Team1

Voting System

Software Design Document

Name(s):

Jake Waro (warox001)
Sami Frank (fran0942)
Allison Miller (mill7079)
Declan Buhrsmith (buhrs001)

Date: 02/27/2020

Table of Contents

1. INTRODUCTION

1.1 Purpose

The purpose of this software design document (SDD) is to provide design details for the Voting System program. The document highlights the system architecture, data structures, activity diagrams and workflows, and user interfaces of the Voting System. The intended audience of this program are election officials responsible for processing a country's elections, in which the country uses a Single Transferable Vote (STV) or a Plurality vote.

1.2 Scope

This SDD highlights the complete design for the Voting System. It is a stand-alone Java program, requiring no internet connection, and runs local to the machine. The program can be executed from either the command line, or from within an integrated development environment (IDE) such as Eclipse. A text-based interface is used, and the user is not expected to interact with the program via the command line interface (CLI).

The developer/testers of the program will be able to include a command line argument to disable shuffling of votes to test the program. The election officials will be able to input the type of vote, the number of seats to fill in the election, the file(s) containing the ballots, and will be able to kick-off the election processing. The processing of the ballots should only be run once for an election. The user will then have access to summary information about the election's results via the text-based interface and will have access to an audit file that tracked how the election was processed.

1.3 Overview

The following sections of the document provide in-depth details about how the system will be designed.

- **Section 2** gives a general description of the functionality, context and design of the Voting System. Any additional background information about the system will be listed here.
- **Section 3** details the architecture of the system. It describes both the modular components of the program as well as how they are associated. Each component has an abstract description, detailing its visibility, implementation, data, and methods. This section strives to give the reader a high level overview of the system's composition.
- **Section 4** provides an overview about the design of the data structures.

- **Section 5** provides diagrams for the components of the system. These diagrams highlight workflows and logic for the different use cases.
- **Section 6** discusses the user interfaces that will be used in the program. This section contains mock-up designs that allow the reader to picture what the interface may look like and how one might interact with it.
- **Section 7** gives a mapping between the system's design and its components to the use cases and functional requirements of the software requirements specification (SRS) document.
- **Section 8** is an index that provides more information on details that may be assumed as prerequisite knowledge, and highlights tangential information in more depth.

1.4 Reference Material

IEEE standard 1016-1998 was used to format this SDD; following is its respective reference:

- IEEE Recommended Practice for Software Design Descriptions," in IEEE Std 1016-1998 , vol., no., pp.1-23, 4 Dec. 1998

1.5 Definitions and Acronyms

Below is a list of acronyms and terms used throughout the document and their respective definitions:

- **CLI** - Command line interface
- **Droop Quota** - upper bound of number of votes to receive in a STV election; if a candidate meets the droop quota, they become a winner, their ballots become removed, and they no longer receive votes.
- **IEEE** - Institute of Electrical and Electronics Engineers
- **IDE** - Integrated Development Environment
- **SDD** - Software Design Document
- **SRS** - Software Requirements Specification
- **STV** - Single Transferable Vote; ranked choice voting

2. SYSTEM OVERVIEW

The Voting System will be implemented as a java application, and does not require connection to any external servers. Figure 1 highlights the general discourse of the program.

General constraints include that the system should be able to process the votes within five minutes. The training time for the application should be minimal, requiring less than fifteen minutes.

There are no security considerations to be made; any concerns are assumed to be addressed elsewhere by election officials. Files are assumed to have been transferred to a local directory in a secure manner, and all users of the system are assumed to have the proper authority to use the application.

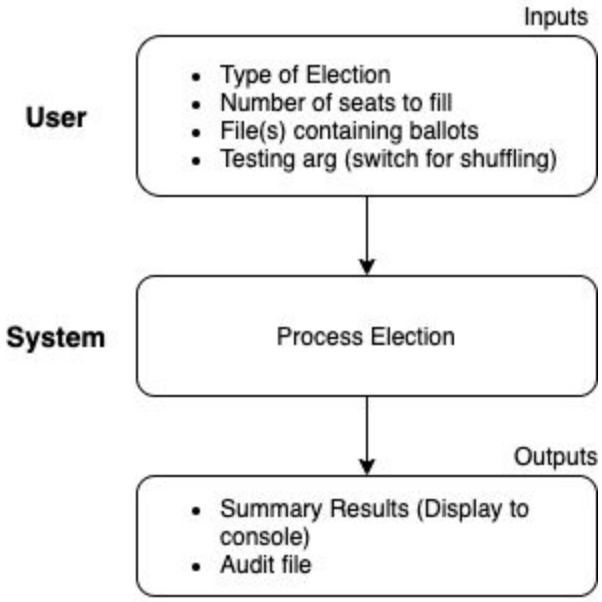


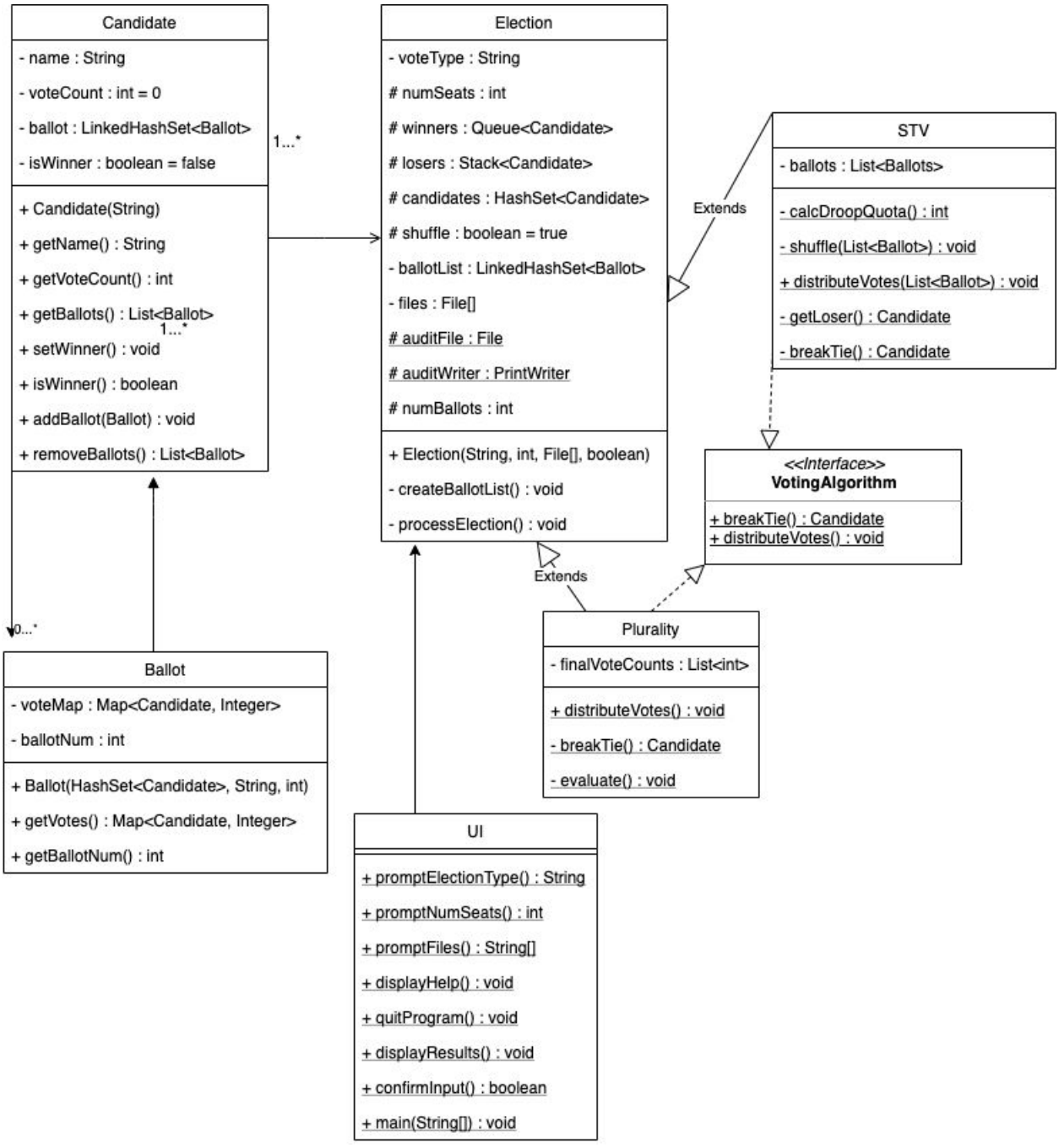
Figure 1 Overview of the Voting System

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The following is a diagram of the Voting System; it shows how the different components of the system are connected. Each class in the diagram is detailed with its respective fields, methods, and their respective visibilities.

System Architecture Diagram



3.2 Decomposition Description

UI

- Voting System entrance
- Collect Election Information
- Quit
- Get Help
- Display Election Results
- **Election**
    - **Candidate**
        - Represent Candidate
    - **Ballot**
        - Represent Ballot
    - **STV**
        - Distributes Votes
        - Calculate Droop Quota
        - Shuffle Votes
        - Break Ties
    - **Plurality**
        - Distributes Votes
        - Calculates Winner(s)
        - Breaks Ties

3.3 Design Rationale

The Voting System is designed in such a way to structure the needs of the system. One need of the system is that multiple ballots have to be stored and tabulated. The best way to do this is to create a ballot class to create consistency in the system when it comes to auditing and tabulating. Another major structure is having a candidate class which will hold information about each person running in the election. The overarching class used is the systems election class, which will take in user input in order to process the election. The team found this the best way to manage the necessary tasks in order to tabulate an election.

4. DATA DESIGN

4.1 Data Description

The main data the Voting System uses is the ballot file(s). These file(s) come in the form of a Windows CSV. The first line of the file will have information pertaining to who is running in that particular election. Each subsequent line (until the end of the file is reached) represents one ballot, which is assumed to be formatted correctly for the respective type of vote.

This first line will be parsed into ‘n’ amount of Candidate class instances. Each instance of the Candidate class will store information pertaining to its respective candidate. This will include:

1. A HashSet of ballot objects.
    a. Allows us to keep track of which ballots are assigned to which candidate.
2. An Integer variable voteCount.
    a. Will mainly be used when the plurality algorithm is run as a way to keep track how many votes are assigned to a candidate.

All subsequent lines of the inputted file(s) will refer to ballots. There will be one ballot per line until the end of file is reached. When the file is processed, each line that contains a ballot will become a new instance of the Ballot class. The Ballot class will store information such as:

1. A variable ballotNum, which will keep track of the number of total ballots, and the order in which each ballot was counted.
2. A map that associates ranks to candidates.

election is processed:

1. Using STV
    a. Votes will be distributed, until the winner(s) is/are determined
        i. Winner status will be stored as a boolean value in the respective candidate object.
2. Using Plurality
    a. An evaluation will be run, which will go through the votes stored in each candidate, determining the winner(s) based on the number of total votes.
        i. Winner status will be stored as a boolean value in the respective candidate object.

After the election is processed:

1. Results are shown on the screen
2. A note about where to find the audit file is produced.

4.2 Data Dictionary

Major Data:

- Ballots
- Candidates
- File(s)
    - Windows CSV formatted

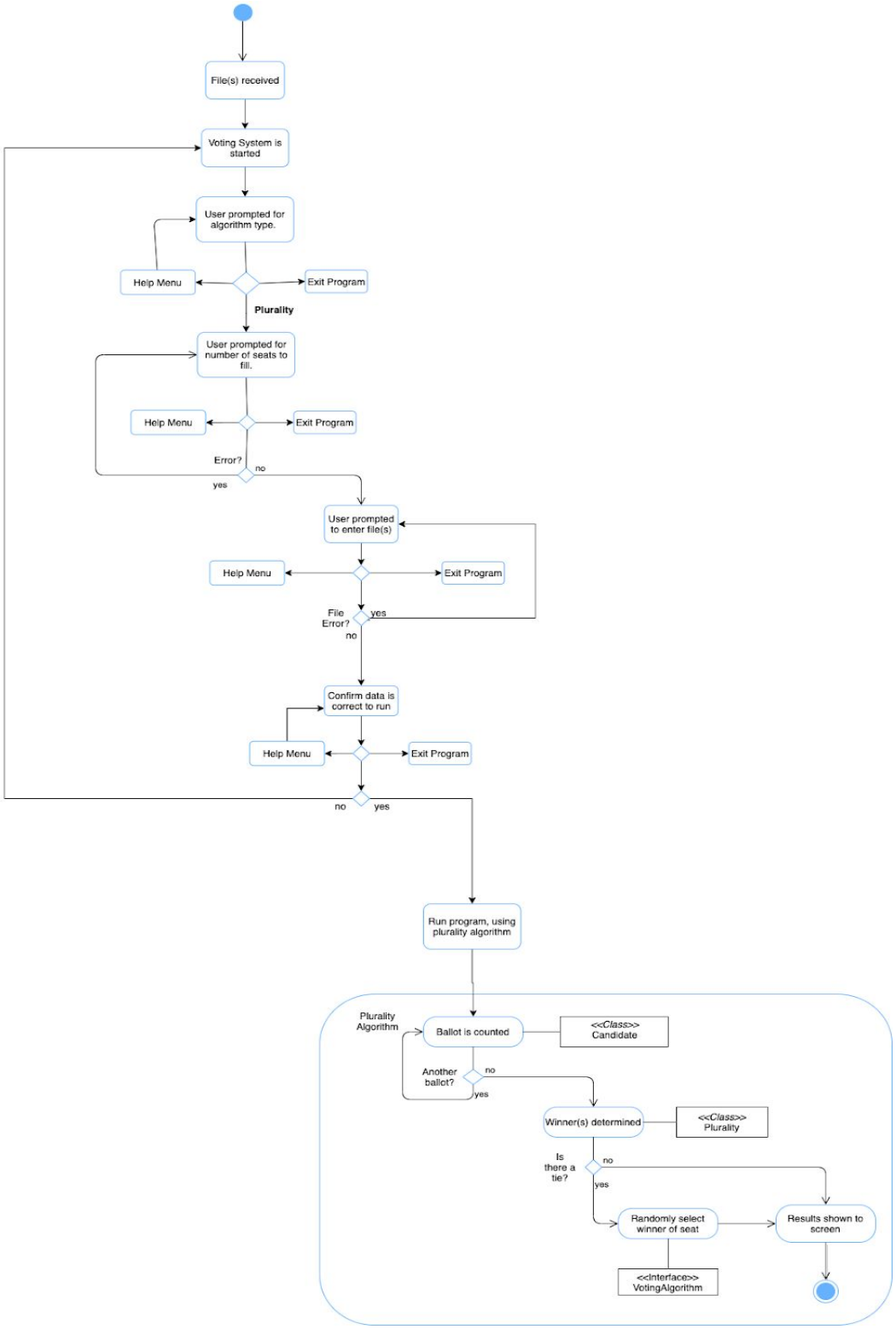
- Files will be parsed into the Voting System's major entities (Ballots and Candidates)

Attribute	**Class**	**Type**
voteType	Election	Private String
numSeats	Election	Private Int
winners	Election	Protected Queue<Candidate>
losers	Election	Protected Stack<Candidate>
candidates	Election	HashSet<Candidate>
shuffle = true	Election	Protected boolean
ballotList	Election	Private LinkedHashSet<Ballots>
files	Election	Private File[]
auditFile	Election	Protected static File
numBallots	Election	Protected Int
name	Candidate	Private String
voteCount	Candidate	Private Int
ballot	Candidate	Private LinkedHashSet<Ballot>
isWinner	Candidate	Private Boolean
voteMap	Ballot	Private Map<Candidate, Integer>
ballotNum	Ballot	Private Int
shuffle	STV	Private Boolean

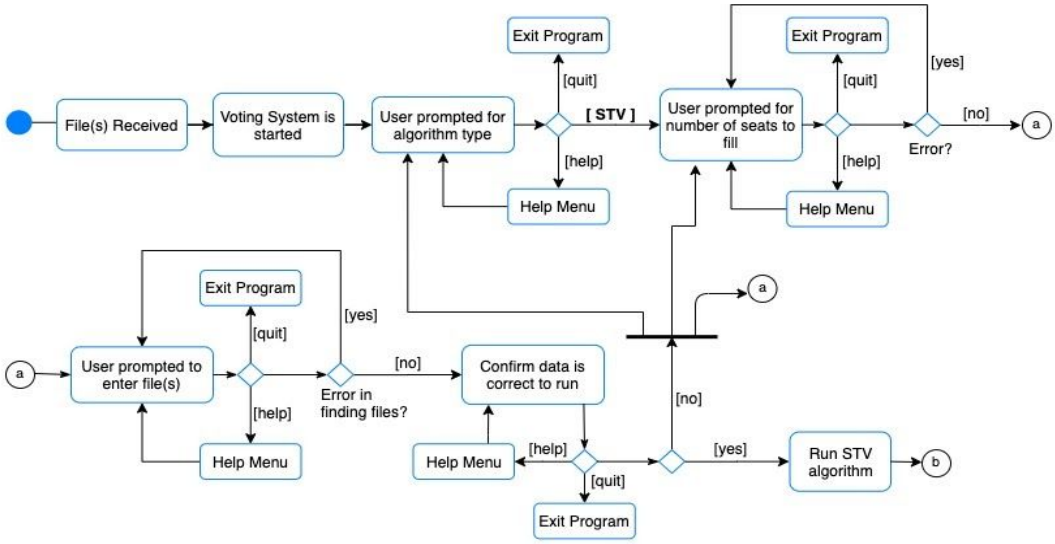
ballots	STV	Private List<Ballot>
finalVoteCounts	Plurality	Private List<Int>

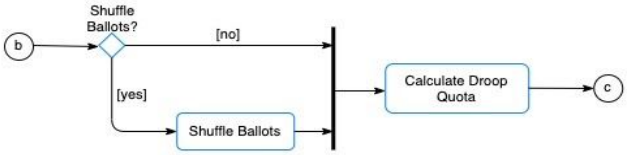
5. COMPONENT DESIGN

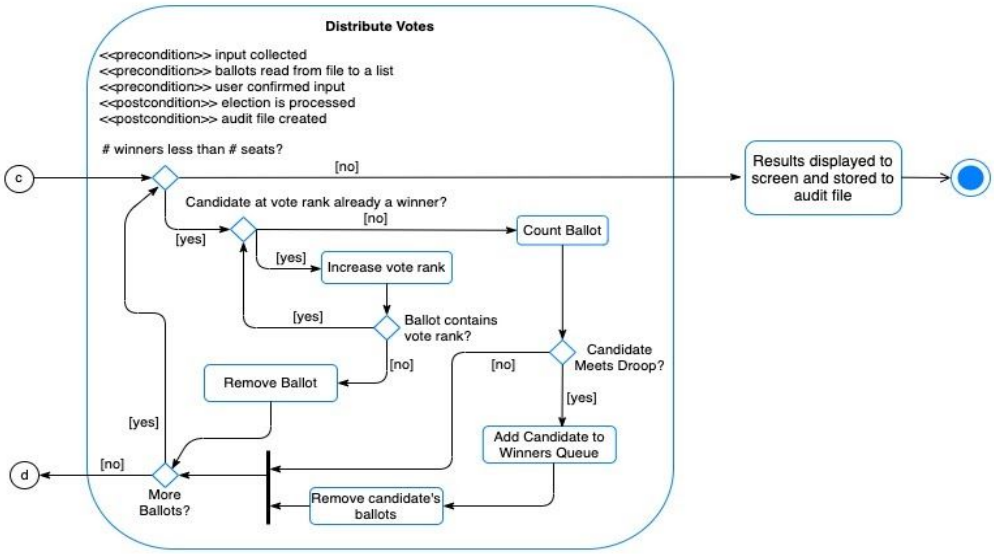
Plurality Activity Diagram

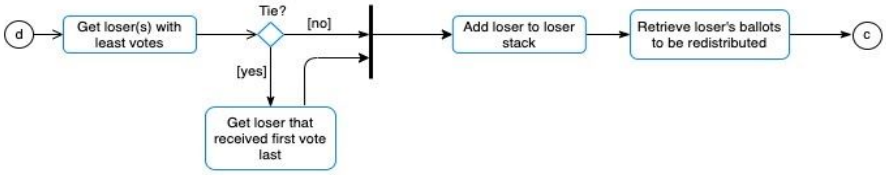


STV Activity Diagram









6. HUMAN INTERFACE DESIGN

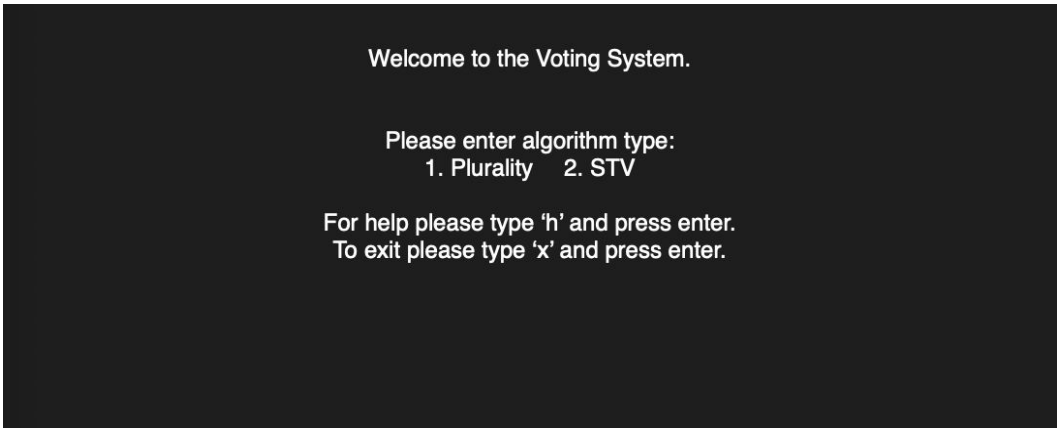
6.1 Overview of User Interface

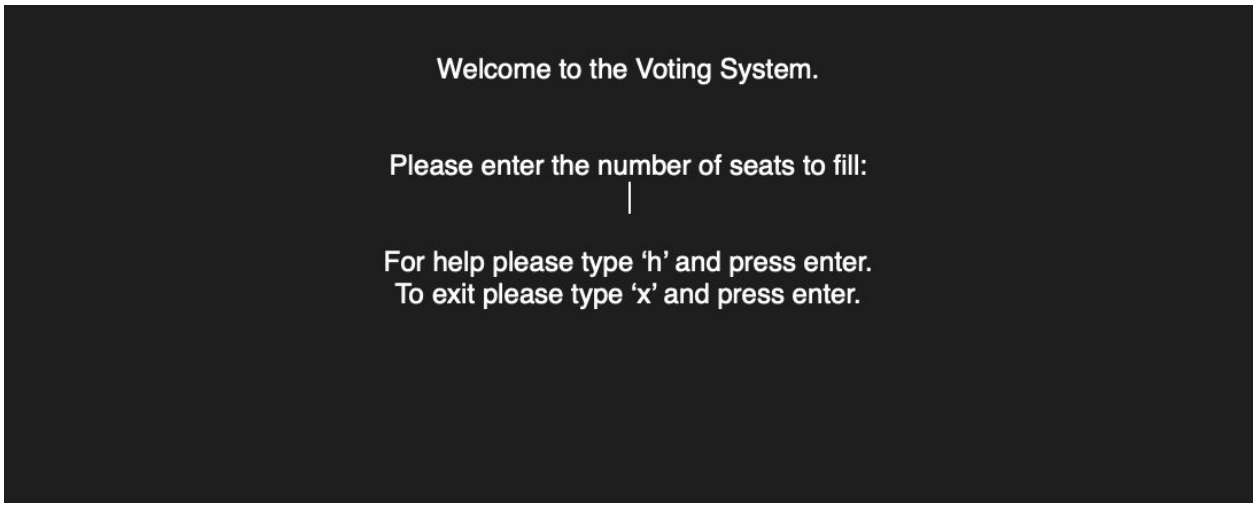
The user will be interacting with the Voting System's text-based interface. Upon startup of the program, the Voting System's text-based interface will open. There will be a sequential flow of data input that the system needs to be able to run. The first prompt will be entering the algorithm type. The two options are labeled by number, only the numbers labeling the algorithm type will be accepted. The user will then be prompted to enter the number of seats to fill. Any integer will be accepted, and it is assumed that it is entered correctly. The user presses enter to continue. The next prompt asks for the file name(s) the system should look for. This should be entered as a string where each file is separated by a singular space. The user presses enter to continue. After this information is entered, the user will see a confirmation screen asking them to confirm all information is correct. If the user deems all information correct and indicates so via the prompt, the system will tabulate the election. If the user notices a mistake and indicates so via the prompt, the user can start the sequence of entering in all information again.

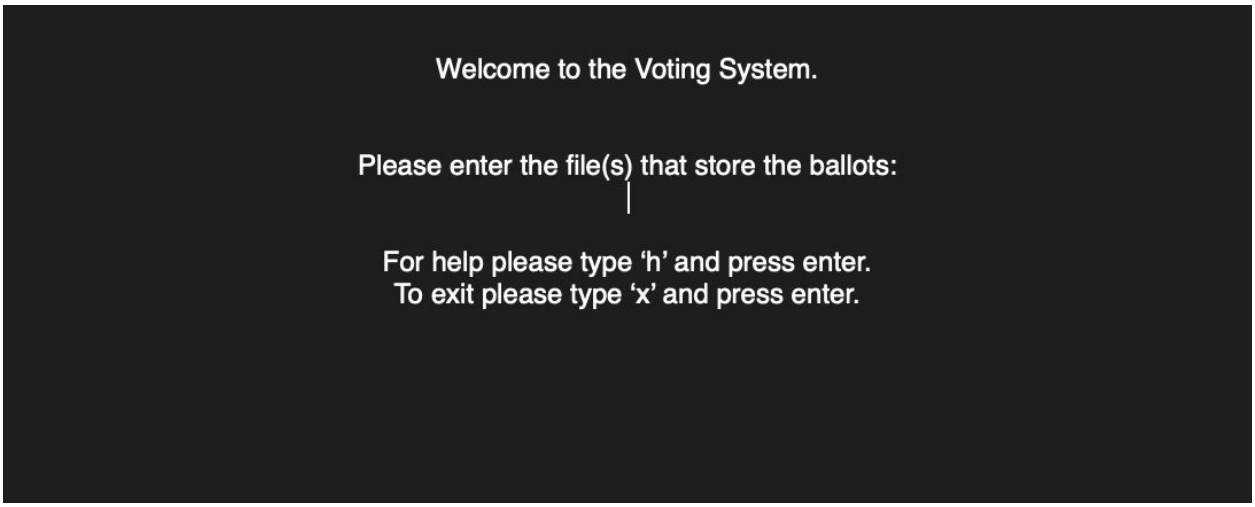
At any time before the user indicates all provided information is correct, the user can type 'h' and press enter to be brought to the Voting Systems help window which will explain the prompts. The user can enter 'x' and press enter to exit the help window. Also, at any time before the user indicates all provided information is correct, and is on a prompt screen, the user can type 'x' to terminate the program.

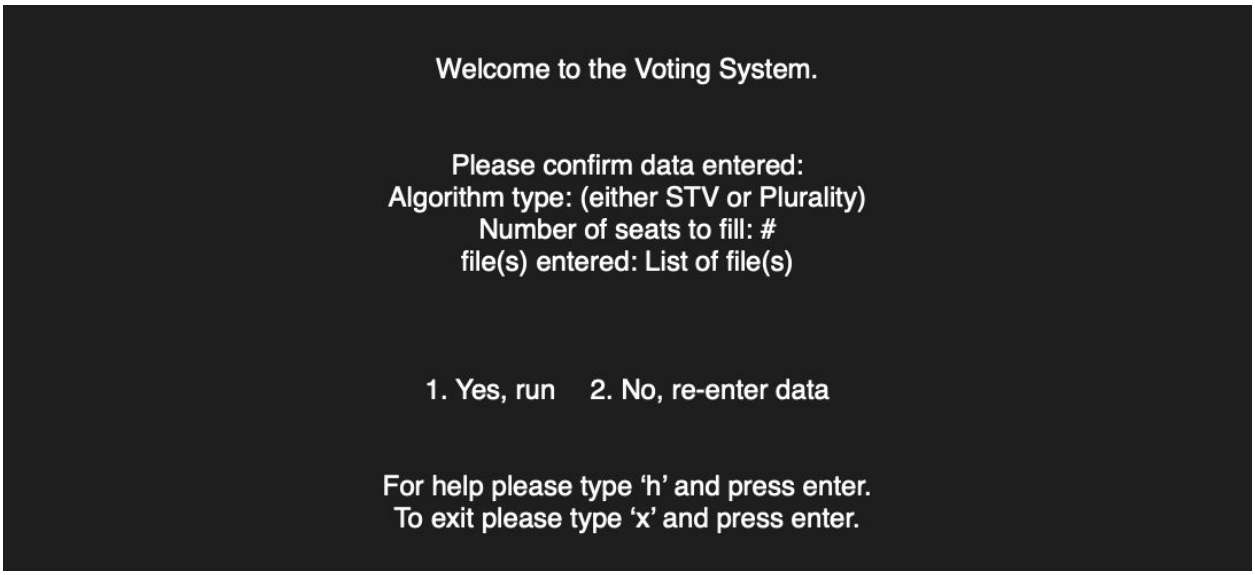
6.2 Screen Images

The next 6 screenshots depict: algorithm type, number of seats to fill, file names to be considered, confirming data entered, help menu, results(Plurality) and results(STV) respectively.




Welcome to the Voting System.
Please enter the number of seats to fill:
For help please type ‘h’ and press enter.
To exit please type ‘x’ and press enter.


Welcome to the Voting System.
Please enter the file(s) that store the ballots:
For help please type ‘h’ and press enter.
To exit please type ‘x’ and press enter.


Welcome to the Voting System.
Please confirm data entered:
Algorithm type: (either STV or Plurality)
Number of seats to fill: #
file(s) entered: List of file(s)
1. Yes, run 2. No, re-enter data
For help please type ‘h’ and press enter.
To exit please type ‘x’ and press enter.

Help Menu.
(Type 'x' and press enter to exit help menu)

Algorithm type: The only input accepted will be the number 1 or the number 2.
1 indicates the Plurality algorithm
2 indicates the STV algorithm
After the selection is inputted into the prompt, press enter to continue.

Entering the number of seats to fill: Only integers will be accepted. After the number is inputted into the prompt, press enter to continue.

Entering file name(s): Files should be inputted into the prompt as one string where each file name is separated by a space. Press enter when complete to move onto the next step.

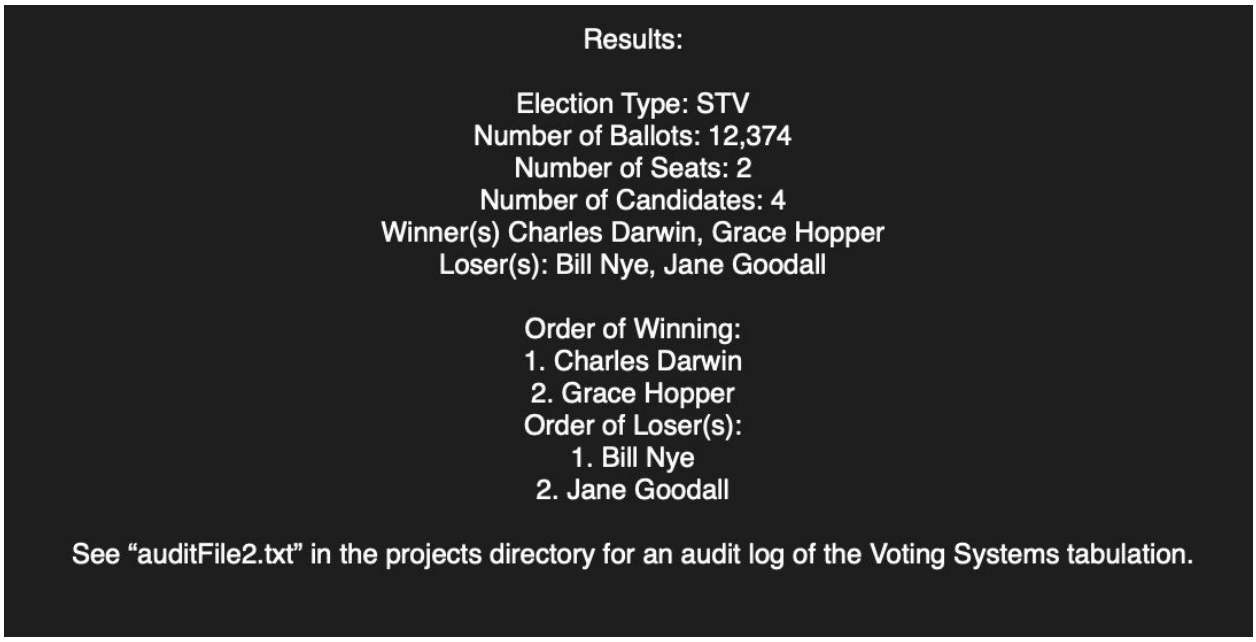
Confirming entered data: The only input accepted will be the number 1 or the number 2.
After the selection is inputted into the prompt, press enter to continue.
1 indicates that all the data is correct, and the next step will be to run the specified algorithm given the data inputted.
2 indicates that some or all of the data is incorrect. This selection will restart the initial data entry.

Results:

Election Type: Plurality
Number of Ballots: 8,489
Number of Seats: 2
Number of Candidates: 4
Winner(s): Charles Darwin, Grace Hopper
Loser(s): Bill Nye, Jane Goodall

Percentage of Votes:
Charles Darwin: 40%
Grace Hopper: 35%
Bill Nye: 15%
Jane Goodall: 10%

See "auditFile.txt" in the projects directory for an audit log of the Voting Systems tabulation.



6.3 Screen Objects and Actions

This Voting System will be using a text-based interface. The user will be using a keyboard to communicate with the system.

7. REQUIREMENTS MATRIX

SRS Functional Requirements	Component & Data Structure
System Feature 4.1 - Vote Selection	**UI** class, **promptElectionType()** method. Vote selection stored in a **private String voteType** field in the Election class.
System Feature 4.2 - Inputting number of seats to fill	**UI** class, **promptNumSeats()** method. The number of seats to fill is stored to **Election** class’s **protected int numSeats** field.
System Feature 4.3 - Inputting file name(s)	**UI** class, **promptFiles()** method. Files saved to **private File[] files** in the **Election** class

System Feature 4.4 - Shuffle Switch	The shuffle switch will be taken as an **argument in the main method** of the **Main** class (the entering point of the program). The **Election** class holds the shuffle switch in the **protected boolean shuffle** field. The value defaults to true, but if the argument is present, then it is set to false.
System Feature Shuffle Ballots	**STV** class, **shuffle()** method. The shuffled ballots are saved to the **private List<Ballots> ballots** field in **STV** class.
System Feature 4.5 - Run Program	**Election** class's **processElection()** method kick-starts the processing of the elections. The logic to actually process the ballots is handled within the **STV** or **Plurality** classes respectively.
System Feature 4.6 - Help Window	**UI** class, **displayHelp()** method.
System Feature 4.7 - Confirm Options	**UI** class, **confirmInput()** method.
System Feature 4.8 - Quit Program	**UI** class, **quitProgram()** method.
System Feature 4.9 - Start System	Program execution from **Command Line Interface** (CLI) or from **Integrated Development Environment** (e.g. Eclipse).
System Feature - Display election results to screen	**UI** class, **displayResults()** method. Method will make use of **Election** class's variables: - **Queue<Candidate> winners** - **Stack<Candidate> losers** - **int numBallots**
System Feature - Generate audit file	**Files auditFile** object. The object's **write()** method will be used to write to the file during any processing event of the election.