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[1] for the full IEEE Recommended Practice for Software Design Descriptions.

[1] http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieeeSDD.pdf

Team 3

Election Voting System

Software Design Document

Name(s): Jacynda Alatoma, Naren Nandyal, Ananya Vegesna, and Long Nguyen

Date: 03/02/2023

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This Software Design Document describes the details and architecture of an Election Voting System for Instant Runoff voting and Closed Party-List voting. The intended audience of this document is election officials, voters, media personnel, developers, and auditors.

1.2 Scope

This document provides a complete description of the design of the Voting System. The scope of this software is to work on any Instant Runoff or Closed Party List voting system. The expected goal is to count ballots/votes and determine clear election winner(s). This includes dealing with any potential issues, such as ties. The benefits of this system include: saving costs, reducing bias and/or election fraud, saving time, and determining a clear election winner.

1.3 Overview

Section 2 of this document is a System Overview. A general description of the functionality, context, and design of this system are given here.

Section 3 introduces the System Architecture and includes an overview of the architectural design. Along with this, it describes the rationale behind our design as well as a decomposition of the subsystems in the architectural design.

Section 4 describes how the data for our system is stored and the data structures that are used. It also describes all the data types that are a part of our system.

Section 5 illustrates all the components of our system and goes into more detail about them.

Section 6 is about the Human Interface Design of our system and provides an overview as well as examples of the system's User Interface.

Section 7 provides a Requirements Matrix that displays which system components satisfy each functional requirement from our SRS.

Section 8 is an Appendix.

1.4 Reference Material

[IEEE] The applicable IEEE Standards are published in "IEEE Standards Collection," 2001 Edition.

1.5 Definitions and Acronyms

UML: Unified Modeling Language

IR: Instant Runoff

CPL: Closed Party List

OOP: Object Oriented Programming

SRS: Software Requirement Specification

SDD: Software Design Document

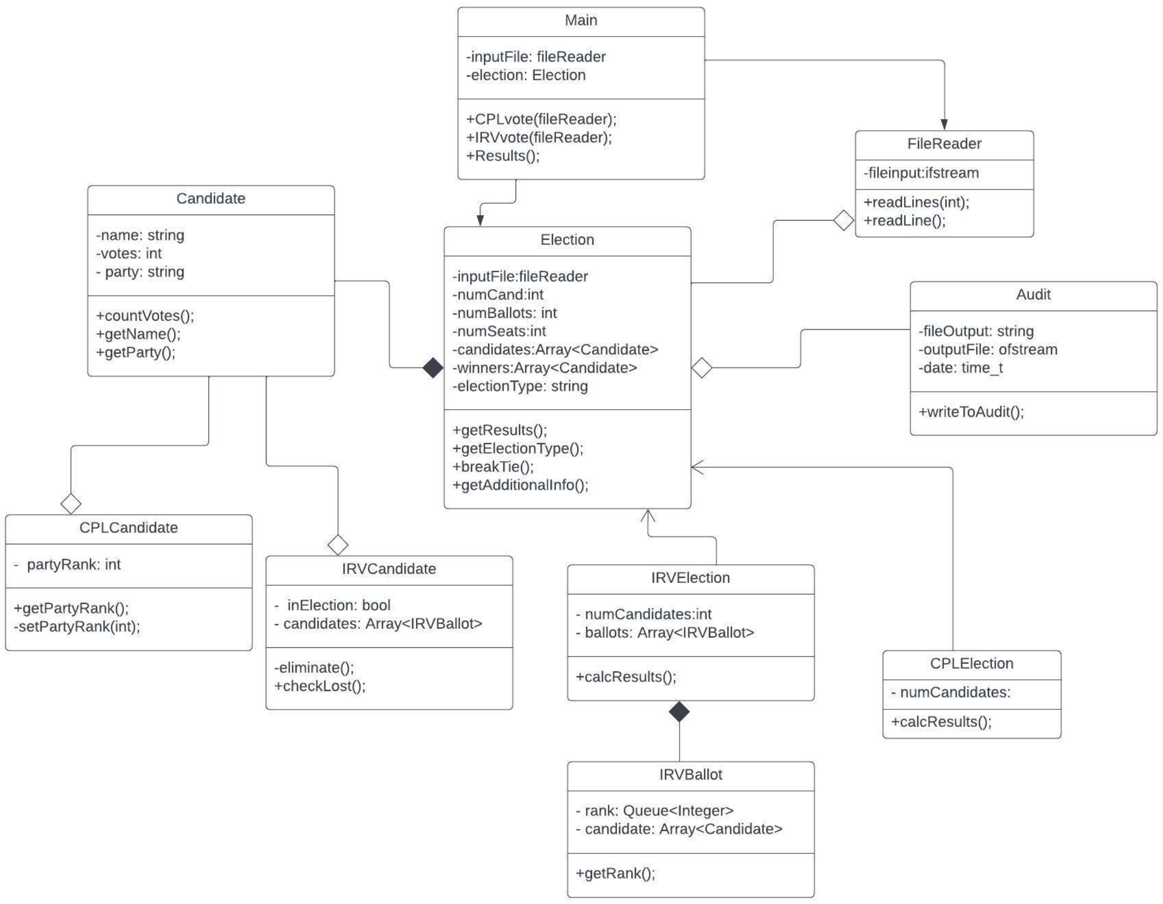
2. SYSTEM OVERVIEW

Our voting system works on Instant Runoff and Closed Party List election types. It must take in a file with all the votes as well as an indication of the type of election to be run (the first line of the file). Once this file is received, the election can begin and a winner determined. This system will be programmed in C++.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

Class Diagram - Also added as an additional file (ClassDiagram_Team3.jpeg)



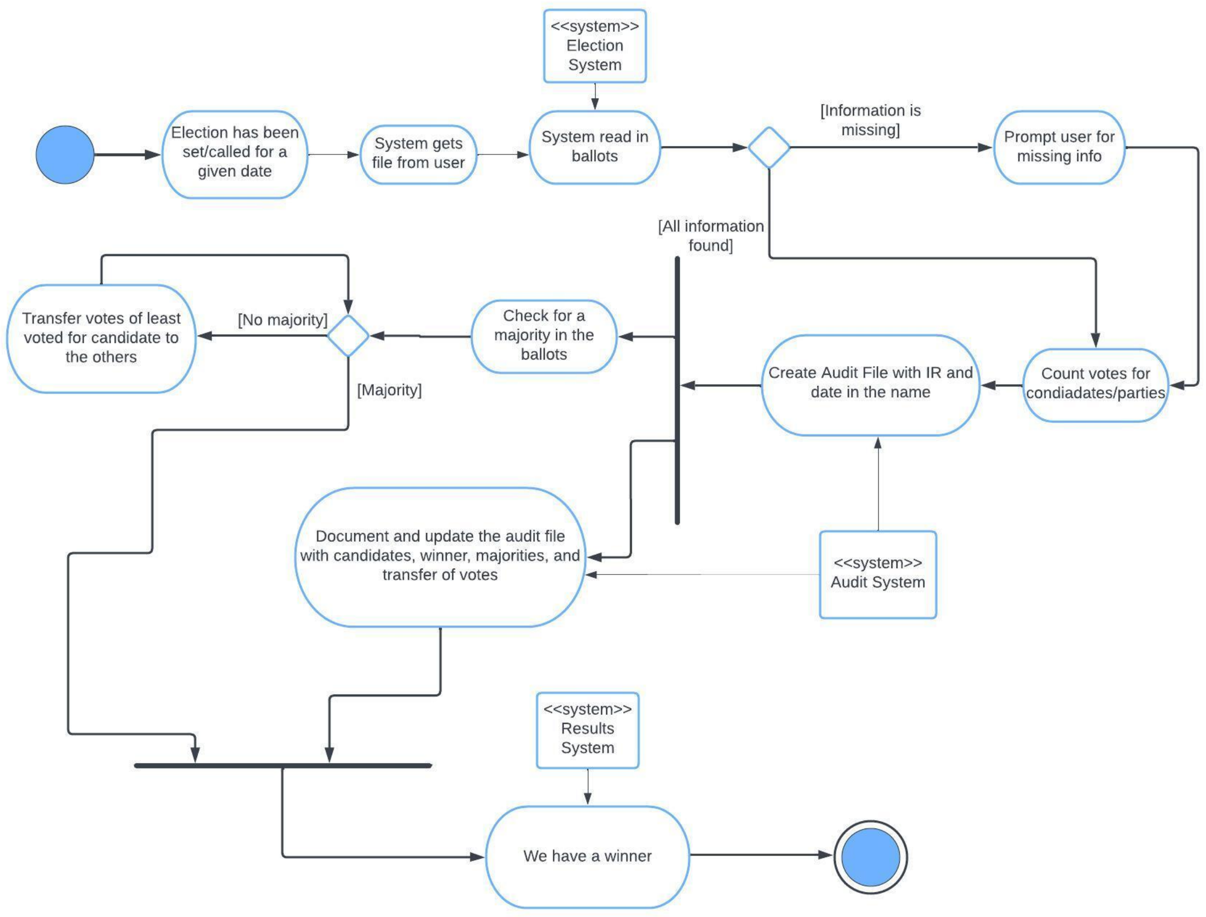
This is the UML class diagram for the system. There are 10 classes total and the details of their function, variables, and attributes will be discussed in the following sections. Here are the relationships between these classes:

- Has-a relationship (denoted by empty fill diamond): Candidate class has a IRVCandidate and has a CPLCandidate. Every Election has an audit file. A file that is read has an election.
- Contains-a relationship (denoted by black filled in diamond): IRVElection contains IRVBallots. An election always contains candidates.
- Is-a relationship (denoted by 2 line arrow head): IRVElection and CPLElection are both (is-a, singular) type of Election.
- Directed association: Main is directly associated with Election and FileReader since it calls on both of those classes to run an election.

3.2 Decomposition Description

UML Activity Diagram (IR)

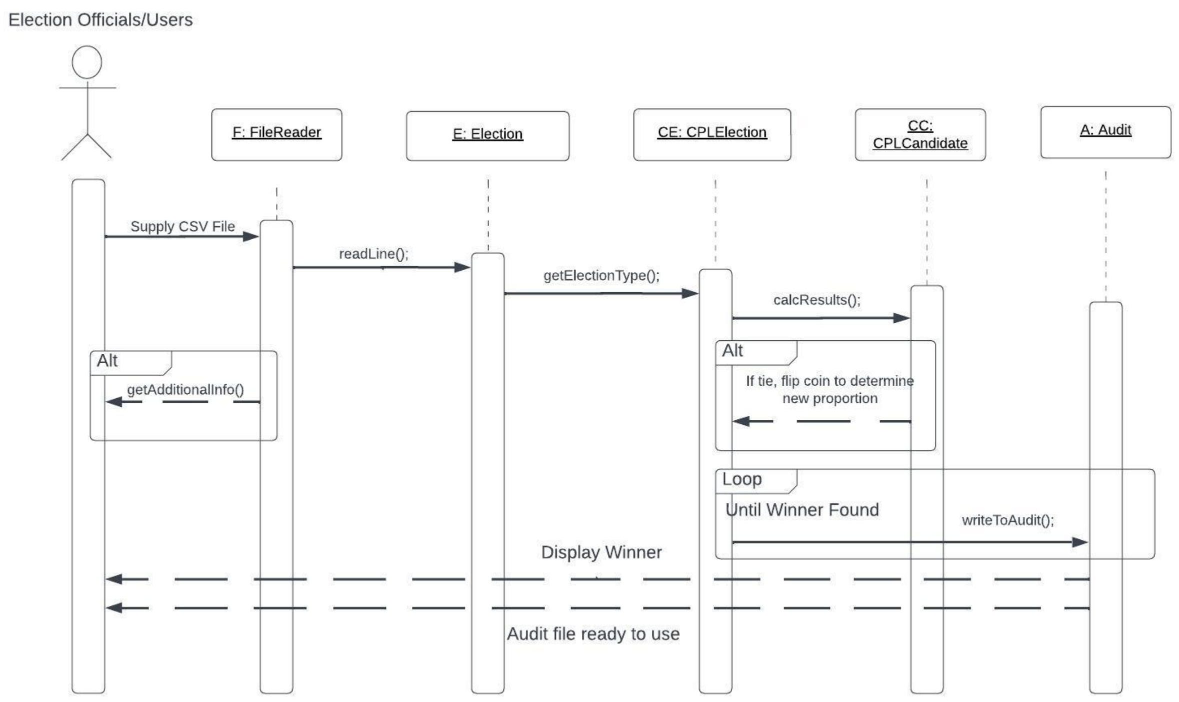
Also added as an additional file (ActivityDiagram_Team3.jpeg)



<<system>> Election System
Election has been set/called for a given date
System gets file from user
System read in ballots
[Information is missing]
Prompt user for missing info
[All information found]
Transfer votes of least voted for candidate to the others
[No majority]
Check for a majority in the ballots
Create Audit File with IR and date in the name
Count votes for condiadates/parties
[Majority]
Document and update the audit file with candidates, winner, majorities, and transfer of votes
<<system>> Audit System
<<system>> Results System
We have a winner

This is the UML activity diagram for an instant runoff election. The diagram starts when an election is called and the system is provided with a CSV file. The ballots work directly with the election system. This system assumes that there is nothing wrong with the CSV file provided, and the program can successfully prompt the user for extra information on the date and type of election (IR) in the filename. If no majority is reached, votes will be transferred to needed. An audit file is created with the name of the type of election and the date as its title. Then the system will count the votes for the candidates and distribute the votes to the second-choice options, and the lowest candidate will be dropped. Documenting all steps is crucial in the audit file, so as this is happening (transfer of votes and selection of winner), everything is also being updated in the audit file. These steps work directly with the auditing system. Once a winner is reached by majority, display the winner to the user and end the system. The display works directly with the results system.

UML Sequence Diagram (CPL)
Also added as an additional file (SequenceDiagram_Team3.jpeg)



This is the UML sequence diagram for a closed party list election. The diagram starts when the election official supplies a CSV file to the system. The system then may either prompt the user for additional information or move on to the next function. Using the FileReader function the CSV file is read into the system. The system then uses the getElectiontype function to determine the type of election. After the election type is found the calcResults function then is used to determine a winner, if there is a tie there is a coin flip to determine the new proportion, and calcResults is run again until a winner is determined. Finally, once the winner is found the writetoAudit function creates an audit file and saves it into the same directory as the CSV file, and gives it to the user.

3.3 Design Rationale

Since we are using C++ which is object-oriented, we wanted to focus on reusability which is why most of the classes have some type of inheritance. One design that was considered was instead of having a “candidate” class we would have a “ballot” class instead, each ballot that was read from the file would’ve been stored in an array of ballots. This wasn’t chosen due to the inefficiency of having to process every ballot in the array every time there isn’t a majority.

4. Data Design

4.1 Data Description

The voting system will first read in the .csv file of ballots. Then, based on the information (i.e. election type), it will create different data structures:

Election: the object that runs the actual election and gets the results.

- IRV Election
- CPL Election

Ballot: this object holds ballot values for IRV; it holds the ranks of the candidates.

Candidate: this object represents the actual candidates in the election; holds information on the party, number of votes, and name.

- IRV Candidate: tells whether the candidate is still in the election and also an array of the candidates.
- CPL Candidate: hold the candidates' rank in their party.

Audit: this object records what goes on in the election process.

4.2 Data Dictionary

Class	Item/Function/Paramter	Type/Return Type	Description
Audit	fileOutput	string	The name of the audit file
	outputFile	ofstream	The output audit file object
	date	time_t	From ctime library, for date of the election
	writeToAudit();	void	Documents steps to audit file
Candidate	name	string	The name of a candidate
	votes	int	Votes a candidate has
	party	string	Candidate’s party
	countVotes();	int	Counts a candidate’s votes
	getName();	string	Returns the name of the candidate
	getParty();	string	Returns a candidate’s party
CPLCandidate	partyRank	int	Rank of candidate within their party
	getPartyRank();	int	Returns a candidate’s rank in their party
	setPartyRank(int);	void	Set a candidate’s rank in their party
CPLElection	numCandidates	int	The number of candidates in the election

	calcResults()	Array<Candidate>	Store the winner(s) of election
Election	inputFile	fileReader	The input file with the election data needed to count votes and run an election
	numCand	int	The number of candidates in an election
	numBallots	int	The number of ballots to be counted in the election
	numSeats	int	The number of seats a party gets from an election (CPL)
	candidates	Array<Candidate>	An array of type Candidates, which holds all the candidates in the election
	winners	Array<Candidate>	An array of Candidates which lists the election winner(s)
	electionType	string	The type of election being conducted (IRV vs. CPL)
	getResults();	Array<Candidate>	Returns an array of the winner(s)
	getElectionType();	string	Returns the type of election being run (IRV vs. CPL).
	breakTie();	void	Tiebreaker function to resolve any ties and reach a winner.
	getAdditionalInfo();	void	Collects additional information from a user if needed.
FileReader	fileInput	ifstream	The pointer to the election file being read

	fileAudit	string	The name of the audit file
	readLines(int);	void	Read specific lines from a file
	readLine();	void	Read the current line of the file
	voteToAudit();	void	Writes all current changes and updates of the election to an audit file
IRVBallot	rank	Queue<Integer>	A queue of the candidate rankings
	candidate	Array<Candidate>	An array of the candidates.
	getRank(int);	int	Returns the rank of a candidate.
IRVCandidate	inElection	bool	Boolean representing whether the candidate has been eliminated
	candidates	Array<IRVBallot>	An array of the candidates
	eliminate();	void	Eliminate a candidate from the election
	checkLost();	boolean	Returns a boolean representing whether a candidate is out of the election
IRVElection	numCandidates	int	The number of candidates in this IRV election
	ballots	Array<IRVBallot>	The ballots of candidates and votes
	calcResults()	Array<Candidate>	Store the winner
Main	inputFile	fileReader	The input file that holds the election information
	election	Election	The election that is taking

			place.
	CPLVote(fileReader);	void	Conducts CPL election
	IRVVote(fileReader);	void	Conducts IRV election
	Results();	Array<Candidate>	Returns the winner(s) of an election

5. Component Design

In this section, we take a closer look at what each component of our ballot system does in a more systematic way. We provide some pseudocode/descriptions for each of the functions included in our classes.

5.1 Audit

```
writeToAudit() {
    // writes contents of election to a file
    // use ctime class to get the day
    date = time_now
    election = getElectionType()
    make a string of election type and time
    create point to ofstream file and write to it
}
```

5.2 Candidate

```
countVotes() {
    // count the number of votes a candidate has and return an integer value.
    initialize voteCounter
    loop through ballots
    update voteCounter as looping
    return voteCounter
}

getName() {
    // get and return a string representing a Candidate's name
    return name
}

getParty() {
    // get and return a string stating a candidate's political party.
    return party
}
```

5.3 CPL Candidate

```
getPartyRank() {
    // returns a candidate's rank within their party
    return rank
}

setPartyRank(int rank) {
    // set a candidate's rank in their party
    candidate.PartyRank = rank
}
```

5.4 CPLElection

```
calcResults() {
    // calculate the results of a CPL election.

}
```

5.5 Election

```
calcResults() {
    // calculate the election results
}
```

```
getResults() {
    // returns an array of the winner(s)
    return winners
}

getElectionType() {
    // returns the type of election that is being run
    return electionType
}
breakTie() {
    // resolves any ties in order to reach a winner
    remove last member from queue with candidate rankings
    redistribute candidate's votes to other candidates
    check whether there is now a winner
}

getAdditionalInfo() {
    // collects and additional information from a user, if needed
    println("Please provide additional information: ")
    // store info dpeneding on what the user enters (if statement)
}
```

5.6 FileReader

```
readLines(int) {
    // read lines specified by the parameter
    // may call ballot counters or store values here
}

readLine() {
    // reads the first line of the file, determines the election
    electionType = from the file
}
```

5.7 IRVBallot

```
getRank() {
    // this returns the rank of the candidate
    return rank
}
```

5.8 IRVCandidate

```
eliminate() {
    // eliminate a candidate from the election
    remove last candidate from ranking array
}
checkLost() {
    // Returns a boolean if candidate is in the election
    return inElection
}
```

5.9 IRVElection

```
calcResults() {
    // Stores the winner and returns them
}
```

5.10 Main

```
CPLVote(fileReader) {
    // calls and conducts CPL election
    // determines a winner
}

IRVote(fileReader) {
    // calls and conducts IR election
    // determines a winner
}

Results() {
    // returns the results of the election
    return winner(s)
}
```

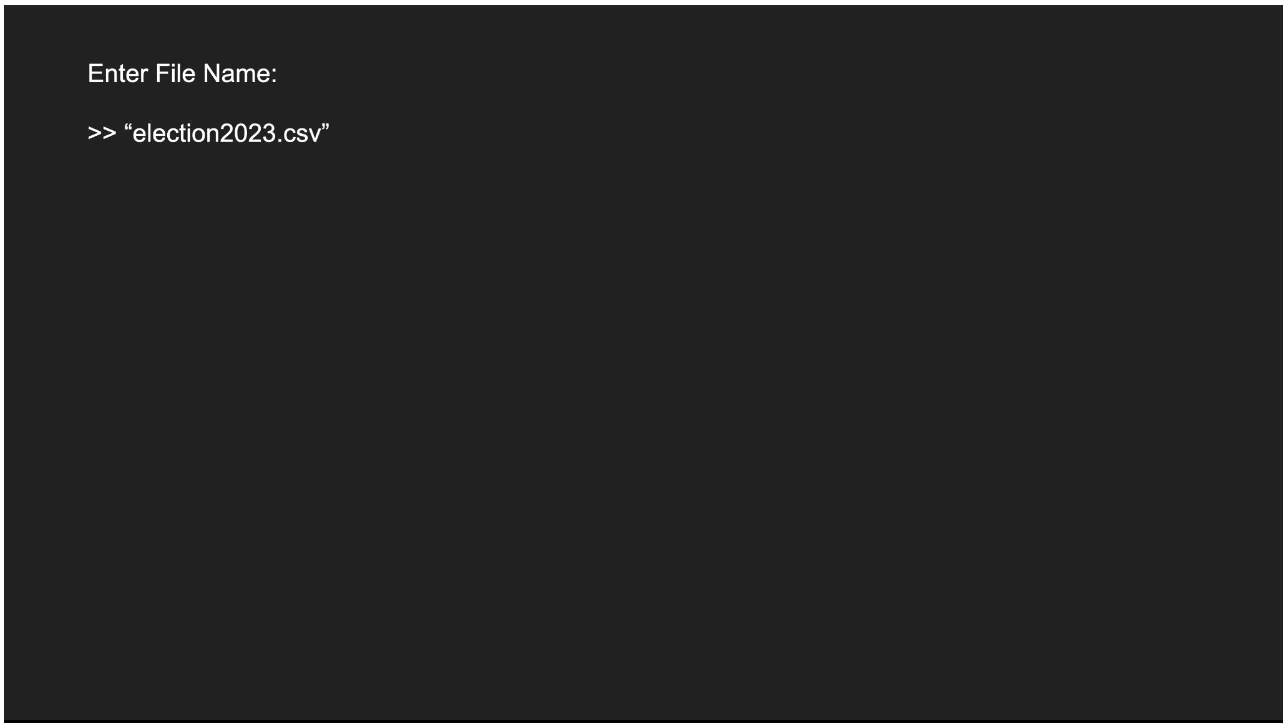
6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The terminal will prompt the user to enter a file and then after the filename is entered, the screen will either display a text that prompts the user for more information if needed or text that displays the winner of the election along with other election information like the number of ballots cast, the winners and the stats for everyone such as the number of votes received, the percentage of votes received, the winner(s), etc. An audit file will then be created and saved to the same directory.

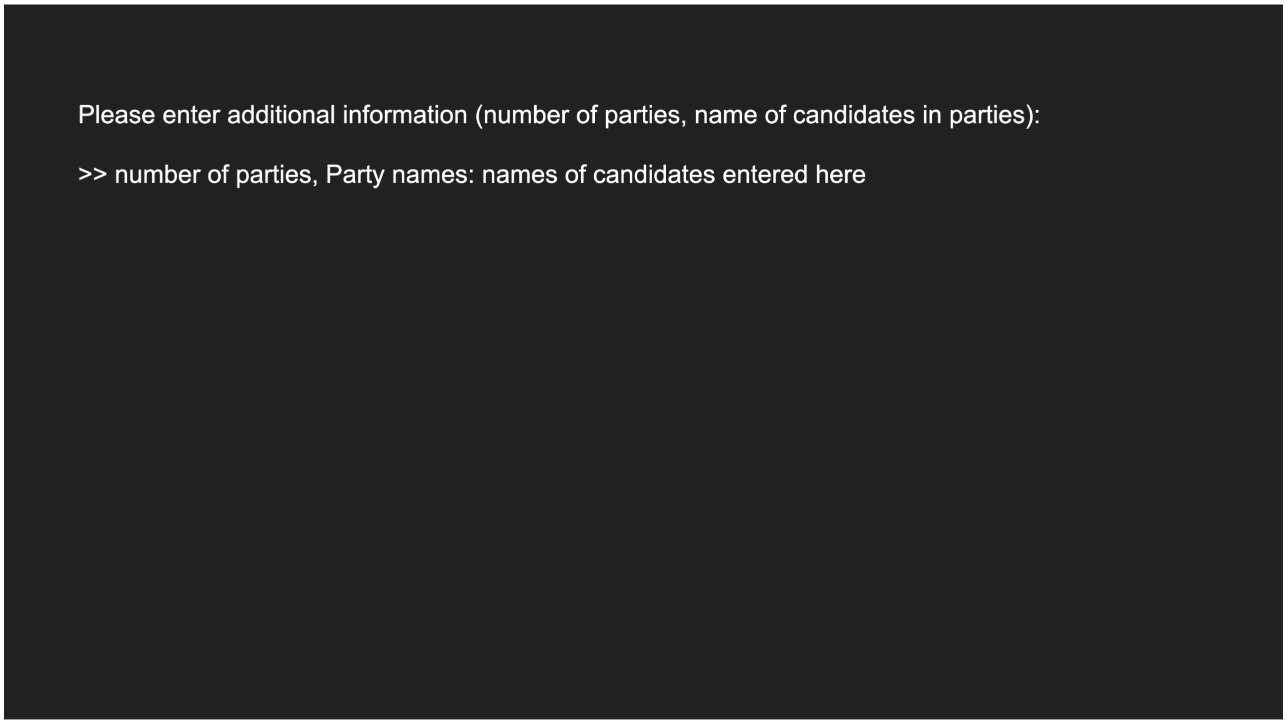
6.2 Screen Images

First screen that prompts for the election file



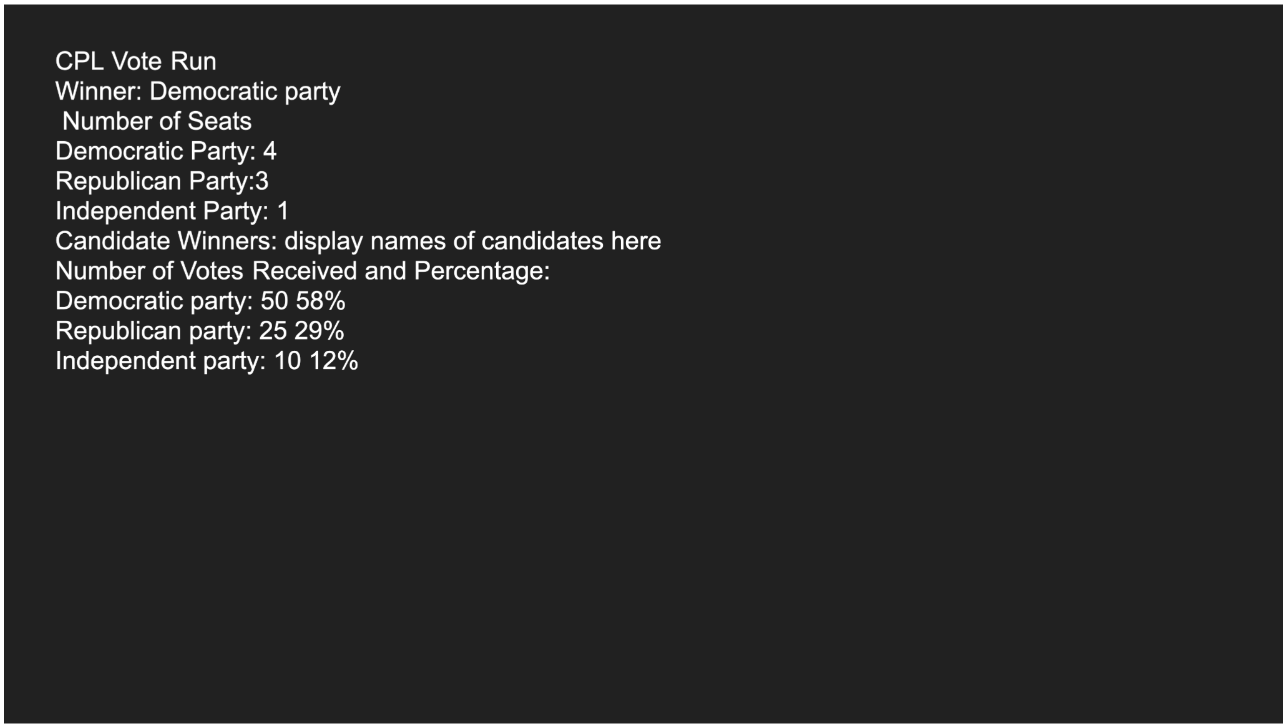
```
Enter File Name:

>> “election2023.csv”
```

Screen that is displayed if additional information is needed



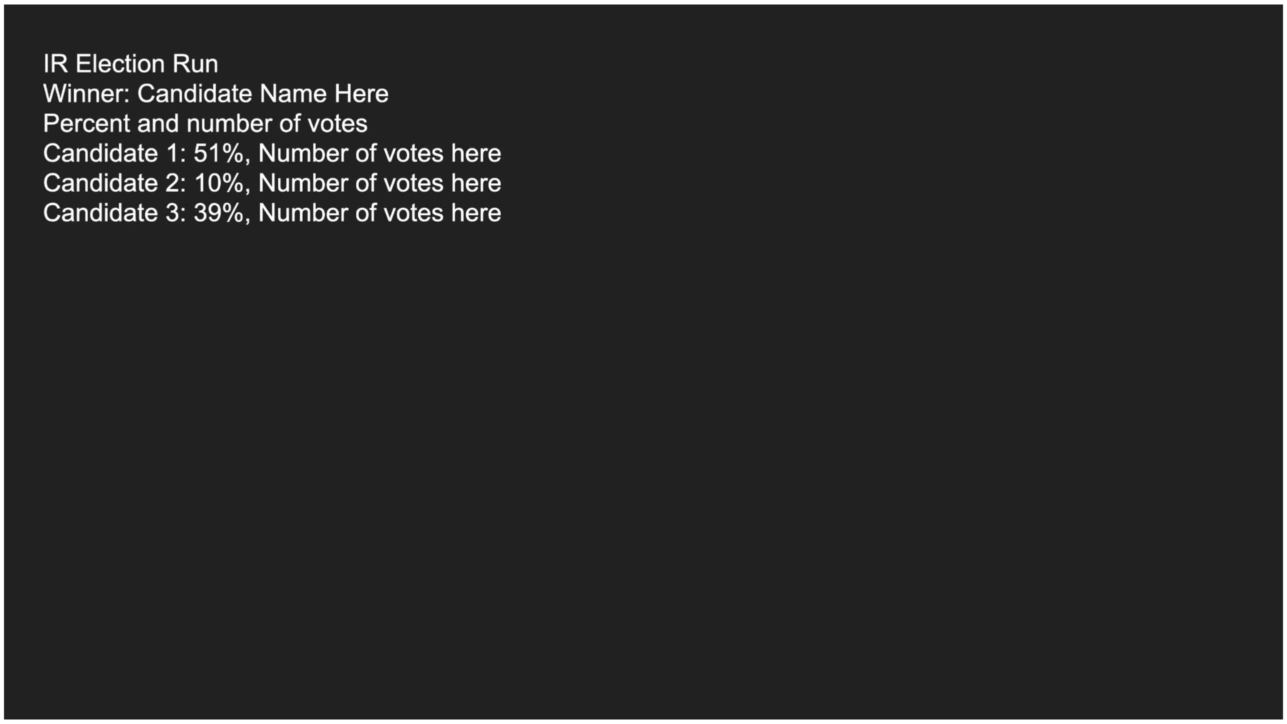
```
Please enter additional information (number of parties, name of candidates in parties):

>> number of parties, Party names: names of candidates entered here
```

Screen that is displays results after CPL election is run



```
CPL Vote Run
Winner: Democratic party
 Number of Seats
Democratic Party: 4
Republican Party:3
Independent Party: 1
Candidate Winners: display names of candidates here
Number of Votes Received and Percentage:
Democratic party: 50 58%
Republican party: 25 29%
Independent party: 10 12%
```

Screen that displays results after IR election is run



```
IR Election Run
Winner: Candidate Name Here
Percent and number of votes
Candidate 1: 51%, Number of votes here
Candidate 2: 10%, Number of votes here
Candidate 3: 39%, Number of votes here
```

6.3 Screen Objects and Actions

Only text is used to prompt users to enter a filename and add additional information if needed. Text is also used to display the results of the elections. No additional graphics are used.

7. Requirements Matrix

This requirements matrix will reference information from the SRS labeled SRS_Team3 and UseCases_Team3.

Requirement ID	Description	System Component/Function
UC_001	System will break a tie between 2 candidates/parties and will select a random winner	breakTie()
UC_002	System will create an audit of election	VoteToAudit()
UC_003	System will display winners at the end of election	results()
UC_004	System will prompt for additional information	getAdditionalInfo()
UC_005	System will run CPL election	CPLvote()
UC_006	System will run IRV election	IRVvote()
UC_007	System will name audit file based off date and election	VoteToAudit()
UC_008	System will identify CSV file	inputFile:fileReader
UC_009	System will process header of CSV file	readLines(int)
UC_010	System will process ballots of CSV file	readLine()