Team 25

Vote Counting System

Software Design Document

Name(s):

Josh Trimble, trim0039, Team 25

Myat Mo, mo000007, Team25

Caden Potapenko, potap009, Team25

Roman Woolery, woole022, Team25

Date: 02/28/2021

Table of Contents

1. INTRODUCTION

1.1 Purpose

This software design document details the architecture and software design of vote-counting software used to run IR and OPL style elections. The expected audience of this document includes programmers, election officials, and testers.

1.2 Scope

The vote-counting software detailed in this SDD is coded in C++ and designed to implement ballot processing for IR and OPL on existing election data organized into one file. The software will simulate or run a real ballot count on this file and return its results along with a file for election auditing and a file to share with the media.

1.3 Overview

Section 1, Introduction: Covers the purpose of the SDD and the voting software it details. It summarizes the overall document as seen in this overview section (1.3) and lists references and definitions used throughout the document for readers to understand the SDD's language and formatting.

Section 2, System Overview: Gives a general overview of the functionality, context, and design of the voting software.

Section 3, System Architecture: This shows the voting software's architecture, given a high-level overview of how the software is broken down and flow charts of how it uses functions. Section 3 also includes a UML diagram of the classes in this software and UML activity diagrams to visualize the IR and OPL voting flow. Section3 also elaborates on the reasoning behind these design decisions.

Section 4, Data Design: Details how attributes of the voting data file used in the voting system software are divided and stored in the vote-counting software. It lists the use of voting data and its variable type. The data dictionary lists functions used to interface with the election data and their respective parameters.

Section 5, Component Design: Lists all functions included in the project's code, along with pseudocode and or explanations of their intended functionality. This previews how the use of each function in the design of the voting software.

Section 6, Human Interface Design: Details the UI of the voting software. This section explains how to use and interpret the software while running an election. Users should read this document for an in-depth understanding of operating the software during elections or testing.

Section 7, Requirements Matrix: Lists the software's functions/classes as they correspond to the overall system's use cases given the SRS and shows their fulfillment in the software's design.

1.4 Reference Material

This document is based on the IEEE template for SDD documents.

1.5 Definitions and Acronyms

Term	**Definition**
IEEE	Institute of Electrical and Electronics Engineers.
SDD	Software Design Document
SRS	Software Requirements Specification
UML	Unified Modeling Language (diagram for visualizing a program's organization)
IR	Instant Runoff
OPL	Open Party Listing

2. SYSTEM OVERVIEW

This project is designing a vote-counting software for either IR or OPL style elections. This voting data is input by a single CSV file, and its data managed as described in section 4. The software will be able to run through various election scenarios. These include dealing with ties by coin toss odds, managing run-off elections while eliminating candidates, reallocating votes by ranked preference, or settling elections by popular vote when narrowed down. The software produces election results, an audit file, and a media file after utilizing the provided data. Users can find further details on this system's flow and its modular program structure in section 3.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

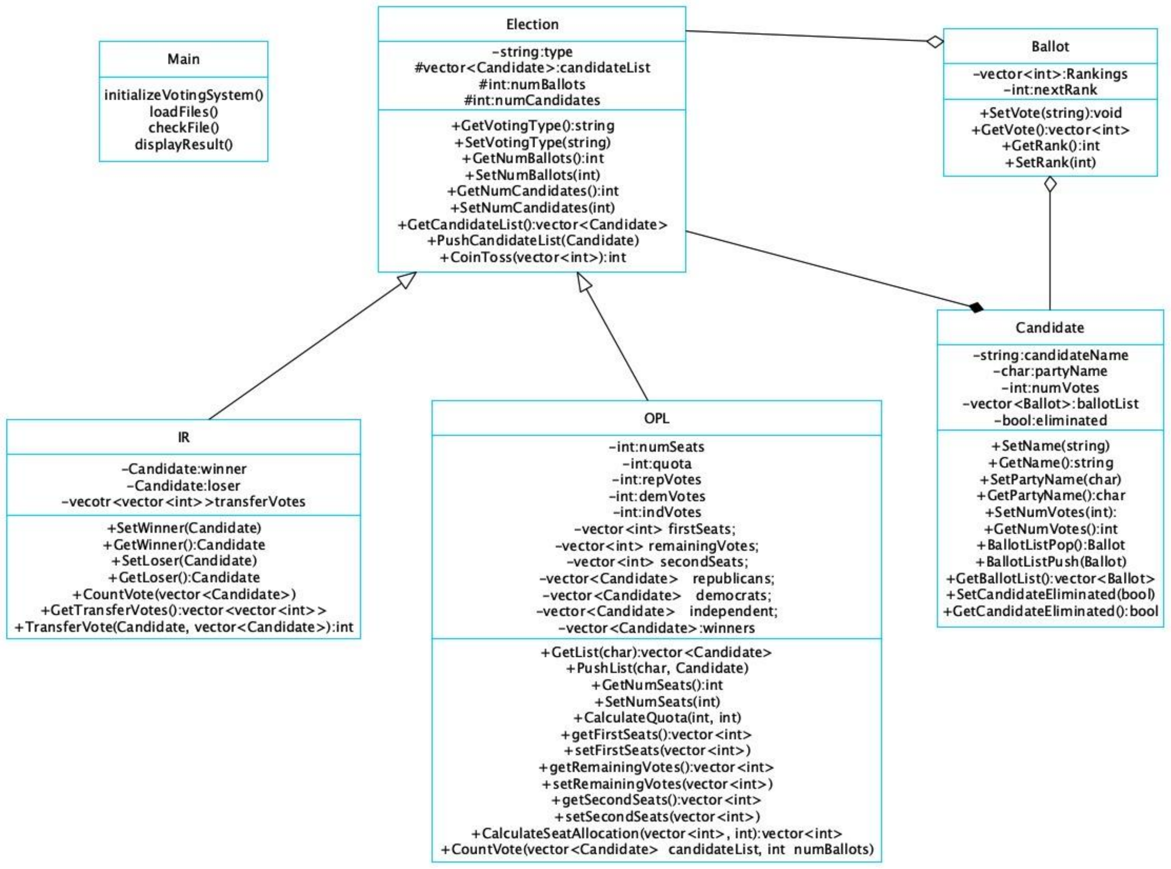

Main
initializeVotingSystem()
loadFiles()
checkFile()
displayResult()
Election
-string:type
#vector<Candidate>:candidateList
#int:numBallots
#int:numCandidates
+GetVotingType():string
+SetVotingType(string)
+GetNumBallots():int
+SetNumBallots(int)
+GetNumCandidates():int
+SetNumCandidates(int)
+GetCandidateList():vector<Candidate>
+PushCandidateList(Candidate)
+CoinToss(vector<int>):int
Ballot
-vector<int>:Rankings
-int:nextRank
+SetVote(string):void
+GetVote():vector<int>
+GetRank():int
+SetRank(int)
Candidate
-string:candidateName
-char:partyName
-int:numVotes
-vector<Ballot>:ballotList
-bool:eliminated
+SetName(string)
+GetName():string
+SetPartyName(char)
+GetPartyName():char
+SetNumVotes(int):
+GetNumVotes():int
+BallotListPop():Ballot
+BallotListPush(Ballot)
+GetBallotList():vector<Ballot>
+SetCandidateEliminated(bool)
+GetCandidateEliminated():bool
IR
-Candidate:winner
-Candidate:loser
-vecotr<vector<int>>transferVotes
+SetWinner(Candidate)
+GetWinner():Candidate
+SetLoser(Candidate)
+GetLoser():Candidate
+CountVote(vector<Candidate>)
+GetTransferVotes():vector<vector<int>>
+TransferVote(Candidate, vector<Candidate>):int
OPL
-int:numSeats
-int:quota
-int:repVotes
-int:demVotes
-int:indVotes
-vector<int> firstSeats;
-vector<int> remainingVotes;
-vector<int> secondSeats;
-vector<Candidate> republicans;
-vector<Candidate> democrats;
-vector<Candidate> independent;
-vector<Candidate>:winners
+GetList(char):vector<Candidate>
+PushList(char, Candidate)
+GetNumSeats():int
+SetNumSeats(int)
+CalculateQuota(int, int)
+getFirstSeats():vector<int>
+setFirstSeats(vector<int>)
+getRemainingVotes():vector<int>
+setRemainingVotes(vector<int>)
+getSecondSeats():vector<int>
+setSecondSeats(vector<int>)
+CalculateSeatAllocation(vector<int>, int):vector<int>
+CountVote(vector<Candidate> candidateList, int numBallots)

Figure 3.1 UML Class Diagram for Vote Counting System

This diagram shows the subsystems of the Vote Count System as Classes. Each box represents the Class's attributes, methods, and scope. Relations between them are presented through the connections between each Class box.

The functionality of the system:

Subsystems:

- **Main**
  - **Roles and responsibilities**
    - Serves as the interface between the user and the system
    - Loads and verifies the file
    - Initializes the voting system based on the first line of the file and the user input for voting type
    - Creates an audit file that updates with election progress
    - Displays the result of the election and saves it to a media text file

- **Election Class**
  - **Roles and responsibilities**
    - Serves as the general class that keeps track of the candidate list, number of ballots, number of candidates, and type of election.
    - Implements coin toss for any ties
  - **The relation between each subsystem**
    - IR and OPL inherit from it.
    - Has an instance of the Ballot class
    - Contains an instance of the Candidate Class
- **IR Class**
  - **Roles and responsibilities**
    - This class keeps track of the winner, current loser, and distribution of votes for an IR election.
    - Used to count votes in an IR election
    - Used to redistribute the votes for candidates
  - **The relation between each subsystem**
    - Inherits from Election Class to utilize coin toss and candidate list functionality to keep track of candidates and their ballots
    - Sister to OPL but will never run in the same election with that class.
    - Contains instance of candidate class which has an instance of ballot class
- **OPL Class**
  - **Roles and responsibilities**
    - Used to keep track of total votes and lists of each party's candidates, quota, and the total number of seats
    - Used to allocate seats in an OPL election
    - Calculates the quota and uses it to set allocation
  - **The relation between each subsystem**
    - Inherits from Election Class to utilize coin toss and candidate list functionality
    - Sister to IR but will never run in the same election with that class.
    - It contains an instance of the candidate class but does not utilize the ballot storage functionality, and it will only use the candidate class to sort the parties based on initial pass-through ballots.
- **Ballot Class**
  - **Roles and responsibilities**
    - Used to store the vote rankings for each Ballot in the election
  - **The relation between each subsystem**

        - Used by Candidate Class to keep lists of all the votes for a particular candidate in an IR election
        - Used by Election Class to instantiate all Ballots as a Ballot object
- **Candidate Class**
    - **Roles and responsibilities**
        - Keeps track of each candidates name, party, list of ballots (only for IR), and number of votes
        - Provides the functionality for adding and removing Ballots to and from a candidates instance of the Ballot list
    - **The relation between each subsystem**
        - Has an instance of the Ballot class used in IR voting
        - Used by the Election Class.
        - Used by IR Class and OPL Class

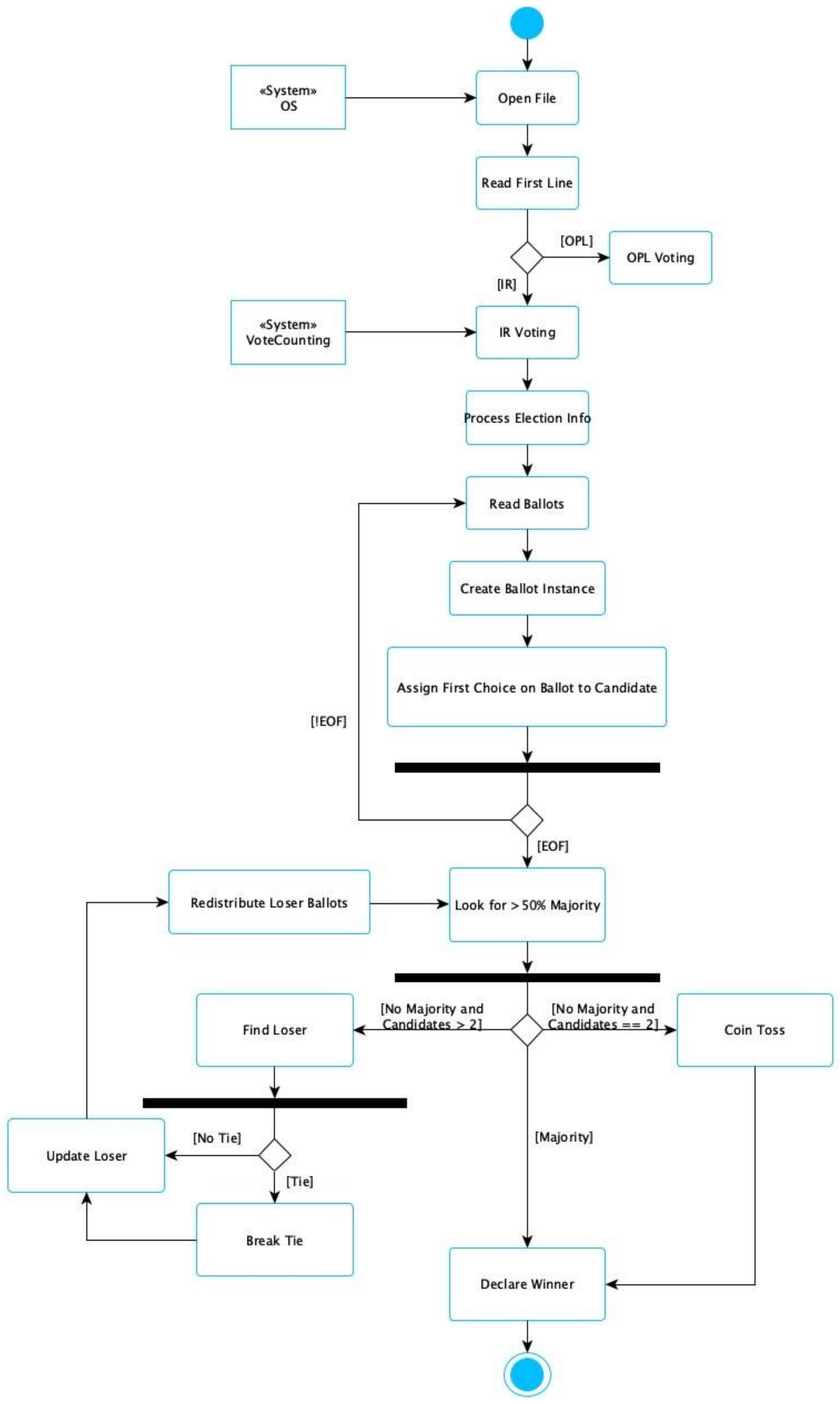


Figure 3.2 UML Activity Diagram for IR Voting

This diagram shows the flow of activities in the Vote Counting System from when an IR election closes to a declared winner.

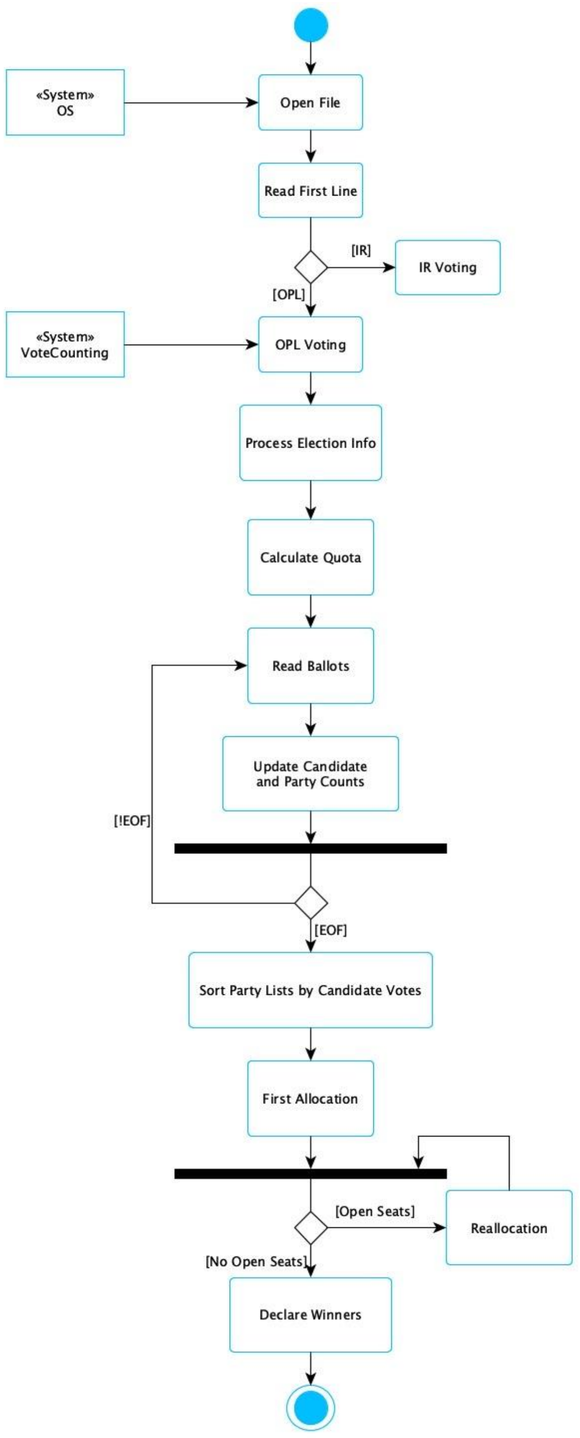


Figure 3.3 UML Activity Diagram for OPL Voting

This diagram shows the flow of activities in the Vote Counting System from when an OPL election closes to declared winners.

3.2 Decomposition Description

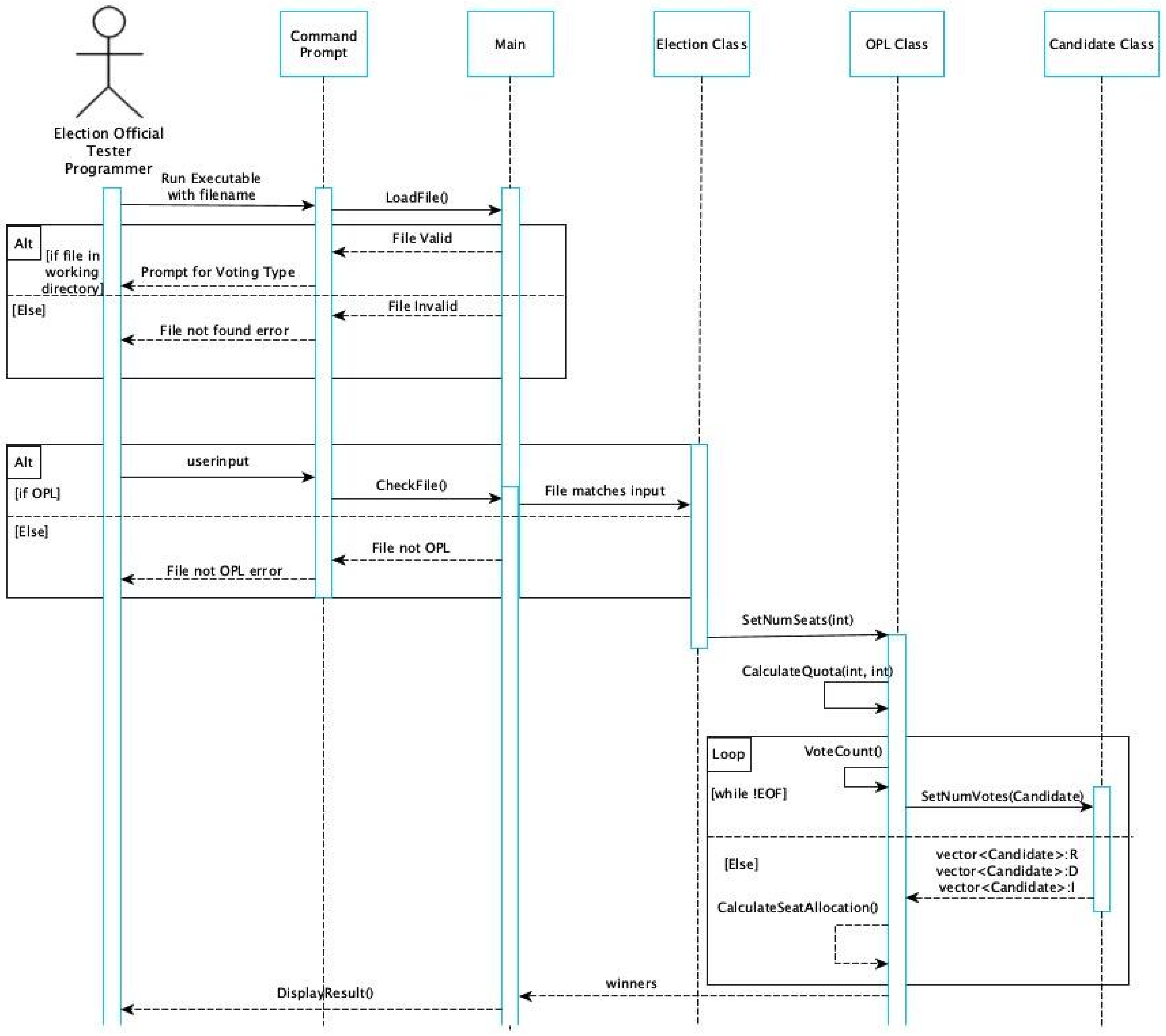


Figure 3.4 UML Sequence Diagram for OPL Voting

This diagram represents the OPL vote-counting sequence from the program's execution to a return of the results. Specifically, the transfer of information between the user, objects, and classes involved in the system.

For IR Voting:

- The sequence will work similar to OPL but with changes specific to the IR requirements. The user interaction with the system is the same but with IR CSV file and IR as user input for election type.
- Instead of using the OPL class, IR will be, and no quota calculation made.

- The Ballot Class will be added, and that is where each Ballot will be sent upon being read from the CSV.
    - Ballots will also be assigned to the #1 ranking the voter had. If not present, will continue to the next rank, etc.
- After all Ballots have been processed and an EOF read, the IR class will determine a 50% majority. If there is a majority, then the declared winner will be sent back to the user.
    - Else, a forward message of no majority will be sent to the candidate class, where each candidate's ballots will be redistributed.
    - Once there is a majority hit in the loop, the winner and election info will be sent to the user.
    - If only two candidates remain and no majority, then coin toss to determine the winner.

3.3 Design Rationale

Rationale:

- IR and OPL are both elections that share some similarities, so we chose to inherit them from a more general election class.
- Candidates have several attributes that should be grouped, so we chose to make a candidate a type of class used by both elections.
- Ballots will only really be needed in IR elections because it requires the ballot's storage to be redistributed as part of the instant runoff. This class will be used by the candidate class and the IR election class but not the OPL class because there is no need for storage in that case.

Critical Issues and Trade-offs:

- One issue was how to do a coin toss for more than two candidates.
    - We decided to create a list of the tied candidates and generate 1,000 random numbers from 0 to (#candidates - 1), and the 1,000th random number will be the candidate selected and returned.
    - **Other Architectures Considered:**
        - Creating an algorithm that does a tournament-style coin toss to determine the winner was considered.
- Another issue was where and how to store the ballots.
    - Because IR will be using the ballots for each candidate, The candidate class will have an instance of a list of Ballots assigned to them throughout the distributions and redistributions. This decision makes it possible to keep track of the ballot and the ballot's current choice for redistribution upon an elimination.
    - **Other Architectures Considered:**
        - Instead of keeping track of a list of int lists, each ballot will have a different selection rank based on which candidate has been eliminated and their choices.

4. DATA DESIGN

4.1 Data Description

The input is a CSV file where the data for the election is stored. Assuming it passes the validity test, which is the responsibility of the user to pre-format. Attributes are listed in the table below.

Table 4.1 Data Field Type and Sizes

Attribute Name	Attribute Type	Attribute Size
votingType	string	30
candidateList	vector<Candidate>	10
numBallots	int	500000
numCandidates	int	30
rankings	vector<int>	10
nextRank	int	30
candidateName	string	40
party	char	2
numVotes	int	500000
ballotList	vector<Ballot>	500000
eliminated	bool	1
winner	Candidate	2
loser	Candidate	2

transferVotes	vector<vector<int>	numTransfers*sizeof(int)
numSeats	int	500000
quota	int	500000
repVotes	int	500000
demVotes	int	500000
indVotes	int	500000
firstSeats	vector<int>	10
remainingVotes	vector<int>	10
secondSeats	vector<int>	10
republicans	vector<Candidate>	30
democrats	vector<Candidate>	30
independent	vector<Candidate>	30
winners	vector<Candidate>	30

4.2 Data Dictionary

Table 4.2 Function Data Dictionary

Function	**Parameter**
ballotListPop	
ballotListPush	Ballot

calculateQuota	int, int
calculateSeatAllocation	vector<int>, int
checkFile	string
displayResult	string
pushCandidateList	Candidate
calculateSeatAllocation	vector<int>, int
calculateQuota	int, int
coinToss	vector<int>
countVote	vector<Candidate>
countVote	vector<Candidate>, int
getBallotList	vector<Ballot>
getCandidateList	vector<Candidate>
getDemVotes	int
getFirstSeats	vector<int>
getIndVotes	int
getList	char, vector<Candidate>
getLoser	Candidate
getName	string
getNumBallots	int
getNumSeats	int
getNumvotes	int
getParty	char

getQuota	int
getRemainingVotes	vector<int>
getSecondSeats	vector<int>
repVotes	int
getTransferVotes	vector<vector<int>>
getVotingType	string
getVote	vector<int>
getWinner	Candidate
initializeVotingSystem	
isEliminated	bool
loadFile	
pushList	char, Candidate
setDemVotes	int
setFirstSeats	vector<int>
setIndVotes	int
setLoser	Candidate
setName	string
setNumBallots	int
setNumCandidates	int
setNumSeats	int
setNumvotes	int
setParty	char

setRemainingVotes	vector<int>
setRepVotes	int
setSecondSeats	vector<int>
setVotingType	string
setVote	string
setWinner	Candidate
transferVote	Candidate, vector<Candidate>

5. COMPONENT DESIGN

In this section, we take a closer look at what each component does in a more systematic way. The pseudocode for classes and functions listed in section 3 are provided here.

Main

```
void initializeVotingSystem(){
        '''
        Purpose:    Ask the user to choose between IR or OPL voting system,
                    load the correct CSV file, read the file and process counting votes
        Input:      None
        Return:     None
        '''
        // prompt user to enter 1 for 'IR' voting or 2 for 'OPL' voting
        // check if user input is either 1 or 2
        // error message if the input is not either 1 or 2, otherwise continue
        // create Election object
        // set Election::setvotingType to the user choice (if user input is 1, set it to 'IR',
else 'OPL')
        // call loadFile(votingType)
        // start vote counting process:
        // if votingType == "IR":
        //      create IR object
        //      call IR::countVote(Election::candidateList)
        // else:
        //      create OPL object
        //      call OPL::countVote(Election::candidateList, Election::numBallots)
        // displayResult(votingType)
}
```

```
void loadFile(string votingType){
        '''
        Purpose:        Load the correct CSV file for the user choice of voting system, and
                        read the file to save in the corresponding classes and variables
        Input:          votingType (the type of voting that the user chose)
        Return:         None
        '''
        // local variables:
        //      string fileName; to save the fileName the user entered
        //      string firstLine; to save the first line of the file which contains the type of
voting
        //      string row; to save each row that contains ballot information
        //      int firstRank; to save the first-choice vote for corresponding candidate

        // prompt user to enter the file name and save it to fileName
        // call checkFile(fileName)
        // open fileName
        // read the first line of file and save it to firstLine
        // while (votingType != firstLine)
        //      error message
        //      prompt user to enter new file name and save it to fileName
        //      call checkFile(fileName)
        //      read the first line of file, and save it to firstLine
        // message to inform user that the correct CSV file has been uploaded

        // read 2nd line, and save to numCandidates
        // read 3rd line, and create Candidate object for each candidate:
        // for (i=0; i<numCandidates; i++):
        //      create Candidate object for each candidate
        //      add Candidate::name and Candidate::partyName
        //      add each candidate object to Election::candidateList
        //      for OPL, add candidates to their corresponding party:
        //      if votingType is 'OPL':
        //              if 'R': republicans.push_back(candidate)
        //              if 'D': democrats.push_back(candidate)
        //              if 'I': independent.push_back(candidate)

        // if (votingType is 'IR'):
        //      read 4th line, and save to Election::numBallots

        // if (votingType is 'OPL'):
        //      read 4th line, and save to OPL::numSeats
        //      read 5th line, and save to Election::numBallots
        // the rest of the lines contain ballots:
        // for (j=0; j<numBallots; j++):
        //      read current line, and save it to row
        //      create a Ballot object
        //      set Ballot.setRank = 1;
        //      for (i=0; i<numCandidates; i++):
        //           row[i] represents each vote for a candidate:
        //           if row[i] is Null:  set row[i] = 0
        //           else:   convert row[i] to integer
        //           add it to Ballot::rankings; rankings.push_back(row[i])
        //           if row[i] == 1: (this is the first-choice vote for IR)
        //                firstRank = i
        //    add the ballot to corresponding candidate:
        //    Election::candidateList[firstRank].ballotList.push_back(Ballot)
        // update Candidate::numVotes = ballotList.size()
}

void  checkFile(string fileName) {
        '''
        Purpose:     Keep asking user to enter the correct file
                     by checking the file exists and the file is CSV
        Input:       fileName (the name of the CSV file)
        Output:      None
        '''
        // local variables:
        //      FILE *file; to attempt to open the file to check if it exists
        //      bool error; to save boolean value to check user input the correct file
        // initialize error= true;
        // while (error):
        //      if (file exists AND type of fileName has extension .csv):
        //           error = false;
        //      else:    error message
        //           prompt user to enter the file name and save it to fileName
        // message to inform user that the file is successfully loaded
}

void  displayResult(string  votingType) {
        // display election result based on the type of voting
        // if (votingType == "IR"):
        //      there are 2 cases for the result:
        //      if case 1:      display case 1 result
        //      else:           display case 2 result
        // if (votingType == "OPL"):
```

```
        //        // display OPL voting result
        // save the result to a media text file
        // message to inform the user that the voting result has saved to (filename)
}
```

Election Class

```
class  Election {
   private:
        string   votingType;
   protected:
        int   numCandidates;
        int   numBallots;
        vector<Candidate>   candidateList;
};

void  setvotingType(string  voteType) {
        // set the type of voting
        // votingType = voteType;
}

void  setNumBallots(int  num) {
        // set the total number of ballots
        // numBallots = num;
}

void  setNumCandidates(int  num) {
        // set the total number of candidates
        // numCandidates = num;
}

void  getVotingType() {
        // return the type of voting
        // return  votingType;
}

int  getNumBallots() {
        // return the total number of ballots
        // return  numBallots;
}

int  getNumCandidates() {
        // return the total number of candidates
        // return  numCandidates;
}
```

```
vector<Candidate>  getCandidateList() {
        // get all the candidate list
        // return  candidateList;
}

void  pushCandidatetList(Candidate  newCandidate) {
        // add new candidate to the list
        // candidateList.push_back(newCandidate);
}

int  coinToss(vector<int>  numbers) {
        '''
        Purpose:        Randomly choose one number from the list of integers
        Input:          vector<int>  numbers; list of indices in candidateList
        Return:         The chosen candidate object
        '''
        // local variable:         int  chosenNum;
        // choose a random number from numbers list and save it to chosenNum
        // return chosenNum
}
```

IR Class

```
class  IR: public  Election {
   private:
        Candidate   winner;
        Candidate   loser;
        vector<vector<int>>   transferVotes;
};

void  setWinner(Candidate  candidate) {
        // set the candidate as winner
        // winner = candidate;
}

void  setLoser(Candidate  candidate) {
        // set the candidate who got fewest votes as loser
        // loser = candidate;
}

Candidate  getWinner() {
        // return the winner candidate
        // return winner;
}
```

```
Candidate  getLoser() {
        // return the candidate who got fewest vote
        // return loser;
}

vector<vector<int>>  getTransferVotes() {
        // return the transfer votes list for all candidates
        // return transferVotes;
}

Candidate  countVote(vector<Candidate>  candidateList) {
        '''
        Purpose:        Count first-choice vote for all candidates, do transfer voting if
                        require and, determine winning and losing candidates
        Input:          vector<Candidate> candidateList; all the candidates object
        Return:         the candidate object who wins the election
        '''
        // local variables:
        //        int minVotes; to save the minimum vote count
        //        vector<int> minIdx; to save the indices of the candidates who receive the
lowest vote
        //        int majorityVote; to save the 50% of total votes
        //        int candidateLeft; to save the number of candidates left after elimination
        //        vector<int> idxRemainder; to save the indices of two candidates who tie
        // initialize minVoteCount = Election::numBallots
        // initialize minIdx = -1
        // initialize candidateLeft = candidateList.size();
        // 1. calculate the 50% of total votes
        // majorityVote = 0.5 * Election::numBallots;
        // 2. loop until there is a winner or only 2 candidates left
        //    from the fewest vote count candidate
        // while (winner is NULL AND candidateLeft > 2):
        //        2.1 check if one candidate gets majority vote, set winner and break loop
        //        for (i=0; i<candidateList.size(); i++):
        //                if candidateList[i].isEliminated() == false:
        //                        if candidateList[i].ballotList.size() > majorityVote:
        //                                set winner to candidateList[i]
        //                                break
        //                        2.2 get minimum vote counts and save it to minVoteCount
        //                        if minVotes > candidateList[i].ballotList.size():
        //                                minVoteCount = candidateList[i].ballotList.size()
        //        2.3 get all candidates who got minimum votes, and save them to minIdx
        //        for (i=0; i<candidateList.size(); i++):
```

```
        //              if (! candidateList[i].isEliminated()):
        //                  if (candidateList[i].ballotList.size() == minVoteCount):
        //                      minIdx.push_back(i)
        //      2.4 for all candidates who got minimum votes, call coinToss to select one
        //      set loser = candidateList[Election::coinToss(minIdx)]
        //      2.5 transfer votes from the loser candidate
        //      call transferVote(loser, candidateList)
        //      reduce the candidateLeft --;
        // 3. if there is a winner, return winner; else, only two candidates must have left after the loop
        // if (winner is not NULL):
        //      return winner
        // else: (winner is NULL)
        //      get the two candidates left:
        //      for (i=0; i<candidateList.size(); i++):
        //              if candidateLIst[i].isEliminated() == false:
        //                      idxRemainder.push_back(i)
        //      winner = candidateList[Election::coinToss(idxRemainder)]
        //      return winner
}

void  transferVote(Candidate  loser, vector<Candidate>  candidateList) {
        '''
        Purpose:        Transfer votes from the candidate with the fewest votes to others
        Input:          Candidate loser; the candidate with the fewest votes
                        vector<Candidate> candidateList; all candidate list
        Return:         None
        '''
        // local variables:
        //      Ballot vote; to store the current ballot from loser candidate
        //      int nextRank; to save the index of the next ranked candidate
        //      int idx; to save the index of the candidate who gets the transfer vote
        // 1. eliminate the loser candidate
        // set loser.setCandidateEliminated = true
        // 2. get each ballot of the loser candidate
        // for (j=0; j<transferVote.size(); j++):
        //      transferVote = loser.getBallotList()[j];
        //      2.1 get the next rank in the ballot
        //      nextRank = transferVote[i].getRank() + 1;
        //      2.2 transfer the ballot to the corresponding candidate
        //      for (i=0; i<candidateList.size(); i++):
        //              2.2.1 if the candidate has not been eliminated, save the current index to idx
        //              2.2.2 else update the next rank in ballot
        //      candidateList[idx].getBallotList.push_back(transferVote)

}
```

OPL Class

```
class  OPL: public  Election {
   private:
        int   numSeats;
        int   quota;
        int   repVotes;
        int   demVotes;
        int   indVotes;
        vector<int> firstSeats; // to save the seats from first allocation
        vector<int> remainingVotes; // to save the remaining votes after seat allocation
        vector<int> secondSeats; // to save the seats from second allocation
        vector<Candidate>   republicans;
        vector<Candidate>   democrats;
        vector<Candidate>   independent;
        vector<Candidate>   winners;
};

void  setNumSeats(int  num) {
        // set the number of seats
        // numSeats = num;
}

void  setRepVotes(int  num) {
        // set the number of votes for republicans
        // repVotes = num;
}

void  setDemVotes(int  num) {
        // set the number of votes for democrats
        // demVotes = num;
}

void  setIndVotes(int  num) {
        // set the number of votes for independent
        // indVotes = num;
}

int  getNumSeats() {
        // return the total number of seats
        // return numSeats
```

```
}

int  getQuota() {
        // return the quota value
        // return quota;
}

int  getRepVotes() {
        // return the votes republicans received
        // return repVotes;
}

int  getDemVotes() {
        // return the votes democrats received
        // return demVotes;
}

int  getIndVotes() {
        // return the votes independent received
        // return indVotes;
}

vector<int>  getFirstSeats(){
        // return the first seat allocation
}

void  setFirstSeats(vector<int>){
        // set the private attribute firstSeats
}

vector<int>  getRemainingVotes(){
        // return the remaining Votes
}

void  setRemainingVotes(vector<int>){
        // set the private attribute remainingVotes
}

vector<int>  getSecondSeats(){
        // return the secondSeats allocation
}

void  setSecondSeats(vector<int>){
        // set the private attribute secondSeats
```

```
}

vector<Candidate>  **getList**(char  value) {
        // return the list of each party depends on the value
        // value must be 'R', 'D', 'I', or 'W'
        // if (value == 'R'):
        //        return republicans;
        // else if (value == 'D'):
        //        return democrats;
        // else if (value == 'W'):
        //        return independent
        // else: return winners;
}

void  **pushList**(char  value, Candidate  candidate) {
        // add a new candidate to corresponding party list based on input value
        // if (value == 'R'):
        //        republicans.push_back(newCandidate);
        // else if (value == 'D'):
        //        democrats.push_back(newCandidate);
        // else if (value == 'I'):
        //        independent.push_back(newCandidate);
        // else: winners.push_back(newCandidate);;
}

void  **calculateQuota**(int  numVotes, int  numSeats) {
        // calculate and return quota value
        // quota = (int)ceil(numVotes/numSeats);
}

vector<int>  **calculateSeatAllocation**(vector<int>  numVoteList, int  quota) {
        '''
        Purpose:        Calculate seat allocation for each party
        Input:          vector<int> numVoteList; list of the number of votes for each party
                        int quota; quota value
        Return:         vector<int> totalSeats; total number of seats each party received
        '''
        // local variables:
        //        int allocatedSeats; to save the total number of seats after first allocation
        //        int remainingSeats; to save the remaining seats after first allocation
        //        vector<int> totalSeats; to save total number of seats each party received
        // initialize allocatedSeats = 0;
        // 1. calculate the first allocation of seats and save in corresponding variables
        // for (i=0; i<numVoteList.size(); i++):
```

```
        //      firstSeats[i].push_back((int)numVoteList[i] / quota);
        //      remainingVotes.push_back(numVoteList[i] % quota)
        //      allocatedSeats += 1;
        // 2. get the remaining seats
        // remainingSeats = numSeats - allocatedSeats;
        // 3. calculate the second allocation of seats
        // 3.1 get the largest remainders from remainingVotes
        // for (i=0; i<remainingSeats; i++):
        //      3.2 add the remainingSeat to those largest remainders party

        // update secondSeats list
        // 4. save final total number of seats for each party to totalSeats
        // for (i=0; i<numVotesList.size(); i++):
        //      totalSeats[i].push_back(firstSeats[i]+secondSeats[i])
        // return totalSeats
}

void  countVote(vector<Candidate>  candidateList, int  numBallots) {
        '''
        Purpose:      Count votes for each party, calculate seat allocation calculate the
                      percentage of votes to seats, and determine winning party
        Input:        vector<Candidate> candidateList; the list of candidates
                      int numBallots; total number of ballots
        Return:       None
        '''
        // local variables:
        //      vector<int> numVoteList; to save list of number of votes for each party
        //      vector<int> totalSeats; to save total number of seats each party received
        //      int seats; to save the number of seats each party received
        // 1. calculate quota
        // call calculateQuota(numBallots, numSeats);
        // 2. count votes for each party
        // initialize repVotes = demVotes = indVotes = 0;
        // for (i=0; i<republicans.size(); i++):
        //      repVotes += republicans[i].getBallotList().size()
        // for (i=0; i<democrats.size(); i++):
        //      demVotes += democrats[i].getBallotList().size()
        // for (i=0; i<independent.size(); i++):
        //      indVotes += independent[i].getBallotList().size()
        // add total votes for each part to numVoteList:
        // numVoteList = [repVotes, demVotes, indVotes]
        // 3. calculate seat allocation and set it to totalSeats
        // totalSeats = calculateSeatAllocation(numVoteList, quota)
        // 4. sort each party list:
```

```
        // sort(republicans); sort(democrats); sort(independent);
        // 5. get the winning candidates from each party depending on total seats
        // 5.1 loop each party and get kth candidates (k=total seats for each party)
        // first value in totalSeats is for republicans, so add kth rep candidates to winners
        // for (i=0; i<totalSeats[0]; i++):
        //      5.2 add those kth candidates to winners list
        //      winners.push_back(republicans[i]);
        // for (i=0; i<totalSeats[1]; i++):
        //      winners.push_back(democrats[i]);
        // for (i=0; i<totalSeats[0]; i++):
        //      winners.push_back(independent[i]);
}
```

Ballot Class

```
class  Ballot() {
   private:
        vector<int>   rankings;
        int   nextRank;
};

void  setVote(string value) {
        // separate values by comma, convert those to integers
        // set those values to rankings
}

void  setRank(int rank) {
        // get the nextRank value
        // nextRank = rank;
}

vector<int>  getVote() {
        // return the list of rankings
        // return rankings
}

int  getRank() {
        // return the nextRank value
        // return nextRank;
}
```

Candidate Class

```
class  Candidate() {
   private:
        string   candidateName;
        char   party;
        int   numVotes;
        vector<Ballot>   ballotList;
        bool   eliminated;
};

void  setName(string  name) {
        // set the name of the candidate
        // candidateName = name;
}

void  setParty(char  name) {
        // set the name of the party associated with the candidate
        // partyName = name;
}

void  setNumVotes(int  num) {
        // set the number of votes for the candidate
        // numVotes = num;
}

void  setCandidateEliminated(bool  value) {
        // set true or false value to eliminated
        // eliminated = value;
}

string  getName() {
        // return the name of the candidate
        // return name;
}

char  getParty() {
        // return the name of the party associated with the candidate
        // return partyName;
}

int  getNumVotes() {
        // return the number of votes the candidate received
        // return numVotes;
}

bool  isEliminated() {
        // return true if the candidate is eliminated, false otherwise
        // return eliminated;
```

```
}

vector<Ballot>  getBallotList() {
        // return the list of ballots of the candidate
        // return ballotList;
}

void  ballotListpush(Ballot  newBallot) {
        // add a ballot to ballotList
        // ballotList.push_back(newBallot);
}

vector<Ballot>  ballotListpop(  ) {
        // given the index of the ballot in the list, remove the ballot from ballotList
        // ballotList.erase(idx)
}
```

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

Instant Runoff (IR) Voting Interface

This section illustrates the two voting systems' user interfaces: Instant Runoff voting system (IR) and Open Party List voting system (OPL). Since we are going to implement this project in C/C++, this will be the text-based user interface. Our voting system starts by prompting the user to choose between the Instant Runoff voting system or the Open Party List voting system.

The system asks to provide the name of the CSV file for IR voting. An error message will display for the following cases:

- the file does not exist
- the file is not .csv format
- the file is loaded successfully, but the first line of the file does not match "IR"

Following the error message, the user will be asked to re-enter the file's name until it is successfully loaded into the system and the first line matches "IR."

After the user confirms to continue, the system will begin counting the first-choice votes for each candidate. The message "Counting votes in progress" will be displayed to inform the user during this process.

After the vote-counting process, two cases can result.

Case 1:

One candidate receives over 50% of the first-choice vote. Then, the system will display the result of the election.

Case 2:

If no candidate receives a majority, the candidate with the fewest votes is eliminated, and the system will begin the transfer voting process.

The voter's first-choice for the candidate with the fewest votes will be collected, and those votes will be transferred to the second-choice candidates. If the result is case 2, the system will repeat the transfer voting process.

Open Party List (OPL) Voting Interface

The system asks to provide the name of the CSV file for OPL voting. An error message will display for the following cases:

- the file does not exist
- the file is not .csv format
- the file is loaded successfully, but the first line of the file does not match "OPL"

Following the error message, the user will be asked to re-enter the file's name until it is successfully loaded into the system and the first line matches "OPL".

After the file is uploaded successfully and the user confirms to continue, the system will begin the process of counting the votes for candidates from within the same party. The message "Counting votes in progress" will be displayed to inform the user during this process.

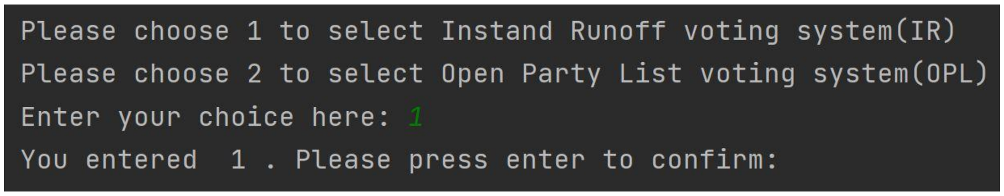
To calculate the seat allocation for OPL voting, the largest remainder formula will be implemented. The system will calculate the quota and the allocation of seats for each party using the number of votes counted and seats. The message "Calculating seat allocation" will be displayed to inform the user.

At the end of the vote-counting and seat allocation process, the system will display the result of OPL voting and the information of the total number of seats each party won and the percentage of votes to the percentage of seats for each party.

After the result is displayed, the system will create and open an audit file to be viewed to verify the results. A message will be displayed to inform the user of this process. Then, the user will be asked whether to share the result with other authorized users.

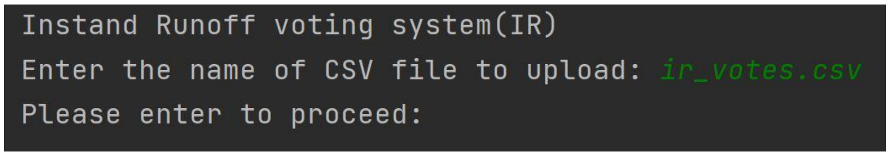
In a special case when there is a tie in either IR or OPL voting, a fair coin toss will be run by the program. In this case, the user will be informed that the fair coin toss process has begun, and the result will be shown at the end.

6.2 Screen Images



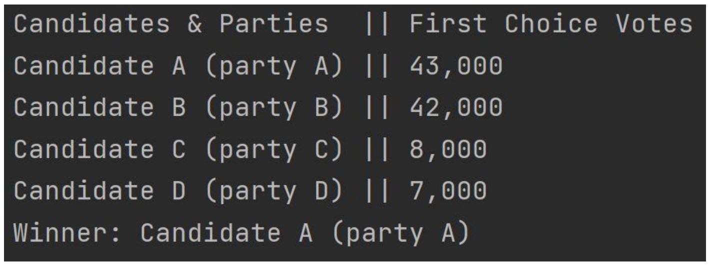
```
Please choose 1 to select Instand Runoff voting system(IR)
Please choose 2 to select Open Party List voting system(OPL)
Enter your choice here: 1
You entered  1 . Please press enter to confirm:
```

Figure 6.1 User input for choosing the type of voting system



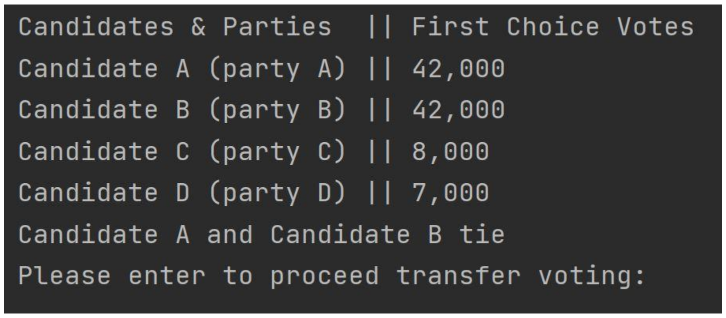
```
Instand Runoff voting system(IR)
Enter the name of CSV file to upload: ir_votes.csv
Please enter to proceed:
```

Figure 6.2 Interface for uploading CSV file for IR voting

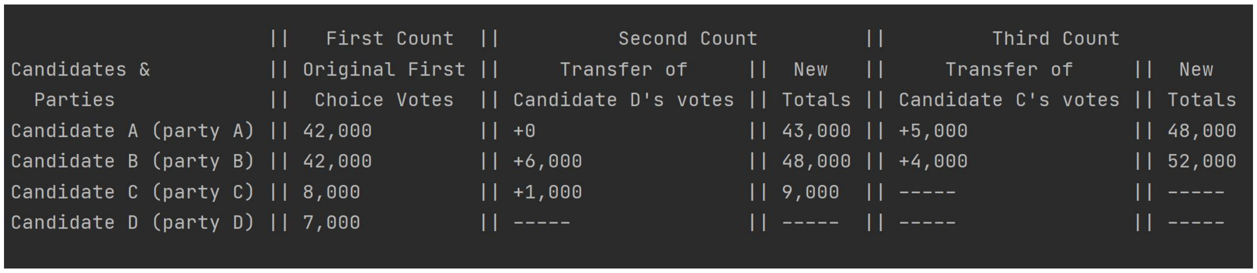


```
Candidates & Parties  || First Choice Votes
Candidate A (party A) || 43,000
Candidate B (party B) || 42,000
Candidate C (party C) || 8,000
Candidate D (party D) || 7,000
Winner: Candidate A (party A)
```

Figure 6.3 Result of the election for case 1

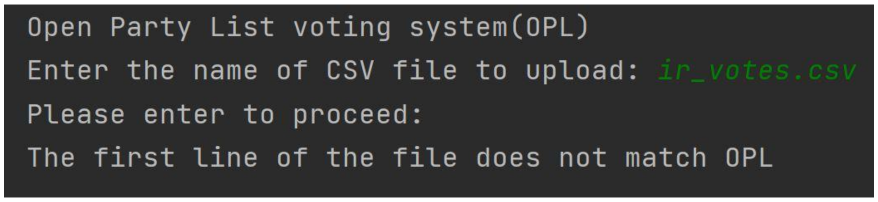


```
Candidates & Parties  || First Choice Votes
Candidate A (party A) || 42,000
Candidate B (party B) || 42,000
Candidate C (party C) || 8,000
Candidate D (party D) || 7,000
Candidate A and Candidate B tie
Please enter to proceed transfer voting:
```



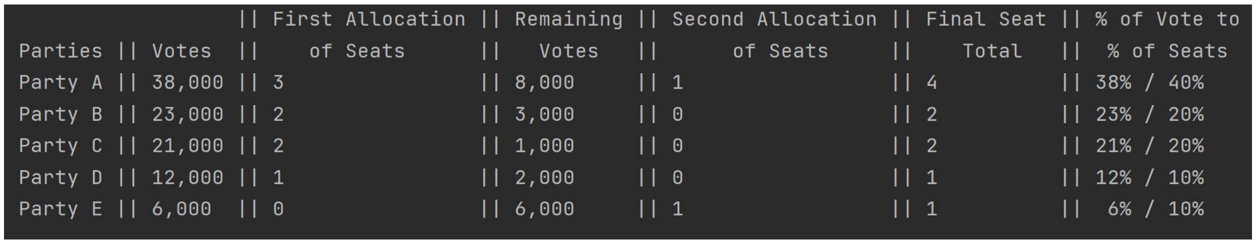
```
                     ||   First Count  ||          Second Count         ||         Third Count
Candidates &         || Original First ||     Transfer of      ||  New   ||     Transfer of      ||  New
  Parties            ||  Choice Votes  || Candidate D's votes || Totals || Candidate C's votes || Totals
Candidate A (party A) || 42,000         || +0                  || 43,000 || +5,000              || 48,000
Candidate B (party B) || 42,000         || +6,000              || 48,000 || +4,000              || 52,000
Candidate C (party C) || 8,000          || +1,000              || 9,000  || -----               || -----
Candidate D (party D) || 7,000          || -----               || -----  || -----               || -----
```

Figure 6.4 and 6.5 Result of the election for case 2



```
Open Party List voting system(OPL)
Enter the name of CSV file to upload: ir_votes.csv
Please enter to proceed:
The first line of the file does not match OPL
```

Figure 6.6 Interface for uploading CSV file for OPL voting



```
                  || First Allocation || Remaining || Second Allocation || Final Seat || % of Vote to
Parties || Votes  ||    of Seats      ||   Votes   ||      of Seats     ||    Total   ||  % of Seats
Party A || 38,000 || 3                || 8,000     || 1                 || 4          || 38% / 40%
Party B || 23,000 || 2                || 3,000     || 0                 || 2          || 23% / 20%
Party C || 21,000 || 2                || 1,000     || 0                 || 2          || 21% / 20%
Party D || 12,000 || 1                || 2,000     || 0                 || 1          || 12% / 10%
Party E || 6,000  || 0                || 6,000     || 1                 || 1          ||  6% / 10%
```

Figure 6.7 Result of the election for OPL voting

6.3 Screen Objects and Actions

The text-based interface will handle every action. Terminal/Command Prompt will have its default options, but the program will be handled by commands entered.

7. REQUIREMENTS MATRIX

Use Case ID	**Function/Method**	**Additional Info**
OPL_01	SetNumSeats()	OPL Class Method, Reads from file
OPL_02	CalculateQuota()	OPL Class Method, info needed read from file

VS_01	LoadFiles()	Called in main, opens user argument
VS_02	User input from prompt, match with getline(filename)	System calls in main
VS_03	SetType()	Defined in election class, string read from file in main
VS_04	SetNumCandidates()	Election Class method, get int from VS_02
VS_05	CandidateListPush(Candidate)	Election class, instantiate candidate from string read from VS_02
VS_06	SetNumBallots() and GetNumBallots()	Ballot class, updates each time a ballot is read and created with VS_02
VS_07	CoinToss()	Defined in Election class
VS_08	saveToAuditFile()	This will open the audit file and save needed election info to it
VS_09	displayResult()	Called in main, same info as VS_11
VS_10	getline() until EOF	Creates an instance of ballot from string that's read
VS_11	displayResult()	Same information as VS_09

Table 7.1 Requirements Matrix