Team 21

Voting System

Software Design Document

Sean Beaulieu	beaul116
Ann Huynh	huynh441
Shey Pemberton	pembe020
Jasper Rutherford	ruthe124

University of Minnesota
CSCI 5801

March 2022

1. INTRODUCTION

1.1 Purpose
This software design document describes the architecture and system design for Voting System. The expected audience is the programmers and testers who will create and maintain the Voting System.

1.2 Scope
This document describes the design of the Voting System. The Voting System uses two types of voting algorithms: Instant Runoff Voting and Party List Voting using Open Party List. This system will be used multiple times a year for local and general elections. The long-term goal is for this system to become part of an integrated online voting system, in which the preprocessing will be done by the system described in this document.

1.3 Overview
This document contains an introduction, system overview, system architecture, data design, component design, human interface design, requirements matrix, and appendices.

System overview: A general description of the functionality, context and design of the system.
System architecture: Specifies design components that work together to perform all of the functions in the system.
Data design: Describes data structure entities and their organizations within the system.
Component design: Describes components in a systematic way, including pseudocode.
Human interface design: Describes the functionality of the system from a user's perspective and discusses screen images, objects and actions.
Requirements matrix: Provides cross references that traces components and data structures to the functional requirements described in the software requirements specification document.

1.4 Reference Material
IEEE Template for Software Design Documents
Project 1 Waterfall Methodology SRS Document for Voting System
    Provided by Shana Watters for the 5801 Software Engineering course at UMN
Project 1 Waterfall Methodology SDD Document Template
    Provided by Shana Watters for the 5801 Software Engineering course at UMN
SRS for Voting System
    Created by Team 21 for the 5801 Software Engineering course at UMN

1.5 Definitions and Acronyms

Term	Definition
CSELabs	College of Science and Engineering Computer labs at UMN
CSV	Comma Separated Value
IEEE	Institute of Electrical and Electronic Engineers
IRV	Instant Runoff Voting
OO	Object Oriented
OPL	Open Party Listing
UMN	University of Minnesota

2. SYSTEM OVERVIEW

This product is an implementation of existing voting algorithms, Instant Runoff Voting and Party List Voting with Open Parties, that are contained in one program with the ability to process ballots designed for either voting algorithms. This is a new, self-contained product; the long-term goal is for this product's voting system to be integrated into an online voting system.

- The user enters the name of a CSV file containing ballot data into the command line.
    - The actual file containing ballot data is procured by a separate Ballot Handling system unrelated to the system described in this document.
    - This file must be stored in the same directory as the Voting System program.
- The user may enable or disable shuffling of ballots by entering the appropriate command into the command line.
    - Shuffling is enabled by default for IRV type elections.
    - Shuffling is unused for OPL type elections.
- System reads in ballots from the input file and processes them according to election type.
    - The first line of the input file contains information about the election type as well as information as necessary about the parties and candidates.
- System produces an audit file using the File Designator object.
- System produces a report file.
- System will display results of the election on the screen.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

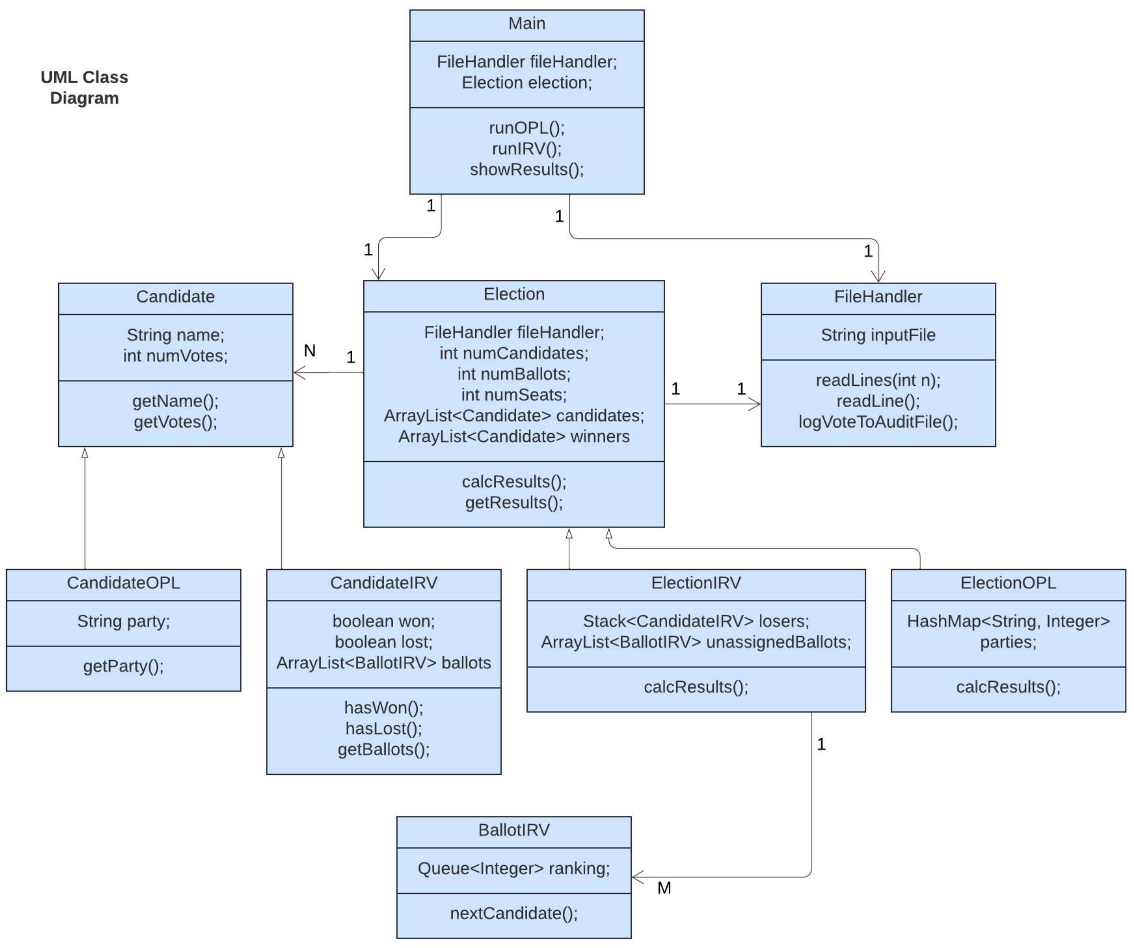
For the Voting System, there are nine primary modules: Main, FileHandler, Election, ElectionOPL, ElectionIRV, Candidate, CandidateOPL, CandidateIRV, and BallotIRV.

- Main is the main program where all user inputs are taken in.
- FileHandler is constructed to handle all file inputs and outputs.
- Election is designed to run/process an election.
- ElectionOPL extends Election and is designed to run/process OPL based elections.
- ElectionIRV extends Election and is designed to run/process IRV based elections.
- Candidate is used to represent a candidate in any election.
- CandidateOPL is used to represent a candidate in an OPL based election.
- CandidateIRV is used to represent a candidate in an IRV based election.
- BallotIRV is used to represent a ballot in an IRV based election.

Main has a FileHandler and an Election. The FileHandler is used to read in initial election information which is used to set up the Election object. The Election object will be either an ElectionOPL or an ElectionIRV, depending on the specified vote method.

Each Election object uses Candidate objects to represent each candidate and track each candidate's vote count. Additionally, each Election object uses the FileHandler to read in ballot information and to report updates to the audit file. If the Election object is an ElectionIRV, then the Election object will use BallotIRV objects to represent each ballot and the order in which candidates have been selected on that ballot.

Figure 3.1.1: A UML class diagram represents the architecture as discussed.



3.2 Decomposition Description

The activity process for Instant Runoff Voting is shown in Figure 3.2.2 and is described as follows. First, the file is checked for existence. Next, the file handler gets created and can begin to read in the ballots and the input file. A check to determine whether the ballots get shuffled or not gets created. While there are still unassigned ballots, individual votes get added to the next candidate on the ballot. If that candidate's votes are greater than the assigned votes, their status gets changed to a winner. Once all the unassigned ballots are counted, if the number of winners is greater than zero, then the lowest candidates' votes get assigned to the next candidate. Then, the election is finished.

The activity process for Open Party Listing is shown in Figure 3.2.1 and 3.2.3, and can be described as follows. Firstly, the file is checked for existence. Next, the file handler gets created and can read and initialize the election class. A loop is run to add all of the candidates to the election class. Another loop is run which tallies the votes, assigns the vote to the party and candidate, and also logs the vote into the audit file. After the votes have been tallied, a check is run to determine the party winners of the election. Then, another loop is run which determines which candidate has the most votes in the party, awards them a seat, logs the winner into the audit file, and then removes the candidate from the possible list of winners. Finally, a final check occurs to determine if all the seats have been filled, and if so, the election is finished.

Figure 3.2.1: Sequence Diagram for Open Party Listing

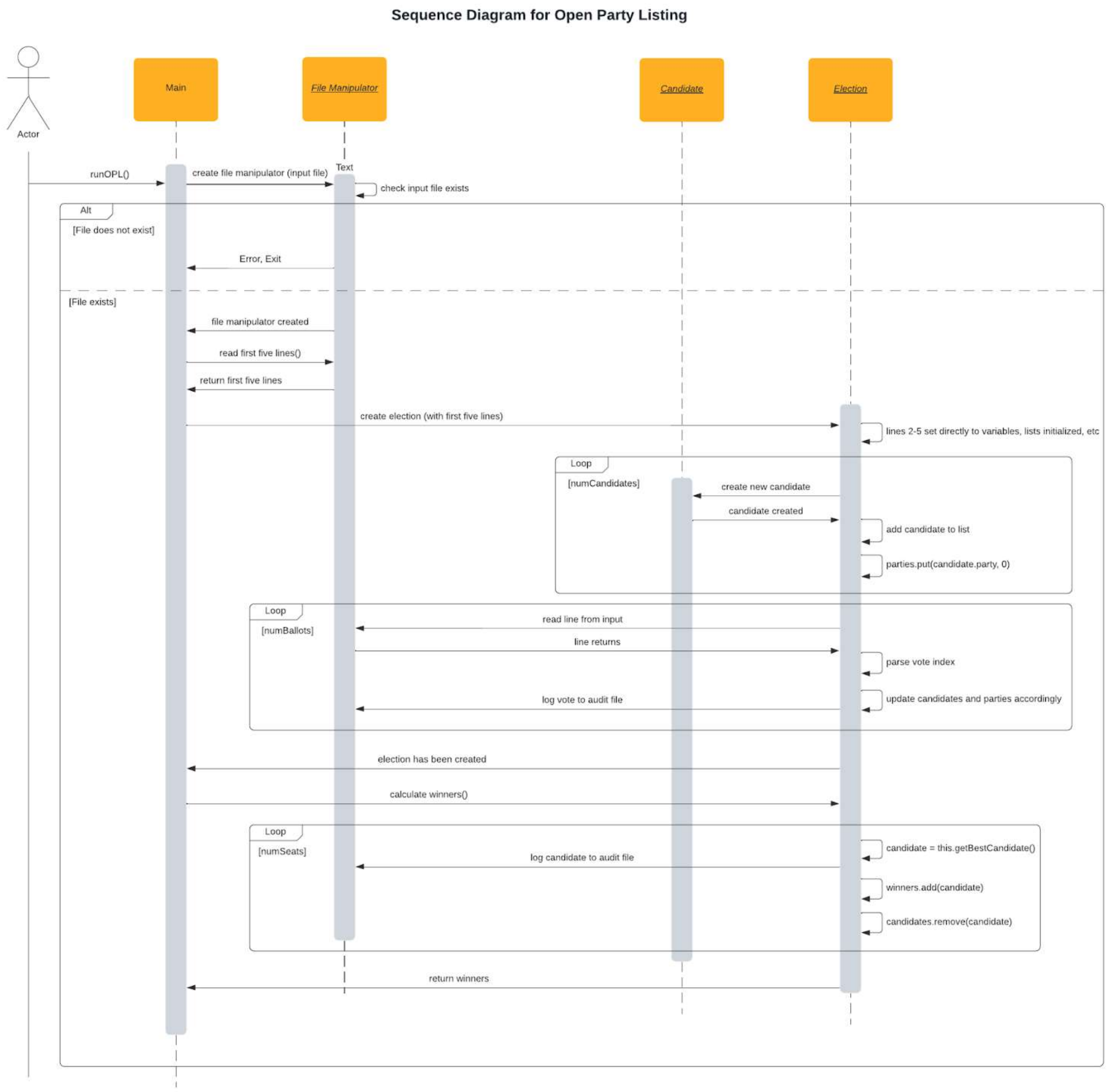


Figure 3.2.2: UML Activity Diagram for IRV

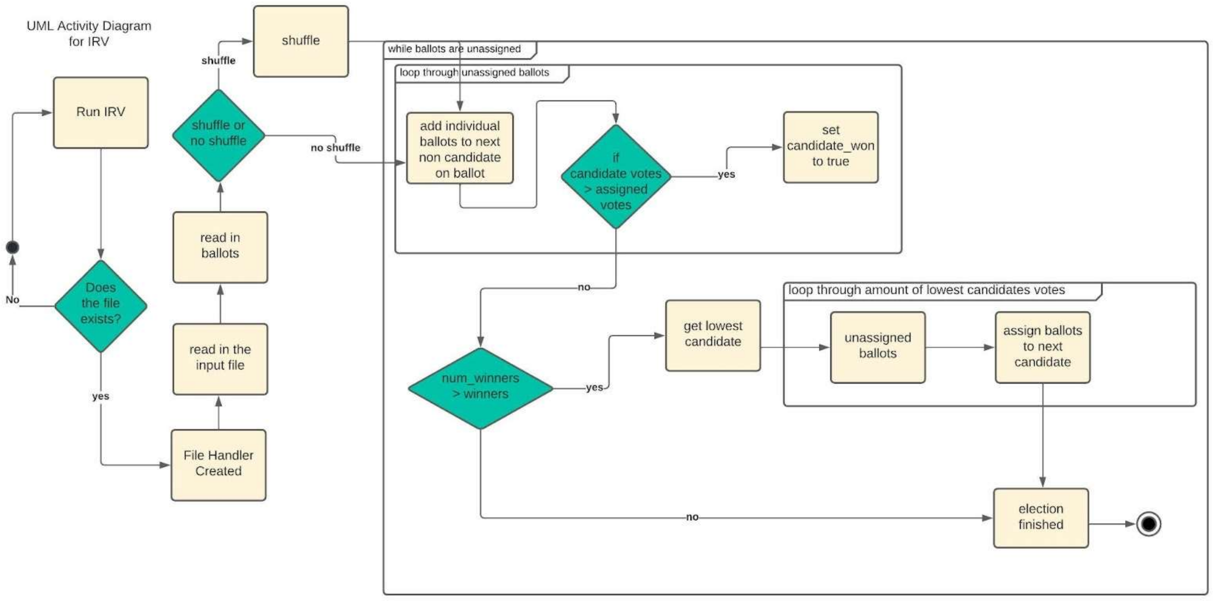
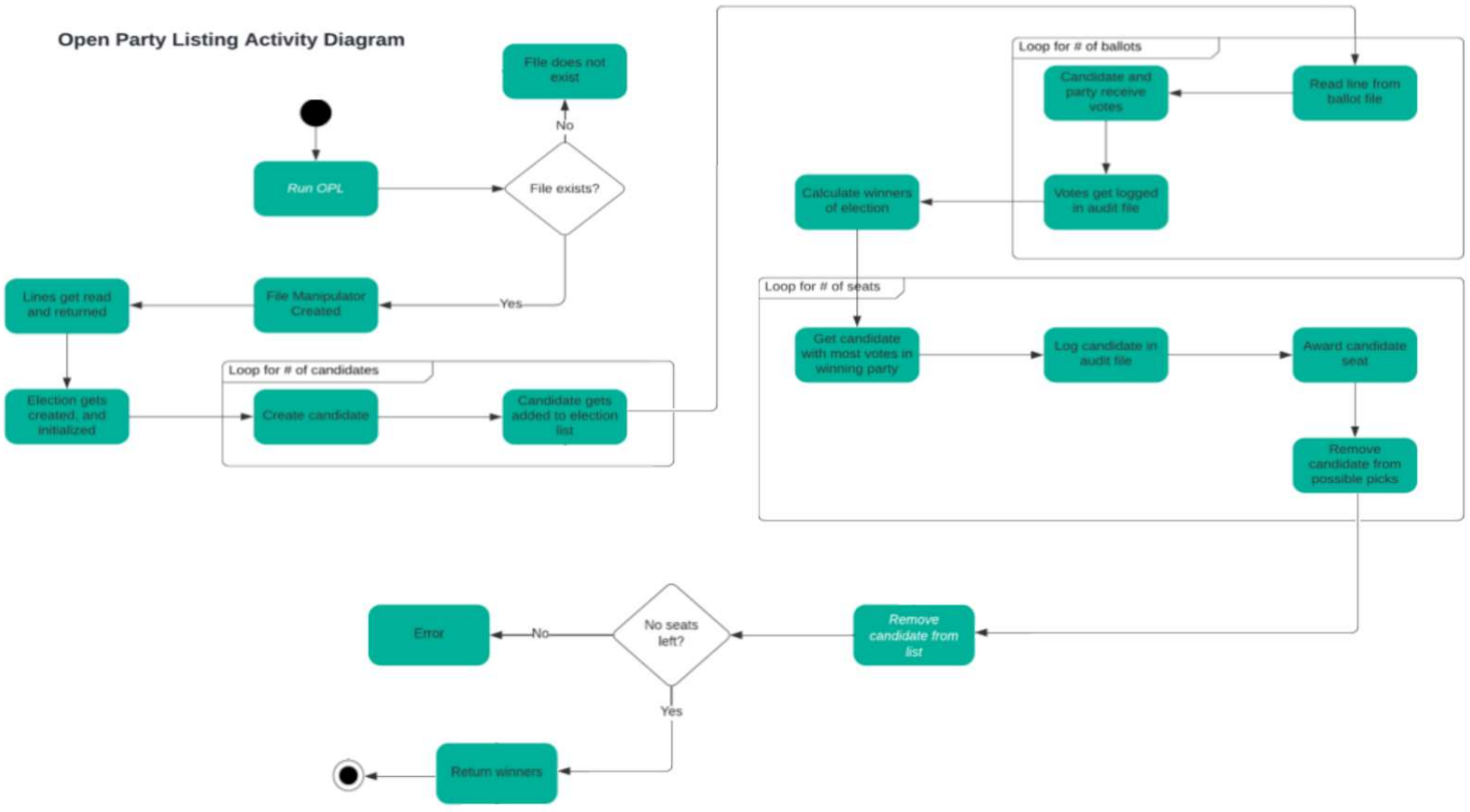

UML Activity Diagram for IRV
Run IRV
Does the file exists?
No
yes
File Handler Created
read in the input file
read in ballots
shuffle or no shuffle
shuffle
no shuffle
shuffle
while ballots are unassigned
loop through unassigned ballots
add individual ballots to next non candidate on ballot
if candidate votes > assigned votes
yes
set candidate_won to true
no
num_winners > winners
yes
get lowest candidate
loop through amount of lowest candidates votes
unassigned ballots
assign ballots to next candidate
no
election finished

Figure 3.2.3: UML Activity Diagram for OPL


Open Party Listing Activity Diagram
Run OPL
File exists?
No
File does not exist
Yes
File Manipulator Created
Lines get read and returned
Election gets created, and initialized
Loop for # of candidates
Create candidate
Candidate gets added to election list
Loop for # of ballots
Read line from ballot file
Candidate and party receive votes
Votes get logged in audit file
Calculate winners of election
Loop for # of seats
Get candidate with most votes in winning party
Log candidate in audit file
Award candidate seat
Remove candidate from possible picks
Remove candidate from list
No seats left?
No
Error
Yes
Return winners

3.3 Design Rationale

Initially, the Voting System architecture contained five main modules: Main, FileHandler, Candidate, Ballot, and Election. Main program took in the user inputs. FileHandler processed file inputs and outputs, which included the input ballot file, the output audit file, and output report file. Candidate would contain child classes which represent candidates for both OPL and IRV elections. Ballot represented ballots for both OPL and IRV election use. Election would hold all the candidates and ballot information, and output file data would be drawn from this module.

After some consideration and further planning, we modified the Candidate, Election, and Ballot architecture.

- Candidate
    - We realized that this would end up behaving differently/tracking things differently for IRV vs OPL, and for clarity/simplicity the decision was made to create a single simpler Candidate object, and split the specialization into CandidateIRV and CandidateOPL.
- Election
    - We realized that this would also end up behaving differently/tracking things differently for IRV vs OPL, and for clarity/simplicity the decision was made to create a single simpler Election object, and split the specialization intoElectionIRV and ElectionOPL.
- Ballot
    - OPL voting does not actually need to reassign votes at any stage, so the decision was made to simply assign votes as they are read in for OPL (and not use the ballot object). Ballot was then renamed to BallotIRV to better match the naming scheme developed for Election/Candidate

4. DATA DESIGN

4.1 Data Description

General Data Organization:

Information	**Where it is stored**	**How it is represented**
Candidate Names	Candidate.name	String
Candidate Votes	Candidate.numVotes (int)	int
The winner(s) of an election	Election.winners	ArrayList<Candidate>
The candidates in an election	Election.candidates	ArrayList<Candidate>
Number of candidates	Election.numCandidates	int
Number of ballots	Election.numBallots	int
Number of seats	Election.numSeats	int

Exclusive OPL Data Organization:

Information	**Where it is stored**	**How it is represented**
Party votes	ElectionOPL.parties.at(party)	int
Each Candidate's Party	CandidateOPL.party	String

Exclusive IRV Data Organization:

Information	**Where it is stored**	**How it is represented**
Each ballot's ranking of candidates	BallotIRV.ranking	Queue<Integer>
The losers of an election	ElectionIRV.losers	Stack<Candidate>

4.2 Data Dictionary

BallotIRV	Object that represents one ballot in an IRV based election
BallotIRV.ranking	Queue of integers that represents the order in which candidates have been ranked on any given ballot in an IRV based election
Candidate	Class that represents a candidate in an election
Candidate.name	A string that represents the candidate’s name in an election
Candidate.numVotes	An int that represents the number of votes attributed to a candidate in an election
CandidateIRV	Object which represents a candidate in an IRV based election.
CandidateIRV.numVotes	Integer which represents the number of votes that this candidate has in an IRV election
CandidateOPL	Object which represents a candidate in an OPL based election.
CandidateOPL.party	A string representing the party of a candidate in an OPL election
Election.candidates	An ArrayList of candidates listed in the election
Election.numBallots	An int representing the total number of ballots in an election
Election.numCandidates	An int representing the number of candidates in an election
Election.numSeats	An int representing the number of seats in an OPL election
Election.winners	An ArrayList of candidate(s) that have won the election
ElectionIRV	An Object which represents an election that follows the IRV voting method
ElectionIRV.losers	Stack of candidates which represents all the candidates who have had their votes redistributed.
ElectionIRV.unassignedBallots	ArrayList of BallotIRV objects which represents all the ballots in the election which have not been assigned to a candidate/which are being redistributed
ElectionOPL	Object which represents an election that follows the OPL voting method
ElectionOPL.parties	A HashMap which maps strings (each string representing a party) to an integer (representing the number of votes attributed to this party).

5. COMPONENT DESIGN

5.1 OPL pseudocode

```
check file->     fail
          success-> start reading file
Try {
   read "OPL"->
          check for OPL-> fail
                              success->
                              read numCandidates -> int candidates = readNumCandidates
                              read numSeats
                              read numBallots
                              read candidates()* -> candidates into listOfCandidates
                              for(numBallots){
                                        int index = read ballot index
                                        candidate = candidates[index]**
                                        candidate.votes++;
                                        parties[candidateParty]++;***
                                        // log to audit file?
                              }
                    }
catch{
   file error -> report this
   exit
}

for(numSeats) {
   candidate = getBestCandidate()
   winnersAdd(candidate)
   candidateWon = true;
}


*
read candidates
   for(numCandidates){
          candidates.add(new candidate)
          partiesPut(candidate.party, 0)
   }

**
candidate char party -> index of this candidate's party
int votes

***
Parties -> hashmap string ->int
Parties[R] -> num votes for R Party
```

5.2 IRV Pseudocode

```
Read ballot file
Check filename -> error
               -> good
->read initial info from ballot file(size, numBallots,etc.)
loop(numBallots){
        Read new ballot to unassigned ballot list
}
if(shuffle){
        Shuffle
}
while (electionNotFinished){
        Loop through unassigned ballots{
                Add ballot to next non won candidate on ballot
                if(no more candidates on ballot){
                        Forget it
                if(that candidate_votes > allowed_votes){
                        Candidate_won = true
                }
        }
        if(num_winners < goal_winners){
                Lowest_candidate = get_lowest_candidate(candidate)
                for(lc_votes){
                        Unassign ballots, advance ballots to next candidate(ballot.queue.pop())
                }
        }
        Else{
                Election finished = true
        }
}
```

6. HUMAN INTERFACE DESIGN

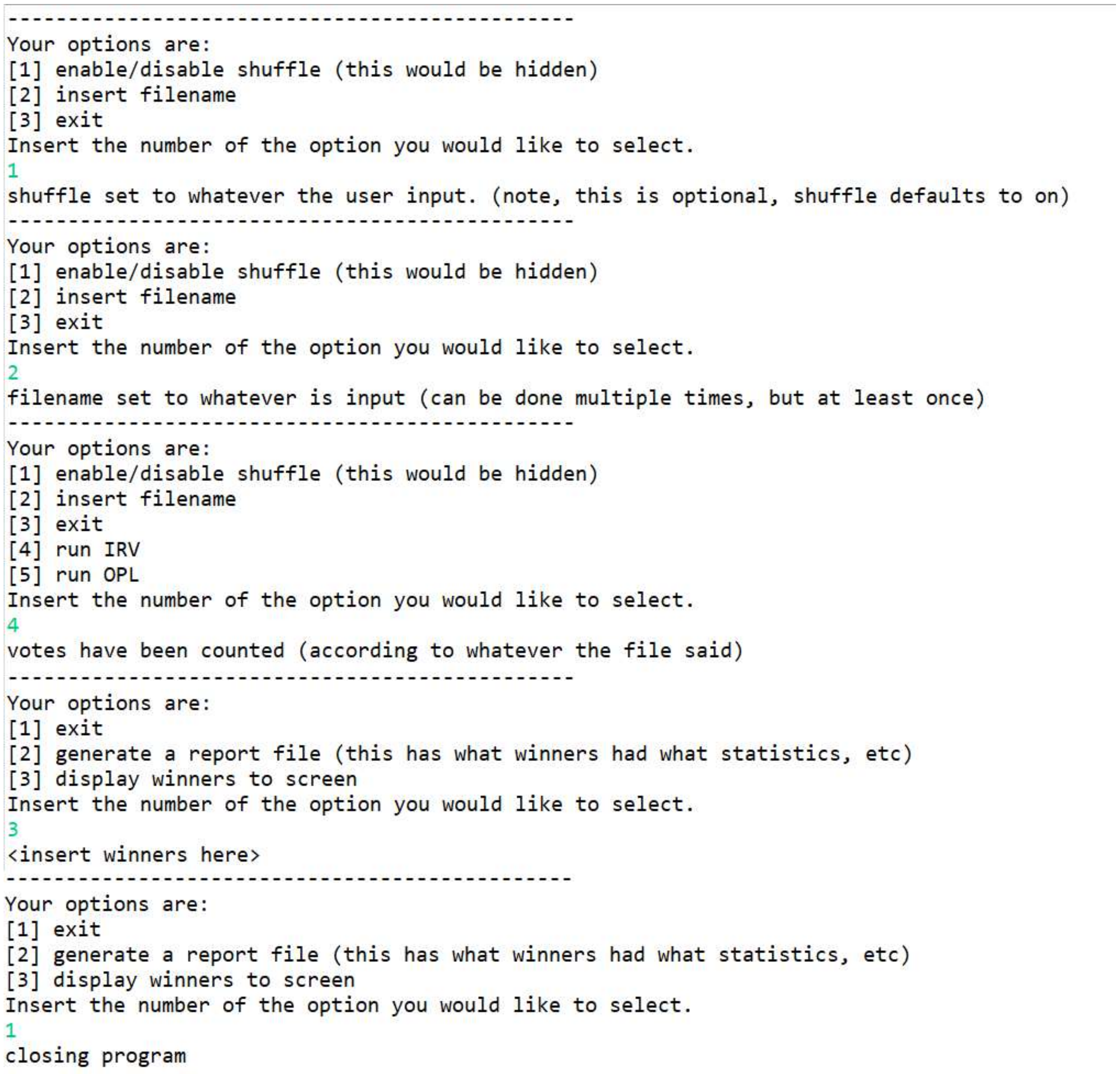
6.1 Overview of User Interface

The user will use the command line interface to enter appropriate commands to achieve desired functionality.

COMMANDS	**FUNCTIONALITY**
fileName	Enter input file name.
runOPL	Run OPL election processing.
runIRV	Run IRV election processing.
shuffle	Toggle shuffling
generateReport	Generates a general report file
displayWinners	Displays any winners to the screen
exit	Exits the program

6.2 Screen Images

Mockup of user interface, shows the order that commands would be available for use (Note, in the real program the available commands will not be listed; the user will need to type the full command). If a command is entered that is not available for use at that time (Ex: inserting filename after vote results have been calculated), then an error message will be displayed.



```
------------------------------------------------
Your options are:
[1] enable/disable shuffle (this would be hidden)
[2] insert filename
[3] exit
Insert the number of the option you would like to select.
1
shuffle set to whatever the user input. (note, this is optional, shuffle defaults to on)
------------------------------------------------
Your options are:
[1] enable/disable shuffle (this would be hidden)
[2] insert filename
[3] exit
Insert the number of the option you would like to select.
2
filename set to whatever is input (can be done multiple times, but at least once)
------------------------------------------------
Your options are:
[1] enable/disable shuffle (this would be hidden)
[2] insert filename
[3] exit
[4] run IRV
[5] run OPL
Insert the number of the option you would like to select.
4
votes have been counted (according to whatever the file said)
------------------------------------------------
Your options are:
[1] exit
[2] generate a report file (this has what winners had what statistics, etc)
[3] display winners to screen
Insert the number of the option you would like to select.
3
<insert winners here>
------------------------------------------------
Your options are:
[1] exit
[2] generate a report file (this has what winners had what statistics, etc)
[3] display winners to screen
Insert the number of the option you would like to select.
1
closing program
```

7. REQUIREMENTS MATRIX

SRS Functional Requirement	**SDD Components and Data Structures**
001 Input Election File Name	IRV/OPL Activity / Class Diagrams
002 Shuffle or Not Shuffle Ballots	IRV Activity Diagram
003 Run Instant Runoff Voting	IRV Activity Diagram
004 Run Open Party Listing Voting	OPL Activity Diagram
005 Coin Flipping	OPL Sequence Diagram
006 Display Winner to Screen	IRV/OPL Activity Diagram
007 Produce Audit File	IRV/OPL Activity / Class Diagrams
008 Produce a Report	IRV/OPL Activity / Class Diagrams