Team24

Voting System

Software Design Document

Name: **Michael Vang (vang2891), Matin Horri (horri031), Shivali Mukherji (mukhe105), Wenjing Jiang (jian0508)**
Section: 5801-001
Date: 03/03/2023

Table of Contents

1. Introduction

1.1 Purpose

This software design document will provide the design structure of a voting system software. Currently, it is equipped to handle a closed party election or an instant runoff election. The document will explain how the system will be implemented, data managed, components and interfaces.

The intended audience of this document is for the project developers, interested developers who wish to expand the current system, and people who wished to maintain the software. Some interfaces such as the UI may be shared to the client and other stakeholders where their input will matter on the UI.

1.2 Scope

The document contains the complete description of the voting system. As mentioned, the voting system will only be able to handle two election types: an instant runoff and a closed party listing. The voting system is a tool to process a CSV file containing collected ballots files of an election and determine an appropriate winning scenario – ie, in the case of instant runoff, a candidate is the winner. In the case of closed party, it is the seats and candidates who won the seats is the winner. Once the software process the file, conduct the appropriate election type, an audit file will be produced that detailed the intricate steps of the election.

This software hopes to provide an efficient, effective, and integrity for processing an election in a timely manner. The audit file can also be use to verify integrity of the software and election, and can be kept for record keeping.

The entirety of the program will be written in C++. This system is intended to be used for all general elections.

1.3 Overview

This document will include a process model of the instant runoff voting system, sequence diagram of the closed party listing, and the class diagram of the software. The following sections will be explained with their description.

Section 2 will provide a detailed overview of the voting system.

Section 3 will contain the process model digram, sequence diagram, and the class diagram. The process model will show how the voting system will operate from the first step of calling the instant runoff election to determining the winner. The sequence diagram will provide an overview of the closed party voting system and how it will interact with objects. Then the class diagram will provide the arching view of the system from the variables and methods.

Section 4 will describe the design choices of the system, including data structures, and why certain design were made.

Section 5 will include detailed pseudo code of the main components of the software. The pseudo code will show how each of the methods work, and how the ballots are being processed in the system.

Section 6 will include an overview of the human interface design.

Section 7 will include a table that shows which system components satisfy the use cases discussed in the SRS document.

1.4 Reference Material

To reference to the diagrams related in this document, the following github repo can be accessed at https://github.umn.edu/umn-csci-5801-01-S23/repo-Team24/tree/main/SDD. The SRS is also included in that github repo in the case of needed reference.

1.5 Definitions and Acronyms

Acronyms	Defintion
IR	Instant run off
CPL	Closed party listing

2. System Overview

The voting system in total is composed of 20 classes, 4 of them being abstract classes and 1 of them being a template class. These abstract classes are BallotProcessing, VoteBallot, VoteSystem, and SpecialCase. These 4 abstract classes will allow further election types to be developed and implemented into the software if so chooses.

BallotProcessing is used to do the heavy work of reading the CSV. It will create an according VoteBallot which contain the condensed important information to be sent off to VoteSystem. In which case, VoteSystem will handle the process of determining the election and winner. SpecialCase is to handle any type of special cases that may occur in an election. For example, when there's no majority between two candidates in an IR, the popularity special caase will be applied.

The rest of the classes are derived based on the surrounding 4 abstract class in how they will interact with the system, the main class, and the actual performance of the election.

One of the important design choice that will appear in the diagrams is how an array will map to another array. To take an example, in order to handle ballots for a certain entity whether that is a candidate or a party, a mapping of an integer array will map to another array containing so called entity.

For example, there is an array of candidates of class Candidate. Say the array candidates contain 3 candidates [John, Mike, Sally]. There will be another array of type int containing the ballots of each candidate. Say this array is called mapBallot and contains [100,20,30]. The index of mapBallot array will correspond to the same index of candidate array. In this case, John has 100 ballots, Mike has 20 ballots, and Sally has 30 ballots.

3. System Architecture

3.1 Process Model of IR

Please refer to the github repo listed below. The following diagram is called "ActivityDiagram_Team24.pdf". The repo is also referred to this document section 1.4 Reference Material.

https://github.umn.edu/umn-csci-5801-01-S23/repo-Team24/blob/main/SDD/%20ProcessModel_Team24.pdf

3.1.1 Description of IR Process Model

To start off, an election will be called. In this case, since the model describes IR, the election called will be an IR election. From there, all the prep work and busy work is done to get ready for the election: getting running candidates, handling ballots and distributing to polling places.

When the actual election occurs and people are voting, there is a automated system that will scan and record each ballot and condense it into a CSV file. This CSV file contains information about the type of election that occured, details pertaining to that election such as parties, candidates, seats, e.t.c. and then how many ballots are there in the file.

The election officials will received this CSV file and load it into the software, Voting System Software. The software will prompt the file and attempt to bring the file in. The File Processing system will be able to determine what kind of election is be held and if the Voting System Software will be able to handle it. Since the assumption made above that this is an IR, the necessary checking won't be needed.

The CSV file will be processed by an IRProcessing system which will give the needed information for IR to do its election.

The Instant Runoff Process Model initiates by receiving a list of candidates with their respective percentage of votes from the ballot system. If a clear majority is present in the ballots, the system declares the winner and displays the results via the display system. Additionally, an audit file that includes all the details of the voting process and candidates is generated using the audit system. Both of the procedure has to be done before the winner is officially declared.

However, if there isn't a clear majority, the system checks if there are only two candidates presented. If there is, the winner is determined based on the popularity of the candidates, then it will loop back to the original route.

If more than two candidates exist, the system eliminates the candidate with the lowest percentage of votes. If there are two candidates with the same lowest number of votes, it will require the Tiebreaker system to determine which candidate will be eliminated. Afterwards, the system officially determines the eliminated candidate, and start to redistribute the votes using the IRProcessing System.

The above will repeat till a winner has been declared.

3.2 Sequence Diagram of CPL

Please refer to the github repo listed below. The following diagram is called "SequenceDiagram_Team24.pdf". The repo is also referred to this document section 1.4 Reference Material.
https://github.umn.edu/umn-csci-5801-01-S23/repo-Team24/blob/main/SDD/SequenceDiagram_Team24.pdf

3.2.1 Description of Sequence Diagram for CPL

The Ballot Processing system receives the input CSV file, processees it and send with a list of parties with percentages and available seats to the CPL system. The CPL System will create a link to the audit file, and record details of processing votes and assigning seats. During the CPL voting, the program first enter the alternate course of detecting and breaking a tie.

If there is no tie between seats, the CPL system will distribute the available seats to the parties and the party system returns the candidates who have taken the seats. If there is a tie between the seats, the CPL system will request the Tiebreaker system for breaking the tie. The Tiebreaker System will return the winning party.

Then, the CPL system distributes the seats and receives the candidates who have taken the seat from the party system. After the seat allocation, the program will enter the second alternate course of assigning the remaining seats if there is an overflow.

If there is an overflow of seats, the CPL requests the lottery system for drawing a lottery which will return the winning parties for the leftover seats. Then, the CPL system distributes the seats and receives the candidates who have taken the seat from the party system.

If there is no overflow, the program will not enter the second alternate course and send the results to the display system which hold the display results of CPL. In which case, the Display system will send the display results to the actor. The audit file at this time will be labeled and produced to the actor.

3.3 UML Class Diagram

Please refer to the github repo listed below. The following diagram is called "ClassDiagram_Team24.pdf". The repo is also referred to this document section 1.4 Reference Material.
https://github.umn.edu/umn-csci-5801-01-S23/repo-Team24/blob/main/SDD/ClassDiagram_Team24.pdf

3.3.1 Brief Description of Class Diagram

It is assumed that all getters and setters functions are definined in the class diagram. As such, classes in the diagram will not show getters and setters functions. The class diagram features 20 classes with 4 of them being abstract classes and one of them being a template class. The Main class will be the entrypoint of the system and is also the terminal interface for the user.

4. Data Design

4.1 Data Description

Most of the data structures used will be arrays and arraylist. This was chosen because of ease and simplicity. A FILE object will be used to handle I/O operations such as for the audit file. One of the important data design concept was the mapping which was briefly explained in the document section 2. To reiterate the concept, there will be two arrays with the same length. Each of their index corresponds to each other in terms of defined concepts. Example given in section 2 was candidates and their ballots (candidates[], mapBallot[]).

Such examples can be found in CPLVoteSystem where there is a mapAllocatedSeats[] to parties[]. To expand, consider the following: parties[Democratic, Republican] and mapAllocatedSeats[4,5]. This would mean, Democratic has 4 allocated seats and Republican has 5 allocated seats.

4.2 Data Dictionary

There is no major system data architecture.

5. Component Design

The following is a detailed pseudo code representation of the two major election systems: IR and CPL. The pseudo code will provide in relation of the processing system and the actual voting election.

5.1 CPL System

5.1.1 CPLBallotSystem (Processing Ballots)

```
CPLBallotSystem(File *File) {
setUp() {
        Int numParty = 2nd line
        Parties[] parties = new Parties[]

        Parse the next line (for loop)
            Party1 = New Party(Democrat)
            Parties[].add(Party1);

            Party2 = New Party(Republican)
            Parties[].add(Party1);

    for(int i; i < Parties.length(); i++) {
        candidates = i-th line after party line
        Parties[i].add(candidates)
 }

    numBallot[int] = numBallot[6];
    Parties[Democrat, Republican     ]
    mapBallot[0,0 ...]
    mapAllocatedSeat[0,0 ...]
    mapRemainBallot[0,0     ]

    numSeat = 1st line after candidate lines
    numberOfBallotsLineToRead = 2nd line after
    candidate lines
}

read() {
for(int i=0; i < numberOfBallotsLineToRead; i++) {
        party_idx = find(   1    , line)
        mapBallot[party_idx] += 1 }
}
calculate() {
quota = mapBallot.sum()/numSeat
}

output() {
        CPLBallot processed = new CPLBallot(parties, mapBallot,
        mapAllocatedSeats, mapRemainBallot, numSeat)
        return processed
}
```

5.1.2 CPLVoteSystem (Conducting Election)

```
CPLVoteSystem(CPLBallot incomingBallot) {
    processedBallot = incomingBallot
    Candidate[] seatWinners
    Party[] partyWinners
    Party parties[] = processedBallot.getParties()
    int mapAllocatedSeat = processedBallot.getMapAllocatedSeat()
    int mapRemainingBallots = processBallot.getRemainingBallots()
    int status
    specialCase.add(TieBreaker(), CPLLottery())

    startElection() {
        while(status != 1){
                conductElection(processedBallot))
            notFilledParty  = new Parties[]
            int overflow = 0;
            for (int i=0; i < numParty; i++) {
                If (mapAllocatedSeat[i] < Parties[i].maxSeat) {
                    overflow += (Parties[i].maxSeat -
                mapAllocatedSeat[i])
            }
                else {
                    notFilledParty.add(Parties[i])
                }
            }

            If (overflow > 0) {
                popSystem = specialCase.get(CPLLottery())
                popSystem.setContested(notFilledParty, overflow)
                popSystem.run()
                partyWinners = popSystem.getResult()
                for (int i=0; i < partyWinners.length(); i++) {
                    mapAllocatedSeat[winners[i].idx] +=1
                    }
            }
            status = 1
            for (int i=0; i < numParty; i++) {
                candidates = parties[i].getCandidates()
                      seat = mapAllocatedSeat[i]
                        Assign seats to candidates
                      seatWinners.add(candidates[0:seat])
            }
    }




conductElection(processedBallot){
    //First allocation
    for (int i=0; i < numParty; i++) {
    mapAllocatedSeat[i] = mapBallot[i] / quota
    mapRemainBallot[i] = mapBallot[i] % quota
}

    remainSeat = numSeat - mapAllocatedSeat.sum()
    sortedRemainBallot = sort(mapRemainBallot)

    for (int i; i < remainSeat; i++) {
        If sortedRemainBallot[i] == sortedRemainBallot[i+1]:
            enter tiebreaker system
            party_idx = find(sortedRemainBallot[i], mapRemainBallot)
            mapAllocatedSeat[party_idx] += 1
    }
}
```

5.2 IR System

5.2.1 IRBallotSystem (Processing Ballots)

```
IRBallotSystem(File *File) {
setUp() {
        numCandidates = read 2nd line
        IRCandidate[] IRCandidates
        int[] mapBallot
        int[] mapPercentage
        Parse next line (for loop of i < numCandidates)
                IRCandidates[].add(IRCandidate(name, party)
            bLinesToRead = parse next line
}

read() {
for(int i=0; i < bLinesToRead; i++) {
        String line = parse line
        split = toInt(line.split(','))
        forCandidate = split.getIndexOf(1)
        Ballot balloti = new Ballot(split)
        IRCandidates[forCandidate].ballots.add(balloti)
        IRCandidates[forCandidate].numBallots++

        mapBallot[forCandidate]++
}

calculate() {
        for(int i = 0; i < mapPercentage.length(); i++) {
                mapPercentage[i] = mapBallot[i]/bLinestoRead
        }

output() {
        IRBallot processed =
            new IRBallot(IRCandidates, mapBallot,
            mapPercentage, mapBallot.sum)
        return processed




}

Redistribute(IRCandidate eliminated) {
        if(!IRCandidates.exist(eliminated)) {
                Return false
        }
        eliminateIndex = IRCandidates.getIndexOf(eliminated)
for(int i = 0; i < eliminated.numBallots; i++) {
        nBallot = eliminated.ballotList[i]
        nBallot.increaseRank()
        forCandidate = nBallot.getMap.getIndexOf(nBallot.rank)

        eliminated.numBallots--


        //This function below will redirect
        //the pointer of nBallot to its
        //newly assigned candidate
        IRCandidates[forCandidate].addBallot(nBallot);
        IRCandidates[forCandidate].numBallot++

        mapBallot[forCandidate]++
        mapBallot[eliminateIndex]--
}
}
```

5.2.2 IRVoteSystem (Conducting Election)

```
IRVoteSystem(IRProcessing process, IRBallot incomingBallot) {
        processedBallot = incomingBallot
        ballotSystem = process
        Candidate winner
        Candidate candidates[] = processedBallot.getCandidates()
        Int status
        specialCase.add(TieBreaker(), IRPopularity())

startElection() {
        while(status != 1) {
                if(conductElection(processedBallot)) {
                        status = 1;
                }

                if(candidates.length() == 2) {
                        popSystem = specialCase.get(IRPopularity())
                        popSystem.setContested(candidates[0], candidates[1])
                        popSystem.run()
                        winner = popSystem.getResult()
                        status = 1;
                }

                eliminateCandidate[] = getElimination();
                if(eliminateCandidate[].length > 1) {
                        tieSystem = specialCase.get(TieSystem())
                        tieSystem.setContested(eliminateCandidate[])
                        tieSystem.run()
                        eliminated = tieSystem.getResult()
                }

                eliminated = eliminateCandidate[0]
                if(process.redistribute(eliminated)) {
                        processedBallot = process.output()
                }
        }



 conductElection(processedBallot) {
         candidates[] = processedBallot.getCandidates()
         mapPercentage[] = processedBallot.getMapPercentage()
         for(int i = 0; i < mapPercentage.length(); i++) {
                 if(mapPercentage[i] >= 0.51) {
                         winner = candidate[i]
                         Return true
                 }
         }
         return false
 }

 getElimination() {
         lowestCandidate[]

         Simple for loop to traverse candidates[] and access
         the candidate with the min numBallots.
         Add it to the array. If theres another
         candidate with the same min numBallots, add it to the array.
         If theres someone lower than the stored minimum, reset the array.

         Return lowestCandidate[];
```

6. Human Interface Design

6.1 Overview of User Interface

1. **Welcome information**

   When the software is launched, the welcome information which includes the name, version of the software, and other information for initializing will be shown in the terminal.

2. **Input CSV file**

   After welcome information, the program prompts the user to input a ballot file in CSV format. As the program reads the file successfully, it returns “read successfully”.

3. **Processing Ballot**

   As the program starts to process the ballots, the number of candidates and the number of ballots will be displayed in the terminal. A message “processing is done” will let the user know the ballot processing has been finished.

4. **Processing Ballot**

   As the program starts to process the ballots, the number of candidates and the number of ballots will be displayed in the terminal. A message “processing is done” will let the user know the ballot processing has been finished.

5. **Election starts**

   The type of election will be displayed in the terminal. Each step and brief result will be shown in the terminal.

6. **Results display**

   The program displays the winner in the terminal and lets the user know an audit file is generated.

6.2 Screen Images

Welcome information

```
Voting System Software v 1.0.0

To view all the commands, type in "help"
The following command will return details of how to use the command "cmd": type in "man <cmd>".
To load in a CSV file for process with or without .csv, type in "read_file <filename>"
```

Input CSV file

```
Input Ballot file in CSV format: path/IRBallot.csv
…
Read successfully
```

Processing Ballot

```
Ballot processing starts
4 candidates are found
Reading 6 ballots
…
Processing is done
```

Election starts

```
IR starts
…
No majority is found
Candidates more than 2
Removing the last candidate and redistributing votes
…
XXX is removed
Next round starts
…
A majority is found
```

Results display

```
XXX wins the election
An audit file IR_results.txt is generated
```

6.3 Screen Objects and Actions

The display class will be associated with all the display to the user. Main class will handle the commands user inputted.

7. Requirements Matrix

Use cases	Description
USC_001 **Bring file into the system**	The system will receive a file from an external source. This file will be entered through a prompt or through the command line. Afterward, the software will determine if the file is a valid File. In the UML class diagram, USC_001 will be covered in the readFile() function in main.
USC_002 **Processing Ballot**	After a file has been opened, the ballot file will be processed according to the election voting system that has been determined. During the process, each candidate or party will be tallied up. USC_002 will be covered in main conductBallot function. IBallotProcessing is the abstract class for this use case, in which case the CPLProcessing and IRProcessing handles this use case respective to the main course/alternative course.
USC_003 **Running the Instant Runoff Voting System**	The program has finished processing the ballot and is determined to process its instant runoff election. The winner will be determined with vote distribution and tiebreakers if needed USC_003 will be located in the IRVoteSystem class. IRVoteSystem will also determine redistribution and any special cases if it so occurred. The main work of redistribution occurs in IRVoteProcessing. The activity model in this document can also be referred for USC_003.
USC_004 **Popularity Case for IR**	If there is no clear majority in IR between only two candidates, then popularity will win between the two. USC_004 will occur in the IRPopularity class which is apart of ISpecialCase.

USC_005 **Running the Closed Party Voting System**	The program has finished processing ballots and is determined to process them as a closed-party voting system. USC_005 will be located in CPLVoteSystem class. The sequence diagram in this document can also be refer to.
USC_006 **CPL Seat Allocation Overflow**	In closed party listing, when there is an overflow of seats allocated to a party where all their candidates have already filled their seats, the system will conduct a lottery of potential parties in parliament for the open seats. Each winner of the lottery will receive one of the seats until all seats have been filled. USC_006 will be located in CPLLottery class which is apart of ISpecialCase.
USC_007 **Determine the Tie**	Whenever there is a tie situation that has occurred between 2 things, i.e., a candidate or party, a fair coin will be tossed and determine who's the winner. USC_007 will occur in the TieBreaker class which is apart of ISpecialCase.
USC_008 **Displaying the results**	After a successful voting system has been conducted, the terminal will display the results of the election to the user. If there was a tiebreaker or popularity vote, the terminal will indicate as such and the outcome. USC_008 will occur in the Display class. IRVotingSystem and Main will also be utilizing the Display class which will handle the USC_008 courses of displaying results,
USC_009 **Producing an Audit File**	To procure information such as the election's result, distribution of ballots, and any recordings of certain voting systems. USC_009 will occur in the AuditFile class. Additionally, IRVotingSystem and CPLVotingSystem will utilize the AuditFile class to procure the following information and display the relevant information to the pertaining class.
USC_010 **Labeling Audit File**	Label a processed audit file by the type of voting election that had occurred and the date it was conducted. The audit file will then be output into the same directory as the CSV file. USC_010 will occur in the AuditFile in the labelFile() function.