and other[11] contexts.

*The third dimension provides an extra degree of freedom for conveying*

*information*[10]

*“Teachers talk a lot and you just sort of tune out, but when you see things it is there and suddenly it all makes sense.” – Pupil comment*[11]

In addition, the models representing the data change colour to indicate what their current context within the algorithm is.

### Third-party extension

There’s a challenge of avoiding wasting effort reinventing the wheel when seeking to teach more algorithms than just those included within my software solution.

To overcome this the public API should be such that a third-party can add additional algorithms whilst making use of already written code without having to reproduce those features themselves.

For this I have instrumented an innovative approach to store algorithms as: a vector of actions, a current state, a current progress through the current action, and a function to build the vector of actions.

To add your own algorithm, you merely need to implement the AAlgorithm abstract class, and create new implementations of the AAction class if needed (as more implementations are added, the need to add new ones goes down, since existing ones can be reused on new algorithms with no hassle.

You then need to build up a code representation for the user to see while they interact with the algorithm.

This entire process (to add a dynamic animation of a new algorithm capable of taking custom input) can take as little as four hours from scratch, provided you are acquainted with the API.

## Originality

My project has very little originality, I can safely say no significant new ground has been broken, tools animating algorithms have been done numerous times. As far as I know my project introduces almost no new notable functionality, not because I copied existing tools, but because almost every feature I could think of had already been done by someone else, or was outright detrimental to implement, or too time consuming to implement. There’s some originality in the fact that my program combines features in a unique way as far as I have been able to observe in prior work done in this field.

Online there are animated examples of sorting algorithms in 2 dimensions, many that allow custom user input, many that visually represent the sorts in a similar way, many that have intuitive and broadly functional playback control mechanisms[12].

There are also videos of sorts that are rendered in 3 dimensions[13]. And programs that produce 3D animations of algorithms from custom input data, with playback control[14].

There are also programs that allow you to add your own animated algorithms using their framework which will do most the heavy lifting[14].

There are programs which are similar in almost every way such as ViSA[15].

Mine is the only example I can find that makes use of traversal of the vertical space to show recursion depth, allows custom input, and is in 3D all at the same time. And it is subjective, but I think the clarity of the movement of data in my animated algorithms is superior to all prior work I’ve seen.

# Prior Work

This section aims to review the literature that was used to inform the approach of the project.

## Interactivity

While it is widely believed that you learn better by doing. This has been shown empirically by a group of researchers at Carnegie Mellon University in their paper “Learning is Not a Spectator Sport”[8].

In the paper they estimate that 1 standard deviation increase of “Doing” grants the benefits of more than 6 times the benefits of reading or watching.

However, as far as relevance to this project, there are issues, the MOOC (Massive Open Online Course) only really has quizzes when it comes to interactivity. While a quiz is interactive, and my program is interactive, a quiz requires you to recall knowledge where as my program doesn’t, the paper itself cities theories that learning by doing is

*important because most of human expertise involves tacit knowledge of the cues and conditions for deciding when, where, and what knowledge to bring to bear in complex situations.*[16]

This sentiment is corroborated by Eiman Abdel Meguid and Matthew Collins in their paper on Students’ perceptions of lecturing approaches[9].

Within said paper they use empirical evidence to assess using a large sample size of 300 students or more. The evidence suggested that the instrument was “highly reliable”.

Neither paper controls for the fact that the interactivity is in the form of testing (despite that not being the only form of interactivity) so we cannot draw concrete conclusions for whether our brand of interactivity will produce as good results. However, it seems safe to conclude that it will produce better results than an otherwise equivalent but not interactive program.

## 3D virtual tools for teaching

I began this project with a bias that hands on learning is better, not just in the context of it being interactive as previously discussed, but because (like general interactivity may) I theorise it engages more of the brain, like spatial awareness.

While as a computer science student I could have possibly opted to use a micro controller with a servo arm to teach algorithms in the real world, I decided based on my research that a 3D virtual application would provide adequate benefits.

The Gaia 3D white paper[11] which was written about by the BBC[17] says that

*The research results suggested that the 3D animated models were able to represent information in the most economical manner to facilitate learning and comprehension, thus simplifying complex, abstract and impossibly large amounts of information into a coherent form*

While the BBC article only covers them using 3D to teach biology, a subject where the vast quantity of information is apparent, I think it is still applicable in my project. My reasoning is that there is more information to be conveyed with algorithms than one might realise, while a bubble sort may be simple, a quick sort needs to convey the recursion depth and position, as well as the sizes of each recursion, as well as the pivot, etc.

And regardless of whether there is an information bottleneck, the paper also says it improved student engagement, despite not being an active exercise, the students were much less likely to “tune out”.

*“Teachers talk a lot and you just sort of tune out, but when you see things it is there and suddenly it all makes sense.” – Pupil comment*[11]

However, Gaia is a business venture, it exists to make money, we cannot take their word as gospel because they have inherent biases. But, Professor Anne Bramford (the white paper author) is highly certified and has multiple works in the field of the value of creative education.

## Previous Animated Algorithm(s) software

I will mention four examples of software with the same or similar aims as mine.

Firstly, Algorithm Animation Using 3D Interactive Graphics[10], which dates back all the way to 1993.

This program was for algorithms on graphs, so is less directly relevant, but does claim there are three uses of 3D that are all relevant to my project.

*for providing additional information about objects that are intrinsically two-dimensional, for uniting multiple views, and for capturing a history of execution.*

Vectors of integers are intrinsically one dimensional, however, when you consider recursion, they become intrinsically three dimensional.

Later, in the paper they discuss a heap sort, which is relevant as it is a type of sort, they specifically cover how the use of 3D provides value as they can attribute data to a 2D position using depth (as their Figures 3a-3d show).

The paper is nearly 30 years old, showing just how far we’ve come since in the tools we can use, if it was good then when teachers were just as good, then it should be even better now. Even at that time, they were also able to cite several programs that animated algorithms in 3D.

Secondly, ViSA[15]. The paper is a short coverage of a 2D program that is more directed towards university students, as shown by the figures of the UI that are quite complicated, certainly too advanced for a typical 12-year-old with no programming experience.

Their previous work research is extensive and covers a vast number of examples to anchor myself from.

Thirdly, Algorithm Explorer[14]. A 3D animated algorithms visualisation software. This is the most sophisticated of the four programs in my opinion. And the program mine is most like despite any discrepancies in quality.

It allows the building of new algorithms from a simple API; this is particularly relevant because that is one of the more uncommon features in the area of Animated Algorithms that I implement.

They have an interesting approach to their algorithm creation, giving the creator three possible “worlds” to create in, each with a different theme. This is an interesting way to tackle algorithm variety, but I believe it goes against abstraction principles and is ultimately a short cut that sacrifices long term maintainability and extendibility.

Their prior work research is also extensive, providing further programs to anchor myself on.

However, I do not feel this program makes good use of the vertical space when showing recursion.

Finally, Visualizing Sorting Algorithms[18]. I chose this one because of their user feedback survey. As shown in my challenges and their solutions section, this paper had users provide feedback on what features they would have liked to have.

Until now, the benefits to user experience from implementing any features from past projects was mostly speculative. But with these user studies I can be more confident that my features to colour blocks based on their state, allow users to control speed of animation playback, and showing recursion through distinct separation of blocks, are all valuable.

# Requirements

I decided early on to use the V-Model methodology, I laid out the requirements, designed a system to meet those requirements, implemented that system, then unit tested the system (or tried to, but could not overcome some compilation issues), then applied system testing to confirm that all “**must**” requirements were met. This was done for bubble sort first, then a shorter second V-Model work flow was done for merge sort, then another for quick sort.

This was after considering AGILE, which I am most familiar with, but I opted not to use due to the overhead. The overhead of AGILE is worth it when you have a customer, especially when working in a team or on a large project, but on a small project with one person who is also the customer, like my project is, tracking work items, assigning story points, etc. are all less efficient.

I have opted to include justifications within some requirements, they are marked in bold and are not part of the requirement themselves. This is to provide a better flowing reading experience than if they were written is a separate section with sporadic references to this section.

Imperatives (**must**, **should**, **could**) are in bold.

Requirements that were not met are underlined.

Please refer to the glossary for any terms that don’t have clear or apparent meaning.

## User Interface

1. There **must** be no console window in the release version of the program.
2. Each interactive graphical user interface elements (e.g. buttons, sliders) **must** have a stated purpose viewable by the user.
3. Each page **must** not have more than 10 distinct interactive graphical user interface elements **to avoid overwhelming the user**.
   1. Except for all step control choices being considered a single element **due to their unified nature**.
4. There **must** be a “Main menu” page, a hub that allows the user to navigate using a single mouse click to the pages: “Learn”, “Options”, and “About”.
5. When the program is executed, the window **must** present the “Main menu” page to the user as the initial visible state.
6. There **must** be an “Exit” button on the “Main Menu” page that closes the program.
7. There **must** be a “Learn” page that allows the user to navigate to any of the introductions to each algorithm, the pages hosting the interactive algorithms, and an introduction to the programming concepts used within the algorithms.
8. The introduction to the programming concepts used within the algorithms **must** introduce the concepts of variables, Boolean expressions, if statements, while statements, for loops, and vectors.
9. The introductions to each algorithm and the pages hosting the interactive algorithms **must** not be able to be navigated to if the introduction to programming concepts used within the algorithms has not yet been navigated.
10. Each of the pages hosting the interactive algorithms **must** not be able to be navigated to if their respective introduction has not been navigated to yet.
11. The relevant pages that have been used **must** persist between program executions provided the user doesn’t tamper with anything used to facilitate the persistence.
12. Each page other than the “Main menu” page **must** allow the user to navigate to the previous page with only a mouse click.
13. Each page that hosts an interactive algorithm **must** show a C++ code representation of said algorithm.
    1. Use of non-existing functions are permitted if the description of what it does is on that page.
14. Each code representation of an algorithm **must** highlight to the user which part of the code is currently being executed if part way through a step animation.
15. Each code representation of an algorithm **must** highlight to the user which part of the code is next to be executed if not part way through a step animation.
16. Each code representation of an algorithm **must** contain step explanations for any non-trivial steps (e.g. setting an index variable to 0 at the start of a for loop).
17. Each page that hosts an interactive algorithm **must** provide step control to the user.
18. Step control **must** not be able to exceed the number of steps an algorithm has.
19. Step control must not be able to precede the first step (i.e. cannot go to a negative step).
20. Each page that hosts an interactive algorithm **must** allow the user to provide custom data using any of:
    1. In-application input (e.g. using a keyboard, or if the user is disabled, they may use a different piece of hardware to input data).
    2. File input (e.g. using a file explorer pop up or entering a file path).
21. Each page that hosts an interactive algorithm **must** allow the user to load randomised data with a single mouse click.
22. Each page that hosts an interactive algorithm **must** allow the user to load data that has an apparently random order but is always the same whenever it is loaded, with a single mouse click **to allow the user to quickly compare algorithms using the same data without having to provide their own**.
23. Each page that hosts an interactive algorithm **could** allow the user to load ascending sorted data with a single mouse click.
24. Each page that hosts an interactive algorithm **could** allow the user to load descending sorted data with a single mouse click.
25. The number of values that are loaded when loading generated data (random, ascending, descending), **must** be configurable by the user.
26. The number of values that are loaded when loading generated data (random, ascending, descending), **must** persist between program executions provided the user doesn’t tamper with anything used to facilitate that persistence.
27. Each page that hosts an interactive algorithm **must** pause the step controller and disable interactivity with the rest of the user interface when a modal dialog is shown.
28. Each page that hosts an interactive algorithm **must** allow the user to view the step breakdown for the current active configuration (i.e. data).
29. Each page that hosts an interactive algorithm **must** allow the user to navigate to the “Options” page with a single mouse click.
30. Each page that hosts an interactive algorithm **must** allow the user to navigate to the “Main menu” page with a single mouse click.
31. Each page that hosts an interactive algorithm **must** display to the user, for each variable within the algorithm the variable’s value, if the variable is both set and in scope.
32. Each page that hosts an interactive algorithm **must** display to the user for each variable within the algorithm that the variable is out of scope, if the variable is out of scope.
33. Each page that hosts an interactive algorithm **must** display to the user for each variable within the algorithm that the variable is not set, if the variable is in scope but not set.
34. Each page that hosts a recursive interactive algorithm **must** display to the user the current position within the “tree of recursion” that the algorithm is currently at.
35. Each page that hosts an interactive algorithm **must** allow the user to view the valid range of input values.
36. Each page that hosts an interactive algorithm **must** allow the user to view the valid range of number of input values.
37. The “Options” page **must** allow the user to view/set the following
    1. The colour of the text/number models.
    2. The default colour for the box models.
    3. The colour of box models that relate to the “left” recursive call.
    4. The colour of box models that relate to the “right” recursive call.
    5. The colour of the highlighted box models.
    6. The colour of the box models that are sorted.
    7. The colour of executing code.
    8. The colour of non-executing code.
    9. The rate at which animation occurs.
    10. Whether or not the camera is “free”.
    11. Whether or not to use full screen.
    12. Whether or not to show the frames per second in the title bar.
38. The configuration in the “Options” page **must** be used by the program for their respective purposes.
39. The configuration in the “Options” page **must** persist between program executions provided the user doesn’t tamper with anything used to facilitate that persistence.
40. The “About” page **must** credit anyone legally required to be credited.
41. The “About” page **should** credit anyone who provided significant aide to the project.
42. The “About” page **should** tell the user what language the software solution was written in.
43. The “About” page **could** tell the user what development environment the software solution was developed in.
44. The “About” page **should** tell the user that this software solution was made as part of my final year project for my computer science degree at the University of Leicester.
45. The “About” page **must** tell the user that the algorithms are not optimised.
46. The “About” page **must** tell the user that the C++ code displayed may not compile.
47. The “About” page **must** tell the user the legal risks of redistributing the program.
48. The program **must** close if the user presses the “Esc” key at any time, if the program is responding.
49. The program **must** become full screen if the user presses the “F11” key at any time, if the program is responding and is not full screen.
50. The program **must** stop being full screen if the user presses the “F11” key at any time, if the program is responding and is full screen.
51. The user interface **must** not obstruct the animation at immediately before the start or immediately after each step if the user is not interacting with a modal.
52. The 3D scene **must** be rendered within the window regardless of window dimensions.

## 3D Rendering

1. Models that are children to other models **must** incorporate the model matrix of the parent model when they are being rendered by the parent. **This is to** **allow** **us to move/scale/rotate composite models, or groups of models, without moving each one individually**.
2. Polygons of rendered models representing the data **must** always be simulated to be lit more than complete darkness.
3. Polygons of rendered models representing the data, that are not facing away from the light, **must** be simulated to appear better lit, based on how directly the plane of the polygon is facing the light.
4. Polygons of rendered models representing the data, **must** be simulated to appear better lit when the surface of the polygon is positioned such that if it were a mirror in real life the user would be able to see the light source.
5. Rendered models **must** have a colour from which lighting multiplies upon.
6. The user **must** be able to alter the view matrix’s position to view from and position to look to.
7. The projection matrix **must** use the current value of the program window aspect ratio whenever it is used to render.

## Animated Algorithms

1. There **must** be an animation of bubble sort.
2. There **must** be an animation of merge sort.
3. There **must** be an animation of quick sort.
4. There **should** be an animation of swapping two integers without a temporary variable.
5. There **should** be an animation of converting denary to binary.
6. There **should** be an animation of converting binary to denary.
7. There **could** be an animation of gnome sort.
8. There **could** be an animation of comb sort.
9. There **could** be an animation of cocktail sort.
10. There **could** be an animation of insertion sort.
11. There **could** be an animation of binary insertion sort.
12. Each algorithm’s code **must** be correct, i.e. for any given input it **must** always produce the correct output. E.g. bubble\merge\quick sort **must** always produce a sorted vector.
13. Each step of each algorithm **must** be correct, i.e. for any given input the algorithm **must** always have the correct state on any legal step.
14. The data of each algorithm **must** be visualised using models that visually match the values being represented.
15. Each step animation **must** be seamless (i.e. no model should move from one place to another in one frame unless the distance is so low that it appears to be moving).
16. Models **must** not clip at any point (i.e. polygons from distinct models must not occupy the exact same space at the exact same time).
17. There **must** be an animation for the action of comparing two arbitrary elements of some data.
18. There **must** be an animation for the action of swapping two arbitrary elements of some data.
19. There **must** be an animation for the action of taking an arbitrary number of adjacent elements from some data.
20. There **must** be an animation for the action of appending some data to some other data.
21. There **must** be an animation for the action of prepending some data to some other data.
22. There **must** be an animation for the action of taking the head (first element) of some data and appending it to some other data.
23. The animation of each algorithm **must** make visible which elements are sorted at the current step.
24. The animation of each algorithm **must** make visible the recursive position of each element at the current step.
25. The animation of each action **should** make the cubes accelerate/decelerate rather than have a constant velocity.
26. The animation of each action **could** make a sound at the end **to indicate it has been completed**.

# Design

At the highest level there are 4 intercommunicating systems. The main (loop), the GUI engine, the 3D rendering engine, and the algorithms engine.

There is separation of concerns, meaning for example the GUI engine never renders 3D models and never performs actions on the data, the 3D rendering engine never renders any GUI and never performs actions on the data, and the algorithms engine never renders any GUI and never renders 3D models. The main (loop) doesn’t provide direct functionality other than processing user input and window manipulation, but it does moderate communication between the 3 other systems.

The high-level architecture of the intercommunication is visualised in Figure 33. Everything else (within reason) is intra-communication.

## Main (loop)

The main (loop) follows the basic structure:

1. Setup
2. Loop until break is prompted
3. Clean up
4. Terminate program

The loop’s high-level structure can be summarised as:

* Update time
* Process/poll input
* Initialise the ImGui frame
* Update the view matrix
* Configure shader uniforms
* Clear the render buffer
* Render desired 3D scene
* Generate desired ImGui page + process ImGui interaction
* Render ImGui
* Swap render buffers
* Repeat

There is no multithreading. There are two call backs, one for mouse input, one for window resizing, but these are synchronous.

## GUI engine

Within the GUI engine there are pages, each used to render a window with unique contents, the contents of which are covered in the implementation and requirements.

There isn’t any notable architecture within the engine, a different method is called depending on which page should be currently showing, and that method renders the window for that page. Simplicity is good when it works if it is maintainable, such is the case here.

The requirements determine the specific navigational layout. Examples of each page can be seen in Figure 39, Figure 40, Figure 41, Figure 42, Figure 43, Figure 44, Figure 45, Figure 46, Figure 47, Figure 48, and Figure 49.

## 3D rendering engine

The 3D rendering engine is only an extension of following a tutorial. Much of it isn’t my own design but rather the design of Joey DeVries[19], what little of it that is of my own design is mostly just extracting functions so that I could create scenes using function calls rather than hardcoding them, which would not only have been bad practice, but also impossible to meet the requirements with, most notably the requirements involving custom data input.

In my project, the model data (triangles, stored as three positions and three normals each, totalling 18 floats per triangle) are passed in to a vertex buffer object at the next free location, the size of the model data and the index the data were stored at are both stored in a struct. The vertex buffer object is then bound to a vertex array object, the ID of which is also stored in the same struct.

Then when it comes time to render the model, this struct is used to retrieve these values which allows rendering.

This is better explained in the implementation section.

## Algorithms engine

### Main Engine

Each algorithm implements the abstract algorithm class, which powers the functionality that is common across all algorithm implementations. It has public members to:

* Get the current code ID: used by the GUI engine to highlight the code that corresponds with the current step.
* Get the current step: used by the GUI engine to facilitate relative step changes (increment one or decrement one), or set it directly using a slider.
* Get the number of steps: used by the GUI engine to prevent requesting the step be changed to outside the valid range, as well as jump to the last step.
* Set the step to a new value: used by the GUI engine and it performs the necessary steps, in reverse if needed, to go from the current step, to the desired step.
* Add progress: used by the GUI engine as it progresses the animation by the given amount and increments the step if the amount given is enough to complete said step.
* Get current progress: used by the GUI engine to decide whether the step back one step button goes to the previous step, or just resets the current step, the latter being the case when the value is above zero, meaning that it is currently mid step.
* Render: used by the Main (Loop) to render the algorithm. It calls the render method on each of its models, as well as adjusting their colours based on the state of the algorithm. This function is the only virtual public member.
* Get step break down: used by the GUI engine to return a map with all the types of actions within the algorithm along with how many times they occur.

It has protected pure functions to declare the contract of an algorithm implementation.

* Build actions: the implementations must implement this method such that it populates the vector of actions given the initial state.
* Build render node: the implementations must implement this method such that it sets up the initial scene of models given the initial state.

It also has a few protected members to aid the creation of inheriting algorithms and enforce the contract.

* Clear: will clear up the member variables of the base class to an initial state.
* Initialise: will call the Build render node and build actions functions of the base class, then reset the state to zero whilst preserving the actions.
* Push back and do action: will move a passed action into the actions vector and then executes it, returning the result of the execution. This is so that in the build actions function the implementation can be significantly simpler. Using this function, you can almost exactly copy line for line the code of the algorithm you want to animate. You then change the existing actions, such as increment/compare/assign, to use the actions engine versions, push back and do those actions in the same location in program flow. The base algorithms engine will then do the heavy lifting. The conversion requires no creative thought and provided the actions have been implemented already a parser could be made to do it without too much difficulty.
* The actions vector: all the actions.
* The render node: the scene of models.
* The current step.

In addition, each implementation adds its own member variables required to store the state.

For all the sort algorithms this includes the vector of elements that make up the data.

For bubble sort this includes the index at which to next swap, the number of sorted elements, and a convenience variable to store the length of the data rather than repeatedly accessing it.

For both merge sort and quick sort this includes the recursive branches, which are of the same respective type, as well as a public function to return the location of the current active part of the recursive execution.

For quick sort this includes the partition index, as well as two other indices used in creating the partition.

The algorithms engine has an actions sub engine that only it is privy to operate.

### Actions engine

Each action implements the abstract action class, which powers the functionality that is common across all action implementations. It has public members to:

* Get type: return an enumerable to be used to build the step breakdown.
* Get code Id: return the code Id for this action and is used to get the current code Id.
* Get progress: return the progress through this action and is used to get the current progress.
* Add progress: if the progress added will complete the action then just return the excess, otherwise, set the progress of each timeline to the current progress plus the added progress.
* Reset progress: set the progress of each timeline to 0.

It also has pure public functions for:

* Do: the implementations must implement this method such that it manipulates the algorithm state and the 3D scene from immediately before this action such that it is as it should be after the action is done.
* Undo: the implementations must implement this method such that it can take the algorithm state and 3D scene from immediately after this action such that it is as it was before the action was done.

There are protected members to access the play progress and timelines.

Each implementation must have a constructor that sets up the timelines required to animate the respective action, to do this each constructor may have significantly different parameters. They must also each have a specified duration that is constant for that action type.

The action sub engine has a timelines sub engine that only it is privy to operate.

### Timelines engine

The timeline sub engine powers the animation functionality using paths, it is very basic, only containing a public vector of paths and a public function to set the progress within that vector of paths.

The timeline sub engine has a paths sub engine that only it is privy to operate.

### Paths engine

Each path implements the abstract path class, which powers the functionality that is common across all path implementations. It has public members to:

* Get duration: return how long the path is.
* Set progress: adjust the matrix reference to be proportionally along the path as the new progress divided by the duration. The path is decided by the easing function, the start and end positions, and the specific path implementation.

There are also protected members to store the progress and the easing function.

# Data Structures

There are no particularly complicated to understand data structures used, the complexity and innovation arises from how they are used, so this section will be brief.

## 3D rendering engine

Each render node has zero to many child nodes in a vector. This forms a multiway tree of variable depth. Children do not know their parent in this tree.

## Algorithms engine

When correctly set up: the main (loop) has zero to many algorithms, each algorithm has zero to many actions, each action has zero to many timelines, and each timeline has one to many paths. Each is stored in vectors. This forms a multi way tree of up to four depth (including the root).

Children of all types do not know their parent in this tree.

# Algorithms

Due to my maintainable architectural, design, and data structure choices I haven’t needed to rely on complex or innovative algorithms to tackle any challenges, most challenges were solved using either loops, basic recursion techniques, switch statements, or mathematical formula(s) (which will be covered in the implementation), but not multiple together. This also applied to even the most difficult challenges I faced.

So instead in this section I will instead cover the algorithms that I am teaching, despite the fact they are not are not advanced, innovative, complicated, or difficult.

Since they are for use as teaching aids, they are not optimised in any way that would complicate them to the user.

Each are best explained in the application itself if you prefer to refer to that instead.

### Build actions

Each algorithm had a specific implementation of the previously discussed build actions. These are effectively just the original sorting algorithms with a quick conversion applied to cause them to build the data. So, since this is not implementation, I will discuss the base sorting algorithm, since they are both the same in an abstract way apart from the fact that for each action my algorithms also store it in the actions vector. This is like a watered-down version the command pattern[20].

#### Bubble sort

Figure 1 shows the code the build actions method on which my bubble sort is based. I would make it more abstract but each part of the for statement is crucial when building the actions, since it must be extremely deliberate. It is not meant to be actual code, but rather very “code like” pseudo-code.



Figure , bubble sort code

This algorithm works by making “passes” along the data, each pass involves comparing each pair of adjacent elements in ascending order, if the values corresponding to those indices are not locally sorted, then they are swapped. This causes each pass to bring the value most desired to be at the end to the end, hence the name bubble sort since elements “rise” like bubbles rise in water, until eventually you pass enough times that it is sorted.

In the outer for loop the doneCount stops at data.size() – 1, because once all but one elements are sorted we know the last element is sorted, since a lone element cannot be out of order.

The inner loop stops at data.size() – doneCount – 1, because we don’t need to check the already sorted elements, nor do we need to compare the last previously unsorted element to the first sorted element, because by the time the index is at that value the correct element will have been swapped into that place if it wasn’t already there.

An illustration of the execution of a bubble sort can be seen in Figure 2.

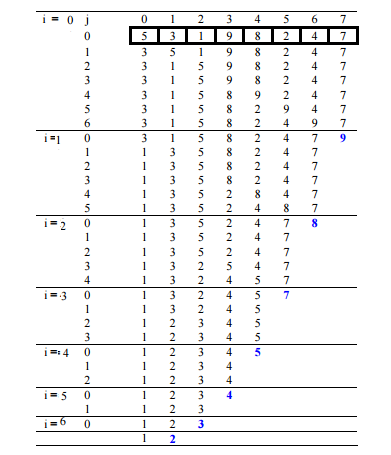


Figure , bubble sort execution from Geeks for Geeks[1]

#### Merge sort

Figure 3 shows the code the build actions method on which my merge sort is based. It is not meant to be actual code, but rather very “code like” pseudo-code.



Figure , merge sort and merge code

The .getFirsthalf, .getLastHalf, .erases, and .appends would not compile as they are, they are like pseudo-code for something not intended to be taught. Their meaning is self-explanatory, except for perhaps erase. Erase with a single size\_t as input in this context is meant to represent that the element at the index of the given size\_t will be erased.

The algorithm works by splitting the input data recursively until the data is trivial to sort (data of length 1 or 0 is already sorted, thus trivial to sort). Then before returning, the two recursive branches each supply a sorted list, which are merged together in the merge function. The merge function simply compares the first element of each child vector and pushes it back on the data to be returned repeatedly until one child vector is empty, then finally it appends the remaining child vector to the data to return and returns it.

An illustration of the execution of a merge sort can be seen in Figure 4.



Figure , merge sort execution from Geeks for Geeks[2]

#### Quick sort

Figure 5 shows the code the build actions method on which my merge sort is based. It is not meant to be actual code, but rather very “code like” pseudo-code.



Figure , quick sort and partition code

.takeRange is would not compile as it is, it is pseudo-code for something not intended to be taught. It creates a vector including values starting from the index given by the first parameter, up to the index before the last parameter (it does not take the element at the index of the last parameter). The copied range is then erased from the source and the copy is returned. Similarly .prepend and .append would also not compile, but their behaviour is self-explanatory.

The algorithm works by partition choosing a pivot. In our case this is just the first element (which is generally not optimal, but as stated before, nothing that makes the algorithms harder to understand will be included). All the values are then placed either before the pivot if they are less than or equal to the pivot, or after the pivot otherwise. This is done through comparing each element to the pivot sequentially, and either leaving it where it is, or placing it next to the group of “greater than” elements at the end of the data. The pivot is then swapped between the two groups and its index is returned.

Then quicksort is called recursively on each side of the partition (pivot excluded) and the result prepended or appended to the data depending on whether it was data from before the pivot or after the pivot respectively.

Once the data is short enough to be trivially sorted (data of length one or zero is already sorted, hence trivially sorted), then it returns the data.

An illustration of the execution of a quick sort can be seen in Figure 6.



Figure , quick sort execution from Geeks for Geeks[3]

# Implementation

All code is written in C++, except for the shaders which are written in GLSL. The program was developed in visual studio 2017.

This section will focus on the innovative and/or impactful implementations, especially those used to overcome the more difficult challenges.

Any implementation that is not predominately mine will usually be excluded, there will be exceptions when the implementation is particularly critical.

## Main (loop)

### Camera position control

Figure 7 shows the code for the camera position control. If the globally available camera (from Settings, a singleton for storing settings) is free then the code executes once per frame.

The first few lines are simple, they read the current key state of the relevant keys then check if they are pressed as a Boolean expression. The unary minus operator of Boolean and Boolean produces an integer in C++, as the Booleans are implicitly converted into integers equalling 0 or 1 for false or true respectively. The result is that if both buttons are pressed the result is 0, if just the first button is pressed the result is 1, and if just the second button is pressed the result is -1.

The next line declares the variables that we will be using the store the change in camera position.

Note, yaw is about the vertical world axis.

The first if statement checks if there should be horizontal movement, also known as strafing. If this is the case then, using trigonometry, we calculate the vector for relative horizontal movement which is dependent on the yaw of the camera rather than using absolute horizontal, we then multiply that vector by the time since last frame to produce a constant rate of movement, independent from the framerate. We then multiply the resultant vector by the speed of the camera, and finally we then multiply the resultant vector by the key press variable to invert it if ‘a’ is pressed instead of ‘d’.

The second if statement is identical but for forward/backwards movement and using the ‘w’ and ‘s’ keys instead of ‘d’ key and ‘a’ key respectively. The yaw is also adjusted by 90 degrees since forwards/backwards is a 90 degree difference from right/left.

We then calculate the vertical movement; we don’t check there is any before calculating because there is no trigonometry since the vertical movement is in the absolute vertical axis regardless of pitch and yaw of the camera. Therefore, there is low or possibly negative efficiency to checking for vertical movement before calculating it.

Finally, we update the camera position to be equal to the difference in position added to the current position.



Figure , camera position control code

### Camera angle control

Figure 8 shows the code for camera angle control. If the globally available camera (from Settings, a singleton for storing settings) is free then the code executes up to once per frame (if the mouse moved).

As the first comment says, the first if statement has the purpose of ensuring that if you have the LMB pressed while opening the program that it will not think you dragged the mouse from (0,0) (default double values) to its current position.

Next is just some basic arithmetic to get the difference in position since last movement.

We check if the LMB is pressed, if it’s not then we do nothing because we don’t want to adjust the camera angle if the user is not holding the LMB.

Then, we change the yaw and pitch by a factor of the difference in X and Y mouse positions respectively. The factor like with camera position control is based on delta time, to ensure the same real-world rate of change independent of framerate, and a tailored speed variable.

Finally, we clamp pitch to avoid letting the camera go upside down (which would serve no useful purpose and only cause confusion among users).



Figure , camera angle control code

## GUI engine

### Set camera to see scene

Figure 9 shows the code for repositioning the camera and pointing it at the scene such that the whole scene is visible from start to finish. If the globally available camera (from Settings, a singleton for storing settings) is free then the code executes once every time the window is altered, algorithm is altered, or page is changed to an algorithm page.

Firstly, the maximum width and height of the scene is calculated using values passed by the callers as well as global constants that are used when assigning positions to scene models.

The data size is based on the initial data size of the currently being viewed algorithm, in case some of the data has been taken for use in recursion. The data depth is based on the maximum depth of the execution of the currently being viewed algorithm, so that as the depth changes, we don’t have to reposition the camera.

Then, the fraction of the window that the algorithm will occupy is calculated by subtracting the fraction of the window the GUI occupies. This is calculated to ensure we meet the requirement of the GUI not obscuring the algorithm.

Then, the depth required to capture both the width and height of the algorithm is calculated, the larger value is used to ensure the entire width and height are visible.

The camera is then positioned at the calculated depth and at half the calculated width (with the padding removed).

The height is calculated by first centring the camera like we did with width. Then we subtract the height at the depth, multiplied by the fraction of the screen that the GUI occupies, and divide it by 2, this centres it now in the space that the GUI doesn’t occupy, rather than the centre of the entire window.

Finally, reset the pitch and yaw in case the user was looking in a different direction.



Figure , set camera to see scene code

Please look at Figure 34, Figure 35, and Figure 36 to see some examples of how the camera repositions itself to contain the entire algorithm.

### ImGui usage

This section will give some example code of usage of ImGui with comments on what the usage does. These are were not challenging or innovative, but they are required context in order to understand how the GUI is generated.

Figure 10 shows the code to create a window, it is not uncommon for ImGui elements to follow this basic pattern of starting and ending method, but most just use a single method call.

What is common is that most ImGui elements rely heavily on pushing and popping configuration as shown before and after the window.



Figure , example empty ImGui window

This code, as the first method name suggests, creates the ImGui window in the centre of the application window. What isn’t obvious is that the ImGui window will resize to fit the contents. This is passed in as information in the “windowFlags” parameter. How windowFlags is set is shown in Figure 11. Some lines are commented out both to make it quick to turn on certain flags if the programmed desires and more importantly to make it clear what flags are not set from a glance.



Figure , windowFlags configuration code

The automatic sizing can be overridden in one or both of the dimensions simply by passing the desired value(s) as a parameter to the ImGui::Begin without changing windowFlags.

Figure 12 shows typical code for a button, simply bread and butter for the GUI code, most elements follow this pattern, possibly without the if statement. The if statement is entered when the element is interacted with. For a check box where you might suspect the if statement to be entered when the check box is checked, no, it’s when, and only when, it is adjusted by the user.

Setting the previous page and current page is self-explanatory, but the reason it is so important to monitor these values is so that when the user changes pages, if the new page contains an algorithm that has data loaded the camera needs to move. One of the downsides to an immediate mode GUI like ImGui is that you render the window next frame instead, so we need to set this flag so that it knows to reposition the camera, since ImGui has no knowledge of what it was rendering last frame by default, nor the changes since last frame.



Figure , button with tool tip code

Figure 13 shows how the above code looks in application when the user is hovering the mouse over said button.

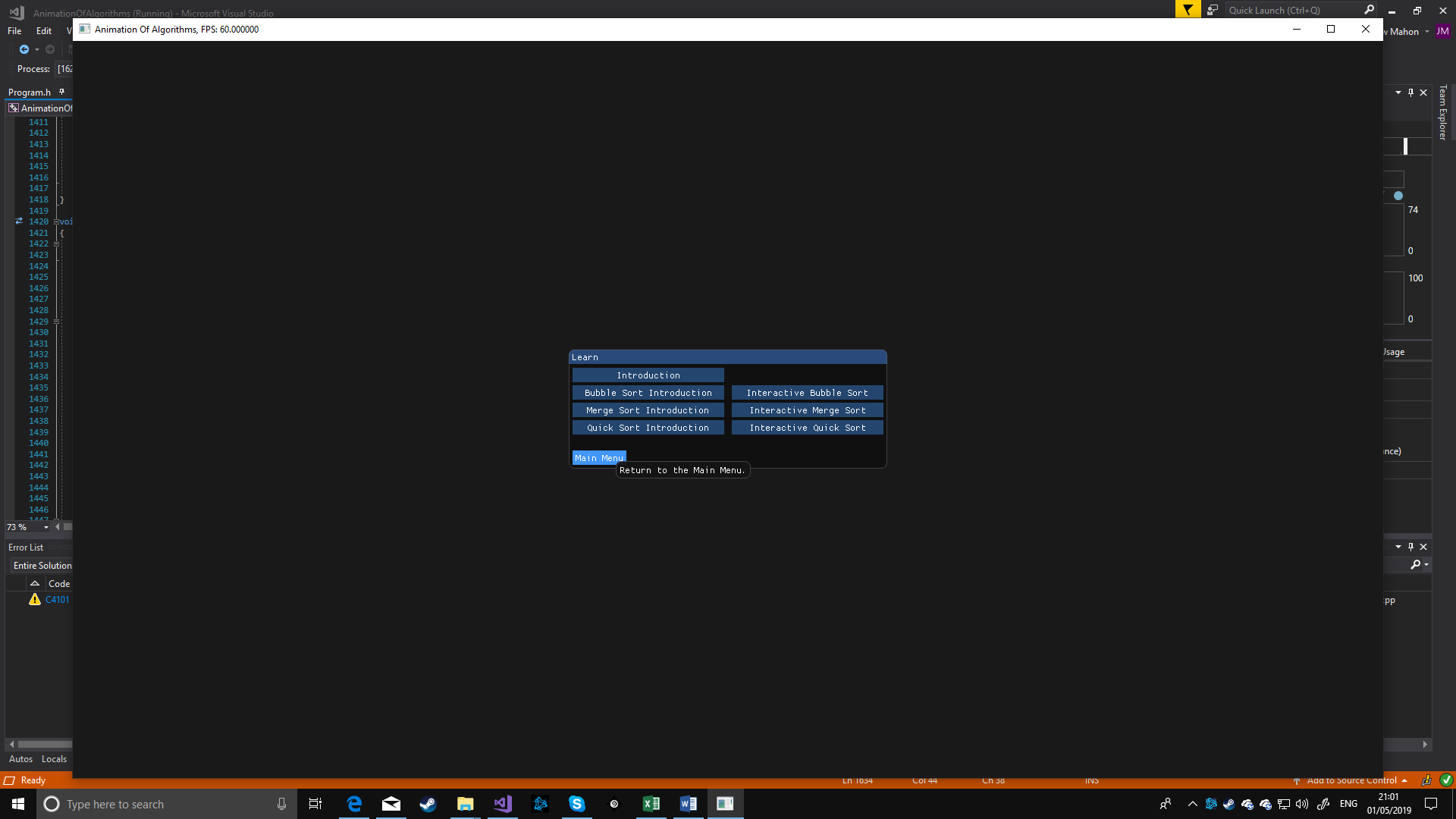


Figure , example of button being hovered over

Based on what I’ve said so far everything else in the GUI is basically just LegoTM - putting together these simple building blocks in the correct order to get what you want. While there are over 1500 lines of code for the GUI, there is not much more to it. There are a few tricks to get things where you want like ImGui::SameLine(), and columns, but nothing worth sharing.

## 3D rendering engine

All macros are powered by GLAD[21], a useful online tool used to define the macros used throughout the rendering engine.

Most of the low-level graphics programming is not programmed by me[19], all I did was extract code into methods when they were not properly parameterised.

However, while I could just think of the low-level OpenGL code as a “black box”, I decided to inform myself of how it works.

To render a model in this project, which is the end goal:

1. Before you do anything first you need to do some basic initial set up, calling a function to initialise OpenGL, set the version, etc.
2. You need to initialise the model by loading its data into a vertex buffer object, binding that to a vertex array object, then storing the vertex array object ID, the size of the data you stored, and the position you stored it in the vertex buffer array.
3. You need to bind a shader program, which is compiled at run time using a vertex shader (which transforms vertices of the model into screen space, and in our case calculates absolute world space of the vertices in order to pass on to the next shader) and a fragment shader (which calculates the colour the given pixel should be based on whatever inputs and processes you give it).
4. Pass in any “uniforms” to the shader, values that are constant among all executions of that shader until they are set again.
5. Using in the previously stored values, such as the vertex array object ID, bind the vertex array object and call to draw the section of it that contains the model (you may have multiple models stored in one vertex array object).

In the following sections I will explain the above process in far more detail, along with the code to do it.

### Initialise the model

The tutorial, as far as the chapter I stopped at, never formalised how to load multiple models into one vertex array object, however it became a necessity to do exactly this part way through my project. This is because copious use of different vertex array objects is extremely slow and switching between them is where a bottleneck can appear.

To avoid this, I made a function shown in Figure 14 to take many models and initialise them using the OpenGL API using just one vertex array object.

The code works by loading the model data from each file and concatenating them all into one large vector, all the while storing the location and size of each model within that vector into an array of structs.



Figure , code for initialising multiple models with one vertex array object

Then the data has the code shown in Figure 16 applied to it, this stores it in a format that the shaders can use. As you can see the code begins with more binding, because of the internal state machine OpenGL uses there will be a lot of binding!

Secondly, we configure the vertex attributes. This is telling OpenGL how our data is formatted, it doesn’t format the data, it is stored in a formatted state in the files we loaded.

First parameter: the position of vertex attribute in the vertex shader.

Second parameter: the “size” of the vertex attribute, for position and normal this is three, as in three floats.

Third parameter: the type, in both cases just floats but more adventurous shaders may use other types.

Fourth parameter: a flag that will have the data be normalised before use if set to true, while it is important for our normals to be normalised, we don’t need to enable this as they are already normalised, and the calculations to normalise them again are wasted.

The cube model is from the tutorial[19], and the number models are from Thingiverse[22], a site for sharing models to 3D print.

Firth parameter: the size of the “stride”, this determines the gap to leave between each vertex’s data’s start. Meaning if the stride is 24 bytes, like it is in our case (sizeof(float) is four and FLOATS\_PER\_VERTEX is six, because it is the sum of FLOATS\_PER\_POSITION and FLOATS\_PER\_NORMAL), then if the byte index of the start of first vertex is zero, then we are telling it that the second vertex will start at 24 bytes in. This is visualised in Figure 15.

Final parameter: the offset or displacement, in order to make use of the fifth parameter, or vice versa, we need to tell OpenGL where the first part of the vertex data for this attribute is stored, if they all started at byte zero then there’d be no way to distinguish one from the other. In our case we have “tightly packed” our data, meaning there are no padding bytes (which is perfectly legal), so with positions starting at zero bytes, and being three floats long, then followed by the normal starting immediately afterwards. Three floats are 12 bytes, but we let the compiler do that equation for us. This is visualised in Figure 15.

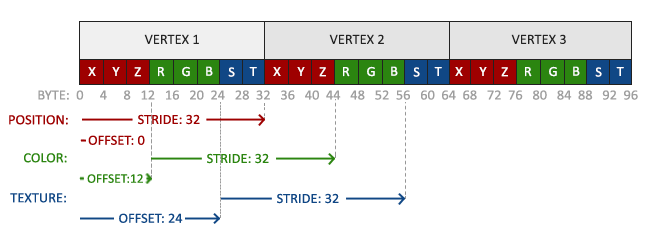


Figure , visualisation of stride and offset, from Joey DeVries’s tutorials[19]



Figure , code for storing model data using the OpenGL API

And that’s it! If we pass on all our model paths to the first function, and an array of models to contain them, then the result is an array of model structs that contain the information needed to render our models. Difficult to understand at first, but eventually it clicks, and you’ll wonder how you ever were confused.

### Shader class

There is a shader class in my project, 95% from Joey DeVries’s tutorial on OpenGL[19]. Most of it is just wrapping existing “gl” macros, an example is shown in Figure 17.



Figure 17, example Shader class wrapping function

There is however interesting code, shown in Figure 18, when first loading shaders, none of which is of my doing, but I will explain regardless.

Firstly, the code for the two shaders, vertex and fragment, are compiled and checked for correct compilation. Then, a shader program is created, and the two compiled shader programs are attached to it. Finally, we “link” the shader program, this is the process of compiling the program that consists of both shaders. This can fail if the two shaders are not compatible, for example, if the fragment shader needs some variables that the vertex shader doesn’t output, then the linking will fail.



Figure 18, shader initialisation code

The shader code (Figure 37 and Figure 38) itself is 100% from the tutorial[19], but I can explain how it works. When a model is rendered (the calls involved will be discussed later), every vertex is processed in isolation by the vertex shader. Then shapes are assembled based on the parameters used to call the render (in our case always triangles). Then if you have bound a geometry shader (which we have not) then it is executed.

Following that, there is rasterization, where the previous data is interpolated into pixels that fill the screen space. The interpolation is based on the pixel’s distance to each vertex, the closer it is to a vertex the more in common it will have with the data calculated in the vertex shader, if it’s dead centre then it will have the sum of a third of each of the vertices values, etc.

Then finally the fragment shader is applied which calculates the colour! And sometimes the depth, but not in our case.

Firstly, the vertex shader works out the position of the vertex in absolute world space (not based on the position of the camera, which is how the rendering is done – the world is moved around the camera and the camera stays still, at least mathematically it works that way). We need this to calculate specular lighting, the lighting due to the most direct reflecting of light, such as when the reflection of light from a wrist watch is blinding.

Then it works out the normal after accounting for the model’s rotation, which again is crucial for the lighting.

Finally, it calculates the vertex’s position in screen space, this considers the camera’s position and the direction it is facing, as well at the aspect ratio of the screen and field of view, even culling vertices that are very close or very far away!

These three are then passed into the fragment shader.

The fragment shader calculates “ambient” which is just a cheat to stop things being pitch black if the light isn’t on them. This simulates how even if you look under your bed, and there’s no direct path for the light, you can still see something other than pitch black because of light reflecting off things.

“Diffuse” which is the component of light spreading diffusely after hitting a surface, because the surface isn’t perfectly smooth it scatters light. The next part of the code simulates that.

Finally, “specular”, a component of light based on a bias that occurs despite not being perfectly smooth, surfaces are rarely so rough that there isn’t a bias towards light being reflected in a certain direction (the direction of the normal to the surface). The “specular” block simulates this.

Then the summation of these components is the colour the pixel. This is done for every pixel on screen, usually at least 60 times a second. On a 1920x1080 monitor at 60 frames per second that is over 120 million (124416000) executions of this shader! Simply amazing!

### Pass uniforms and render the model

I wrote my own “RenderNode” class. It is called a node because they take a tree like structure, where root renders itself then each of its sub trees recursively.

It stores which model (if any) it should render, as well as that model’s translation, scale, rotation, and sheer relative to its parent (child nodes are affected by all transformations upon their parent recursively). It also stores the colour the model should be rendered, but as a pointer so that the user can change it live in the options.

Figure 19 shows how to pass uniforms to the shader and render the model.

Firstly, we bind the shader the set the uniforms, setting them is easy provided you use the same name as you did in the shader, otherwise it won’t work. In side the Shader class it uses the name and its own shader ID to find the location using another API call.

Then we bind the vertex array object that our model is within, then we make a call to draw the array of triangles from the vertex index (not the float index, which is what is stored in model.first and model.count) to the last vertex index (calculated by the API given the “count” of vertices).

Finally, we render the child nodes in the same way if the recursive flag is set.



Figure , render a RenderNode code

## Algorithms engine

### Main engine

#### Set step

Figure 20 shows the code to set the step of an algorithm, this includes all the positioning and data manipulation, due to a proper separation of concerns it is quite simple for what it achieves. It is part of the base class for algorithms (AAlgorithm) because the abstract concept of actions will act upon their respective type if they are designed to properly meet the action contract.

To change the step to a new step firstly we must reset the progress of the current step, to ensure that if the step was partially way through a step and we set the step to the same step we are on that it resets the animation, which only really appears in the step controller as “go back one step” if you are partially through a step.

Then we clamp the new step to a legal value, within 0 and the maximum number of actions.

Finally, we iterate either forwards, incrementing step, doing the action for the step after each increment, or backwards, decrementing step, undoing each action for the step before after each decrement.

Provided each action is correctly programmed and constructed this will always ensure that settings the step will produce the expected state for both the variables of the algorithm and the 3D scene.



Figure , set step code

#### Add progress

Figure 21 shows the code to “add progress” to the current step, which in layman’s terms is to put the algorithm forward an arbitrary amount of time (it is scaled by the animation speed before this point and is based on the delta time since last frame). It is part of the base class for algorithms (AAlgorithm) because the abstract concept of actions will act upon their respective type if they are designed to properly meet the action contract.

To do this, first as usual we check to avoid doing things on the last step since that’s when the animation is over. If not then we pass the baton to the action instance, if so, then we return false to tell the GUI to stop playing.

After the action has progress added to it, there’s a chance that the amount of progress added to it was equal or more than enough to complete it, for example, it was 0.99 seconds through a 1 second long action and then deltaTime was 0.05, resulting in a 0.04 second excess. Regardless of whether the excess is 0 or 1000, if it is not negative, it is still indicative to the step being completed, and we increment the step to reflect that.

If the user is playing the animation continuously then we want any excess time to be added onto the next step, otherwise there would be a one frame pause at the end of every action, which is imperceptible at any decent frame rate, but I like to do things right!

Finally, we return a value indicating whether to continue playing if the user has pressed the “animate next step” button. If the step has completed, we return true, so the GUI knows to pause.



Figure , add progress code

#### Push back and do action

Figure 22 shows the code for a very simple but very useful function that I came up with to allow me to easily convert existing algorithms into animated ones.

As is shown in the next section, by on-the-fly constructing, pushing back, and doing these actions as part of the program flow allows you to simply substitute existing actions in an algorithm for my counter parts of a similar name as showcased in Figure 23.



Figure , push back and do action code



Figure , showcase of the transition of an original action to one used to generate and animation

#### Algorithm implementation

The code is too large to show here, and I hope I’ve described it well enough already to make it understood, there is almost no difference between the algorithms that are being animated, and the code to generate their actions, the main difference is highlighted in Figure 23.

The other difference is that recursion is done by calling the method on children rather than within the same instance. This probably wasn’t required but it felt like the most intuitive way to manage scope of the variables. Since the data and the variables involved in the algorithm are member variables and always in scope, having child instances produces an emulation of something like the scope.

### Actions engine

The actions engine is like the algorithms engine in terms of abstraction, there are two methods that are implemented in the abstract class for an action in order to manage step progress. One simply resets all the timelines (each action has a vector of timelines; each timeline manages the “path” or animation that a single “RenderNode” and its children take). This is a simply for each loop.

The second adds progress to each timeline, and is also essentially a simple for each loop with some additional value processing at the start in order to calculate excess and the absolute progress by adding the deltaTime to the current progress.

And in general the actions abstract class defines the contract to which the other actions must follow, having a Do, and Undo.

The interesting code in the actions engine is for some of the action implementations.

They each follow the basic structure of a constructor, which sets up the timeline(s) and stores any references required to alter the data later in a command pattern[20] like fashion. A Do method, which takes the data from the state before the action and sets it to after the action. The Do method must also do the same but for the 3D scene. The Undo is just the Do but in reverse, I did consider forgoing the Undo and instead when the user wanted to go to a previous step, I’d reset the algorithm to its initial state and “Do” all the steps up until theirs. I opted in the end not to do this, because if the algorithm was hundreds of steps long, and the user dragged the step progress bar backwards, we’d see order n! complexity (exclamation mark not for dramatic effect but rather mathematical). The Do method also returns an enumerable, this is used to decide program flow live when building the algorithm, such as whether or not to enter an if statement, it is essentially just passing forward the result of the action it is emulating, or no result if the emulated action doesn’t have a result.

An example of a constructor can be seen in Figure 24, an example of a pair of Do and Undo in Figure 24Figure 25. All the actions follow this pattern, there are numerous as defined in the requirements and sharing them all would be irresponsible. The only notable difference is that many actions store multiple paths into one timeline, when this happens, they occur in sequence.



Figure , example of "swap" action's constructor



Figure , example of the "swap" action's Do and Undo functions

### Timelines engine

The timelines engine is a very small engine, consisting of a single method, shown in Figure 26, and a single member variable.

The method works by finding the path that is occurring at the given progress, for example if you had three paths, one lasting 2 seconds, the next lasting 1 second, and the last lasting 4 seconds, if the progress was 2.5 seconds then you’d be in the second path because the second path starts at 2 seconds and lasts 1 second (ending at 3 seconds), 2.5 is between 2 and 3.

Then we repeat this practice of passing the baton, this time to the path, by passing the progress relative to the start of the path, continuing the above example, 0.5 in that case.



Figure , timeline engine's set progress code

### Paths engine

There are two paths, but abstraction to facilitate the creation of more. The paths themselves represent a path shape, but they also store an enumerable used to decide what easing function to use, easing is explained in the glossary.

The linear path is quite simple, it calculates what fraction of time through the path’s progress the absolute path progress is, applies the desired easing function to mimic mass of the object, then applies a lerp[23] between the start and end matrices.

The semi-circle path is shown in Figure 27, it is less straightforward. It involves doing the same as the linear path up to after the easing function is applied. Then we take the diameter that the imaginary circle both the start and end matrices share. This diameter is directional such that it is pointing towards the start matrix, which is crucial later.

Then we set the matrix reference to the midpoint, this is done as a separate step to reduce complexity, it’d also be fine to just translate it to the centre as part of the next step along with the other transformations.

Finally, we rotate the model the angle we want it to be at in the arc of the semi-circle, done in radians, then we translate it to the edge of the semicircle, because the diameter is also directional this means that when the angle is low, like at the start, the translation will result in it being near the start. Then we rotate it back, if we didn’t do this then the cubes would appear to rotate themselves, not simply move in an arc. And example of how it looks with and without the end rotation can be seen in Figure 28 and Figure 29 respectively.



Figure , semi-circle path code

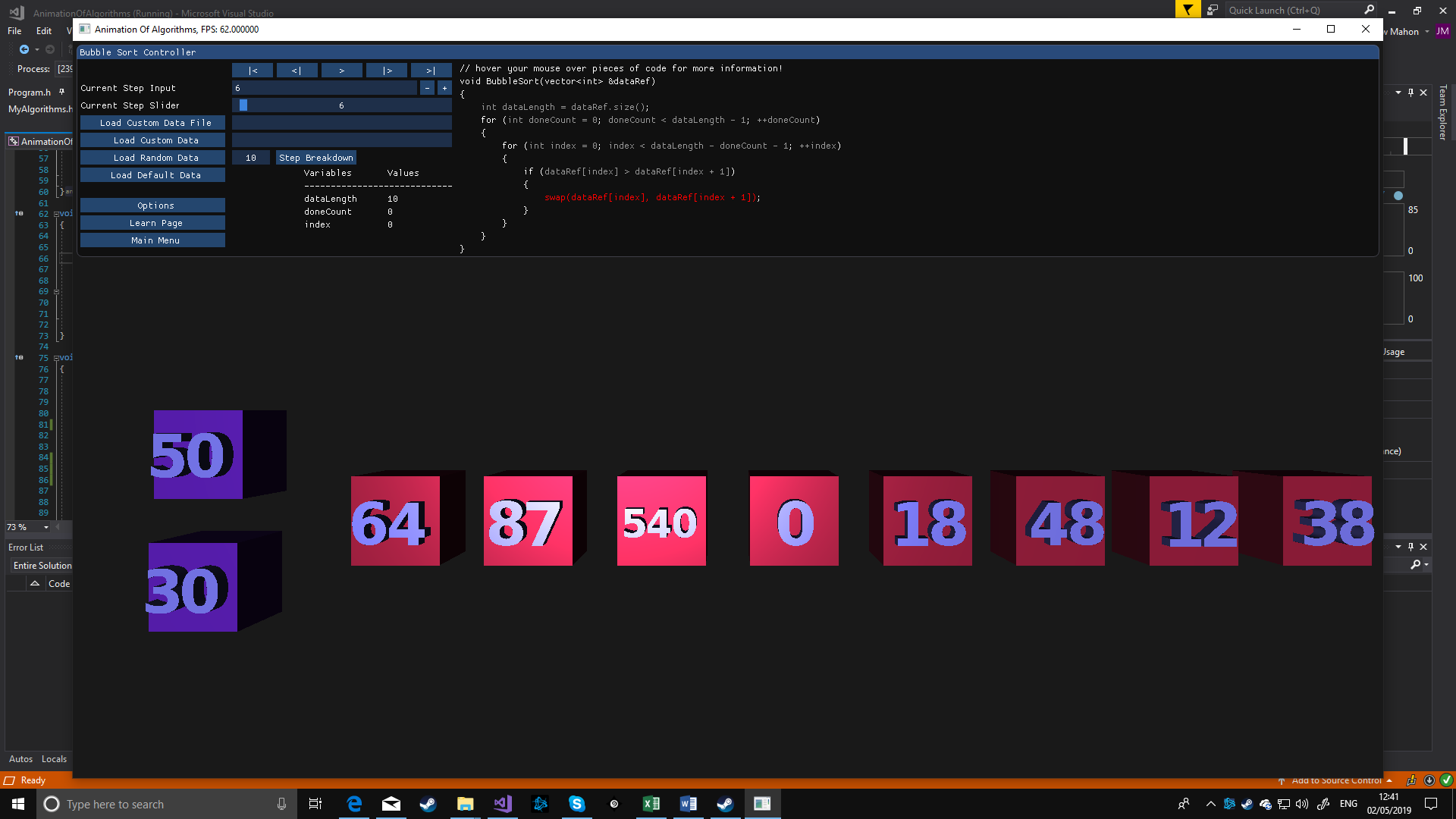


Figure , swap with reverse rotation

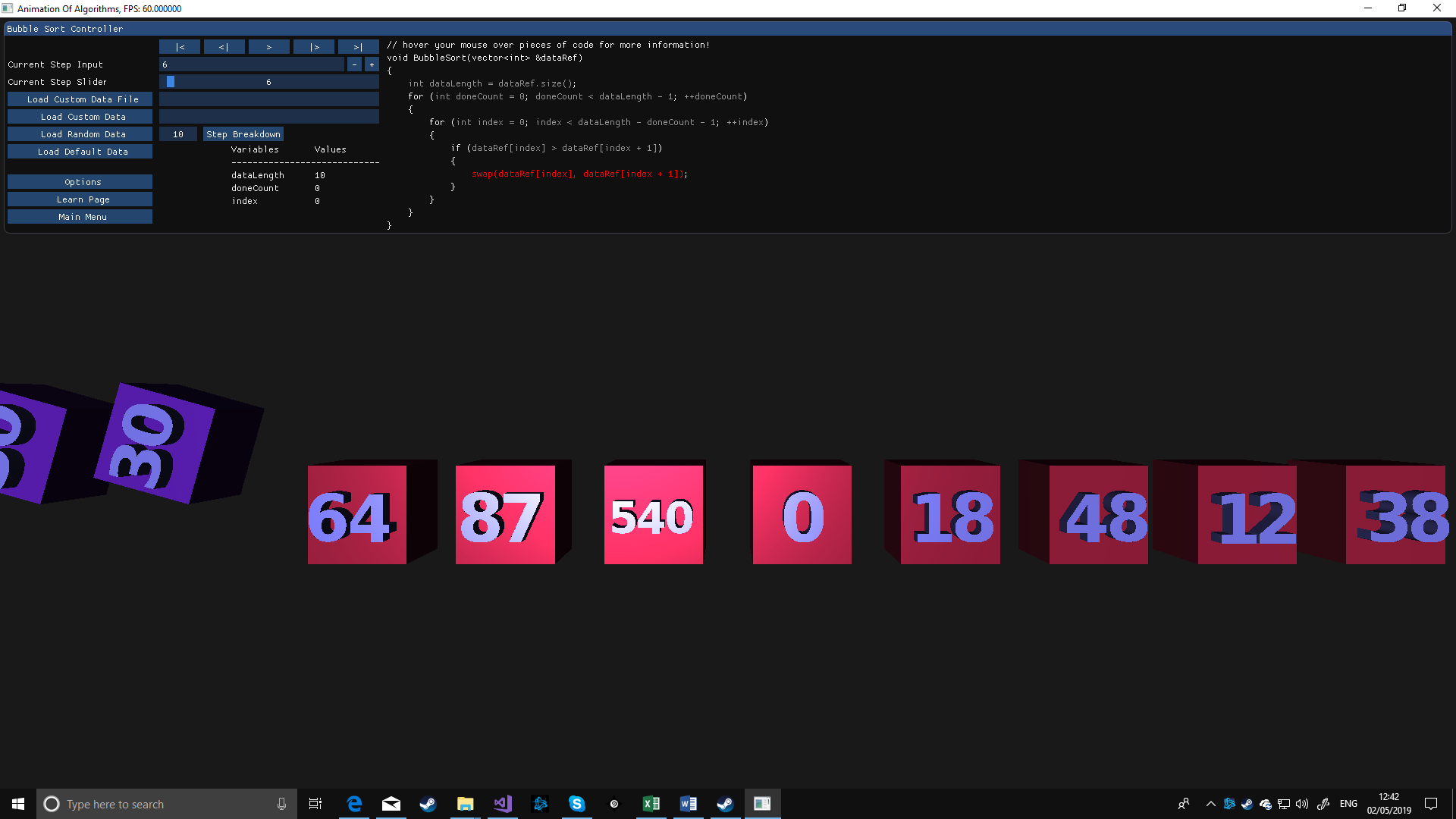


Figure , swap without reverse rotation

## Miscellaneous

In this section I will cover any implementation that is not in one of the above categories, mostly consisting of “helper methods” or similar.

### Settings singleton

Global variables are frowned upon, but sometimes they are the lesser of many evils, when that is the case it is recommended to use a singleton design pattern. As I have done to store settings of the application. This works by having a static function in a class with a private constructor, the static function returns a statically instantiated function variable of an instance of the class, since it is static it is only instantiated once and every following time it just returns the same one, and since we are returning a reference to it, not a copy, any changes made persist. The result is I can call this static function anywhere in the solution (that I include the header in) and make these changes to global settings in a manageable way.

#### Saving and loading

In order to persist settings between executions there are functions to save and load them to a file, these are of my own design (though I am aware libraries exist that can do this).

Figure 30 shows a version of the writeFile function with all but two settings omitted to save space. As you can see it simply writes to the file each member variable in the format memberVariableName=memberVariableValue, each on a new line.

When we wish to load the settings there are two stages, first to parse the file into a map of string keys and string values. Then to cycle through that map and take each value string and set the associated member variable from it. The code to do the prior is shown in Figure 31, and the code to do the latter is shown in Figure 32 (with all but two variables omitted to save space).



Figure , code to write two member variables into a file like a map



Figure , code to read a map created like in Figure 30



Figure , code to apply settings from the map read from the file

#### Creating a “text box”

To help third-parties do more interesting things I have made a general function to set up a cuboid with text written on it in 3D. It was one of the more interesting and challenging functions. The code for it can be seen in “AddTextBox” in GraphicsHelpers.cpp of the software solution (too large to have here even in the appendix).

The way it works is by calculating the width and height the text would normally occupy, then finding the necessary scaling to fit within the space they are designated to fit within.

Then each character is positioned and scaled in a loop and affixed to the cuboid.

Sounds easy, but when you have to account for the different sizes of each character and the fact that letters are aligned at their highest point and the numbers are aligned at their lowest point, things get more complicated.

# Testing

This section aims to present the testing done to facilitate robustness of the project. Every single requirement that is met has a least one test to verify that it has been met.

Many of the features are heavily dependent on the GUI, and ImGui has no unit testing framework[24], therefore much of the testing was always going to be manual. However sadly I was unable to get a test project to compile due to some linker issues I was not able to resolve in time. This is unfortunate because I wrote unit tests professionally for 14 months (I’m confident in my unit test writing skills, at least relative to what’s expected of a third-year computer science student).

Unit tests would have been ideal for testing the algorithms’ and actions’ correctness not only due to their speed, but their reliability as well.

To try and overcome this, I’ve tried to supplement robustness by having more tests, there are over 110 manual tests overall.

All tests must be run on a 64-bit windows machine with OpenGL 3.3 compatible drivers installed.

All tests must pass, and as of submission of the software system: do pass.

## Requirement tests

These are tabulated into the same groups as the requirements are in.

Table 1 shows manual tests for user interface requirements.

Table 2 shows manual tests for 3D rendering requirements.

Table 3 shows manual tests for animated algorithm requirements.

## Stress tests

Table 4 shows tests to verify that the program can handle taxing/extreme situations. Such as extremely long data, or long periods of being open and animating.

# Critical Appraisal

In this section I will evaluate the resultant software system against what my aims and objectives were. The impact my software system could have in the real world. And the personal development that I have undergone during the implementation of this project.

## Critical analysis

In this section I will analyse how well my software system achieves the aims and objectives laid out in the introduction, and therefore how well it over comes the challenges laid out in the introduction.

My aims were broken into 4 challenges, which is what I will be focusing on in the four following sections, as overcoming those challenges is not only indicative that I’ve overcome the aims, but also are easier to concretely evaluate.

To summarise, I think my primary aim was satisfied well, but the secondary supporting aim of third-party support was severely lacking, which in the real-world context would make this project significantly less useful than in the isolation of the classroom in which it shines.

### An intuitive GUI

I set out to create an intuitive GUI so the user doesn’t spend time figuring out the GUI, which in turn would impair learning because more time spent on the GUI is less time spent on the algorithms.

To summarise I think I overcame this challenge competently, with some room for improvement.

My justification is subjective, as I didn’t amass a formal user study to confirm user feedback on a large scale. I can say however that I am confident that I have provided enough tool tips and descriptive labels, verified by testing, to make understanding the GUI easy. It’s anecdotal, but my novice computer literate parent was able to use the program with no guidance.

I especially liked the code display, and how the executing code becomes highlighted, a feature that was simple and quick to implement but provided significant understanding of what was going on.

The step control was also a nice feature and is what makes the actual interactive algorithm intuitive to control, it is the most important part of the GUI to be intuitive and I was able to make it intuitive. Though I cannot take all the credit, the step control is only so intuitive because of the widespread use of the remote controls.

In hind sight I would have liked audio for each button press, just as feedback that the press occurred, and to perhaps reduce monotony when using the program for extended periods of time.

### Interactivity

I set out to make the program interactive, especially the algorithms. While hardcoded “videos” may have helped people understand the algorithms better, being interactive is far better as studies show.[8][9]

To summarise I think I overcame this challenge as well as could be expected, with some room for improvement.

#### Step control

As the requirements laid out, and the tests verify, you can play animations forward continuously, or just for a single step, pause animations, you can return to the previous step, and you can jump to the end and start of the algorithm. All the above done with just one click each.

The combination of the above gives the user control, they are given the tools to interact, this allows them to explore. If for example they misunderstood a step, they have the tools to quickly return to it, and play it individually, and repeat this process as many times as they would like until they do understand. The same could be said for a video, but with a video it is harder to pinpoint and return to a specific part.

In hind sight I’d like to have added the ability to play in reverse, as well as play a single step in reverse. The framework allows this to be added without breaking anything, nor would it require significant work, but it was low on my list of priorities as the deadline approached and thus didn’t make the cut.

I’d also would have liked to add the ability to jump back 5 steps, through usage I found that it wasn’t uncommon for me to want to see a step again but by the time I get to the “go back a step” button the step is already a couple later than when I made the choice to go back a step, this combined with the fact that context is useful meant jumping back several steps may have been useful. I opted not to use this feature in the final software solution as I couldn’t come up with a way to incorporate it without cluttering the already dense step control.

#### User data input

As the requirements laid out, and the tests verify, you can input custom data, either directly, or by loading a file.

This gives the user agency; they can satisfy their curiosity and approach their “what if” moments head on. Rather than having to go to their browser and search what happens when the starting list is in the correct order, or all the same, or in reverse order, they can try themselves.

The ability to load from a file specifically means that the user can quickly try the same data on each of the algorithms if they are not satisfied with the default data.

The ability to load random data allows the user to relax this agency when they are satisfied with their questions, or worried that they can’t input data without bias (which in turn might bias the algorithm’s performance), the ability to configure the length of the random data allows them to view things in a more compact manner, especially useful for recursive algorithms.

In hindsight I would have liked to add the ability to configure the random number generator, giving the user the ability to decide if the data should start sorted, sorted in reverse, nearly sorted, nearly sorted in reverse, half sorted/unsorted, etc. This would prompt the user to explore scenarios they haven’t even conceived of, but without taking away their interactivity.

#### Algorithm configuration

In hindsight I would have liked to allow the user to decide whether the algorithm should sort in ascending or descending order, as well as sort elements of different types such as decimals and strings. These would be on my list of top priorities if I were to continue the project, I see it as a great short coming that they didn’t make it into the final software solution.

### Understanding

I set out to help users understand algorithms, this is and always was the primary aim, all the challenges stem from this in some way or another so this section contains the most significant critique of the project successes and failures.

To summarise I think I overcame this challenge for the three implemented algorithms but failed to overcome this challenge as much as I would have liked for the concept of algorithms in general. In layman’s terms, I believed I opted to focus too much on quality over quantity (though ultimately this is subjective).

#### 3D rendering

As stated in the introduction, teaching using 3D resources has been shown to aid learning in this[10] and other[11] contexts. Though the 3D used in this project is basic, we can only speculate as to whether that is satisfactory as there are no (meta) studies reviewing whether the quality of the 3D has significant impact on the learning that I could find.

In hindsight I would have liked to use a directional light instead of point light, add shadows, textures, and use different materials, to make the models appear more lifelike (which may a contributor to the success of using 3D resources). However, I opted not to do this because I was focused on other features that were more important, even if I had more time there are countless features I’d rather spend time implementing before this as I have no evidence to believe that more “beautiful” 3D will significantly enhance user learning.

#### Animation

The project is titled ANIMATED Algorithms, so it is no surprise that one of the challenges was to animate the algorithms.

To summarise I think I overcame this challenge competently with room for improvement.

As the requirements laid out, and the tests verify, each action upon the data has an animation. It is subjective, but I think these animations accurately represent what is happening even to someone who doesn’t understand the algorithms.

The animations are good because they incorporate acceleration and deceleration, this is referred to as “Slow in” and “Slow out”, and together are one of the 12 core principles of animation. Occasionally, when appropriate I even incorporated “anticipation”, another of the core principles of animation, an example of this can be found with swaps, which before swapping have a small “wind up” by moving the opposite direction just before swapping.

*As action starts, we have more drawings near the starting pose, one or two in the middle, and more drawings near the next pose. Fewer drawings make the action faster and more drawings make the action slower. Slow-ins and slow-outs soften the action, making it more life-like.*[25]

I am also glad I implemented the ability to let the user adjust the animation speed, as when surveyed on a similar application, users requested this feature in the feedback[18].

In hindsight I would have liked to incorporate more principles of animation, such as squash and stretch.

*[Squash and Stretch] gives the illusion of weight and volume to a character as it moves.*[25]

One possible way to do this would be to have the cubes stretch as they accelerate, and squash as they decelerate, at least for the vertical actions. Though I imagine there is a point beyond where incorporating more principles of animation no longer serves to reduce tedium, but instead distracts the user, I suspect am not yet that that point as the current effects are fairly subtle, but I must stay wary of this phenomenon if I were to develop the animation further.

#### Colours

I am happy with the results of how things are coloured.

The currently of interest cube is always the given highlighted colour, the sorted cubes are always a sorted colour, and cubes of different groups are properly indicated via their colours.

When surveyed on a similar application, users requested this feature in the feedback[18] so the inclusion of it in my project is indicative of addressing user wants and needs.

In hindsight the code is not properly implemented for colours, there is a failure to maintain a separation of concerns between the algorithms and actions engines. I discuss this problem in detail in the third-party extension section of this critical appraisal.

#### Live code UI

This is possibly my favourite feature; I am confident that this feature significantly contributes to user understanding by providing context to the animation they are watching.

In hindsight there is nothing I’d change about this, I’m very satisfied with it.

#### Algorithms

I planned to do five algorithms but, due to time constraints, in the end I did three. Adding more algorithms on their own is not time consuming due to the success of my framework, but adding GUI live code, tests, running the tests, writing an introduction, etc. all take time.

In the end I opted to focus on polishing the system and framework that supported these existing algorithms rather than adding more. As an academic exercise I didn’t think adding more algorithms was of much merit due to the straight forward nature. It is essentially not much harder than data entry for someone who has done it three times prior as I had.

### Third-party extension

This section is like the understanding section in that it ties in heavily with the other challenges. However, where the understanding section is almost like a superset to the other challenges this section is instead a measure of the extendibility of the other challenges, so that others do not have to tackle those challenges from the start as I mostly did.

In summary I think I did a poor job of overcoming this challenge. For example, third parties can add their own algorithms without the source code, but it requires reimplementing significant parts of the user interface, and there is much more room for improvement (described below).

It was not a complete failure though, there is still plenty of reusability, especially with the actions and algorithms engines (and the 3D rendering base can be used by third-parties without issue).

In hindsight what I should have done ideally was to create a code parser, from which each algorithm, their live code, and GUI, is generated from given code. Then allow third-parties to submit their own code. However, that is vastly different from and harder to do than what I’ve done.

The following sections will instead analyse the flaws that I made with respect to the overall approach I took (of using abstraction, rather than the overall approach mentioned just now that I should have taken in the first place to best overcome this challenge).

#### GUI engine

Initially I’d never considered the third-party extension in terms of the GUI, I was instead focused on the algorithm class itself being extendable. The result of this is that the buttons to navigate to the various algorithm pages from the learn page are hard coded in which is a significant extendibility issue.

In hindsight what I should have done instead is allow the third-party to pass in pointers to the functions used to generate their pages and then the learn page would generate the GUI to navigate to them.

There are also issues with the file loading widgets of the GUI not being abstract enough, currently they always and only expect a list of integers. This should be fixed to allow third-party injection of data using their own custom entry field(s).

With proper abstraction and method abstraction, third-parties could build their GUI for their own algorithms using mostly methods from my project, however I failed to make this the case, which in turn contributed to my critical appraisal of the third-party extendibility aim being so negative.

#### Algorithms engine

In summary I did not properly abstract my algorithms engine, this would impede third-party use more than I deem acceptable.

There is an abstract algorithms class that does a lot of the heavy lifting, and this is useable by third parties to reduce the amount of work they need to do by a significant amount.

However, there is not even an interface, let alone implementations, for the abstract concepts of sorting algorithms and recursive algorithms. There were several cases of duplicate code that could have been avoided because I was in a rush to get things working.

In hindsight I should have properly put common code between the algorithms in abstract classes, all three algorithm implementations should have instead implemented a list sorting algorithm class (and recursive algorithm class in the case of merge sort and quick sort) instead of directly implementing the base algorithm class.

I also wish the user could sort anything that implements a less than operator.

In hindsight I would implement this by internally storing the values as integers representing the final order, the final order calculated using a sorting algorithm using templates. Provided there are models to represent them of course, so strings and decimals would be ok.

#### Actions engine

In summary I did not properly separate concerns when implementing the actions engine, this would impede third-party use more than I deem acceptable.

Each action handles both adjustments to data and animations, and sometimes colour adjustments.

This would be fine if those separate concerns had their own engines, where action could be built be passing in different objects that would each handle one of the above. However, this is not the case. Currently if you want to use the same data adjustment of an existing action, but change the animation, you must completely copy it then make the changes you want and have that as an entirely new action. This is less of an issue with the data manipulation and actions themselves because the animations are already quite good.

The issue arises when colours are involved, as I discovered mid project, changing the colours of cubes in the actions was not always the same for each algorithm that used that action. So, I resorted to writing code in the algorithms’ themselves to set the cube colours based on the current state. Apart from this being a violation of separations of concerns, it is also problematic as third-parties will have to work out all the cube colouring themselves when the framework should be doing it for them, or at least making it trivial for them to do.

So, in hindsight I would have much more atomic actions, and they could be grouped into another class that would replace the current position of action in the architecture as it currently is.

Also, I’d store data and the model that represents it in a tuple and store the vector of tuples rather than two separate vectors that I and any third-parties must constantly keep in sync.

### Software quality

As a percentile among my peers I think I did well in producing software of good quality, however, I can still see much room for improvement.

Due to how I highlight the currently executing code it was extremely convenient to just use magic numbers, in hindsight I would overcome this by at least storing the significant values in member constants, such as the point where each variable enters and leaves scope, etc.

There are numerous magic strings and formats, especially in the GUI engine, the fix for this is simple and I should of course do it, but its time-consuming nature had left me to prioritise more “flashy” features. In hindsight perhaps this was an error of judgement, it certainly would be in a professional setting.

## Real world context

I think the idea of a teaching tool that doesn’t require a teacher is valuable to society.

Teaching is a highly skilled profession and therefore requires significant investment, both in terms of money and time. This investment is obviously paid back by the generation of skilled individuals they produce, but it is a good opportunity, if one could supplement the learning of a teacher using automated tools such as mine, you could potentially reduce the number of teachers needed. By reducing the number of teachers needed, some people that would have become teachers will instead take on a different career, or we can have the same number of teachers and increase the average education of each student. Both scenarios are valuable, which is why automated learning tools such as mine are so valuable.

There is really no business application other than as possibly being sold commercially, though at the current quality I don’t think it is suitable over other paid alternatives.

The technologies I have used have been appropriate for use in society. Windows is the most common operating system in Europe[26], especially in the education sector, so naturally if I had to choose one platform for a teaching tool windows was the appropriate choice. OpenGL 3.3 was released in 2010[4] and all current graphic cards support it out of the box, in most cases they just need the appropriate drivers installed if a computer can’t run OpenGL 3.3.

## Personal development

One major flaw that I am sad that I was not able to overcome was my tendency to leave everything until the last minute.

In a professional environment I have no trouble focusing on work, I feel this is a biproduct of not wanting to be dishonest nor steal (not working while at work feels like both of those to me, even when it’s not my fault!). However, I cannot say this extends to a project like this where I am not being paid and I am the only victim of my lack of diligence, I was hoping I could learn to apply myself despite the different environment, but I was not successful. I’d be lying if I said things haven’t improved at all in this regard, but I am far from being able to say that it is a small issue.

However, you know what they say, you never lose, you only succeed or learn. And in this case what I learned was that perhaps, despite its overhead, AGILE for a personal project could be beneficial to my time management. Perhaps having laid out goals will help me curb my procrastination habit.

One positive example of growth is that I am far better versed in C++ than I was at the start of this project. Before now I’d only produced one C++ application, and it was all in one file despite being quite large, there were no classes, and it was riddled with other flaws. After the growth in my C++ skills it causes me distress to look at my code from just 1 project ago! Which I am taking as a good sign.

Another positive example of growth is that mid-way through the project when I was struggling with separating concerns and solving some problems, I had an epiphany. The epiphany was that EVERY problem that wasn’t trivial was made up of smaller problems. And to solve the larger problem I just needed to work out what smaller problems I needed to solve. Then that I should program based on the assumption that they had been solved, then go solve them, performing these steps recursively.[27]

This is akin to the principles of recursion, just in this case the smaller problems are not the same problem as the containing problem.

I feel like this epiphany has helped me grow as a programmer more any other individual event in my life, it’s a shame it couldn’t have happened at the start of the project instead, perhaps I would have done a better job with my abstraction if it had!

As a past and future professional programmer, I regard both these progressions of my abilities as useful. C++ is used on some embedded systems, which my past and future employer make frequently, so the utility of this is obvious. I also create video games as a hobby, and C++, especially 3D in C++, is useful for that as well as most modern games are in 3D.

And the epiphany I had is useful no matter where I work, less so the more theoretical the job I imagine, but still always useful.

# Conclusion

I have successfully created a windows application that in 3D visualises three common or famous sorting algorithms: bubble sort, merge sort, quick sort. It conveys them in a far more understandable way than a textbook or other static media might, and in a moderately more understandable way than a video might.

I have incorporated features that prior work has found to be in demand or useful, in a unique combination, as well as additional features that according to animation and education theory will improve the ability of my program to educate.

I failed to address third-party extension to the degree I set out to but did provide support that allowed creation of new animations without third-parties having to reimplement “boiler plate” code for 3D rendering and playback management. These were significant tasks and the fact that I allow third-parties to use these features will save them time, possibly in the order of weeks, as these features in total accounted for most of the development time.

The quality of the user interface is high enough to be used in a professional setting.

Overall the above remarks let me conclude that this project was successful but would benefit from future work to improve the third-party extendibility. The most ideal outcome being that just by supplying C++ code, or pseudo-code, that the third-party could have the application produce an animated algorithm, complete with GUI, for it; without relying on a human at all.

# Glossary

* (Algorithm) state: e.g. data contents, child branch(es) state(s), values for variables, etc. are all examples of things that together can define the state of an algorithm.
* Action: a synonym of step, not shown to the user but used in a programming context **because in that context it’s more appropriate to what they are, whereas to a user I think step is more intuitive**.
* An animation (of a step/action): the visual representation of the transition from one state to the state that will be once the given step/action is executed.
* An animation (of an algorithm): the union of processing input, rendering it visually, and generating all steps and animations between those steps, using the algorithm of the same name.
* Easing function: will convert a float between 0 and 1 into another value such that a linear rate of change may appear different. The easing functions used were from nicolausYes’s GitHub repository of easing functions[28].
* Engine: word used to refer to a system or class(es) that “powers” the accomplishment of a goal or goals.[29]
* Introduction (to an algorithm): a section of text that explains in a more abstract way how an algorithm works, **not for the purpose of the user immediately understanding, but so that they can “take in” the interactive algorithm better because they have some idea of what to expect**.
* Modal: a window that pops up and prevents interaction with the source window until the modal is closed.
* Model: used interchangeably with what is referred to in the code as a “RenderNode”, a “RenderNode” renders an actual model using a shader and configuration.
* Model matrix: a matrix used to store the translation, scale, rotation, etc. of a model. It is multiplied with the view and model matrix in the vertex shader to calculate the position of each vertex in screen space.
* Page: a distinct state of the program, within a specific Dear ImGui window appears.
* Progress (through an action/timeline): the number of seconds through the duration of the given action/timeline.
* Projection matrix: a matrix used to store the desired field of view, the aspect ratio of the viewport, as well as the range of legal distance a pixel can be from the camera before it is not rendered.
* Step breakdown: e.g. 50 steps may have the composition of 20 comparisons, 25 swaps, and five variable assignments.
* Step control: the ability to set the step to any step from any step, animate until the next step, or animate until the end of the steps, pause any current animation, set the step to the first step, and set the step to after the last action.
* Step explanation: e.g. rather than just being categorised as a comparison, instead also including why there’s a comparison (what purpose it serves in the context of this algorithm).
* Step: an individual discrete part of the algorithm. E.g. comparing two values, swapping two values, removing the head of a list, arithmetic operations, indices, co/sine and tangent.
* Tree of recursion: e.g. if you merge sort 16 values then you will first enter the left branch of the tree of recursion with the 8 lowest index values, then the left branch of the left branch with the 4 lowest index values, etc.
* View matrix: a matrix used to store the position to view the 3D scene from and the position to look to, as well as the direction to be considered as “up” when rendering.

# Bibliography

1.

Koedinger K, Kim J, Jia J, McLaughlin E, Bier N. Learning is Not a Spectator Sport. In ACM; 2015. p. 111–20. (L@S ’15). Available from: http://dl.acm.org/citation.cfm?id=2724681

2.

udiprod. Merge Sort vs Quick Sort [Internet]. 2014. Available from: https://www.youtube.com/watch?v=es2T6KY45cA

3.

DeVries J. Learn OpenGL [Internet]. 2014. Available from: https://learnopengl.com/

4.

Meguid EA, Collins M. Students’ perceptions of lecturing approaches: traditional versus interactive teaching. 2017 Jan 1;8:229.

5.

Barnes D. Breaking Down The Problem [Internet]. 1996. Available from: https://www.cs.kent.ac.uk/people/staff/djb/ProblemSolving/doc1.n-S-3.html

6.

Zhu X 1, Lee Y, Simon HA, Zhu D. Cue recognition and cue elaboration in learning from examples [Internet]. Vol. 93. 1996. Available from: https://www.ncbi.nlm.nih.gov/pubmed/8577767

7.

Atkinson D, Thomas F, Johnston O. 12 principles of animation [Internet]. 1930. Available from: http://minyos.its.rmit.edu.au/aim/a\_notes/anim\_principles.html

8.

nicolausYes. Easing Functions [Internet]. Available from: https://github.com/nicolausYes/easing-functions/blob/master/src/easing.cpp

9.

6brueder. Alphabet, Numbers, and Common Symbols Models [Internet]. 2011. Available from: https://www.thingiverse.com/thing:15198

10.

Wikipedia. Lerp [Internet]. Available from: https://en.wikipedia.org/wiki/Linear\_interpolation

11.

GLAD [Internet]. Available from: https://glad.dav1d.de/

12.

Brown M, Najork M. Algorithm animation using 3D interactive graphics. In ACM; 1993. p. 93–100. (UIST ’93). Available from: http://dl.acm.org/citation.cfm?id=168651

13.

Wikipedia. OpenGL [Internet]. Available from: https://en.wikipedia.org/wiki/OpenGL

14.

Halim S. VisuAlgo [Internet]. 2011. Available from: https://visualgo.net/bn/sorting

15.

OrehovaCki T. Visualization of Sorting Algorithms [Internet]. 2016. (Fundamentals of Computer Graphics). Available from: https://search.credoreference.com/content/entry/crcfcg/visualization/0

16.

Wikipedia. GLFW [Internet]. Available from: https://en.wikipedia.org/wiki/GLFW

17.

TutorialsPoint. Command Pattern [Internet]. Available from: https://www.tutorialspoint.com/design\_pattern/command\_pattern.htm

18.

Wikipedia. Computer Animation [Internet]. Available from: https://en.wikipedia.org/wiki/Computer\_animation

19.

Ita\_c, Mayank Khanna 2. Merge Sort [Internet]. Available from: https://www.geeksforgeeks.org/merge-sort/

20.

ChitraNayal. Bubble Sort [Internet]. Available from: https://www.geeksforgeeks.org/bubble-sort/

21.

ocornut. Dear ImGui [Internet]. Available from: https://github.com/ocornut/imgui

22.

Faria B. Visualizing Sorting Algorithms [Internet]. 2017 p. 222. (Peterson’sÂ® Graduate Programs in Business, Education, Information Studies, Law & Social Work 2018). Available from: https://digitalcommons.ric.edu/cgi/viewcontent.cgi?article=1129&context=honors\_projects

23.

Market Share of Operating Systems in Europe [Internet]. Available from: http://gs.statcounter.com/os-market-share/all/europe

24.

BBC, Bamford A. 3D learning tools positive for pupils [Internet]. 2011. Available from: https://www.bbc.co.uk/news/technology-15115059

25.

Palak Jain 5, UditChaudhary. Quick Sort [Internet]. Available from: https://www.geeksforgeeks.org/quick-sort/

26.

Wikipedia. Software Engine [Internet]. Available from: https://en.wikipedia.org/wiki/Software\_engine

27.

Carson E, Parberry I, Jensen B. Algorithm explorer: Visualizing Algorithms in a  3D Multimedia  Environment. 2007 Mar 7;39(1):155. Available from: https://www.researchgate.net/publication/221537475\_Algorithm\_explorer\_Visualizing\_algorithms\_in\_a\_3D\_multimedia\_environment

28.

Bamford A. Gaia 3D White Paper [Internet]. Available from: https://www.gaia3d.co.uk/case-studies/the-3d-in-education-white-paper/

29.

ocornut. ImGui testing framework isn’t mature [Internet]. Available from: https://github.com/ocornut/imgui/issues/435#issue-121861448

# Appendix

In this section are all the figures, tables, etc., that are too large to organically include within the body of the other sections without impairing the flow of the text.

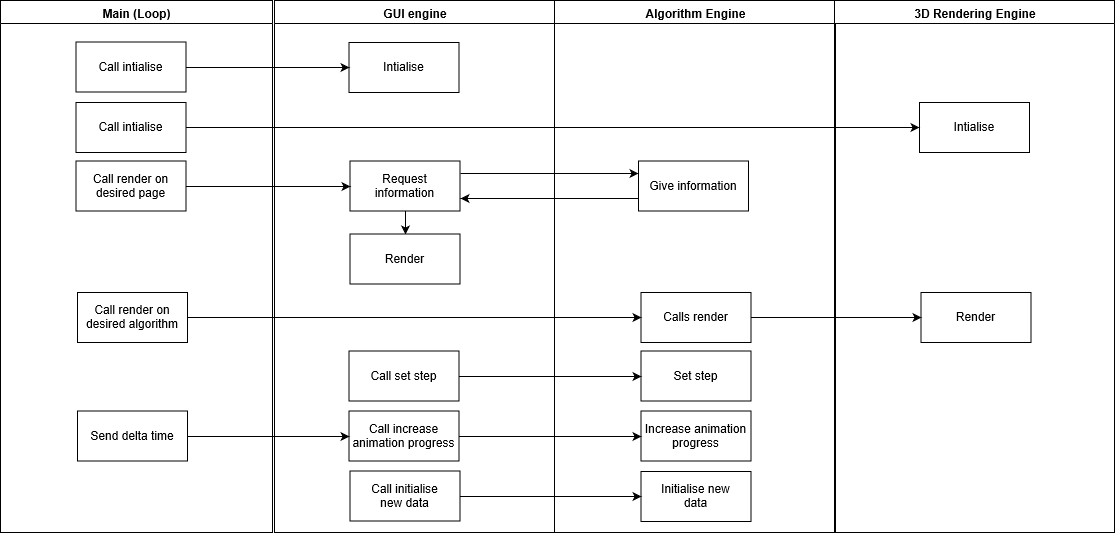


Figure , high-level architecture of intercommunication between handlers of separate concerns

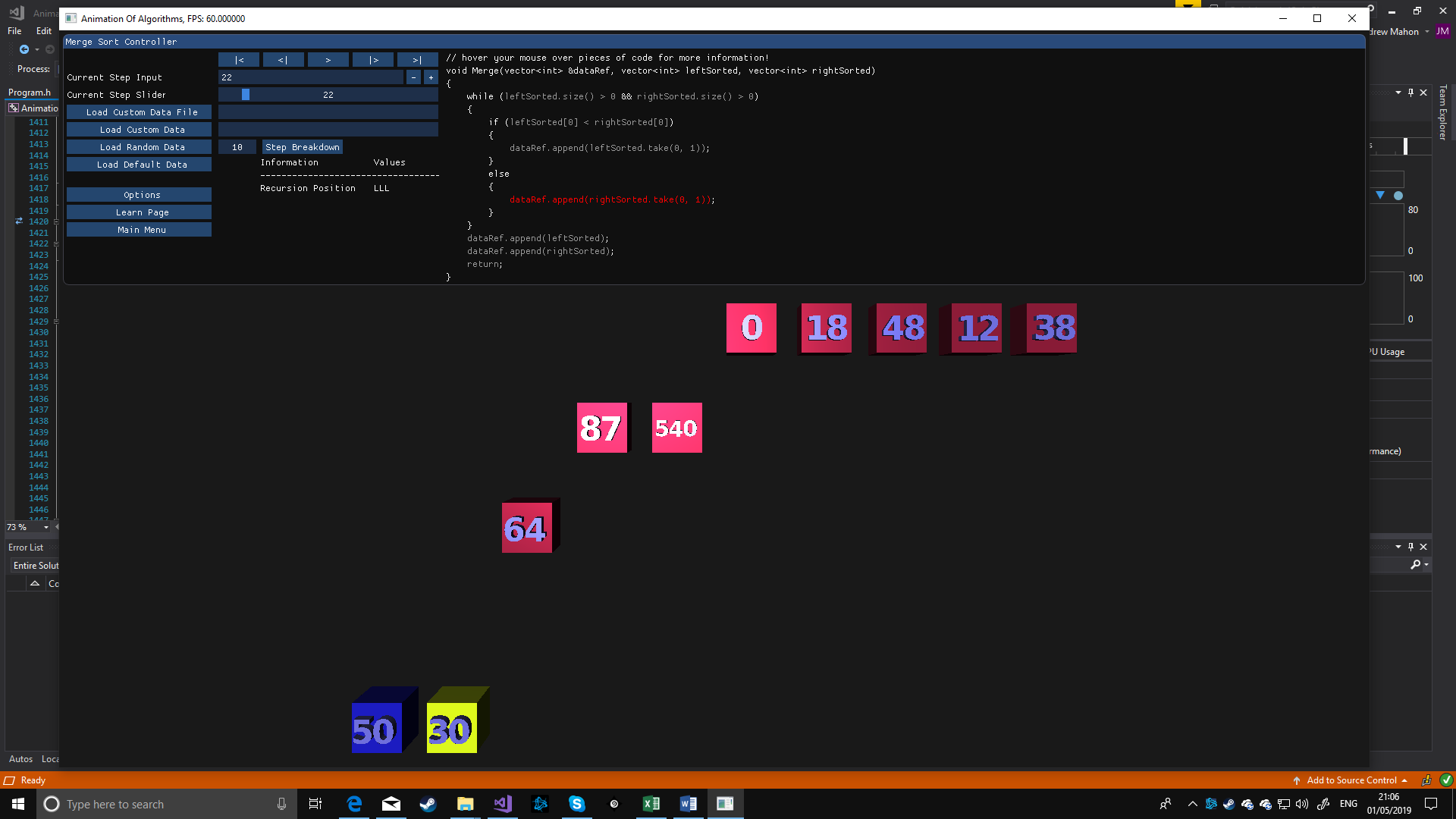


Figure , example 1 of automatic camera positioning: typical window

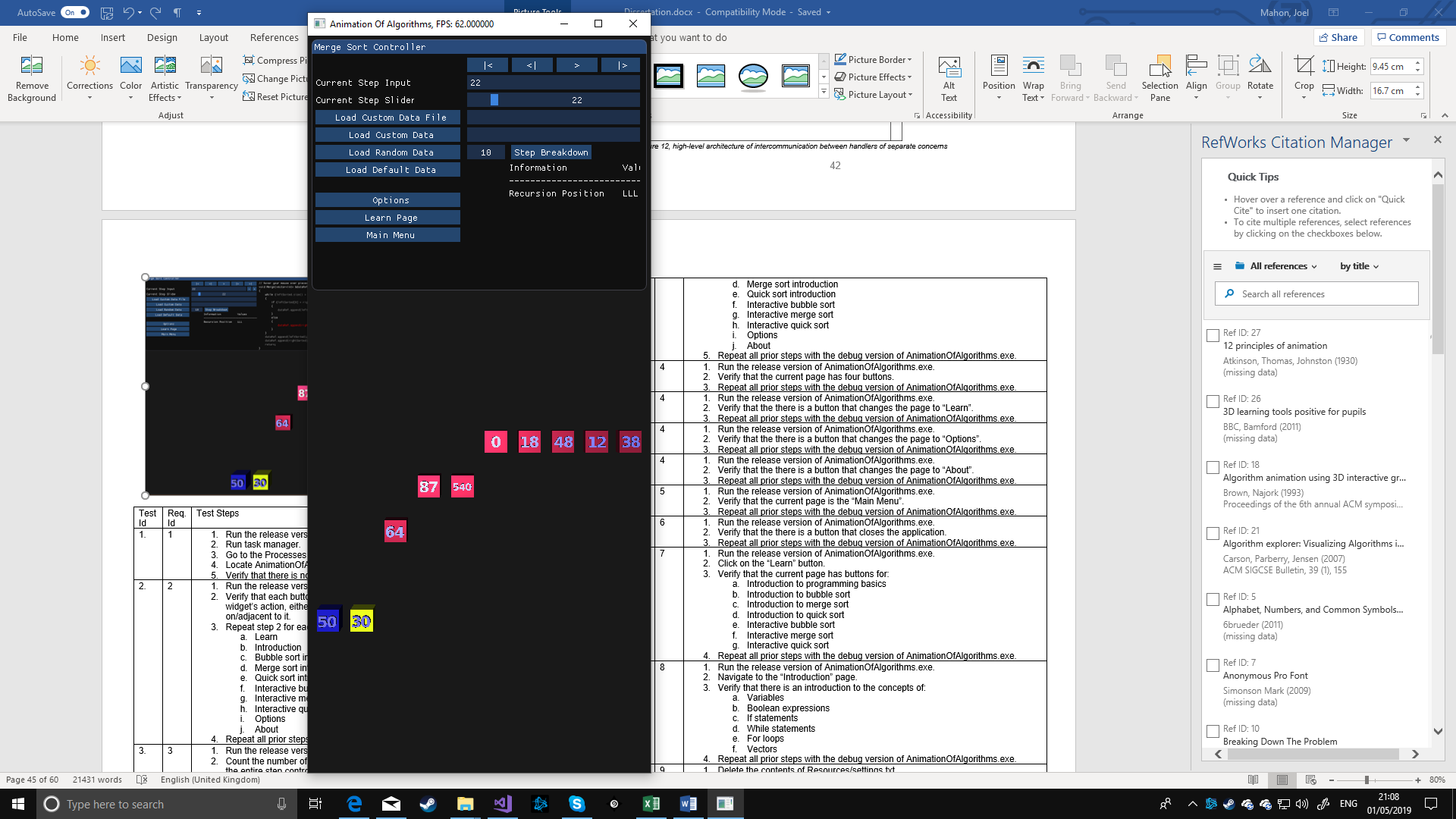


Figure , example 2 of automatic camera positioning: narrow window

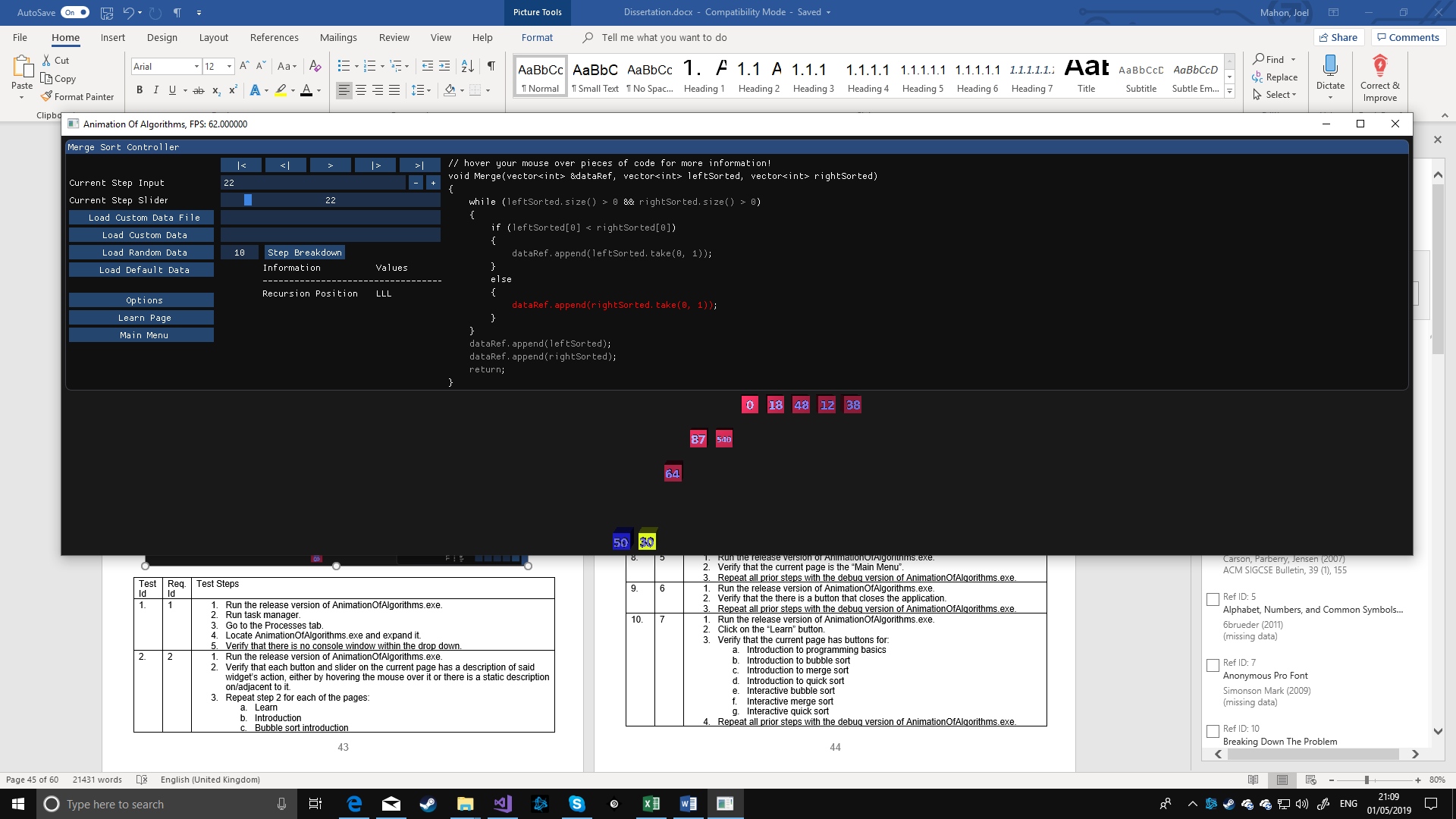


Figure , example 3 of automatic camera positioning: wide window



Figure 37, vertex shader code



Figure 38, fragment shader code

|  |  |  |
| --- | --- | --- |
| Test Id | Req. Id | Test Steps |
|  | 1 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Run task manager. 3. Go to the Processes tab. 4. Locate AnimationOfAlgorithms.exe and expand it. 5. Verify that there is no console window within the drop down. |
|  | 2 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Verify that each button and slider on the current page has a description of said widget’s action, either by hovering the mouse over it or there is a static description on/adjacent to it. 3. Repeat step 2 for each of the pages:    1. Learn    2. Introduction    3. Bubble sort introduction    4. Merge sort introduction    5. Quick sort introduction    6. Interactive bubble sort    7. Interactive merge sort    8. Interactive quick sort    9. Options    10. About 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 3 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Count the number of interactive GUI elements on the current page, only counting the entire step controller (5 playback buttons, input integer, and integer slider) as 1. 3. Verify that the count to be less than or equal to 10. 4. Repeat steps 2 to 3 for each of the pages:    1. Learn    2. Introduction    3. Bubble sort introduction    4. Merge sort introduction    5. Quick sort introduction    6. Interactive bubble sort    7. Interactive merge sort    8. Interactive quick sort    9. Options    10. About 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 4 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Verify that the current page has four buttons. 3. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 4 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Verify that the there is a button that changes the page to “Learn”. 3. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 4 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Verify that the there is a button that changes the page to “Options”. 3. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 4 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Verify that the there is a button that changes the page to “About”. 3. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 5 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Verify that the current page is the “Main Menu”. 3. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 6 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Verify that the there is a button that closes the application. 3. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 7 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Click on the “Learn” button. 3. Verify that the current page has buttons for:    1. Introduction to programming basics    2. Introduction to bubble sort    3. Introduction to merge sort    4. Introduction to quick sort    5. Interactive bubble sort    6. Interactive merge sort    7. Interactive quick sort 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 8 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Introduction” page. 3. Verify that there is an introduction to the concepts of:    1. Variables    2. Boolean expressions    3. If statements    4. While statements    5. For loops    6. Vectors 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 9 | 1. Delete the contents of Resources/settings.txt 2. Run the release version of AnimationOfAlgorithms.exe. 3. Navigate to the “Learn” page. 4. Verify that nothing happens when clicking each of these buttons:    1. Introduction to bubble sort    2. Introduction to merge sort    3. Introduction to quick sort    4. Interactive bubble sort    5. Interactive merge sort    6. Interactive quick sort 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 9 | 1. Delete the contents of Resources/settings.txt 2. Run the release version of AnimationOfAlgorithms.exe. 3. Navigate to the “Learn” page. 4. Verify that a page change happens when clicking each of these buttons:    1. Introduction to bubble sort    2. Introduction to merge sort    3. Introduction to quick sort 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 10 | 1. Delete the contents of Resources/settings.txt 2. Run the release version of AnimationOfAlgorithms.exe. 3. Navigate to the “Introduction” page. 4. Navigate to the “Learn” page. 5. Verify that nothing happens when clicking each of these buttons:    1. Interactive bubble sort    2. Interactive merge sort    3. Interactive quick sort 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 10 | 1. Delete the contents of Resources/settings.txt 2. Run the release version of AnimationOfAlgorithms.exe. 3. Navigate to the “Introduction” page. 4. Navigate to the “Bubble Sort Introduction” page. 5. Navigate to the “Learn” page. 6. Verify that nothing happens when clicking each of these buttons:    1. Interactive merge sort    2. Interactive quick sort 7. Verify that clicking the Interactive bubble sort button navigates to the “Bubble Sort” page 8. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 10 | 1. Delete the contents of Resources/settings.txt 2. Run the release version of AnimationOfAlgorithms.exe. 3. Navigate to the “Introduction” page. 4. Navigate to the “Merge Sort Introduction” page. 5. Navigate to the “Learn” page. 6. Verify that nothing happens when clicking each of these buttons:    1. Interactive bubble sort    2. Interactive quick sort 7. Verify that clicking the interactive merge sort button navigates to the “Merge Sort” page 8. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 10 | 1. Delete the contents of Resources/settings.txt 2. Run the release version of AnimationOfAlgorithms.exe. 3. Navigate to the “Introduction” page. 4. Navigate to the “Merge Sort Introduction” page. 5. Navigate to the “Learn” page. 6. Verify that nothing happens when clicking each of these buttons:    1. Interactive bubble sort    2. Interactive merge sort 7. Verify that clicking the interactive quick sort button navigates to the “Quick Sort” page 8. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 11 | 1. For each of test 12 to 17 inclusive close the program and open it again without deleting the contents of any files before performing the verifications. |
|  | 12 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the following pages and perform step 3 for each:    1. Learn    2. Introduction    3. Bubble sort introduction    4. Merge sort introduction    5. Quick sort introduction    6. Interactive bubble sort    7. Interactive merge sort    8. Interactive quick sort    9. Options    10. About 3. Verify that the page has a button to navigate to the previous page. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 13 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Verify that the page has code for bubble sort visible. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 13 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort page”. 3. Verify that the page has code for merge sort visible. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 13 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Quick sort page”. 3. Verify that the page has code for quick sort visible. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 13 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort page”. 3. Load default data. 4. Set step to 19 5. Verify that the page has code for merge visible. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 13 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Quick sort page”. 3. Load default data. 4. Set step to 2 5. Verify that the page has code for partition visible. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 15 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Verify that the first to be execute code statement is highlighted. 5. Increment step. 6. Verify the code statement that was to be executed next is now highlighted. 7. Repeat steps 5 to 6 until the step can no longer be incremented. 8. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 15 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort page”. 3. Load default data. 4. Verify that the first to be execute code statement is highlighted. 5. Increment step. 6. Verify the code statement that was to be executed next is now highlighted. 7. Repeat steps 5 to 6 until the step can no longer be incremented. 8. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 15 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Quick sort page”. 3. Load default data. 4. Verify that the first to be execute code statement is highlighted. 5. Increment step. 6. Verify the code statement that was to be executed next is now highlighted. 7. Repeat steps 5 to 6 until the step can no longer be incremented. 8. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 14 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press play animation. 5. Verify that during the animation up until it is complete that the correct code was highlighted. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 14 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort page”. 3. Load default data. 4. Press play animation. 5. Verify that during the animation up until it is complete that the correct code was highlighted. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 14 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Quick sort page”. 3. Load default data. 4. Press play animation. 5. Verify that during the animation up until it is complete that the correct code was highlighted. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 16 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Hover over each part of the code. 4. Verify that some elements have informative tool tips. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 16 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort page”. 3. Hover over each part of the code. 4. Verify that some elements have informative tool tips. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 16 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Quick sort page”. 3. Hover over each part of the code. 4. Verify that some elements have informative tool tips. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press play animation. 5. Verify that the animation plays continuously until the end. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press play animation. 5. Press pause animation. 6. Verify that the animation is paused. 7. Repeat steps 1 to 6 for the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press play animation. 5. Pause the animation (within 0.5 seconds at default animation speed). 6. Press play animation. 7. Verify that the animation continues from its current progress. 8. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press animate next step 5. Verify there is an animation, but it does not continue after completing the step. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Set step to 2. 5. Press go back one step. 6. Verify that the step is 1. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Set step to 2. 5. Press play animation. 6. Press go back one step (within 0.5 seconds at default animation speed). 7. Verify that the step is 2. 8. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press skip to end. 5. Verify that the step cannot be further incremented. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press skip to end. 5. Press return to start. 6. Verify that the step is 0. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 17 | 1. Repeat tests 34 to 41 with “Merge sort” page and “Quick sort” page. |
|  | 18 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press skip to end. 5. Press play animation. 6. Verify that nothing happened. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 18 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press skip to end. 5. Press animate next step. 6. Verify that nothing happened. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 18 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press skip to end. 5. Enter a value higher than the current step in the integer entry. 6. Verify that the current step didn’t change. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 18 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press skip to end. 5. Enter press the + button on the integer entry. 6. Verify that the current step didn’t change. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 18 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press skip to end. 5. Drag the step slider to the right as far as you can. 6. Verify that the current step didn’t change. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 19 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press go back one step. 5. Verify that nothing happened. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 19 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Enter a negative value in the integer entry. 5. Verify that the current step didn’t change. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 19 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Drag the step slider to the left as far as you can. 5. Verify that the current step didn’t change. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 18, 19 | 1. Repeat tests 43 to 50 with “Merge sort” page and “Quick sort” page. |
|  | 20 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Enter “1 213 3 4 5 9 10 3” into the custom data field 4. Press load custom data. 5. Verify the entire data is shown in 3D. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 20 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Enter “1 2 3 a4 5” into the custom data field 4. Press load custom data. 5. Verify the first 3 elements of the data are shown in 3D. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 20 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Enter “Resources/Data/defaultData.txt” into the custom data path field 4. Press load custom data path. 5. Verify the elements of the data in the path are shown in 3D. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 21 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Press load random data. 4. Verify there are random elements shown in 3D. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 6. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 22 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Press load default data. 4. Verify there are the elements from “Resources/Data/defaultData.” shown in 3D. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 6. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 25 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Press load random data. 4. Verify there are as many random elements shown in 3D as the configuration is set to 5. Adjusted the configured number of random elements. 6. Repeat steps 3 and 4. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 8. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 26 | 1. Test 57 but close and reopen the program after step 5. |
|  | 27 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press learn. 5. Verify that the animation is paused. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 27 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press options. 5. Verify that the animation is paused. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 27 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press step break down. 5. Verify that the animation is paused. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 27 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press learn. 5. Verify that you cannot interact with anything outside the modal but within the window. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 27 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press options. 5. Verify that you cannot interact with anything outside the modal but within the window. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 27 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press step break down. 5. Verify that you cannot interact with anything outside the modal but within the window. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 28 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press step break down. 5. Verify that it opens a new modal with step breakdown information. 6. Verify that playing the algorithm performs all these steps as many times as was said in the modal. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 8. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 29 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Click options. 4. Verify the options page is opened in a modal. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 6. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 30 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Click main menu. 4. Verify the current page is the main menu. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 6. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 31, 32, 33 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load default data. 4. Press play animation. 5. Verify all declared variables are shown as their expected value at any given time. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 34 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort” page. 3. Load default data. 4. Press play animation. 5. Verify the position within the recursion tree that is shown is the expected value at any given time. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Quick sort” page. |
|  | 35 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Mouse over the load custom data button. 4. Verify the valid range of input values is shown. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 6. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 36 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Mouse over the load custom data button. 4. Verify the valid range of number input values is shown. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 6. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 37, 38 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Options” page. 3. Verify the following things can be seen and changed:    1. The colour of the text/number models.    2. The default colour for the box models.    3. The colour of box models that relate to the “left” recursive call.    4. The colour of box models that relate to the “right” recursive call.    5. The colour of the highlighted box models.    6. The colour of the box models that are sorted.    7. The colour of executing code.    8. The colour of non-executing code.    9. The rate at which animation occurs.    10. Whether or not the camera is “free”.    11. Whether or not to use full screen.    12. Whether or not to show the frames per second in the title bar. 4. Set “free camera” to true. 5. Verify the camera speed can be set. 6. Verify each change that was made in step 3 and 5 has gone into effect. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 39 | 1. Run test 72 but close and reopen the program after step 5. |
|  | 40 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “About” page. 3. Verify the creator of the anonymous pro font is credited. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 41 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “About” page. 3. Verify the creator of the digit models is credited. 4. Verify Joel Mahon is credited. 5. Verify the creator of learnopengl.com is credited. 6. Verify stack overflow is credited. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 42 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “About” page. 3. Verify that it says the software was made in C++ and GLSL. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 43 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “About” page. 3. Verify that it says the software was developed in visual studio 2017. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 44 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “About” page. 3. Verify that it says this software solution was made as part of my final year project for my computer science degree at the University of Leicester. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 45 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “About” page. 3. Verify that it says the algorithms are not optimised. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 46 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “About” page. 3. Verify that it says the C++ code shown in program may not compile. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 47 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “About” page. 3. Verify that it says there are legal risk of redistributing the program and that before doing so you should check all those that are credited for legal restrictions of redistribution of their work. 4. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 48 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Press the ‘Esc’ key. 3. Verify that the program closed. 4. Repeat steps 2 and 3 for every page. 5. Repeat steps 2 and 3 while a modal is open, on each of the interactive algorithms’ pages, for each modal. 6. Repeat steps 2 and 3 while an animation of an algorithm is occurring. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 49 | 1. Repeat test 82 but with F11 while the screen is not maximised 2. Verify that each time you press F11 the screen becomes maximised. |
|  | 50 | 1. Repeat test 82 but with F11 while the screen is maximised 2. Verify that each time you press F11 the screen stops being maximised. |
|  | 51 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Configure random data length to 10. 4. Load random data. 5. Press play animation. 6. Verify that at the start and end of each step of the entire animation that the ImGui window does not obscure any 3D models. 7. Repeat all prior steps for the other legal random data lengths. 8. Repeat all prior steps with a full screen window, a narrow window, and a wide window. 9. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 10. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 52 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Configure random data length to 10. 4. Load random data. 5. Press play animation. 6. Verify that during entire animation that the 3D models stay within the window. 7. Repeat all prior steps for the other legal random data lengths. 8. Repeat all prior steps with a full screen window, a narrow window, and a wide window. 9. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 10. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |

Table , manual test steps for verifying user interface requirements

|  |  |  |
| --- | --- | --- |
| Test Id | Req. Id | Test Steps |
|  | 1 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Press play animation. 5. Verify that when cubes representing data move that the numbers on them move along with them in an identical fashion. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 2 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Set free camera to true in options. 5. Move the camera using WASD/Space/Shift inside a cube. 6. Verify that the surfaces are not black. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 8. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 2 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Set free camera to true in options. 5. Move the camera using WASD/Space/Shift inside a number. 6. Verify that the surfaces are not black. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 8. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 3 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Verify that the surfaces of the models appear to incorporate diffuse lighting. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 6. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 4 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Set free camera to true in options. 5. Move the camera using WASD/Space/Shift to many varying locations. 6. Verify that the surfaces of the models appear to incorporate specular lighting. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 8. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 5 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Verify that the models have more than 1 colour between them. 5. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 6. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 6 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Set free camera to true in options. 5. Verify that you can move the camera using WASD/Space/Shift. 6. Verify that you can look around by holding LMB and dragging the mouse around. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 8. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 7 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Change the dimensions of the window to many different combinations. 5. Change the window the full screen and not. 6. Verify that the 3D rendering is not distorted at any time. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 8. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |

Table , manual test steps for verifying 3D rendering requirements

|  |  |  |
| --- | --- | --- |
| Test Id | Req. Id | Test Steps |
|  | 14 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Press play animation. 5. Verify the models of the data are representative of the data (e.g. a 5 has a 5 model, a 67 has a 6 model and a 7 model next to each other). 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 15 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Press play animation. 5. Verify the models never move (seemingly) instantly. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 16 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Press play animation. 5. Verify the models never clip with each other. 6. Repeat steps 3 to 5 4 more times and with the default data. 7. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 8. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 17 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load the default data. 4. Press play animation. 5. Verify there is an appropriate animation when two elements are compared. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 18 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load the default data. 4. Press play animation. 5. Verify there is an appropriate animation when two elements are swapped. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 19 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort” page. 3. Load the default data. 4. Press play animation. 5. Verify there is an appropriate animation when data is (re)moved from a vector. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Quick sort” page. |
|  | 20 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort” page. 3. Load the default data. 4. Press play animation. 5. Verify there is an appropriate animation when data is appended to more data. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Quick sort” page. |
|  | 21 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Quick sort” page. 3. Load the default data. 4. Press play animation. 5. Verify there is an appropriate animation when data is prepended to more data. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 22 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort” page. 3. Load the default data. 4. Press play animation. 5. Verify there is an appropriate animation when the first element of some data is appended some other data. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 23 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load the default data. 4. Press play animation. 5. Verify that whenever an element is confirmed to be sorted that it is turned green. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 24 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load the default data. 4. Press play animation. 5. Verify that whenever an element is moved to a recursion that its vertical position is reduced by a regular interval. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 25 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load the default data. 4. Press play animation. 5. Verify that all animations have acceleration/deceleration. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 1 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load the default data. 4. Press play animation. 5. Verify that bubble sort is what is being executed. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 2 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Merge sort” page. 3. Load the default data. 4. Press play animation. 5. Verify that merge sort is what is being executed. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 3 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Quick sort” page. 3. Load the default data. 4. Press play animation. 5. Verify that quick sort is what is being executed. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. |
|  | 12 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load random data. 4. Press skip to end. 5. Verify that the data is now sorted. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 13 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load the default data. 4. Press play animation. 5. Verify that all states are correct. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 13 | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Load the default data. 4. Set the step to a random value. 5. Set the step to another random value. 6. Set the step back to the first random value. 7. Verify the state is the same both times you go to the first random step. 8. Repeat steps 4 to 7 10 times. 9. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 10. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |

Table , manual test steps for verifying animated algorithms requirements

|  |  |
| --- | --- |
| Test Id | Test Steps |
|  | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Hold ctrl and click the random data amount. 4. Set the value to 100. 5. Press load random data. 6. Verify there are 100 random elements shown in 3D. 7. Press play animation. 8. Verify that the animation plays in its entirety without error. 9. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 10. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |
|  | 1. Run the release version of AnimationOfAlgorithms.exe. 2. Navigate to the “Bubble sort” page. 3. Press load random data. 4. Leave the program running for 8 hours. 5. Verify that the program has not crashed and behaves normally. 6. Repeat all prior steps with the debug version of AnimationOfAlgorithms.exe. 7. Repeat all prior steps for with “Merge sort” page and “Quick sort” page. |

Table , stress test steps

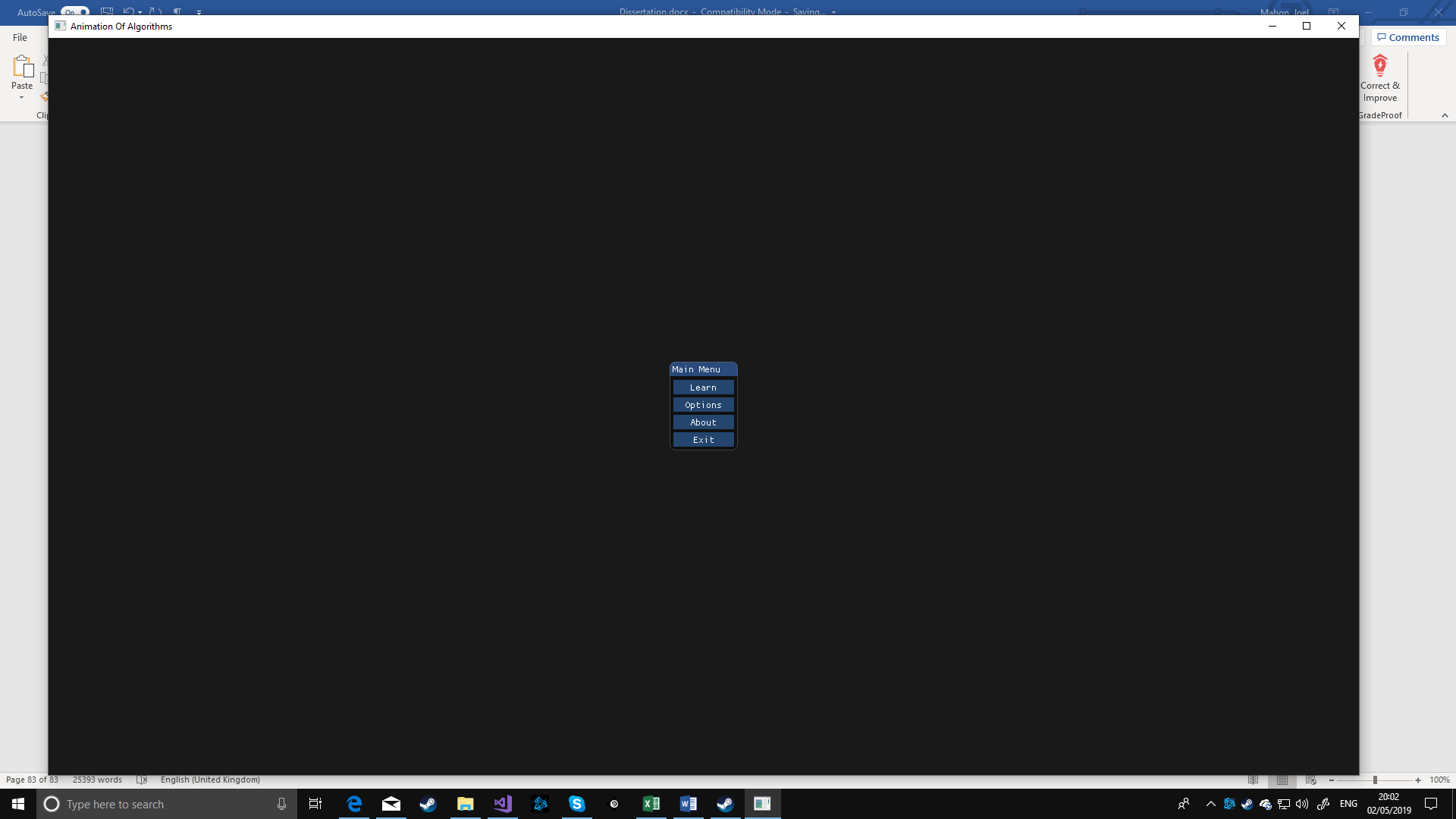


Figure 39, main menu page

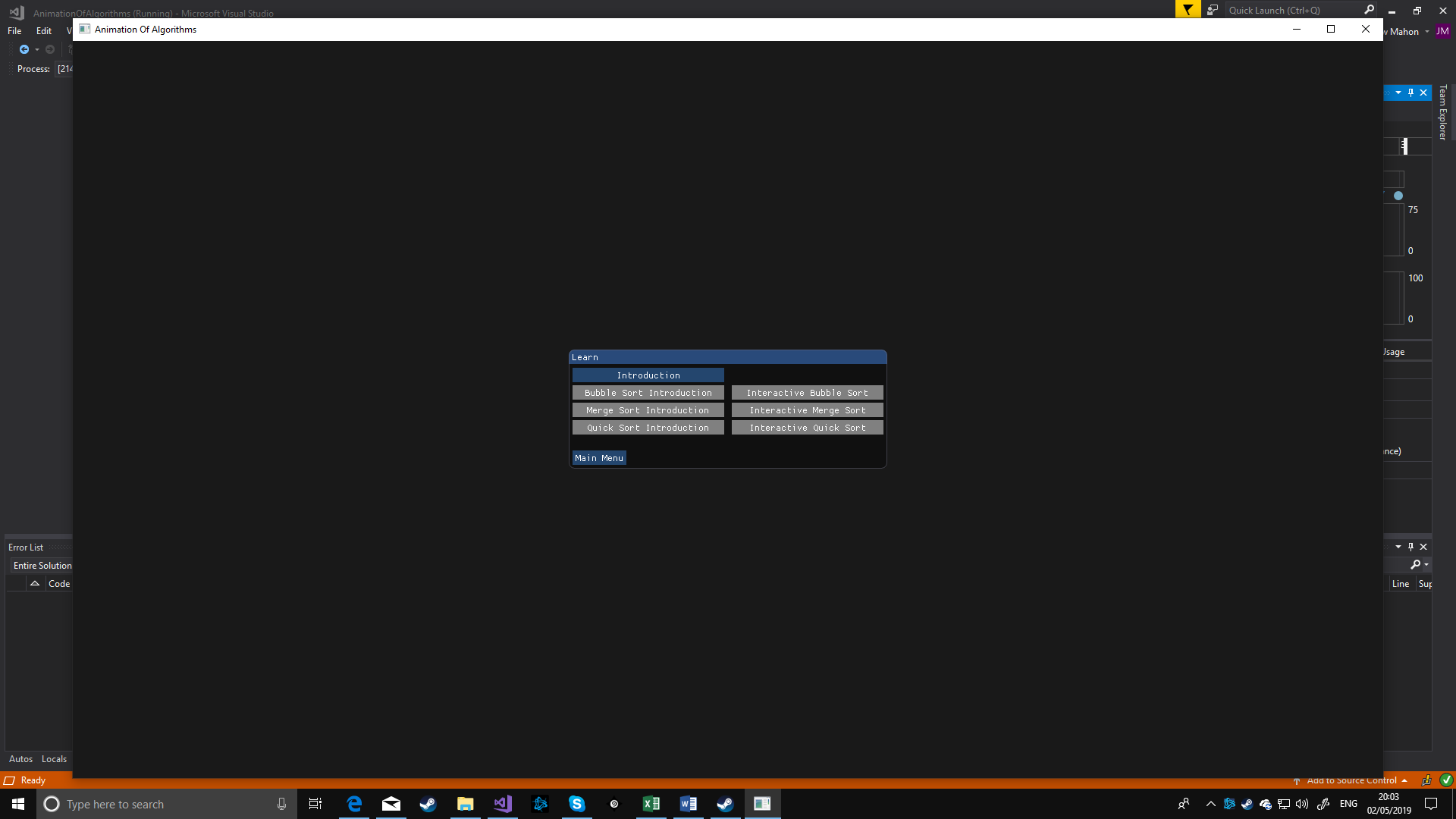


Figure 40, learn page

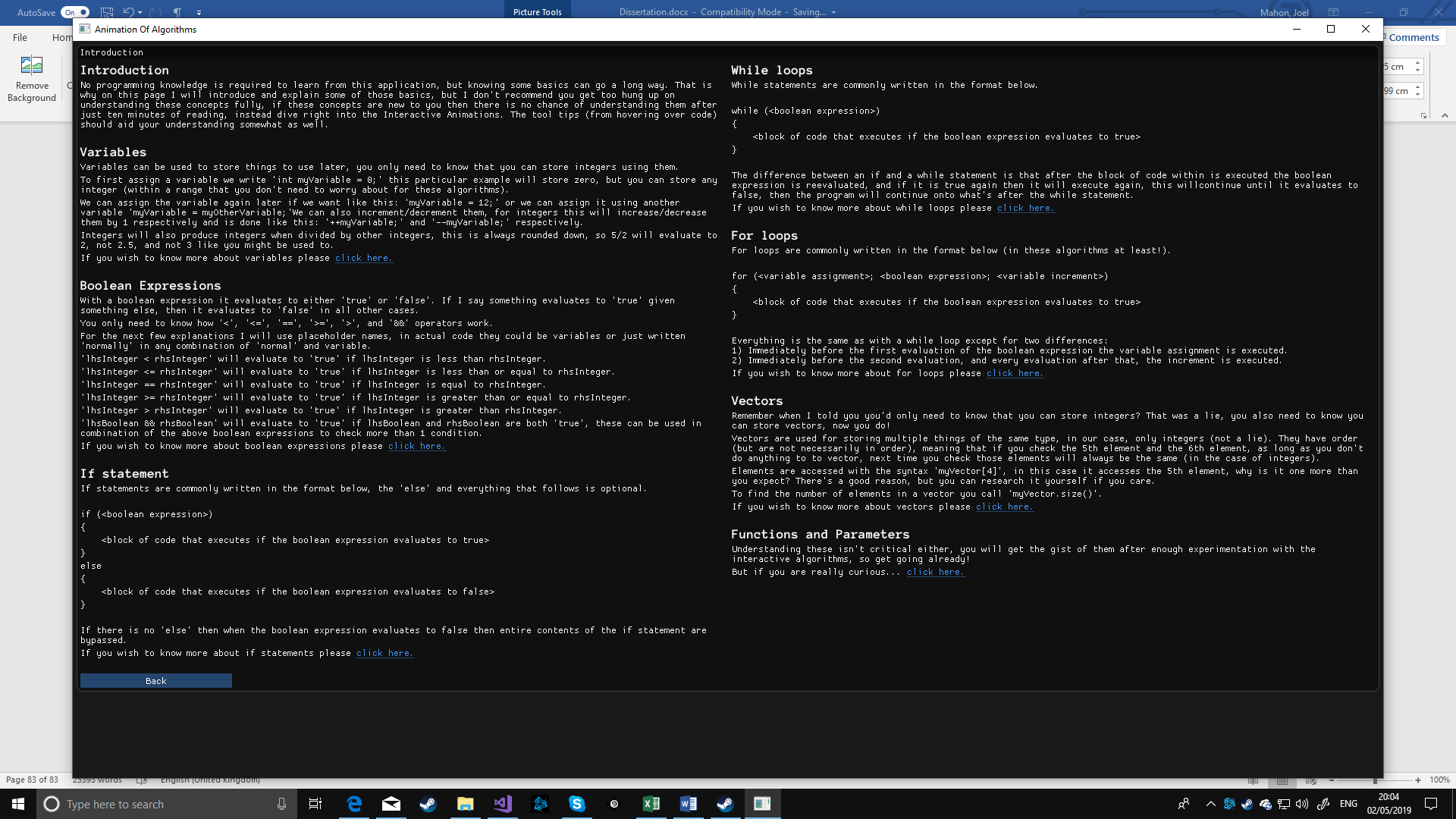


Figure 41, introduction page

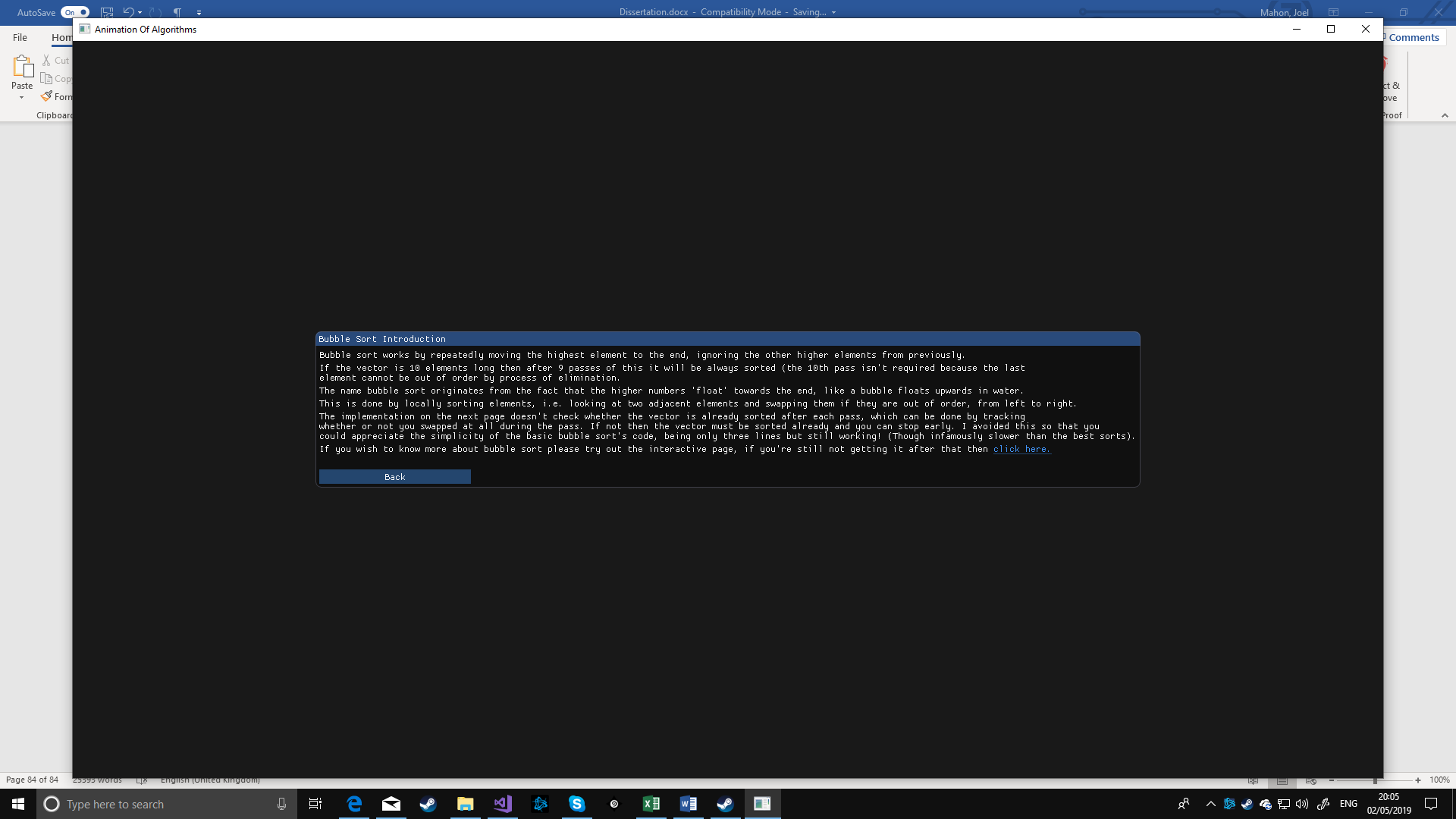


Figure 42, bubble sort introduction page

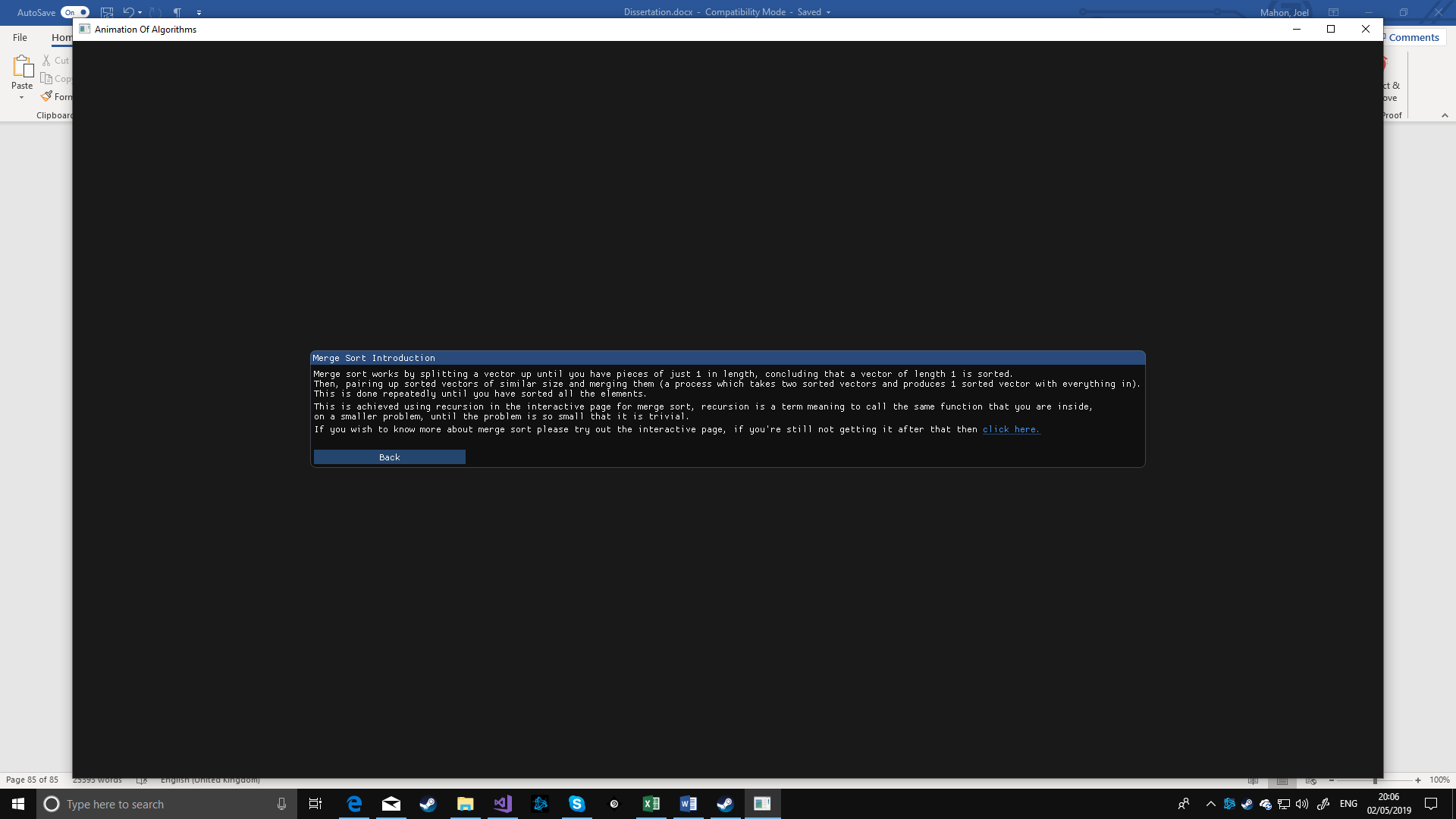


Figure 43, merge sort introduction page

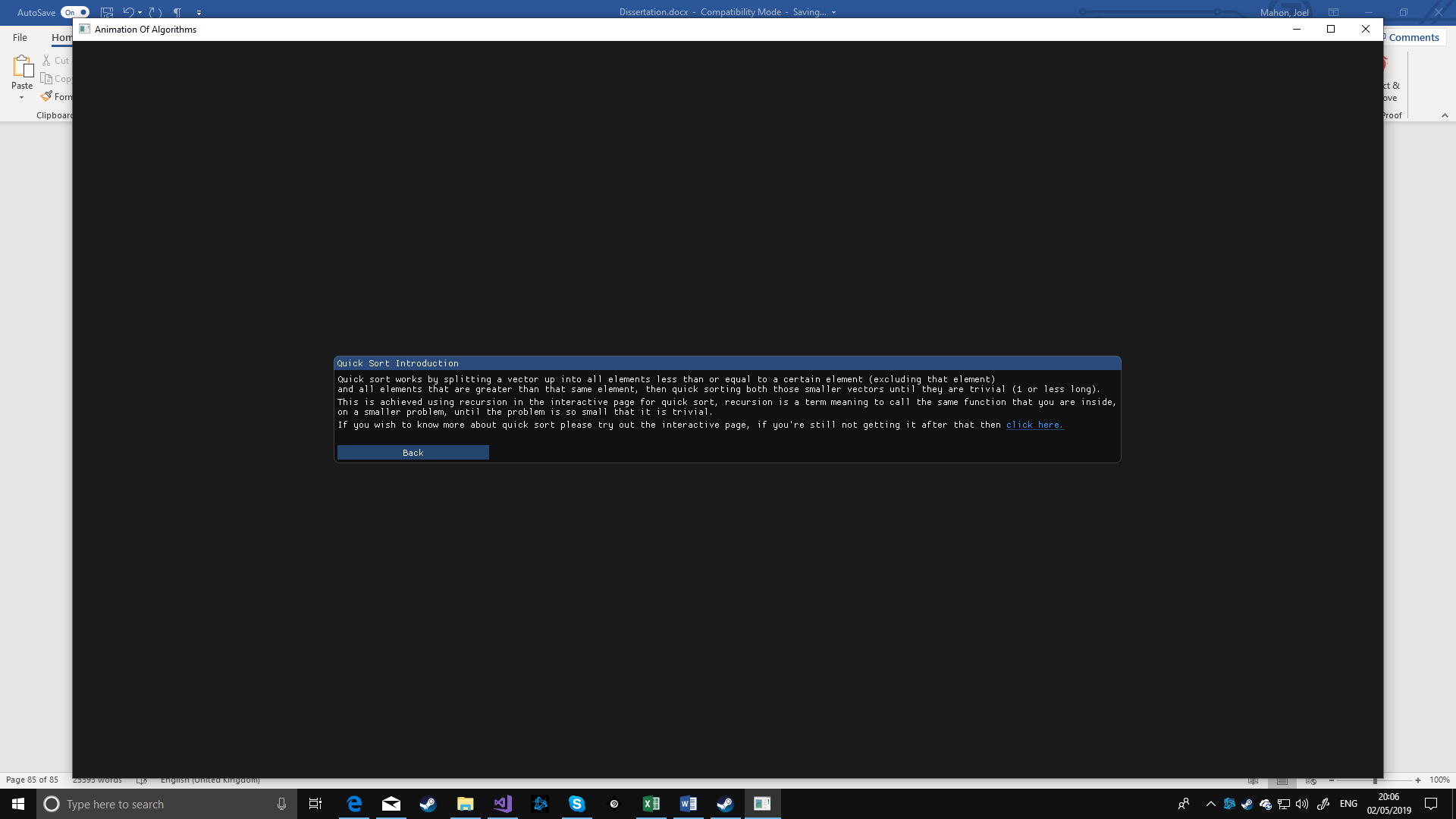


Figure 44, quick sort introduction page

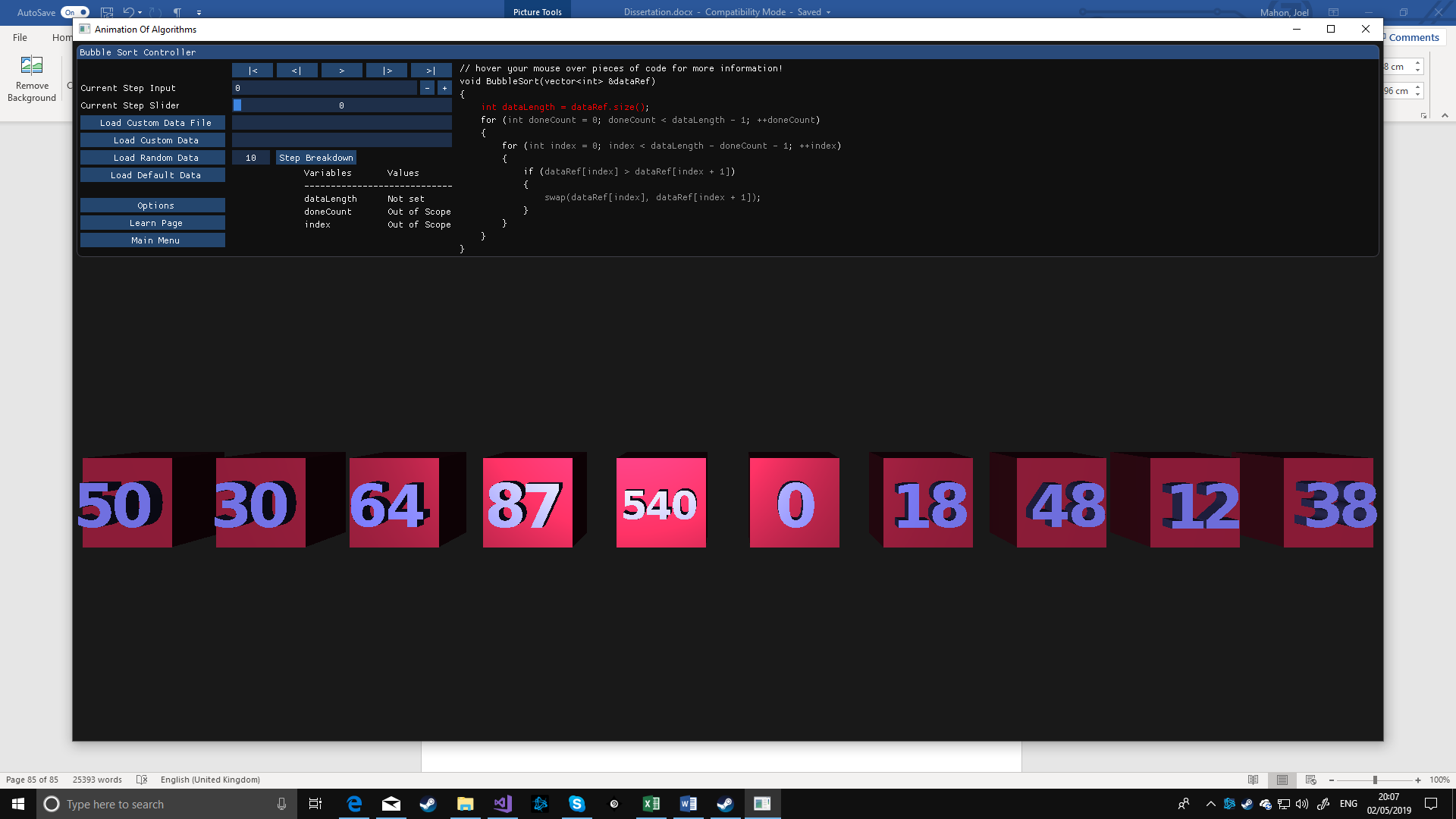


Figure 45, interactive bubble sort page

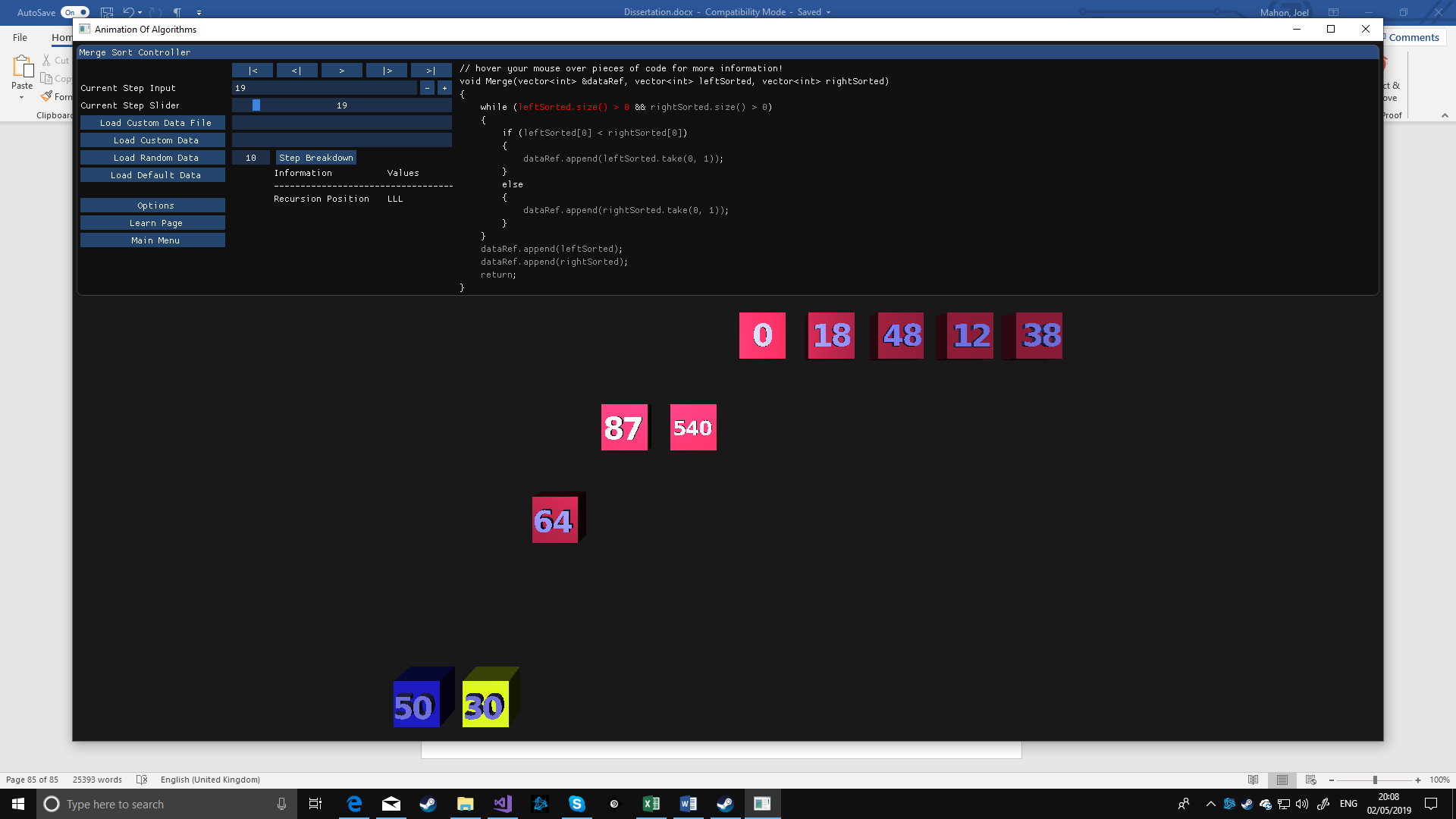


Figure 46, interactive merge sort page

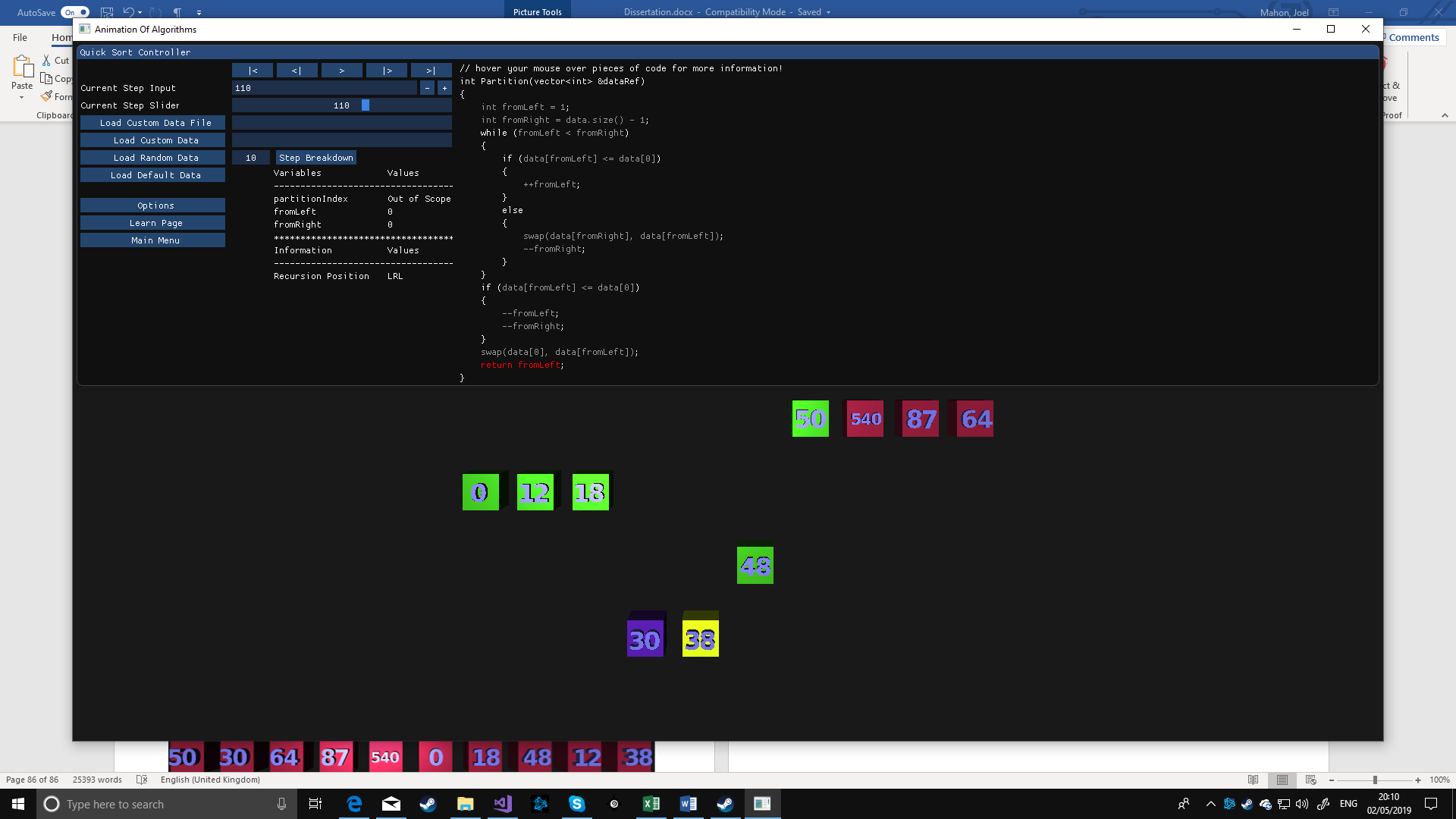


Figure 47, interactive quick sort page

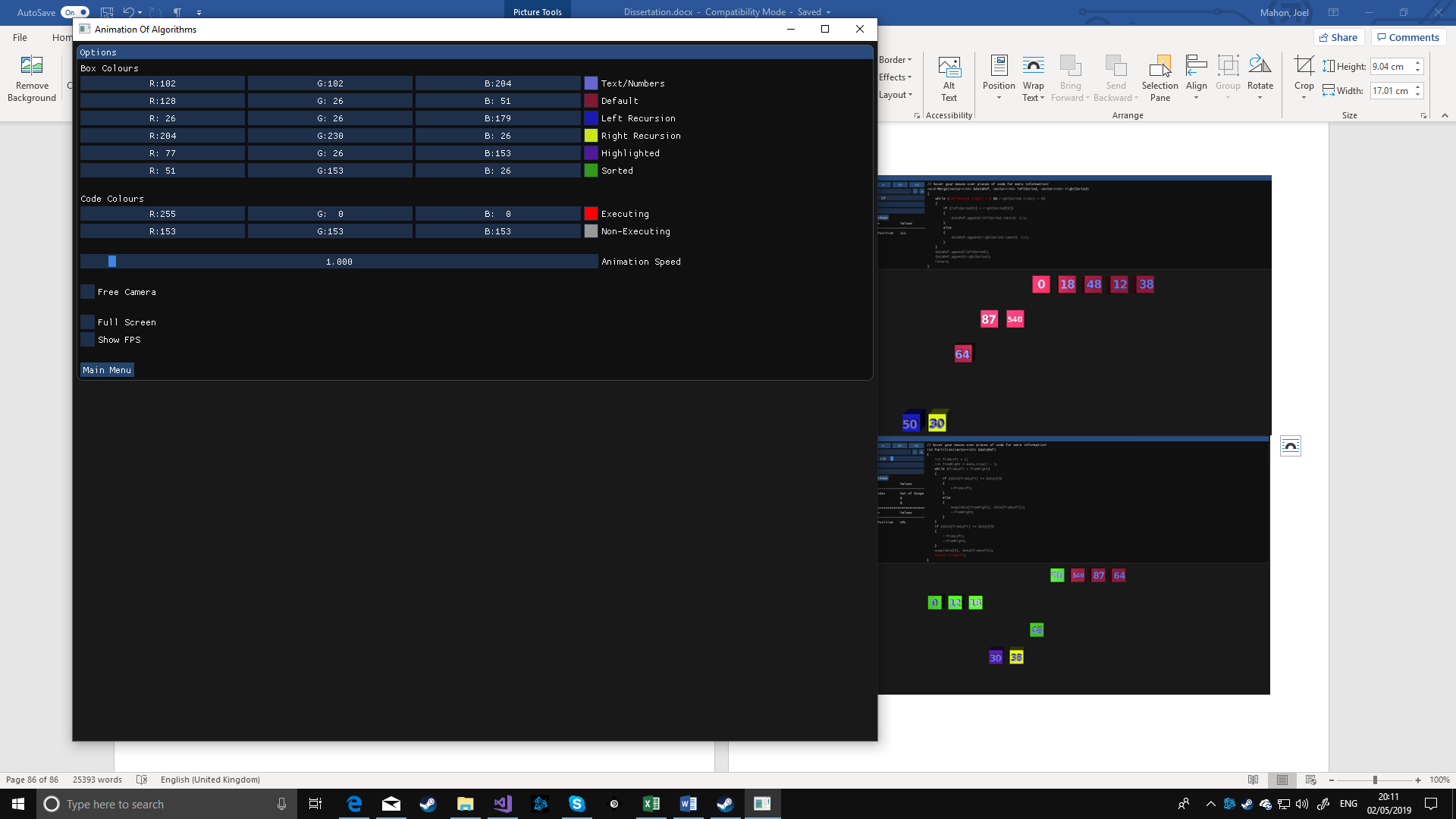


Figure 48, options page

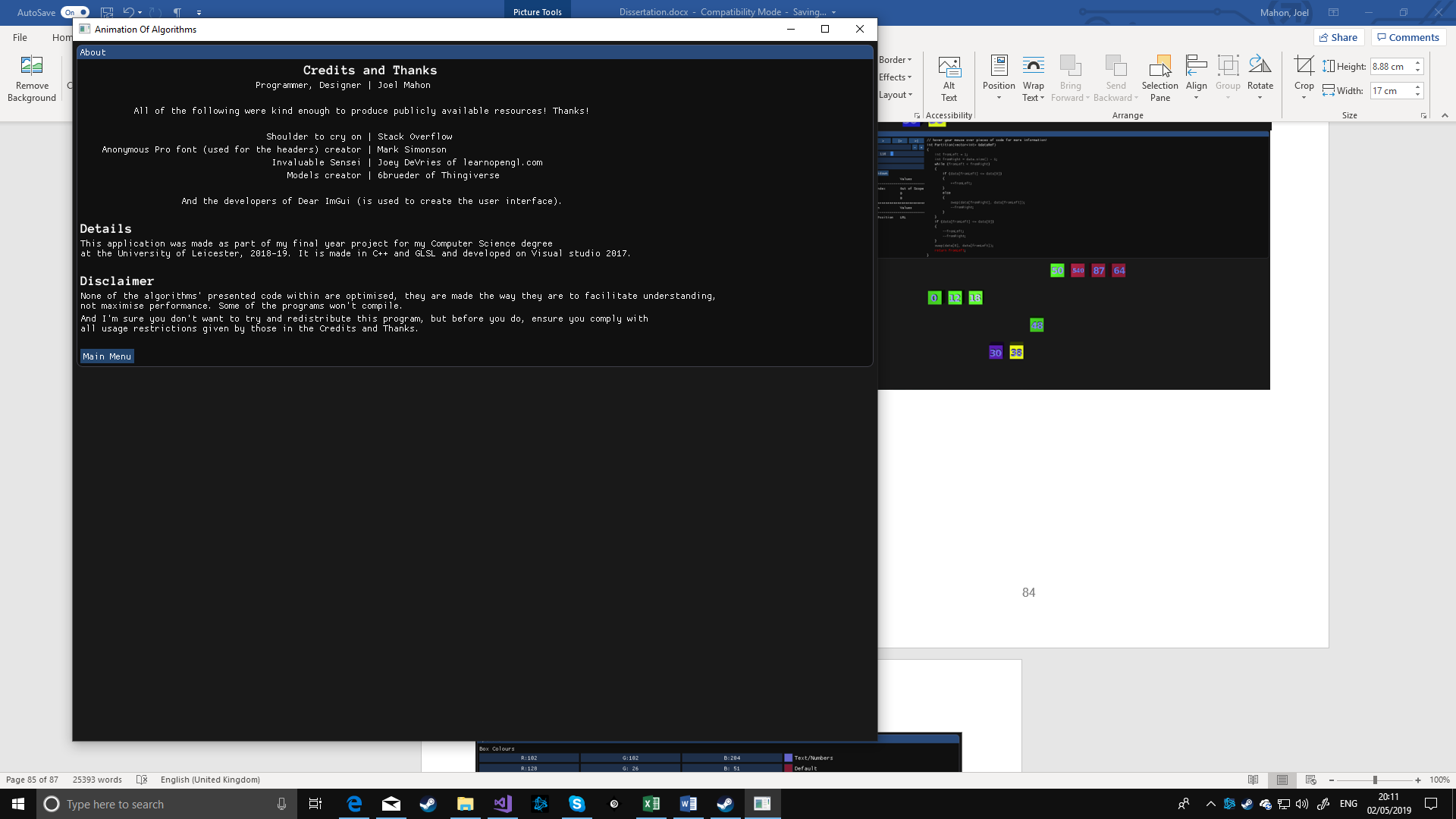


Figure 49, about page