# Contents

# 1. File tree

```
.
+-- docs
|   +-- developer_documentation
|   |   +-- developer_documentation.pdf
|   |   +-- developer_documentation.tex
|   +-- specification
|   |   +-- specification.pdf
|   |   +-- specification.tex
|   +-- user_documentation
|       +-- error_finder.png
|       +-- generate_votes.png
|       +-- run_algorithm.png
|       +-- save_as.png
|       +-- user_documentation.pdf
|       +-- user_documentation.tex
|       +-- user_interface_mane.png
+-- README.md
+-- src
|   +-- main.py
|   +-- misra_gries.py
|   +-- preflib_vote_parsers
|   |   +-- copeland_parse.py
|   |   +-- minimax_parse.py
|   |   +-- plurality_parse.py
|   |   +-- stv_parse.py
|   +-- proceess.py
|   +-- sampling.py
|   +-- vote_generator.py
|   +-- vote_rules
|       +-- brute_force
|           +-- graph.py
|           +-- graph_voting.py
|           +-- plurality.py
|           +-- stv.py
|           +-- vector.py
+-- test
    +-- all_test.py
    +-- graph_voting_test.py
    +-- misra_gries_test.py
    +-- plurality_test.py
    +-- sampling_test.py
    +-- stv_test.py
```

This is whole file tree of the project, but we will focus just on "src" and "test" folders. That means this:

```
+-- src
|   +-- main.py
|   +-- misra_gries.py
|   +-- preflib_vote_parsers
|   |   +-- copeland_parse.py
|   |   +-- minimax_parse.py
|   |   +-- plurality_parse.py
|   |   +-- stv_parse.py
|   +-- proceess.py
|   +-- sampling.py
|   +-- vote_generator.py
|   +-- vote_rules
|       +-- brute_force
|           +-- graph.py
|           +-- graph_voting.py
|           +-- plurality.py
|           +-- stv.py
|           +-- vector.py
+-- test
    +-- all_test.py
    +-- graph_voting_test.py
    +-- misra_gries_test.py
    +-- plurality_test.py
    +-- sampling_test.py
    +-- stv_test.py
```

# 2. SRC

## 2.1 Main

Used for handling the input interface so most of the functions are for generating windows with Tkinter.

## 2.2 Process

This class is basicly connector between all the other classes, it calls all the algorithms and connect them with the interface.

## 2.3 Misra Gries

## 2.4 Sampling

## 2.5 Vote generator

## 2.6 Vote rules

### 2.6.1 Graph + graph voting

### 2.6.2 Vector + plurality

### 2.6.3 STV

## 2.7 Preflb vote parsers

All the parsers work basicly the same they initilize the vote rule and convert the lines of Preflib file into something that the vote rules implementation understand.

# 3. Test

Tests are basic unit tests, that looks something like this:

```
import unittest
from src.vote_rules.brute_force.stv import stv

def test_stv1(self):
  candidates = [1, 2, 3, 4, 5]
  votes = [[4, 2, 3, 1, 5], [3, 1, 2, 4, 5], [4, 1, 3, 5, 2], [5, 1, 3, 2,
      4]]
  S = stv(candidates, votes)
  expected_output_1 = [4, 3, 5, 1, 2]  # Expected output dictionary
  self.assertEqual(S.stv(), expected_output_1)
```

The only not so easy tests ware tests for sampling, since sampling is randomized, so we needed to add something called error marginand run it on big enough data,. That basicly means, how much error we allow, so we let error margin be 0.09.