

## Software Design Document (SDD) Template

Software design is a process by which the software requirements are translated into a representation of software components, interfaces, and data necessary for the implementation phase. The SDD shows how the software system will be structured to satisfy the requirements. It is the primary reference for code development and, therefore, it must contain all the information required by a programmer to write code. The SDD is performed in two stages. The first is a preliminary design in which the overall system architecture and data architecture is defined. In the second stage, i.e. the detailed design stage, more detailed data structures are defined and algorithms are developed for the defined architecture.

This template is an annotated outline for a software design document adapted from the IEEE Recommended Practice for Software Design Descriptions. The IEEE Recommended Practice for Software Design Descriptions have been reduced in order to simplify this assignment while still retaining the main components and providing a general idea of a project definition report. For your own information, please refer to IEEE Std 10161998<sup>1</sup> for the full IEEE Recommended Practice for Software Design Descriptions.

<sup>1</sup> <http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieeeSDD.pdf>

Team 4

## **Voting System Software**

### Software Design Document

Name (s):

Bryan Yen Sheng Lee, lee03627

Cedric Tan Yee Shuen, tan00205

Sherryl Ooi Shi Tyng, ooi00004

Lab Section: Section 002

Workstation: GitHub

Date: (03/02/2023)

## **TABLE OF CONTENTS**

<b>1. INTRODUCTION</b>	<b>4</b>
1.1 Purpose	4
1.2 Scope	4
1.3 Overview	5
1.4 Reference Material	5
1.5 Glossary	6
<b>2. 2. SYSTEM OVERVIEW</b>	<b>7</b>
<b>3. SYSTEM ARCHITECTURE</b>	<b>8</b>
3.1 Architectural Design	8
3.2 Decomposition Description	9
3.3 Design Rationale	11
<b>4. DATA DESIGN</b>	<b>12</b>
4.1 Data Description	12
4.2 Data Dictionary	12
<b>5. COMPONENT DESIGN</b>	<b>14</b>
<b>6. DIAGRAMS</b>	<b>23</b>
6.1 UML Class Diagram For The Entire System	23
6.2 UML Activity Diagram For Instant Runoff (IR) Voting	25
6.3 Sequence Diagram For Closed Party List (CPL) Voting	28
<b>7. USER INTERFACE DESIGN</b>	<b>30</b>
7.1 Overview of User Interface	30
7.2 Screen Images	31
7.3 Screen Objects and Actions	35
<b>8. REQUIREMENTS MATRIX</b>	<b>36</b>

# **1. INTRODUCTION**

## **1.1 Purpose**

This Software Design Document provides the design details of Voting System Version 1.0 that is capable of performing the following two types of voting system: (1) Instant Runoff Voting system and (2) Party List Voting using Closed Party List.

The expected audience is the election authorities including state and local election officials, who are from different jurisdictions that used voting systems in their areas.

## **1.2 Scope**

This document contains a complete description of the design of Voting System Version 1.0

The basic architecture is a java program from a client server side. The basic program will be in Java GUI. The election authorities will have full access to the system, which could start the election process, view the process, and print the results of voting and to run a fair coin toss if the majority votes is a tie. The election authorities will have no access to make changes, including edit or delete the votes, and could not pause the system once the voting process has started.

## 1.3 Overview

The remaining chapters and their contents are listed below.

**Section 2** is a Development Diagram that shows the physical locations where the system actually exists. This allows a clear explanation of where each design entity will reside. Each part will work in unison to accomplish each requested task.

**Section 3** is the Architectural Design that specifies the design entities that collaborate to perform all the functions included in the system. Each of these entities has an Abstract description concerning the services that it provides to the rest of the system. In run, each design entity is expanded into a set of lower-level design operations that collaborate to perform its services.

**Section 4** concerns the Data Design, including the Data Description and the Data Dictionary.

**Section 5** contains the Component Design with the Use Case Realizations. Each Use Case stated in the SRS Document can be traced by the given design objects.

**Section 6** introduces the Diagrams, including the UML Diagram for the entire system, UML diagram for IR voting and Sequence diagram for CPL.

**Section 7** discusses the User Interface Design, with the screen images, objects and actions. It also shows how it can be created with maximum user efficiency and ease of use.

**Section 8** covers the Requirements Matrix.

## 1.4 Reference Material

Fowler, M. and Scott, K. (1999, August 18). UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language.

<https://pja.mykhi.org/6sem/MAS/books/Addison%20Wesley%20-%20UML%20Distilled,%202nd%20Edition.pdf>

Gama, P. (2014, May 8). Software Design Description (SDD) sample.

[https://www.slideshare.net/peny\\_mg/sdd-software-des-sample](https://www.slideshare.net/peny_mg/sdd-software-des-sample)

IEEE Template for Software Design Document. (n.d.). Retrieved from

<http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieeeSDD.pdf>

Lee, B., Ooi, S., and Tan, C. (2023, February 16). Software Requirements Specification for Voting System Software.

[https://docs.google.com/document/d/1Y5ulm0F4HCb\\_4FsIooUbUKqt-xsEY9Pc/edit#](https://docs.google.com/document/d/1Y5ulm0F4HCb_4FsIooUbUKqt-xsEY9Pc/edit#)

## 1.5 Glossary

**Ballot:** a means of registering a vote, or a device to cast votes in an election.

**CPL:** Closed Party List ballot. It describes the variant of party-list systems where the voters can effectively only vote for political parties as a whole.

**CSE Labs:** Lab machines at the College of Science and Engineering, University of Minnesota. Details can be found here <https://cse.umn.edu/cseit/classrooms-labs>

**CSV:** Comma Separated files, a text file that has a specific format that allows data to be saved in a table structured format.

**Developers:** A team of developers with a software engineering background, developing the software.

**Dialog box:** Graphical components that are usually used to display errors give information to the user.

**Election officials:** Authorizers who run an election, did not belong to any parties.

**Git/Github:** Open source distributed version control system designed to handle everything, including any set of computer files.

**GUI:** A graphical user interface that incorporates icons, buttons, pull-down menus, and a mouse.

**IDE:** An integrated development environment that provides comprehensive facilities to programmers for software development.

**IR:** Instant Runoff. A type of ranked preferential voting method. Used the majority voting rule in a single-winner election.

**Java:** A high-level, class-based, and object-oriented programming language.

**Linux:** An operating system, version of Unix

**macOS:** An operating system, version of Unix

**Residents:** The voter who is on the enumeration list and who has been given the means to vote.

**Testers:** A team of developers with a software engineering background; who create test cases to test every scenario.

**U of MN:** The University of Minnesota.

**Windows:** An operating system

## 2. SYSTEM OVERVIEW

The voting system is capable of performing two types of voting — Instant Runoff (IR) voting and Closed Party List (CPL) Voting. The primary functions of the voting system include allowing election officials to view election information, view election results, and break ties in order to determine the winner in an election. It reads the csv file, performs the necessary checks and processing, displays the results, and generates the audit file for the election officials. Within the context of this project, an efficient voting system can run 100,000 ballots in under 4 minutes. In order to achieve an optimal workflow, the voting system comprises four modules that work together to perform the necessary tasks:

1. **Read File module:** This module is responsible for checking if Comma Separated Files (CSV) are formatted correctly and located in the correct directory. It then reads the file.
2. **Check File module:** This module is responsible for checking if the CSV file and the election results adheres to the necessary requirements. There are three subsystems implemented throughout the voting system:
  - **(Subsystem #1) File Format Checks:** Checks to see if the file structure is not modified and if there are no write-in candidates
  - **(Subsystem #2) Instant Runoff Checks:** Checks Instant Runoff (IR) voting results to see if there's a majority and if candidates are ranked in a particular order
  - **(Subsystem #3) Check for Ties:** Checks to see if there's a tie in election results between two or more parties/candidates. Toss a coin if there's one to determine the winner.
3. **Display Results module:** This module is responsible for displaying the election results depending on the voting type. The displayed election results include the type of election, number of seats, winning party, and winning candidate(s).
4. **Generate Audit File module:** This module is responsible for generating the audit file with election information at the time for both election types. The information includes the type of voting, number of candidates, candidates, number of ballots, calculations, number of votes per candidate, progression of election. Overall, the objective of the audit is to be able to replicate the election itself to ensure transparency in the voting process.

### 3. SYSTEM ARCHITECTURE

#### 3.1 Architectural Design

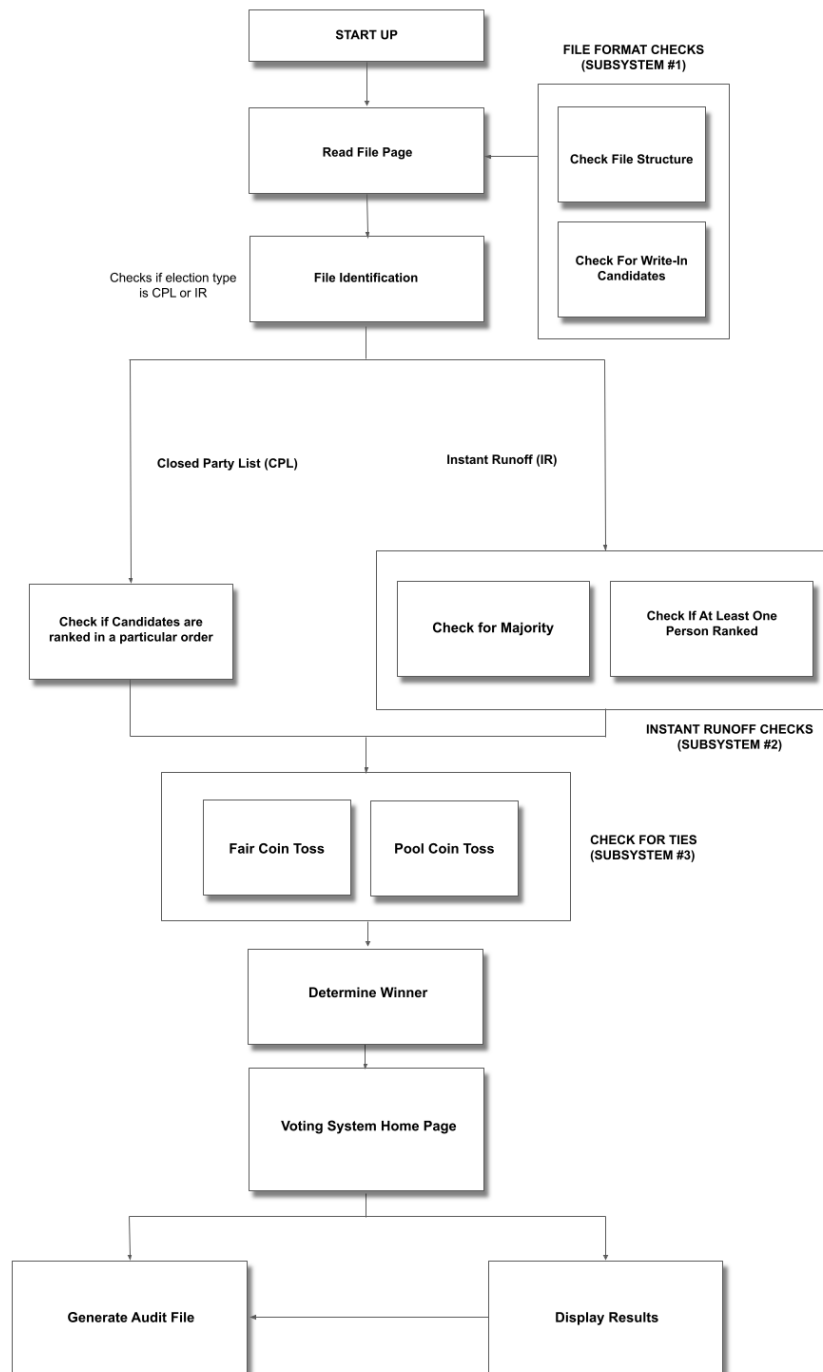


Figure 3.1: The Architectural Design of The Voting System



### 3.2 Decomposition Description

The functional description of the voting system is illustrated through two diagrams — a top level data flow diagram (DFD) and a structural decomposition diagram. These diagrams provides a decomposition of the subsystems in the architectural design of the voting system:

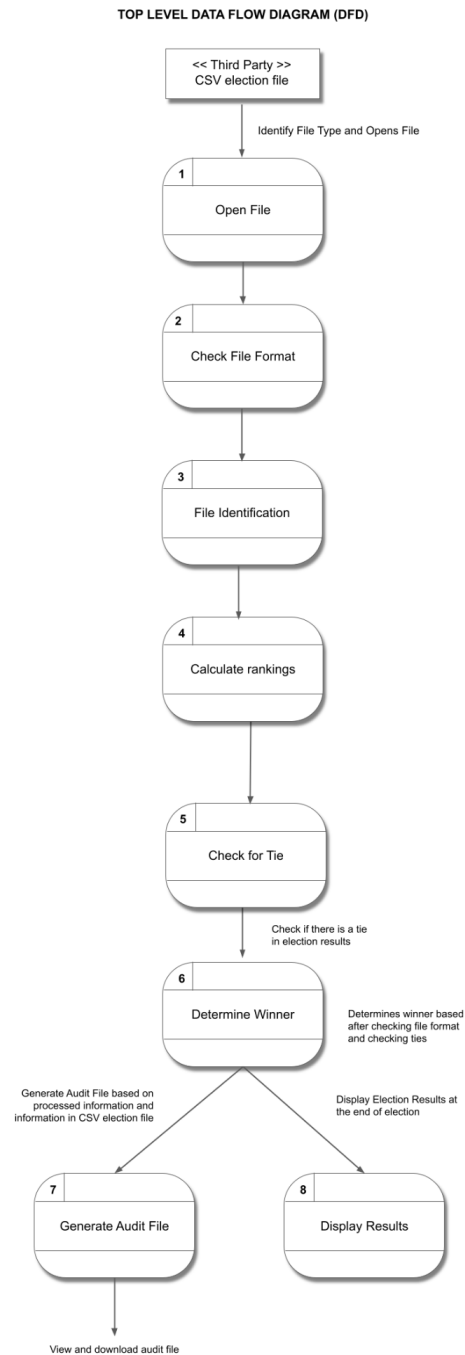


Figure 3.2: Top Level Data Flow Diagram (DFD) of the Voting System

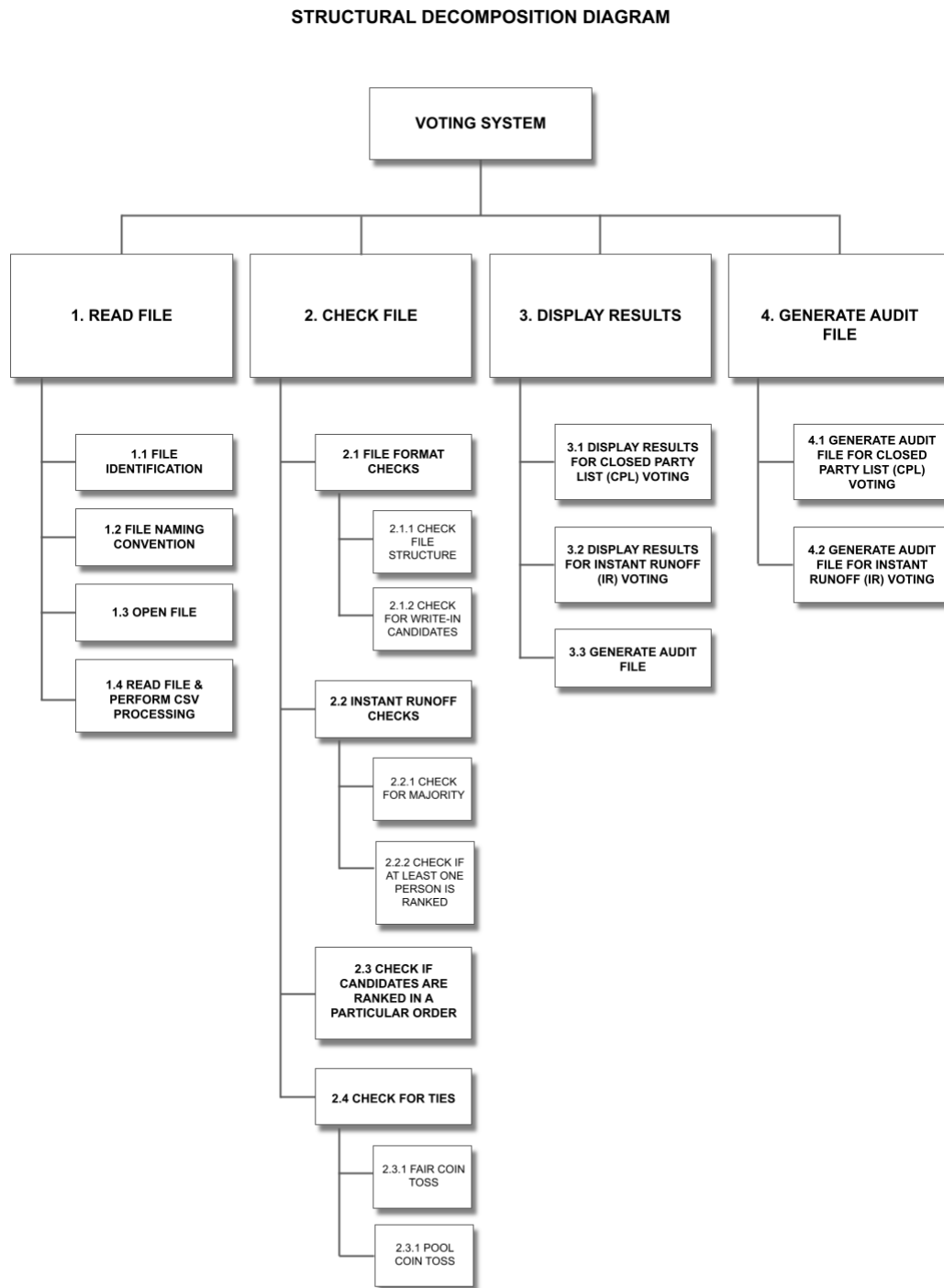


Figure 3.3: Structural Decomposition Diagram of Voting System

### 3.3 Design Rationale

The design rationale behind the architectural design described in 3.1 is process-based as it heavily emphasizes on the workflow of the voting system. A process viewpoint illustrates how the voting system, at runtime, is composed of interacting processes and is useful for making judgements about non-functional system characteristics such as performance. In the voting system, files have to be checked before they could be processed and the information processed must adhere to voting requirements for different election types. Given that there were multiple functional requirements implemented to check the Comma Separated Files (CSV) and voting requirements to fulfill, these checks were compartmentalized under series of different subsystems whose functions differ from each other — file format checks, Instant Runoff (IR) checks, and checks for tie. The implementation of subsystems in the voting system leads to better workflow and better-managed software development in the long run.

## 4. DATA DESIGN

### 4.1 Data Description

The data is imported from a Comma Separated Files (CSV), the file includes the descriptions of the votes and ballots. It will be stored as an array to ease the calculations of the ballots. The results will be exported to a CSV file for easier visualization of the results. The data fields names, descriptions, types, and sizes are given in the following table.

### 4.2 Data Dictionary

**Table 1. Data Field Names, Descriptions, Types and Sizes**

Attribute Name	Attribute Description	Attribute Type	Attribute Size
candidateName*#	Stores the name of a candidate	String	100
candidatesName[*] #	Stores the name of the candidates in an array	An Array of String	500
candidatesRanking*#	Stores the rank of the candidates in a particular order	Tuples with Int and String	20
electionDate*#	Stores the Election Date	String	50
electionType*#	Stores the Type of Election (IR or CPL)	String	4
filename*#	Stores the name of the File	String	100
numOfBallots*#	Stores the number of ballots of the election	Int	100, 000
numOfCandidates* #	Stores the number of candidates	Int	500
numOfParties*#	Stores the number of parties	Int	100
numOfSeats*#	Stores the number of seats for a party	Int	500
numOfVotesPerCandidate*#	Stores the number of votes for a candidate	Int	50, 000, 000

numOfVotesPerParty*#	Stores the number of votes for a party	Int	50, 000, 000
partiesName[]*#	Store party names in an array	An Array of String	100
partyName*#	Stores the name of a party	String	100
results*#	Stores the result of the winner which contains the data of WinningCandidate, TotalNumOfVotesPerCandidate, WinningParty, and TotalNumOfVotesPerParty	String	1000
scopeOfElection*#	Stores the scope of the election (1 - State, 2 - National, or 3 - Local)	Int	3
totalBallots[][]*#	Stores the number of ballots for each candidate according to rank	A 2D Array of Integer, Integer	50, 000, 000
totalNumOfBallots*#	Stores the total number of ballots	Int	50, 000, 000
totalNumOfSeats*#	Stores the total winning number of seats	Int	500
totalNumOfVotesPerCandidate*#	Stores the total number of votes per candidate	Int	50, 000, 000
totalWinningNumOfVotesPerParty*#	Stores the total winning number of votes per party	Int	50, 000, 000
winningCandidate*#	Stores the name of the candidate that wins the election	String	100
winningParty*#	Stores the name of the party that wins the election	String	100

\* - required fields

# - visible or not visible (determined by election officials)

^ - never visible to anyone other than the election officials

## 5. COMPONENT DESIGN

```

openFile(String filename)

    import the file

    IF CheckFileFormat(filename) == FALSE THEN

        return fail

    ELSE

        readfile(filename)

```

The above pseudocode represents the operation on how the system will open a file. The system will pass in the filename into the openFile function. When the openFile(String filename) function is called, the system will import the file into the system. If the file type is not a CSV file or the file does not exist in the system, the system will print an error message for the user and return a failure value. Otherwise, if the system successfully imported the file and the file exists in the system, the system will call the readfile(String filename) function and start the reading process.

```

readFile(String filename)

    electionType = read first line

    IF electionType is IR THEN

        IF there is not at least one candidate is ranked THEN

            print error message

            return fail

        numOfCandidates = read second line

        candidatesName[] = read third line with comma separated

        totalNumOfBallots = read fourth line

        FOR i in totalNumOfBallots DO

            totalBallots[i][] = read line, separated by comma

        checkMajority()

        calculateRanking(electionType, numOfCandidates, candidatesName[],

            totalNumOfBallots, totalBallots[][]))

```

```

ELSE IF electionType is CPL THEN

    IF the candidates are not ranked in a particular order THEN

        print error message

        return fail

    numOfParties = read second line

    partiesName[] = read third line with comma separated

    FOR i in numOfParties DO

        partiesCandidates[i] = read line

    totalNumOfBallots = read line

    FOR i in totalNumOfBallots DO

        totalBallots[i][] = read line

    calculateRanking(electionType, numOfParties, partiesName[],

                    totalNumOfBallots, totalBallots[][]))

ELSE

    print error message

    return fail

```

The above pseudocode represents the operation on how the system will read a file. The system will pass in the filename into the readFile function. When the readFile(String filename) function is called, the system will start to read the CSV file starting from the first line. The system will start to identify the file by reading in the first line of the CSV file to determine the type of election where it is Instant Runoff (IR) voting or Closed Party List (CPL) voting.

If the type of election is IR, the system will perform the Instant Runoff (IR) checking by checking if the ballots will have at least one candidate ranked as their top choice. If there is not at least one candidate is ranked, the system will send an error message to the election officials and return to the menu page. If the system has at least one candidate ranked as their top choice, the system will proceed to read in all the election information which includes the number of candidates, name of the candidates, total number of ballots, and how many votes a candidate had. Then, the system will check for majority by calling the checkMajority() function. The system will then calculate the ranking of each candidate by calling the calculateRanking() function.

If the type of election is CPL, the system will perform the Closed Party List (CPL) checking by checking whether the candidates are ranked in a particular order. If the candidates are not ranked in a particular order, the system will return an error message to the election officials and failure value is returned. If the candidates are ranked in a particular order, the system will proceed by reading in all the election information which includes the number of parties, name of the parties, and the candidates from the order of each party are ranked in order. The system will then calculate the ranking of each party by calling the `calculateRanking()` function.

```
checkMajority()
```

```

    IF there is one candidate wins more than 50% of the votes THEN
        checkForTie()

        return

    WHILE no majority DO

        find the candidates with the fewest votes

        checkForTie()

        transfer the ballots of the defeated candidate to whichever of the

            remaining candidates they marked as their number two

                choice

        update the candidate and the candidate's ballots information

    return

```

The above pseudocode represents the operation on how the system will check the majority for the Instant Runoff (IR) voting. The system will pass in the filename into the `checkMajority` function. If the type of election is IR, the system will call the `checkMajority` function. The system will perform the Instant Runoff (IR) checking by checking if the ballots will have at least one candidate ranked as their top choice as well as checking the majority. If there is not at least one candidate is ranked, the system will send an error message to the election officials and return to the menu page. The system will check for the majority by checking the conditions as follows. If a candidate receives over 50% of the first choice votes, he or she is declared elected. If no candidate receives a majority, then the candidate with the fewest votes is eliminated. The system will check for ties for both conditions. If either one of the two conditions occurs a tie, the system



will either perform a fair coin toss when there is a tie between two candidates or perform a pool coin toss when there is a tie between three or more candidates. The system will prompt the election officials before performing any of the coin tosses. If the election officials choose to run the fair coin toss or the pool coin toss, the system will randomly select a candidate either to eliminate or to declare as elected based on the situation occurring in an unbiased mode. Then, the ballots of supporters of this defeated candidate are then transferred to whichever of the remaining candidates they marked as their number two choice. The ballots and candidates information are updated in the system. After this transfer, the votes are then recounted to see if any candidate now receives a majority of the vote. The process of eliminating the lowest candidate and transferring their votes continues until one candidate receives a majority of the continuing votes and wins the election.

```
calculateRanking(String electionType, Int numOfCandidates/Parties, String
candidates/PartiesName[], Int totalNumOfBallots, Int totalBallots[][])
```

```
    IF electionType is IR THEN

        FOR i in totalNumOfBallots DO

            FOR j in numOfCandidates DO

                INT k = 1

                FOR k in numOfCandidates DO

                    IF totalBallots[i][j] == k THEN

                        candidates[i][k] ++

                    max of candidates[numOfCandidates][1] == firstRank

            ELSE IF electionType is CPL THEN

                FOR i in totalNumOfBallots DO

                    FOR j in numOfParties DO

                        IF totalBallots[i][j] == 1 THEN

                            totalWinningNumOfVotesPerParty[j] ++

                        max of totalWinningNumOfVotesPerParty[numOfParties] == firstRank
```

The above pseudocode represents the operation on how the system will calculate the rankings of the candidates or parties. The system will pass in the election type, number of candidates or parties, the candidates or parties name, total number of ballots and the ballots itself.

The function will first run an if else statement to check the election type, if the election type is IR, the function will run calculation for IR elections, on the contrary, if the election type is CPL, the function will run calculation for CPL elections.

If the election type is IR, the function will run three nested for loops, the first for loops will run a loop to run through all ballots that read from the file; the second for loops will run a loop to run through the number of candidates, which is usually lower than 10. In the for loop, we will initialize a temporary integer variable. After that, we will run the third for loop, which checks the ballots, the candidates have each array to store their ballots. The function will increase each vote to their respective candidates. Lastly, the function will run a maximum of the winning votes of each candidate, and the first candidate will be the first rank of the election.

If the election type is CPL, the function will run two nested for loops, the first for loops will run a loop to run through all ballots that read from the file. The second for loops will run a loop to run through the number of parties, and run the calculation inside the for loops. If the ballots vote for the party, the party winning votes will increase by 1. Lastly, the function will run a maximum of the winning votes, the first party will be the first rank of the election.

```
checkFileFormat(String filename)

    IF file type is not csv THEN

        print error message

        return FALSE

    IF file does not exist THEN

        print error message

        return FALSE

    IF file structure has been modified outside the program THEN

        print error message

        return FALSE

    IF there are write-in candidates detected in the file THEN

        print error message

        return FALSE

    return TRUE
```

The above pseudocode represents the operation on how the system will perform the checking operation for check file format. The system will pass in the filename into the checkFileFormat function. When the checkFileFormat function is called, the system will check whether the file type is in .csv format. If the file type being imported into the system is not a .csv file, the system will send an error message to the election officials and return a False boolean, indicating an error. If the system fails to locate the file in the system, an error message indicating that the file does not exist within the system will be sent to the election officials and False is returned. If the file structure is being modified outside the program, the system will print an error message and return False to the election officials. If there are write-in candidates detected in the CSV file, the system will print an error message to the election officials and False is returned. Otherwise, True is returned to the election officials.

```

checkForTie(totalNumofWinningBallots[][]))

    SORT totalNumOfWinningBallots[][] Descending to firstRank, secondRank,
        .....
    maxNumber = 0

    FOR i in numOfcandidates DO

        set the max = totalNumOfWinningBallots[i][0]

        FOR current = totalNumOfWinningBallots[i][j]

            IF current >= max THEN

                max = current

                maxNumber ++

    IF firstRank == secondRank THEN

        IF maxNumber > 2 THEN

            boolean YesOrNo = prompt user

            return poolCoinToss(YesOrNo, maxNumber)

        YesOrNo = prompt user

        return fairCoinToss(YesOrNo)

    return firstRank

```

The above pseudocode represents the operation on how the system will check for a tie. The system will pass in the total number of winning ballots into the checkForTie function. When the checkForTie function is called, the function will first sort the total number of winning ballots descendingly, so the first item in the array will be the maximum number of votes. However, with the sorting algorithm, we can't make sure that there is no tie in the winning ballots. Hence, we are using a for loop to check how many maximum numbers are in the number of winning ballots. If the first rank equals to second ranks, it means that there are at least a tie in the election, if the maximum number is greater than 2, it means that there are a tie with 3 candidates or parties in the election, hence, we will run a pool coin toss, with the user answer of running a coin toss or not, and the number of maximum. The function will run a fair coin toss after prompting the user's option on whether to run a coin toss or not. Lastly, if there is no tie, the function will just return the first ranking of the total number of winning ballots.

```
poolCoinToss(bool YesOrNo, int maxNumber)

    IF YesOrNo == No

        return

    ELSE

        Return rand.nextInt(maxNumber)
```

The above pseudocode represents the operation on how the system will run a pool coin toss. The system will pass in a boolean variable which prompts the user to check if the user wants to run a pool coin toss during tie into the pool coin toss function. If the user indicates to run a pool coin toss, the function will run a random function to choose a number out of the pool, the candidates or the parties with that respective number will determine as the winner.

```
fairCoinToss(bool YesOrNo)

    IF YesOrNo == NO

        return

    ELSE

        return rand.nextInt(2)
```

The above pseudocode represents the operation on how the system will run a fair coin toss. The system will pass in a boolean variable which prompts the user to check if the user wants to run a fair coin toss during tie into the fair coin toss function. If the user indicates to run a fair coin toss,

the function will run a random function to choose either 1 or 2 out of the pool, the candidates or the parties with that respective number will determine as the winner.

```
determineWinner()

    return results of checkForTie(totalBallots[][])
```

The above pseudocode represents the operation on how the system will display the winner by taking the results from the checkForTie function. There is nothing being passed into the function. When the determineWinner() function is called, the system will return the results of the checkForTie function.

```
generateAuditFile()

    create new file

    write electionType, new line

    IF electionType == IR THEN

        write numOfCandidates

        FOR i in numOfCandidates

            write rank, candidatesName, totalNumOfWinningBallots, new line

    ELSE IF electionType == CPL THEN

        write numOfParties

        FOR i in numOfParties

            write rank, partiesName, partiesCandidates,
            totalNumOfWinningBallots, new line

    DownloadOrNo = prompt user //to download file or not

    IF DownloadOrNo == Yes THEN

        write file
```

The above pseudocode represents the operation on how the system will generate an audit file for both Instant Runoff (IR) voting and Closed Party List (CPL) voting. The system will pass in no parameters into the generateAuditFile() function. When the generateAuditFile() function is

called, the function will first create a new file. After the file is created, the function will write the type of election into the audit file. If the type of election is Instant Runoff (IR) voting, the function will write in the number of candidates followed by each candidate's rank, name, and total number of winning ballots. Otherwise, if the type of election is Closed Party List (CPL) voting, the function will write in the number of parties followed by each party's rank, name, and total number of winning ballots. Moreover, the function will prompt the user whether to download the file where DownloadOrNo is a boolean. If the DownloadOrNo is equal to Yes, then the system will download the file.

```
displayResults()

    print electionType, new line

    IF electionType == IR THEN

        print numOfCandidates

        FOR i in numOfCandidates

            print rank, candidatesName, totalNumOfWinningBallots, new
line

    ELSE IF electionType == CPL THEN

        print numOfParties

        FOR i in numOfParties

            print rank, partiesName, partiesCandidates,
totalNumOfWinningBallots, new line
```

The above pseudocode represents the operation on how the system will display the result in a given format. The system will pass in no parameters into the displayResults function. When the displayResults() function is called, the system will print the type of election followed by the number of candidates, as well as each candidate's rank, name, and total number of winning ballots.

## 6. DIAGRAMS

### 6.1 UML Class Diagram For The Entire System

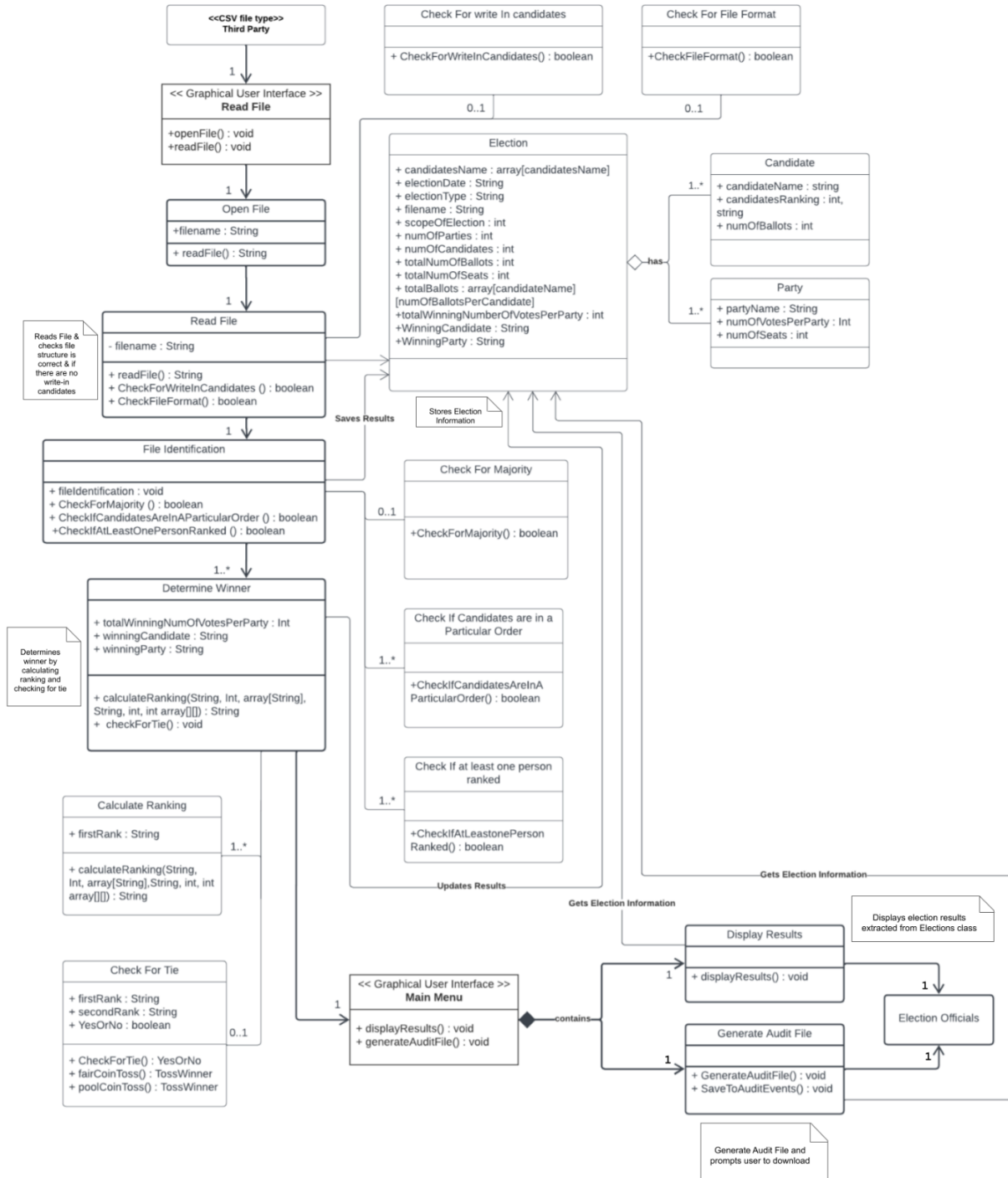


Figure 6.1: UML Class Diagram for Voting System

The diagram above is the UML Class Diagram for the voting system. It follows a linear process starting from when the voting system receives the Comma Separated Files (CSV). A read file menu pops up which prompts the user to open and read file. Here the election information are extracted from the CSV file and stored in the elections class. The election class has both candidates and parties which stores specific information about each candidate and party. When `readFile()` is called, it calls two functions — `CheckForWriteInCandidates()` and `CheckForFileFormat()` — to examine whether the file structure is appropriate and if there are no write-in candidates.

Then, the `FileIdentification()` function identifies whether the election type is Instant Runoff (IR) voting or Closed Party List (CPL) voting. If it is an IR voting type, two functions — `CheckForMajority()` and `CheckIfAtleastOnePersonRanked()` — are called to perform the necessary checks.

Else, if it is a CPL voting type, the `CheckIfCandidatesAreRankedInAParticularOrder()` function is called to check through the information. This information is then updated in the election class.

The next step is to determine the winner by calculating the ranking and checking if there's a tie by calling both `calculateRanking()` and `CheckForTie()` functions. If a tie occurs between two parties/candidates, the `fairCoinToss()` function is called. Else, if a tie occurs between more than two parties/candidates, the `poolCoinToss()` function is called. The `calculateRanking()` function returns a tuple containing both candidateNames and their rankings. The winner's information is then updated in the election class.

After that, the main menu pops up which prompts the user to display results or generate an audit file by calling the `DisplayResults()` and `GenerateAuditFile()` functions. For the `DisplayResults()` function, election results are extracted from the election class and displayed using a dialog box. On the other hand, for the `GenerateAuditFile()` function, all election information is also extracted from the election class and condensed into an audit file to be exported. Here the election officials will interact with the final results generated from `DisplayResults()` and `GenerateAuditFile()`.



## 6.2 UML Activity Diagram For Instant Runoff (IR) Voting

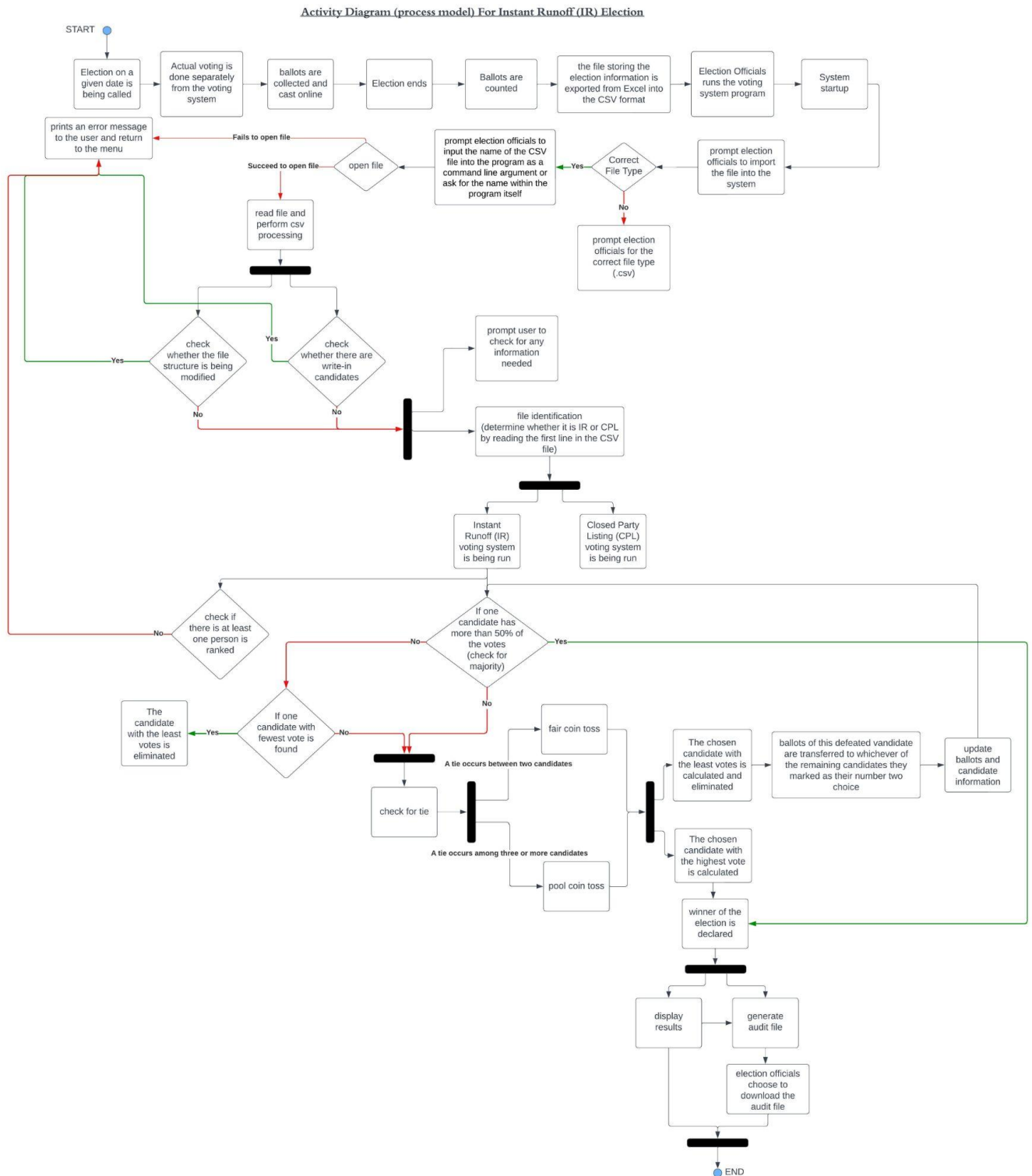


Figure 6.2: UML Activity Diagram For Instant Runoff (IR) Voting

The diagram above is the UML Activity Diagram For Instant Runoff (IR) Voting. The process model starts with an election being called on a given date. The actual voting will be done separately from the voting system. Ballots will be collected and cast online. After the election ends, the ballots collected will be counted. The file storing the election information will be exported from Excel into CSV format. It will be a comma separated values (CSV) file where each row is separated by a newline. All the preprocessing of the file will be done before it is being imported into the voting system program.

Next, the election officials run the voting system program and turn the start button on the panel to the “on” position. The system is activated in a normal mode and the welcome screen is displayed. First, the system will prompt the election officials to import the election file that they want into the voting system program. Then, the system will check whether the file being imported is a valid file type, which is .csv file. If the file being imported is not a .csv file, the system will return an error message and prompt the election officials to import the correct file type. If the file type is valid, the system will proceed by prompting election officials to input the name of the CSV file into the program as a command line argument or ask for the name within the program itself. The system will then open the file for reading purposes. If the system fails to open the file, an error message will appear and the election officials will be returned to the menu. Otherwise, if the system succeeds in opening the file, the system will start to read the file and perform CSV processing. Next, the system will perform two checking operations simultaneously, which are to check whether the file structure is being modified outside the program as well as to check whether there are write-in candidates in the file. The system will only proceed to the next stage if and only if the file structure is not modified outside of the program and there are no write-in candidates in the file. Otherwise, the system will send an error message to the election officials and return to the menu page.

After the checking process is done, the election officials can be prompted for any needed information that cannot be extracted from the file. At the same time, the system will start to identify the file by reading in the first line of the CSV file to determine the type of election where it is Instant Runoff (IR) voting or Closed Party List (CPL) voting. Since it is an activity diagram for the Instant Runoff (IR) voting, when the type of election is IR, the system will perform the Instant Runoff (IR) checking by checking if the ballots will have at least one candidate ranked as their top choice as well as checking the majority. If there is not at least one candidate is ranked, the system will send an error message to the election officials and return to the menu page. The system will check for the majority by checking the conditions as follows. If a candidate receives over 50% of the first choice votes, he or she is declared elected. If no candidate receives a majority, then the candidate with the fewest votes is eliminated. The system will check for ties for both conditions. If either one of the two conditions occurs a tie, the system will either perform a fair coin toss when there is a tie between two candidates or perform a pool coin toss when there is a tie between three or more candidates. The system will prompt the

election officials before performing any of the coin tosses. If the election officials choose to run the fair coin toss or the pool coin toss, the system will randomly select a candidate either to eliminate or to declare as elected based on the situation occurring in an unbiased mode. Then, the ballots of supporters of this defeated candidate are then transferred to whichever of the remaining candidates they marked as their number two choice. The ballots and candidates information are updated in the system. After this transfer, the votes are then recounted to see if any candidate now receives a majority of the vote. The process of eliminating the lowest candidate and transferring their votes continues until one candidate receives a majority of the continuing votes and wins the election. After a winner is elected, the winning candidate and the information will be displayed on the screen. The information of the election includes the type of voting, number of candidates, candidates, number of ballots, calculations, how many votes a candidate had, etc. The system will list the winner(s) and show how the election progressed so that the audit file could replicate the election itself. After the audit file is generated, the election officials could choose to download the file in a pdf form for future use. Finally, the activity diagram ends.

### 6.3 Sequence Diagram For Closed Party List (CPL) Voting

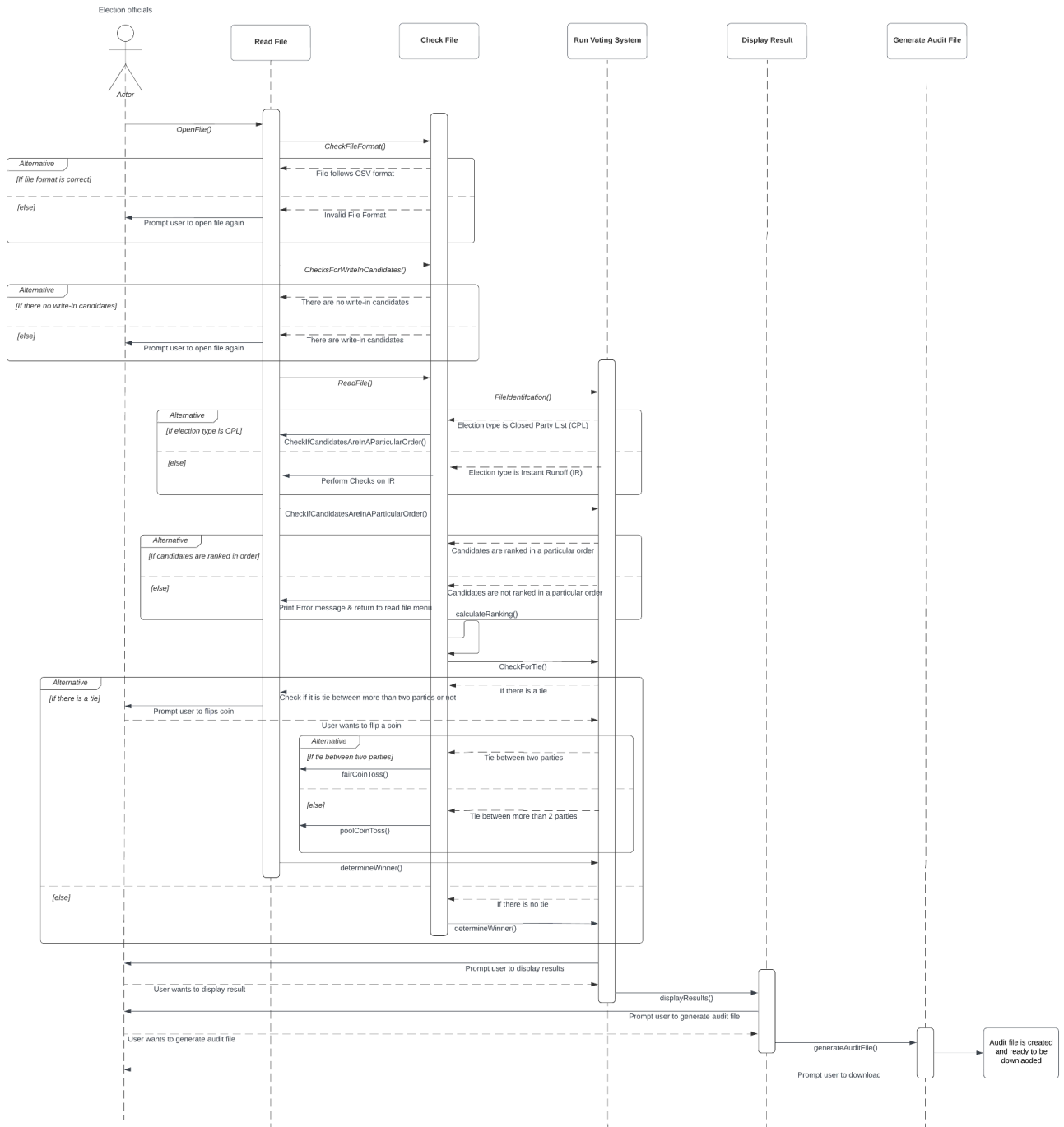


Figure 6.3: Sequence Diagram For Closed Party List (CPL) Voting

The actors, which are the election officials, will run the election through the main function. The main function will run the `openFile()` function to open the file provided by the actor.

First, the system will prompt the election officials to import the election file that they want into the voting system program. Then, the system will check whether the file being imported is a valid file type, which is .csv file. If the file being imported is not a .csv file, the system will return an error message and prompt the election officials to import the correct file type. If the file type is valid, the system will proceed by prompting election officials to input the name of the CSV file into the program as a command line argument or ask for the name within the program itself. The system will then open the file for reading purposes. If the system fails to open the file, an error message will appear and the election officials will be returned to the menu. Otherwise, if the system succeeds in opening the file, the system will start to read the file and perform CSV processing. Next, the system will perform two checking operations simultaneously, which are to check whether the file structure is being modified outside the program as well as to check whether there are write-in candidates in the file. The system will only proceed to the next stage if and only if the file structure is not modified outside of the program and there are no write-in candidates in the file. Otherwise, the system will send an error message to the election officials and return to the menu page.

After the checking process is done, the election officials can be prompted for any needed information that cannot be extracted from the file. At the same time, the system will start to identify the file by reading in the first line of the CSV file to determine the type of election where it is Instant Runoff (IR) voting or Closed Party List (CPL) voting. Since it is the sequence diagram for the Closed Party List (CPL) voting, when the type of election is CPL, the system will perform the Closed Party List (CPL) checking by checking if the ballots will have at least one party ranked as their top choice as well as checking the majority. If there is not at least one party is ranked, the system will send an error message to the election officials and return to the menu page. The system will check for the majority by checking the conditions as follows. If a party receives over 50% of the first choice votes, he or she is declared elected. If no party receives a majority, then the party with the fewest votes is eliminated. The system will check for ties for both conditions. If either one of the two conditions occurs a tie, the system will either perform a fair coin toss when there is a tie between two parties or perform a pool coin toss when there is a tie between three or more parties. The system will prompt the election officials before performing any of the coin tosses. If the election officials choose to run the fair coin toss or the pool coin toss, the system will randomly select a party either to eliminate or to declare as elected based on the situation occurring in an unbiased mode. Then, the ballots of supporters of this defeated party are then transferred to whichever of the remaining parties they marked as their number two choice. The ballots and party information are updated in the system. After this transfer, the votes are then recounted to see if any party now receives a majority of the vote. The process of eliminating the lowest party and transferring their votes continues until one party receives a majority of the continuing votes and wins the election. After a winner is elected, the

winning party and the information will be displayed on the screen. The information of the election includes the type of voting, number of parties, parties, number of ballots, calculations, how many votes a candidate had, etc. The system will list the winner(s) and show how the election progressed so that the audit file could replicate the election itself. After the audit file is generated, the election officials could choose to download the file in a pdf form for future use.

## 7. USER INTERFACE DESIGN

### 7.1 Overview of User Interface

*Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.*

The voting system provides a user interface for election officials to access election information. It must be designed to meet the needs of the end users and support the overall goals of the software system. The voting system will primarily be used by election officials to obtain audit files, view election-related information, and break ties in order to determine the winner. The system reads the csv file, displays the results, and generates the audit file. Thus, in order for the election officials to access these functions, the user interface of the voting system must be intuitive and easy to use. Figure 6.1 below illustrates the relationship between the interfaces:

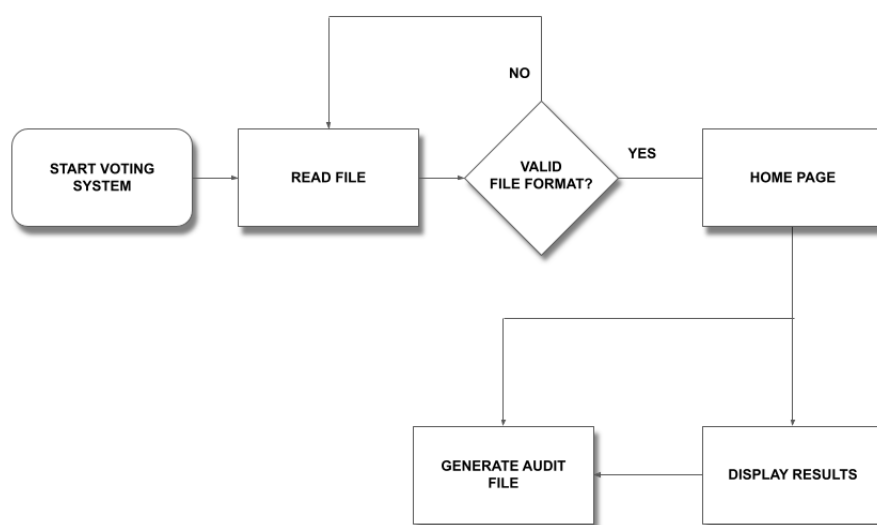
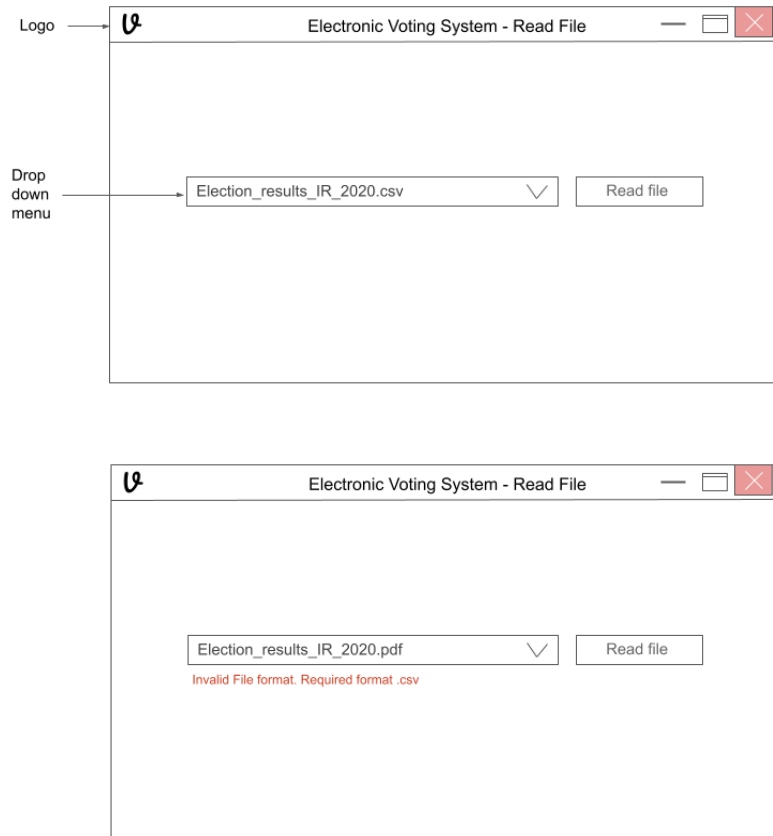


Figure 7.1: User Flow Diagram

## 7.2 Screen Images

### 7.2.1 Read File Menu

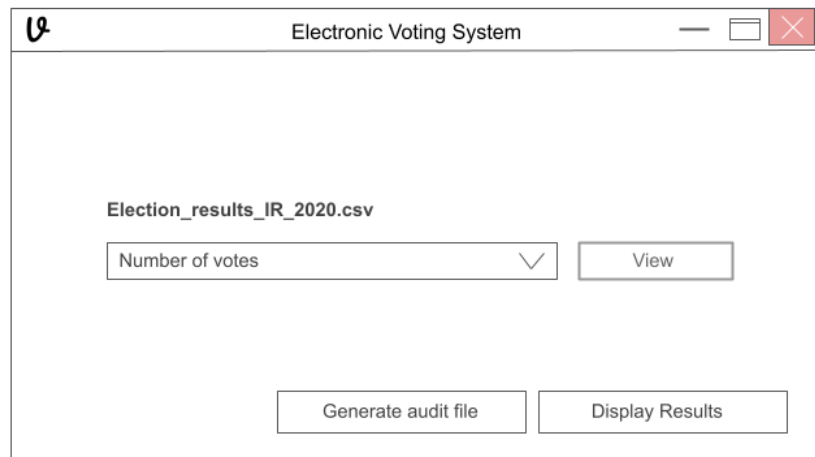


*Figure 7.2: Read File Menu*

In the read file Interface, the screen layout is designed to be simple. Users can interact with the drop-down list to look for files within the directory and click on the read file button when they have selected the file they are looking for.

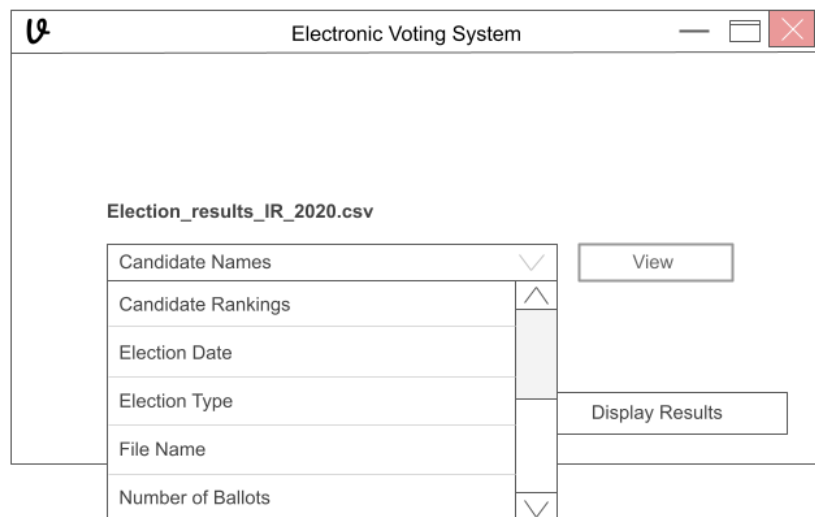
### 7.2.2 Main Menu

When the file is read successfully, the main menu interface pops up. Here the users can select to view specific information, display results, or generate audit file. This is illustrated in figure 6.3 below:



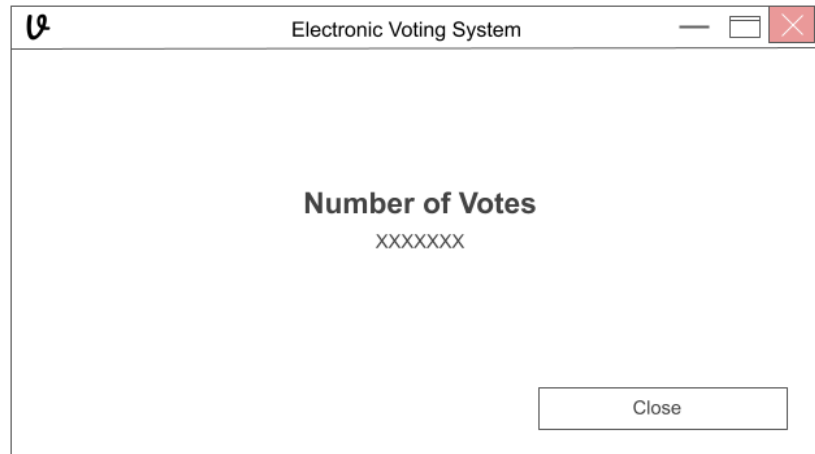
*Figure 7.3: Main Menu*

When specific information is selected through the drop down menu, users can select the view button and a dialog box with the selected information appears. This is illustrated in both figure 7.4 and 7.5 below:



*Figure 7.4: Main Menu with drop down menu*

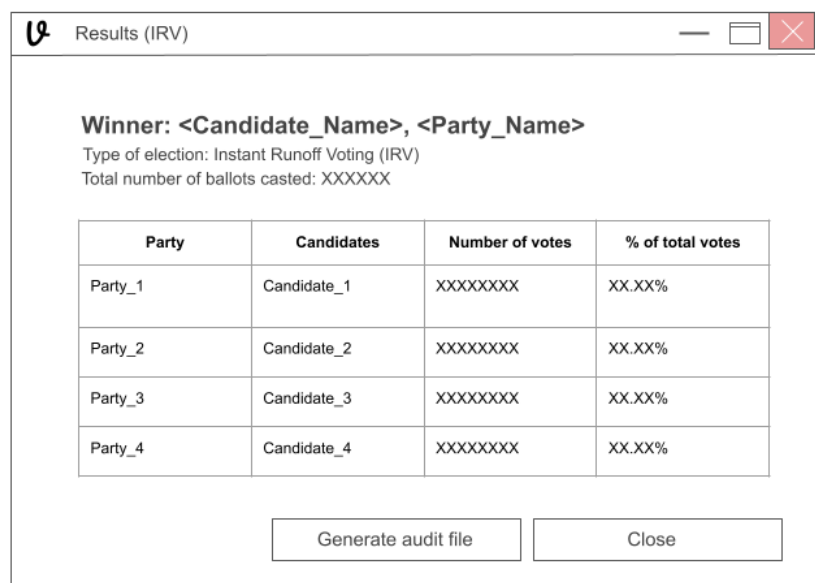





*Figure 7.5: Information Dialog Box*

### 7.2.3 Display Results Dialog Box

When the user selects the 'display results' button, a message dialog box pops up depending on the type of election. The key difference between the Instant Runoff Voting dialog box and the Closed Party List dialog box is that all candidates are listed in the Closed Party List voting box. After displaying the results, users can either close the dialog box or generate an audit file. The diagrams below illustrate the two types of dialog boxes when displaying results:



*Figure 7.6: Display results for Instant Runoff (IR) Voting*

 Results (CPL)

**Winner: <Party\_Name>**

**Winning Candidates (in order): (1) Candidate\_1, (2) Candidate\_2, (3) Candidate\_3  
(4) Candidate\_4, (5) Candidate\_5**

Type of election: Closed Party List

Total number of ballots casted: XXXXXX

Party	Candidates	Number of votes	% of total votes
Party_1	1. Candidate_1	xxxxxx	xx.xx%
	2. Candidate_2	xxxxxx	xx.xx%
	3. Candidate_3	xxxxxx	xx.xx%
	4. Candidate_4	xxxxxx	xx.xx%
	5. Candidate_5	xxxxxx	xx.xx%
Party_2	1. Candidate_1	xxxxxx	xx.xx%
	2. Candidate_2	xxxxxx	xx.xx%
	3. Candidate_3	xxxxxx	xx.xx%
	4. Candidate_4	xxxxxx	xx.xx%
	5. Candidate_5	xxxxxx	xx.xx%
Party_3	1. Candidate_1	xxxxxx	xx.xx%
	2. Candidate_2	xxxxxx	xx.xx%
	3. Candidate_3	xxxxxx	xx.xx%
	4. Candidate_4	xxxxxx	xx.xx%
	5. Candidate_5	xxxxxx	xx.xx%
Party_4	1. Candidate_1	xxxxxx	xx.xx%
	2. Candidate_2	xxxxxx	xx.xx%
	3. Candidate_3	xxxxxx	xx.xx%
	4. Candidate_4	xxxxxx	xx.xx%
	5. Candidate_5	xxxxxx	xx.xx%

Generate audit file

Close

*Figure 7.7: Display results for Closed Party List (CPL) voting*

## 7.3 Screen Objects and Actions

### 7.3.1: Read File Menu

**Drop-down Menu for files in directory:** The drop down menu provides a list of files in the directory when the user interacts with by clicking the downward arrow.

**Read File Button:** The read file button reads the selected file from the drop down menu.

### 7.3.2: Main Menu

**Drop-down Menu to view specific information:** The drop down menu provides a list of information extracted from the csv file which includes:

- Candidate names
- Election dates
- Election type
- File name
- Number of ballots
- Number of seats
- Number of votes per candidate
- Number of votes per party
- Party Name
- Result
- Scope of election
- Total number of ballots
- Total number of seats
- Total number of votes per candidate
- Total number of votes per party
- Winning candidates
- Winning party

**View Button:** The view button reads the selected information from the drop-down menu.

**[Information Dialog Box] Close Button:** The close button exits the dialog box and returns to the main menu.

**Generate Audit File Button:** The generate audit file button compiles the given and processed information into an audit file to be exported.

**Display Results Button:** The display results button displays the election results based on the type of election — Instant Runoff voting or Closed Party List voting.

### 7.3.3: Display Results Dialog Box:

**Generate Audit File Button:** The generate audit file button compiles the given and processed information into an audit file to be exported.

**Close Button:** The close button exits the dialog box and returns to the main menu.

## 8. REQUIREMENTS MATRIX

ID	Use cases name	Descriptions	System components
UC_001	System Startup	This use case describes the actions performed by the system during start-up and initialization phase.	main()
UC_002	File Identification	The use case describes how the system recognizes the type of the file.	checkFileFormat()
UC_003	File Naming Convention	The use case describes how the name of the file is being named by the system.	checkFileFormat()
UC_004	Open File	The use case describes how the system opens a file which contains ballots that were cast online.	openFile()
UC_005	Read the file and perform CSV processing	This use case describes the actions performed by the system by reading in a file and performing CSV processing.	readFile()
UC_006	Prompt user for information	When the program is run, the user can be prompted for any needed information that cannot be extracted from the file.	main()
UC_007	Create an audit file for Instant Runoff	The use case describes how an audit file is produced for Instant Runoff Voting System. The audit file contains election information at the time.	generateAuditFile()
UC_008	Create an audit file for Closed Party List	The use case describes how an audit file is produced for Closed	generateAufitFile()

		Party List Voting System. The audit file contains election information at the time.	
UC_009	Document have at least one person ranked	This use case ensures that the ballot has at least one of the candidates ranked as their top choice.	determineWinner()
UC_010	Run two types of elections	The use case ensures that the system is capable of handling both types of elections which are Instant Runoff Voting (Plurality/Majority Preferential Voting) and Closed Party List Voting (Proportional representation).	calculateRanking()
UC_011	Ensures file structure does not change	This use case ensures that there will be no changes in the file structure outside of the program since the election files will come in the predetermined format.	Not satisfied
UC_012	Fair Coin Toss	Upon a two-way tie in voting the winner shall be determined by a fair coin flip.	fairCoinToss()
UC_013	Pool Coin Toss	The use case describes how the system generates a common unbiased random bit to determine the winner.	poolCoinToss()
UC_014	Check for Majority	This use case checks if there is not a clear majority in an Instant Runoff Voting, then the popularity wins after all votes have been handed out.	checkForMajority() determineWinner()
UC_015	Check if candidates are listed in a particular order	This use case checks if the candidates are listed in a particular order in the Closed Party List Voting.	Not satisfied
UC_016	Disallow write-in candidates	This use case ensures that write-in candidates are not allowed in the system for both types of election.	Not satisfied

UC_017	Display Election Results	This use case illustrates the actions performed by the system by showing the outcomes which include the winner(s) and information about a specified election on the screen.	displayResults()
--------	--------------------------	---	------------------