SOFTWARE DESIGN DOCUMENT (SDD)

for

Project #1 - Ballot Counting System (BCS)

Prepared by:
Jason C. Phadnis
Kobin J. Khadka
Brandon C. Schenck
Gavin Huang

Team 5

2/24/2021

1 Introduction

1.1 Purpose

This software design document specifies the inner workings of the program for the Ballot Counting System, henceforth referred to as BCS. This allows us to create a well thought out design before starting to code the program. This document will therefore both decrease development time of the program in the long run and allow developers to more easily understand and edit the program in the future. This document is intended to be used by the election officials using the software and potential developers.

1.2 Scope

The BCS is the software that is used in the voting process. In this software, it is capable of running two different types of elections (Instant runoff and Open Party Listing). The user will be able to input the ballots through a csv file. The system will sort and count the votes. When all the votes are counted, the system will determine the winner based on the algorithm it contains. When the software is finished running, it will display the results on the screen and create an audit file. This allows the user to get the results of an election quickly and accurately.

1.3 Overview

Section 2 of this document covers a summary of the program we are developing. Section 3 goes in depth into the classes that we will be creating and their variables and functions. We also go into detail about the overall process for both the Instant Runoff and Open Party Listing algorithms. In section 4 we go over how we will be storing data throughout the runtime of our program. In section 5 we go more in depth into the process of the running program from beginning to end. Section 6 covers what the user interface will look like.

1.4 Reference Material

N/A

1.3 Definitions and Acronyms

- BCS: Ballot Counting System - the name of the program we are creating
- IR: Instant runoff election type
- OPL: Open party listing election type

2 System Overview

The Ballot Counting System takes in a CSV file with election information and uses that information to run Open Party Listing or Instant Runoff voting algorithms (depending on which one is specified on the first line of the CSV file) to count the votes and declare a winner or

winners of the election. Our system will also print the results of the election to the screen, create a press release file with necessary statistics and information for the media to use, and it will create an audit file that will be used as a “history” of what happened in the election. Our Ballot Counting System will be an Object Oriented system using classes and methods to turn information from the file into objects that are easier for the system to use.

3 System Architecture

3.1 Architectural Design

Refer to the class diagram uploaded in github.

The program is simplistic in its goal: it runs the election and reports the result in an audit file. In the class diagram, it demonstrates the relationship between the interfaces and classes in the program. The Ballot Counting System is associated with all the classes in the program. This means that it will interact and call OPL ballot, IR ballot, Report, Party, and Candidate classes. It is a linear and non complex way to achieve the goal of providing the election results quickly and accurately.

3.2 Decomposition Description

3.2.1

Refer to IR and OPL activity diagram uploaded in github

The first step is to check that the user provided a CSV file. If the file does not exist, the system will provide an error message in the terminal. It will allow the user to input the file name again. If the file exists, it is opened and and voting type is checked (Open party listing or Instant Runoff). If it is an open party listing, then the program will run the open party listing algorithm. If it is an instant runoff, then the program will run the instant runoff algorithm.

For open party listing, the system retrieves the ballot info file the user inputted from earlier. It retrieves the number of candidates and seats. After retrieving the information, the program will create a candidates object for each candidate and group candidates by party. It will also sum each party's total votes. The quota is then calculated. It will allocate seats to each party and determine the remainder of votes. If there is still a remainder, the program will repeat until there is no remainder. After distributing the seats amongst party members who have the most votes, it will check if all the seats have been filled. If it is filled, then the system will finish and create an audit file. If it is not filled and there is a tie conflicting the election result on who gets a seat, there will be a coin flip to eliminate the loser. As stated again, the system will finish and create an audit file.

For the instant runoff, the system retrieves the ballot info file the user inputted from earlier. It will create a candidate object for each candidate and distribute the votes. If a candidate retrieves more than fifty percent of the votes, the system will declare a winner and send the results to create an audit file. If there is no candidate with over fifty percent of the votes, it will check if there is a tie. When there is a tie, there will be a coin toss and eliminate the top candidates with the most votes. If there is no tie, it will eliminate the candidate with the least amount of votes. The program will cycle until there is a candidate with more than fifty percent of the votes.

3.2.2
Refer to OPL sequence diagram in github.

We start by obtaining a filename from the user i.e. the election official. If the file name given is not that of an appropriate file in the directory of the program, then an error message is returned by main. Otherwise main opens the file. Using this file, main creates a Candidate object for each candidate and creates a Party object for each different party. These Party objects also have a list of these Candidate objects which are added with addCandidate(candidate:Candidate). Once all candidates are created, main loops through all the ballot info creating an OPLBallot object for each one. The Candidates' vote counts are also updated based on where all of these ballots are meant to go. After going through all of the ballots, all of the Parties have their party-wide vote count set to the total votes of their candidates. Then the ballot counting system (main) calculates the quota and determines how many seats each party gets. These are set with Party's setNumSeatsToDistribute(numSeats:int). Main uses the distributeSeatsToCandidates() function of each Party to determine the winners. Finally, main uses Report's generateReport() function to gather the necessary information, displays the results to the console, and generates the press release file.

3.3 Design Rationale

The methods and classes that were developed for this program allow developers to update, modify, and reuse in the future. We try to eliminate complex algorithms and make it simple and efficient. This means that the objects created are all connected to the ballot counting system (main). It will help us achieve the goal of processing at least 100,000 votes in 8 seconds. In our current analysis of our architecture which was provided in 3.1, there are no major negative tradeoffs for creating this way. For an open party list, the candidates will be stored in an arraylist of Party objects. This will make storing the candidate's object much easier and help with determining the number of seats for each party and assigning them.

4 Data Design

4.1 Data Description

The system will accept a .CSV file as an input that contains party name, candidates name and the votes they have received. We will have a Candidate object for each candidate which in turn has the party name and the candidate's name stored as string data and vote counts will be stored as an integer data type. For IR, the Candidates will be stored in an array, and the system will distribute votes. Ballot data will be stored as IRBallot objects which store the names of the candidates in a string array and their party name will be saved as a character array. The Candidate objects will have an arraylist of the ballots that are going towards them.

As for OPL, as we read in candidates, we will create a Party object for each unique party. Each of these objects will have an arraylist of candidates in that party. There is another arraylist to track which candidates in that party have won a seat in the election. The Party objects will themselves be stored in an arraylist. Ballots are stored as OPLBallot objects which have both party name and the candidate's names stored as string data.

The system will also create a report of the election which keeps track of the election information. To keep track of the information the system will save information like election type which is string type data, name of the candidates as string and rank of the candidate as an integer array and location of the file as string.

4.2 Data Dictionary

Table Below lists all the classes, their attributes and the methods that will be used to develop the system. It also includes data types of different attributes, return type of methods used and the parameter accepted by the methods.

Class	Attributes	Methods
Candidate	name: string party: char totalVotes: int IRBallotList: ArrayList <IRBallot >	Candidate(name: String, party: char) getName(): string getParty(): char addVote(): void getVoteCount(): int addBallotToList(irBallot: IRBallot) : void

		getIRBallotToList(): ArrayList<IRBallot>
IRBallot	rankedCandidateNames:string[] rankedCandidateParties:char[] rankedCandidateIndices:int[] topPickIndex:int = 0 id: int	IRBallot(ballotString: string, candidateParties: char[], candidateNamesOnly: String[], id: int) : void getTopPick(): string getTopPickParty: char setNewTopPick(eliminatedCandidate[]): void getID() : int
OPLBallot	selectedCandidate: string selectedParty: char id: int	OPLBallot(ballotString: String, candidateList: String[], newID: int) : void getCandidate(): string getParty(): string getID(): int
Party	candidates:ArrayList()<Candidate> candidatesGivenSeats:ArrayList()<Candidate> partyVoteCount:int partyName: char numSeatsToDistribute:int	Party(partyName:char) addCandidate(candidate:Candidate):void getCandidateList():ArrayList()<Candidate> setPartyVoteCount(votes:int):void getPartyVoteCount():int getPartyName():char setNumSeatsToDistribute(numSeats:int): String distributeSeatsToCandidates():void getCandidatesGivenSeats():ArrayList<Candidate>

Report	electionType: String numCandidates: int numBallots: int winner: String	Report(electionType: String, numCandidates: int, numBallots: int, winner: String): void PrintReport: void generateReportFile(dateTime: String, candidateStandings: String): void

5 Component Design

The program starts with a main class that takes in user input to find and read in a file of ballots. The main will create Candidate objects using the candidate information from the input file. Each object will have a name, a party, a total number of votes that are going to them, and getter functions for these three. Furthermore, candidates will have a list of Ballot objects that are going to them, and we will be able to increase their vote count. From the file the main finds whether to run the IR election or OPL election and creates the appropriate ballot object for each ballot.

For IR, main will have an array of all of the Candidates. It will also have a local variable to keep track of the candidates who have been removed up to any given point. IRBallot objects have a list of strings which are the names of the candidates ordered from highest pick to lowest pick. Therefore, the number one pick would be at index 0. A list of those candidates' parties is in a similar array, and in a third array is the index for that candidate as they are stored in the Candidate array in main. Another variable keeps track of the current top pick starting at 0. Whenever the IR algorithm in main removes a candidate, main goes through each ballot associated with the removed candidate and updates their topPickIndex to that which represents their highest pick who is still in the running.

For OPL elections, main instead makes OPLBallots from the input file's data. These objects have a string representing the chosen candidate and a char for that candidate's party. Main will create an arraylist of Party objects for each party involved in the election. All of the Candidate objects will be added to arraylists within the appropriate Party object. The Party objects have local variables to keep track of the total number of votes each party has, the number of seats available, and the arraylist of candidates who have been assigned seats. Main has a local

variable for the quota that it calculates. A local 2D array will store the Candidate objects based on their party.

Main will use a coinToss function to resolve any relevant ties that come up. Furthermore, we will be writing to a file detailed information about the election as it proceeds; this will be the audit file. Once the end result is calculated, a Report object takes in the necessary information from main to both display the results to the terminal and to create a press release file. Such information includes the election type, the candidates, the number of votes, and of course the winners of any and all available seats (only one for IR) along with the party that they represent. For OPL, this information will be organized by party to make the results easier to read.

6 Human Interface Design

6.1 Overview of User Interface

The user will use the command line interface to run the system. They will be greeted by a welcome message and then will be prompted for a filename. If they enter in an incorrect filename they will be notified as such and prompted for a valid filename. After entering in a valid filename the system will print to the screen what algorithm is being run on the file. The system will then process the file and display the results of the election and notify the user stating that a press release file and audit file have been created.

6.2 Screen Images

Starting the Program In the command line interface

Welcome to the Ballot Counting System!

Please enter a filename: _____

Results page in command line interface

Audit file named: …. created successfully
Results:
Results….
Results….
Results….

Press release file named: … created successfully

6.3 Screen Objects and Actions

They will type a valid filename into the command line interface when prompted and then hit the enter button.

7 Requirements Matrix

N/A

8 Appendix

N/A