Team 11
Voting System
Software Design Document

Names: Hoai Bui, Ryan Mower, Eric Palmer, Emma Spindler
Date:

Table of Contents

1. Introduction

1.1 Purpose

This software design document describes the architecture and system design of the Voting System for Open Party elections and Instant Run-Off elections to determine a winner for the type of election. This document is made for the developers and testers of the Voting System.

1.2 Scope

The Voting System is a tool intended to be used for both Instant Runoff and Open Party Listing elections. Election officials will organize ballots into a CSV file and the software will perform the appropriate operations in order to determine the winner(s) based on the specified election type. The results along with other important information regarding the election will be returned in an audit and media report to be reviewed and shared. The elected officials will be able to run the Voting System to get results from the election and will be provided a media file to share with media personnel. The Voting System allows up to 100,000 ballots to be read in 8 minutes allowing the winner to be determined in a more timely manner. The Voting System eliminates any human error in counting ballots assuming the CSV file is accurate.

1.3 Overview

Section 2 contains the project design and the functionality provided by the program to the user. Section 3 consists of the program structure and the various subsystems that make up the overall system of the project. Section 4 explains how data will be stored and organized, as well as the major data types used in the project. Section 5 describes a more detailed look at the program, explaining the algorithms and function implementation. Section 6 explains how the user will interact with the project by detailing the user interface and the user's methods of inputting and receiving data. Section 7 discusses how our project meets all the requirements detailed in the SRS.

1.4 Reference Material

1.5 Definitions and Acronyms

2. System Overview

Our project is a voting system which can process a CSV of votes file to produce statistics about the voting process in the form of a media or audit file, as well as displaying these results to the terminal. It will also have the ability to recreate each step of the voting process along the way. (fill in information about the project design).

3. System Architecture

3.1. Architectural Design

There are three main subsystems within the Voting System program including:

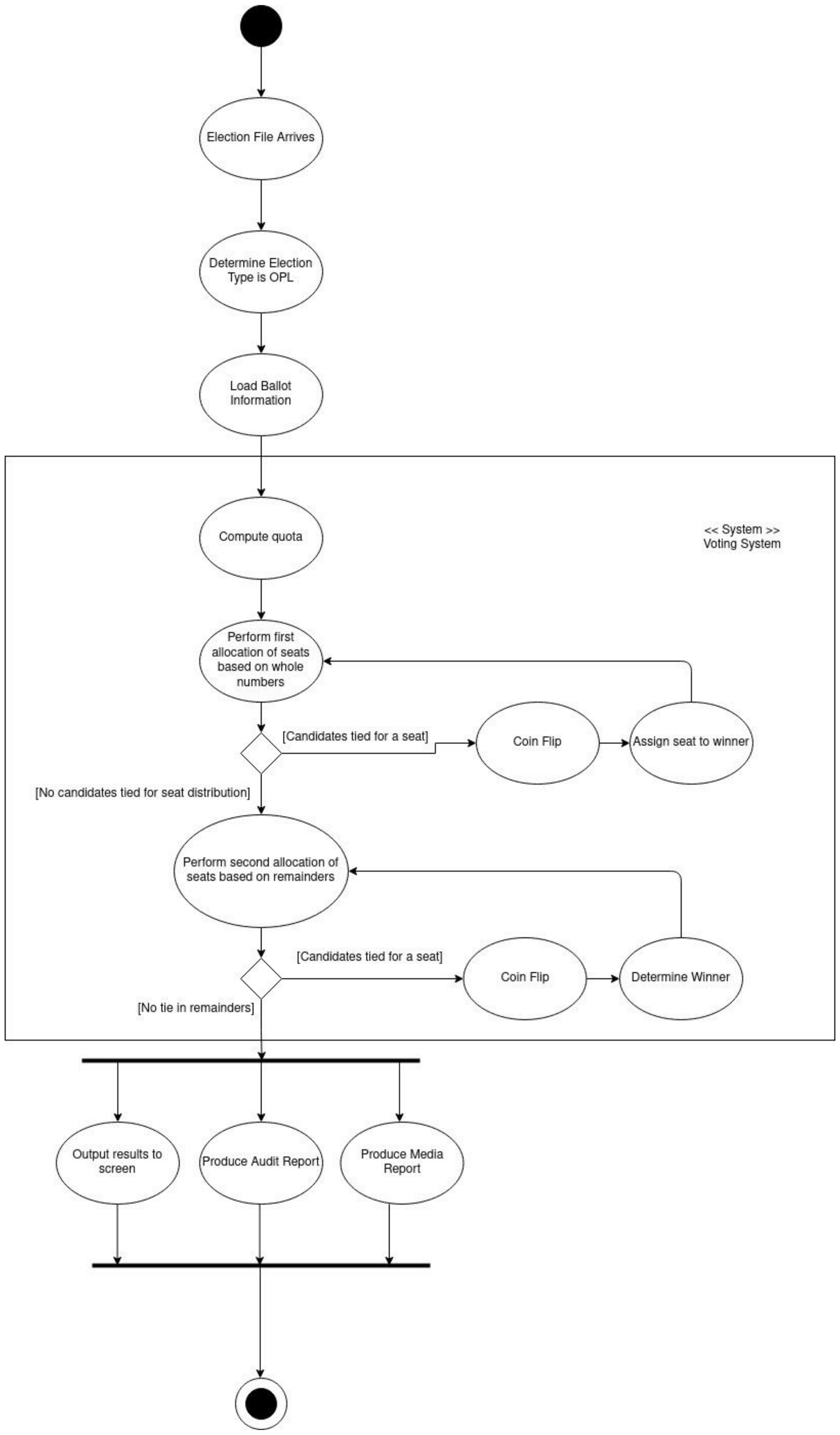
1. Subsystem 1
    a. Loading Election Information from a CSV file.
2. Subsystem 2
    a. The two different election algorithms, Open Party Listing and Instant Runoff.
3. Subsystem 3
    a. Generation of election results, outputting the data to the screen, the creation of the media and audit report.

(Subsystem 1) The first subsystem, the loading of election information from a CSV file, is the first module activated during program execution. This subsystem is responsible for storing all of the election information regarding ballots, number of seats, the candidates and all other required information to compute the election results into memory. It does this by parsing the CSV and assigning the variables accordingly. This will allow for the future substems to perform the appropriate calculations. Once the election information has been loaded into memory, these subsystems responsibilities are completed and control will be passed down to the next subsystem.

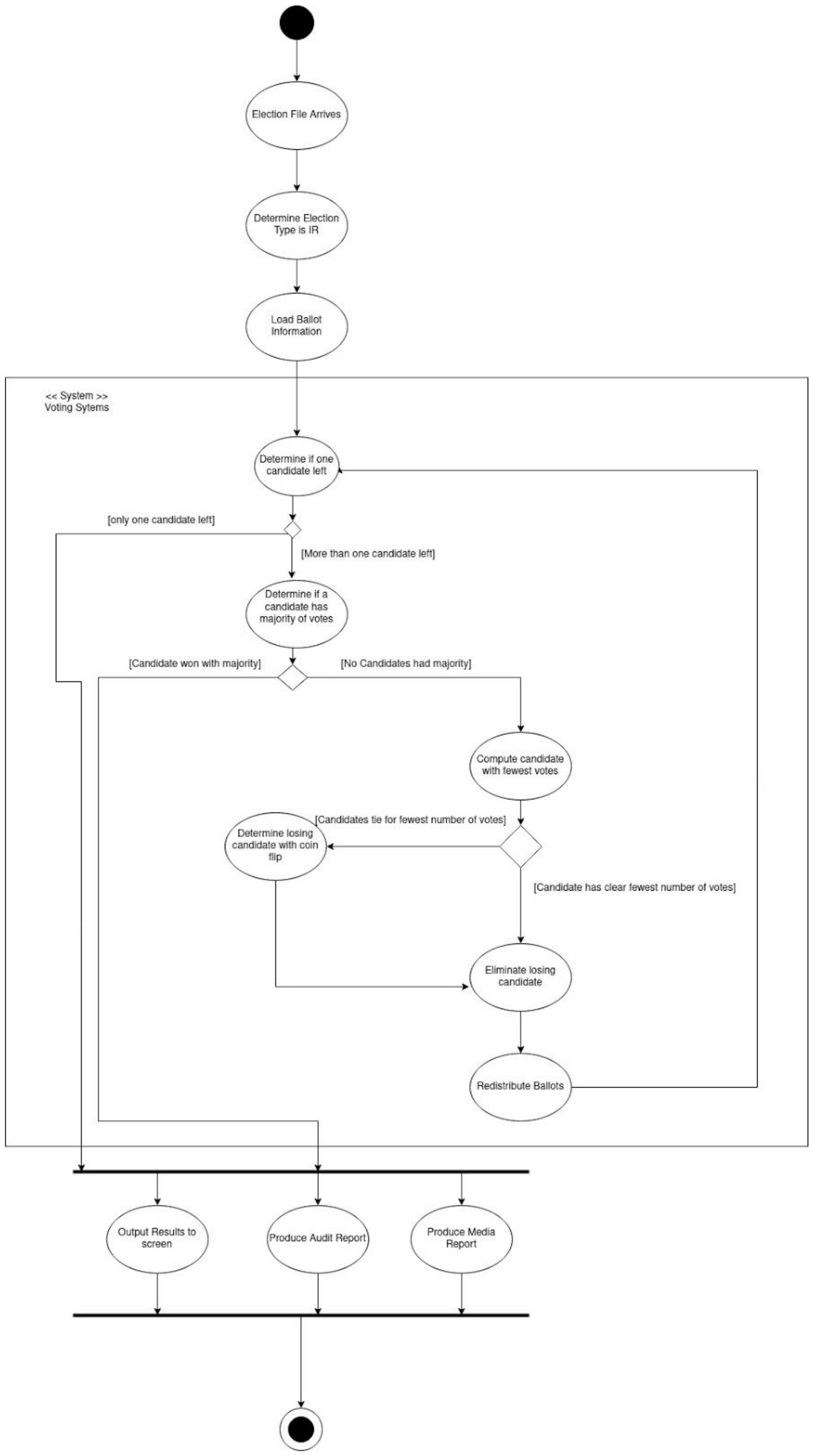
(Subsystem 2) Open Party Listing and Instant Runoff are both election algorithms that are considered subsystems. Their responsibilities include calculating the election results corresponding to the running election type. Since all of the required election information is loaded into memory from the first subsystem, the OPL and IR algorithms will calculate the election results accordingly. After the results are computed, control will be passed onto the final subsystem.

(Subsystem 3) Generation of election results is responsible for outputting the election results in several different forms. These forms of communication include outputting the results to the screen and creating a media and audit report. Once control is directed at this subsystem, it will take the results computed from Subsystem 2 and output them to the screen and create an audit and media report. Both OPL and IR have their own desired output format and Subsystem 3 will abide by these rules.

OPL_Activity_Diagram -
This diagram shows the flow the Voting System will endure during program execution for an instant runoff election type. It depicts the control paths that will be taken depending on the election information.

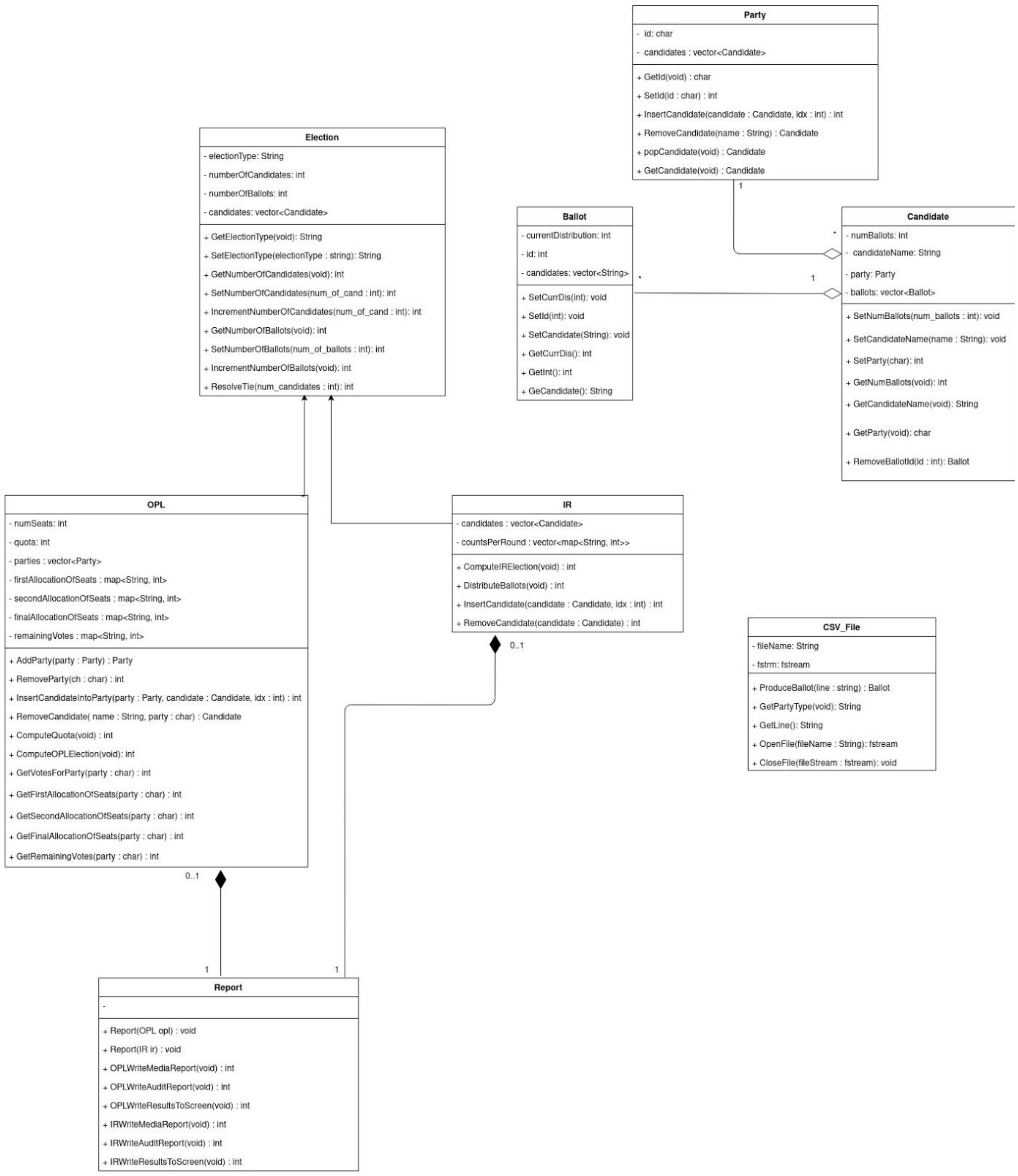

Election File Arrives
Determine Election Type is OPL
Load Ballot Information
Compute quota
<< System >> Voting System
Perform first allocation of seats based on whole numbers
[Candidates tied for a seat]
Coin Flip
Assign seat to winner
[No candidates tied for seat distribution]
Perform second allocation of seats based on remainders
[Candidates tied for a seat]
Coin Flip
Determine Winner
[No tie in remainders]
Output results to screen
Produce Audit Report
Produce Media Report

IR_Activity_Diagram -
This diagram shows the flow the Voting System will endure during program execution for an instant runoff election type. It depicts the control paths that will be taken depending on the election information.


Election File Arrives
Determine Election Type is IR
Load Ballot Information
<< System >>
Voting Sytems
Determine if one candidate left
[only one candidate left]
[More than one candidate left]
Determine if a candidate has majority of votes
[Candidate won with majority]
[No Candidates had majority]
Compute candidate with fewest votes
[Candidates tie for fewest number of votes]
Determine losing candidate with coin flip
[Candidate has clear fewest number of votes]
Eliminate losing candidate
Redistribute Ballots
Output Results to screen
Produce Audit Report
Produce Media Report

3.2 Decomposition Description

UML_Class_Diagram -
The UML class diagram displays the class layout, structure, and interactions for the Voting System.


Party
- id: char
- candidates : vector<Candidate>
+ GetId(void) : char
+ SetId(id : char) : int
+ InsertCandidate(candidate : Candidate, idx : int) : int
+ RemoveCandidate(name : String) : Candidate
+ popCandidate(void) : Candidate
+ GetCandidate(void) : Candidate
1
*
Election
- electionType: String
- numberOfCandidates: int
- numberOfBallots: int
- candidates: vector<Candidate>
+ GetElectionType(void): String
+ SetElectionType(electionType : string): String
+ GetNumberOfCandidates(void): int
+ SetNumberOfCandidates(num_of_cand : int): int
+ IncrementNumberOfCandidates(num_of_cand : int): int
+ GetNumberOfBallots(void): int
+ SetNumberOfBallots(num_of_ballots : int): int
+ IncrementNumberOfBallots(void): int
+ ResolveTie(num_candidates : int): int
Ballot
- currentDistribution: int
- id: int
- candidates: vector<String>
+ SetCurrDis(int): void
+ SetId(int): void
+ SetCandidate(String): void
+ GetCurrDis(): int
+ GetInt(): int
+ GeCandidate(): String
*
1
Candidate
- numBallots: int
- candidateName: String
- party: Party
- ballots: vector<Ballot>
+ SetNumBallots(num_ballots : int): void
+ SetCandidateName(name : String): void
+ SetParty(char): int
+ GetNumBallots(void): int
+ GetCandidateName(void): String
+ GetParty(void): char
+ RemoveBallotId(id : int): Ballot
OPL
- numSeats: int
- quota: int
- parties : vector<Party>
- firstAllocationOfSeats : map<String, int>
- secondAllocationOfSeats : map<String, int>
- finalAllocationOfSeats : map<String, int>
- remainingVotes : map<String, int>
+ AddParty(party : Party) : Party
+ RemoveParty(ch : char) : int
+ InsertCandidateIntoParty(party : Party, candidate : Candidate, idx : int) : int
+ RemoveCandidate(name : String, party : char) : Candidate
+ ComputeQuota(void) : int
+ ComputeOPLElection(void): int
+ GetVotesForParty(party : char) : int
+ GetFirstAllocationOfSeats(party : char) : int
+ GetSecondAllocationOfSeats(party : char) : int
+ GetFinalAllocationOfSeats(party : char) : int
+ GetRemainingVotes(party : char) : int
IR
- candidates : vector<Candidate>
- countsPerRound : vector<map<String, int>>
+ ComputeIRElection(void) : int
+ DistributeBallots(void) : int
+ InsertCandidate(candidate : Candidate, idx : int) : int
+ RemoveCandidate(candidate : Candidate) : int
0..1
CSV_File
- fileName: String
- fstrm: fstream
+ ProduceBallot(line : string) : Ballot
+ GetPartyType(void): String
+ GetLine(): String
+ OpenFile(fileName : String): fstream
+ CloseFile(fileStream : fstream): void
0..1
1
1
Report
-
+ Report(OPL opl) : void
+ Report(IR ir) : void
+ OPLWriteMediaReport(void) : int
+ OPLWriteAuditReport(void) : int
+ OPLWriteResultsToScreen(void) : int
+ IRWriteMediaReport(void) : int
+ IRWriteAuditReport(void) : int
+ IRWriteResultsToScreen(void) : int

The above diagram depicts the class layout and structure for the Voting System. In the bottom right is the class *CSV_FIle*, which is an independent class used to interact and parse the CSV file. *Party* and *Election* are the two parent classes. *Party* contains a list of *Candidates* and *Candidates* contain a list of *Ballots*. All ballots in a candidate's list means those ballots are counting as votes for that candidate. *Election* is a parent class that has two children, OPL and IR. OPL stands for Open Party Listing and IR stands for Instant Runoff. Both OPL and IR are election types, so they will have unique methods that compute their specified election results from the provided election information.

3.3 Design Rationale

Several design choices were contemplated before implementation. The independent *CSV_File* class was unmodified because it provides an efficient way to parse the CSV file and load the appropriate data into memory. However, the *OPL* and *IR* class ascend from the Election class. These two classes were originally combined, but were split into OPL and IR because their methods have unique computations. However, much of the information was related. To minimize code, a *Election* class was created which stores important election information both OPL and IR use.

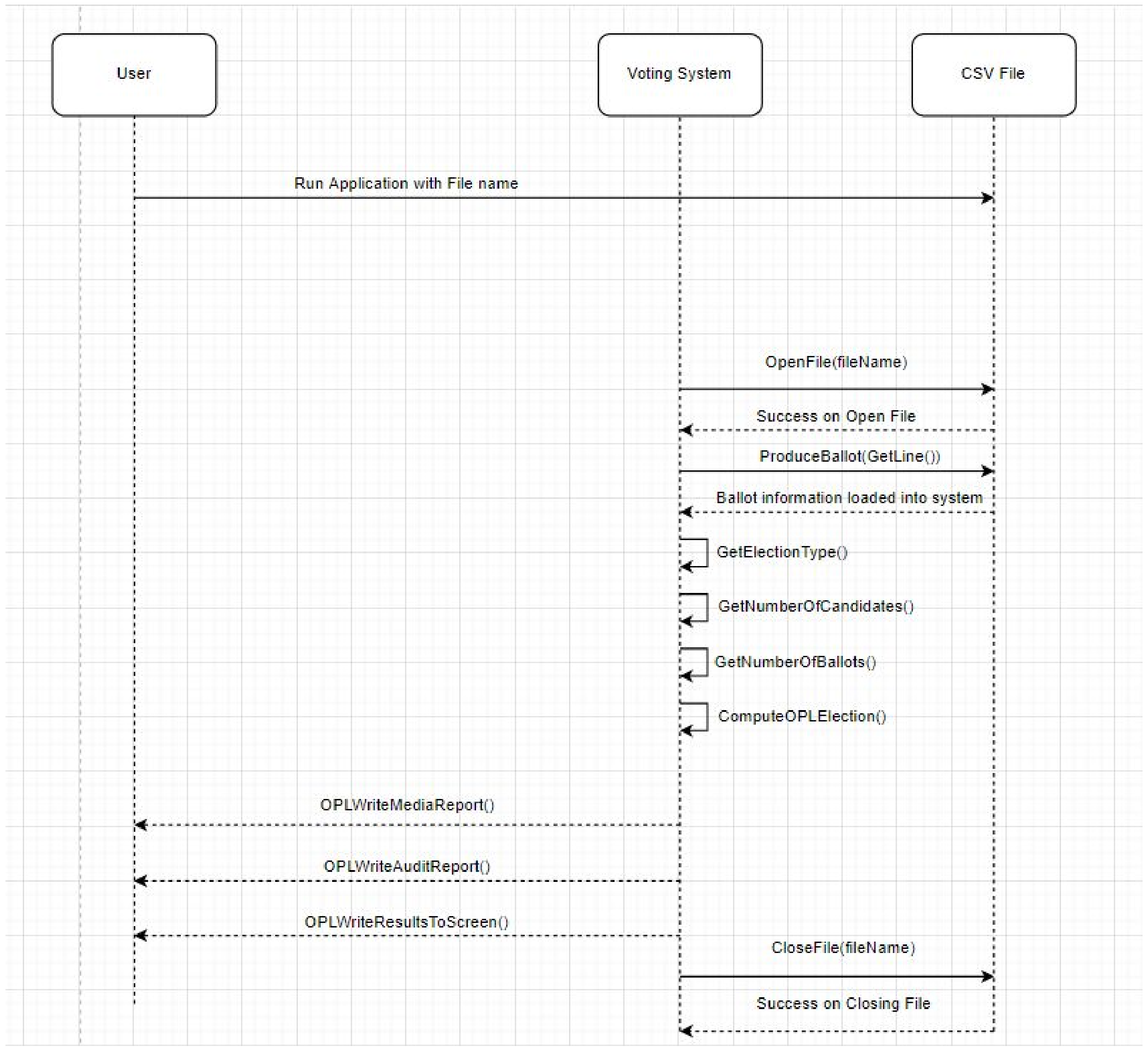
Another architectural design was necessary for improved program efficiency. If ballots were the only form of tracking which candidate it is for, redistributing them would require constant traversal over the list of ballots to update their candidate. This proves incredibly inefficient and solved by having specific candidates contain a list of ballots that are for them. In the Instant Runoff election, ballots corresponding to an eliminated candidate can be redistributed to the ballot's next favorite candidate. This will prevent touching all ballots when it is not required.

Deciding what information *OPL* and *IR* classes hold was another heavily contemplated topic. Producing the audit report during the algorithm execution would be more efficient, but require much more complexity to produce the audit and media reports accurately. To combat this issue, extra field variables were added to the *OPL* and *IR* to track the required audit and media report information. This solved the problem because the *Report* class could then access those extra field variables and create the proper reports.

4. Data Design

4.1 Data Description

Sequence Diagram - Below is the sequence diagram which displays the interactions between the various entities involved with the voting system.



Entities in the Voting System are implemented as classes. Each ballot, candidate, party, election, report, and CSV is represented as its own object. The CSV_File object will parse the CSV file taken in. The information will be instantiated into the Ballot and Candidate classes. Objects are also created for each party and they are populated with information from all of the candidates.

4.2 Data Dictionary

Class	**Data Members**	**Methods**
Ballot This class	- **currentDistribution(int)** Keeps track of which count it's in	- SetCurrDis(int): void

creates an object for every ballot.	- **id(int)** Ballot ID - **candidates(vector<string>)** The candidate(s) voted for	- SetId(int): void - SetCandidate(String): void - GetCurrDis(): int - GetInt(): int - GetCandidate(): String
Candidate This class creates an object for every candidate.	- **numBallots(int)** The total count of ballots the candidate receives - **candidateName(String)** Name of the candidate - **party(Party)** The party in which the candidate belongs to - **ballots(vector<Ballot>)** Stores all the ballots the candidate receives	- SetNumBallots(num_ballots : int): void - SetCandidateName(name : string): void - SetParty(char): int - GetNumBallots(void): int - GetCandidateName(void): string - GetParty(void): char - RemoveBallotId(id : int): Ballot
CSV_File A class for parsing the CSV file	- **fileName(String)** Name of the CSV file - **fstrm(fstream)** Object for reading the file	- ProduceBallot(line : string) : Ballot - GetPartyType(void): string - GetLine(): string - OpenFile(fileName : string): fstream - CloseFile(fileStream : fstream): void
Election Parent class for each election type with basic members and methods that all elections will share.	- **electionType(String)** The type of election - **numberOfCandidates(int)** The number of candidates in the election - **numberOfBallots(int)** The number ballots in the election - **candidates(vector<Candidate>)** Contains all of the candidates	- GetElectionType(void): String - SetElectionType(electionType : string): String - GetNumberOfCandidates(void): int - SetNumberOfCandidates(num_of_cand : int): int - IncrementNumberOfCandidates(num_of_cand : int): int - GetNumberOfBallots(void): int - SetNumberOfBallots(num_of_ballots : int): int - IncrementNumberOfBallots(void): int - ResolveTie(num_candidates : int): int

IR This class is responsible for computing an Instant Runoff election.	- **countsPerRound(vector<map<String, int>>)** Creates a vector that keeps track of each count in the election	- ComputeIRElection(void) : int - DistributeBallots(void) : int - InsertCandidate(candidate : Candidate, idx : int) : int - RemoveCandidate(candidate : Candidate) : int
OPL This class is responsible for computing an Open Party List voting election.	- **numSeats(int)** The number of seats open for candidates.. - **quota(int)** Used to fill out the proper number of seats per party - **parties(vector<Party>)** Parties in the election - **firstAllocationOfSeats(maps<String,int>)** First determination of seats for each party - **secondAllocationOfSeats(maps<String,int>)** Second determination of seats for each party - **finalAllocationOfSeats(maps<String, int>)** Final determination of seats for each party - **remainingVotes(map<String, int>)** Votes remaining	- AddParty(party : Party) : Party - RemoveParty(ch : char) : int - InsertCandidateIntoParty(party : Party, candidate : Candidate, idx : int) : int - RemoveCandidate(name : String, party : char) : Candidate - ComputeQuota(void) : int - ComputeOPLElection(void): int - GetVotesForParty(party : char) : int - GetFirstAllocationOfSeats(party : char) : int - GetSecondAllocationOfSeats(party : char) : int - GetFinalAllocationOfSeats(party : char) : int - GetRemainingVotes(party : char) : int
Party This class creates an object for every party that a candidate can belong to.	- **id(char)** First letter of party name - **candidates(vector<Candidate>)** All of the candidates in the specific party	- GetId(void) : char - SetId(id : char) : int - InsertCandidate(candidate : Candidate, idx : int) : int - RemoveCandidate(name : String) : Candidate - popCandidate(void) : Candidate - GetCandidate(void) : Candidate
Report The class is responsible for creating media and audit reports		+ Report(OPL) : void + Report(IR) : void + OPLWriteMediaReport(void) : int + OPLWriteAuditReport(void) : int

for the desired election type.		+ OPLWriteResultsToScreen(void) : int + IRWriteMediaReport(void) : int + IRWriteAuditReport(void) : int + IRWriteResultsToScreen(void) : int

5. Component Design

Function	Description
- SetCurrDis(int): void	Stores the index into the candidate list representing which candidate the ballot is for.
- SetId(int): void	Stores the ballot ID
- SetCandidate(String): void	Stores the candidate that the ballot votes for
- GetCurrDis(): int	Returns the index into the candidate list representing which candidate the ballot is for.
- GetInt(): int	Returns the ballot's ID
- GetCandidate(): String	Returns the candidate(s) the ballot votes for
- SetNumBallots(num_ballots : int): void	Stores the count of the total number of ballots candidate(s) receive
- SetCandidateName(name : string): void	Stores the name of the candidate
- SetParty(char): int	Stores the party of the candidate
- GetNumBallots(void): int	Returns the number of vote(s) the candidate(s) receive
- GetCandidateName(void): string	Returns the candidate(s) name
- GetParty(void): char	Returns the party of the candidate(s)
- RemoveBallotId(id : int): Ballot	Searches through the list of all ballots and removes the ballot with the ID
- ProduceBallot(line : string) : Ballot	Creates a ballot object
- GetPartyType(void): string	Returns a string consisting of the first letter of the party of each candidate in order of their

	appearance on the current line of the CSV
- GetLine(void): string	Returns the string of characters on the next line of the CSV file (increments each time it is called)
- OpenFile(fileName : string): fstream	Opens a CSV file to be read by the program
- CloseFile(fileStream : fstream): void	Closes the CSV file
GetElectionType(void): String	Returns the type of election (open party listing or instant runoff)

- SetElectionType(electionType : string): String	Stores the election type
- GetNumberOfCandidates(void): int	Returns the number of candidates running
- SetNumberOfCandidates(num_of_cand : int): int	Stores the number of candidates running
- IncrementNumberOfCandidates(num_of_cand : int): int	Returns the number of candidates and increments the number of candidates
- GetNumberOfBallots(void): int	Returns the number of ballots in the election
- SetNumberOfBallots(num_of_ballots : int): int	Stores the number of ballots in the election
- IncrementNumberOfBallots(void): int	Returns the numbers of ballots in the election and increments the number of ballots in the election
- ResolveTie(num_candidates : int): int	Returns a random number from 1 to num_candidates, indicating which candidate will be chosen as the winner
- ComputeIRElection(void) : int	Computes the results of the IR election, returning 0 on success and 1 on failure
- DistributeBallots(void) : int	Redistribute ballots to candidates that remain in election
- InsertCandidate(candidate : Candidate, idx : int) : int	Returns a 0 on success of adding a candidate and a 1 on failure
- RemoveCandidate(candidate : Candidate) : int	Returns a 0 on success of removing a candidate and a 1 on failure

- AddParty(party : Party) : Party	Returns party added to election
- RemoveParty(ch : char) : int	Returns a 0 on success of removing a party and a 1 on failure
- InsertCandidateIntoParty(party : Party, candidate : Candidate, idx : int) : int	Returns a 0 on success of adding a candidate to a party and a 1 on failure
- RemoveCandidate(name : String, party : char) : Candidate	Returns candidate removed
- ComputeQuota(void) : int	Returns the quota
- ComputeOPLElection(void): int	Determines the election results by calling ComputeQuota, followed by a loop which accesses GetVotesForParty(party) , GetFirstAllocationOfSeats(party), GetRemainingVotes(party), GetSecondAllocationOfSeats(party), GetRemainingVotes(party), and GetFinalAllocationOfSeats(party) for every party.
- GetVotesForParty(party : char) : int	Returns the number of votes for the party(s)
- GetFirstAllocationOfSeats(party : char) : int	Returns the number of seats a party gets in first seat allocation

- GetSecondAllocationOfSeats(party : char) : int	Returns the number of seats a party gets in second seat allocation
- GetFinalAllocationOfSeats(party : char) : int	Returns the number of seats a party gets in final seat allocation
- GetRemainingVotes(party : char) : int	Returns the remaining number of votes left from previous seat allocation
- GetId(void) : char	returns the first letter of the party name, which is used to identify the party in other functions
- SetId(id : char) : int	sets the id of the party to a letter, used to resolve cases where multiple parties have the same first letter
- InsertCandidate(candidate : Candidate, idx : int) : int	Adds a candidate to a party, returning a 0 for success or 1 for failure
- RemoveCandidate(name : String) :	Removes a candidate from a party and

Candidate	returns that candidate
- popCandidate(void) : Candidate	Removes the first party in the candidates list from a party and returns it
- GetCandidate(void) : Candidate	Returns all the candidates in the party (we should update the type for this)
+ Report(OPL) : void	Calls OPLWriteMediaReport(), OPLWriteAuditReport(), and OPLWriteResultsToScreen() to return all election information
+ Report(IR) : void	Calls IRWriteMediaReport(), IRWriteAuditReport(), and IRWriteResultsToScreen() to return all election information
+ OPLWriteMediaReport(void) : int	Saves the instant runoff election results to a media report file, returning a 0 for success or 1 for failure
+ OPLWriteAuditReport(void) : int	Saves the open party election results to an audit file, returning a 0 for success or 1 for failure
+ OPLWriteResultsToScreen(void) : int	Prints the results of an open party election to the terminal, returning a 0 for success or 1 for failure
+ IRWriteMediaReport(void) : int	Saves the instant runoff election results to a media report file, returning a 0 for success or 1 for failure
+ IRWriteAuditReport(void) : int	Saves the instant runoff election results to an audit file, returning a 0 for success or 1 for failure
+ IRWriteResultsToScreen(void) : int	Prints the results of an instant runoff election to the terminal, returning a 0 for success or 1 for failure

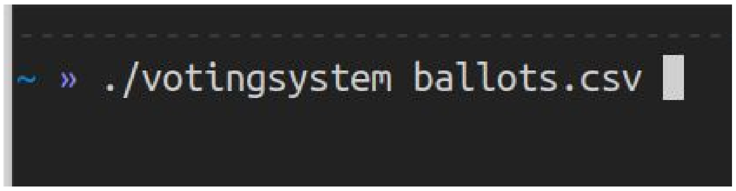
6. Human Interface Design

6.1 Overview of User Interface

The user interface will use the bash terminal. A lot of the user's interaction with the Voting System will be outside the scope of the program. They will create and modify the CSV input file using a tool of their choice, whether that's Excel, Google Sheets, or some command line tool. Various text editors will be able to read and modify the generated audit and report file.

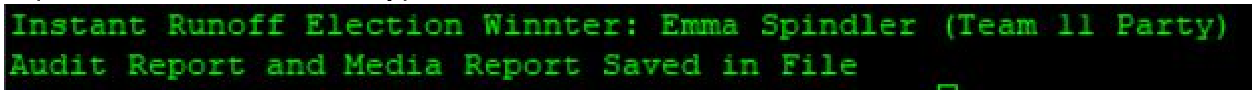
6.2 Screen Images

The user will be able to run the program using textual input seen below.

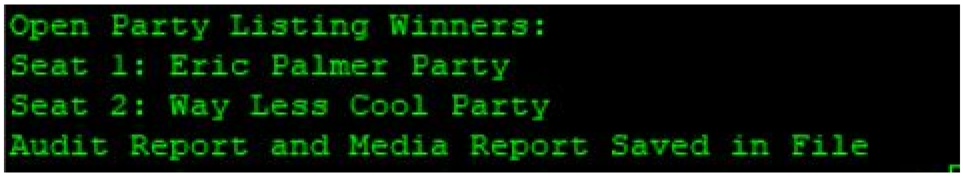


```
~ » ./votingsystem ballots.csv
```

The following results will be displayed once the Voting System Finishes Running dependent on the election type.



```
Instant Runoff Election Winnter: Emma Spindler (Team 11 Party)
Audit Report and Media Report Saved in File
```



```
Open Party Listing Winners:
Seat 1: Eric Palmer Party
Seat 2: Way Less Cool Party
Audit Report and Media Report Saved in File
```

6.3 Screen Objects and Actions

The only object on the screen will be the command line, which can be interacted in by typing the name of the program executable followed by a space and the path to a csv file of appropriately formatted voting information.

7. Requirements Matrix

Use Case	Relevant System Components
Coin Flip (CER_001)	- ResolveTie(num_candidates : int): int
Store Voting Information (ALBF_001)	+ Report(OPL) : void + Report(IR) : void + OPLWriteMediaReport(void) : int + OPLWriteAuditReport(void) : int + IRWriteMediaReport(void) : int + IRWriteAuditReport(void) : int
Program Execution (CER_002)	terminal
Produce Audit Report (PER_001)	+ OPLWriteAuditReport(void) : int + IRWriteAuditReport(void) : int

Compile Program (MISC_001)	terminal
Produce Media Report (PER_002)	+ IRWriteMediaReport(void) : int + OPLWriteMediaReport(void) : int
Display Election Information in Terminal (PER_003)	+ OPLWriteResultsToScreen(void) : int + IRWriteResultsToScreen(void) : int
Sharing Election Results with Media Officials (PER_004)	+ IRWriteMediaReport(void) : int + OPLWriteMediaReport(void) : int
Load CSV File (ALBF_002)	- OpenFile(fileName : string): fstream - GetLine(): string - GetPartyType(void): string - ProduceBallot(line : string) : Ballot
Report Filename Convention (PER_005)	+ Report(OPL) : void + Report(IR) : void + OPLWriteMediaReport(void) : int + OPLWriteAuditReport(void) : int + IRWriteMediaReport(void) : int + IRWriteAuditReport(void) : int
Determining Election Type (CER_003)	- GetElectionType(void): String - SetElectionType(electionType : string): String

8. Appendices

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