Team Project Report - Group E2

Michael Chaharbaghi
Andy Chen
Sam Durdy
Thomas Huw Harris
Mitchell Savage
Misha Wagner

24/03/2016

1 Table of contents

Contents

2 Introduction

Voronoi Generation is the method of dividing a two dimensional plane into polygons, based on a set of points referred to as seeds. The area contained within a polygon is all points on the plain closest to the polygon's seed. The Voronoi Diagram was named in 1908 after the first man to study them, the Russian and Ukrainian Mathematician Georgy Voronyi. [1]

This project's goal was to create a product to help visualise Voronoi diagrams as well as teach users to give them a basic understanding of the algorithm and how it works. This report acts as a summary of the product as well as explaining, demonstrating, and evaluating our team work, visual design, software engineering, and risk management.

First of this document will go into great detail about the design process of the product through screenshots and details of the various external tools we used to aid our work. After this we'll demonstrate the extent of the software engineering utilised for this product creation. This includes our `UML class diagrams`, Model View controller design pattern usage as well as substantial project planning which was employed to ensure as smooth a creation process as possible.

Next it will cover the comprehensive risk management which was undertaken before development, and analysed after, to ensure all potential risks were minimised. This is followed by the team work section, which details how the team thrived and fared under the pressure of the issues brought on by the project. Here is where the report will go into more depth using SVN activity evidence as well as a Gantt chart.

Finally, is a conclusion to this document, which will summarise the findings detailed in this report.

Also appended is the specification, individual pages for each team member, and the appendixes for any attachments, references, and linked work

3 Software Design

3.1 Model View Controller

Outline

The Model View Controller (MVC) architectual pattern was adopted to the structure of the code. This allowed the team to work in parallel in each of the three sections of the MVC pattern.

The three sections are also represented in our package structure, with the only naming exception being that the `gui` package relates the View of MVC.

As Model, View, and Controller are loosely coupled, we will talk about their structure in three separate sections.

Model

The model contains two key classes: `Canvas`, and `VoronoiPolygon`.

`VoronoiPolygon` is the representation of a polygon, and contains a centre point as well as a set of edges. It contains methods for all operations on itself, including trimming by a line, getting the bisection between itself and another polygon, checking if it is a complete polygon, and joining itself with the border of the canvas. For that last method, `VoronoiPolygon` has a `Canvas` field, too retrieve information on the dimensions of the panel.

The Canvas is essentially a container for it's polygons, and contains methods that effect each of them. This includes a method for adding a new point to the canvas, which is the main functionality of the program.

This model package also contains a mathematics package. This contains adapted versions of decimals, points, and lines to use Java's `BigDecimal` class. The use of `BigDecimal` was essential to the project due to the precise calculations needed for generating polygons. Because small numbers were often manipulated into larger numbers, the margins of error produced were significant when floating points were used. For example, this was the case for calculating the intersections of lines. However, while Java's BigDecimal library had sufficient precision, it measured equality exactly - which was still

a problem after rounding values at the last digit. To solve this, a wrapper was implemented around it: `model.math.MarginedBigDecimal`. This had a separate equality function. The `model.math.LineBD` and `model.math.PointBD` exist to have structures that support our `MarginedBigDecimal`, otherwise Java's built in classes would have been used.

View

The view section of our product is what provides the visual representation to the user of all the hard work done by the algorithm. Here we have 4 main view classes `gui.MainGUI`, `gui.WelcomePanel`, `gui.WelcomePanel` and finally `gui.ManualFrame`. Each classes title describes it's function.

The `gui.MainGUI` class is the core frame of the product. It is the JComponent which contains most other JComponents during runtime.

The first of the components to make use of the JComponent is `gui.WelcomePanel`. This is the welcome page designed for the product from which it is possible to navigate to all the various different sections of the program to access the various features. It is also from here where any settings can be changed such as the language or font size.

After visiting the welcome page then one of the next possible views for the user will be the canvas page created by the class `gui.CanvasPanel`. It is here where the majority of the work relating to the view is done. From here the user can control various parameters of the algorithm, such as play mode (continuous or step-by-step), number of nodes or play speed (if appropriate). Once the user has finished setting parameters, a button panel at the bottom of the screen can be used to begin, exit, or go home and also to later on reset. This button panel was designed to be as simplistic and intuitive as possible. It was decided to not have an excessive amount of verbose and unnecessary labels.

The final feature of the view is the `gui.ManualFrame`. This is perhaps the simplest in terms of class complexity. It is the smallest of the 3 panels (GUI class excluded) but it is home to one of the more exciting features of our product. Here the user is able to use their mouse to click wherever they want on the canvas to add a seed at that point. As they add in each point, the Voronoi diagram automatically adjusts each polygon to accommodate this new point.

Controller

The controller feeds input from the view into the model, and does so with the `controller.CanvasController` class. The view has a `CanvasController` field, which is used to send information to the Model. This is achieved through delegating methods of the model inside `CanvasController`. This provides a layer of abstraction between the two, decoupling the model and the view.

The controller was also used for managing updates to the display. This was done using an Observer pattern, where the controller sets the view to observe the model. This meant whenever the model was updated, the view would be too without having a strong coupling between the two sections.

The `controller` package also has a `Holder` class. This is what prevents the model from progressing ahead of the view, in modes where the progress of the algorithm is retrained (e.g. step-by-step mode). The model calls a methods called `hold()`, which blocks until the controller calls a method called `next()`. This allows us to control the speed of Voronoi generation.

3.2 Maven usage

In order to manage dependencies, we used the build tool Maven [2]. This works by including and XML file detailing all the dependencies of the project. There are two alternatives to using a build tool like Maven:

1. Pushing large Java .jar files into the SVN repository to maintain consistent dependencies.
2. Notifying all team members whenever a dependency is required, and letting them know what exact version is being used so they can import it independently.

These are very flawed solutions:

1. Having large files on SVN can cause updates to be slow, and take up a lot of room on the repository.

2. Team members may download the incorrect version of files, which can lead to a lot of confusion and errors across projects.

From this, we decided to use Maven to manage our dependencies, as it solves the dependency problem without the two aforementioned flaws. Maven was used to manage logging libraries, testing libraries, and JSON parsing libraries. It was also used to set a project-wide Java version, so that all team members were working with the same version.

3.3 Logging usage

The logging library Simple Logging Facade for Java (SLF4J) [3] was utilised across our project. Each class that required it had it's own logging object. This had the benefit of us being able to set different logging levels for each class. There were different logging levels available, with `DEBUG`, `INFO`, `WARN`, and `ERROR` being used the most across the project, each applied appropriately. For example, `DEBUG` was used mainly when printing out results of calculations across the model, while `WARN` was used when a calculation succeeded but with unexpected results.

SLF4J was used in conjunction with Logback. Logback is an implementation of SLF4J, which supplies a `logback.xml` file in which logging configurations can be defined. Such configurations were logging levels for the whole project, logging levels for individual classes, and the format of log printing.

Using logging libraries proved as a useful tool. The ease of setting logging levels for different classes meant that useful logs could be left in, and when debugging was necessary they could be reactivated. This improved the time in which we could solve bugs significantly, without printing too much information.

3.4 JSON usage

JSON [4] was used across the `gui` package for handling languages. This code can be found in the `gui.information` package. The class structure here is `JsonHandler` as a superclass, which has methods for reading from JSON files. This is inherited by `LanguageHandler`, which adds functionality for setting languages by language codes.

The `LanguageHandler` class is inherited by a further two classes: `ExplanationHandler` and `PromptHandler`. One is for storing the explanations of each step of the algorithm, and one for storing the prompts for buttons on the screen. The reason these are in two separate classes is because the structure for the underlying JSON for each file was different, and so had to be retrieved differently.

Arguably, it could be a better approach to combine the `JsonHandler` and `LanguageHandler` classes, as `JsonHandler` is only inherited by `LanguageHandler`. However, keeping them separate is better for future extensions to the code, as if a new requirement needs the functionality of parsing JSON but not of setting languages, then it should only inherit the JSON functionality. On top of this, handling JSON and handling languages are two separate purposes, and should not be tightly coupled in the code.

4 Visualisation Design and HCI

The program is mainly composed of 3 sections: the welcome panel, the tutorial panel and the manual panel. All three panels will be explained in greater depth in the following sections.

4.1 Welcome Panel

Figure 1 shows the first panel that the user sees when they go into the program.

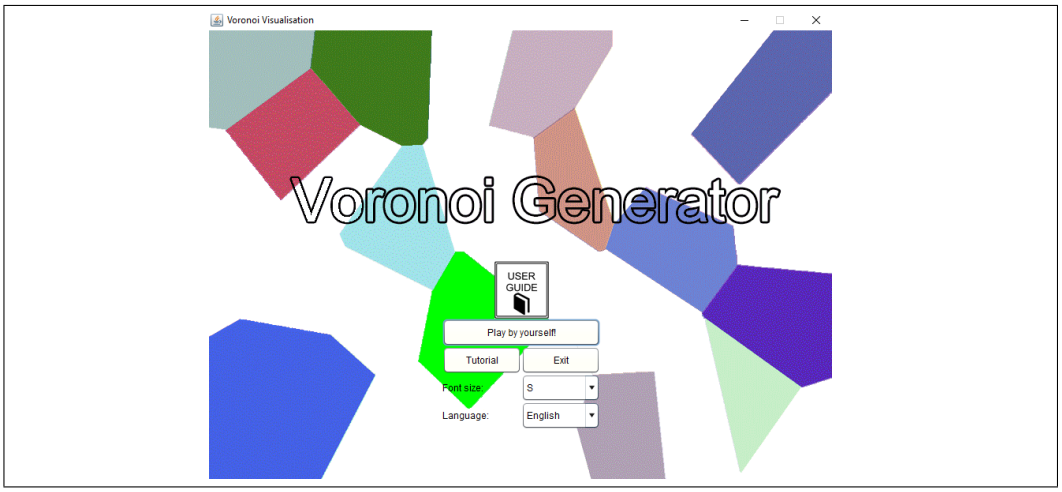

Figure 1: Welcome Panel

Here the user can choose to start a tutorial of how the Voronois are generated, or to run the software as a Voronoi generator ('Play by yourself') where the users can place seeds on their own. Furthermore, the program allows the user to switch between English, Spanish and Chinese language options. These were decided upon because they are the most widely spoken languages, which will allow us to increase the number of potential users. Moreover, the font size of the program can be changed, so that users can choose the font size they like and are comfortable with.

A user guide is also provided, where the user can learn more about Voronoi diagrams, and how to use the software.

The program contains three visualisation modes so that users are able to

learn about the algorithm in a variety of ways. They are able to see how a Voronoi polygon is generated in more detail in the tutorial modes, which will give them an understanding of the backbone of the algorithm. Users are also able to place points by themselves in manual mode, which allows them to see how the algorithm reacts to different conditions and arrangements of seeds.

4.2 Tutorial

4.2.1 Control Panel

If the user decides to start the program as a tutorial, the control panel, shown in Figure 2, is opened. At the top of this window are a number of useful tips for using the software. The team observed during testing that many users do not read the user guide, and so this panel was included so that users will not be as confused when they enter the program.

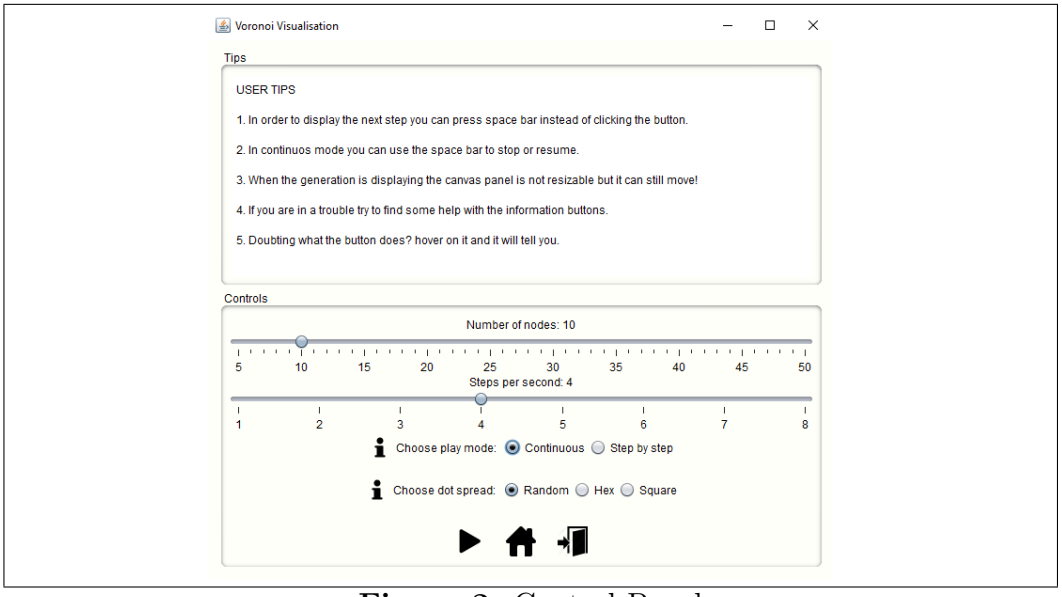

Figure 2: Control Panel

Below this are a number of controls for the visualisation. First, the user can decide whether to display the visualisation continuously or step-by-step. In step-by-step mode, the user can step through the algorithm, with each step progressing the algorithm a stage. This allows a user to see what is

happening at each stage, as well as reading a short explanation which goes into more detail.

In continuous mode, the visualisation is shown continuously, at a speed set by the 'steps per second' slider. This mode allows the algorithm run faster than they would in step-by-step mode, which may be an easier way for some people to learn. This mode does not have any explanations, as they are unreadable when the visualisation is running quickly.

The user can also change the seed generation mode of the visualisation between random, which places points randomly on the screen; square, which places points in a uniform formation, generating square polygons; and hex, which will generate hexagonal or diamond polygons, depending on the window's aspect ratio.

Both of these modes also contain a number of nodes slider, used to change how many points are added to the canvas. This is limited to 50, as the program will often run poorly when too many points are added. All of these options also have information buttons and tooltips to explain more about what they do.

4.2.2 Visualization Panel

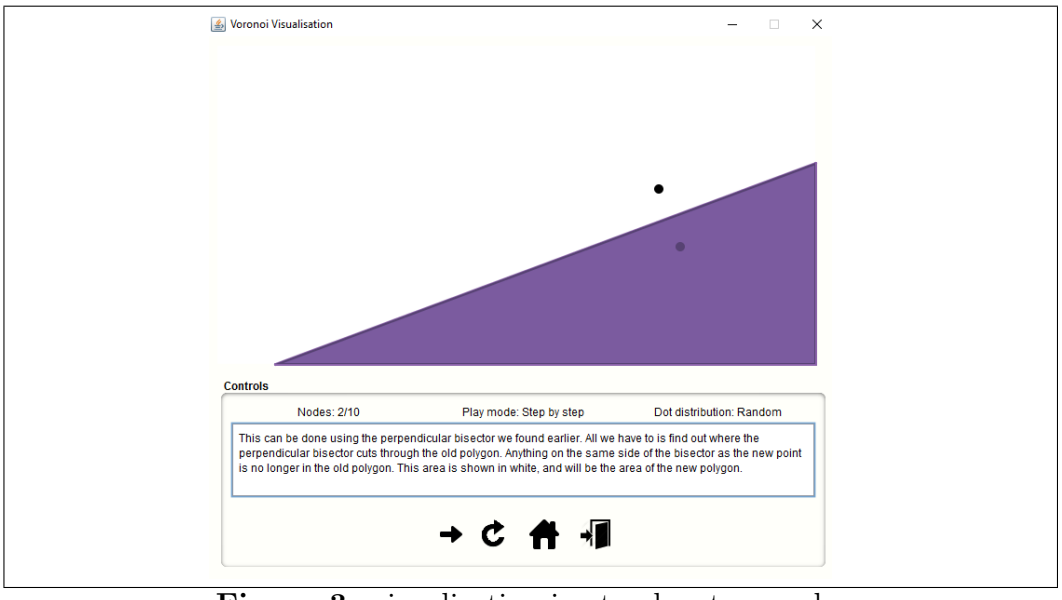

Figure 3: visualisation in step by step mode.

Shown in figures 3 and 4 are the visualisation panels for continuous and step-by-step modes. As well as containing the visualisations of the algorithm, as mentioned before the step-by-step mode contains an explanation for each step. Both modes also contain some information labels showing the options chosen for the visualisation. They also contain buttons to restart the visualisation (i.e. go back to the control panel), return to the welcome panel, and exit the program. In step-by-step mode, there is a step forward button included at the bottom to move forward a stage. In continuous mode, this is replaced by a pause button or a play button, which can be used to stop and start the visualisation as it is running. These buttons can also be controlled using the space bar.

Figure 4: visualisation in continuous mode.

4.3 Manual Panel

Figure 5: Manual Panel.

Figure 5 shows the manual panel. By pressing play, the user enters manual placement mode, where they can place seeds on their own. It must be noted that the user is unable to change the size of the frame whilst in visualisation mode, as this would require the program to regenerate all of the existing polygons. After changing the size of the frame, the user can press the restart button to go back to main screen of the manual panel, where they can resize the frame and start a new visualisation.

5 Software Engineering

5.1 Agile Development

This project followed the agile methodology. Of the twelve principles that the Agile Manifesto [5] is based on, we primarily focused on the Agile principles that were applicable to a University project. For example, "Best architectures, requirements and designs emerge from self-organizing teams" is not applicable to this project as the group has to be self-organizing because of the project setting. The agile principles which were most closely followed were:

Welcome changing requirements, even in late development
While formulating the project scope it became apparent to Dr. Kerber and Prof. Claridge that there had been a misinterpretation of the brief. After being discussion with Manfred and Ela our project scope was altered accordingly, focusing on teaching Voronoi Polygon generation well as opposed to looking at world generation which had previously been included in the project's scope. While at first other aspects of world generation were included as extensions to the project it became clear that these extensions would detract from the focus of the project, and so were excluded.

Later in development after presenting our prototype to Dr. Kerber it was suggested that the user should be able to alter the size of the font to ease usability. It was also suggested that the user should be able to pick where to put the points for the Voronoi Polygons. Both suggestions were enthusiastically taken on board and have been implemented to some extent in the final product.

Face-to-face conversation is the best form of communication (co-location)
The vast majority of development happened in the company of others. At meetings, after establishing the progress of the project and talking about how development could progress and be made more effective, team members would spend several hours coding, and improving the project while co-located. This is particularly when debugging code. All members of the team were exceedingly reliable in attending meetings and were attentive throughout coding sessions. Often pair programming was employed to increase productivity. During debugging, Sam,

Mitch and Huw debugged code together, finding it much more effective than tackling the task alone.

Working software is the principle measure of progress
Throughout the project the team strived for demonstrative, working code. For the first few weeks building a basic infrastructure for code and getting the core algorithm working was focussed on. Only once this was complete did the team worry about the aesthetics of the project.

Sustainable development, able to maintain a constant pace
Progress on the product was very steady. The team met up for progress establishment and collating work between one and three times every week. This was beneficial as it meant not only that the product was always moving towards completion, but also that the team was never over stressed about deadlines. This reduced the pressure on each member on the team, which no doubt contributed greatly to how cohesively team worked together. By avoiding over exposure to the rest of the team, the group maintained a professional, friendly operation at all times.

Continuous attention to technical excellence and good design
The project was implemented using the Model-view-controller design pattern, see bellow for more details.

Regularly the team reflects on how to become more effective and adjusts accordingly
As stated before the team met up between once and three times a week, every week. At the start (and also usually the end as well) of each meeting the progress made would be established, followed by the plan to continue development, and how our short term goals fit into the long term goals of the project. A prime example of the group adjusting to become more effective is the group's decision to collectively spend a whole week focussing solely on refactoring, and clearly segmenting the code. This decision was taken after the prototype demonstration, as before this point the team's focus on getting a working prototype had somewhat distracted from the separation of model, view and controller. Following the refactoring, the model-view-controller patter was much more closely followed, ensuring well designed code in the final product

However there are definitely ways in which Agile methodology could have been more closely followed. Namely there was a failure to interact with the intended end user of the product. This meant that there was no real way of

gauging customer satisfaction. Occasionally peers were asked to review the software, and the group took part in a session with other groups to compare and give feedback on software. However, while students were part of the target audience, there was no engagement with secondary school students and teachers which has resulted in a lack of knowledge of how well the product achieved what we set out to attain.

5.2 Model View Controller

This project implemented the Model-View-Controller (MVC) architecture pattern. The MVC pattern separates a project into three distinct segments.

1. View is what the user sees on screen. This segment ended up being the majority of the code in the project. This is mainly because Swing is quite verbose, and as such maintaining the loose cohesion between this segment and the other segments was vital for clean, simple code.

2. The Model manages the Voronoi Polygon generation algorithm. Functions within the `Canvas` class can be called by the controller to add seeds to the canvas, and the function `getJavaPolygon()` in the `VoronoiPolygon` class can be called by the view to retrieve an `java.AWT` object to be shown on screen. The `model.math` package contains classes used to help the model's calculations.

3. The controller is the segment which the user interacts with. These classes then interact with the model which goes on to update the view. In this project the controller is called called using Swing buttons, and clicking on the canvas. One of the main classes here is the Holder class. This class controls the speed of execution of the algorithm. This allows the user to progress through the algorithm at their own pace (in step by step mode) or by continuously at a constant pace (in continuous mode).

This implementation was enforced by separating the project into different packages for Model, View and Controller. This maintained a clear separation between each aspect of the project, forced maintenance of MVC, and encouraged a loose coupling between classes.

5.3 Project Planning

This project has evolved a lot over the course of development, which should not be perceived as negative. The nature of Agile development means that changes to requirements a welcomed.

5.3.1 UML Diagrams and Class Structure Planning

Almost immediately after the submission of the specification, a UML diagram was drawn up to keep track of classes and their purpose (See Appendix A, Figure 10). This was very helpful when talking about design, but was very simplistic, as it was only designed for the scope of the prototype. As can be seen there is no package structure, and no indication of which classes constitute Model, which constitute View and which constitute controller. With a lack of clear indication on the diagram, by the time submission of the prototype, for any given class it was unclear which segment that class belonged to, to anybody but the creators of that class.

As such it became obvious that refactoring was necessary, as was an updated diagram (Appendix A, Figure 11). The class structure changed very little after this (Appendix A, Figure 12). The main change being moving packages around to further emphasise the MVC pattern.

These UML diagrams can also be viewed in the SVN repository in the folder `UML Diagrams`.

5.3.2 Time Management and Gantt Chart

One area of planning which the team failed to follow through with was time management. During the creation of the specification a Gantt chart was created to plan the project across the term (Figure 6) however it was never referenced during the execution of the project. As it became clear that the project would remain focused on Voronoi Polygons, the correct thing to do would have been to update the Gantt chart to the changed requirements so that the team could have clear deadlines to work to. There was a failure to do this, which didn't prove harmful for this project. However were the team to do this on a larger, longer term project it could spell disaster. Perhaps this lapse in commitment to time management could be attributed to the

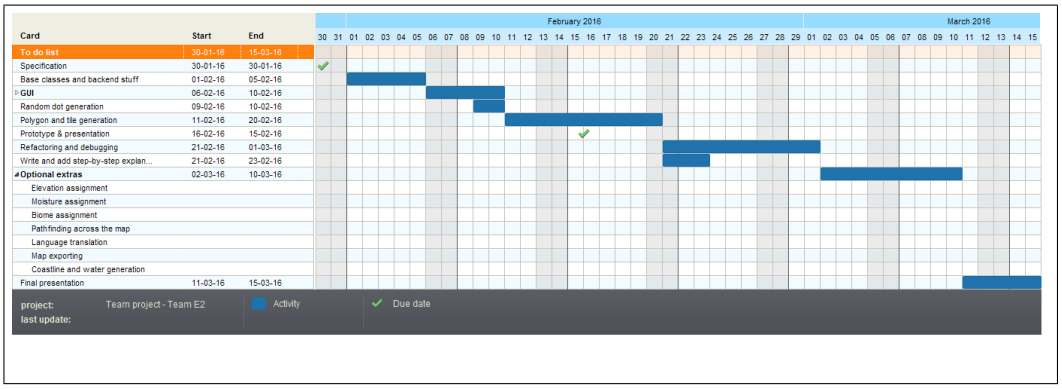

Figure 6: The gantt chart created for the Specification document

Gantt chart not being properly stored on the group's Google Drive folder, and so being inaccessible. But regardless of cause, this lack of awareness for time scale should be corrected in the future.

6 Risk Management

Across the project, several problems were encountered. In every case some time was lost, but risks were handled in appropriate ways and damage to the product minimised.

1. **Issues with SVN** During development, there were several issues with SVN. Most of these came from project restructuring. Major project restructuring occurred twice in development: one when Maven was integrated, and one when we started to use branching. This meant moving files between directories. SVN by default doesn't use "moves", but instead treats moves as deletion and addition. So if a team member had any work in a directory that was being moved, it would be deleted. This meant that all code had to be integrated before making major restructures to the project.

   This was avoided by coordinating when restructuring occurred. While this did bottleneck development, restructuring without acknowledgement from the entirety of the team would have slowed down production even more due to the potential of lost work.

2. **Issues with the wireless connection at University** As mentioned in the Software Engineering section of this report, the team often practices co-location (working side-by-side). As this was done at the university, several problems were encountered with wireless connection.

   This lead to problems with committing code to SVN, and at times could grind production to a halt before internet connection was re-established.

   One way this was dealt with was a greater segmentation of tasks. If there was a consistent problem with internet, each member of the team was assigned a task to be continued after the meeting. While collocation was a useful tool, the team managed to deal without it when issues occurred during meetings.

7 Team Work

7.1 Team Roles

For the majority of the project, the work was split between team members following our MVC design pattern.

Sam, Mitchell and Huw worked on the Model section. This involved creating the backend of the program, which dealt with adding new seeds to the canvas, generating a new Voronoi polygon based on this point, and processing any changes that needed to be made to existing polygons. This section required the most people because it was perhaps the most difficult part of the product to work out, due to the amount of bugs that continued to arise.

The work was split up fairly evenly between the three programmers, and most of the coding was done in a group, with all of us coming up with ideas on how to solve a problem, and one person implementing these ideas in code. This made it much easier to fix bugs and errors, and led to the project being worked on faster than it would have if we had each worked separately.

Misha worked on the Controller section. This involved creating a way of linking the GUI and the algorithm, allowing it to control things like how fast the algorithm was running. Because this was a smaller section than the others, he also worked on the maths used by the program. In particular, he implemented `MarginedBigDecimal`, a class which allowed us to more easily deal with floating point and rounding errors.

Andy and Michael worked on the View section, and were responsible for building the GUI, and dealing with any human/computer interaction considerations. Michael worked mainly on the design of the user interface, allowing user interaction with the algorithm and displaying the polygons. Meanwhile, Andy worked on making the interface as user friendly as possible, by including explanations and translations, as well as working on other GUI components such as the 'manual placement' mode.

7.2 Organisation

The team was organised using Trello [6], a website which facilitated the creation of a list of items that needed to be worked on, allocation of items,

and the monitoring of progress. This allowed tracking of changes that need to be made, and features that still need to be added to make the program function correctly. The board up was split up into sections based on MVC, as well as the documentation sections.

At the start of the project, a list of tickets was created which needed to be implemented into the final product. This was separated between Model, View and Controller, and each ticket was assigned to at least one team member. We also used it to keep track of our overall timeline, as well as any general changes that needed to be made, such as adding JavaDoc to the code and our intended coding style.

A number of labels was used to keep track of the items in the board, with purple indicating bug fixes, blue indicating improvements, and red indicating additional functionality. In the timeline there is also a green label indicating that something is completed, and an orange label to indicate what we should currently be working on.

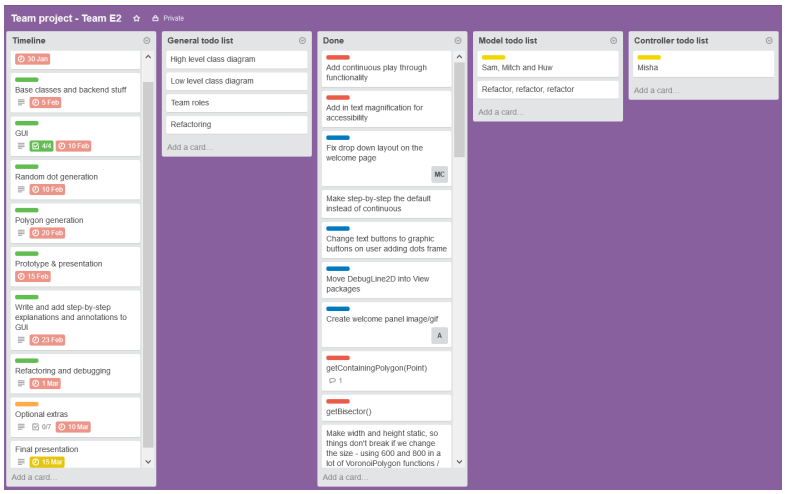

Figure 7: Snapshot from our Trello board used for team organisation

7.3 SVN Activity

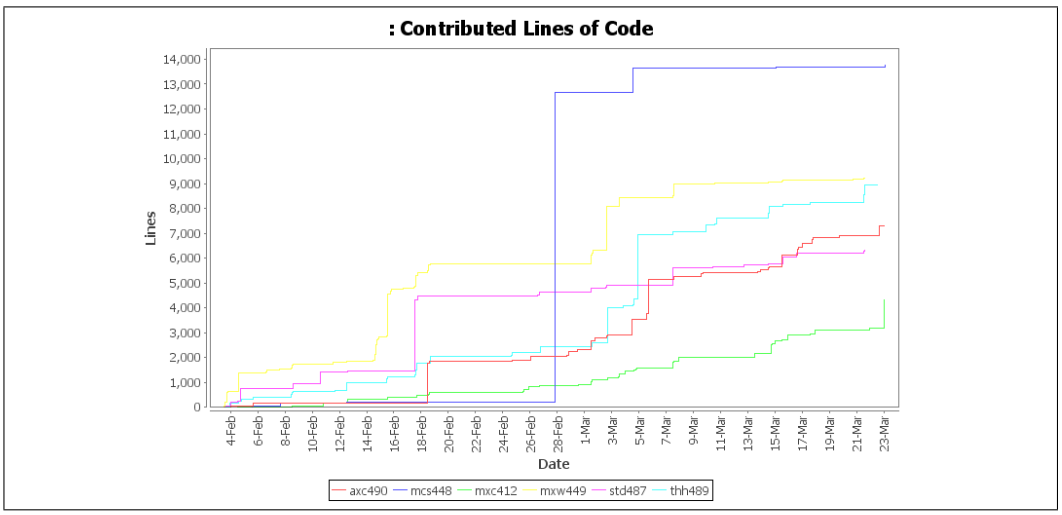

Figure 8: A graph showing the contributions of each team member

Subversion (SVN) is a version control system who was used extensively throughout the project. It allowed the team to work simultaneously on different sections of the code, and also kept backups of every stage of development throughout the project.

Figure 8 includes a graph showing the number of lines contributed by each member of the team. It is important to note that certain parts of this graph are inaccurate. Due to the team's use of co-location, pieces of code which may have been worked on by a whole group are only included under a single user. As well as this, a significant modification can be seen by Mitchell on 28th February, when he accidentally committed a number of metadata files which were all counted as new lines of code.

8 Evaluation

The original project idea of a full world generator was something the team was excited about, However, it was flawed from the start due to misunderstanding with the project's requirements. This slowed down planning considerably as the project's scope and goals had to be changed to suit the brief that had been given. That time would have been better spent on gaining a deeper understanding of what the brief entailed, which in turn would have enabled us to spend more time on planning out the needs of the product and our distribution of time on the constituent tasks.

This issue was solved by scaling back the original, overambitious idea into a single part of the world generation algorithm: Voronoi Diagrams. By cutting down on the features of the idea, its functionality could be better tailored to be an educational visualization of an algorithm. This also allowed us to gain a better understanding of the algorithm, and in turn create simpler and more focused software which imparted this knowledge onto the user.

The next major issue was the version of the algorithm that to be used. The consensus that found when looked into the different methods available, was that the best way of creating the polygons was through a method called Delaunay Triangulation. Despite being the quickest, most efficient and easiest to implement, Delaunay triangulation was one of the hardest to visualize. The algorithm was complex, and as a result very difficult to explain. This made it unsuitable for the project specification so other ways of showing how the polygons were formed were looked into. It was decided to use a method that consisted of bisections, as it was the way that best taught our own team, and gave the most insight to what a Voronoi polygon was. Unlike the triangulation, it is possible to step through each stage of a point being added into a diagram. Although this gave a better visualization to work with, this algorithm was much more difficult to implement.

The method of bisection involved development encountereding a large amount of bugs and edge cases which slowed down progress. This was mainly due to the maths involved and making sure everything lined up perfectly, otherwise bugs would appear normally many steps ahead, making them time consuming to track down - such as if the bisection passed through the corner of the canvas, or a polygon touched the edge of the canvas in more than one place. Floating point numbers are difficult to compare and say if they are equal, forcing us to use a large amount of precision when calculating points and

lines. This resulted in a large amount of computing power being used for these comparisons, that slowed down the algorithm considerably. All of this considered lead to more team members assigned to this side of the algorithm. We solved this by using our own version of of a margined big decimal, which compared by checking if two numbers were with in a small proximity of each other. To prevent this problem again, we could have spent a longer time on planning for these cases in our implementation, avoiding the need to track them down.

Another area that could have been made more efficient was the first week of implementation. It was known that the base algorithm was the highest priority as most of the other areas rely on it to the point where work could not begin until a solid foundation had been created. This meant that all members of the team were working on the same code simultaneously. Although segmenting was attempted by separating each method and assign it to others, it was only due to exceptional team coordination and excellent communication that minimal issues were created. As the project expanded, the team segregated naturally into three groups, which allowed more efficient work and with confidence that there would be no work overlap.

9 Individual - Michael Chaharbaghi

During the course of this team project my primary role has been to help create, with the help of Andy, a functional and attractive GUI for the algorithm to be displayed in. Having started off creating a base welcome and canvas panel myself, I am pretty happy and proud that they both remain as part of the underlying design and code for the final product, albeit with significant refactoring and tweaks. After my base was created, me and Andy worked closely together to perfect the GUI, ironing out any bugs we found and finding ways to tweak it to make it as functional and as high a quality as possible. Andy and I worked well together I feel as a mini-team if you will, to build and end up with, what I think, is a very elegant high quality GUI for our Voronoi Generator.

Once we had a GUI we were happy with, I moved on to helping with the refactoring of the GUI package classes. Although this is only 1 section of the overall product, it turns out that some of the classes in here make it quite a substantial section which certainly took its time to refactor.

The final workload I was assigned as part of this project is a few sections of this report as well as a section on the presentation which I was more than happy and willing to take on. A portion of the MVC section along with the introduction was my designated sections within the report and, moving on to the presentation, the target audience section was mine to present.

I'll be honest and say that this team project has surpassed the initially quite low expectations I held coming in to the start of the term. Having met my team the first few times, I knew my expectations were happily not going to be realised and that everything was going to go well. We all got on pretty much instantly, we had no conflicts that I can think of, and overall we just worked pretty effectively and efficiently as a team. The project has helped me further enhance my team working ability and allowed me to develop and utilise various software engineering practices I learnt about in my first year including various agile programming techniques such as pair programming. Overall I feel this module has been a positive experience which I am glad to have participated in.

10 Individual - Andy Chen

During the creation of our program I have been mainly working on GUI with Michael, I worked on the design on trying to make the GUI to be more user friendly, we decided to add buttons and tooltips that provide information of different components of our program helping the user to understand what the components do, something that Michael and I have been working on till the completion of the project. Furthermore, I have added a combo box for language options and became responsible of translating the program to Spanish and Chinese.

After the prototype and taking Manfred advice, I have added the function of making the font size resizable and putting the dots manually. Then after the demo presentation between groups, we have taken the advice of making the font size larger and added the Tips panel.

I also worked on the appearance of the GUI with Michael, I made a Gif with Photoshop for the welcome panel of the GUI, found a proper theme for our program and some appearance design like the color of the buttons and panel.

Furthermore, during the last state of our program I translated everything that was not translated yet and made sure that the translation is correct and understandable. Additionally, Michael and I worked on improving the user experience with the current program, one thing that I have fixed is maximizing the size of the canvas panel in the visualization panel; moreover, making the canvas panel of the manual frame resizable and made the reproduction of the visual panel to be controlled by mouse click or pressing the space bar.

I have had a very good experience with this team, since everything was very clearly distributed, everyone worked really hard and we have had a very nice atmosphere between us. The thing that I really liked is that everything was on time so when one group finished one thing we could try it out with the GUI we made. Furthermore, this is the first time that I create a program with a team whichs topic is completely chosen by us and the feeling of reliance on my teammates and being relied really push me to work.

In conclusion, this has been a great and very useful experience, and I am very happy with my team and with the work we have done together.

11 Individual - Sam Durdy

This project has not been a negative experience for me and if anything, a slightly positive one. My team worked well together, and everybody pulled their weight. There were no conflicts between the team members and everybody was friendly, reliable and attentive throughout the project.

During the project I mostly worked on the back end. I started off by producing a very rough prototype on my own, which didn't work whatsoever but did produce a solid basis for the prototype moving forward. Very little, if any, of the code from my initial draft remains in the project unmodified though the structure of the code remains roughly similar. This got the ball rolling, which, following that metaphor, continued to roll at a steady pace for the remainder of the term. From there the team worked together to get to the prototype. Huw, Mitchell and I focused on the back end of the code, namely the model, in the model view and controller and managed to iron out many of the bugs before the prototype presentation.

After the prototype presentation we spent a week re-factoring and redesigning the project file structure as well as updating trello, and planning the remaining time we had. This was a good decision as it allowed us to work efficiently moving forward. After that week we split back up and Mitchell, Huw and I continued to work on fixing bugs as Michael, Andy and Misha continued to gradually add more features to the software.

Once all known bugs had been removed, I moved on to doing some refactoring, as well as writing the explanations for the algorithm in step by step mode. From here I started the report, in which I worked on the software engineering section, which I also worked on for the presentation.

12 Individual - Thomas Huw Harris

I found this project to be a positive experience, because I gained a lot of valuable experience working in a team, as well as learning how to use a number of new technologies, such as SVN and Maven. No conflicts came up through the duration of the project, and we all worked well together.

At the start of the project, I helped to develop the base algorithm, I helped to debug some problems in the code, and worked on adding in JUnit tests for the algorithm. Following the prototype presentation, the team split up into groups, and I was part of the group working on the 'model'. We initially began rewriting parts of the algorithm so that certain edge cases did not cause errors, for example intersecting lines hitting the corner of a polygon. The majority of our time was spent on looking for and fixing bugs, and working on test cases to make sure that these fixes worked correctly.

After removing all of the bugs we could find from the code, I began working on adding JavaDocs to the model classes, as well as commenting the more complex methods so that they could be more easily understood. I then went on to work on the test report, including moving our list of test cases over to latex. Finally, I started on the final report, where I wrote the 'Team Work' section, which I also talked about in the presentation.

13 Individual - Mitchell Savage

Throughout our project our team worked incredibly well and closely together, and as a result many of the key milestone and contributions in the project were the result of smaller collaborations between many people.I really enjoyed working with my team, and strongly believe that we couldn't have worked any better together and maximised the amount of work not done through our synergy

For the specification, we split up the work into sections and each wrote a preliminary draft for it - my section being the system features. We then met up and integrated the parts together, and edited the document as a whole, resulting in everyone checking and changing each section.

Despite having to change our products scope at the last minute, we got to work on the software quickly. Sam wrote a rough start to the program, and we all began working a basis for the algorithm, as we couldn't realistically split up the work into section at this early stage. Once we had an algorithm that could put 1-2 points on the canvas (albeit unreliability) we then split up into our teams, which i worked mostly on the model, attempting to get correct behaviour. I spent most of my time at this stage on tracking down the major bugs in the logic of the algorithm and coming up with solutions, for example when our trim function was duplicating lines or the edge case of a polygon touching the sides of the canvas twice. I would then work together in a pair/trio with Sam and Huw on implementing a fix.

Once we had an acceptable model for the prototype presentation, I worked on re-factoring, cleaning up and commenting the code. I also organised the prototype presentation - simply because everyone else was working on something else. We didn't have any concept of a leader throughout any stage of the project, as people took responsibility for areas as they cropped up. Towards the end of the project I continued working mainly together with Sam and Huw on bug fixes, and extending the model to work with the additional functionality we were adding, along with implementing some myself - such as square and hex distributions. My final contributions to the project were the opening, demo, evaluation and closing of the final presentation, and the evaluation and summary of the report.

I was glad that I got to work mainly on debugging and rewriting other peoples code, as understanding others work was the area I thought I would have struggled the most with before this project.

14 Individual - Misha Wagner

Before our team started development, I was active in the design of the plan. I participated in the original research for the product, pitching the world-generation idea. After the sepcification was completed and the project changed to a subset of it's original, myself and the rest of the current model team worked on the creating the algorithm. The work was segmented across all team members. I was responsible for the first working version of the `addPoint` method, which brings all other components of the algorithm together. In the model, I also worked on other functions such as `trim` , `getJavaPolygon`, and `isComplete`. These were all large parts of the algorithm, and although for the second part of the project I worked primarily on the `controller` package, I had a significant part in the model at the beginning of the project.

We were getting many issues with precision of floating point numbers, and this became apparent to me quickly. Because of this, I took the initiative to go through our project, and convert our code to use the Java `BigDecimal` library. This required a lot of work: all mathematics across the project had to be changed to use `add`, `subtract` etc. methods, rather than +, -, *, / operators. I also wrote wrappers around the new decimals for points and lines. This lead to cleaner code, as a lot of the line arithmetic was moved to these new classes, cleaning up utility classes significantly.

Around halfway through the project, I moved to work on the controller section. Although the feedback loop for passing input from the view to the model was simple, this meant I had the opportunity to design the `Holder` class, which stops the algorithm from progressing on time-restricted modes. This was something that could lead to high-coupling in the project, but I managed to avoid this by the use of the `Holder` class.

Multi-lingual support before I worked on it was functional, but had a lot of duplicated code. This was refactored by me, through the use of a JSON file that links texts to language codes. I also believe that this sped up development for the GUI team, as duplicate code can lead to many strange bugs, and this is something I eliminated.

Overall, I believe I pulled my weight across the project. I am happy with my own, and each of my team member's contributions. I believe that at no point in the project did anyone drop below our expectations.

15 Summary

In conclusion, our team project gave us the chance to work on a larger scale product that could only be achieved through working as a team and stepping out of our individual comfort zones into an environment that modelled a real world industry workplace. The project as a whole has improved our knowledge of how to work together effectively and efficiently as a group that aims towards a common goal. It has also allowed us to get some experience with using tools such as Subversion and Trello, which are becoming increasingly common place in the software industry, and as we used them we saw how they can be integrated and become key tools in larger applications of software.

Using an agile methodology meant that we got to put into practice software design techniques that we have learnt in software engineering, which benefited our project in the form of sustainable development and the ability to adapt to challenges or changes. This helped improve our GUI and usability design, as we could constantly revise and shape our end product according to new ideas or user feedback.

We began our project with the intention of making a quality software product that fulfilled our brief, and through the techniques and approaches shown above, we believe that our final product is the accomplishment of these objectives, while gaining valuable, workplace-like experience.

References

[1] Wolfram Alpha, *Voronoi Diagrams*,
Online at `http://mathworld.wolfram.com/VoronoiDiagram.html` [last visited 23/03/2016]

[2] Apache Software Foundation, *Apache Maven Project*,
Online at `https://maven.apache.org/` [last visited 23/03/2016]

[3] Quality Open Software, *Simple Logging Facade for Java*,
Online at `http://www.slf4j.org/` [last visited 23/03/2016]

[4] JSON.org, *JavaScript Object Notation*,
Online at `http://www.json.org/` [last visited 23/03/2016]

[5] Beck, Kent, et al., *Manifesto for Agile Software Development*,
Online at `http://agilemanifesto.org/` [last visited 23/03/2016]

[6] *Trello*,
Online at `http://www.trello.com/` [last visited 23/03/2016]

[7] Apache Software Foundation, *Apache Subversion*,
Online at `https://subversion.apache.org/` [last visited 23/03/2016]

16 Appendix A: UML Diagrams

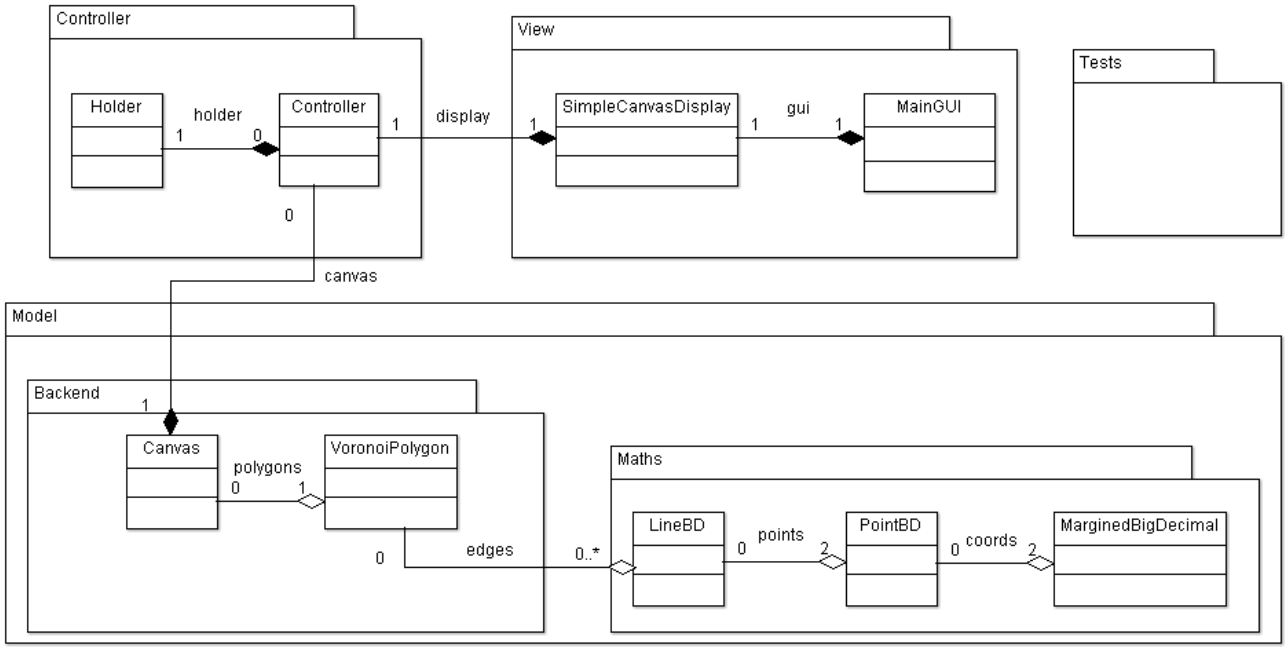

Figure 9: A simple UML diagram created following the prototype demonstration. This diagram was used to plan out how the code would be structured for the remainder of the project

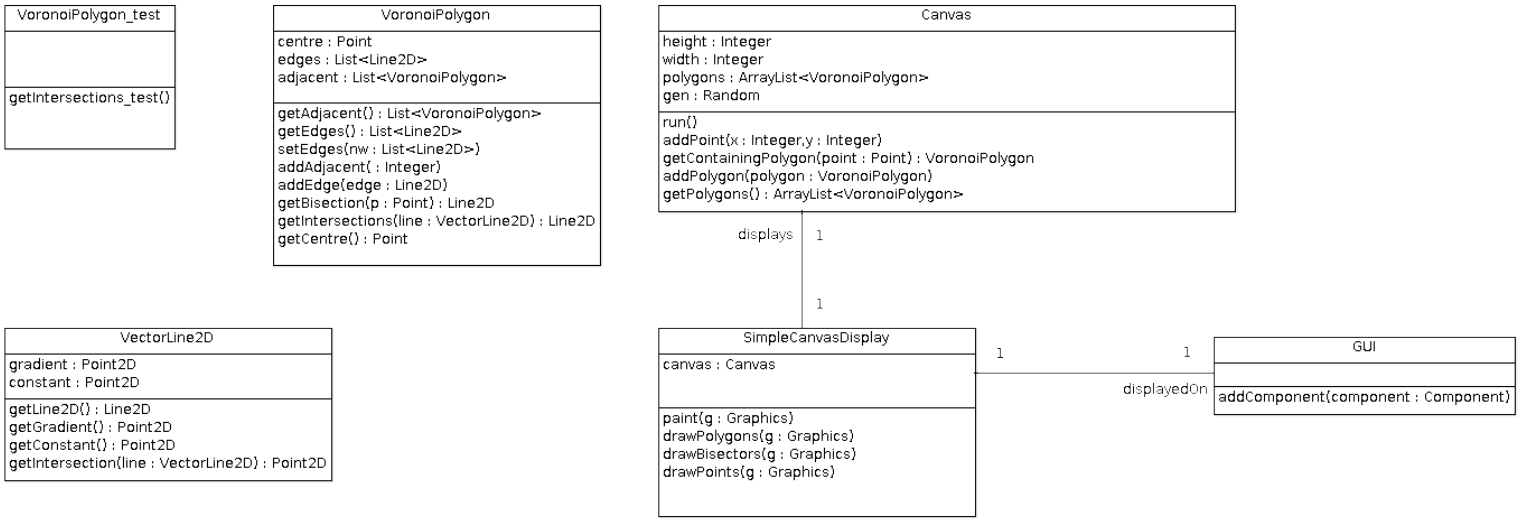

Figure 10: The first iteration of the class diagram, last modified 04/02/2016

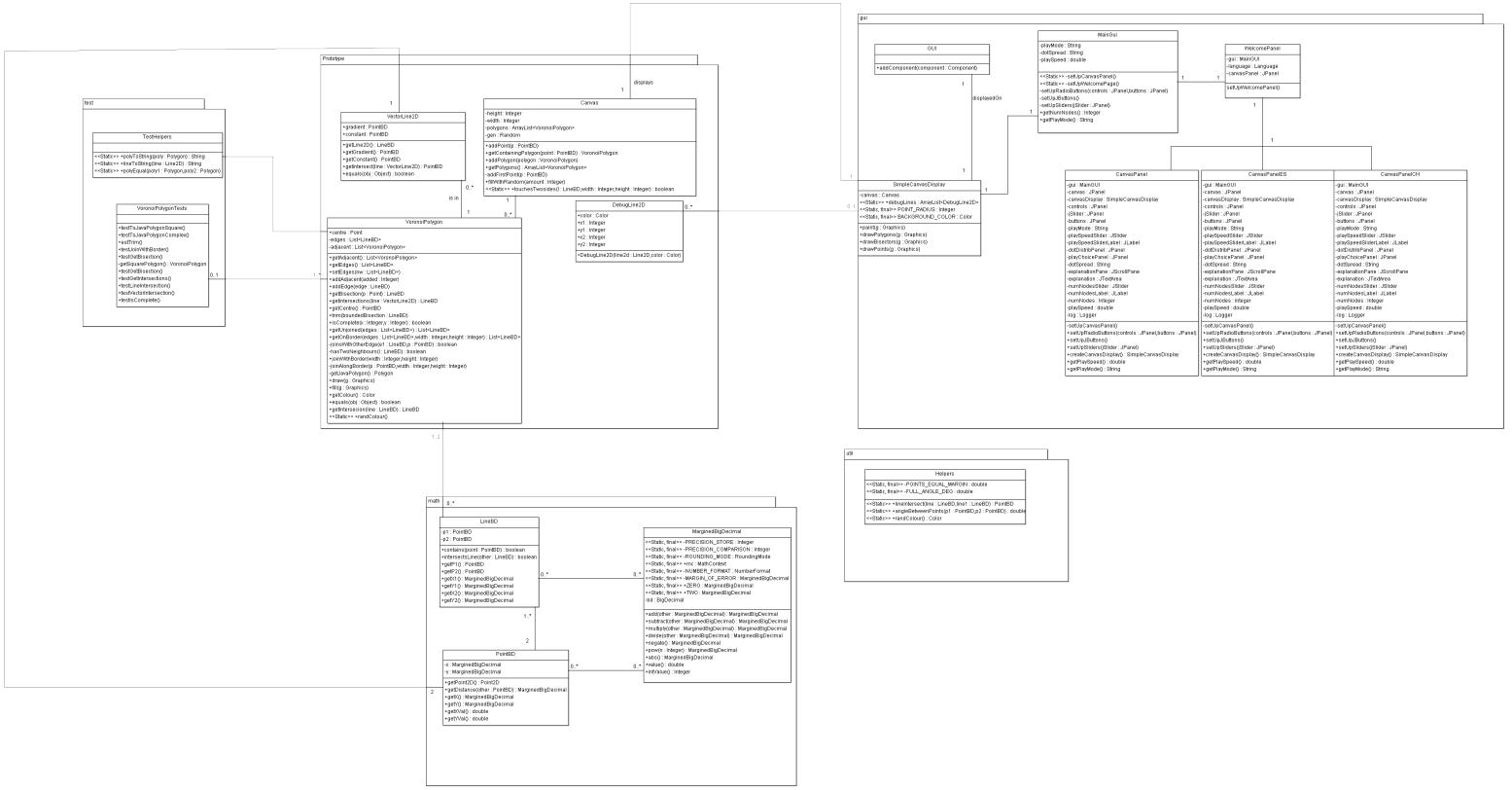

Figure 11: The second iteration of the class diagram, last modified 18/02/2016

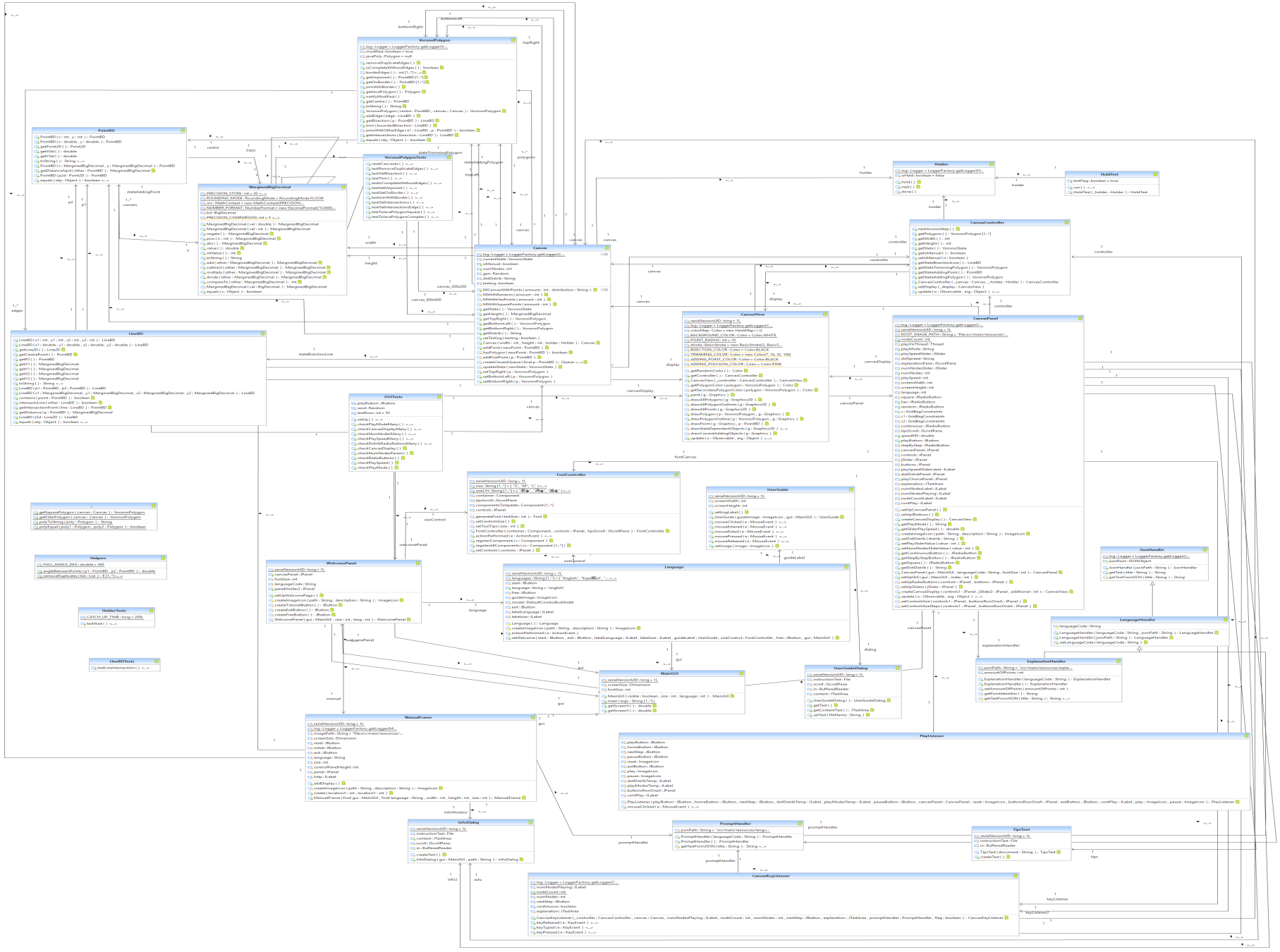

Figure 12: The final class structure