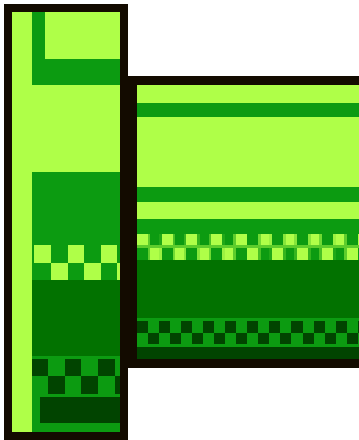




Aalto University / CS-A1121

Pipe Flow

A simulation program for pipe flow



Tamm Hugo

12.05.2023

Contents

1. Personal Information – Henkilötiedot.....	2
2. Introduction - Yleiskuvaus.....	2
3. Usage – Käyttöohje	2
4. External libraries – Ulkoiset kirjastot	4
5. Program structure – Ohjelman rakenne.....	4
6. Algorithms – Algoritmit.....	5
7. Data structures – Tietorakenteet	6
8. Files – Tiedostot	6
9. Testing – Testaus	6
10. Known flaws and bugs of the program – Ohjelman tunnetut puttee ja viat	6
11. Best and weakest points – Parhaat ja heikoimmat kohdat	7
12. Deviations from the plan – Poikkeamat suunnitelmasta	7
13. Work order and schedule – Työjärjestys ja aikataulu	8
14. Evaluation of the final result – Arvio lopputuloksesta	8
15. References – Viitteet	9
16. Attachments – Liitteet.....	9

Pipeflow – Hugo Tamm

1. Personal Information – Henkilötiedot

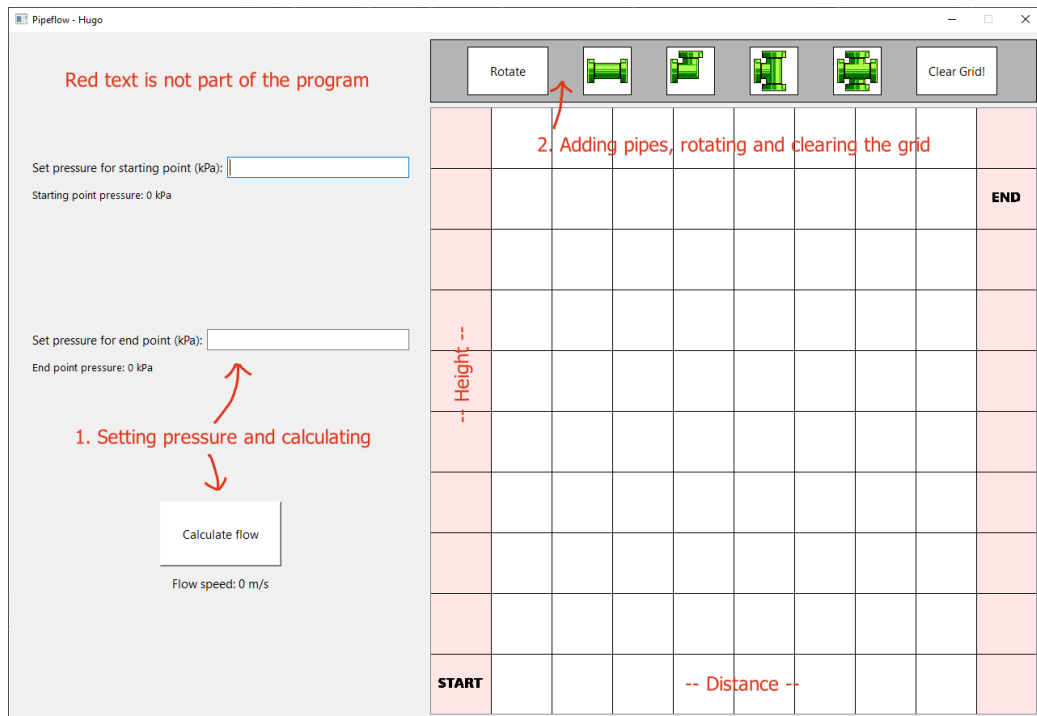
- Hugo Tamm, 1020296, Energia- ja konetekniikka, 12.05.2023

2. Introduction - Yleiskuvaus

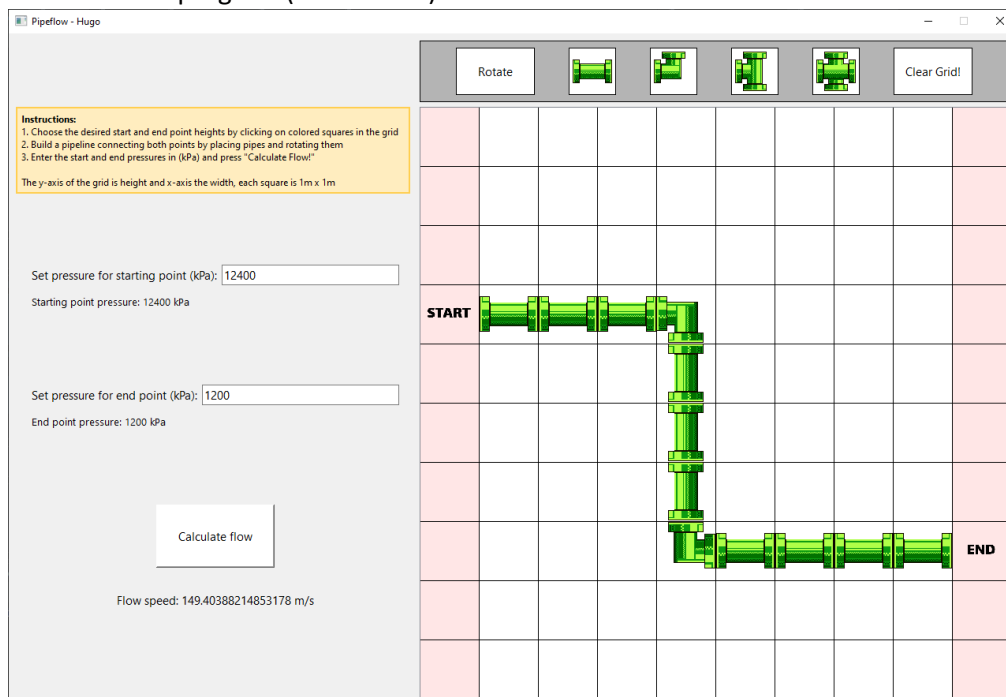
- The main purpose of the program created is to simulate pipe flow, where you can set the start and end points to the wanted height and connect them with pipes. The points are primarily connected with straight pipes and 90-degree bend pipes, but other variations are also available despite not really serving any purpose other than to just demonstrate that they can be added if needed.
- The goal was to reach a demanding level of difficulty and exceed it, for example, by creating a small pipe connecting game in the program. Using Bernoulli's law, you can calculate the speed of the liquid (in this case water) inside the pipe. In terms of versatility, the initial plan was to add a lot of pipe variants, i.e. different thicknesses and shapes, but in the end, only different shapes got implemented and no thickness variations.
- The program was implemented with a graphical user interface, which allows for the user to input values (for pipe pressure) and see the result of the calculations done using those values. The main window consists of a large grid with a start and an end point, both of which can be moved by the user. The window has its own compartment for pipes that can be selected and placed in the grid by clicking on an empty cell. Removing pipes can be done by clicking on any placed pipe in the grid.
- In the end, the project should meet all the medium requirements and a bit above into the hard ones, like the ability to move the starting and the ending points. Despite lacking in the calculation of the flow speed, it still has a proper placeholder for a better equation to be implemented in the future. The option to add a pump to the pipe system and other elements is not currently implemented, but the foundation for adding more elements is there, with a bit of modifications to the current code.

3. Usage – Käyttöohje

- The program is used by choosing the right heights for both start and end points, then building a wanted pipeline through the grid connecting both the points to each other, and finally entering desired pressure values in kPa into the input fields and pressing the "Calculate flow"-button.
- Picture of the program (08.05.2023)



- Picture of the program (11.05.2023)



- As marked in the picture, tools for adding pipes, rotating them, and clearing the grid are above the grid. The horizontal cells in the grid represent the distance and vertical cells represent height, which are used for the calculations. Now currently the size of the grid is fixed to a size of 750x750 (pixels), where each cell is 75x75 (pixels). This could be expanded to allow for the user to set any size for the grid as it is designed to be responsive, but the layout of the main window breaks when used with certain values. The current size of the window is 1280x850, which is a very arbitrary size yes.

- No pipe game has been implemented due to lack of time, so there is no button or tab to access it, despite initial plans to create one.

4. External libraries – Ulkoiset kirjastot

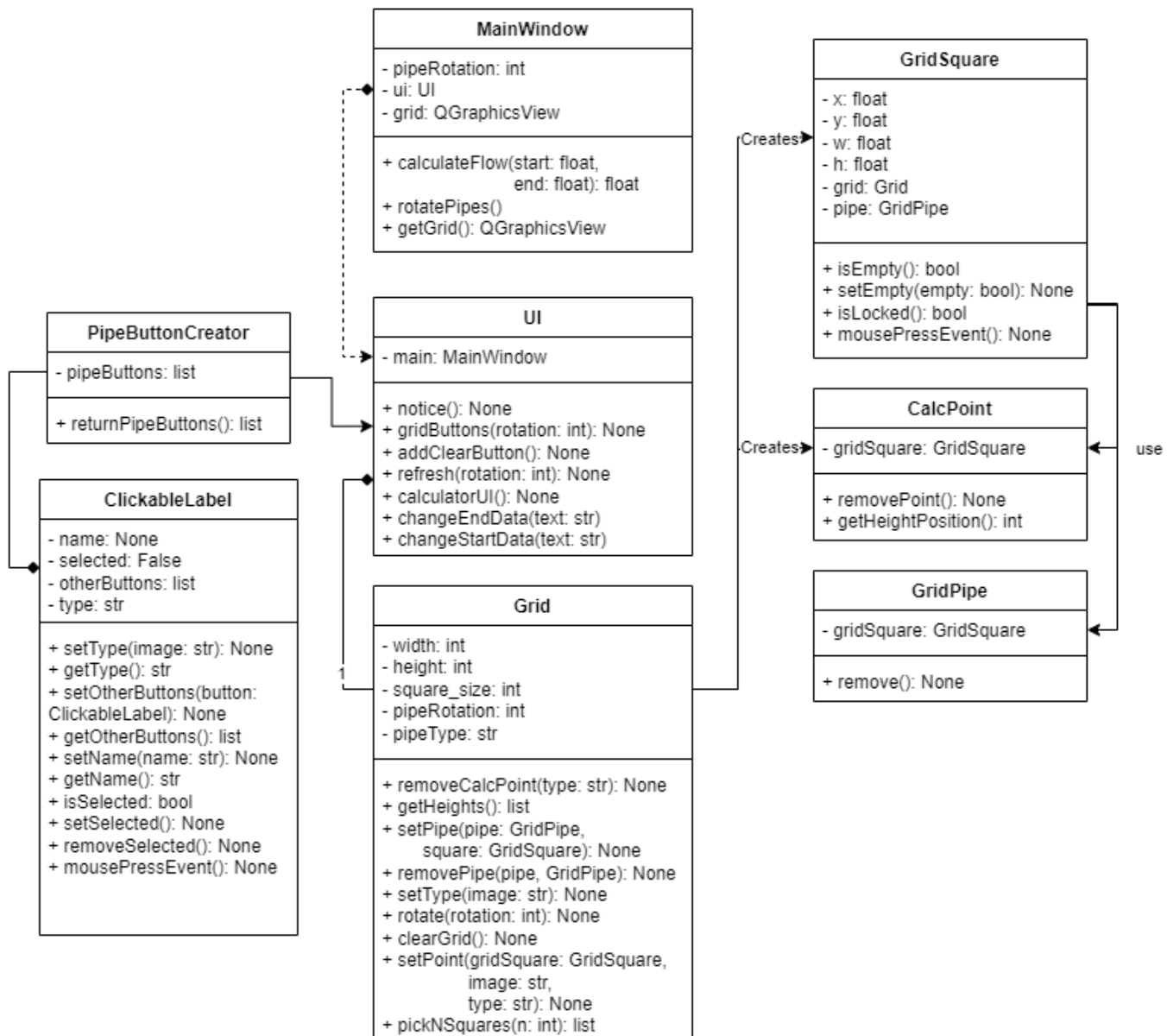
- External libraries used in the program are: PyQt6, math, random
- PyQt6 being the main library and focus of the project.
- Math was used for the sole purpose of flooring a few numbers and using `math.pi` and `math.pow`.
- Random was used to choose a random element from a list and to shuffle a list, using `Random.choice()` and `Random.shuffle()`. Both methods are in a way, an essential part of python, since picking a random number will always rely on some kind of module.
- Sys was imported for the use in starting the app, being used only once:
 - `app = QApplication(sys.argv)`

5. Program structure – Ohjelman rakenne

- The *MainWindow* class serves as the foundation for the entire program, bringing all functions together into a coherent unit. It initializes the *UI*, calculates flow speed based on user input, and handles some grid and *UI* functions such as rotating pipes. Although some of these methods could be moved to the *UI* class, having them available in *MainWindow* makes them easily accessible from anywhere, and helps to balance out the *UI* class, which would otherwise be over 200 lines long.
- The *UI* class plays an important role in bringing the code to life, as it initializes the *Grid* object, controls elements in the grid, and displays information to the user.
- The user interface of the program utilizes the *PipeButtonCreator* class, which is responsible for creating clickable buttons with images. This class relies on the *ClickableLabel* class, which is inherited from the *QLabel* class in the PyQt library.
- The *Grid* is a custom class that inherits from the *QGraphicsView* and is responsible for creating and managing a grid of *GridSquare* objects. It is an essential part of the project, taking up half of the window. It allows for the placement of pipes, rotating them as well as setting start and end points to be used in the calculations. The grid is built with *GridSquare* objects, which are essentially empty cells waiting to be filled by *GridPipe* objects or *CalcPoint* objects, which represent start and end points.
- The *GridPipe* class is also a custom class that inherits from the *QGraphicsPixmapItem* class. It is used to create pipe objects that are placed on the grid squares in the program. When a *GridPipe* object is created, it is passed a *GridSquare* object as a parameter to identify which square it belongs to. The *GridPipe* class has a `remove` method that sets the `empty` property of the *GridSquare* object to `True`, which indicates that pipe should be removed from the grid square.

- The purpose of the *GridSquare* class is to represent a single square on a grid. It inherits from *QGraphicsRectItem* and has properties such as its position on the grid, whether it is empty or locked, and a *GridPipe* object associated with it. It also has methods to check and set its emptiness and locked status. The *mousePressEvent* method is overridden to handle clicks on the square, which can result in placing or removing a pipe from the grid, setting start or end points for calculations, and updating the UI accordingly.

Down below is a UML diagram of the project.



6. Algorithms – Algoritmit

- The *MainWindow* class contains a mathematical computation to determine the flow speed of a fluid through pipes. This involves using the Bernoulli equation to calculate the pressure

difference between the start and end points of the pipes, and then using this to calculate the flow speed.

- The equation below is the extended Bernoulli's equation which is very useful in solving most fluid flow problems. It takes into account the energy losses due to friction and other non-conservative forces.

$$\frac{p_1}{\rho g} + \frac{1v_1^2}{2g} + h_1 + H_{pump} = \frac{p_2}{\rho g} + \frac{1v_2^2}{2g} + h_2 + H_{friction}$$

Currently the equation used to calculate the flow speed is:

$$v = \frac{(2(P_1 - P_2) + 2g * (h_1 - h_2) + H_f)}{\rho \pi A}$$

Something feels off about this, but at least it's not giving 700 m/s speeds :D

- The *Grid* class contains an algorithm for creating a grid of *GridSquare* objects by iterating over the width and height of the grid and creating a *GridSquare* object for each position in the grid. It also check whether each square is meant for the start or end points and adds these squares to the respective lists 'self.startPoints' and 'self.endPoints'

7. Data structures – Tietorakenteet

- No custom data structures were necessary for the program. Built-in python lists are primarily used for most things. All lists are mutable, though not all are changed after initialization.

8. Files – Tiedostot

- The program doesn't require any external files to work, everything is built-in and in working order as is. The program doesn't return any files or save results.

9. Testing – Testaus

- Testing of the program is a bit lackluster and non-existent. Currently only the grid is being tested, checking for the right number of pipes after adding them and removing them, as well as testing if the "Clear Grid!"-button works.

10. Known flaws and bugs of the program – Ohjelman tunnetut puttee ja viat

- The program in its current state has a few flaws, mostly related to calculating the flow in the pipes. Some of the flaws are also just missing features that would greatly improve the program.
 1. The flow can be calculated even though the pipeline isn't continuous and connected.
 2. Calculations with the extended Bernoulli's equation is most likely wrong and missing a few key values, like Darcy's friction factor, which is currently just guessed to be around 0.1.

3. The window and the layout are not responsive to screen size and will go out of bounds for screens with resolution smaller than 1920x1080.
4. Clicking the “Rotate” or the “Clear grid” buttons will remove the color from the selected pipe icon, so the user can’t be sure which pipe is currently selected if placed.
5. The pipes in the grid are not used in the calculations and are essentially functionless.

11. Best and weakest points – Parhaat ja heikoimmat kohdat

- Best points:
 1. Adding, removing, and rotating pipes.
 - Getting the pipes to “sync up” with the grid and be able to rotate and well placed was a fun challenge.
 2. The ability to move the start and end points.
 - The colored areas in the grid can be clicked to change the height of the start and end points, which effectively also change the resulting calculations.
 3. Layout.
 - The layout looks kind of good, lining this properly with intricate detail is a bit hard with python, but it turned out okay.
- Weakest points:
 1. Calculations
 - The weakest point is the calculations’ part. It’s highly probable that whatever value is currently returned from the inputs is not correct and won’t be of any practical use.
 2. Pipeline
 - Currently, placing pipes in the grid serves no purpose as they are not taken into consideration when calculating the flow speed.
 - Fixing this would first require an understanding of the extended Bernoulli’s equation and how to manage the pipes properly. Perhaps a linked list could be a good data structure to construct a pipeline where each pipe is aware of the surrounding pipes, or at least the next one after it.
 3. The code itself
 - Navigating around different files and classes feels a bit complicated at times, not being sure how each element is related to one another. This could be improved by perhaps formatting the code a bit better and having a clearer structure of the code.

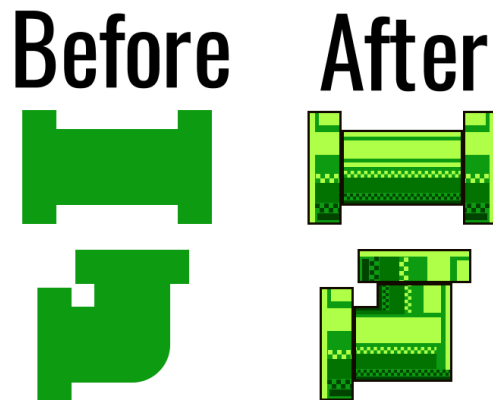
12. Deviations from the plan – Poikkeamat suunnitelmasta

- To be honest, I never opened the initial plan for the program after creating it. I had a plan in my head on how things should turn out to be, which in hindsight was not an efficient way of approaching a bigger project like this. Having other courses on top of Y2 took away some attention from the project, leading to not having enough time to do things properly.

- In terms of schedule, there weren't any significant dates that I was trying to get something done by, I just worked on the project when I had the time for it.
- I tried to approach the creation of the program in a logical way, building from the more essential things such as the window, grid, and pipes, toward the more subtle details like the layout, colors, style around selected buttons etc.

13. Work order and schedule – Työjärjestys ja aikataulu

- I finished the initial project plan on February 23
- Major progress was made around March 12 – 20
 - Added a window, pipes, grid, buttons.
 - Cleaned up the messy code a bit.
 - Worked on the styling and layout.
 - Also improved the initial images for the pipes.



- On March 24, a few typos were fixed and added rest of the pipes.
- April 27, Added UI class and moved most of the user interface related methods there to clean up the main file.
- May 2, added the rotation functionality to pipes and pipe icons on buttons.
- May 9, tried to add Bernoulli's equation to the calculations.
- May 10 – 11, a lot of fixes and maintenance
 - Added moveability to start and end points.
 - Changed equation for calculation.
 - Cleaned up the code.
 - Added comments to necessary sections.

14. Evaluation of the final result – Arvio lopputuloksesta

- The project in its entirety turned out great, but with a few fundamental flaws. The fact that the calculations for the flow speed are incorrect and the pipes not playing any role other than to just be visuals makes the whole program useless.
- On a positive note, most other things that were planned to be added came out fine. The visuals, the grid, the layout, and the pipe functionality are all great things that were a worthy challenge that taught a good lesson.

- The cons of the project are mostly the calculations and pipes not having a meaning. The proper calculation results sort of depend on the pipes working as there currently is no way to know how long the pipeline is, how many bends there are, if it's continuous, if it splits into two. This is a major problem, that could be addressed with a proper data structure and an algorithm, using perhaps a linked list to construct an ordered and a sequenced list of pipes, that contain the information of the surrounding pipes.
- The program could be improved by fixing said problems and even expanding to add an option for pumps, valves, other fluid mechanical things. Adding more grid elements is easy, as they just need to be fed through the *pipeButtonCreator* method which creates the buttons and icons for them, though the method name wouldn't be very describing anymore.
- If I were to start the project all over again, I would focus more on creating better classes and a network between them. Right now, it feels a bit messy to navigate around different classes.

15. References – Viitteet

- https://mycourses.aalto.fi/pluginfile.php/1440943/mod_folder/content/0/Luento%2014%20-%20Putkivirtausten%20ratkaiseminen.pdf?forcedownload=1
- <https://www.nuclear-power.net/nuclear-engineering/fluid-dynamics/bernoullis-equation-bernoullis-principle/extended-bernoullis-equation/>
- <https://forum.qt.io/>
- <https://python-qt-tutorial.readthedocs.io/en/latest/>
- <https://stackoverflow.com/>
- <https://www.youtube.com/>
- <https://www.pythonguis.com/tutorials/pyqt6-creating-your-first-window/>
- <https://www.w3schools.com/>

16. Attachments – Liitteet

- Pictures below show the overview on how the program would typically be used:
 1. Open the program.
 2. Build a pipeline by connecting pipes. Pipes can be rotated.
 3. Enter the pressures for both the starting and ending points.
 4. Press "Calculate flow" to see the results.

