



h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbeit

FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

MASTER THESIS

Title: Power System Demonstrator in 3D

Submitted by: Sharareh Partovi Kisomi

1st Academic Supervisor: Prof. Dr. Athanasios Krontiris

2nd Academic Supervisor: Prof. Dr. Klaus-Martin Graf

Industrial Supervisor:

Completion Date: 14.03.2022

Student:

Sharareh
First (Given) Name

Partovi Kisomi
Last (Family) Name

Date of Birth: 18.09.1986

Matr.-No.: 762637

1st Academic Supervisor: Prof. Dr. Athanasios Krontiris

2nd Academic Supervisor: Prof. Dr. Klaus-Martin Graf

Title: Power System Demonstrator in 3D

Abstract: (max 10 Lines)

This thesis presents the development of Power System Demonstrator in 3D application, which provides an interactive 3D environment for simulating power flow of two interconnected networks with high voltage DC and high voltage AC. It uses Unity application, a gaming engine, for building a real-time 3D environment, and leverages the capability of Pandapower, an open-source, Python-based power flow analysis tool. The application is developed in object-oriented programming language Python and C#. The architecture provides a TCP network connection through the localhost for the real-time communication between Pandapower and Unity. The development process and the work flow of the app is explained.

In partial fulfilment of the requirements of the **University of Applied Sciences Hochschule Darmstadt (h_da)** for the degree **Master of Science in Electrical Engineering** carried out in collaboration with **Industrial Enterprise**

Company: Hitach Powergrids Germany

Address: Kallstadter Street 1, D-68309, Mannheim

.....
.....
This Master Thesis contains confidential data and may only be made available to the supervisor, the members of the examination board and authorized members of Darmstadt University of Applied Sciences.

(Signature)

1st Academic Supervisor:

Student:

Sharareh

First (Given) Name

Partovi Kisomi

Last (Family) Name

1st Academic Supervisor: Prof. Dr. Athanasios Krontiris

2nd Academic Supervisor: Prof. Dr. Klaus-Martin Graf

Declaration

I hereby declare that this thesis is a presentation of my original research work and that no other sources were used other than what is cited.

I furthermore declare that wherever contributions of others are involved, this contribution is indicated, clearly acknowledged and due reference is given to the author and source.

I also certify that all content without reference or citation contained in this thesis is original work.

I acknowledge that any misappropriation of the previous declarations can be considered a case of academic fraud.

Darmstadt, 13.03.2022

(Date)

Sharareh Partovi K.

(Signature)

Abstract:

By increasing in industrialization and urbanization all over the globe the demand for energy is also increasing steadily and as a consequence the demand for power generation is going higher. On the other hand, with the attitude of declining of conventional fossil fuel energy resources in favor of environmental protection and to comply with carbon emissions restrictions, countries are more and more encouraged to utilize renewable energy sources for supplying the required power. However, integrating renewable energy sources to power networks poses variability and uncertainty to the grid due to their fundamental nature. To overcome these challenges, High voltage direct current (HVDC) transmission have been given significant consideration because of their efficiency for transmitting bulk power over long distances and the security and reliability that they bring to the power networks. Currently, HVDC transmission projects are being built around the world, especially in Europe and Asia. This trend is increasing and HVDC is being more proposed as complement to AC power transmission to decision makers.

With that in mind Power system in 3D application is developed with the aim to simulate the results of power flow calculation in an interconnected network with the utilization of HVDC link. The app is designed as an informatics app with an interactive UI capability. The visualization is in a 3D environment with the insight of giving its audience a better understanding of how modifying power networks component can improve the system. By modifying the values of the network components, user can visualize the changes in the networks in real-time. The application is developed in Unity, a game engine for deploying 3D games, and Pandapower, an open-source, Python-based power system analysis tool for power flow calculation. The architecture of the app is based on networking through localhost to communicate between Unity and Pandapower. The programs are written in object-oriented Python and C# language. This thesis walks through the steps of development of the application.

Table of Contents

Table of Contents	i
Table of Figures	iii
List of abbreviations	iv
1.....INTRODUCTION	1
1.1. Motivation and Objectives	1
1.2. Related Work	2
1.3. Outline	3
2.....BACKGROUND	4
2.1. HVDC System	4
2.1.1. HVDC and Reliability	5
2.1.2. Conversion of AC lines to HVDC	5
2.1.3. Decreasing Cost	6
2.2. Power Flow Analysis	7
2.2.1. Basic formulation	7
2.2.2. Admittance matrix and power flow equation	8
2.2.3. Gauss-Seidel Method	9
2.2.4. Newton-Raphson Method	10
2.2.5. Fast-Decoupled Method	10
2.2.6. DC Power Flow Method	11
3.....APPLICATION VIEW	13
4.....SYSTEM DEVELOPMENT	17
4.1. Open-Source Tools for Power System	17
4.1.1. ANDES	17
4.1.2. PYPOWER	18
4.1.3. PANDAPOWER	19
4.2. Integrating Pandapower Solver and Unity	22
4.2.1. Embedding Python in Unity	22
4.2.2. Python server and Unity client communication	23
4.3. Python Codes	26
4.3.1. Starting Network Connection	27
4.3.2. Defining Pandapower Network in Python	28
4.3.3. Sending data to Unity	29
4.4. Unity Codes	30
4.4.1. Accepting Connection from Python	30

4.4.2.	Defining Pandapower Elements in Unity	31
4.4.3.	Packaging and Converting Data for transmission	31
4.4.4.	Defining pandapower elements in Unity Editor	32
4.4.5.	Interface between UI and Python	32
4.4.6.	UI Design	33
5.....	SUMMARY AND FUTURE WORK	34
	Appendix	v
	Appendix A. Coding Development	v
	Appendix B. Create a Pandapower Network in Unity Editor	xxvi
	Appendix C. Create UI in Unity Editor	xxxi
	Bibliography	xxxii

Table of Figures

Figure 1. The Scenario in GSA	2
Figure 2. Standard View of GSA.....	3
Figure 3. DC link embedded inside an AC system.....	4
Figure 4. HVDC point-to-point connection.....	4
Figure 5. HVDC Link for Wind Farm Integration to the System.....	5
Figure 6. Comparison of HVDC to HVAC lines.....	6
Figure 7. PSD Standard View.....	13
Figure 8. PSD standard View	14
Figure 9. Use case Diagram.....	14
Figure 10. Generator Data in UI.....	15
Figure 11. Submit Button	15
Figure 12. Bus Result and DC Line Components	16
Figure 13. Error Message and Restart Button	16
Figure 14. Electric power system analysis in pandapower.....	20
Figure 15. Architecture of PSD app	23
Figure 16. Connection in Unity side.....	25
Figure 17. Connection in Python side	26
Figure 18. Pandapower Element.....	29
Figure 19. Data Formatting in Python	30
Figure 20. bus data frame in pandapower.....	31
Figure 21. Data Formatting in C#.....	32
Figure 22. Adding a script to an empty GameObject	xxvii
Figure 23. bus data frame in pandapower.....	xxvii
Figure 24. bus and generator GameObject with Components	xxviii
Figure 25. line data in pandapower	xxix
Figure 26. AC line and it components.....	xxix
Figure 27. Line models and Particles	xxx
Figure 28. Adding Collider Componen	xxx

List of abbreviations

PSD	Power System Demonstrator
HVDC	High Voltage Direct Current
HVAC	High Voltage Alternating Current
GSA	Grid Stability App
VSC	Voltage Source Converter
GS	Gauss_Seidel
DAE	Differential-algebraic Equation
GIL	Global Interpreter Lock
CLR	Common Language Runtime
GIL	Global Interpreter Lock
TCP	Transmission Control Protocol

1. INTRODUCTION

1.1. Motivation and Objectives

The trends in industrialization and urbanization are increasing steadily all over the globe which leads to continuous increase in energy demand and consequently increase in generating more power. In this trend the role of utilizing High voltage DC (HVDC) system for transmitting bulk energy has been highlighted, as they are well established for long-distance, point to point power transfers comparing to the conventional high voltage AC (HVAC). However, carbon emissions restrictions and environmental protection restricts the generation of fossil fuel sources which encourage the decision-makers to incline more to renewable energies. The typical characteristic for renewable energy sources (RES) is variability and unpredictability due to their nature. HVDC also has the unique capability to address the challenges of integration of asynchronous networks and enhance reliability and stability to interconnected networks. Although there HVDC has been widely used in modern networks, especially in Europa and Asia, there are many potential areas that HVDC could be considered as an effective solution for balancing the secure demand and supply.

Having that in mind, Power System Demonstrator in 3D is an application which have been developed with the purpose of giving a better insight to its audience by providing a real-time, 3D environment for visualization of power flow analysis in an interconnected network. The main objective of the PSD application is to use an open-source power system analysis tool as the power flow solver in the backend of the application.

For creating the application Unity game engine is opted. Unity is a cross platform for 3D Games and provides an intuitive interaction with the audience. The language used for programming Unity is C#. Among the available open source power flow analysis tool, Pandapower is chosen. Pandapower comes with a steady-state power flow solver and is developed in Python language. There were several approaches for embedding Pandapower with Unity and make an interface that they interact with each other in real-time: In_Process_ API, which is integrating Pandapower inside Unity. There are challenges that hinder this approach because there are third libraries that need to be Pandapower environment, like numpy, numba. These libraries are tied to CPython and cannot feasibly be integrated in Unity. The other option is using Out_of_Process API, which is using a network connection that transmits data between Unity and Pandapower through a local port. After the investigation, the TCP (Transmission Control Protocol) connection approach has been opted. The architecture and the process of developing the application is elaborated in this document.

In the following an exciting application which is for 3D visualization of power flow analysis is introduced.

1.2. Related Work

Grid Stability App is developed by Hitachi ABB Power Grid. It is designed as an informative app for visualizing the power flow in a power grid. It aimed at highlighting the capabilities of the HVDC technology regarding the enhancement in reliability and stability in a power grid.

This application is made in Unity software and is built for Windows. The components of GSA as follow: Three different scenarios defined for the demonstrated network that animates the changes in the frequency of the grid and power flowing to the grid in the network with and without applying HVDC technology. Figure 1 shows an example of one of the scenarios. The audience can choose between these predefined scenarios are:

HVDC enhancing stability with large disturbances

HVDC enabling higher integration of renewables

HVDC balancing renewable generation

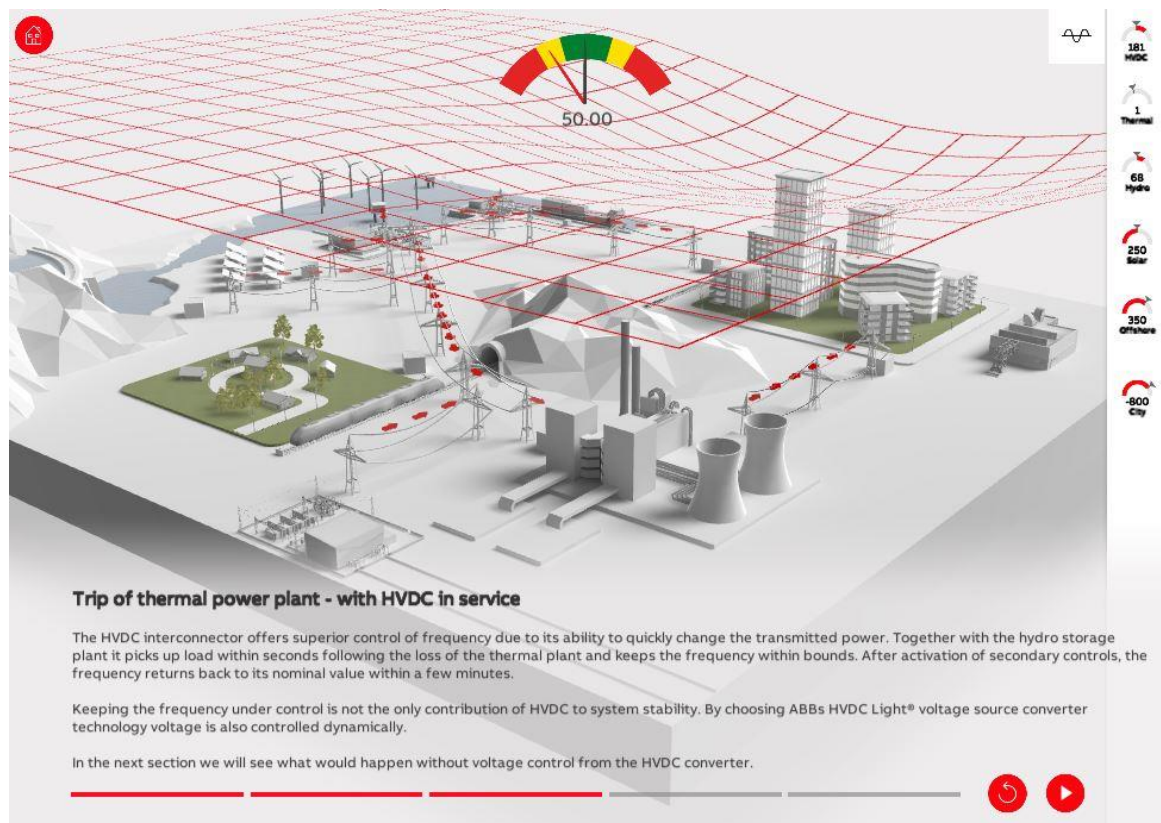


Figure 1. The Scenario in GSA

In addition, the audience can explore the network and make changes in its settings. The standard view panel of GSA is shown in Figure 2 .

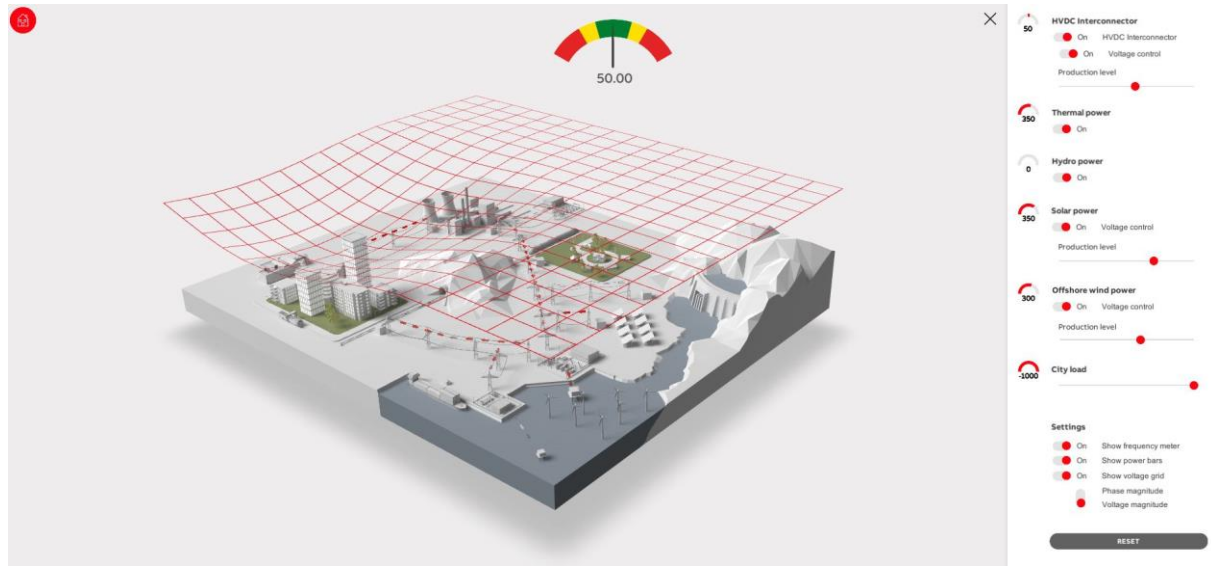


Figure 2. Standard View of GSA

The control panel of GSA provides the audience with the ability to change the settings. The demonstrated network in the app contains generation plants, offshore wind and solar power, and an HVDC interconnector which can control the voltage. There is one load node and the lines are AC lines. The generators can also be set to voltage controlled ones. The power production level of generators can be modified, and depending on the settings, the slack bus can change. There is a frequency meter for visualizing the frequency variation in real time when settings are changed. GSA provides a frequency calculation and power flow calculation for the power network. The solver for power flow calculation is written with Newton-Raphson method and with C# programming language.

1.3. Outline

The remaining parts of this thesis are structured into the following chapters:

Chapter 2 introduces HVDC system and its contribution in today's interconnected systems. Then, the basics of power flow analysis, followed by the different methods that can be applied for solving equations in power flow analysis.

Chapter 3 gives an overall view of the environment of developed app in the play mode.

Chapter 4 presents open-source, Python-based tools for power flow calculation that can fulfill the purposes of the app. This is followed by the investigation process of different approaches for integrating Python and Unity processes. Then the architecture and development of the codes are explained.

Chapter 5 summarizes the work and proposes the directions of future works.

2. BACKGROUND

Today, the demand for electricity is continuously rising, and more power is required to be transmitted through the power systems which are often located far from the load centers. On the other hand, the need for connecting asynchronous networks are increasing by more numbers of micro-grids. High voltage DC (HVDC) system is nowadays considered a fairly mature technology that offer effective solutions for abovementioned challenges. This chapter firstly gives a brief overview of HVDC system and highlights how HVDC technologies can address challenges in power systems. Next, the power flow analysis and its methods are explained.

2.1. HVDC System

HVDC system is a high-voltage electric power system which uses direct current for the transmission of electrical power. HVDC system mainly consists of two converter stations at either ends of the HVDC link. At sending station, the converter transformer takes the electric power from AC network, steps the AC voltage up to the required level, rectifies it to DC and transmit it through the line. At the receiving point the inverter terminal converts DC to AC power and inject it into the receiving AC network [1].

Figure 3 to Figure 5 depict different applications of HVDC. The classical HVDC is a DC link embedded in an AC network. One other application of DC link is pint-to-point connection that is applied between two areas with two different frequencies. HVDC links also play a significant role for integration of renewable energy sources especially offshore wind farms, into the power networks.

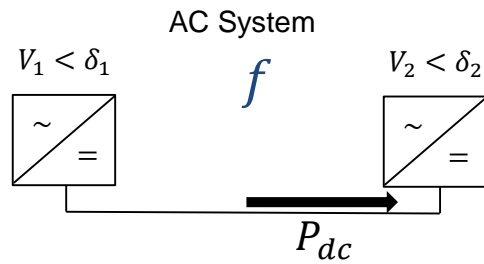


Figure 3. DC link embedded inside an AC system

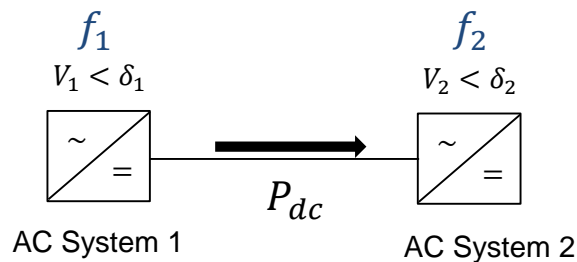


Figure 4. HVDC point-to-point connection

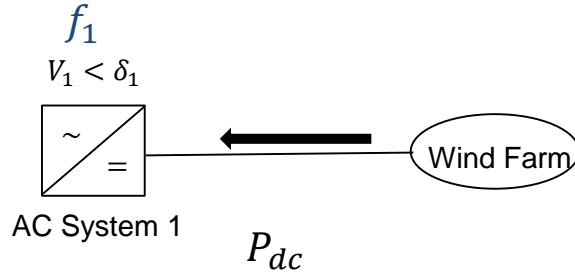


Figure 5. HVDC Link for Wind Farm Integration to the System

Below, the stability, reliability and efficiency of HVDC in the power systems are explained in more details.

2.1.1. HVDC and Reliability

As mentioned before, HVDC has been efficiently used to connect unsynchronized systems with different frequencies while transmitting high amount of power over the distance. The transmitted power can be independently controlled by voltage and frequency. This capability is achieved by voltage source converter (VSC) technology which is designed for controlling active power in the sending end converter [2]. One advantage of VCS converters station is that it helps the black start facility. During a normal power transmission operation, DC capacitors can be charged through these converters in one end. When an AC network connected to the other converter station is experiencing a blackout, the reserved battery will keep the control and protection system of this station alive [3]

VSC is best suited to interconnect remote generation facilities like offshore wind far away from shore for providing large scale of power, to the main power grid. Also due to its fast reactive power flow and voltage control it enhance voltage stability of the system.

2.1.2. Conversion of AC lines to HVDC

Although utilizing an HVDC link improves the power quality of the network, due to the shortage of suitable Right-of-Way it is challenging to provide a new DC line for an existing network. One possible way to is to upgrade an ac line to HVDC. In this way the transmission capability increases. “A double circuit 230 kV ac line limited to 400 MW transmission capacity can be used as an HVDC line with a capacity of about 1500 MW. [1]”. The conversion into dc has several advantages. One is that depending on the situation the existing tower and conductors will be re-used, depending on the situation. It also

reduces losses per transmitted MW. As an additional benefit when converting an ac line into dc, the short circuit current decreases in the points where the HVDC line is interconnected into the ac network as an HVDC substation does not contribute to the short circuit capacity [4].

2.1.3. Decreasing Cost

In an AC system, high voltage underground cables generate a large amount of reactive power due to their capacitive characteristic. The charging and discharging of the cables require a large reactive current which is also increase with the length of the cables. So to transport a certain amount of energy from one point to the other point additional current must flow in the cables to satisfy their reactive current. That is why the maximum practical length of cables in AC system is between 50 to 100 km. However, in DC system the capacitance of cables is charged only once, no additional current is required for that. Also, in AC systems the overhead lines longer than 200 km are highly that to push the power to the lines voltage needs to be increased to overcome voltage drop and phase shifts of the line. Therefore, overhead lines over long distances are more efficient via DC lines [5]

One criteria that affect the cost of HVDC system is the cost of converter substations. With the development of innovative technology and the decrease of power electronics devices HVDC costs has reduced. Although, converter substations are more complex than HVAC substations, and they require additional converting equipment and more complicated control system. Nevertheless, these costs can be covered by lower costs of DC transmission lines depending on the distance. Figure 6 shows the comparison of HVDC to HVAC lines (losses and typical configuration) [5].

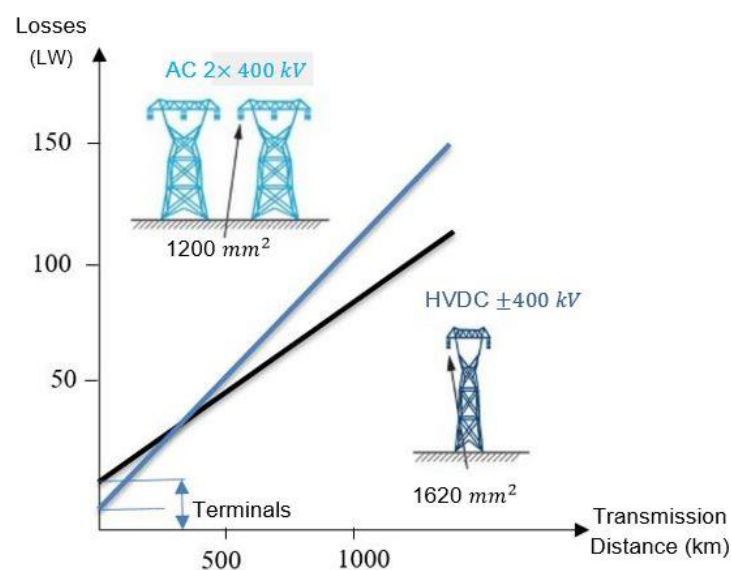


Figure 6. Comparison of HVDC to HVAC lines

Abovementioned is a brief of advantages of HVDC utilization in power systems. Integrating HVDC lines and considering them for transmission network expansion depends on different factors varying in different projects. Therefore, different power system analyses are carried out with the aim of reaching the steady state solution of complete power in the state of planning, control, and operation. In the following the basic of power flow analysis, and the methods for are discussed.

2.2. Power Flow Analysis

Power flow analysis is the numerical analysis in an inter-connected power system. The principal information obtained from a power flow analysis is the magnitude and phase angle of the voltage at each bus and the real and reactive power flowing in each line [6]. The power flow model of a power system is built using a relevant network, load, and generation data. Due to characteristics of power flowing into loads impedance through each transmission line, the equations which describe the model are non-linear. To solve this problem, iterative techniques are commonly used. However, for large AC systems these equations are not feasible and they are simplified as a linear problem in the DC power flow technique. This chapter will provide an overview of power flow fundamentals and different techniques used to solve the power flow problem.

2.2.1. Basic formulation

The purpose of power flow study is to find the complex voltages for all busses. Then based on that, the real active and reactive power flow through each line can be calculate. The starting point of solving power flow problem is to identify the known and unknown variables in the system. Each bus in a system has two known variables and two unknown variables, and based on these variables the busses are categorized into three types:

slack bus: Both voltage magnitude and angles are specified in a slack bus and that is why it is commonly considered as a reference bus. Each power system requires at least a slack bus to provide the mismatch between scheduled generation the total system load including losses and total generation.

generator bus: It is also called PV bus because the net real power is specified and voltage magnitude is regulated.

load bus: Load buses are called PQ buses because both real and reactive power loads are specified. Most of the buses in practical power systems are load buses.

2.2.2. Admittance matrix and power flow equation

The admittance matrix of a power system is the admittance values of both lines and buses which is derived from an abstract mathematical model of the system. “To create the admittance matrix, first a single line diagram of the power flow is converted to an impedance diagram. Then, all voltage sources convert to the current source and from there the admittance diagram is created. Then the admittance matrix can be constructed. The Y-bus is a square matrix with dimensions equal to the number of buses and is symmetrical along the diagonal. [6]”.

$$Y = \begin{bmatrix} Y_{11} & \cdots & Y_{1n} \\ \vdots & \ddots & \vdots \\ Y_{n1} & \cdots & Y_{nn} \end{bmatrix} \quad (2.2.1)$$

diagonal elements (Y_{ii}) are called the self-admittances at the nodes and are equal to the sum of the admittances connected to bus i . The off-diagonal elements (Y_{ij}) are equal to the negative of the admittance connecting the two buses i and j . However, in large systems, Y-bus is a sparse matrix. The net injected power at any bus can be calculated using the bus voltage (V_i), neighboring bus voltages (V_j), and admittances between the bus and its neighboring buses (Y_{ij})[6].

$$I_i = V_i y_{i0} + (V_i - V_1) y_{i1} + \cdots + (V_i - V_j) y_{ij} \quad (2.2.2)$$

Rearranging the elements as a function of voltages, the current equation becomes as follows:

$$I_i = V_i (y_{i0} + y_{i1} + \cdots + y_{ij}) - V_1 y_{i1} - V_2 y_{i2} - \cdots - V_j y_{ij} \quad (2.2.3)$$

The power equation at any bus can be written as follows:

$$S_i^* = P_i - jQ_i = V_i^* I_i \quad (2.2.4)$$

And real and reactive power can be calculated from the following equations:

$$P_i = \sum_{j=1}^n |V_i| |V_j| |Y_{ij}| \cos(\theta_{ij} - \delta_i + \delta_j) \quad (2.2.5)$$

$$Q_i = - \sum_{j=1}^n |V_i| |V_j| |Y_{ij}| \sin(\theta_{ij} - \delta_i + \delta_j) \quad (2.2.6)$$

And the current (I_i) can be written as a function of the power as follows:

$$\frac{P_i - jQ_i}{V_i^*} = V_i \sum_{j \neq i}^{j=0} y_{ij} - \sum_{j \neq i}^{j=1} y_{ij} V_j = V_i Y_{ii} + \sum_{j \neq i}^{j=1} Y_{ij} V_j \quad (2.2.7)$$

2.2.3. Gauss-Seidel Method

The Gauss-Seidel (GS) method, also known as the method of successive displacement, is the simplest iterative technique for solving power flow problems. In general, GS method has the following iterative steps to reach the solution for the function $f(x) = 0$. [6].

- Rearrange the function into the form $x = g(x)$ to calculate the unknown variable.
- Calculate the value $g(x^{[0]})$ based on initial estimates $x^{[0]}$.
- Calculate the improved value $x^{[1]} = g(x^{[0]})$

Continue solving for improved values until the solution is within acceptable limit

$$|x^{[k+1]} - x^{[k]}| \leq \epsilon \quad (2.2.8)$$

The rate of convergence can be improved using acceleration factors by modifying the step size α .

$$x^{[k+1]} = x^{[k]} + \alpha [g(x^{[k+1]}) - x^{[k]}] \quad (2.2.9)$$

In the context of a power flow problem, the unknown variables are voltages at all buses, but the slack. Both voltage magnitudes and angles are unknown for load buses, whereas voltage angles are unknown for regulated/generation buses [7].

One problem with the Gauss-Seidel iteration is that convergence can be very slow, and sometimes even the iteration does not converge despite that a solution exists. Furthermore, no general results are known concerning the convergence characteristics and criteria[6].

2.2.4. Newton-Raphson Method

This method begins with initial guesses of all unknown variables (voltage magnitude and angles at Load Buses and voltage angles at Generator Buses). “Next, a Taylor Series (the Taylor series of a function is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point.) is written, with the higher order terms ignored, for each of the power balance equations included in the system of equations . The result is a linear system of equations that can be expressed as: [7]”

$$\begin{bmatrix} \Delta\theta \\ \Delta|V| \end{bmatrix} = -J^{-1} \begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} \quad (2.2.11)$$

Where ΔP and ΔQ are called the mismatch equations.

And J is a matrix of partial derivatives known as a *Jacobian*: $J = \begin{bmatrix} \frac{\partial \Delta P}{\partial \theta} & \frac{\partial \Delta P}{\partial |V|} \\ \frac{\partial \Delta Q}{\partial \theta} & \frac{\partial \Delta Q}{\partial |V|} \end{bmatrix}$

To determine the next guess ($m + 1$) of voltage magnitude and angles the linearized system of equations is solved based on:

$$\theta^{m+1} = \theta^m + \Delta\theta \quad (2.2.12)$$

$$|V|^{m+1} = |V|^m + \Delta|V| \quad (2.2.6)$$

The process continues until a stopping condition is met. A common stopping condition is to terminate if the norm of the mismatch equations is below a specified tolerance.

2.2.5. Fast-Decoupled Method

It is a variation on Newton-Raphson that exploits the approximate decoupling of active and reactive flows in well-behaved power networks, and additionally fixes the value of the Jacobian during the iteration in order to avoid costly matrix decompositions. Also referred to as "fixed-slope, decoupled NR". Within the algorithm, the Jacobian matrix gets inverted only once, and there are three assumptions. Firstly, the conductance between the buses is zero. Secondly, the magnitude of the bus voltage is one per unit. Thirdly, the sine of phases between buses is zero. Fast decoupled load flow can return the answer within seconds whereas the Newton Raphson method takes much longer. This is useful for real-time management of power grids [7].

2.2.6. DC Power Flow Method

DC power is an extension to the Fast-decoupled power flow formulation [7]. “Dc power flow looks only at active power flows and neglects reactive power flows. This method is non-iterative and absolutely convergent but less accurate than AC power flow solutions. It is used wherever repetitive and fast load flow estimations are required. In DC power flow, nonlinear model of the AC system is simplified to a linear form through these assumptions:

- Line resistances (active power losses) are negligible i.e. $R \ll X$.
- Voltage angle differences are assumed to be small i.e. $\sin(h) \approx h$ and $\cos(h) \approx 1$.
- Magnitudes of bus voltages are set to 1.0 per unit (flat voltage profile).
- Tap settings are ignored. [8]”

Based on the above assumptions, voltage angles and active power injections are the variables of DC power flow. Active power injections are known in advance. Therefore, for each bus i in the system [7]:

$$P_i = \sum_{j=1}^N B_{ij} (\theta_i - \theta_j) \quad (2.2.7)$$

in which B_{ij} is the reciprocal of the reactance between bus i and bus j . B_{ij} is the imaginary part of Y_{ij} . As a result, active power flow through transmission line i , between buses s and r , can be calculated[7]:

$$P_i = \frac{1}{X_{Li}} (\theta_s - \theta_r) \quad (2.2.8)$$

where X_{Li} is the reactance of line i . DC power flow equations in the matrix form and the corresponding matrix relation for flows through branches are represented in (A.9) and (A.10)[7].

$$\theta = \frac{1}{B} P \quad (2.2.9)$$

$$P_L = (b \times A) \quad (2.2.10)$$

Where

P_L $N \times 1$ vector of bus active power injections for buses 1, ..., N

B $N \times N$ admittance matrix with $R = 0$

θ $N \times 1$ vector of bus voltage angles for buses 1, ..., N

P_L $M \times 1$ vector of branch flows (M is the number of branches)

b $M \times M$ matrix (b is equal to the susceptance of line k and non-diagonal elements are zero)

A $M \times N$ bus-branch incidence matrix

Each diagonal element of B (i.e. B_{ii}) is the sum of the reciprocal of the lines reactances connected to bus i . The off-diagonal element (i.e. B_{ij}) is the negative sum of the reciprocal of the lines reactances between bus i and bus j . A is a connection matrix in which a_{ij} is 1, if a line exists from bus i to bus j ; otherwise zero. Moreover, for the starting and the ending buses, the elements are 1 and -1, respectively[7].

This chapter introduced HVDC systems and the advantages they bring to modern power systems. Also, different numerical methods were reviewed which are used in power flow studies to investigate the bus voltages and amount of power flow through transmission lines. With this background, the rest of this thesis dedicates to steps of the development of PSD app.

3. APPLICATION VIEW

This chapter gives the reader an insight into the standard view of PSD app, the GUI, and the information that can be visualized in the play mode.

The power network used in PSD app contains two distinct power grids interconnected via HVDC and HVAC point to point lines.

The standard View of the app is shown in Figure 7 and . The AC lines are in white, and the DC line is shown in orange color. Each element can be selected by clicking on the element in the scene. Then the information of that element pops up on the screen.

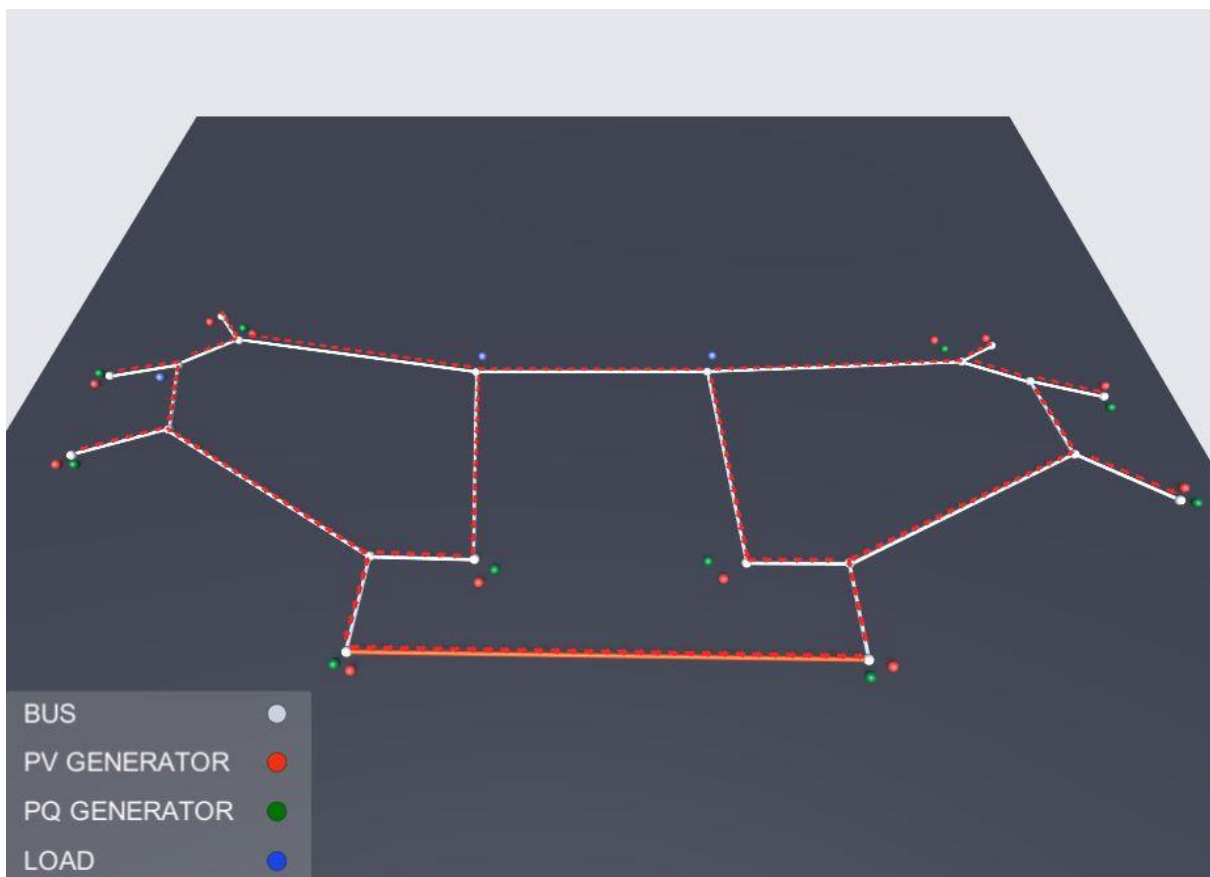


Figure 7. PSD Standard View

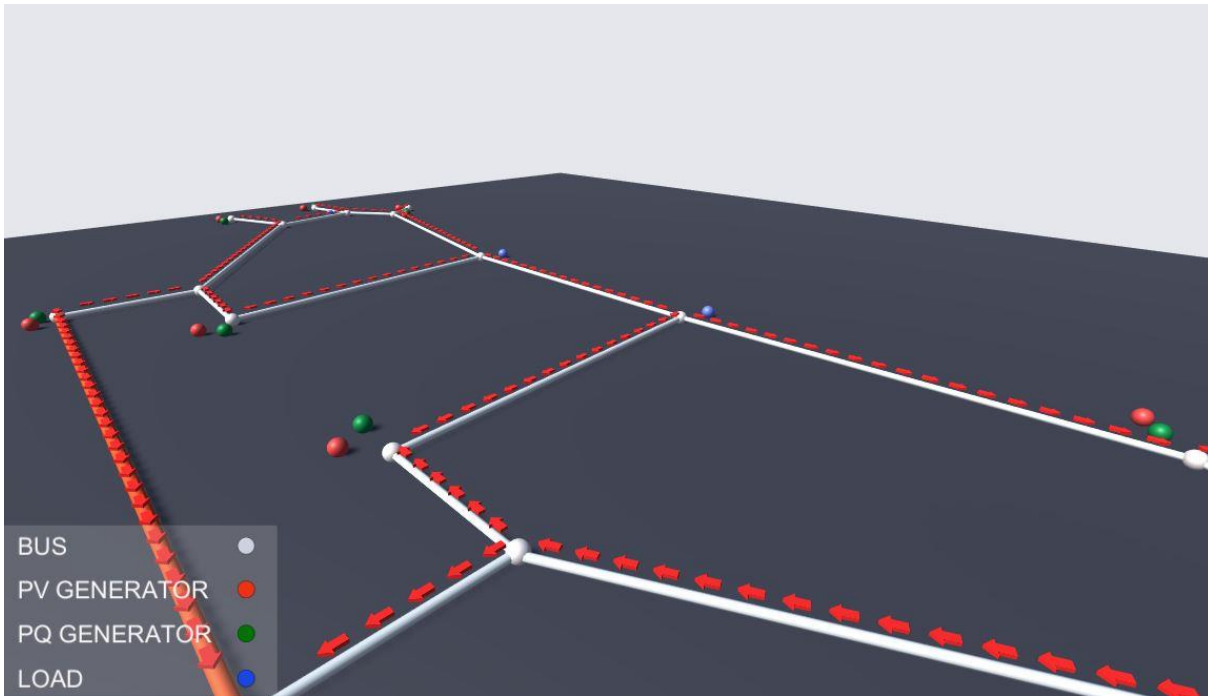


Figure 8. PSD standard View

Figure 9 an insight into user's possible interaction with the application.

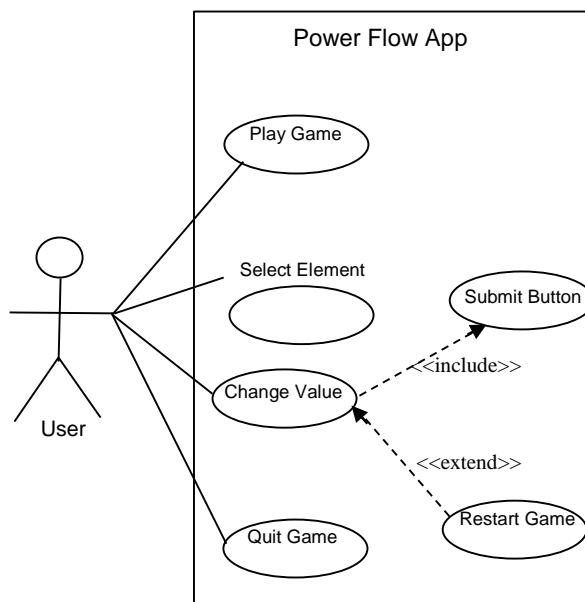


Figure 9. Use case Diagram

Figure 10 to Figure 12 illustrate an example of a Generator bus. When user changes a value in the input fields, a submit button pops up and by clicking on that new power flow will be visualize on the scene.

The network information is summarized in appendix A, which gives the reader an insight into the essential parameters need to simulate the network. The 3D model of the network is created in Unity application.

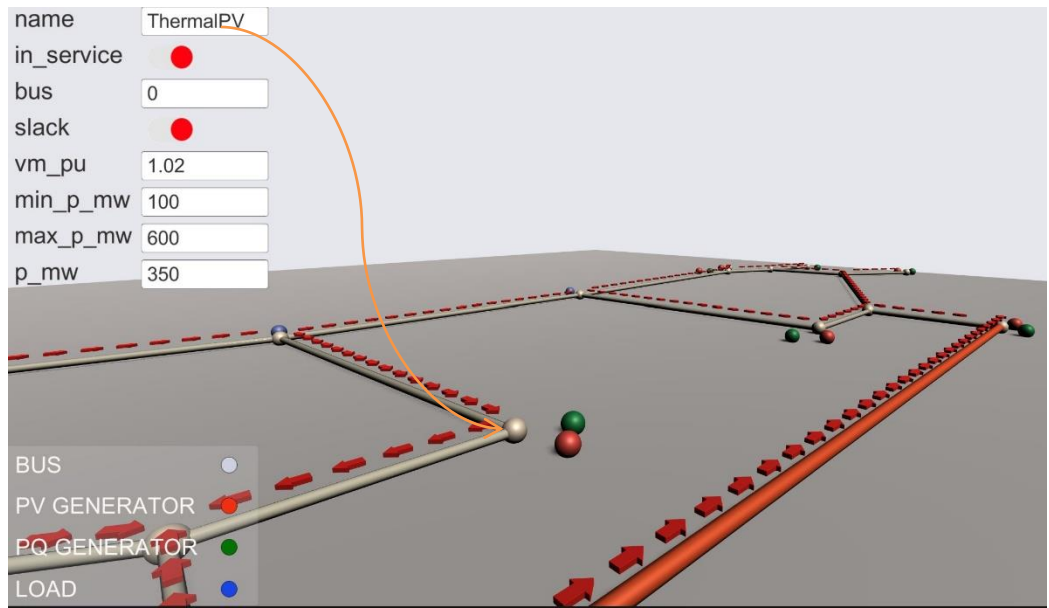


Figure 10. Generator Data in UI

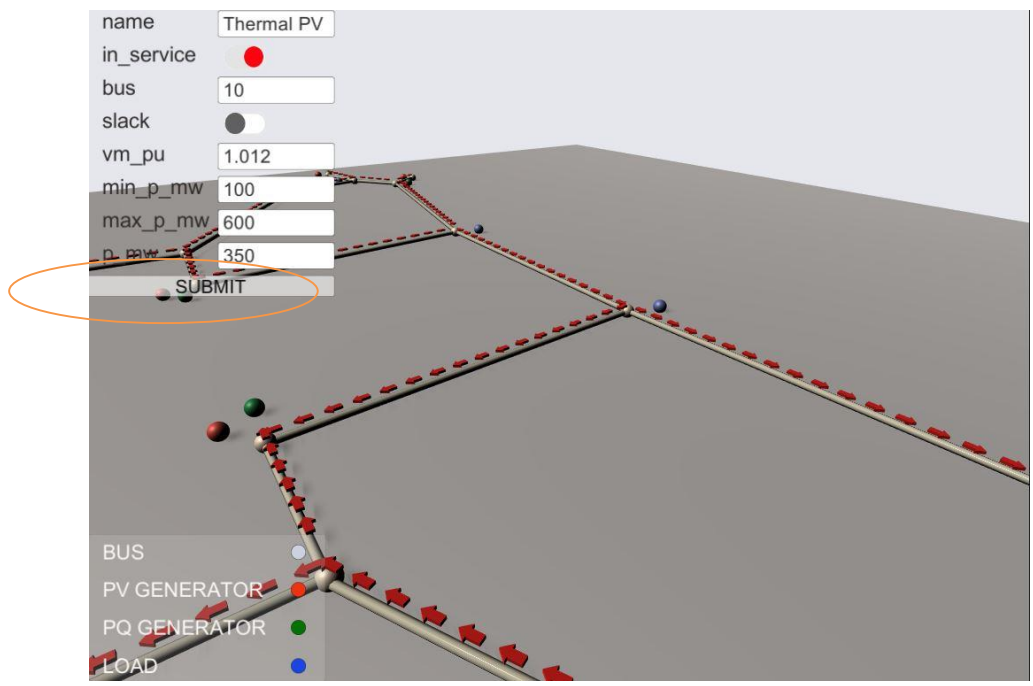


Figure 11. Submit Button

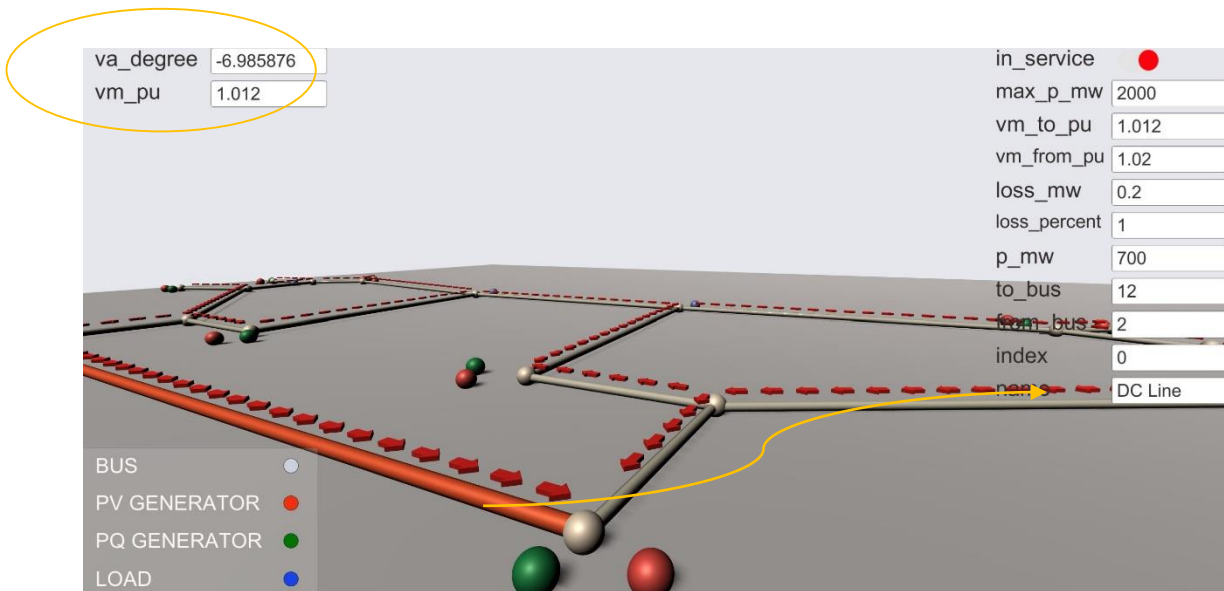


Figure 12. Bus Result and DC Line Components

When the input data are not in range and power flow cannot converge an Error pop ups, as shown in Figure 13. and by clicking on the Restart Button would reset the values and the game starts restarts again.

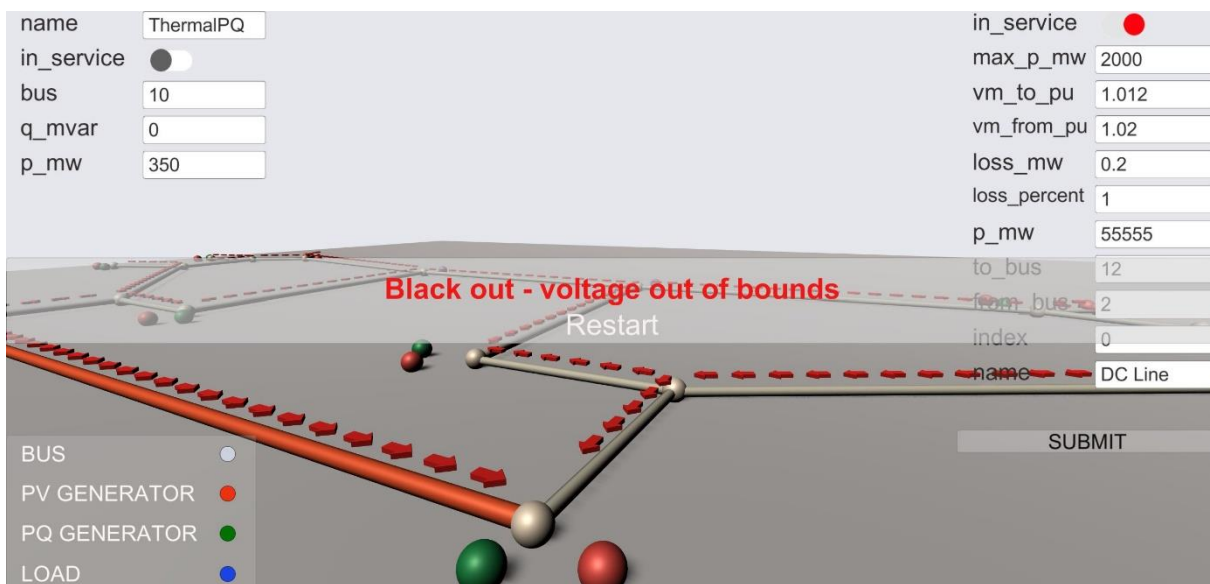


Figure 13. Error Message and Restart Button

4. SYSTEM DEVELOPMENT

Just similar to any other applications, the developed app in this thesis is also comprised of a back- and front-end. In what follows, the explanations of the back-end development delineated without going into too much details regarding the code beneath each script.

Python-based open-source power system analysis tools, and how the PSD app can benefit from Pandapower solver capabilities. Next different approaches for integrating the chosen solver to the app is discussed. The rest of chapter is dedicated to elaborate the development of Python and Unity codes.

4.1. Open-Source Tools for Power System

Modern power systems are inter-connection of different systems and they are complex systems which require computer models and simulation in order to analyze them and investigate their behaviors. For the power system studies, there are various types of open source tools available today. Many of these packages have been developed in Python language because it is easy to learn and its implementation is also free. In this section three free and open source packages are introduced, and their basic design choices with regard to grid modeling and data structure, as well as the power flow solver used for these programs are explored. At the end of this session an overview of advantages and drawbacks of choosing Pandapower as the solver for Power System Demonstrate app is discussed.

4.1.1. ANDES

ANDES is a is fully open-source and written in Python for power system simulation, control and analysis. It has features for applying deep learning power systems. ANDES comes with power flow calculation, time-domain simulation, and full eigenvalue calculation. It contains a hybrid symbolic-numeric framework for modeling differential algebraic equations (DAEs) for numerical analysis¹. used transfer functions and discrete components. The built-in processing and code generation functions will convert the descriptive models into executable code. Which is useful when one needs to test and control a new model that has not existed. “ANDES has adopted numba for compilation. It compiles the generated code for model equations and Jacobian functions are compiled into machine” [9].

Data structure: Parameters are typically numerical and externally supplied data for defining specific devices. They are from input files and in general constant once initialized [10].

Power Flow: In ANDES the methods used for power flow calculation is Newton-Raphson. In Time-domain simulation the default method for running the calculation is the Implicit Trapezoidal Method (ITM). Trapezoidal Rule is a rule that evaluates the area under the curves by dividing the total area into smaller trapezoids rather than using rectangles. It first solves the power flow with Newton Raphson method as a starting point. Next, the numerical integration simulates for n seconds. The information of

¹ <https://docs.andes.app/en/stable/tutorial.html>

running a power flow in ANDES contains four sections: a) system statistics, b) ac bus and dc node data, c) ac line data, and d) the initialized values -of other algebraic variables and state variables. Compared to other tools such as PSAT, Dome and PST, ANDES provides an easier way for developing differential-algebraic equation (DAE) based models for power system dynamic simulation². ANDES has a descriptive modeling framework that allows quick prototype of new models and controllers and uses equations and modeling blocks. Such blocks include the commonly.

4.1.2. PYPOWER

PYPOWER is a part of MATPOWER to the python programming language for solving steady-state power system simulation and optimization problems with an extension program for time-domain simulations. In the following the basic design choices in PYPOWER are explored.

Data structure: The datastructure in PYPOWER is based on matrices, defining the electric attributes of the network in a casefile in the form of a bus/branch model (BBM), which defines the network as a collection of buses which are connected by generic branches. Branches are modeled with a predefined equivalent circuit and are used to model multipole elements like lines or transformers. Buses are attributed with power injections or shunt admittances to model single pole elements like loads, generators or capacitor banks. Since the BBM is an accurate mathematical representation of the network, electric equations for power systems analysis can be directly derived from it.

Power Flow: The power flow solver in PYPOWER is based on Newton's method & Fast Decoupled method for solving AC and DC power flow respectively³.

PYPOWER_Dynamic: For transient stability simulation, an open source program called PYPOWER-Dynamic is built on PYPOWER⁴. The aim of a transient stability program is to solve a system of differential-algebraic equations (DAE) involving the electrical network (which is assumed to be algebraic for the time-scales involved in stability runs) and dynamic elements interfaced to the network (e.g. machines, controllers, non-linear loads, etc).

The approach adopted by PYPOWER-Dynamics is to solve the DAE system using a partitioned solution approach and a current injection model to form the algebraic network equations⁵:

Partitioned solution approach: in this approach, the differential equations and algebraic equations are solved separately (i.e. partitioned) and are interfaced within a single time step. The order in which the

² <https://github.com/cuihantao/andes/blob/master/README.md>

³ <https://pypi.org/project/PYPOWER/>

⁴ <https://github.com/susantoj/PYPOWER-Dynamics/wiki>

⁵ <https://github.com/susantoj/PYPOWER-Dynamics/wiki/2.1.-Program-Structure-and-Flow>

equations are solved is somewhat arbitrary and in PYPOWER-Dynamics, the differential equations are solved first.

Current injection model: in this model, the current injections at each network bus $[I]$ are known (i.e. calculated) and the bus voltages $[V]$ are solved algebraically at each time step using the bus admittance $[Y_{bus}]$ matrix, i.e. $[V] = [Y_{bus}]^{-1}$

There are two types of dynamic models supported in PYPOWER-Dynamics ⁶:

built-in models: are hard-coded dynamic models of common network elements such as synchronous machines, external grids, induction machines, converters, etc.

user-defined controller models: are the dynamic models of machine controllers such as AVRs, governors, etc that are created by the user in the form of controller definition files.

4.1.3. PANDAPOWERR

“Pandapower is a Python based open source tool for static and quasi-static analysis and optimization of balanced power systems.” [11]. It provides a static analysis of three phase AC, DC and optimal power flow problems. Pandapower combines the data analysis library pandas and the power flow solver PYPOWER for power system analysis and optimization in distribution and sub-transmission networks. However, in Pandapower there are significant improvements in some respects in order to offer a faster and a more user friendly tool for power flow studies.

Network Models: Pandapower uses an element-based model (EBM) to model electric grids. An element is either connected to one or multiple buses and is defined with characteristic parameters. Instead of a generic branch model, there are separate models for lines, two-winding, three-winding transformers etc.

Data Structure: Pandapower is based on a tabular data structure, where every element type is represented by a table that holds all parameters for a specific element and a result table which contains the element specific results of the different analysis methods. The tabular data structure is based on the Python library pandas [12]. It allows storing variables of any data type, so that electrical parameters can be stored together with status variables and meta-data, such as names or descriptions. The tables can be easily expanded and customized by adding new columns without influencing the Pandapower functionality.

Power Flow: The Pandapower power flow solver is based on the Newton-Raphson method. The implementation was originally based on PYPOWER, but has been improved in some aspects. For carrying out network analysis all element models in Pandapower need to be converted into their equivalent representation. Figure 14. Electric power system analysis in pandapower Shows the translation of the data structure to BBM structure for running the analysis [6].

⁶ <https://github.com/susantoj/PYPOWER-Dynamics/wiki/2.1.-Program-Structure-and-Flow>

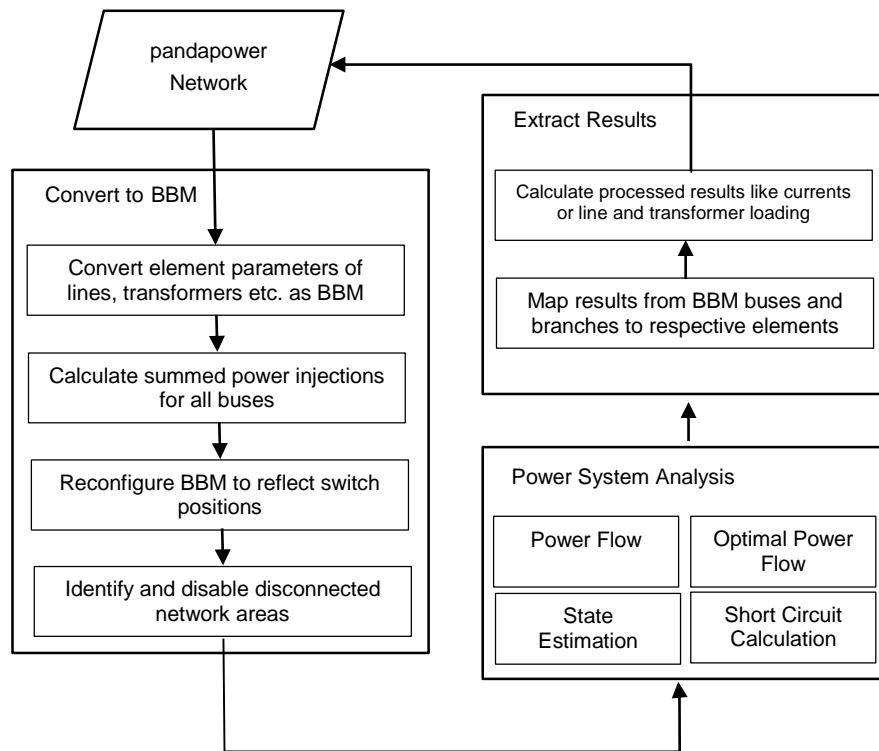


Figure 14. Electric power system analysis in pandapower

Internal power flow parameters, such as node type for the power flow calculation (slack, PV or PQ node) or per unit conversions, are carried out automatically by Pandapower. This improves user convenience and reduces the risk of incoherent input data. Pandapower offers three different methods to initialize the complex voltage vector for the AC power flow calculation. It can either be the result of a previous power flow calculation, the solution of a DC power flow or a flat start.

Pandapower uses PYPOWER power flow solver. Although generating nodal matrix is easy in PYPOWER, there are some problems for the user when defining a network in PYPOWER:

- the casefile in PYPOWER only contains pure electrical data. Meta information, such as element names, line lengths or standard types, cannot be saved within the datastructure.
- since there is no API for creating the casefile, networks have to be defined by directly building the matrices.
- the user has to ensure that all bus types (PQ, PV, Slack) are correctly assigned and bus and gen table are coherent.
- the datastructure is based on matrices means deleting one row from the datastructure changes all indices of the following elements.
- power and shunt values can only be assigned as a summed value per bus, the information about individual elements is lost in case of multiple elements at one bus.

Comparison between Properties of Pandapower and PYPOWER:

In Pandapower the likelihood of making errors when creating a network is reduced. also It uses an intuitive and commonly used model plate parameters (such as line length and resistance per kilometer) instead of parameters like total branch resistance in per unit). Also, variables can be accessed by name instead of by column number of a matrix. Which makes it easier to work with, in the scope of this project. There are also other advantages when considering Pandapower as the power flow analysis tool:

since all information is stored in pandas tables, all inherent pandas methods can be used to access any information that is stored in the Pandapower dataframes like query, statically evaluate, iterate over and more. Element parameters, power flow results or a combination of both the power flow results are processed to include not only the classic power flow results (such as bus voltages and apparent power branch flows), but also line loading or transformer losses). In addition, power flow in Pandapower is accelerated with a numba implementation that allows very fast construction of nodal point admittance and Jacobian matrices compare to PYPOWER. Moreover, power flow has different possibilities for initialization of power flow, from DC power flow or from previous results.

On the other hands, Pandapower does not support transient simulation and dynamic short-circuit simulation which is possible to be calculate with PYPOWER or ANDES.

PSD app is started developing with utilizing Pandapower as its solver with a steady-state power flow calculation. As mentioned above Pandapower the tabular data structure facilitates accessing and exchanging the data. It also very well documented. Next, the processes of finding a solution for integration of Pandapower in Unity is explained.

4.2. Integrating Pandapower Solver and Unity

In Unity standard scripting language for development of game is C#. However, the solver that is chosen for the application is Pandapower solver which is written in Python. The starting point of the project is to define a solution for communication between Unity and Pandapower which suits best for the purpose of Power System Demonstrator app. There are two approaches introduced by Unity for communication with Python: In-Process API ⁷ and Out-Of-Process API ⁸.

The In-Process API is to host Python directly from Unity using a library which allows interoperability between the host (C#) and the hosted script (Python). The out-of-process API allows Python scripts to have a tight integration with Unity but still run in a separate process. It's used typically to integrate third-party packages. The out-of-process API opens a localhost socket on a fixed port number. And the communication is through the localhost socket. These two approaches have been examined for development of Power System Demonstrator app and the process are discussed below.

4.2.1. Embedding Python in Unity

The In-Process API is integrating Python in Unity which can be the most straightforward way for Unity and Pandapower communication. But it also can have some limitations. Not all third party python libraries are usable in integrated Python to Unity. To work with Pandapower a distribution with numpy, scipy and numba libraries are required for the power flow calculation. These libraries are tied to CPython and it is not possible to integrate C-libraries with Python integrated in Unity. However, there is a package that can hook up Python and Unity, **Pythonnet** ⁹. Pythonnet is a package that provides an application scripting tool for .NET and is in CPython so it is compatible with other Python libraries, it can use pip or Conda to download these libraries. (.NET is a framework that can be used to develop an application and Unity uses .NET platform to make the applications)

At this level, for the development of the app, programming started with Pythonnet in Unity. These steps have been done: setting the PythonDLL property, calling to python inside GIL to manage the GIL (Global interpreter lock) to allow only one thread to hold the control of the Python interpreter, importing Python modules using dynamic mod, and declaring python objects as dynamic type. Nevertheless, there were some difficulties in the coding part. Pythonnet does not implement Python as a first-class CLR language, and there are different data type systems in .NET than in Python. .NET system does not have an exact language mapping to Python syntax. Some nice thing in C#, such as iterable data structures, are trivially mapped as iterables in Python, but others are not directly mapped. Some data type can be converted automatically, but some do not.

⁷ <https://docs.unity3d.com/Packages/com.unity.scripting.python@2.0/manual/inProcessAPI.html>

⁸ <https://docs.unity3d.com/Packages/com.unity.scripting.python@2.0/manual/outOfProcessAPI.html>

⁹ <https://www.nuget.org/packages/pythonnet/>

After examining Pythonnet, and facing challenges with the process of coding and converting Python functions to C#, the second approach for this application was taken into account. Socket communication has several advantages that outweigh the complexity of integrating Python to Unity approach.

4.2.2. Python server and Unity client communication

The out-of-process API which is via localhost socket allows Python scripts to have a tight integration with Unity but still run in a separate process. It's used when integrating third-party packages are needed¹⁰. For that we use a TCP communication, TCP communication is a way for data communications over IP between two processes in a system. TCP sockets provide an open bi-directional connection between two endpoints. Each connection is uniquely identified using the combination of the client socket, and server socket. In a connection through a local system the IP address and send port and received port is the same between server and client.

For PSD app a Python server and a Unity Client is defined. Figure 15 Shows the general architecture and components for The rough idea is to send data from Unity to Python and visa-versa, in real time.

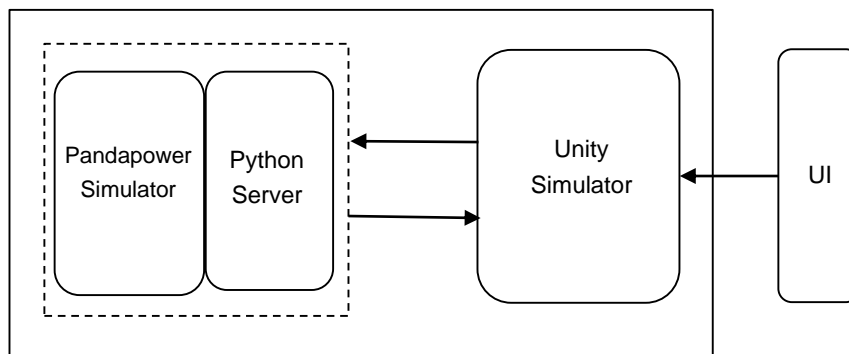


Figure 15. Architecture of PSD app

The general steps of the architecture can be explained as:

1. At the start of the game the main network, and the elements that are predefined in the network, is created in the environment of Unity. Each element data is packed in a message to be sent.
2. The messages, containing element data, are sent one by one to a running server in python via TCP.
3. In Python, based on the data received from Unity, a Pandapower network is created, and the power flow calculation is run, which produces the results tables for buses and lines.
4. The results, one after the other, are sent back to Unity as a message via TCP.

¹⁰ <https://docs.unity3d.com/Packages/com.unity.scripting.python@2.0/manual/outOfProcessAPI.html#limitations>

5. The visualization program performs corresponding to the results.
6. The Python waits for new data being sent from Unity (unless the client or server stops)

These steps are illustrated for both Unity and Python in Figure 16 and Figure 17 respectively. The data is being send through the ports as a message. Therefore, a message is the basic unit responsible for communication, which needs to be broken down into its similar structures of data when it is received to its destination.

Network programing brings benefits for the app. After writing the program for connection, the rest of the codes are in pure Python and C# languages. So it is easy to directly work with Pandapower. That is also beneficial for future development, that other functionalities of Pandapower can be easily utilized in the app. the rest of programing is done in pure Python language and pure C#.

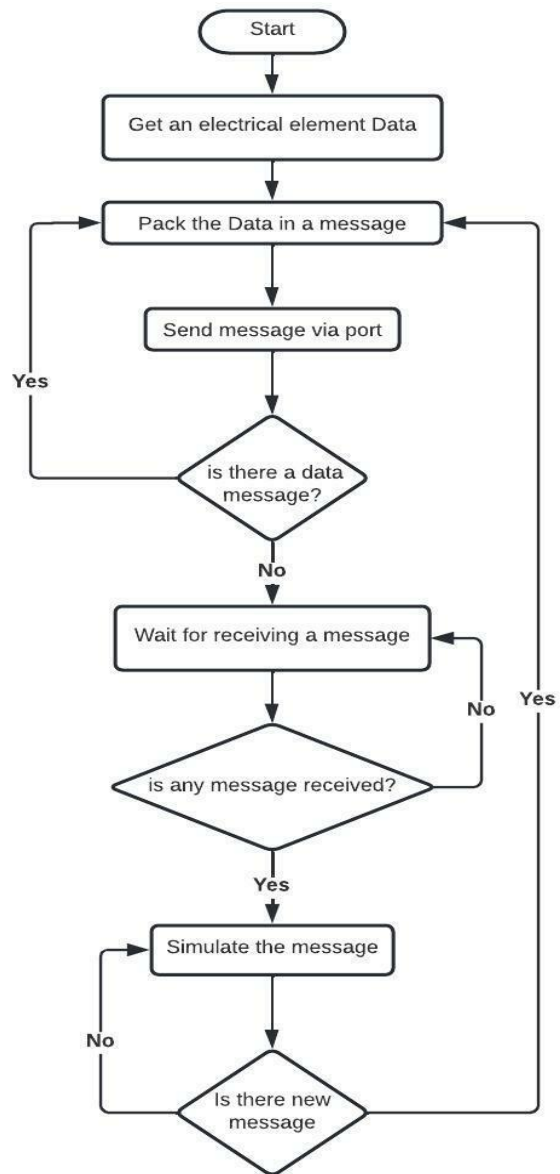


Figure 16. Connection in Unity side

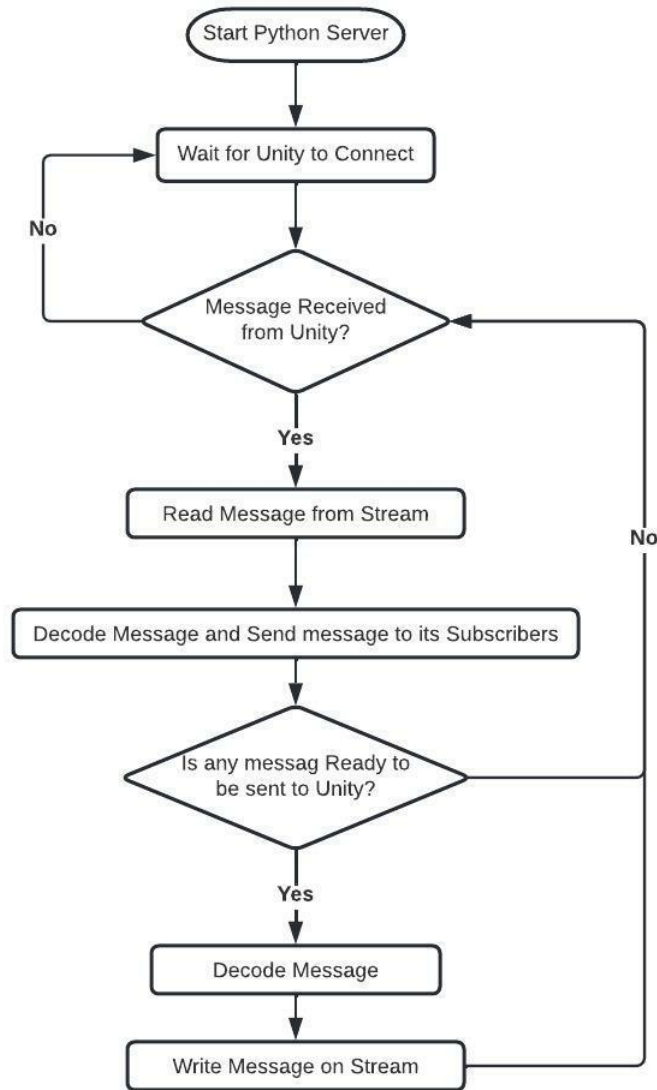


Figure 17. Connection in Python side

4.3. Python Codes

Coding in Python contains three main scripts. Starting from writing a script which enables networking. Then a new class for handling the creation of Pandapower network. Another class is also developed to handle the receiving and sending data.

For the socket programming, the asyncio module in Python is used. Asynchronous programming, is a feature that allows the program to juggle multiple functions without waiting or getting hung up on any one of them. It's a way to efficiently handle tasks in networking, where most of the program's time is spent waiting for a task to finish. In programming a thread is the smallest sequence of programmed instructions that can be managed independently. In this way multiple threads are used to process the connections. So one thread is responsible for initiating the sending or receiving of data, other threads complete the connection to the network device and send or receive the data ¹¹.

Also an event-driven programming is applied in Also an event-driven programming is applied which means the flow of the program is determined by events. An event in programming is an action or occurrence which can be recognized by the program, asynchronously from the external environment, and needs to be handled by the program ¹².

4.3.1. Starting Network Connection

Network script is responsible for starting the connection mechanisms between Python and Unity processes. Asyncio library is used to run python coroutine to have a control over the execution. The flow of the program is determined by events.

There are two classes that are responsible for networking. UML diagram of the program is in Appendix A. 1 Appendix A. 1 Network Class UML Diagram.

Server Class runs the server whenever a signal is received in the main Python program. It handles the Unity which wants to connect to the server and accept the connection. In this class the port name and IP address of the server are defined. (Appendix A. 2)

When the connection is accepted a peer is created which has the ability to read from, or write to the stream. The received client (Unity) would have a peer name. (Appendix A. 3)

Peer Class manages writing text to a stream. The method (Appendix A. 4 **Error! Reference source not found.**) writes characters to a stream in a specified encoding ¹³.

A read method is written that is responsible for encoding the message from byte to a string format. Whenever the encoding is done, an even is raised and send the string out of the class to its respective event handler.

In read method whenever a message is received the received bytes are first encoded to a string format and then an event is fired which send this message with string format to Client.py. Appendix A. 5

¹¹ <https://www.olivierpons.com/2020/11/13/asynchronous-python-tcp-server-and-its-c-unity-client/>

¹² <https://pypi.org/project/Events/>

¹³ <https://docs.python.org/3/library/asyncio-task.html>

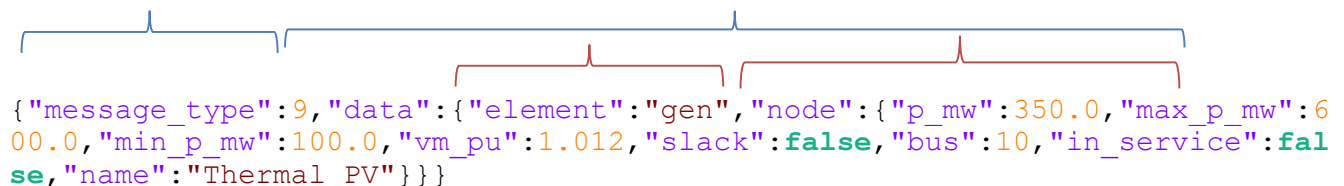
4.3.2. Defining Pandapower Network in Python

Power_network script is responsible for creating a Pandapower network. The methods that are used from Pandapower is shown in Appendix A. 6. (For developing the app and applying more functions from Pandapower, more methods can be added to this script).

Basics Pandapower functions (Appendix A. 7), for creating network, creating buses, creating lines, and so on. two important methods are also written in this class to get the index of the changed electric element in Unity environment, and then replacing the new data on the network table.

In Play mode of the app in Unity, when the user selects an electric element and changes the values of its parameters, a new message is packed in a JSON format. JSON format is purely a string and contains only properties, no methods. It has two primary parts, keys and values. Together they make a key/value pair. A key is always a string enclosed in quotation marks. And the value can be a string, number, array, or any object.

Here is an example: the user selects a generator and turns it off. The index of generator is 3. There are commands that are defined in Unity and Python with the points to the same function. Here the command is type 1, which informs that an electric element in the Pandapower network is changed and needs to be replaced. The whole message will be packed as below, which is a nested JSON:



```
{ "message_type": 9, "data": { "element": "gen", "node": { "p_mw": 350.0, "max_p_mw": 600.0, "min_p_mw": 100.0, "vm_pu": 1.012, "slack": false, "bus": 10, "in_service": false }, "name": "Thermal PV" } } }
```

In Pandapower each element must have a unique name in a network, and all attributes for each element is listed with their properties (defaults, etc.) and are accessible from the network object. The data structure of Pandapower element is shown in Figure 18 [11].

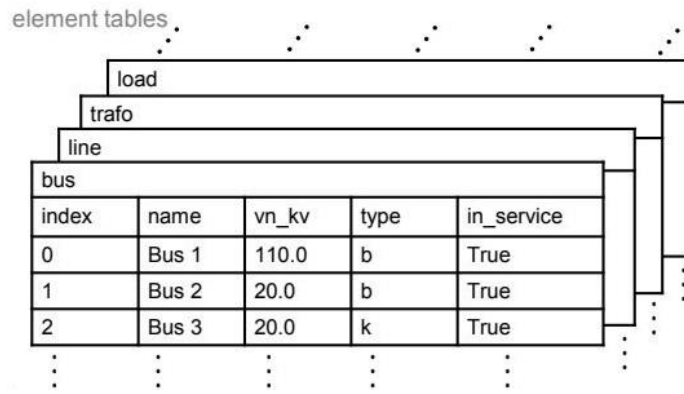


Figure 18. Pandapower Element

Get_element_index method (Appendix A. 8) helps to get the name of the changed element, telling the program in which dataframe it needs to go, in this example `"element": "gen"`, and finding the index of the element with the specified name, in this example `"Thermal PV"`. And then with the help of *change_node* method (Appendix A. 9) replaces the new data. After the replacement in the Pandapower network table, *Run_network* method for running the power flow calculation is called to execute.

Run_network method (Appendix A. 10) is responsible for executing the power flow calculation. This function is called at once at the beginning of the game, and then every time that an electric element is changed in the play mode of the app.

Beside running the power flow calculation, this method is also sending the results of the calculation out of the Pandapower class in a JSON format. The results are written in three types:

Results of buses, Result of lines, Result of DC lines. In addition, in cases that the power flow cannot converge an error is raised, which is handled in this method, and sends a string to Unity to execute desired actions. (For future development more error can be handled in this part)

The specific functions (event handler) are defined in Client class that they are responsible to change the format of JSON string to a string format which is ready to be encoded and be sent through network.

4.3.3. Sending data to Unity

Client Class is responsible for changing data to a readable format for pandapower and on the other side, Unity. Appendix A. 11 shows the UML diagram of this class. Whenever a message is received Client instance gets aware and parses the message to Pandapower containers. Whenever the result of the power flow calculation is ready to be sent Client gets informed and prepares the JSON format to.

When message is received its format is in a JSON string. *json.load()* method helps to change the format to a Python object which is readable for Pandapower tabular data structure. *json.dumps()* method helps to convert the a Python object to the string¹⁴. These functions are available in Appendix A. 12 to Appendix A. 15.

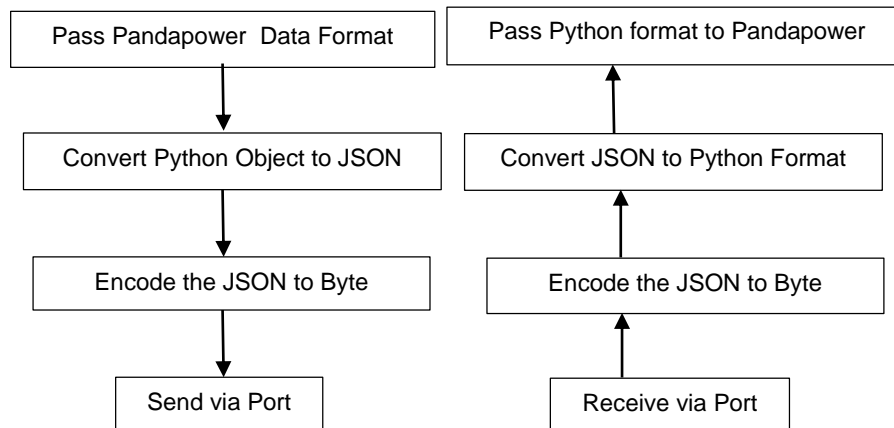


Figure 19. Data Formatting in Python

4.4. Unity Codes

The programming in Unity is divided into three parts, one part is dedicated to network connection to Python. One part is responsible for parsing and generating data from Pandapower format to C# format that is readable for Unity. and one last part is to writing an interface for interacting between UI and Pandapower.

4.4.1. Accepting Connection from Python

TcpServer class is developed for accepting the connection. The UML diagram is shown in Appendix A. 1 Network Class UML Diagram. This class is utilizing a built-in class, *TcpListener*, and its functions which enables Unity to listen for incoming connection attempts on the specified local IP address and port number. When a peer is connected, an event is invoked and inform the program that it is ready to transmit data between peers.

¹⁴ <https://docs.python.org/3/library/json.html>

4.4.2. Defining Pandapower Elements in Unity

For creating a power grid in Unity editor, game objects are created in the scene, and they are assigned to a specific electric element in Pandapower library. This is done by creating classes for each electrical element in Pandapower. The UML diagram is shown in Appendix A. 17.

Element.sc script is holding electric element definitions. For serializing and deserializing the data the variables' name of each class have to be exactly the same as its element's parameters in Pandapower. Figure shows an example of a bus data frame of a Pandapower network, and the its class written in Unity. To use more functionalities of any element, more parameters can be added to this class. For example, Zone can be added in Bus class in Unity. The undefined parameters will be filled in by default values in Pandapower (Appendix A. 19).

Element.sc also holds the classes and dictionaries for the results of the power flow calculation, which will be explained in detail.

	name	vn_kv	type	zone	in_service
0	bus 0	380.0	b	A	True
1	bus 1	380.0	b	A	True
2	bus 2	380.0	b	A	True
3	bus 3	380.0	b	A	True

Figure 20. Bus Data frame in Pandapower

4.4.3. Packaging and Converting Data for transmission

As explained before, the communication process can be abstract down to a ping-pong packet, Unity sends a “ping”, Python does the calculation and produces a “pong” packet which being sent back to Unity, then Unity updates the simulation. So packet is a basic unit responsible for communication, and needs to be broken down into its similar structures of data when it is received to its destination. This process is done by encoding and decoding the message to UTF-8. UTF-8 is an encoding that can translate any Unicode character to a matching unique binary string, and can also translate the binary string back to a Unicode character.

The JSON needs to be serialized to its data structure. Figure 21 describes the steps for data conversion. its data structure which is usable for program

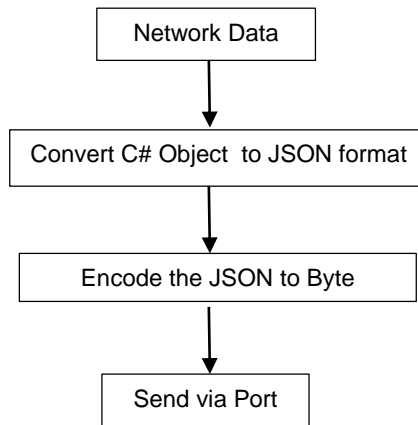


Figure 21. Data Formatting in C#

Packet resembles the data transmitted through the port. A data packet is a unit of data made into a single package that travels along a given network path. Every packet has two main parts. It holds the type of the received and sent objects. (Appendix A. 20).

TcpPeer class does the main task of converting data structures to an exchangeable format in Unity backend. The UML diagram is shown in (). The TCP protocol is a byte stream service. It simply takes the data, encapsulates it, and sends it to the remote peer. The two essential libraries for converting members or classes to and from JSON are *Newtonsoft.Json* and *Newtonsoft.Json.Linq*. Also *System.Net.Sockets* is a key library that Provides the underlying stream of data for network access. In Appendix A. 4 Writing on Stream the following methods are explained in details.

4.4.4. Defining Pandapower elements in Unity Editor

To make a network in Edit Mode, objects are created by adding 3D objects to the scene. To characterize objects, for example, a 3D sphere to be representative of an electric bus element, it requires to have all variables that a bus has. In this app, for each object representative of an element, a class is made. Appendix A. 21. explains an example for creating a class for each Pandapower element.

4.4.5. Interface between UI and Python

Everything that exists in the Unity scene can be accessed and gathered in a class named *NetworkManager*, which is working as an interface between UI and pandapower. This is the class which runs at the beginning of the game, gets all information of existing objects from the scene, sends the data one by one to Python, and then when the result data comes back from Python, this class updates the old values with the new values.

NetworkManager is subscribed to an event raising from *tcpPeer*, that gets informed whenever a packet is received in Unity and encoded result data is invoked. In this case, the method

OnResultRecieved() handles the event, updates the value of buses and lines with the new result values (Appendix A. 24)

4.4.6. **UI Design**

NodeInspector.sc is a script for controlling UI. In this script a System.Reflection is used that provides types of the parameters and retrieves information about member parameters, gives access to the data of a running program. Also a Raycast system is applied. The Physics.Raycast method is used to detect objects with colliders along a specified ray.

The UI designed for the app has two the advantage: It is automatically created and does not depend on the grid, and can be used for different girds. There are a different number of buses and elements the require to be appear in the scene which would fill a lot of space of the scene if it is not designed in a dynamic way. (Appendix A. 25)

5. SUMMERY AND FUTURE WORK

Power System Demonstrator app in 3D provides a 3D representation of power flow in a power network. It is aimed at giving a better insight to its audience illustrating how utilizing HVDC technology in a power system can improve the power flow quality. The simulation of the power network and the calculation of the power flow are done with Pandapower which is an open-source Python-based power system analysis tool. For animating power flow in a 3D environment, Unity application is utilized which is an engine for deployment of 3D games. Communicating between Pandapower solver and Unity has been the main objective of the project as these are developed in two different programming languages. An object oriented programming in Python and C# and a bi-directional network communication between Unity and Python have been picked for developing the codes.

The following are some key points worth to be highlighted to describe the contributions of the PSD app:

- It utilizes an open source power flow solver
- It has a real time simulation of results of the power flow calculation
- The power network contains both DC and AC lines

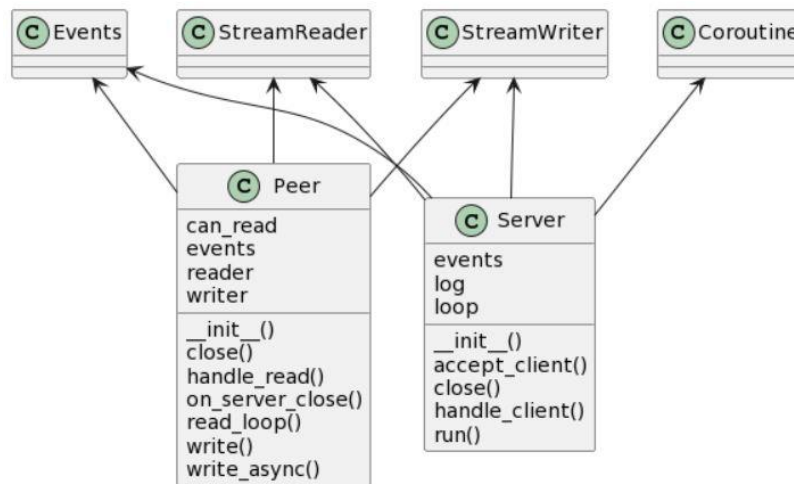
There are some aspects of the PSD application that can be improved in future: more functions of Pandapower can be utilized in the app, like including a visualization of cost function, by giving the user the option to change the cost of energy for calculating optimal power flow. Also the PSD app can be developed in another aspect. As PSD app is using a steady-state power flow analysis it lacks the time-domain analyses. To include visualization of power stability in a modern power network, a power flow analysis tool with the capability of time domain simulation can be used. PyPSA, ANDES, PYPOWER are the tools that can be taken into consideration.

Appendix

Appendix A. Coding Development

Appendix A. 1 Network Class UML Diagram	vi
Appendix A. 2 Running the Server	vi
Appendix A. 3 Accepting the connection of Unity	vi
Appendix A. 4 Writing on Stream.....	vi
Appendix A. 5 Reading from String.....	vii
Appendix A. 6 Pandapower Basic Function Exaples	vii
Appendix A. 7 Pandapower UML diagram.....	viii
Appendix A. 8 Get Element	viii
Appendix A. 9 Changing a Node in Network.....	viii
Appendix A. 10 Running Power Flow Calculation	ix
Appendix A. 11 Client Class UML Diagram	ix
Appendix A. 12 Comment Defined in Pandapower	x
Appendix A. 13 Parsing Received Data	x
Appendix A. 14 Creating Bus Result Format.....	x
Appendix A. 15 Sending Data Python Object.....	xi
Appendix A. 16 UML diagram of TCP server	xi
Appendix A. 17 A Pandapower Class in Unity	xi
Appendix A. 18 Element Class UML Diagram.....	xii
Appendix A. 19 Pandapower Commands Defined in Unity.....	xiii
Appendix A. 20 TCPPeer UML Class Diagram.....	xiii
Appendix A. 21 Packing and Converting Data in Unity	xiv
Appendix A. 22 BusView Class Example.....	xvii
Appendix A. 23 Results Dictionary.....	xvii
Appendix A. 24 Methods for Network Manager.....	xviii
Appendix A. 25 UI Scripting.....	xxiii
Appendix A. 26 Method for Power Flow Visulization.....	xxiv

Appendix A. 1 Network Class UML Diagram



Appendix A. 2 Running the Server

```

async def run(self, on_new_client):

    server = await asyncio.start_server(
        lambda reader, writer:
            self.accept_client(reader, writer, on_new_client),
        '127.0.0.1', 8858)
    addrs = ', '.join(str(sock.getsockname())
        for sock in server.sockets)
    print(f'Serving on {addrs}')
    self.loop = asyncio.get_event_loop()
    async with server:
        await server.serve_forever()

```

Appendix A. 3 Accepting the connection of Unity

```

async def accept_client(self, reader: StreamReader,
                        writer: StreamWriter, on_new_client):
    peer = self.handle_client(reader, writer)
    peer_name = writer.get_extra_info('peername')
    print(f'client connected: {peer_name}')
    self.log.info("")
    on_new_client(peer)

```

Appendix A. 4 Writing on Stream

```

async def write_async(self, jsn: str):
    print('sending: ' + jsn)
    data_bytes = bytearray(jsn, 'utf-8')
    size = len(data_bytes)
    print(f"Sending: {size!r}")
    size_bytes = int.to_bytes(size, byteorder=sys.byteorder, length=4)
    size_bytearray = bytearray(size_bytes)
    buffer = size_bytearray + data_bytes
    self.writer.write(buffer)
    await self.writer.drain()

```

Appendix A. 5 Reading from String

```
async def read_loop(self, reader: StreamReader):
    while self.can_read:
        received_bytes = await reader.readline()
        if len(received_bytes) < 1:
            continue
        json = received_bytes.decode()

        # Raise an Event and send the message out
        self.events.on_received(json)
```

Appendix A. 6 Pandapower Basic Function Exaples

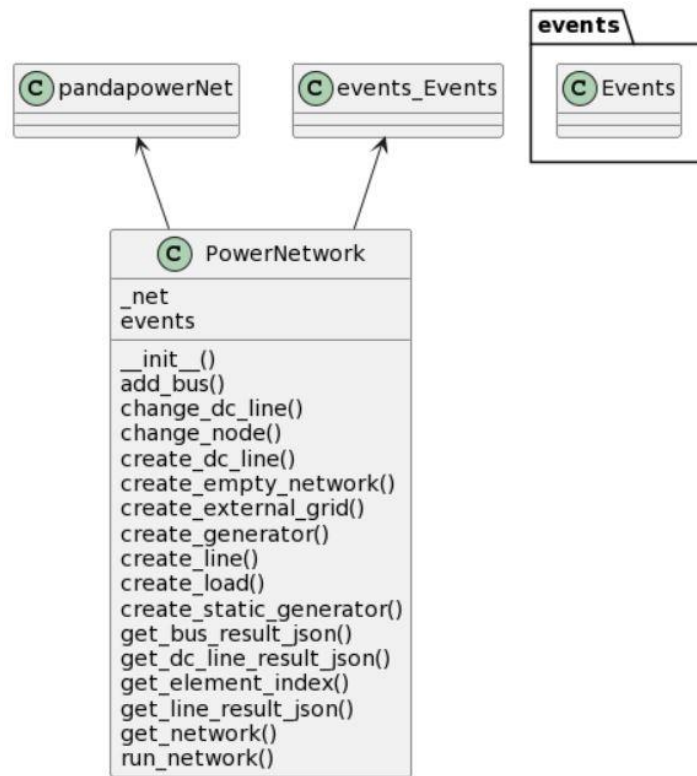
```
class PowerNetwork:
    # The pandapower format network
    _net: pandapowerNet
    events: events.Events
    def __init__(self):
        self.events = events.Events()

    # Initializes the pandapower datastructure.
    def create_empty_network(self, **kwargs):
        self._net = pp.create_empty_network(**kwargs)

    # Adds one bus in table net["bus"].
    def add_bus(self, **kwargs):
        pp.create_bus(self._net, **kwargs)

    # Creates an external grid connection.
    def create_external_grid(self, **kwargs):
        pp.create_ext_grid(self._net, **kwargs)
```

Appendix A. 7 Pandapower UML diagram



Appendix A. 8 Get Element

```

# Returns the element identified by a name
def get_element_index(self, element: str, name: str):
    for i in range(len(self._net[element]['name'])):
        if self._net[element]['name'][i] == name:
            return i
    return -1

```

Appendix A. 9 Changing a Node in Network

```

def change_node(self, element: str, **kwargs):
    index: int = self.get_element_index(element, kwargs['name'])
    for key, value in kwargs.items():
        print(f' key is {key} : value is {value}')
        self._net[element][key][index] = value
    self.run_network()

```

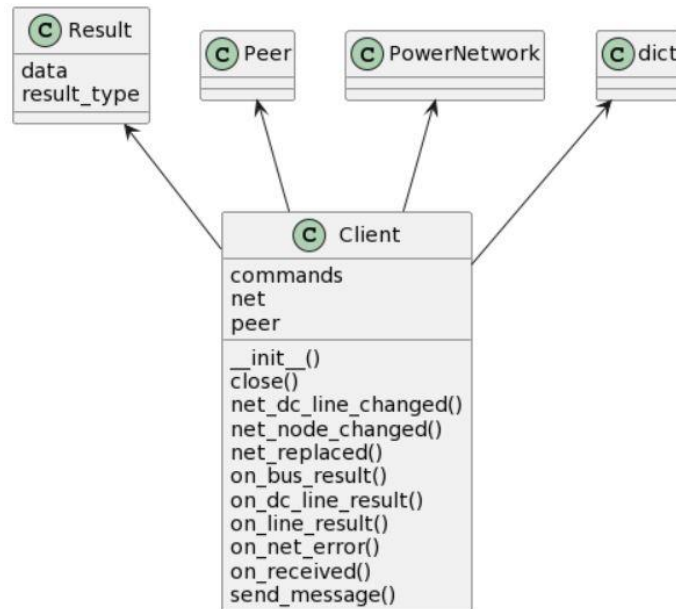
Appendix A. 10 Running Power Flow Calculation

```
# Runs a power flow
def run_network(self):
    try:
        pp.runpp(self._net)
    except LoadflowNotConverged:
        print("Error on Load flow Not Converged")
        self.events.on_error("Error: Load flow Not Converged")
        return

self.events.on_bus_result(self.get_bus_result_json())
self.events.on_line_result(self.get_line_result_json())
self.events.on_dc_line_result(self.get_dc_line_result_json())

# Saves the power flow bus results in JSON format
def get_bus_result_json(self):
    return self._net.res_bus.to_json()
```

Appendix A. 11 Client Class UML Diagram



Appendix A. 12 Comment Defined in Pandapower

```
self.commands = {
    0: self.net.create_empty_network,
    1: self.net.add_bus,
    2: self.net.create_generator,
    3: self.net.create_static_generator,
    4: self.net.create_external_grid,
    5: self.net.create_load,
    6: self.net.create_line,
    7: self.net.run_network,
    8: self.close,
    9: self.net_node_changed,
    10: self.net.create_dc_line,
    11: self.net_dc_line_changed,
    12: self.net_ac_line_changed,
    13: self.net.create_poly_cost,
    14: self.net.run_opp,
}
```

Appendix A. 13 Parsing Received Data

```
# Converts from json to python object
def on_received(self, packet_json: str):
# Deserilize json data to python object
    jsn = json.loads(packet_json)
    print(packet_json)
    index: int = jsn['message_type']
    if 'data' in jsn and jsn['data'] is not None and jsn['data'] != '':
        self.commands[index](**jsn['data'])
    else:
        self.commands[index]()
```

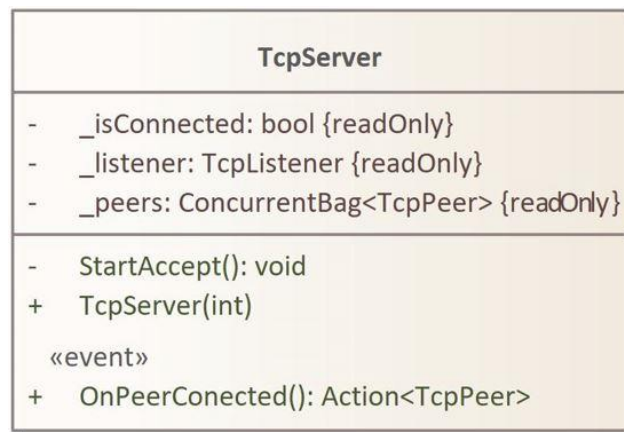
Appendix A. 14 Creating Bus Result Format

```
# Converts the power flow results to a bus result type
def on_bus_result(self, result_json: str):
    result = Result()
    result.result_type = BUS_RESULT
    result.data = result_json
    self.send_message(result)
```


Appendix A. 15 Sending Data Python Object

```
# Converts from python object to json
# Sends the power flow bus results to the peer
def send_message(self, result: Result):
# Serilize json data to python object
    jsn = json.dumps(result.__dict__)
    self.peer.write(jsn)
```

Appendix A. 16 UML diagram of TCP server

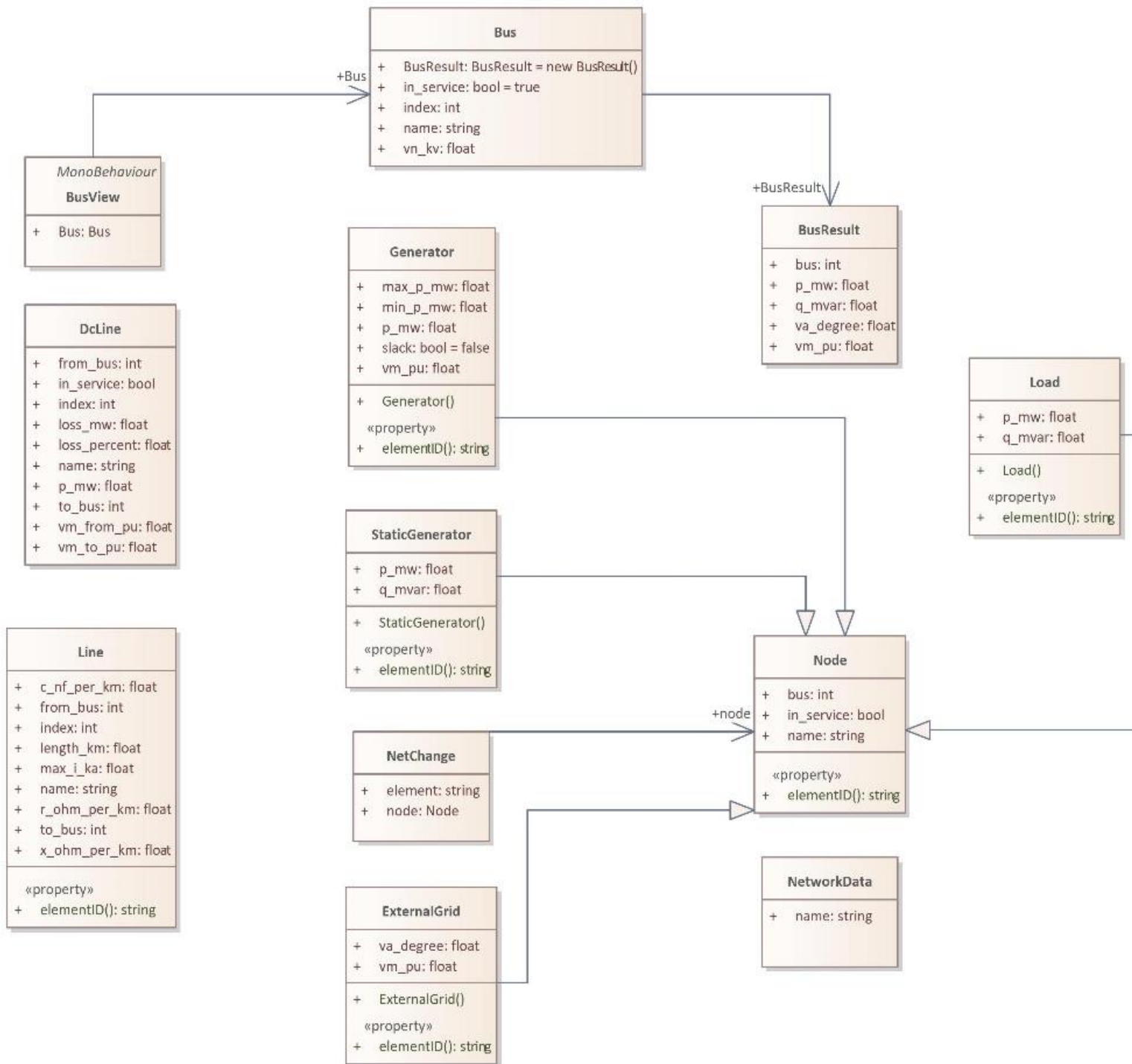


Appendix A. 17 A Pandapower Class in Unity

As mentioned the name of the variables must be the same as the name of the columns of the respective data frame. However, here for each bus another object of *BusResult* Class is instantiated. In this case, an error from pandapower that there is no name with *BusResult* to parse data into it. That is why *[JsonIgnore]* needs to be added and other classes.

```
public class Bus
{
    public int index;
    public string name;
    public float vn_kv;
    public bool in_service = true;
    [JsonIgnore]
    public BusResult BusResult = new BusResult();
}
```

```
class PowerNetwork
```



Appendix A. 19 Pandapower Commands Defined in Unity

```
public enum MessageType
{
    create_empty_network = 0,
    add_bus = 1,
    create_generator = 2,
    create_static_generator = 3,
    create_external_grid = 4,
    create_load = 5,
    create_line = 6,
    run_network = 7,
    close_connection = 8,
    node_net_changed = 9,
    net_replaced = 10,
    create_dc_line = 11,
    dcline_net_changed = 12
}
```

Appendix A. 20 TCPPeer UML Class Diagram



Send method:

This method, first converts the message to a *JSON* string, then encode the string to a list of byte, asynchronously writes a sequence of bytes to the current stream and monitors the cancellation requests.

```
public void Send(Packet message)
{
    // Serializes the specified object to a JSON string
    string json = JsonConvert.SerializeObject(message, JsonSetting);
    json += "\n";
    Log.Info($"Sending: {json}");
    // Represents a character encoding
    byte[] bytes = Encoding.UTF8.GetBytes(json);
    Log.Info($"Sending {bytes.Length} bytes");
    WriteBytes(bytes);
}
```

Write Bytes Method:

```
private async void WriteBytes(byte[] bytes, CancellationToken cancellationToken =
default)
{
    // Asynchronously writes a sequence of bytes to the current stream
    await _stream.WriteAsync(bytes, 0, bytes.Length, cancellationToken)
    .ContinueWith(task =>
    {
        if (task.IsCompleted)
        {
            Console.WriteLine($"{bytes.Length} bytes sent!");

            OnSendFinished?.Invoke();
        }
    }, cancellationToken);
}
```

Receive Method:

The main task of this method is that it reads the incoming byte stream, encodes it to a string format, and *JOBJECT* creates a *JSON* object with keys and values. This is done by *JTOKEN* and parse it to the type of result coming back from Python. For example, if the result is of type of a bus result, the data is parse in a busdataframe type. And at the end of the method the result is invoke to its responsible event handler. All results are handle in NetworkManager.sc (section 4.5.4) which decides what to do with results data, based on the type of the result. For example, if it is a line result, it updates the values in LineResult class.

(for the sake of simplicity the code is simplified here)

An example of the result of power flow calculation is shown below:

```
Recived json: {"result_type": 1, "data":
{"vm_pu": {"0": 1.02, "1": 1.01, "2": 1.012, "3": 1.0112840883, "4": 1.02}, "va_
_degree": {"0": 0.0, "1": -0.4859504238, "2": -1.1459164313, "3": -
1.7485159192, "4": 0.0}, "p_mw": {"0": -
200.7326120586, "1": 0.0, "2": 0.0, "3": 800.0, "4": -
607.340117964}, "q_mvar": {"0": -
140.1615558269, "1": 0.0, "2": 0.0, "3": 0.0, "4": -28.854216542}}}
```

```
private async Task Receive(CancellationToken cancellationToken = default)
{

    byte[] packetBytes = await ReadBytes(size, cancellationToken);
    string json = Encoding.UTF8.GetString(packetBytes, 0, packetBytes.Length);
    Log.Info($"Recived json: {json}");
    JObject jobj = JObject.Parse(json);
    JToken token = jobj["result_type"];
    Result result = default;
    ...

    if (int.TryParse(token.ToString(), out int type))
    {

        switch ((ResultType)type)
        {
            case ResultType.Line:
                try
                {
                    LineDataFrameResult lineResult = new LineDataFrameResult();
                    lineResult.result_type = ResultType.Line;
                }
            }
        }
    }
}
```

```

        string df_json = jobj["data"].ToString();
        lineResult.data = JsonConvert.DeserializeObject<LineDataFrame>(df_json);
        result = lineResult;
    }
    catch (Exception e)
    {
        Log.LogException(e);
        throw;
    }
    break;

case ResultType.DcLine:
    try
    {
        ....
    }

case ResultType.Bus:
    BusDataFrameResult busResult = default;
    try
    {
        .....
    }
    result = busResult;
    break;
case ResultType.Error:
    ErrorResult errorResult = new ErrorResult();
    errorResult.result_type = ResultType.Error;
    errorResult.message = jobj["data"].ToString();

    result = errorResult;
    break;
}
}
OnResult?.Invoke(result);

```

Appendix A. 22 BusView Class Example

BusView.sc is, for example a script that is added to a sphere, so it can have variables that a bus has. *BusView* is derived from *Monobehaviour*. It is the base class from which every Unity script derives. *Monobehaviour* makes all instances of a script execute in Edit Mode. By default, *MonoBehaviours* are only executed in Play Mode. By adding this attribute, any instance of the *MonoBehaviour* will have its callback functions executed while the Editor is in Edit Mode too.

```
public class GeneratorView : NodeView
{
    public override MessageType messageType{get => MessageType.create_generator;}
    public Generator GeneratorData;
    public override Node node
    {
        get => GeneratorData;
        set => GeneratorData = (Generator)value;
    }
}
```

Appendix A. 23 Results Dictionary

LineDataFrame	LineResult
+ i_from_ka: Dictionary<int, float>	+ i_from_ka: float
+ i_ka: Dictionary<int, float>	+ i_ka: float
+ i_to_ka: Dictionary<int, float>	+ i_to_ka: float
+ loading_percent: Dictionary<int, float>	+ loading_percent: float
+ p_from_mw: Dictionary<int, float>	+ OnLoadChanged: Action<float>
+ p_to_mw: Dictionary<int, float>	+ p_from_mw: float
+ pl_mw: Dictionary<int, float>	+ p_to_mw: float
+ q_from_mvar: Dictionary<int, float>	+ pl_mw: float
+ q_to_mvar: Dictionary<int, float>	+ q_from_mvar: float
+ va_from_degree: Dictionary<int, float>	+ q_to_mvar: float
+ va_to_degree: Dictionary<int, float>	+ ql_mvar: float
+ vm_from_pu: Dictionary<int, float>	+ va_from_degree: float
+ vm_to_pu: Dictionary<int, float>	+ va_to_degree: float
	+ vm_from_pu: float
	+ vm_to_pu: float
	«property»
	+ Load(): float

Start():

At the beginning of the Play Mode of the app all the object of the type of nodeView, lineView, dcLineView are found in the scene and gather to their respective lists. It is important that all objects sort in respect to their index number, otherwise there will be an error from Pandas data frame when adding items to the table.

```
private void Start()
{
    tcpPeer = new TcpPeer();
    tcpPeer.Connect(System.Net.IPAddress.Loopback, 8858);
```

Two subscriptions are defined in Start, one is to inform NetworkManager that TCP peer is connected, the other one is when a packet is received and encoded in TCP peer.

```
tcpPeer.OnConnected += TcpPeer_OnConnected;
tcpPeer.OnResult += TcpPeer_OnResultReceived;
```

It is important to add elements in a row by their index, an error is raised in pandapower for creation of the network:

```
nodeViews.AddRange(FindObjectsOfType<NodeView>());
nodeViews.Sort((n1, n2) => n1.BusView.Bus.bus.CompareTo(n2.BusView.Bus.bus));
lineViews.AddRange(FindObjectsOfType<LineView>());
lineViews.Sort((l1, l2) => l1.Line.index.CompareTo(l2.Line.index));

dclineViews.AddRange(FindObjectsOfType<DcLineView>());
dclineViews.Sort((l1, l2) => l1.dcLine.index.CompareTo(l2.dcLine.index));

BusView[] busViews = FindObjectsOfType<BusView>();
for (int i = 0; i < busViews.Length; i++)
{
    if (BusViewDic.ContainsKey(busViews[i].Bus.bus) == false)
        BusViewDic.Add(busViews[i].Bus.bus, busViews[i]);
    else Debug.LogError($"A Bus with the same ID already exists:
{busViews[i].gameObject}", busViews[i]);
}
for (int i = 0; i < nodeViews.Count; i++)
{
    NodeView nodeView = nodeViews[i];
```



```

        nodeView.OnNodeChanged = OnNodeChanged;
    }
    for (int i = 0; i < lineViews.Count; i++)
    {
        if (LineViewDict.ContainsKey(lineViews[i].Line.index) == false)
        { LineViewDict.Add(lineViews[i].Line.index, lineViews[i]); }
        else
        {
            Debug.LogError($"A line with the same Index already exists:
{lineViews[i].gameObject}", lineViews[i]);
        }
    }

    for (int i = 0; i < dclineViews.Count; i++)
    {
        if (DclineViewDict.ContainsKey(dclineViews[i].dcLine.index) == false)
DclineViewDict.Add(dclineViews[i].dcLine.index, dclineViews[i]);
        else
        {
            Debug.LogError($"A dc line with the same Index already exists:
{dclineViews[i].gameObject}", dclineViews[i]);
        }
        DcLineView dcViwe = dclineViews[i];
        dcViwe.OnDClineChanged = OnDClineChanged;
    }
}

```

When connection is successful TcpPeer_OnConnected is called, and is responsible to send messages for each object in the lists mentioned earlier:

```

private void TcpPeer_OnConnected()
{
    SendMessage(MessageType.create_empty_network, NetworkData);
    foreach (var bus in BusViewDic)
    {
        SendMessage(MessageType.add_bus, bus.Value.Bus);
    }
    for (int n = 0; n < nodeViews.Count; n++)
    {
        SendMessage(nodeViews[n]);
    }
}

```

```

    for (int i = 0; i < lineViews.Count; i++)
    {
        SendMessage(lineViews[i]);
    }
    for (int i = 0; i < dclineViews.Count; i++)
    {
        SendMessage(dclineViews[i]);
    }
    SendMessage(MessageType.run_network, string.Empty);
}

```

And the other event handler is when the result is invoked from TcpPeer, here it is handled, depending on the type of the result.

```

private void TcpPeer_OnResultReceived(Result result)
{
    switch (result)
    {
        case BusDataFrameResult busResult:
            UpdateBusResults(busResult);
            break;

        case LineDataFrameResult lineResult:
            UpdateLineResults(lineResult);
            break;

        case DcLineDataFrameResult dclineResult:
            UpdateDcLineResults(dclineResult);
            break;

        case ErrorResult errorResult:
            Loom.QueueOnMainThread(() => ResetScript.SetUpReset());

            break;
    }
}

```

As it shown in case of receiving an error from python the scene asks for restarting the scene and it starts the scene from the beginning. Here is where other codes can be added and also errors can be defined to be handle in a different way.

The flow of updating the line results data is demonstrated as an example. The JSON string that is coming from Python is as a type of dictionary. In Unity dictionaries are required for parsing the results into. At

first the result is parse into a responsible dictionary, LineResultDataFrame class, and then every line result variable has access to these data.

The important note here is that when the value of any variable in line result is changed, the program also needs to be informed to simulate it at the same time. So this class also contains events to invoke the values that are needed for simulation to the out of the class.¹⁵ For simulating the power flow arrows in the Play Mode, p_from_mw, q_from_mvar, p_to_mw, q_to_mvar are invoked when new values are set inside these variables. However, the problem is that these events are not called in the app, because they are using another thread and we cannot call into a running thread. Here comes the main-thread function call queue solution. Loom.sc is to run our functions on the main thread.^{16 17}

```
public class LineResult
{
    public Action<float> OnLineLoadingChanged;
    public Func<float, float> OnLinePfromChanged;
    public Func<float, float> OnLineQfromChanged;
    public Func<float, float> OnLinePtoChanged;
    public Func<float, float> OnLineQtoChanged;
    public float p_from_mw;
    public float q_from_mvar;
    public float p_to_mw;
    public float q_to_mvar;
    public float pl_mw;
    public float ql_mvar;
    public float i_from_ka;
    public float i_to_ka;
    public float i_ka;
    public float vm_from_pu;
    public float va_from_degree;
    public float vm_to_pu;
    public float va_to_degree;
    public float loading_percent;

    public float LinePfrom
    {
        get => p_from_mw;
        set
```

¹⁵ <https://stackoverflow.com/questions/4317479/func-vs-action-vs-predicate>

¹⁶ unity/Loom.cs at master · ufz-vislab/unity – GitHub

¹⁷ <https://forum.unity.com/threads/how-to-queue-a-main-thread-function-call-from-inside-a-job.1139047/>

```

        {
            if (value != p_from_mw)
                OnLinePfromChanged?.Invoke(value);
            p_from_mw = value;
        }
    }

    public float LineQfrom
    {
        get => q_from_mvar;
        set
        {
            if (value != q_from_mvar)
                OnLineQfromChanged?.Invoke(value);
            q_from_mvar = value;
        }
    }

    public float LinePto
    {
        get => q_to_mvar;
        set
        {
            if(value != q_to_mvar)
                OnLinePtoChanged?.Invoke(value);
            q_to_mvar = value;
        }
    }
}

```

UpdateLineResults () :

```

private void UpdateLineResults(LineDataFrameResult lineResult)
{
    LineDataFrame lineDataFrame = lineResult.data;
    for (int i = 0; i < lineDataFrame.i_from_ka.Count; i++)
    {
        LineResult lineRes = LineViewDict[i].LineResult;

        lineRes.pl_mw = lineDataFrame.pl_mw[i];
        lineRes.q1_mvar = lineDataFrame.q1_mvar[i];
        lineRes.i_from_ka = lineDataFrame.i_from_ka[i];
        lineRes.i_to_ka = lineDataFrame.i_to_ka[i];
        lineRes.i_ka = lineDataFrame.i_ka[i];
    }
}

```

```

lineRes.vm_from_pu = lineDataFrame.vm_from_pu[i];
lineRes.va_from_degree = lineDataFrame.va_from_degree[i];
lineRes.vm_to_pu = lineDataFrame.vm_to_pu[i];
lineRes.va_to_degree = lineDataFrame.va_to_degree[i];

float lpfrom = lineDataFrame.p_from_mw[i];
Loom.QueueOnMainThread(() => lineRes.LinePfrom = lpfrom);

float lqfrom = lineDataFrame.q_from_mvar[i];
Loom.QueueOnMainThread(() => lineRes.LineQfrom = lqfrom);

float lpto = lineDataFrame.p_to_mw[i];
Loom.QueueOnMainThread(() => lineRes.LinePto = lpto);

float lqto = lineDataFrame.q_to_mvar[i];
Loom.QueueOnMainThread(() => lineRes.LineQto = lqto);

float loader = lineDataFrame.loading_percent[i];
Loom.QueueOnMainThread(() => lineRes.LineLoad = loader); }}

```

Appendix A. 25 UI Scripting

As explained in Chapter 4, all electric objects in the scene requires to have a collider attached to them. A prefab is also created that shows these parameters in the scene. The created prefab is called FieldDrawer with FieldDrawer script attached to it. The prefab has following parameters that are created in Unity UI:

```

public class FieldDrawer : MonoBehaviour
{
    public Text label;
    public InputField inputField;
    public Toggle toggle;
}

```

Whenever the user clicks on the electric element in the scene, the program checks the type the element. There are two types are defined, either a bus, for representing the results of the bus, or NodeView, to show the elements connected to a specific bus.

```

if (Physics.Raycast(ray, out RaycastHit hitInfo))
{

```

```

if (hitInfo.transform != null)
{
    NodeView n = hitInfo.transform.GetComponent<NodeView>();

    if (n != null)
    {
        SetNodeView(n);
    }

    BusView b = hitInfo.transform.GetComponent<BusView>();

    if (b != null)
    {
        SetBusView(b);
    }
}

```

Depending on what user has clicked on the respective function is called.

GetType() is a built-in method that return these values stored in each parameters of the element. Depending on the type of the parameter, either toggle or input field of prefab is created automatically and the values are filled out on the field that appears on the scene in the play mode:

```

fieldDrawer.toggle.gameObject.SetActive(true);

fieldDrawer.inputField.gameObject.SetActive(false);

fieldDrawer.toggle.SetIsOnWithoutNotify

((bool)fieldInfo.GetValue(_nodeView.node));

```

When the state of the toggle or input field is changed the respective methods are called, these methods new values, notify the program that value has changed and then the new value are packed to send to Python for new calculation:

```

fieldDrawer.toggle.onValueChanged.AddListener(b =>

OnFieldChanged(fieldInfo, b));
}

```

Appendix A. 26 Method for Power Flow Visualization

The methods for visualization is written in LineView.sc script. LineView are attached to PowerLinePrefabs. Whenever the Result of power flow calculation is back to Unity, the Values of each Line is updated in LineResult class. The parameters p_from_mw, q_from_mw,

p_to_mw, and q_to_mw are collected from each line. The direction of the arrows for simulation is defined by the sign of the p_from_mw. The scale of the arrows is changing based on the maximum amount between apparent power at each side of the line:

```
float power = Mathf.Max(Mathf.Sqrt(Mathf.Pow((float)pFrom, 2) +
Mathf.Pow((float)qFrom, 2)), Mathf.Sqrt(Mathf.Pow((float)pTo, 2) +
Mathf.Pow((float)qTo, 2)));
```

The bus results are updated in BusResult class after each time of power flow calculation and they are accessible with the same way. For the future development, these data can be also simulated in the play mode.

Appendix B. Create a Pandapower Network in Unity Editor

To create a network, first a new scene is created in Unity Editor, where GameObjects can be added into this scene. GameObjects are fundamental objects which each of them represents a specific electric element, as it is defined in pandapower network dataframe.

“A GameObject itself is only a 3D solid object, and they do not accomplish much themselves but they act as containers for components, which implement the functionality”¹⁸. To give a GameObject the properties it needs to become a specific element, the specific script is attached to it. Different scripts are developed in developing phase and depending on what kind of element is needed, the respective is added to the GameObject. Developing of these scripts, their functionality and the explanation of the codes are described in Chapter 4.

To begin with, an empty GameObject is created using menu `GameObject > Create Empty`.

NetworkManager script is attached to it as a component to it. The fields in Network Data require to be filled out in Unity Spector. To avoid from any error, all the fields required must be provided as shown in Figure 22. Now the structure of power system network with format of a power network in pandapower is created. NetworkManager gives the access to all functions for starting the game. Loom script is also needed to be attached to network GameObject (the functionality of the scripts is explained in details in chapter 3)

¹⁸ <https://docs.unity3d.com/Manual/GameObjects.html>

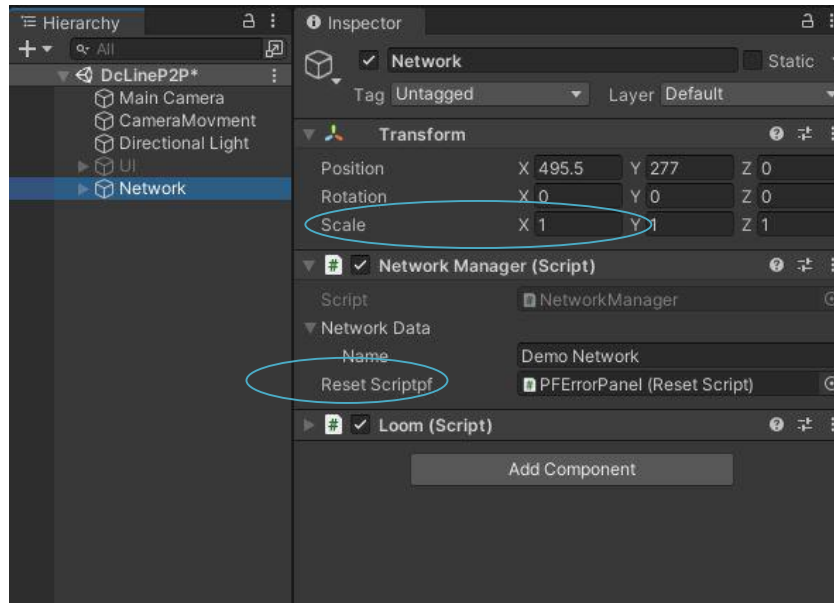


Figure 22. Adding a script to an empty GameObject

The next step is to set the whole network in the scene. For example, for creation of a bus, a solid sphere is added in hierarchy window and BusView script is attached to it. Then the properties that are defined for a bus element are accessible from Inspector Window. These properties are as the same as a bus data frame in Pandapower (Figure 23).

	name	vn_kv	type	zone	in_service
0	bus 0	380.0	b	A	True
1	bus 1	380.0	b	A	True
2	bus 2	380.0	b	A	True
3	bus 3	380.0	b	A	True

Figure 23. Bus Dataframe in Pandapower

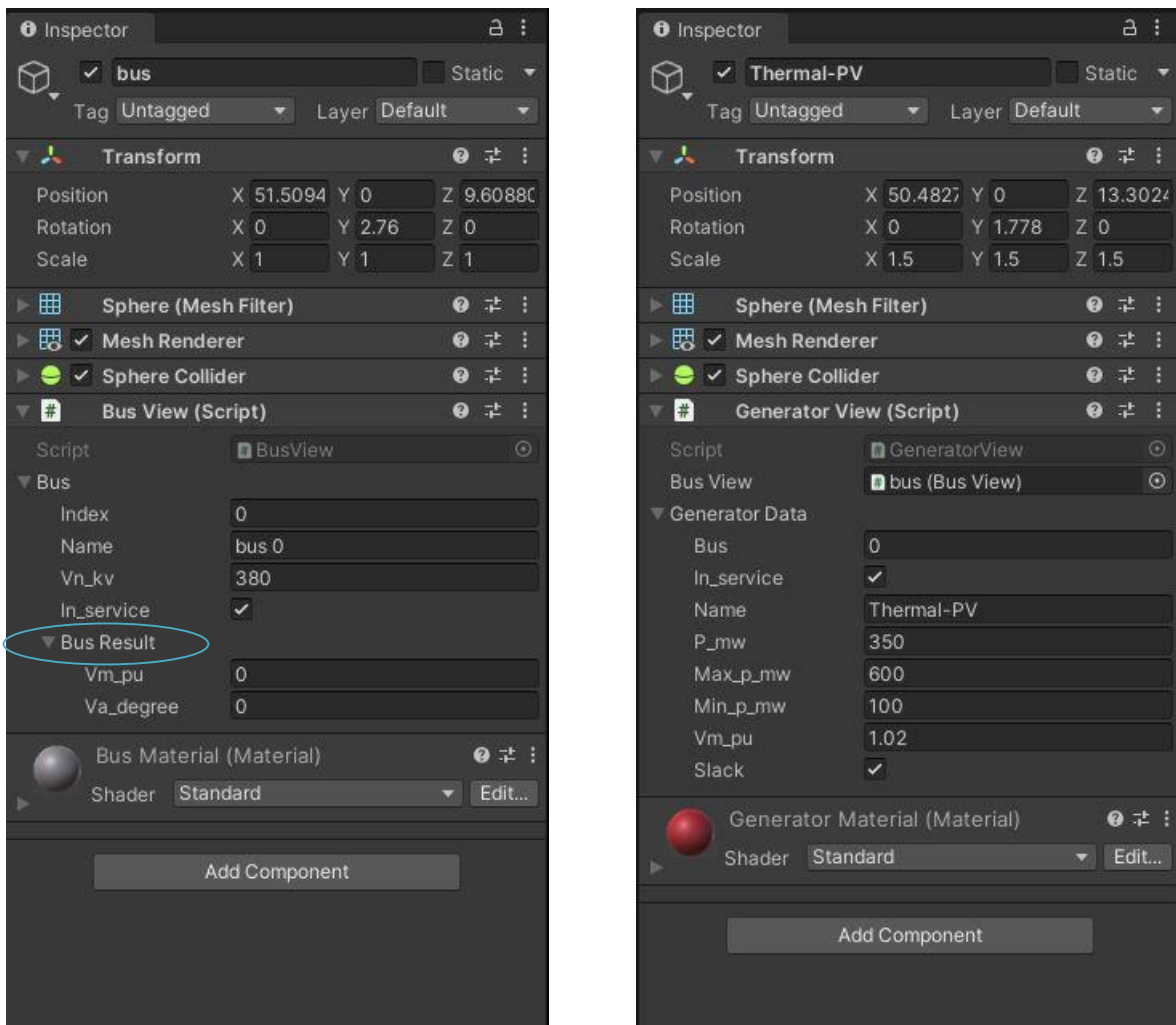


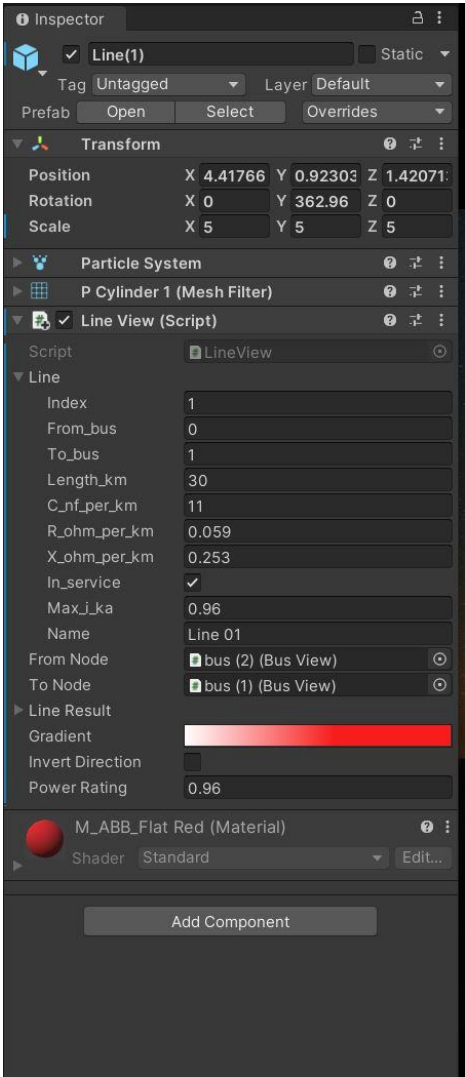
Figure 24. bus and generator GameObject with Components

As it can be seen a ‘BusView’ script also has Bus Results, which are not required to be filled out. Bus Result properties are being upload after starting the game. A bus can also have a material, which can be added from Material Folder. With the same process other electric element of the network can be created and defined: busses, generators, static generators, and loads in the scene attaching ‘BusView’, ‘GeneratorView’, ‘StaticGeneratorView’, and ‘LoadView’ respectively to the objects.

For the line connecting two busses, a 3D model is used. However, the parameter of a line is given to prefab, named ‘PowerLinePrefab’. “A prefab is used when an object is reused a in a particular way, in multiple places in the scene”¹⁹. ‘PowerLinePrefab’ is created with Particle System component and their purpose is to simulate the flow of energy into the line by generating and animating small arrows in the scene. The Particle System component has many properties, which can be adjusted in Inspector. To give the ‘PowerLinePrefab’ prefab the functionality of a pandapower line, a ‘LineView’ script is

¹⁹ <https://docs.unity3d.com/Manual/Prefabs.html>

attached to it. Based on the information of the line element in pandapower network the required fields are filled out. Figure 25 depicts ‘Line-01’ data in pandapower. The field ‘From Node’ and ‘To Node’ is also needs to completed by dragging ‘bus_1’ GameObject and ‘bus_2’ GameObject, from hierarchy to these fields as it is shown in Figure 26. In ‘LineView’ script the Line Results will be updated after starting the game.



```
name                Line 01
std_type            490-AL1/64-ST1A 380.0
from_bus            2
to_bus              1
length_km           30.0
r_ohm_per_km        0.059
x_ohm_per_km        0.253
c_nf_per_km         11.0
```

Figure 25. line data in pandapower

```
type                o1
in_service           True
Name: 1, dtype: object
```

Figure 26. AC line and it components

Figure 27. shows an example of line model between two busses. ‘PowerLinePrefab’ are shown as red arrows in this picture.

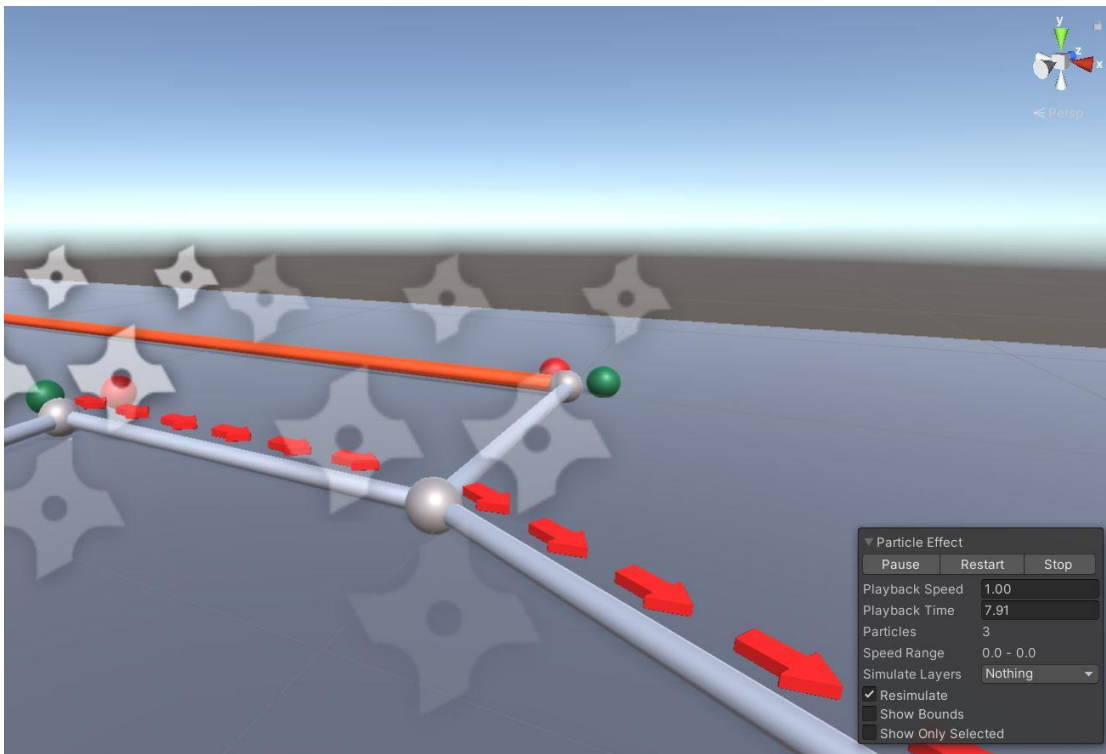


Figure 27. Line models and Particles

Appendix C. Create UI in Unity Editor

The UI for the application is designed by RayCast in Unity. The RayCast function makes it possible to select an object in the game in real-time by clicking on it. It projects a Ray into the scene, returning a boolean value if a target was successfully hit. When that happens, the information of the GameObject pop up in the screen (The information only pop ups on the screen and user have access to them, and can vary the values. However, for further development, the information can be visualized in a graphical form or be animated. When using a RayCast it is needed to attach a Collider Component to each GameObject in the scene (the ones that their information is supposed to seen in Play mode). So the busses, generators, static generators, and lines require a respective Collider Component depending on their shape. RayCast is triggered by clicking on the objects. In Figure 28a Mesh Collider is selected for a dc Line.

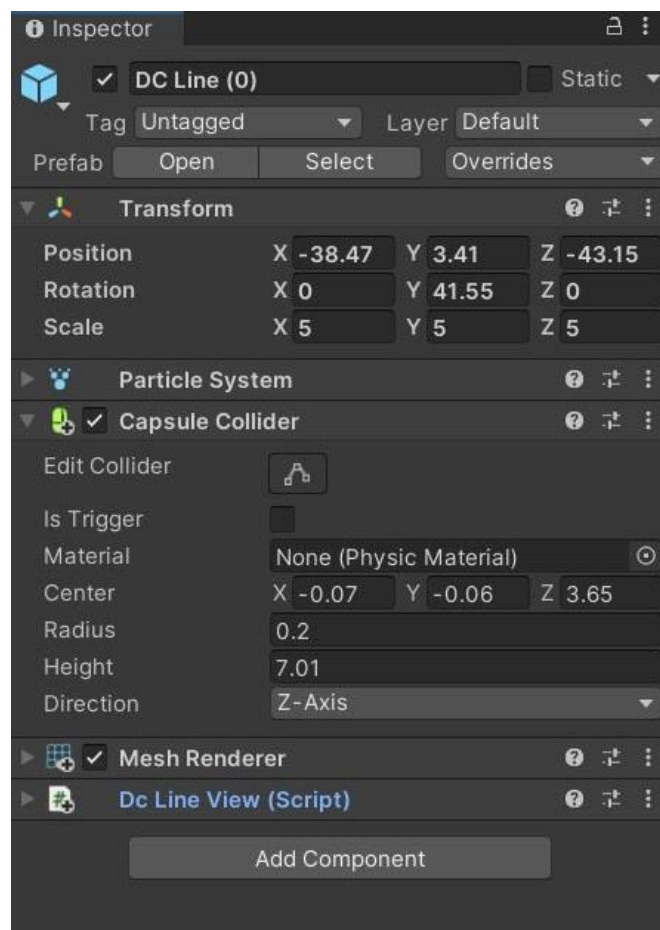


Figure 28. Adding Collider Component

Bibliography

- [1] G. Flisberg, C.-G. Carlsson, R. Moni and A. B. Boveri, Global Trends in Bulk Power Transmission, Ludvika, Sweden: ABB Utilities AB., S-77180.
- [2] Elansari, A.S. and Finney, S.J. and Burr, J. and Edrah, M.F., "Frequency control capability of VSC-HVDC transmission system," 2015.
- [3] Ying Jiang-Hafner and Duchon, Hugo and Karlsson, Michael and Ronstrom, Leif and Abrahamsson, Bernt, "HVDC with Voltage Source Converters ñ," in *IEEE/PES Transmission and Distribution Conference and Exposition*, 2008.
- [4] J. Liang, G. Li, L. Zhang, W. Tang, Y. Cai and W. Sheng, "Upgrading the Power Capacity of a Three-Conductor MVAC," *IEEE Transactions on Smart Grid*, 2018.
- [5] "ABB Review Special Report," ABB Technology Ltd, Zurich/Switzerland, 2014.
- [6] . J. Grainger and W. Stevenson, "Power System Analysis," New York: McGraw–Hill, ISBN 0-07-061293-5, 1994.
- [7] B. Stott and O. Alsac, "Fast Decoupled Load Flow," *IEEE Transactions on Power Apparatus and Systems*, Vols. PAS-93, no. 0018-9510, pp. 859-869, 1974.
- [8] K. ., (. Volkov, Computational Models in Engineering, London, United Kingdom,: IntechOpen, 2020.
- [9] H. Cui and Y. Zhang, "Andes_gym: A Versatile Environment for Deep," March 2022.
- [10] H. Cui, F. Li and K. Tomsovic, Hybrid Symbolic-Numeric Library for Power System Modeling and Analysis, 2020.
- [11] Thurner, Leon and Scheidler, Alexander and Schäfer, Florian and Menke, Jan-Hendrik and Dollichon, Julian and Meier, Friederike and Meinecke, Steffen and Braun, Martin, "Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems," *IEEE Transactions on Power Systems*, vol. 33, pp. 6510-6521, Nov. 2018.
- [12] W. McKinney, "pandas: a foundational python library for data analysis," *Python for High Performance and Scientific Computing*, 2011.
- [13] G. Andersson, "Lecture on Modelling and Analysis of," 2017.
- [14] T. Sauer, Numerical Analysis, Pearson Education, 2006.
- [15] R. Sharma and J. Dhillon, "PyPSA: Open Source Python Tool for Load Flow," in *Journal of Physics*, 2021.
- [16] L. Thurner, A. Scheidler, F. Schäfer, J.-H. Menke, J. Dollichon, F. Meier, S. Meinecke and M. Braun, "Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems," pp. 6510-6521, Nov 2018.
- [17] S. Long and S. Nilsson, "HVDC Transmission Yesterday and," *IEEE power & energy magazine*,, pp. 22-23, 2007.
- [18] "The evolution of HVDC: Transmitting bulk power," [Online]. Available: <http://www.abb.com/cawp/db0003db002698/02de19e1cb36dbd4c12572f400>. [Accessed 2 Dec 2009].

- [19] B. ANDERSEN and C. BARKER, "A new era in HVDC," *IEE Review*, vol. 46, no. 0953-5683, 2000.
- [20] H. t. f. c. o. p. flow, "ABB," [Online]. Available:
<http://www.abb.com/industries/db0003db004333/8e6848f68d2d204fc125748>.
- [21] M. AHRMAN and B. JOHNSON, "The ABCs of HVDC transmission technologies," *IEEE power & energy magazine*, vol. 5, no. 1558-4216, pp. 23-44, March 2007.
- [22] HVDC transmission for lower investment cost, "ABB," [Online]. Available:
<http://www.abb.com/industries/db0003db004333/678bb83d3421169dc12574>. [Accessed 2010].
- [23] "Influence of Embedded HVDC Transmission on System Security and AC Network Performance," *CIGRE*, no. 978-2-85873-230-2, 2013.
- [24] G. Flisberg, R. Moni, C.-G. Carlsson and A. Brown Boveri, "Global Trends in Bulk Power Transmission," [Online]. Available:
https://library.e.abb.com/public/5e44e9e29147761ac1256fda004aeac0/mumbai_conf.pdf. [Accessed 28 02 2022].
- [25] M. Albadi, *Power Flow Analysis in Computational Models in Engineering.*, London, United Kingdom: IntechOpen.
- [26] G. C. C. & M. R. Flisberg, Flisberg, G., Carlsson, C., & Moni, R.S. (2001). *Global Trends in Bulk Power Transmission.*, 2001.
- [27] M. Albadi, *Power Flow Analysis in Computational Models in Engineering.*, London, United Kingdom: IntechOpen.
- [28] F. L. a. K. T. H. Cui, "Hybrid Symbolic-Numeric Framework for Power System Modeling and Analysis," *IEEE Transactions on Power Systems*, vol. 36, pp. 1373-1384.
- [29] U. o. K. G. Department of Energy Management and Power and D. f. D. S. O. Fraunhofer Institute for Wind Energy and, "- Convenient Power System Modelling and Analysis," Fraunhofer IWES, Kassel, 2017.
- [30] L. a. S. A. a. S. F. a. M. J.-H. a. D. J. a. M. F. a. M. S. a. B. M. Thurner, "Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems," *IEEE Transactions on Power Systems*, vol. 33, pp. 6510-6521, 2018.
- [31] ". o. t. a. a. f. o. o. f. p. s. i. I. F. Milano and L. Vanfretti.
- [32] R. S. a. J. Dhillon, "PyPSA: Open Source Python Tool for Load Flow Study," in *Journal of Physics*, 2021.
- [33] L. T. a. A. S. a. F. S. a. J. H. M. a. J. D. a. F. M. a. S. M. a. M. Braun, "pandapower-Convenient Power System Modelling and Analysis based on PYPOWER and pandas," Fraunhofer IWES Universität Kassel, 2018.
- [34] Department of Energy Management and Power, Fraunhofer Institute for Wind Energy and, "Convenient Power System Modelling and Analysis," Fraunhofer IWES Universität Kassel, Kassel, 2017.
- [41] Elansari, A.S. and Finney, S.J. and Burr, J. and Edrah, M.F., 11th IET International Conference on AC and DC Power Transmission, 2015.
- [42] A. Vijayvargia, S. Jain and S. Meena, "Comparison between Different Load Flow," *International Journal of Electrical Engineering*, vol. 9, no. 0974-2158, pp. 127-138, 20016.

