Even Semester (2020)

Assignment Cover Letter

(Group Assignment)

Student Names :	**Aric Hernando**	**2301890314**
	Bryan Putra S.	**2301890983**
	Nicholas Arthur	**2301890301**
Course Code :	**COMP6571**	**Course Name : Data Structures and Algorithms**
Class :	**B2AC**	**Name of Lecturer : Andreas Kurniawan**
Major :	**Computer Science**	
Title of Assignment :	**Dijkstra's Algorithm Visualisation**	
Type of Assignment :	**Final Project**	
Due Date :	**18 - 6 - 2020**	**Submission Date : 18 - 6 - 2020**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Dijkstra’s Algorithm Visualization

Final Project Report

I. Problem Description

So in our project, we decided as a group to try and make a visualization for a famous pathfinding algorithm called Dijkstra's Algorithm. Dijkstra's Algorithm is an algorithm for finding the shortest path between nodes in a graph. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, we can picture the nodes in the graph as cities or malls or a place we want to go to, and the edges being the roads represent driving distances between places connected by a direct road. Another use for Dijkstra is for general network application. For example, we can use this to determine the most efficient path to send a packet of data from one server to another in a network. All of these are applications of Dijkstra’s pathfinding algorithm. While Dijkstra’s Algorithm is only one of many pathfinding algorithms, we see value in creating a project to visualise and implement this algorithm. We believe that this will help us in the future either in graph theory, or pathfinding algorithms in general.

Our goal in making this project is to visualize the algorithm with the simple example above in mind using data structures that we have been taught on how to implement them which are Priority Queue, Linked List and Unordered set.

II. Proposed Alternative Data Structures

As we are implementing Dijkstra’s pathfinding algorithm, we need several data structures to fully implement it. The data structures each fulfill different roles in running the algorithm. First, you need an algorithm to store each node that was added to the graph, this will serve a purpose of enabling the code to easily index and manipulate the data that is inside of each node. Second, you need to create an ordering system in which the program is able to be aware of which node needs to be checked first. This can be done in mainly two ways.

One, the ordering can be done through the variable that is stored inside of the node. In our implementation, this variable is the integer ‘distanceFromStart’. Using this, we can loop through all of the nodes in the list, and then check for the lowest value of them all. This is highly inefficient because one would need to search through every node every time the program needs to find the next node to search through. When found, the

algorithm also needs to check if the node has been visited or node, making this approach not ideal for the implementation of this algorithm.

Second, the ordering can be implemented using a priority queue. A priority queue will solve many of the problems that are present by the first solution. It will solve the problem of searching through every node, resulting in faster time for the algorithm to find the next node to check. The priority queue can be implemented in two ways, the most common way is to create a binary tree, and insert the node and priority through that tree. This has the benefit of being extremely efficient, but has the downside of being more complex to implement and update the tree. Therefore, we used the alternative method of implementation of priority queue, that is, linked list. Using linked list, the program will automatically queue the node based on the priority. We can also easily implement the process for when the program needs to update the order.

III. Data Structures

- Priority Queue

Priority queue is a data structure similar to a regular queue however, with priority queue objects inside the list have their own respective priorities. In priority queue objects with a higher priority will be checked first. The advantage of having a priority queue instead of a normal queue is that using a priority queue, you will be allowed to set a higher priority to objects that are more important and set a lower priority to objects that are less important.

```
37      //  Enqueue a node into the priority queue
38      void enqueue(node<string>* n){
39      //  Checks if node has been visited or not
40          if(visitedNodes.find(n) == visitedNodes.end()){
41              visitedNodes.insert(n);
42              pq.add(n);
43          }
44      }
```

This advantage is implemented in our project to visit the nodes that have a higher priority, the node with the highest priority is the start node , the next nodes that have a higher priority are nodes that have the shortest distance from the start and subsequently after that.

```
31          if (head == NULL || p < head->priority){
32  //              Inserts a node so that it becomes the first
33              t->next = head;
34              head = t;
35          }
```

- Linked List

Linked List is a linear data structure where each element is a separate object. Each element (node) has 2 items, the data inside a node and a reference to the next node(pointer). The last node in the list has a reference to null. The starting point of a linked list is called the head of the list. The head is not a separate node, but a reference to the first node in the list. If the list is empty, the head will be a null reference. We implemented this data structure for our nodes in the graph so we can iterate through them dynamically unlike arrays and we can even use our own functions that we can make ourselves according to how we want to implement the data structures to the visualization.

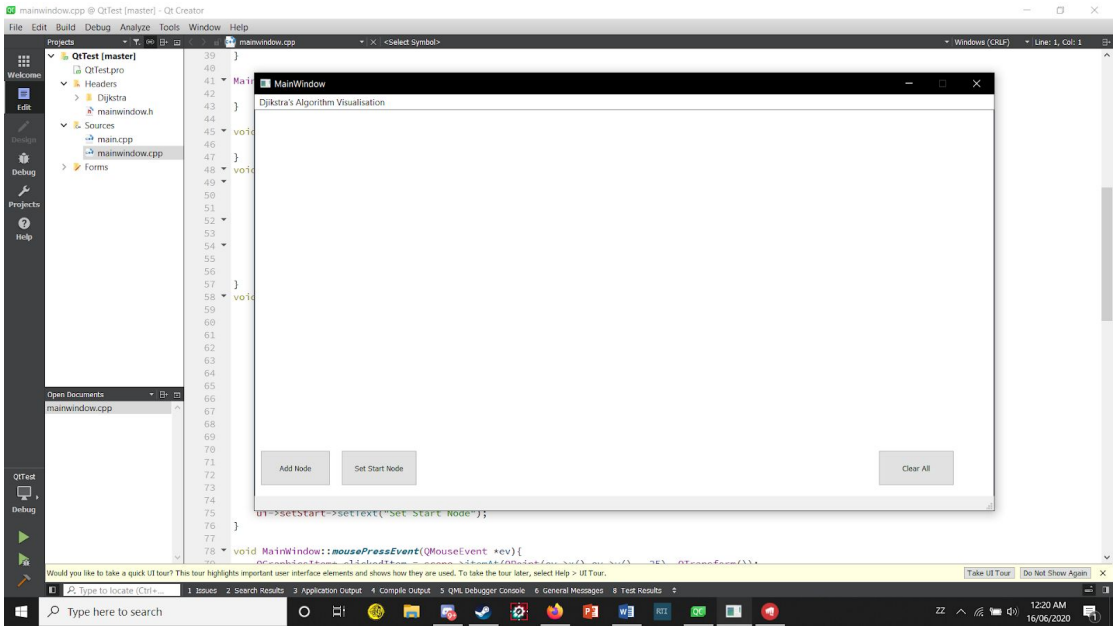
- Unordered Set

An unordered set is a data structure based on a regular hash table, an unordered set has an advantage over a regular set based on it's time complexity. We use this data structure to keep track of the visited nodes. We used set because we can guarantee that all of the data will be unique (As there can be only one copy of a certain node). The unordered set is used because we will often use the built-in 'find' function. This function will return an iterator that points to the element, in case the element didn't exist, it will return the last element of the set. Therefore, we can check the uniqueness of an element by running the code snippet below.

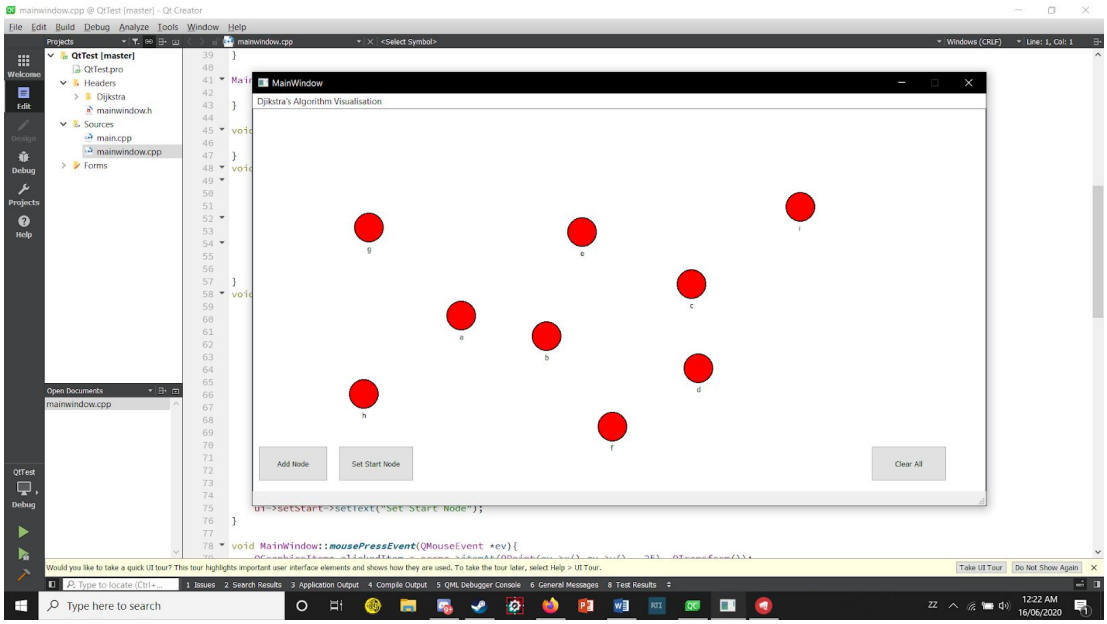
```
37      //  Enqueue a node into the priority queue
38      void enqueue(node<string>* n){
39      //  Checks if node has been visited or not
40          if(visitedNodes.find(n) == visitedNodes.end()){
41              visitedNodes.insert(n);
42              pq.add(n);
43          }
44      }
```

This code will check if an element is already inside the set or not, if not, it will then insert that element to the priority queue, and the visited nodes. By using unordered set, we can ensure a constant lookup time when finding and inserting an element to the set.

IV. Program Manual and Results

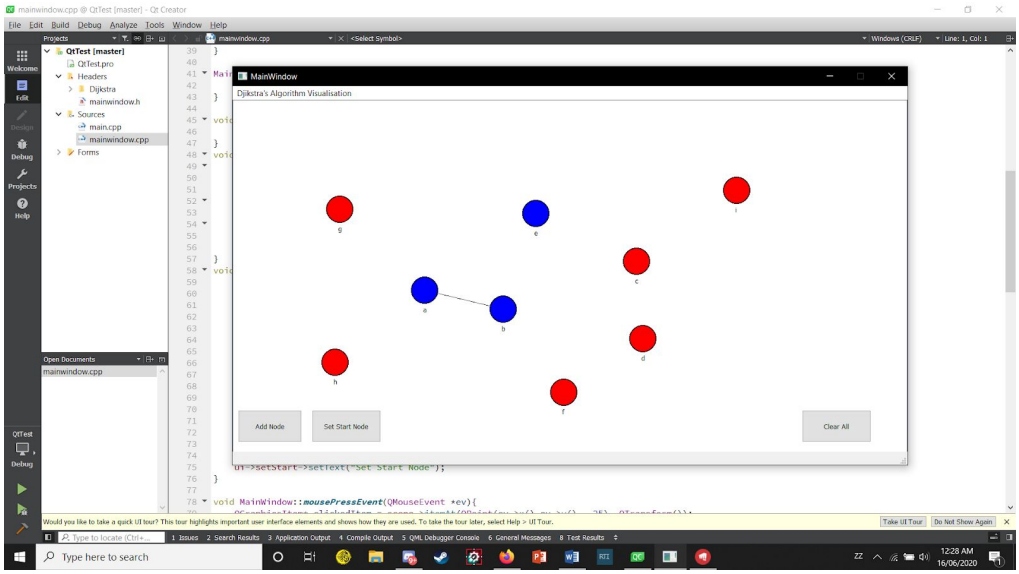


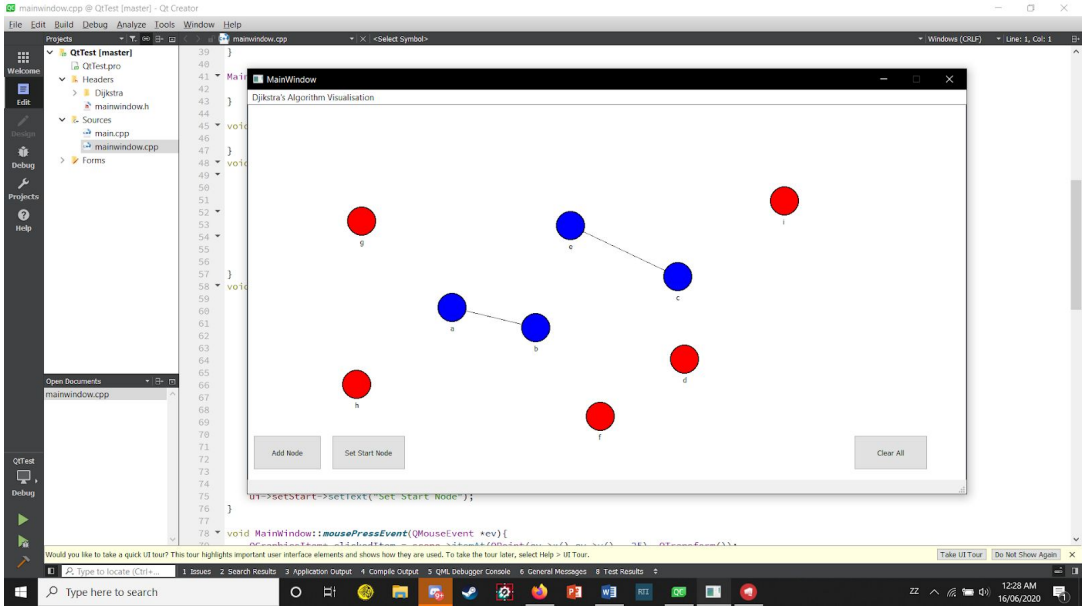
The program starts with an empty space of a window with 3 buttons. So we start by adding nodes to the screen wherever we like, we do that by clicking the add button first then clicking anywhere on the screen to add a new node in our case it's in a form of an ellipse.



As you can see each node has an identifier which are the alphabets a b c d etc. After adding the nodes, we can connect two nodes by clicking a node first and then click

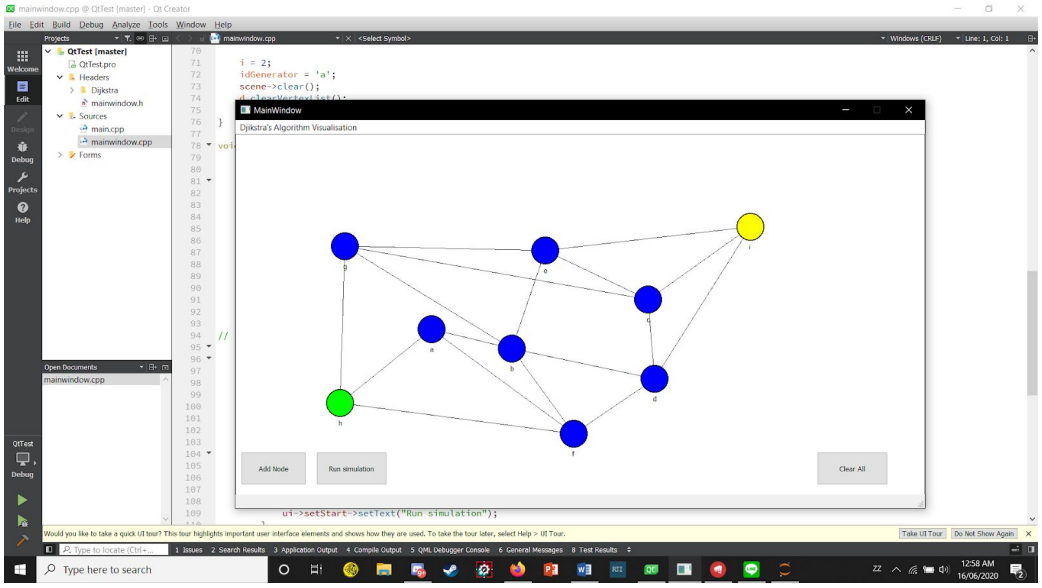
a second node which is the node that we want to connect the first clicked node to. When we click on a node, the node colour will change indicating that we are selecting a node to connect to another node.



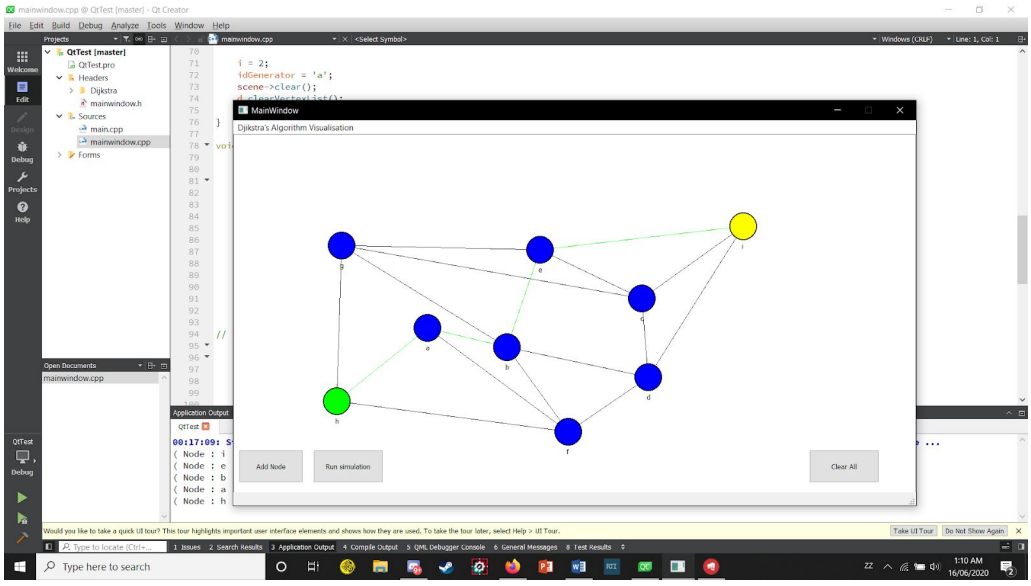


After connecting the nodes according to your preference to try the program, we can use the Set Start Node button to set the source position to a node. Setting the a source position on a node will make the node turn colour to green indicating it is the start node. Then we can press the button again which now says Set Destination Node to basically set a node to be the destination position which we will go to in the program. The indicator for a node to be the destination node is that the colour of the node will change to yellow.

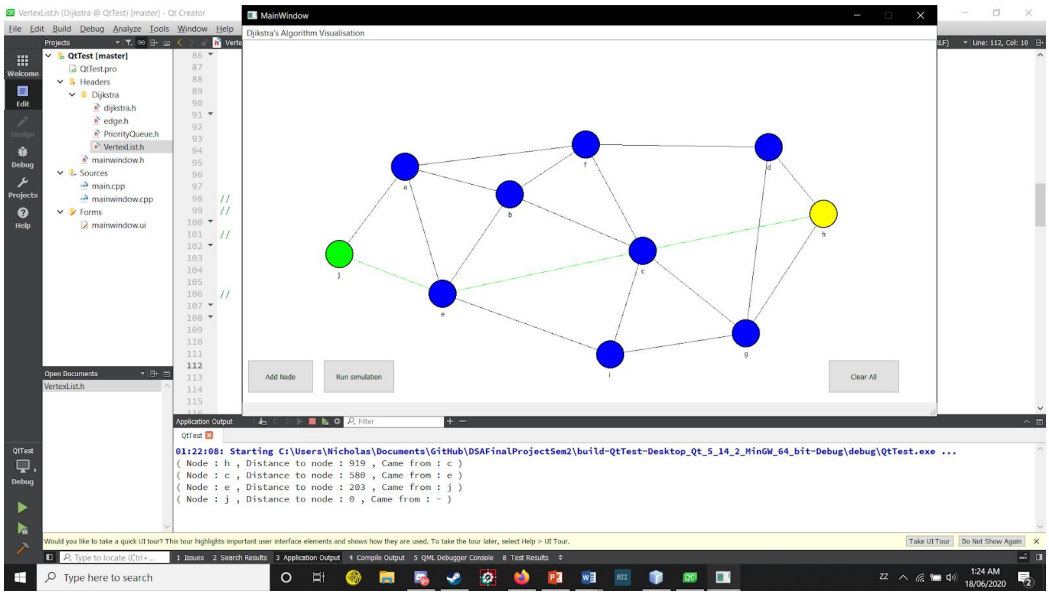
So the green coloured node is the start node and the yellow coloured node is the destination / end node.



After setting the start and the destination node, we can now do the dijkstra's pathfinding algorithm to find the nearest path from start to end from all of the lines(edges) that are available. We can picture the lines as roads that we can walk to from a point to a point. To do that, we can press the Run simulation button to execute the algorithm.



Finally the results are the lines that have changed colour to green. They are the shortest road/path that the green node can use to reach the yellow node after checking the other paths that are available around the nodes.



Not only that, we have the data from the nodes that we passed through from start to end because the algorithm keeps track of all the nodes that we have been through. The information is printed in the console line.

We can also clear the whole thing to add nodes with new paths/roads and different start/end according to our preference with the button clear all.

V. Demo and GitHub Link

https://github.com/Aric-prog/DSAFinalProjectSem2
https://youtu.be/-ekt8lI2YLc

VI. Reflection / Task Division

- **Aric Hernando:**

  I've often wondered about how computers used to do all sorts of things. Pathfinding is one of them. How in the world does one compute the shortest path from A to B. It's always been an idle thought at the back of my head. But when I started learning more about coding in general, it still baffles me how it works, to me it was basically the equivalent of magic. This final project gave me the opportunity to finally find out in full how pathfinding works. Before starting the

project, I had doubts whether we could implement this or not, but as we started inching slowly towards our goal, what seemed magical became increasingly technical, in other words, it slowly started to become somewhat viable.

For the division of labour, at the start, we agreed that I should do the backend logic (The Dijkstra's Algorithm and its data structures), while the rest of the team will divert their efforts more on the front end GUI. So we initially started working on this project completely separately. It was only until the end when they were struggling that i had to help them with the GUI.

- **Bryan Putra:**

  At the start of this project, I was kind of scared of what's going to happen because I was not that confident with my c++. Then our leader suggested the idea of visualizing a pathfinding algorithm which is the dijkstra's algorithm. I watched an explanation from youtube on how the algorithm works, then i found out that it was actually cool to see the algorithm running visually and it made me curious so i went along.

  We divided the tasks and I ended up with the GUI parts along with Nicholas. I never tried GUI on c++ before so I had to search what to make it with. We ended up agreeing on using Qt because one of the other teams were not recommending the wxWidgets that we found. Then i started searching for tutorials and guides on how to use this platform but honestly, the things that i learned are not that much because the tutorials was not that useful for the things that we want to do in the program, so i had to read the official qt documents regarding their items and stuff, however having pointers on the items like ellipses and lines in the qt made our it way easier for the implementation to work. My part was to make lines and link the nodes together to form a graph. I struggled a bit on my part so my leader had to help me on the way and it worked out pretty well i would say.

  This experience I would say is a pretty good and bad experience at the same time for me. The bad part was that I had to focus on different final projects at the same time that made me very stressed and not calm during the 2 weeks holiday but if I look at it in a good way I guess I learned a lot from this experience, like improving time management and priorities.

- **Nicholas Arthur:**

  When I started this project I only had a vague idea on data structures and the requirements of building a c++ project. We divided the tasks between ourselves and I ended up with making the GUI along with Bryan. Of course I had no clue on what to build a c++ GUI with so I did a google search and the most reasonable choice was Qt, or wxWidgets. We made a team decision and ended up using Qt since it looked like it was more beginner friendly.

  I had to start learning Qt first and searched many specific youtube videos on how to start a Qt project and implement it the way we wanted to. Luckily Qt has a built-in scene builder along with built-in items for the scene builder, among these items were what we needed, a circle and a straight line. Even more luckily Qt's items have their own pointers the moment they are made, this made our job easier to locate a specific circle we wanted. I ended up making the button to make nodes appear on the scene viewer. That's where the specific part of my task ended.

  Throughout this project I learned and experienced many new things. I of course learned Qt and many of it's libraries and functions but also learned how and which project to prioritise first, since we had a few other final projects that needed paying attention to. So of course time prioritisation is a skill that I have developed even more.