Team 19

Voting Systems

Software Design Document

Name (s): Isaiah Herr, Emma Barnes, Benjamin Keeney, and Moritz Lindner

Date: (02/28/2021)

TABLE OF CONTENTS

1. INTRODUCTION **2**
1.1 Purpose 2
1.2 Scope 2
1.3 Overview 2
1.4 Reference Material 3
1.5 Definitions and Acronyms 3

2. SYSTEM OVERVIEW **4**

3. SYSTEM ARCHITECTURE **5**
3.1 Architectural Design 5
3.2 Decomposition Description 6
3.3 Design Rationale 10

4. DATA DESIGN **10**
4.1 Data Description 10
4.2 Data Dictionary 11

5. COMPONENT DESIGN **11**

6. HUMAN INTERFACE DESIGN **13**
6.1 Overview of User Interface 13
6.2 Screen Images 14
6.3 Screen Objects and Actions 14

7. REQUIREMENTS MATRIX **16**

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the Voting System that handles Instant Runoff and Open Party Listing elections.

The intended audience is election officials, software developers, and testers of the system to provide a solid understanding of how the system operates.

1.2 Scope

This document contains complete descriptions of the Voting System that is being developed. Everything in this system will utilize the Java language in addition to a GUI for a user to easily interact with. Any developer working on the project is allowed to make any changes that they deem to be a necessary change.

Our group prides itself on providing efficient and easy to use software for our customers based on their needs. This system is a system that will take an election file based on IR and OPL to generate an audit file indicating winners of the election based on votes. This system quickly and efficiently calculates votes for each candidate, outputs the process to the user, and finalizes the top candidate in an audit file provided to the user.

1.3 Overview

This document is separated into seven sections, each of which describe an aspect of the voting system design.

Section 2 describes the overview of the system, including the functionality, context, and design of the product.

Section 3 describes the architecture of the system, specifically showing a modular program structure, a decomposition of that structure, and a rationale for the design of the system architecture.

Section 4 explains the process of transforming, storing, processing, and organizing data within the system alongside a dictionary of all system entries and data types with descriptions.

Section 5 is a systematic analysis of each component in the system, introduced in section 3.2.

Section 6 illustrates the user interface of the system, describing how the user will interact with the system and explains the program from the user's perspective.

Section 7 includes a requirements matrix that cross-references components and data structures of the system to the Software Requirements Specification document (SRS).

A list of each section and subsection of the document can be viewed in the Table of Contents above.

1.4 Reference Material

[IEEE] This document was adapted from the "IEEE Recommended Practice for Software Design Descriptions." *http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieeeSDD.pdf*.

[SRS] The principal source of information comes from the "Software Requirements Specification" document.
https://github.umn.edu/umn-csci-5801-S21-001/repo-Team19/blob/master/SRS/SRS_Team19.pdf

1.5 Definitions and Acronyms

Term	**Definition**
Users	Any person that will be using the system
Developer	Any person that is working on developing and creating the system
Testers	Any person that will be using the system to test for functionality and bugs.
Machine, Device	Refers to the computer that the system will be run on
Election File	File provided by relevant legal body that contains voting type, candidates, and ballots for the election
Audit File	File produced by this software that contains exhaustive information gathered by analyzing an Election File
Media File	Refined/processed form of an Audit File more suitable for direct viewing
System/Software/Program	The product for voting systems that is being developed and documented
Instant Runoff Election/Voting (IRV) (IR)	An election in which an individual ranks candidates from their favorite to their least favorite. If a candidate is eliminated, and is an individual's

	favorite, their ballot will be distributed to their next favorite candidate and so on. The candidate with the most votes is elected.
Open Party Listing (OPL)	An election in which an allocation formula is used. A quota is determined by taking the total number of ballots divided by the total number of seats. If a candidate or party reaches that quota, they are given a seat. The party/candidate with the largest remainders will be given any seats leftover.
Comma Separated Values (CSV)	A delimited text file that uses commas to separate values. It is commonly used in programs such as Microsoft Excel
Awarded seat	A seat given to a official as a result of the election process

Table 1. Definitions and acronyms

2. SYSTEM OVERVIEW

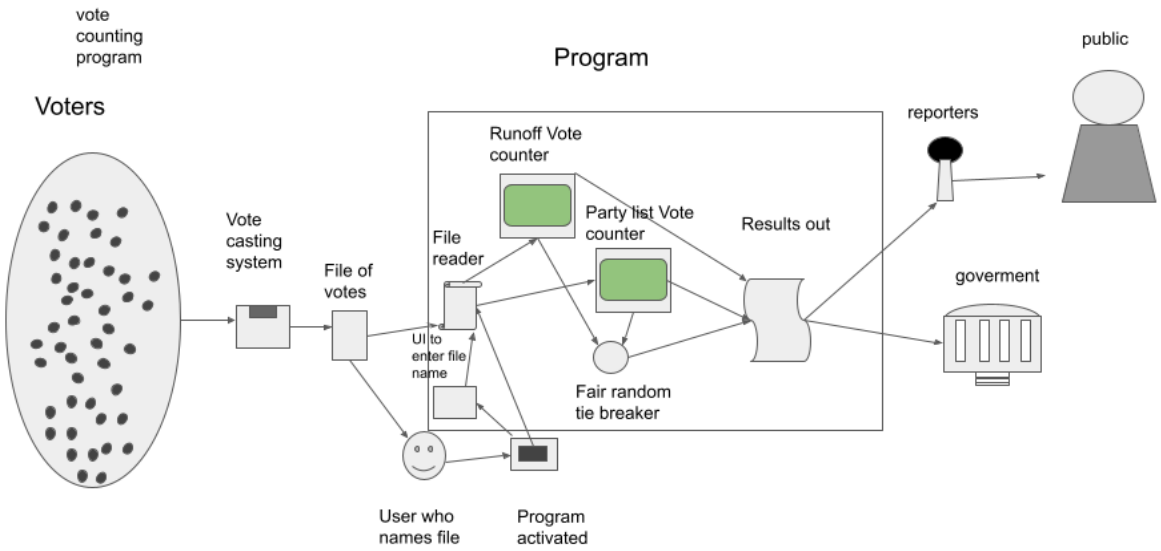


Figure 1. System overview

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

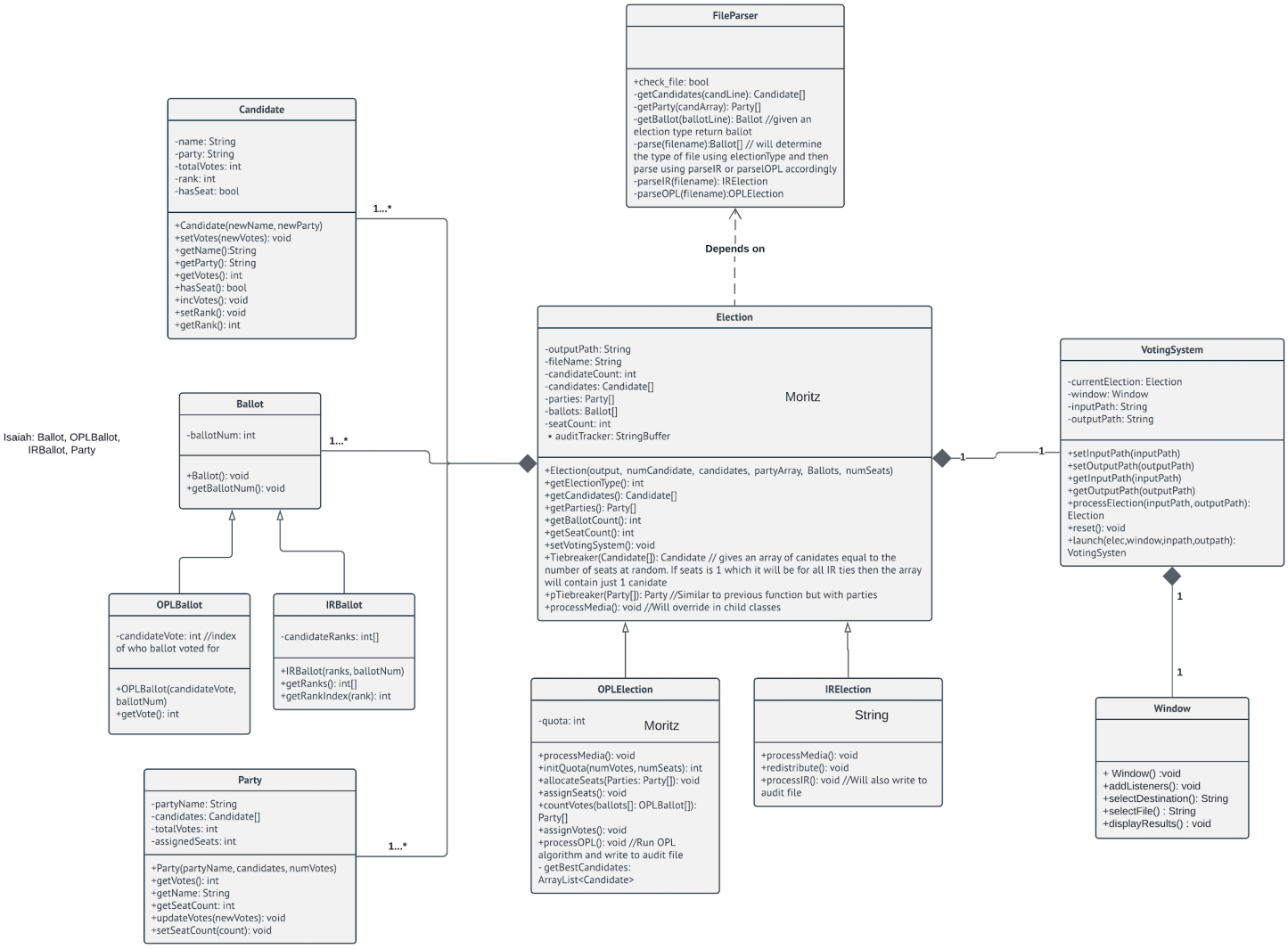


Figure 2. Class diagram

3.2 Decomposition Description

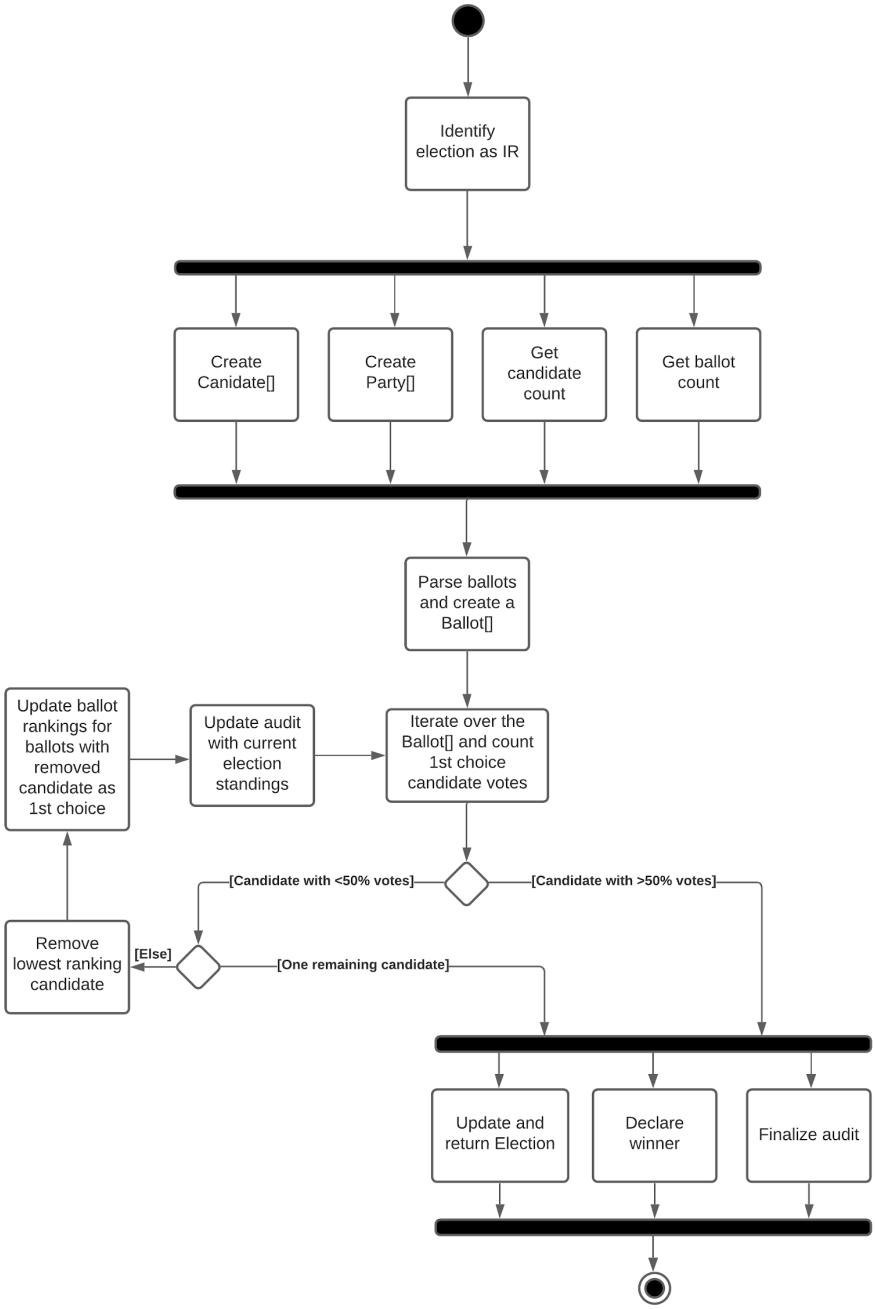
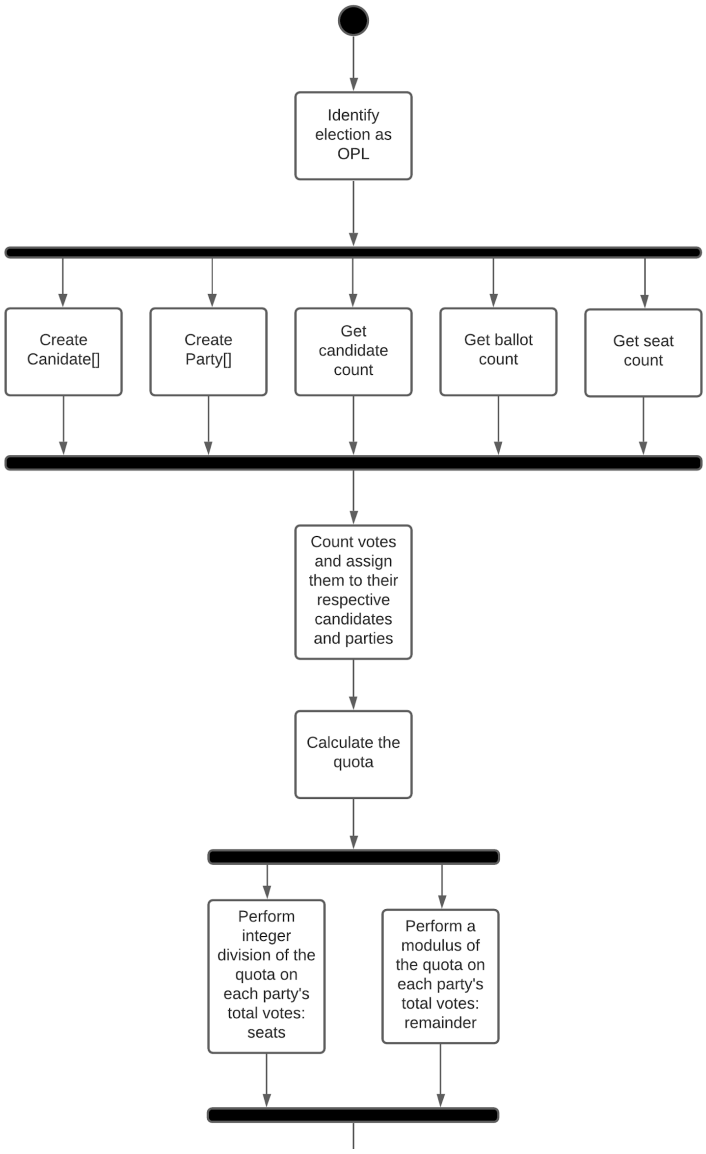


Figure 3. Instant Runoff voting activity diagram


Identify election as OPL
Create Canidate[]
Create Party[]
Get candidate count
Get ballot count
Get seat count
Count votes and assign them to their respective candidates and parties
Calculate the quota
Perform integer division of the quota on each party's total votes: seats
Perform a modulus of the quota on each party's total votes: remainder

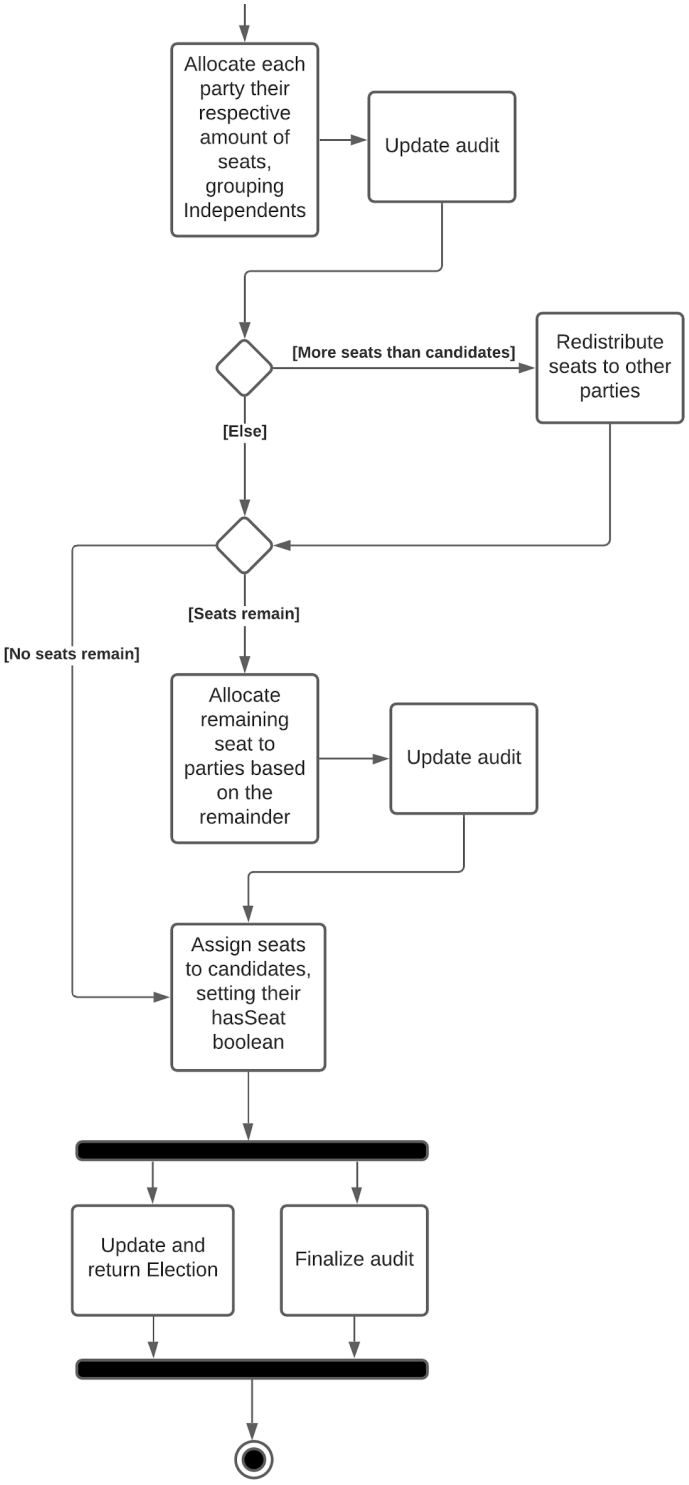


Figure 4. Open Party List voting activity diagram

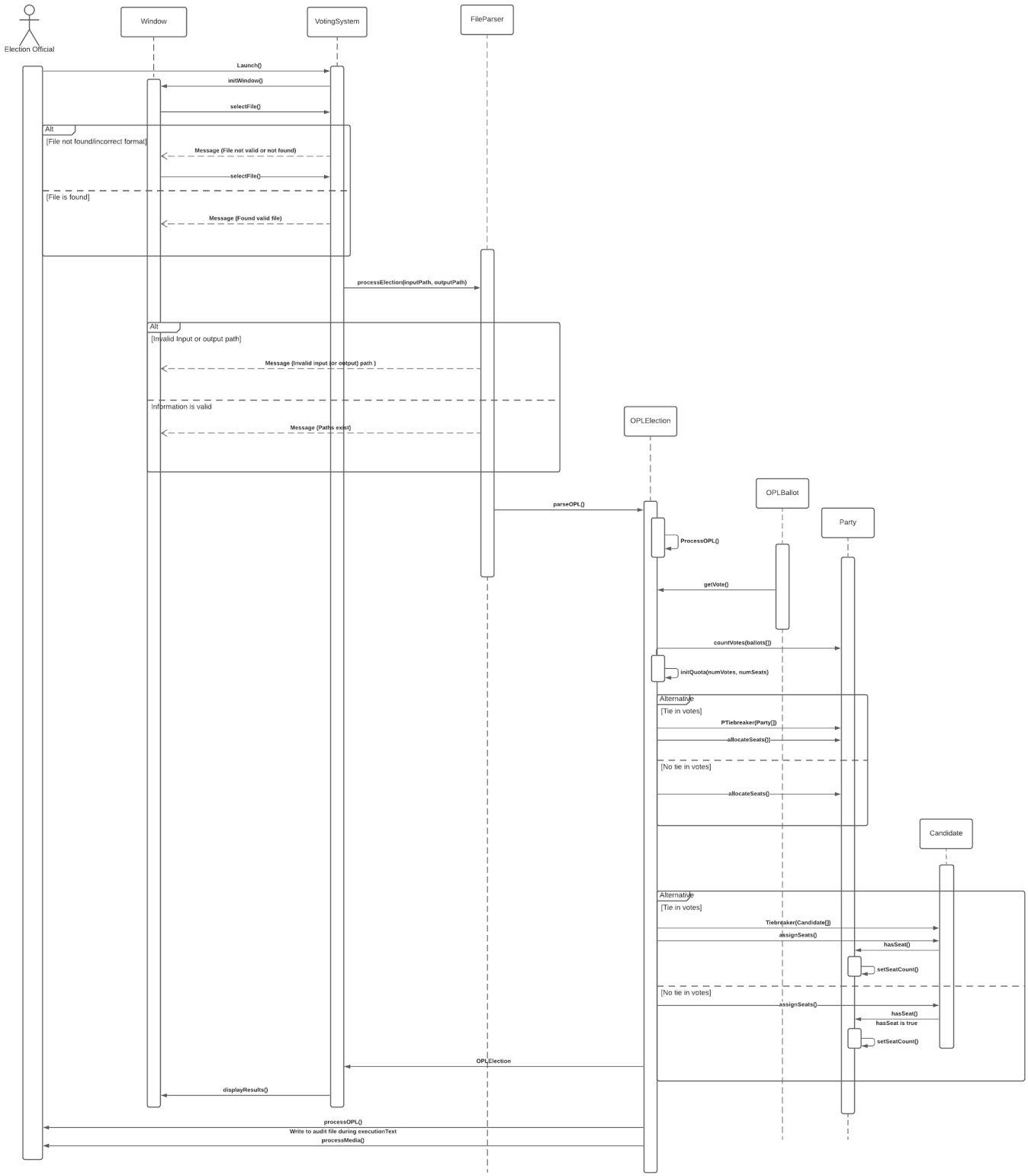


Figure 5. Open Party List sequence diagram

It is assumed that, at any point in the diagrams where a tie could possibly occur, a standard tie-breaking procedure will take place. This will include 1000 coin flips, and the 1001th flip will represent the final result of the coin flip.

3.3 Design Rationale

We chose to have a GUI because we thought it would be an interesting challenge and wanted to introduce the system in a way that was aesthetically pleasing and easy to use for the user. We are including a FileParse as a class to handle getting information from files so that we can easily input that information into an Election class. We want to be able to make sure each class can specialize in their roles to make it easier to tell where a problem in any feature may be coming from.

To represent votes we have a Ballot class. Because IR and OPL ballots need to be able to do different things we have sub classes OPLBallot and IRBallot that inherit from the ballout class. To run the election we have an Election class which similar to Ballot has subclasses OPLElection and IRElection which inherit from Election. We did this with Election because an OPL and IR election have different algorithms that the system must account for in addition to different pieces of information needed (such as the quota for an OPL election). The Election class is responsible for holding a media file's worth of information inside of it. Election represents the state of the election taking place, at any given time.

The Election class contains a Ballot array so the OPLElection and IRElection classes will also contain ballot arrays because of how inheritance works in Java. We also have the Candidate class which represents candidates that can be elected in addition to the party that they are associated with. The Candidate class is responsible for holding election information for each candidate, such as their total number of votes at any given point in the election, and, in the case of OPL, whether or not they received a seat. We also have a class to represent parties called Party and each Party class has an array representing the number of candidates. The Party class differs from the candidate in a way that it sums up the total votes of all candidates associated with a specific party. This is due to the OPL election algorithm.

Lastly we have the VotingSystem class which is a utility class. It contains the Window the Election that is currently being run and the strings to access the input and output files. VotingSystem is a bridge between the GUI and the election being analyzed. It drives the election and accepts responses from the Window. While eleven classes is not the highest number of classes that could exist it is still a lot of classes. If we don't include sub classes then that number goes down to seven which is still a lot but less. Having this many classes in some ways makes things easier as it divides up the work but is also a trade off in that we will have to manage the relationships between classes.

4. DATA DESIGN

4.1 Data Description

The election file will be read by the FileParser class and its data will be used to create an Election Class object. The election class, once everything is calculated, will create the audit file and the media file. The audit file will be created as the election file is being parsed, the media file will be created from the data contained inside the Election object.

4.2 Data Dictionary

Entity	Description
Election file	A text file containing the election type, number of seats, candidates and their parties, and all the votes. This is a file for our program to read in.
Audit file	A text file that shows information about the election.
Media file	Like the audit file but more of a summary containing who won the election the number of votes and the percentages each candidate gets

Table 2. Data sources and their descriptions

5. COMPONENT DESIGN

In 3.2, we gave diagrams explaining the activity diagrams of both IR type election processing and OPL type election processing.

IR Election Processing

Name: IR Election Processing
Type: IRElection
Description: When the user chooses to process a file, if it is determined to be an IR election file, it will be processed as an IR election.
Operations: Process Votes
Name: processIR()
Arguments: None
Return: Audit file
Pre-condition: File must be valid and must have a valid election type of IR
Post-condition: Program will then have run through the instant runoff algorithm
Exceptions: None
Flow of Events:

1. System identifies election as IR
2. System creates a Candidate[] and Party[] and counts number of candidates and number of ballots
3. System counts ballots that indicate a specific candidate as their most preferred, creating a Ballot[]
4. System eliminates candidate with lowest votes
5. Ballots that had the eliminated candidate will then distribute on to their next preferred candidate
6. Repeat steps 2-4 until the final candidate is selected
7. System prepares to create audit file

Operations: Tie Breaker
Name: tiebreaker()
Arguments: Candidate[]
Return: Candidate
Pre-condition: The program is awarding seats for an election
Post-condition: All the seats are awarded and the program will present the results to the user
Exceptions: None
Flow of Events:

1. The candidates tied for most seats are identified
2. Candidates identified are completely randomly chosen one at a time
3. The candidate that is awarded a seat is randomly chosen by doing a coin flip with another randomly chosen candidate that they are tied with
4. We repeat steps 1-3 until there is only one candidate left.

OPL Election Processing
Name: OPL Election Processing
Type: OPLElection
Description: When the user chooses to process a file, if it is determined to be an OPL election file, it will be processed as an OPL election.
Operations: Process Votes
Name: processOPL()
Arguments: None

Return: Audit file
Pre-condition: File must be valid and must have a valid election type of OPL
Post-condition: Program will then have run through the open party list algorithm
Exceptions: None

Flow of Events:

1. System identifies election as OPL
2. System creates a Candidate[] and Party[] and counts number of candidates, number of ballots, and number of seats available for distribution
3. Candidates are grouped by party
4. The quota is defined as dividing the total ballots by the number of seats to fill
5. System counts ballots, recording candidate votes as well as party votes
6. System assigns each party a number of seats equal to the integer division of the total party votes and the quota
7. If not all seats were assigned, the remaining seats are split among parties with the greatest remainder
8. Assign candidates to their party's seats based on their individual votes
9. System prepares to create audit file

Operations: Tie Breaker
Name: Tiebreaker()
Arguments: Candidate[]
Return: Candidate
Pre-condition: The program is awarding seats for an election
Post-condition: All the seats are awarded and the program will present the results to the user
Exceptions: None

Flow of Events:

5. The candidates tied for most seats are identified
6. Candidates identified are completely randomly chosen one at a time then awarded seats. We repeat this process until there are no seats left to award

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

When the program is launched the user is presented with a start screen that says Election processor. They may then click a button with the words "Enter Votes". If they do, the start screen will close, the main menu will show up, and present them with 3 buttons with the titles "Results", "Cancel", and "Configure". Before running any results, the user must input the file needed and the output file path before clicking "Results". If the user clicks the results button before inputting the necessary information in the configuration menu, a warning message pops up telling the user to enter an input file name in the configure menu. If the user clicks the cancel button then the menu closes and the program stops.

If the user clicks the configure button then the configure menu shows up and the main menu stays open. In the configuration there are 2 text boxes that allow the user to enter an input file and an output file name. There is also a choose output destination button that allows the user to choose where the output file goes. If an input file that does not exist is given, then a warning screen will pop up and tell the user to enter a new file. If an input file that exists in the same folder as the program is set as the input file, the program will function normally once launched. If the results button is pressed and the input file exists in the program will run find and create a media file with the set (or if the output file is not set default) then _media.txt and an audit file with the set (or if the output file is not set default) then _audit.txt.

6.2 Screen Images

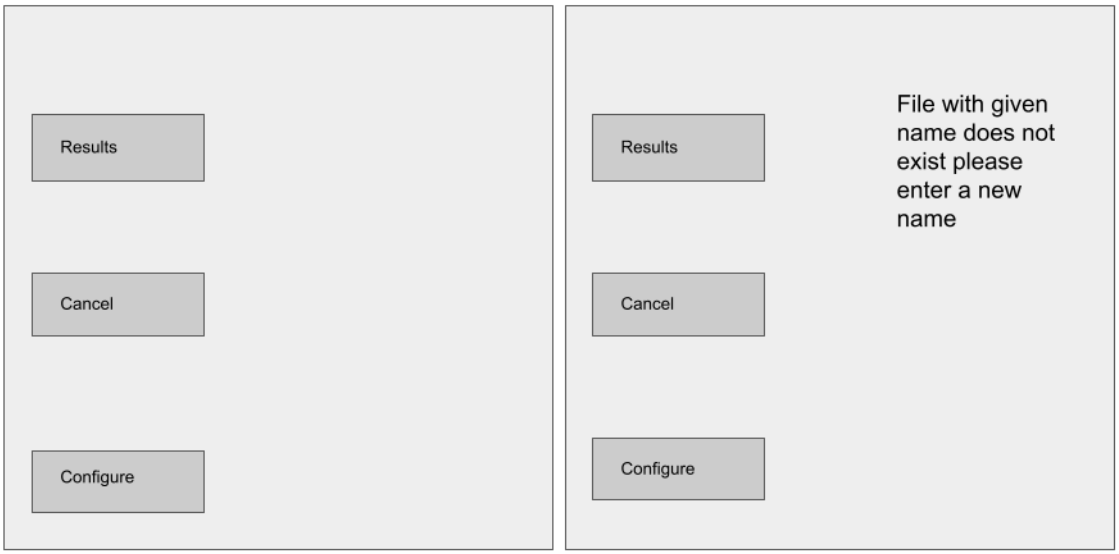


Figure 6.(left) GUI main page, (right) main menu when file with given name does not exist.

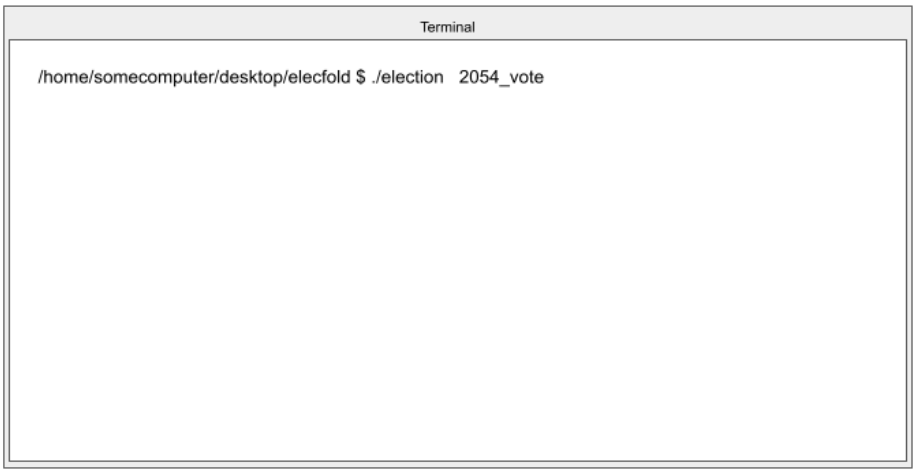


Figure 7. Program being activated in terminal

Figure 8. (left) Start screen and (right) progress bar

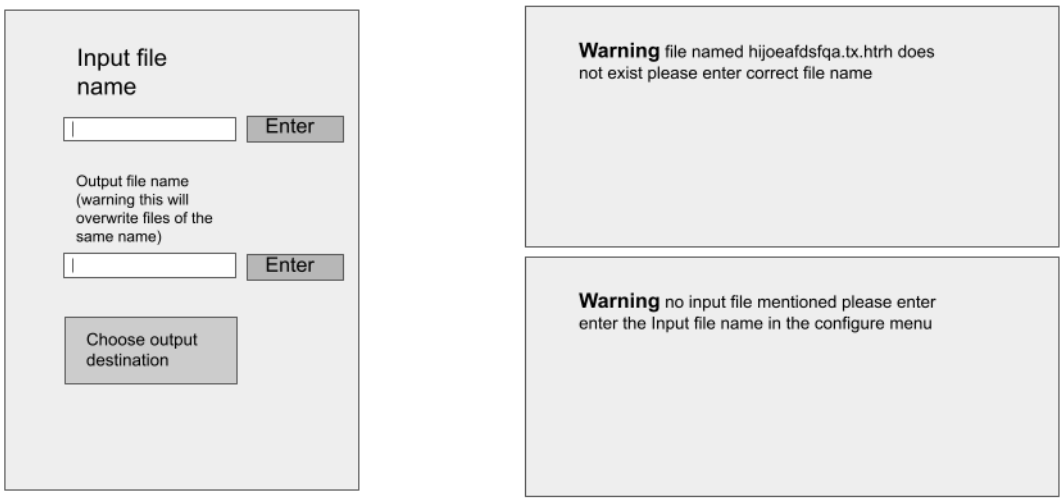


Figure 9. (left) Configuration screen, (top right) file does not exist popup, (bottom right) input file not named popup

6.3 Screen Objects and Actions

The warnings are simply windows with text that are meant to give the user important information about why something won't work and then be closed. The start screen is meant to alert the user the program started and allow the user to jump into the main menu. The main menu is the hub of the GUI allowing users to close the menu and stop the program or have the program run on the given input or open the configuration menu. The configuration menu allows the user to set the name of the program that is being read in and the set an output name if they want.

7. REQUIREMENTS MATRIX

The use cases below will be referencing the use cases found in UseCases_Team19.pdf document associated with the Software Requirements Specification.

Use case	Classes involved	Function involved
UC_01	Window, VotingSystem	buttonListener(), selectDestination(), selectFile(), setInputPath(), setOutputPath()

UC_02	OPLElection, IRElection	processOPL(), processIR()
UC_03	Election	processMedia()
UC_04	Election	Tiebreaker(), pTiebreaker()
UC_05	FileParser	electionType()
UC_06	Window, VotingSystem	buttonListener(), setInputPath(), setOutputPath()
UC_07	Window, VotingSystem	Callbacks up to VotingSystem, reset()
UC_08	Window, VotingSystem	displayResults()
UC_09	FileParser, IRElection, IRBallot, Party, Candidate	parseIR(), redistribute(), getVotes(), getSeatCount(), setSeatCount(), updateVotes(), getRanks(), getRankIndex(), incVotes(), getName(), hasSeat()
UC_10	FileParser, OPLElection, Ballot, Party, Candidate	parseOPL(), redistribute(), getVotes(), getSeatCount(), setSeatCount(), updateVotes(), getVote(), incVotes(), getName(), hasSeat()
UC_11	FileParser, OPLElection, Party	This will involve grouping Independent candidates inside of a single Party object when they are read in from the election file

Table 3. Requirements matrix