Team 2

Straightforward Voting System

Software Design Document

Version 1.0

Prepared by Amir Jalili (jalil014), Arun Sharma (sharm485), Nicholas Lovdahl (lovda015), and Taylor O'Neill (oneil569)

March 20, 2020 (03/20/20)

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Amir Jalili, Arun Sharma, Nicholas Lovdahl, and Taylor O'Neill	3/20/20	Completion of the initial version of the Software Design Document.	1.0

1. Introduction

1.1. Purpose

The purpose of this document is to describe clearly and entirely the architecture and design behind the Straightforward Voting System (SVS). The Straightforward Voting System is meant to tally and determine the results of elections from provided ballots given as CSV files. SVS can operate both plurality and single transferable voting (using the Droop Quota) based elections. This system should assist election officials in determining the winners and losers of such elections in a manner that is quick, fair, and accountable.

The expected audience for this document is the developers of SVS. This document should provide all of the information required to implement SVS correctly (as described in both this document and the Software Requirements Specification document) and to maintain it going forward.

1.2. Scope

SVS is a software that allows election officials to conduct elections based on either plurality voting or a single transferable voting system using the Droop quota. During elections, ballots are collected indicating a voter's selection of candidate(s). SVS will then be given these ballots, the number of seats available, and the type of election, and then determine the winners and losers. SVS will also produce an audit file in addition to these results. This audit file will provide a record of SVS's action in determining the results of the election. SVS will ease the burden on election officials by automating the tallying of ballots. It will also aid in removing human error from the election process, increase the speed at which election results arrive, and provide for a reliable and transparent election process.

1.3. Overview

This section gives an overview of each section of this document. These sections are as follows:

- Section 1 is the Introduction. If you are reading this, you are currently in this section. This section provides context for SVS and this document itself.
- Section 2 is the System Overview. This section provides a general description of SVS as a system - the context in which it operates, its functionality, and a broad description of its design to accomplish this.
- Section 3 is the System Architecture. This section provides a description of SVS's design in greater depth than in the System Overview.
- Section 4 is the Data Design. This section describes how data is organized by SVS and also contains a list of the structures in SVS.
- Section 5 is the Component Design. This section lists each component in SVS in depth, providing a list of every object and each object's methods and attributes.
- Section 6 is the Human Interface Design. This section explains how SVS is used from the perspective of the end user. This section also includes images of what SVS's user interface could look like.
- Section 7 is the Requirements Matrix. This system relates the requirements for SVS as described in the Software Requirements Specification document to the components in SVS's design.

- Section 8 is the Appendices. This section contains additional information which, while not directly related to the design of SVS, could be useful in its implementation or maintenance.

To locate a specific section or subsection, the Table of Contents should be consulted. The Table of Contents begins on page 1.

1.4. Reference Material

This section lists resources referenced by this document. The titles of these resources are given first, along with details to be used to locate them:

- IEEE Std 1016-1998: This document was published for use as a template for Software Design Documents (which has been adapted for this document). It was developed by the Institute of Electrical and Electronics Engineers in 1998.
- Single transferable vote: An article in Encyclopaedia Britannica. This article was created by the editors of Encyclopaedia Britannica, published by Encyclopaedia Britannica, Inc. on April 21, 2017, and accessed on February 21, 2020. See the following url: https://www.britannica.com/topic/single-transferable-vote.

1.5. Definitions and Acronyms

This section contains a list of terms used in the document which may require further explanation:

- SDD: The acronym for “Software Design Document”, a document which contains the information describing the design and rationale behind a software product. In the context of this project, that refers to this document.
- SRS: The acronym for “Software Requirements and Specifications”, a document which lays out the requirements and specifications that a software product should conform to.
- SVS: The acronym for “Straightforward Voting System”, the name of the product being developed and the subject of this document.
- CSV files: CSV is an acronym for “Comma Separated Values”. CSV files are a file type describing a text file which contains a number of values separated by commas. Such files are used to store and organize data, such as values in a spreadsheet, for example.
- Plurality election: A plurality election is a simple voting system where each voter chooses a single candidate on their ballot and the candidates with the most votes win. Should a tie occur in this system, it would be broken by randomly selecting a tied candidate as a winner.
- Single transferable vote election: A single transferable vote election is a voting system in which voters rank candidates based on their preferences. These ballots can then be used to determine the winners and losers based on the collective preferences of the voters. For more information on this system, consult the Encyclopaedia Britannica article on “Single transferable vote” listed in the Reference Material section.
- Droop Quota: The Droop Quota is a quota used in some single transferable vote elections to determine the threshold of votes at which point a candidate can be declared a winner. Any votes after a candidate has been declared a winner are then provided to the voter’s next ranked choice (if they have one).

2. System Overview

The purpose of the Straightforward Voting System is to tally and determine the results of elections from provided ballots given as CSV files. The elections can either be implemented using a Droop Quota algorithm or a plurality algorithm. If an election uses the plurality algorithm, then each voter is only allowed to vote for one candidate. The candidate with the most votes is therefore declared the winner. If the election uses the Droop Quota algorithm, on the other hand, then each voter will fill out a ballot with at least half of the candidates ranked. This method is often referred to as “rank-choice voting”. The winner(s) are determined by the Droop Quota algorithm, which is explained later in this section. The general flow of the system will follow the steps below.

First, the system is designed to specifically handle ballot results that are already stored in CSV files. This means that the system assumes that the polling has already taken place, and that all of the polls were consolidated into CSV files with no errors.

Next, the system will ask the user who is to process the election to input how many seats must be filled, to import the CSV files that hold all of the ballot information, and to select the algorithm the system should run when analyzing the ballots. The system will parse these files and store the information within the system for the respective algorithm to use.

If the system is to run a plurality election, the algorithm will follow these steps to determine the winner(s):

1. A count will be maintained for each candidate. This count is initialized to be zero and will indicate the number of votes the candidate has received.
2. All of the ballots are read through once. A candidate’s count is incremented if they are selected in a ballot.
3. After all of the votes have been counted, the candidate with the largest vote count will be declared a winner. This step repeats while for each available seat, picking the candidate with the next most votes. If there is a tie in a vote count for a given seat, the winner will be chosen at random.

If the system is to run a Droop Quota election, the algorithm will follow these steps to determine the winner(s):

1. The ballots will be shuffled first.
2. The formula shown below is used to calculate the Droop quota:

$$Droop\ Quota\ = \lfloor \frac{Number\ of\ ballots}{Number\ of\ seats + 1} \rfloor + 1$$

3. The shuffled ballots will be selected one-at-a-time. Each selected ballot will be placed in a pile that corresponds to the candidate that was ranked as the top choice on the ballot.
4. If a candidate's pile reaches the Droop Quota, they will automatically be declared elected. The ballots in that candidates pile will be removed from the process. The name of the candidate will be placed in the first location of the “Elected List”. As the process continues, any selected ballot that has a candidate's name that is already on the “Elected List” as the top choice will instead be placed in the pile of the next ranked candidate that is not on the “Elected List”.
5. After the first round of dispersing the ballots that were in the first candidates pile, the candidate with the smallest pile will be removed from the election and placed on the “Non-Elected List”. The ballots in the removed candidates pile will be redistributed. In the case of a tie, the candidate that received a ballot in their pile last will be eliminated.

6. This process will repeat until all of the ballots have been processed. After the algorithm completes, the “Elected List” and “Non-Elected List” will be displayed to the user.
7. An audit report will be created for the user and stored in a text file. The audit report will show how the ballots were assigned to different candidates as the process continued. The top of the report will state the type of election, the number of seats to be filled, the number of candidates, the winners, and the losers.

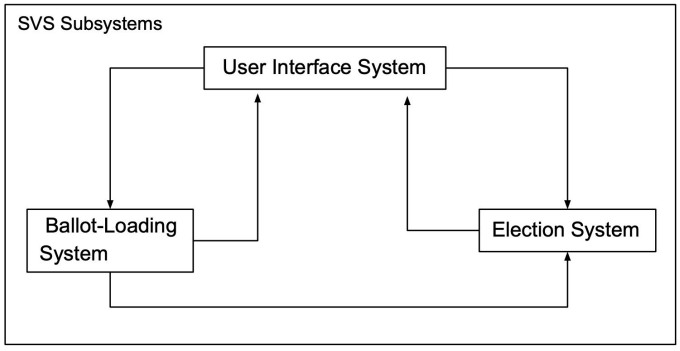
After the system finishes processing the results of the election, the screen will display all of the important details about the election. The details include the election type, number of ballots, number of candidates, and number of seats that were filled. The screen will also display the winners and losers of the election. If the election used a plurality algorithm, the percentage of votes will be shown next to each candidate, respectively. If the Droop Quota algorithm was used, the order of winners and losers shall be displayed.

3. System Architecture

3.1. Architectural Design

SVS is divided into three subsystems: the User Interface System, the Ballot-Loading System, and the Election System. The User Interface System is responsible for interaction of many functionalities for STV systems such as loading input files (i.e. ballots) and running type of election system (i.e. STV or plurality) for the voting officials. Other functionalities include input parameters, progress bar display, etc. The Ballot-Loading System is responsible for loading files from the system and stores them in a data structure which will be used to run Elections. The Election System is responsible to run the type of election (e.g. STV or plurality) and saves the audit file for the user.

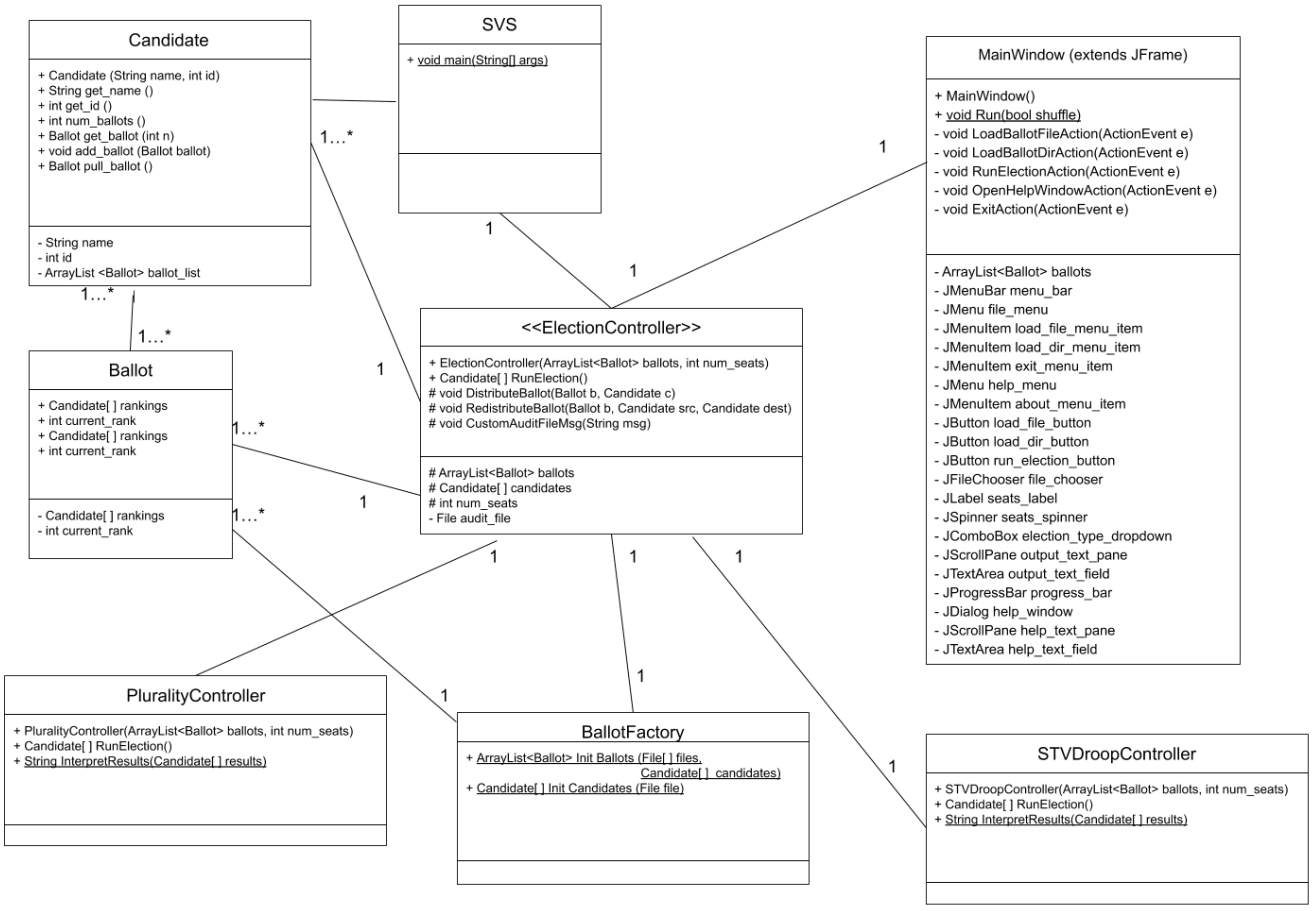
The User Interface is connected to the Ballot-Loading System and Election System providing the user some choice of uploading ballot files (e.g from files or directories) and running the type of election or providing a number of seats after uploading the ballot files. The Ballot Loading System is connected to the election system providing an entire list of candidates in ArrayList<Ballot> ballots which will be responsible for providing input to the Election System. Similarly, The Election system is connected to the user interface and Ballot Loading System.



3.2. Decomposition Description

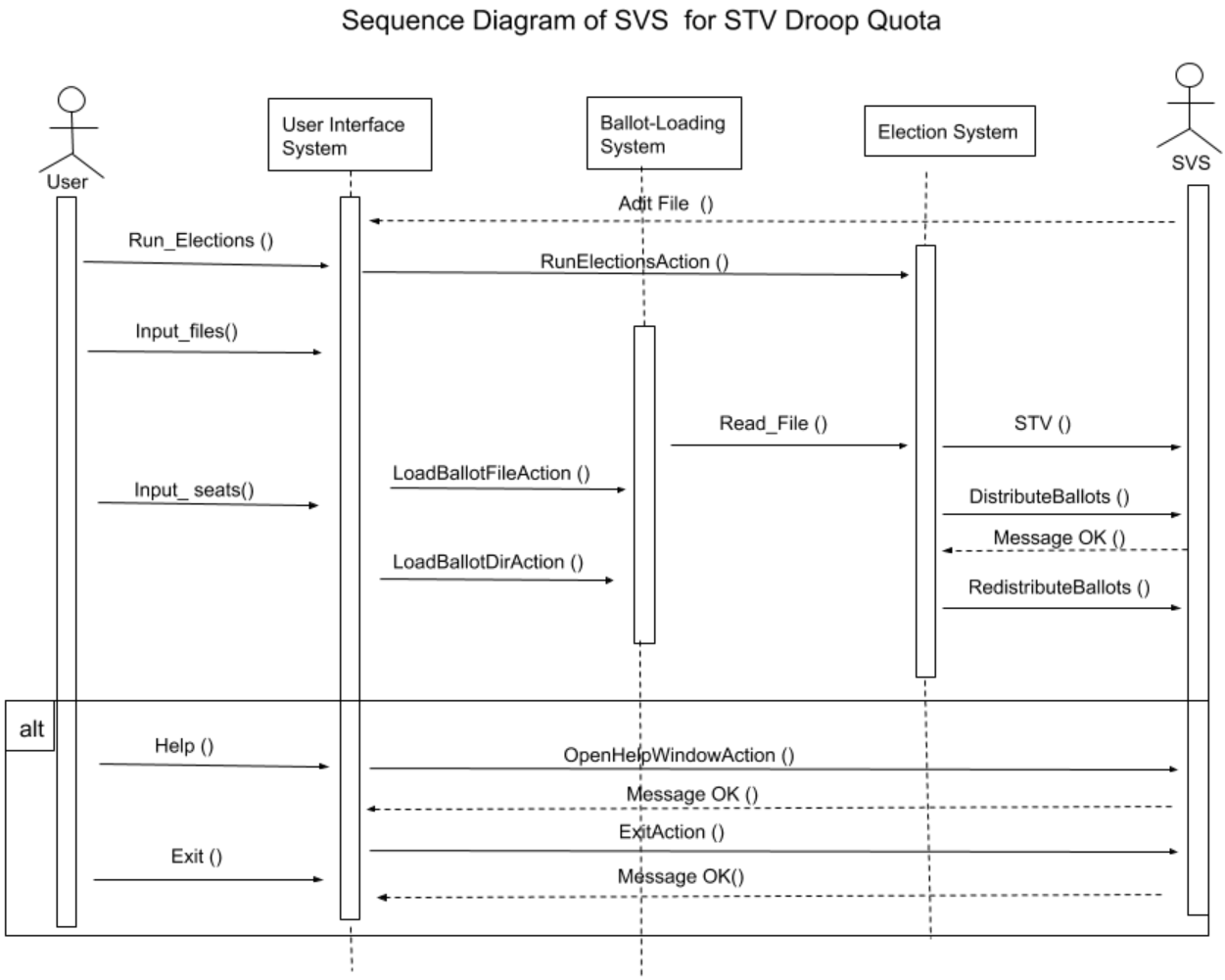
Subsystem Models:
User Interface: The interface provides visual aid to the users and is responsible for input actions such as input files, parameters (e.g. number of seats), status report (e.g. showing progress bars etc.) .
Ballot-Loading System: Takes the overall ballots from the candidates and stores inside the ArrayList of the class name ballots. The data repository consists of Ballots in CSV format.
Election System: Performs election type of the given choice from the voting officials and produces audit files as the output in real time. The subsystem consists of Plurality and STV with a droop quota algorithm which runs in the background to produce the desired audit files and data repository as the ArrayList <Ballots>.

Class Diagram:



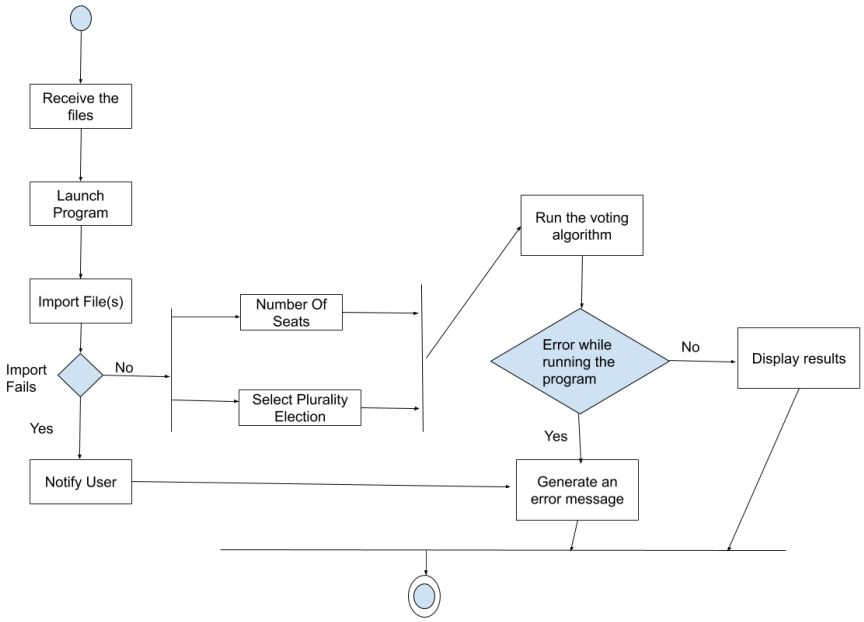
Interface Specifications: The interface specification includes certain actions that users need to perform to access a particular subsystem. For instance, the user needs to provide certain actions involving user input files and parameters (e.g. ballots, number of seats) and other actions such as running elections, exit application etc.

Sequence Diagrams:

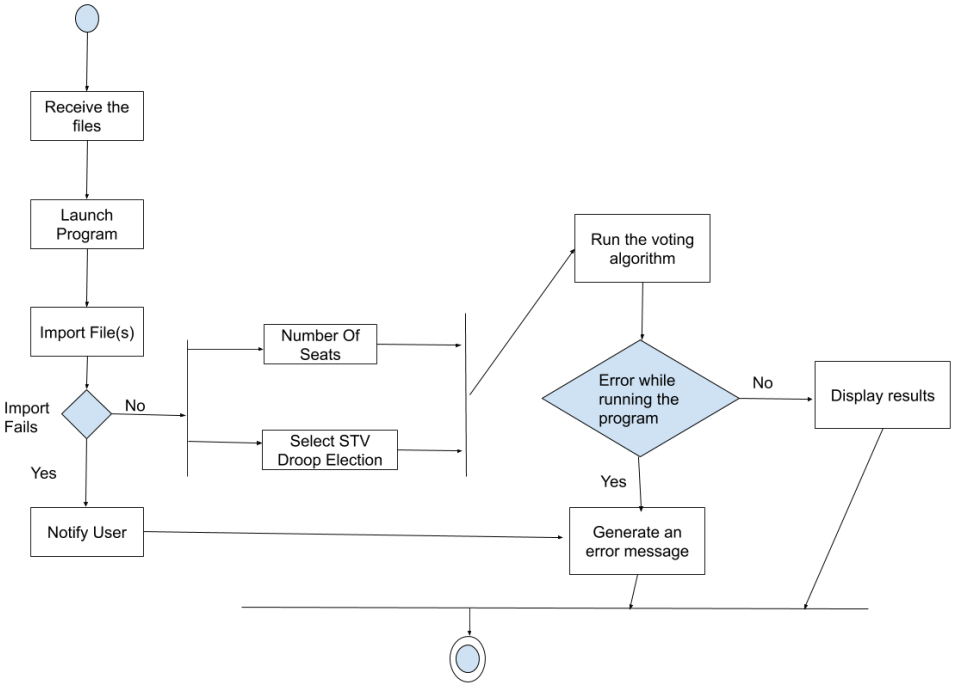


Activity Diagrams:

Activity Diagram for Plurality



Activity Diagram for STV



3.3. Design Rationale

Based on the description of SVS, the team decided that there must be a subsystem for the user interface, ballot-loading, and election processing. A dedicated subsystem for the user interface was critical to the architecture because all other subsystems need to interface with the user interface in order for the system to function properly and run to completion. A battle-loading subsystem was deemed another critical component. This is due to the amount of processing that needs to be done on each CSV file that is entered. The processing involves parsing through each file in the CSV file, and populating the relevant system objects. The election system is the last critical component in SVS. This subsystem is crucial as it processes the election results based on what algorithm the user input into the system. The team also determined that interfacing with the user interface to show election results and generating audit files were critical issues in SVS. It was concluded that the audit file could be generated in parallel to the process of the election algorithm. Therefore, the election subsystem could include this issue. The election subsystem can also easily interface with the user interface to show election results. The team decided on these three subsystems early in the planning process. No other architectures were considered.

4. Data Design

4.1. Data Description

SVS determines the results of elections by manipulating two kinds of classes, Candidate and Ballot. Together, these classes contain the information necessary to determine the result of an election.

SVS first loads all of the ballots from files selected by the user into an ArrayList. Because the ballots are organized in an ArrayList, they can be randomly accessed while maintaining the ability to add an arbitrary number of ballots to the list as they are loaded.

Each ballot is organized as an object of the Ballot class. Ballots each contain an array of references to Candidates, representing the voter's choices and rankings, and an integer corresponding to a particular rank (that is, the rank of the candidate in the ballot who receives that ballot). The array of Candidates is always ordered from the first choice as the first element to the last choice as the last element. While running an election, SVS uses this information to keep track of which candidates receive Ballots, and also to redistribute a Ballot to the next ranked Candidate when needed - this is done by incrementing the integer for the rank of a Ballot - since the choice for that ballot can then be determined as the Candidate in the respective position in the array of Candidates.

Each candidate is organized as an object of the Candidate class. Candidates each have a String for the name of the candidate, an integer for the candidate's id (this corresponds to the number of the column that the candidate is in in the CSV ballot files), and an ArrayList of Ballots which keeps track of the Ballots which are currently distributed to that Candidate. Because the Ballots are organized in an ArrayList, they can be randomly accessed while allowing for a Candidate to have an arbitrary number of Ballots distributed to them.

While running an election, SVS keeps each Candidate in an array. As SVS determines the result of an election, it organizes each Candidate into a second array which orders the Candidates as is appropriate to that election. For a plurality election this would mean that the Candidate with the most ballots would be first and the Candidate with the fewest ballots would be last in the array, and for a single transferable vote election with the Droop quota this would mean that candidates would be ordered based on the order in which they reached Droop (or lost). Once SVS is done, this second array is returned as a result. Thus, this array, between the ordering of Candidates and the information tracked by each Candidate object, acts as a kind of 'key' for the results of an election to be interpreted from.

4.2. Data Dictionary

Class Name	Ballot
Member Methods	+ Candidate[ ] rankings + int current_rank + Candidate[ ] rankings + int current_rank
Member Attributes	- Candidate[ ] rankings - int current_rank

Class Name	BallotFactory
Member Methods	+ ArrayList<Ballot> Init Ballots (File[ ] files, Candidate[ ] candidates, bool shuffle) + Candidate[ ] Init Candidates (File file)
Member Attributes	

Class Name	Candidate
Member Methods	+ Candidate(String name, int id) + String get_name() + int get_id() + int num_ballots() + Ballot get_ballot(int n) + void add_ballot(Ballot ballot) + Ballot pull_ballot()
Member Attributes	- String name - int id - ArrayList<Ballot> ballot_list

Class Name	<<ElectionController>>
Member Methods	+ ElectionController(ArrayList<Ballot> ballots, int num_seats) + Candidate[ ] RunElection() # void DistributeBallot(Ballot b, Candidate c) # void RedistributeBallot(Ballot b, Candidate src, Candidate dest) # void CustomAuditFileMsg(String msg)
Member Attributes	# ArrayList<Ballot> ballots # Candidate[ ] candidates # int num_seats - File audit_file

Class Name	MainWindow
Member Methods	+ MainWindow() + void Run(bool shuffle) - void LoadBallotFileAction(ActionEvent e) - void LoadBallotDirAction(ActionEvent e) - void RunElectionAction(ActionEvent e) - void OpenHelpWindowAction(ActionEvent e) - void ExitAction(ActionEvent e)
Member Attributes	- ArrayList<Ballot> ballots - JMenuBar menu_bar - JMenu file_menu - JMenuItem load_file_menu_item - JMenuItem load_dir_menu_item - JMenuItem exit_menu_item - JMenu help_menu - JMenuItem about_menu_item - JButton load_file_button - JButton load_dir_button - JButton run_election_button - JFileChooser file_chooser - JLabel seats_label - JSpinner seats_spinner - JComboBox election_type_dropdown

	- JScrollPane output_text_pane - JTextArea output_text_field - JProgressBar progress_bar - JDialog help_window - JScrollPane help_text_pane - JTextArea help_text_field

Class Name	PluralityController
Member Methods	+ PluralityController(ArrayList<Ballot> ballots, int num_seats) + Candidate[ ] RunElection() + String InterpretResults(Candidate[ ] results)
Member Attributes	

Class Name	STVDroopController
Member Methods	+ STVDroopController(ArrayList<Ballot> ballots, int num_seats) + Candidate[ ] RunElection() + String InterpretResults(Candidate[ ] results)
Member Attributes	

Class Name	SVS
Member Methods	+ void main(String[] args)
Member Attributes	

5. Component Design

User Interface System:

MainWindow() : This method is called when an application gets started providing visual aid to the user.

RunElection() : This method will call elections and by calling a subsystem Election System.

LoadBallotFileAction(ActionEvent e) : This method will load Ballot Files from the File System of the user and store them in the ArrayList of objects as Ballots by calling ReadFile in the Ballot Loading System .

LoadBallotDirAction(ActionEvent e): This method will load Ballot Files from the Directory of the user.

RunElectionAction (): This method will run the actual elections by calling

OpenHelpWindowAction(ActionEvent e): This method will open the Help Window from the Main Screen.

ExitAction(ActionEvent e): This method is responsible for terminating the application.

Election System:

RunElection (): This method will run elections based on the action from the User Interface System.

Shuffle(): This method will take the boolean input to decide whether shuffling should take the place of given input ballots from the Ballot Loading System based on the user's preference.

Plurality () : This method will run the Plurality algorithm from given sets of inputs and parameters from

LoadBallotFileAction() and LoadBallotDirAction() and number of seat text fields User Interface System.

STV () : This method will run the STV algorithm with a droop quota from given sets of inputs and parameters from LoadBallotFileAction() and LoadBallotDirAction() and number of seat text fields User Interface System.

DistributeBallots (): This method will run for the first iteration where the ballots get redistributed one at a time and populate an elected and non-elected list based on the droop quota calculated in STV() method.

RedistributeBallots (): This method will run multiple times till all the ballots are processed and the elected and non-elected list is finalized.

AuditFile (): This function will create the text audit file in the form of a report consisting of necessary information (e.g. type of election, winners, losers etc.).

Ballot-Loading System:

ReadFile(): This method will create an ArrayList of type ballots from the User Interface System which is later called by methods in the Election System.

6. Human Interface Design

6.1. Overview of User Interface

The User Interface for SVS will be presented graphically. The ‘main hub’ of SVS for the user is the main window. The main window contains the primary interface controls that the user will use to load ballots from either files or a directory, input the number of seats and the type of election, and then begin the process of determining the results of the election.

Ballots can be loaded into SVS by either clicking on one of the two buttons to load ballots (one button to load ballots from files and another to load ballots from a directory) or by selecting the equivalent options in the “File” menu. This will prompt the users to select the files or directory with files to load through a file chooser; the file chooser will provide the user with an interface to help them navigate through their system’s file system to find and select the ballot files. Once the ballot files have been selected, SVS will clear the “Output” text field and will print the result from loading the files.

The “Seats for Election” spinner (a field where the user can enter a number which may also be increased or decreased with the use of two arrow buttons) allows the user to choose the number of seats for an election.

A dropdown menu is available to select the type of election. Each of the two elections that SVS can run, plurality and single transferable vote with the Droop quota, are options.

Together, these controls allow the user to input all of the information SVS needs to conduct an election. Then, the user will click on the “Run Election” button to begin the process of determining the result of the election. While the election is running, the progress bar at the bottom of the screen shows that SVS is busy. When the election is done, the progress bar returns to normal (an empty bar) and the results of the election are shown in the “Output” text field.

For a plurality election this output will consist of a listing of each candidate. The name of the candidate and the number and percentage of ballots they received are written to the output screen. Each candidate is placed on a line of the “Output” text field and they are placed in order from the candidate with the most votes on the top line to the candidate with the fewest votes on the last line.

For a single transferable vote election with the Droop quota this output will also consist of a listing of each candidate. However, candidates will be listed in order of the first to reach the Droop quota on the first line, to the last winner, to the losers where the losers are ordered based on the number of ballots they ended up with.

If the user needs help using SVS, a help window is also available. The help window contains a text field with instructions for using SVS. The help window can be accessed through the “Help” menu by choosing the “About” option.

Also note that some of the use cases for SVS, the creation of the audit file and the disabling of ballot shuffling, are not explicitly exposed to the user through the user interface. Nevertheless, they are a part of the processes which the user can initiate through the user interface - the audit file for an election is generated automatically as a result of running an election and disabling ballot shuffling is respected when loading ballots.

6.2. Screen Images

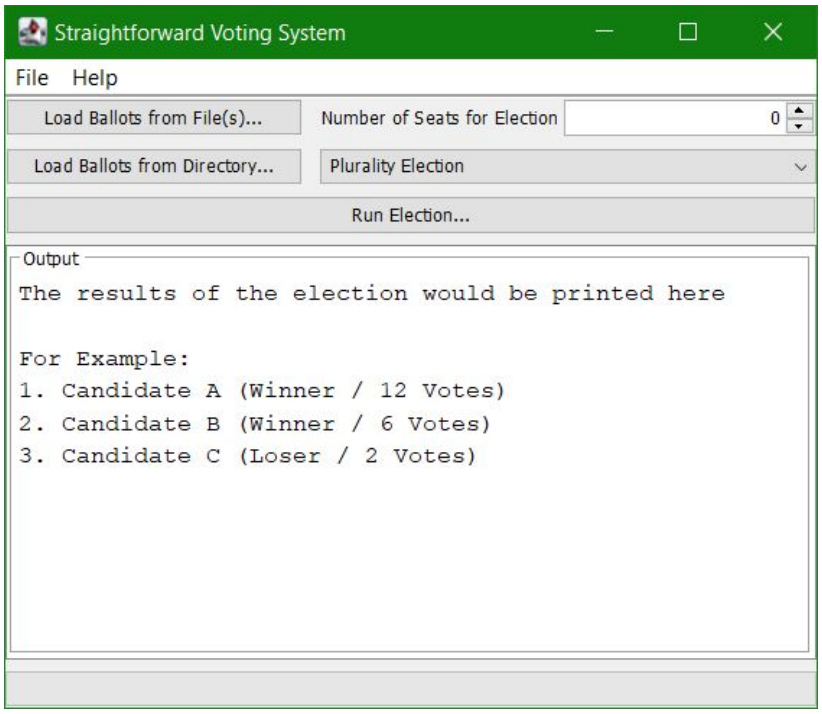


Figure 6.1: The main window for SVS is shown. The aftermath of an election is shown with the results printed in the “Output” text field.

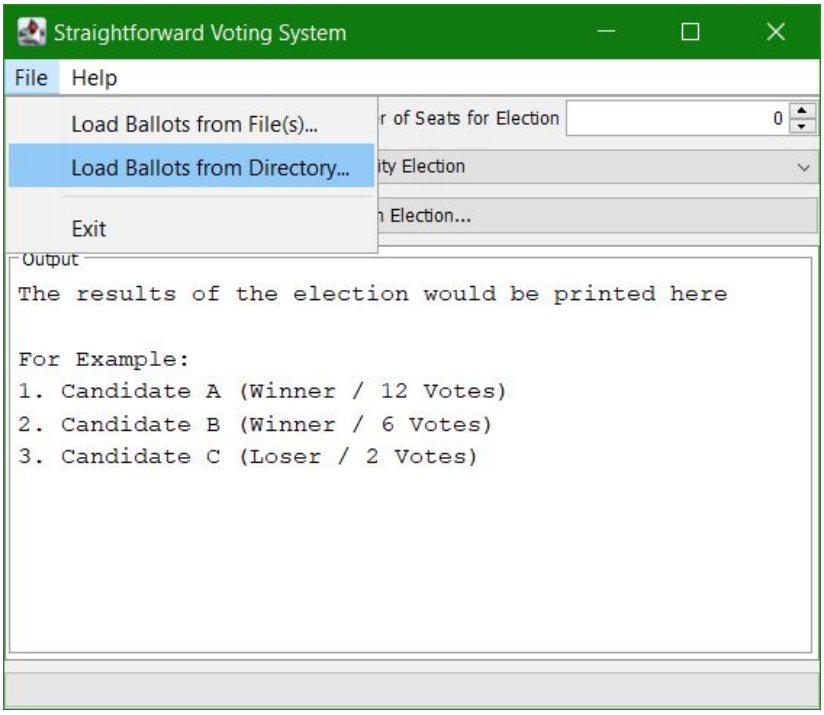


Figure 6.2: The file menu is shown here. There are options to load ballots from files, a directory, and to exit the program.

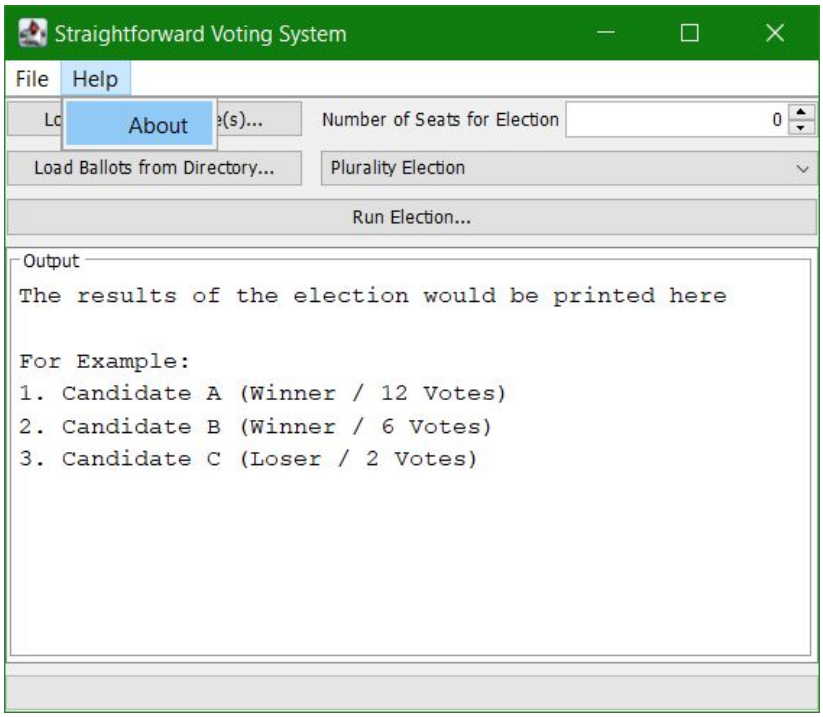


Figure 6.3: The help menu is shown here. The “About” option would open up the help window

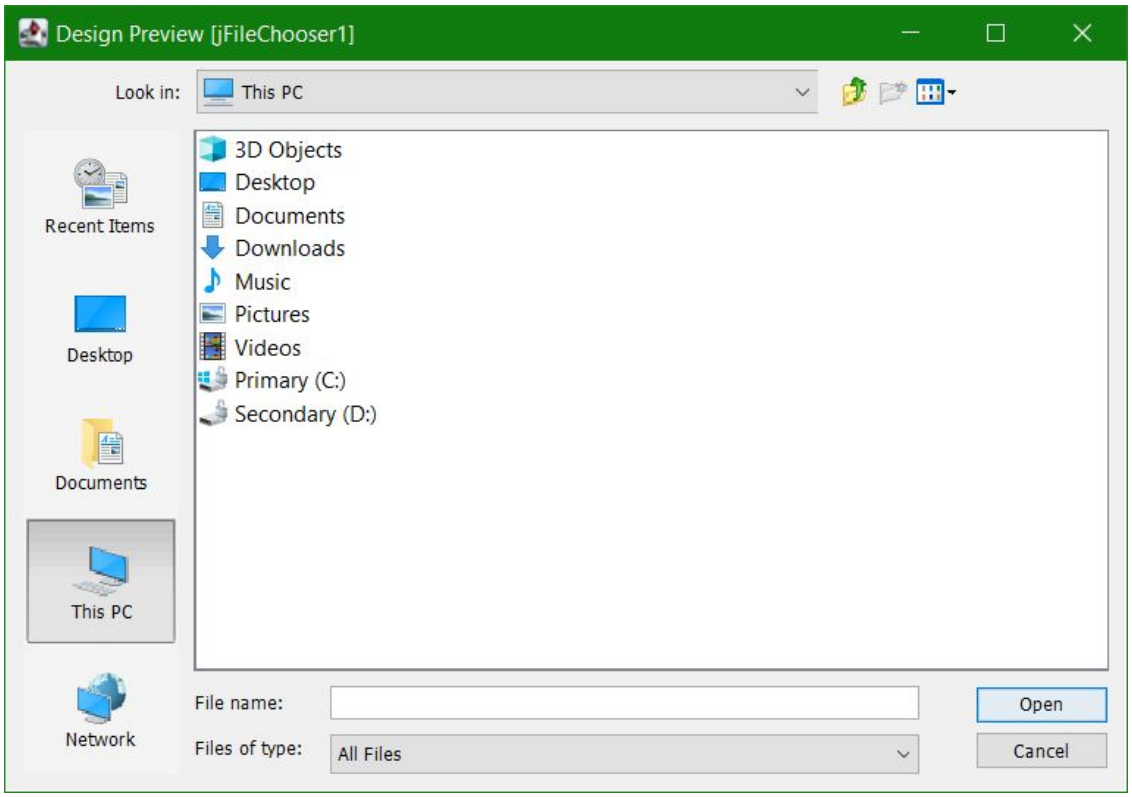


Figure 6.4: The file chooser is shown. This lets the user navigate to select the file(s) or directory to load ballot files from.

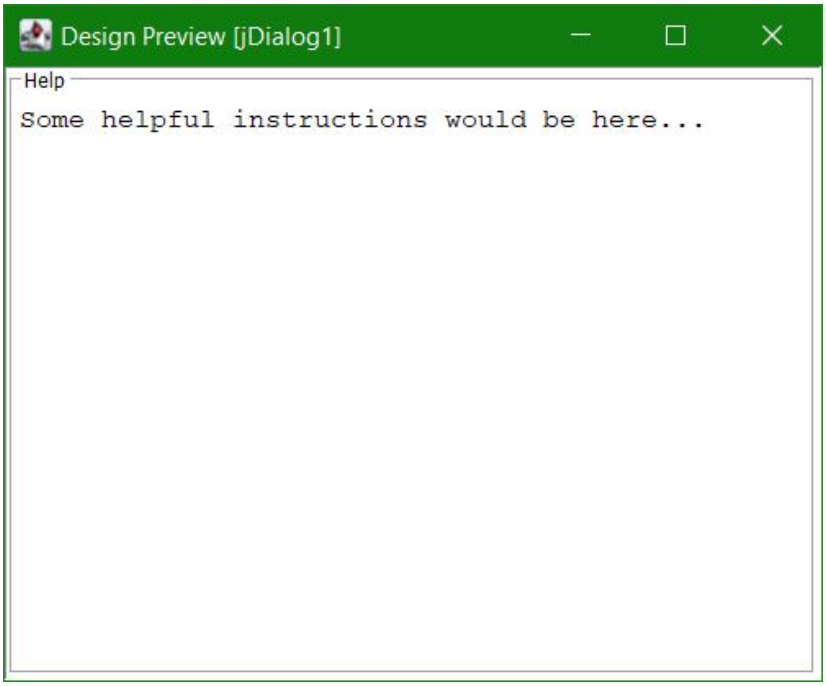


Figure 6.5: The help window is shown. This window would contain text concerning how to use SVS in the “Help” text field.

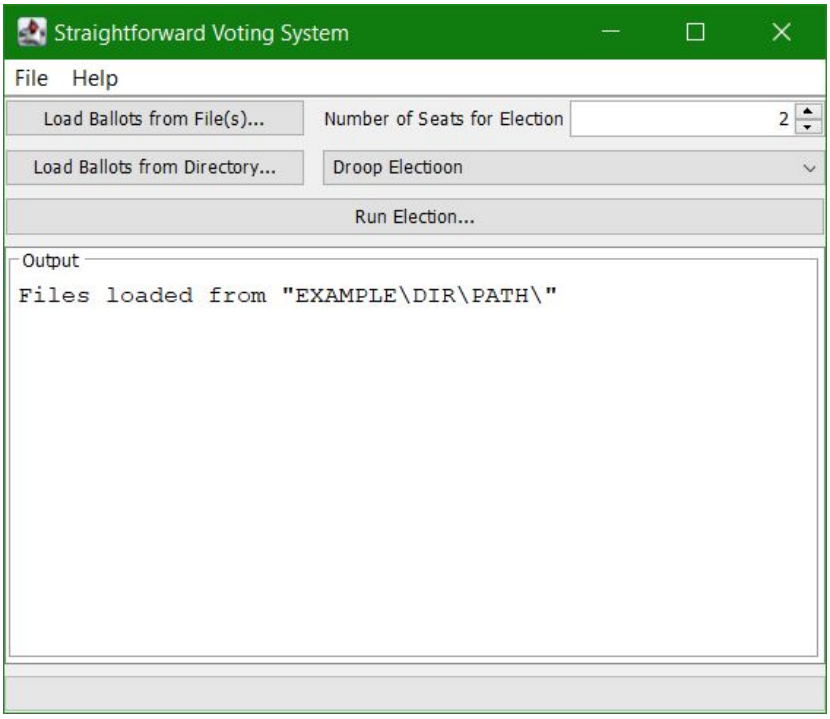


Figure 6.6: The main window is shown after loading some ballot files.

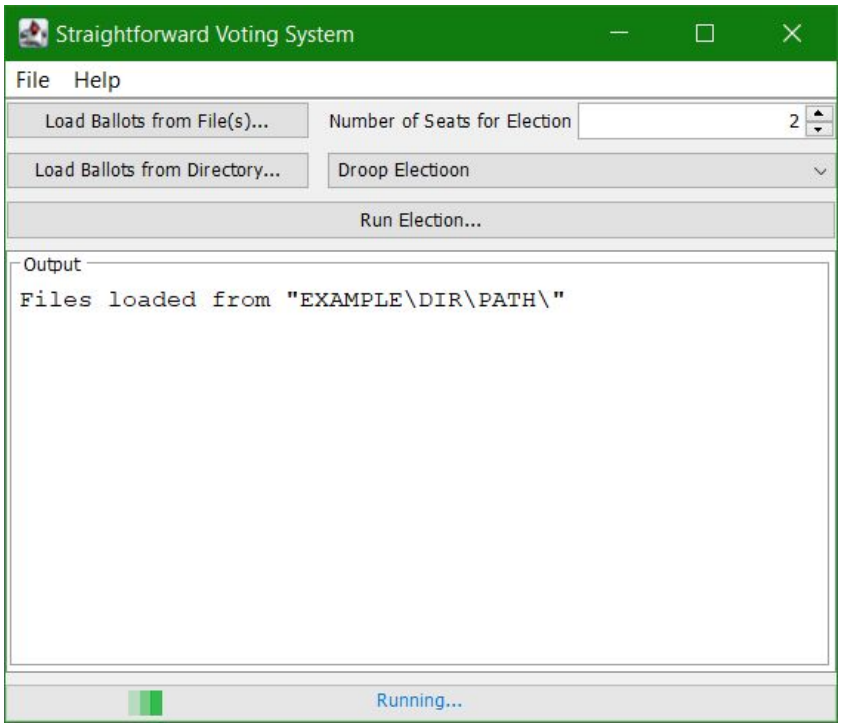


Figure 6.7: The main window is shown while an election is being run. Notice the progress bar at the bottom.

6.3. Screen Objects and Actions

In this subsection, each screen object available to the user through the user interface is listed along with a brief description, followed by a sublist of the actions (if any) that may be taken for each screen object:

- "File" Menu: The file menu provides the user with options to load ballots.
    - Selecting the option to "Load Ballots from File(s)..." will display the File Chooser to the user. The File Chooser will allow the user to select files to load.
    - Selecting the option to "Load Ballots from Directory…" will display the File Chooser to the user. The File Chooser will allow the user to select the directory which contains the files that they wish to load.
    - Selecting the "Exit" option will cause SVS to terminate unless an election is being run at the moment. If this action is taken while an election is running, then SVS will do nothing.
- "Load Ballots from File(s)..." Button: This button allows the user to load ballots by selecting files.
    - Clicking on this button will display the File Chooser to the user. The File Chooser will allow the user to select files to load.
- "Load Ballots from Directory..." Button: This button allows the user to load files by selecting a directory.
    - Clicking on this button will display the File Chooser to the user. The File Chooser will allow the user to select the directory which contains the files that they wish to load.
- File Chooser: The File Chooser is an interface which allows users to navigate their system's file system to find and select either the files or the directory containing the files that they wish to load, depending on how the File Chooser is set when displayed to the user.
    - The user may navigate through their system's file system with the File Chooser by selecting the directories and files in the window of the File Chooser.
    - Clicking on the "Open" button in the File Chooser will select whatever files or directory the user has chosen through the File Chooser.
    - Clicking on the "Cancel" button will quit the attempt to load ballots. In this case, SVS will do nothing.
- "Help" Menu: The help menu provides the user with the option to show the help window.
    - Selecting the "About" option will display the help window to the user.
- Help Window: The help window shows a text field which contains instructions for the user on how to successfully operate SVS. No actions are available to the user, beyond simply closing the help window.
- "Seats for Election" Spinner: This is a spinner, a field where the user can enter a number which may also be increased or decreased with the use of two arrow buttons, which allows the user to input the number of seats in an election.
    - The user may enter a value in the field using their keyboard. If the value entered is not a whole number (an integer greater than or equal to zero) it will be rejected and the field will remain unchanged.
    - Clicking on the upwards arrow will increment the number in the spinner's field by 1.
    - Clicking on the downwards arrow will decrement the number in the spinner's field by 1. The user cannot cause the value in the spinner's field to become less than 0 in this way - attempting to do so will be rejected and the field will remain unchanged.

- Dropdown Menu: The dropdown menu allows the user to select the type of election to be run. The user may choose either a plurality election or a single transferable vote election with the Droop quota.
    - Clicking on the dropdown menu will cause a list to drop down. The user may then select an element from this list (the election type they want to run) by clicking on one of these elements. If the user clicks elsewhere, the list will disappear and the type of election will remain unchanged.
- "Run Election..." Button: This button will run the election using the information which was provided by the user.
    - Clicking on this button will run an election if the user has entered all of the required information. If the user has loaded ballots, then SVS will proceed to determine the results of the election. Otherwise, SVS will print to the "Output" text field that no files have been loaded and an election cannot be run; no election will be run.
- "Output" Text Field: This text field provides the user with feedback on some of their actions in SVS: declaring that the user has loaded files, or giving the results of an election. The user cannot actually interact with this text (except by reading it).
- Progress Bar: The progress bar is normally empty, except when an election is being run. While an election is being run, the progress bar provides visual feedback to the user to indicate that SVS is working. The user cannot actually interact with the progress bar (except by looking at it).

7. Requirements Matrix

This section lists the requirements detailed in the Software Requirements Specification document and how it is satisfied in the software design for SVS. Requirements are on the left side of the table and the relevant cross-reference(s) are on the right. Where possible (for the functional requirements), the codes for the requirements are included.

System Feature: Disable Ballot Shuffle (UC_01)	When SVS is launched, it checks its arguments to see if the "-ds" flag was set to disable the shuffling of ballots. It passes along this information to the MainWindow, which passes it in turn to BallotFactory which either will or will not shuffle the Ballots that it loads, respectively.
System Feature: Load Ballot Files (UC_02)	The User Interface subsystem of SVS uses the LoadBallotFileAction(ActionEvent e) method to allow users to load CSV files to the system.
System Feature: Select Election Type (UC_03)	The user interface contains a drop down menu that allows the user to select the election type.
System Feature: Enter Number of Seats to Fill (UC_04)	The user interface contains a spinner widget that allows the user to input the number of open seats.
System Feature: Run Plurality Election (UC_05)	The Election subsystem calls the method RunElection() to run a plurality election in the case that the user selected the plurality algorithm in the drop down menu on the user interface.

System Feature: Run Droop Quota Election (UC_06)	The Election subsystem calls the method RunElection() to run a Droop Quota election in the case that the user selected the Droop Quota algorithm in the drop down menu on the user interface.
System Feature: Open Help Window (UC_07)	The user interface contains a “Help” menu. If this is clicked on, the “Help” menu will display an “About” option. If this option is clicked, a help window opens for the user that explains how to use the system.
System Feature: Audit File (UC_08)	The ElectionController class manages the audit file. Controllers for particular types of elections can use methods provided by the parent ElectionController class to manipulate Ballots amongst candidates. The ElectionController also provides a special method, CustomAuditFileMsg, to allow them to write more particular information to the audit file if need be. This will allow every step needed to recreate the election to be recorded in the audit file while the election is run.
Performance Requirement: SVS must be capable of determining the results of election in less than five minutes.	SVS uses data types like arrays and ArrayLists to strike a balance between abstraction and speed. SVS also limits IO requests through the disk (a program can spend a great deal of time waiting on an IO request from disk) by reading all of the CSV ballot files into memory in a single pass at the beginning. This will keep SVS running reasonably quickly so as to satisfy its time requirement.
Safety Requirement: SVS should not alter the ballot files which it loads, even if SVS crashes while reading them or determining the results of an election.	BallotFactory opens files in read-only mode. It cannot write to the files in any circumstance. That is, it cannot alter their contents. Since handling the loading of ballots from CSV files is relegated to BallotFactory and since all elections are run using Ballots in memory, the contents of the ballot files used in an election will be safe.
Software Quality Attribute: SVS must be able to be used by Election Officials effectively with only 15 minutes of training.	SVS uses a graphical user interface with only a few screen objects for the user to interact with. These screen objects are sufficient to complete the process of running an election, but are not so numerous as to be overwhelming to a new user. In addition,

8. Appendices

SVS is written in Java and makes references to some objects which are provided in the Java Development Kit. It may be useful to consult the documentation on these provided objects, as well as their attributes and methods, in the course of implementing and developing SVS. Documentation may be found for Java's APIs at https://docs.oracle.com/en/java/javase/14/docs/api/index.html.

As a further note, it is important that the random number generator used by SVS to shuffle ballots does not skew one way or the other. Regular pseudorandom number generation may not be sufficient to fairly shuffle the ballots unless great care is taken with relevant implementation. As such, special care should be taken in this area. Java's SecureRandom class may be useful for implementation since it provides a higher quality random number generator.