Team 14

Voting System

Software Design Document

Names:

Liam O'Neil (oneil779), Bilal Osman (osman353),
Hyehwan Ryu (ryu00032), Soorya Sundravel (sundr030)

Date: 3/3/20

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the voting system project. It details the data design and component interactions within the system. This SDD is to be used to generate the implementation of the voting system project, thus the intended audience is developers and testers who want to learn about the design of the system.

1.2 Scope

The voting system developed is intended to be used to congregate the results of an election. The system is run once the user provides documentation on election information, such as the voting method being used and the tallies of ballots. The program is capable of calculating results of two types of methods: closed party-list voting and instant runoff voting. These two methods will handle other information that is inputted through a list of ballots and parse the data to output election results to an audit file. The audit file will contain calculations and collected external data of the ballots and display candidates who have various ranges of vote percentages. The product is intended for elected officials to handle large collections of ballots and election information and produce wanted results. This software will only be used for election-related purposes and aimed to reduce complexity in ballot counting and evaluation.

1.3 Overview

The system overview section provides the general description of the system functionality and design. Sections 3 and 4 discuss the system architecture and data design, respectively, which describe the structure of the system as a whole, its detailed decomposition, and the structures and functions of data within the system. Section 5 provides an in-depth description of each object member of the system listed in the decomposition. Section 6 details the user interface and provides screen images of the interface. Following this is a requirements matrix linking the requirements of the system with its remedying functionality or components.

1.4 Reference Material

No reference material was used in the development of this design plan.

1.5 Definitions and Acronyms

CSV File - A text file that allows data to be stored in a tabular format. Can be opened and edited through the use of a spreadsheet program. Stands for comma-separated values.

IR - Instant runoff voting or election. A type of election in which voters select candidates in order of preference. First preference votes are tallied and the losing candidate's second preference votes are distributed to the proper remaining candidates. This process is repeated until a candidate receives a majority of the votes.

CPL - Closed party list voting or election. A type of election in which one or more seats are up for election and voters vote for a specific party. Each party lists their candidates in the order they would be elected should they win seats.

2. SYSTEM OVERVIEW

The general purpose of our software product is to determine and deploy the candidates who are deemed to win an election. This is to provide an effortless process for any election official to count, distribute, and determine election results in a streamlined fashion. This will be done by parsing data from a provided file of ballots that were cast outside of this system. As long as the file type is allowed and the structure of the file is correct, then the information in the file will include: the voting system name, number of parties/candidates, parties/candidates separated by commas, number of ballots, list of ballots, and number of seats only for one voting system. All this information will allow the system to simulate a specific voting system to calculate and distribute election results. There are only two voting systems that can be simulated; closed-party list voting and instant runoff voting. These two systems will have different functionality and require specific information to be gathered. Once one of the two systems is done an audit file will be created and output for viewing. This file will hold information such as the type of voting system, number of candidates/parties, candidates/party names, number of ballots, calculations per round, how many votes a candidate had, how many seats were allocated, etc. The software product is aimed to store and provide any calculations related to the election results so users can determine the accuracy of the results.

Instant Runoff Voting

One of the voting systems available to simulate will be instant runoff voting. In instant runoff voting, each ballot will select each candidate through order of preference where '1' is most preferred and so on. However, each ballot may not include a preference rating for all candidates, and there must be one candidate that is chosen as the most preferred. The number of most preferred votes for each candidate is counted and evaluated. If there is a candidate who has

50% or over of the most preferred votes, then they are deemed the winner of the election. If there is not a majority vote, then the candidate with the lowest preference rating is removed, and votes are reallocated. Mainly, the votes who had chosen the losing candidate as their first preference will have their votes moved to their second prefer if any. The votes will be counted again for the remaining candidates. This process continues until there is a candidate with a 50% and over preference rating. If there is a tie, the program is able to provide a coin flip between candidates allowing a candidate to win in any election.

Closed Party List Voting

The other voting system available for simulation is closed party list voting. This is similar to instant runoff but has ballots cast for political parties instead of candidates. Each party has a list of candidates aimed to fill a specified number of seats. The number of seats allocated for a political party will be closely approximated by the percentage of their votes. This is determined using the "largest remainder formula" which is a calculation determining the approximated percentage of votes for each party. First, a quota is determined by taking the total number of valid votes in the district and dividing this by the number of seats. Then, the quota is divided into the vote that each party received and the party wins one seat for each whole number calculated. If there are any seats left, the parties with the highest remainder of votes in order will occupy a seat until there are none left. Like IR voting, the program is able to handle ties in vote percentages to determine who will take the remaining seats. For each party that occupies a seat, a candidate will be elected from that party from an ordered list determined in the ballot file.

Audit File

An audit file will be created to store all data from a simulated election. As mentioned earlier, this will be how election information is accessed to the user for viewing and analysis. All data related to the election will be stored here such as election winner(s), progression of elections, party/candidate ballots and every recast of ballots. All this data is aimed so the audit can replicate the election itself.

3. System Architecture

3.1 Architectural Design

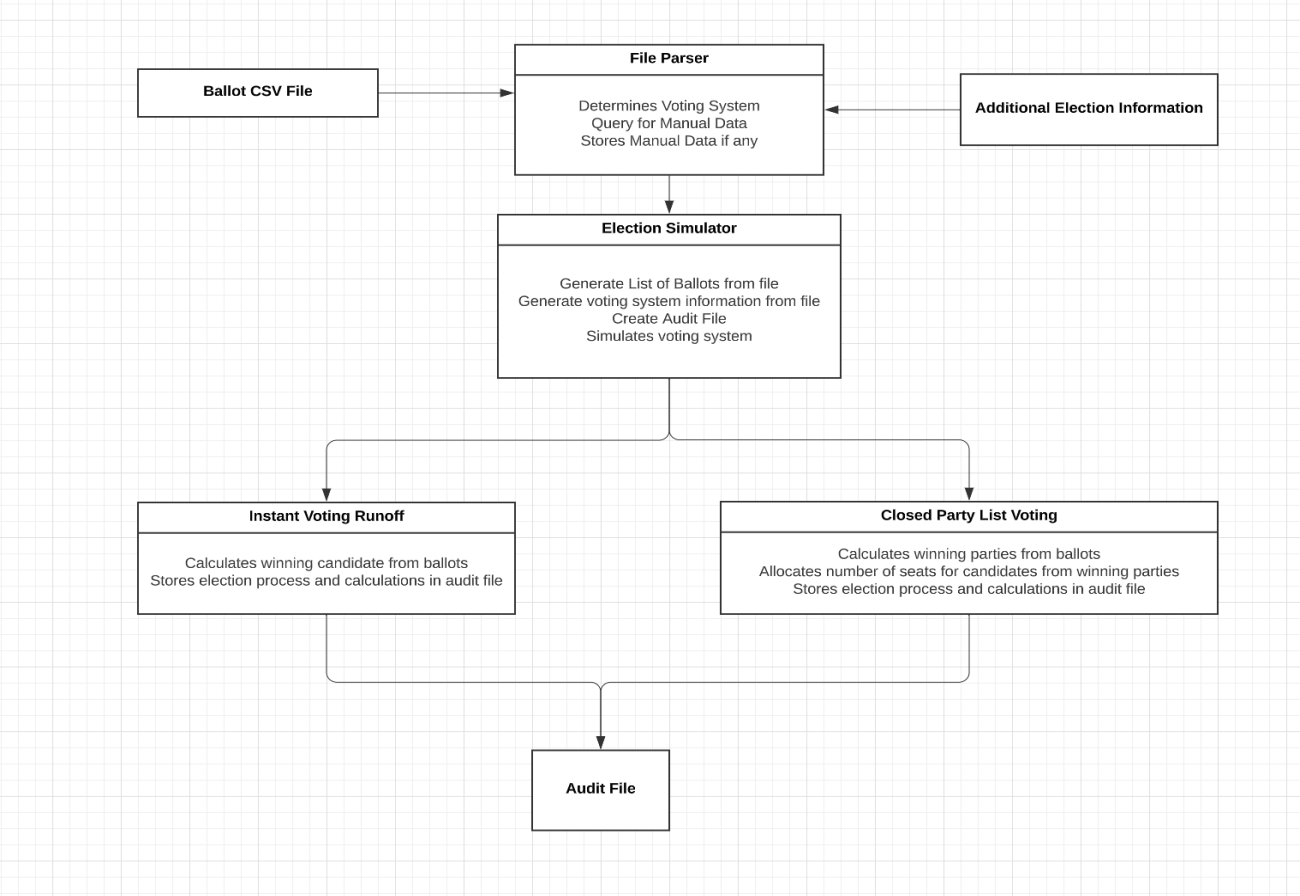


FIGURE 3.1 ARCHITECTURAL SYSTEM DESIGN DIAGRAM

3.2 Decomposition Description

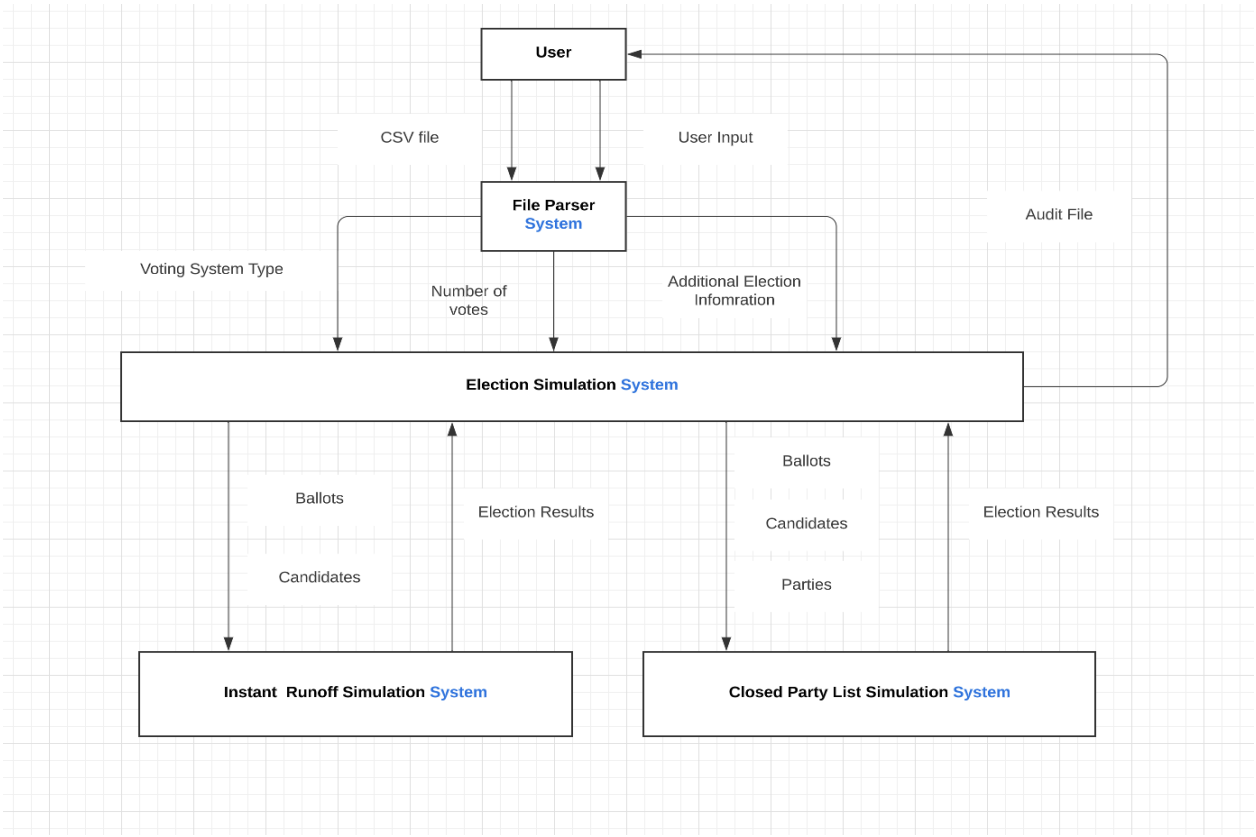


FIGURE 3.2.1: DATA FLOW DIAGRAM

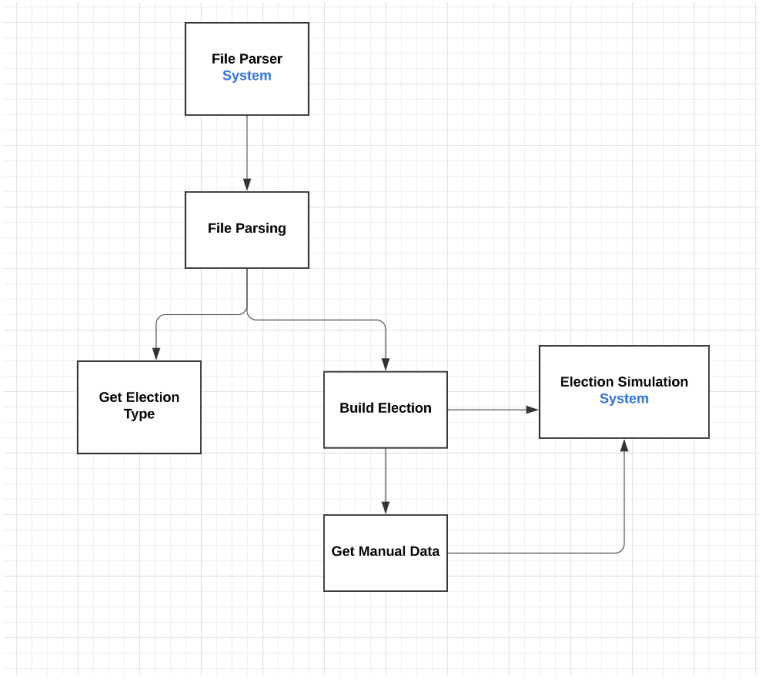


FIGURE 3.3.1: FILE PARSER SYSTEM DECOMPOSITION DIAGRAM

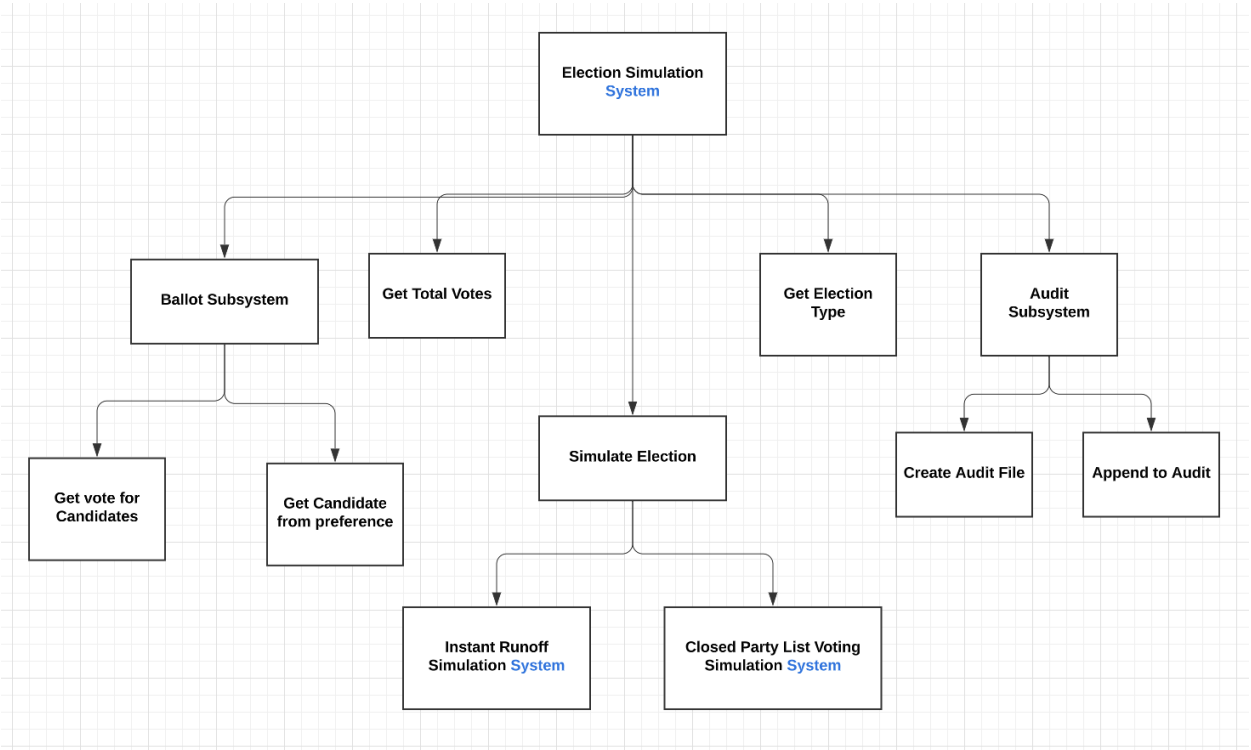


FIGURE 3.3.2: ELECTION SIMULATION DECOMPOSITION DIAGRAM

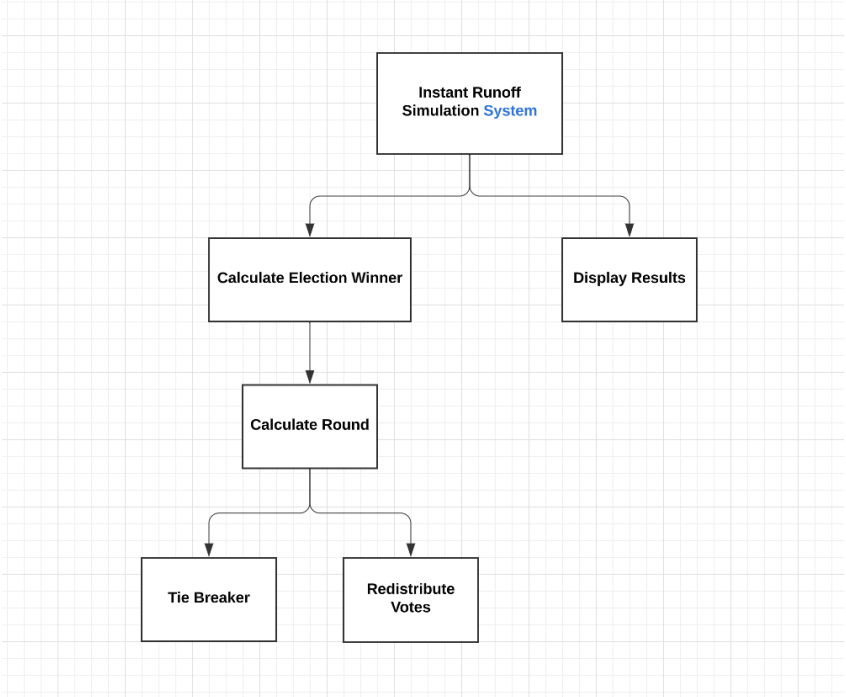


FIGURE 3.3.3: INSTANT RUNOFF SIMULATION DECOMPOSITION DIAGRAM

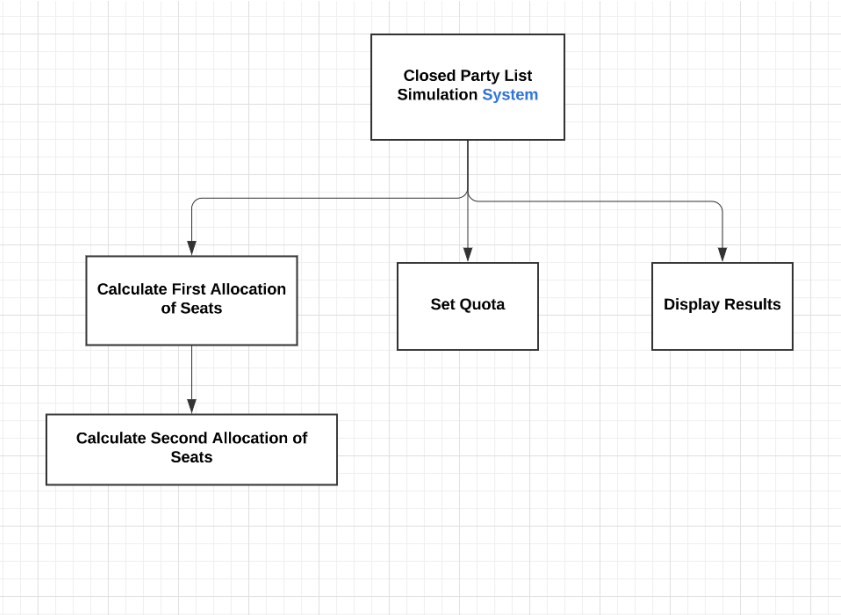


FIGURE 3.3.4: CLOSED PARTY LIST SIMULATION DECOMPOSITION DIAGRAM

3.3 Design Rationale

To develop a system for running elections, there are different active components. By using distinct classes, we can group together related member variables and methods into a singular file. We are able to develop modular programs with functionality that can be tested in self-contained units. Using classes also has the advantage of extensibility that, for future updates, running new election methods doesn't need any modifications to the original code. We can just create a new class for each method of the voting algorithm.

We use the PoliticalParty and IRCandidate classes to group together member variables alike. We can create objects and access them with ease through getter and setter methods. At first, we considered not using separate classes for the political parties/candidates. However, implementing classes makes it much easier for planning out the system and development, as we can group related variables and methods together within the same class.

We have a base class called Election to run the election process. The children classes of InstantRunoff and ClosedPartyList are variations of this voting and inherit the necessary variables. Inheritance helps reduce redundant code. Similarly, having a separate class to read and process the CSV file's information ensures that the voting classes don't have to reimplement the same logic for reading data from files. An issue that we ran into with the InstantRunoff class was storing the preference for each candidate. Using a separate class for Ballot helps store the voting information, and the use of hashmaps associate the candidates with their respective preferences.

4. Data Design

4.1 Data Description

A CSV file contains the original information regarding the type of election and the votes. The file is then read by a file parser, and the appropriate election program is run. The Ballot class stores information about candidates/parties and their corresponding votes/preferences using hashmaps. For CPL, we use a separate class named PoliticalParty that stores relevant information as attributes. Each PoliticalParty object has information about the party name (String), party seats (int), party votes (int), and remainder votes (int). We also used a queue of strings to store the names of candidates for each political party. Since a CPL election uses an ordered list of candidates to get elected, it made sense to use a queue to retrieve the officials in order. Within the ClosedPartyList class, we use a list of PoliticalParty objects, a list of Ballot objects, totalVotes (int), and seatsAvailable (int) to determine the quota and assign the appropriate number of seats to the political parties.

4.2 Data Dictionary

Variables:

auditFile : Audit : The audit file contains the results of the details of the election
auditFileName: String : The name of the audit file
ballots : List<Ballot> : List of ballots containing information about candidates and votes
- candidates : List<IRCandidate> : List of candidates for Instant Runoff Voting
- candidateNames : Queue<String> : Ordered list of candidates for each political party
- electionInfo: String : Election information for IR and CPL
electionType : String : Specify the type of election to run between IR or CPL
- file : File : CSV file with election information to use for parsing
- filepath : String : The path of the audit file within the directory
- losers : Map<IRCandidate, int> : Eliminated candidates in instant runoff, along with their votes
- name : String : The name of Instant Runoff candidate
- parties : List<PoliticalParty> : The list of political party
- quota : double = 0 : Total Votes divided by number of seats. Used in the calculation of allocating seats
- partyName : String : The name of the political party
- partySeats : int = 0 : The number of seats of a party
- partyVotes : int : Number of votes that a party received
- percentVotes : int : Percent of votes for a specific candidate. Used to determine the majority of votes.

- remainderVotes: int = 0 : The number of the remaining votes after the first allocation of seats in CPL voting system
- seatsAvailable : int : Total number of seats available for CPL voting system
totalVotes : int : The total number of votes in an election
- vote : Map<String, int> : A mapping from a candidate or party, to their votes or preference
- votes : int : Number of votes for a specific candidate in IR voting system
- winningCandidate: String : The name of winning candidate in IR voting system

Methods:

+ Audit() : Default Constructor
+ Audit(String filename) : Construtor with specified filename for output
+ appendString(String info) : void : Add information onto the audit file
+ Ballot(Map<String, int> vote) : Ballot Constructor, with voting information
+ calculateFirstAlloc() : void : Calculate the first allocation of seats for CPL
- calculateRound() : void : For each round in IR voting,
- calculateSecondAlloc() : void : Calculate the second allocation of seats for CPL
+ calculateWinner() : void : Determine the winner if there is a majority in votes for a candidate
+ ClosedPartyList(int totalVotes, List<Ballot> ballots, Audit auditFile, int seatsAvailable, List<PoliticalParty> parties) : Constructor for CPL with total votes, voting information, output file, available seats for distribution, and a list of political parties
+ displayResults() : void : Display the result of the election
+ buildElection(String type) : Call the Election class from FileParsing to run simulation
+ Election(String electionType, int voters, String line_ballots, Audit auditFile): Constructor for the Election class. Specifies the type of election, number of voters, ballots, and output audit file.
+ FileParsing(String filename): Search to read a specific CSV file based on the filename
+ getAudit(String filename) : void : Get the output audit file based on the filename
+ getBallots() : List<Ballot> : Return a list of ballots with voting information
+ getCandidateNames(): Queue<String> : Get the names of the candidates for CPL
+ getCandidateFromPreference(int preference) : String : From the ballot class, get the candidate name based on the preference.
+ getElectionType() : String : Get the type of the election, either IR or CPL
- getManualData(String input) : void : given a string, find the data in the CSV file
+ getName() : String : Get the name of the candidate
+ getPartyName(): String : Get the name of the party
+ getPartySeats(): int :Get the number of the seats of the party
+ getPartyVotes(): int : Get the number of the votes of the party
+ getPercent() : int : Returns the percentage of voters for a given candidate
+ getTotalVotes() : int : Get the number of the total votes
+ getVotes() : int : Get the number of the votes of a given candidate
+ getVoteForCandidate(String candidate) : int : Within the Ballot class, returns either the votes or preference for a candidate/party

+ InstantRunoff(int totalVotes, List<Ballot> ballots, Audit auditFile, List<IRCandidate> candidates) : Constructor for the InstantRunoff Class, with parameters of total votes, list of ballots with voting information, output auditfile, and list of candidates. The class determines an elected official.

+ IRCandidate(String name) : The name of the candidate of IR voting system

+ PoliticalParty(String partyName, Queue<String> candidateNames) : Constructor for the PoliticalParty class, containing information about the partyName, and an ordered list of the names of candidates from the party. Used in CPL voting

- redistributeVotes(IRCandidate candidate) : void : Redistributes votes from the eliminated candidate to the voters' second choices. Used in IR voting.

+ setPartyVotes(int votes): void : Set the number of votes for a party

+ setPartySeats(int seats): void : Set the number of seats of the party

+ setPercent(int) : void : Set the percentage of total votes for a candidate

+ setVotes(int) : void : Set the number of votes for an candidate

- setQuota() : void : Sets the quota for CPL voting. Determined by total votes divided by available seats.

+ simulateElection(): void : Runs the election simulation program

- tieBreaker(List<IRCandidate> candidates) : String : Flip a coin to decide a winner when there is a tie between candidates in IR.

5. Component Design

Procedure: File Parsing
Input: filename: String
Output: none: Void
Steps:

1. Open file using filename
2. Collect and store voting system type in a String
3. Store number of votes in an int
4. Store ballots in a String list
5. Store candidates or parties in a String list
6. Close file

Procedure: Get Manual Data
Input: input: String
Output: none: Void
Steps:

1. Parse information from input
2. Store in appropriate data types

Procedure: Get Election Type
Input: none
Output: String
Steps:

1. Return voting system String

Procedure: Build Election
Input: none
Output: String
Steps:

1. Ask user for input for additional information
2. Call 'Get Manual Data' procedure to handle information
3. Create Election object & store all election information into it
4. Simulate election (further steps will be discussed in later procedures)

Procedure: Get Total Votes
Input: none
Output: int
Steps:

1. Return number of votes cast

Procedure: Get Ballots
Input: none
Output: List<Ballot>
Steps:

1. Create a list of ballot objects
2. Loop through each ballot in String list
3. Create a ballot object for each ballot
4. Map each candidate/political party to the ballot's preference ranking of that candidate using a Map structure
5. Append to list of ballot objects

Procedure: Get Vote From Candidate
Input: candidate: String
Output: none: Void
Steps:

1. Loop through ballot hashmap to find the candidate
2. Return the preference number mapped to that candidate

Procedure: Get Candidate From Preference
Input: preference: int
Output: String
Steps:

1. Loop through ballot hashmap to find the preference rank
2. Return the candidate with that preference

Procedure: Create Audit File
Input: filename: String
Output: none: Void
Steps:

1. Store inputted file name as the audit file name in current election object
2. Open and create Audit file using inputted file name
3. Write contents of election information in current election object before simulating election
4. Close file

Procedure: Append to Audit
Input: info: String
Output: none: Void
Steps:

1. Open file using auditFileName attribute
2. Write passing in information to file
3. Close file

Procedure: Simulate Election
Input: none
Output: none: Void
Steps:

1. Using 'electionType' attribute, create a class based on the voting system
2. Create an 'IRcandidate' or 'PoliticalParty' list
3. In the class, create a candidate/party object, depending on voting system, for each candidate with attributes such as name and number of votes
4. Look through all ballots and aggregate each candidate's most preferred votes, calculate and store percentage in 'percentVotes' attributes
5. Store each 'IRcandidate' or 'PolitcalParty' object in respective list
6. Call 'Calculate Winner' procedure to determine results of election
7. Call 'Display Results' procedure to get election information and use 'Append to Audit' procedure to store election process and results into audit file

Procedure: Calculate Winner
Input: none
Output: none: Void
Steps:

1. Look through all ballots and aggregate each candidate's most preferred votes
2. Determine if there is a candidate with a 50% majority
3. If there is, append candidate's name as winner and append other calculations to electionInfo attribute
4. If not, call 'Calculate Round' procedure

Procedure: Calculate Round
Input: none
Output: none: Void
Steps:

1. If there is a tie between two candidate preference, call the 'Tie Breaker' procedure
2. Determine a loser by finding the candidate with the lowest preference percentage
3. Call 'Redistribute Votes' procedure

Procedure: Redistribute Votes
Input: candidate: IRCandidate
Output: none: Void
Steps:

1. All ballots that have chosen this candidate as first preference will be searched for
2. Their next preferred candidate, if any, will have their percentage increased
3. Remove candidate with the lowest preference in the IRCandidate list
4. Store losing candidate in 'losers' hashmap
5. Store distributions and removal in 'electionInfo'

6. Call ‘Calculate Winner’ procedure

Procedure: Tie Breaker
Input: candidates: List<IRCandidate>
Output: String
Steps:
1. Choose one candidate from the list of candidates
2. Candidates not chosen will be stored in ‘losers’ hashmap
3. Store result in ‘electionInfo’
4. Return the candidate name who won

Procedure: Set Quota
Input: none
Output: none: Void
Steps:
1. Calculate the quota by taking the ‘totalVotes’ attribute and dividing this by the ‘seatsAvailable’ attribute
2. Store this value in ‘quota’ attribute

Procedure: Calculate First Allocation of Seats
Input: none
Output: none: Void
Steps:
1. Look through all ‘PoliticalParty’ objects in ‘parties’ attribute and calculate eachof their majority percentage
2. Divide percentage by ‘quota’ attribute
3. Store remainder of value in ‘remainderVotes’ attribute and store quotient in ‘partySeats’ attribute
4. Subtract ‘partySeats’ attribute of the political party from ‘availableSeats’
5. Store calculations and ‘PoliticalParty’ object information in ‘electionInfo’ attribute
6. Call ‘Calculate Second Allocation of Seats’ procedure

Procedure: Calculate First Allocation of Seats
Input: none
Output: none: Void
Steps:
1. Look through all ‘PoliticalParty’ objects in ‘parties’ attribute and find the highest ranking in ‘remainderVotes’ attribute
2. For each seat left in ‘availableSeats’ attribute, add a seat to the ‘partySeats’ attribute in a ‘PoliticalParty’ object
3. Store results in ‘electionInfo’ attribute

Procedure: Display Results
Input: none
Output: none: Void
Steps:
1. Print out the String of the ‘electionInfo’ attribute

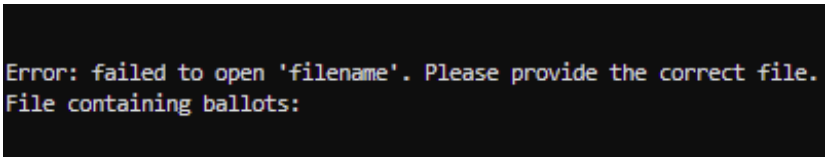
6. Human Interface Design

6.1 Overview of User Interface

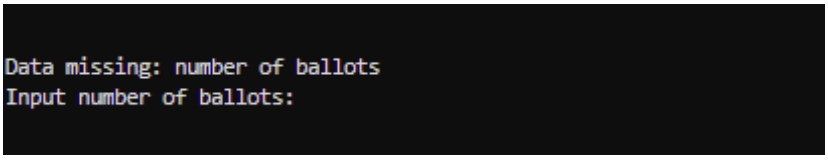
All system interfacing will occur through the command-line terminal, beginning with execution of the system. Following this, the user will see a prompt for a filename input. Upon entering the filename of the CSV containing the ballot data, the user will see error information if any exceptions occur with the input. In the event any expected data is missing from the input file, the user will be prompted to manually input the necessary data. Upon completion of the election calculations, the user will be provided the name and location of the audit file, as well as the results and additional information about the election.

6.2 Screen Images

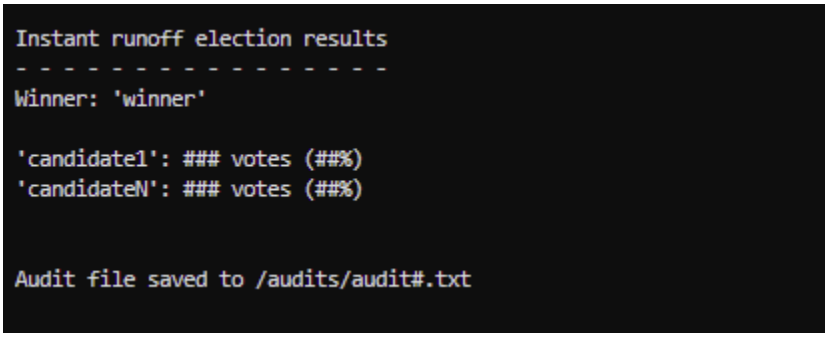
Initial prompt upon executing the system.



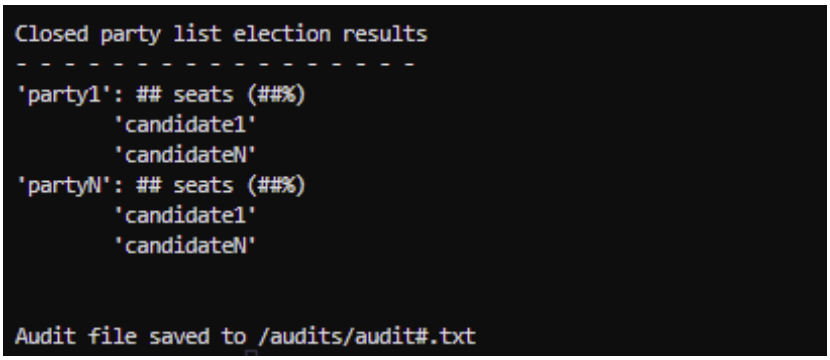
Error presented to the user if ‘filename’ input cannot be opened or read.



An example of the prompt to manually input data if anything is missing from the input file.



Results shown to the user upon completion of the election calculations for an IR election.



Results shown to the user upon completion of the election calculations for a CPL election.

6.3 Screen Objects and Actions

As all interactions occur through the terminal, there are no interactive objects in the user interface. The user will interact with the system only when providing the file containing the ballot data and inputting manual data.

7. Requirements Matrix

REQ-1	Satisfied in the *Main* class.
REQ-2	Satisfied in the constructor and *buildElection* method of the *FileParsing* class.
REQ-3	Satisfied in the constructor of the *FileParsing* class when the *File* attribute is built.
REQ-4	Satisfied in the *Main* class if an exception occurs in the constructor of the *FileParsing* class.
REQ-5	Satisfied in the *buildElection* method of the *FileParsing* class. The data structure used is an *Election* object.
REQ-6	Satisfied in the *buildElection* method of the *FileParsing* class.
REQ-7	Satisfied in the *buildElection* method of the *FileParsing* class.
REQ-8	Satisfied in the *getManualData* method of the *FileParsing* class. This method will be called by *buildElection* in the event data is missing from the input.
REQ-9	Satisfied in the *buildElection* method of the *FileParsing* class when *getManualData* returns the user's input.
REQ-10	Satisfied in the *buildElection* and *getManualData* methods of the *FileParsing* class.
REQ-11	Satisfied in the *getElectionType* method of the *FileParsing* class and the *electionType* attribute of the *Election* class.

REQ-12	Satisfied in the *calculateWinner* and *calculateRound* methods of the *InstantRunoff* class.
REQ-13	Satisfied in the *calculateWinner* method of the *InstantRunoff* class.
REQ-14	Satisfied in the *calculateWinner* method of the *InstantRunoff* class.
REQ-15	Satisfied in the *calculateRound* method of the *InstantRunoff* class, which will use the *losers* attribute to track candidates and the round they were eliminated.
REQ-16	Satisfied in the *redistributeVotes* method of the *InstantRunoff* class.
REQ-17	Satisfied in the *calculateWinner* method of the *InstantRunoff* class, which will repeat calculations until a winner is determined.
REQ-18	Satisfied in the *tieBreaker* method of the *InstantRunoff* class.
REQ-19	Satisfied in all methods of the *InstantRunoff* class, all methods of the *ClosedPartyList* class, and the *auditFile* attribute of the *Election* class. The methods of the two classes append to the *Audit* object at each step in the election progression.
REQ-20	Satisfied in the *setQuota* method and *quota* attribute of the *ClosedPartyList* class.
REQ-21	Satisfied in the *ClosedPartyList* constructor and the *partyVotes* attribute of the *PoliticalParty* class, which are stored in the *parties* attribute of the *ClosedPartyList* class.
REQ-22	Satisfied in the *calculateFirstAlloc* method of the *ClosedPartyList* class.
REQ-23	Satisfied in the *calculateFirstAlloc* method of the *ClosedPartyList* class. Remaining votes will be stored in the *remainderVotes* attribute of each *PoliticalParty* object.
REQ-24	Satisfied in the *calculateSecondAlloc* method of the *ClosedPartyList* class.
REQ-25	Satisfied in the *calculateFirstAlloc* and *calculateSecondAlloc*, which update the *PoliticalParty* objects in *parties* to reflect the number of seats won.
REQ-26	Satisfied in the *calculateRound* method of the *InstantRunoff* class.
REQ-27	Satisfied in the *tieBreaker* method of the *InstantRunoff* class.
REQ-28	Satisfied in the *tieBreaker* method of the *InstantRunoff* class.
REQ-29	Satisfied in the *calculateRound* method of the *InstantRunoff* class, which utilizes the return value of *tieBreaker* (the losing candidate).
REQ-30	Satisfied in the *tieBreaker* method of the *InstantRunoff* class. The *auditFile* attribute of the class will be used throughout the calculations of *tieBreaker*.
REQ-31	Satisfied in the *buildElection* method of the *FileParsing* class, which creates an *Audit* object to pass to the *Election* constructor.
REQ-32	Satisfied in the *buildElection* method of the *FileParsing* class. Upon initialization of the *Audit* object, the method will append the input to the object.
REQ-33	Satisfied in the *buildElection* method of the *FileParsing* class.
REQ-34	See REQ-19.

REQ-35	Satisfied in the *calculateWinner* method of the *InstantRunoff* class and the *calculateFirstAlloc* method of the *ClosedPartyList* class.
REQ-36	Satisfied in the constructor of the *Audit* class upon creation of the *file* attribute.
REQ-37	Satisfied in the *displayResults* methods of the *InstantRunoff* and *ClosedPartyList* classes.
REQ-38	Satisfied in the *displayResults* methods of the *InstantRunoff* and *ClosedPartyList* classes. The information will be gathered from the class attributes *winningCandidates, candidates* and *losers* for an IR election, the *parties* and *seatsAvailable* attributes for a CPL election, and the *ballots* attribute of the *Election* class for both types of election.

8. APPENDICES

This section is optional.

Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document