Software Design Description

for

Voting System

Version 1.0 approved

Prepared by Luisa Jimenez Alarcon, Brian Lu, King Yiu Suen, Scott Deyo

Team 12

February 2021

Table of Contents

Introduction **1**
- **Purpose** **1**
- **Scope** **1**
- **Overview** **1**
- **Reference Material** **1**
- **Definitions and Acronyms** **1**

System Overview **2**

System Architecture **3**
- **Architectural Design** **3**
- **Decomposition Description** **5**
- **Design Rationale** **7**

Data Design **7**
- **Data Description** **7**
- **Data Dictionary** **7**

Component Design **9**

Human Interface Design **15**
- **Overview of User Interface** **15**
- **Screen Images** **15**
- **Screen Objects and Actions** **16**

Requirements Matrix **16**

Appendices **16**

1. Introduction

1.1 Purpose

This Software Design Document provides the design details of CSCI5801 Team 12's Voting System Software (VSS).

The expected audience includes programmers, testers, election officials, and election auditors.

1.2 Scope

This document contains a complete description of the design of VSS.

The basic architecture is a program written in C++ that will be interfaced via the command line window.

The VSS will provide relevant and appropriate results for the different audiences -- election officials and election auditors.

1.3 Overview

The remaining chapters and their contents are listed below.

Section 2 covers the System Overview in depth.

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

Section 4 concerns the Data Structure Design.

Section 5 contains the Use Case Realizations. Each Use Case stated in the Software Requirement Specification (SRS) Document can be traced by the given design objects.

Section 6 discusses the User Interface Design, and how it can be created with maximum user efficiency and ease of use.

Section 7 covers the Requirements Matrix.

1.4 Reference Material

CSCI5801 Waterfall Team12 SRS:
https://drive.google.com/file/d/1iJvCeV43RLh1ACMXyfpNVMvX0gCSwVWT/view?usp=sharing

1.5 Definitions and Acronyms

Term	Definition
CSV	Comma-separated values file.
Election officials	An official responsible for the proper and orderly voting at polling stations.

GCC	GNU Compiler Collection.
IR	Instant Runoff.
OPL	Open Party List.
N/A	Not Applicable.

2. System Overview

The voting system software is built as a solution to get one system capable of performing two types of voting. Plurality and proportional type. As for plurality the system will be running IR voting system and as for proportional type OPL voting system will be running.

The system will have the following functionalities:

- Read input filename: Read the filename user inputted via command line interface.
- Bring file to system: Read the input file and the setup of the election according to the pre-specified format.
- Count votes (instant runoff): Count the number of votes for each candidate and decide the winner(s) based on the instant runoff algorithm.
- Count votes (open party listing): Count the number of votes for each candidate and decide the winner(s) based on the open party listing algorithm.
- Resolve a tie: If there is a tie, randomly select a winner based on a fair coin toss.
- Announce results: Display the winner(s) and information about the election to the screen.
- Save audit file: Produce a file that shows how the election progressed so that the audit could replicate the election itself.
- Save media report: Produce a file containing the summary of the election to be shared with media personnel.

The system is built in C++ and expected to run on Ubuntu 20.04, Windows 10, or Mac OS X 10.15. The code is stored in a Github repository and any team member can contribute to the project. As for constraints, it is expected for the system to run once due to the possibilities of getting different results due to the randomness of tossing a coin.

The system shall be used by the election official, programmers or developers and testers.

3. System Architecture

3.1 Architectural Design

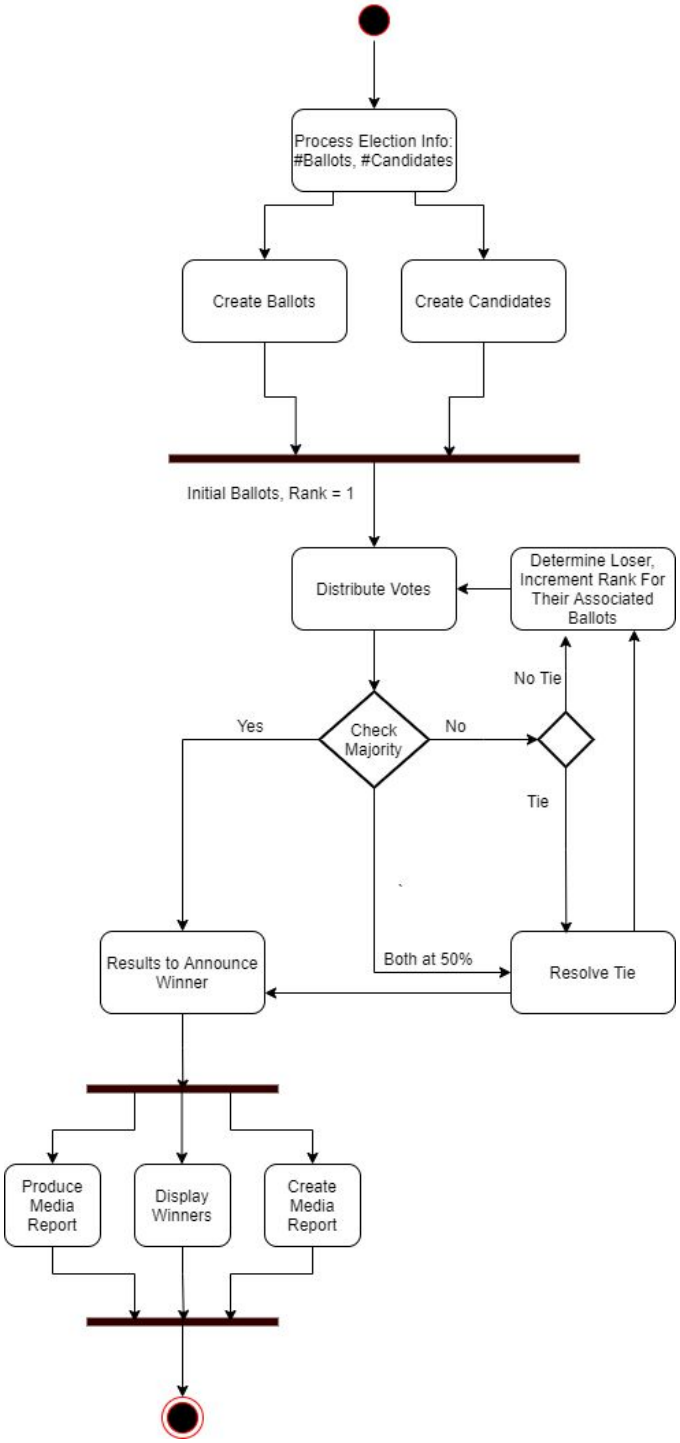

Figure 1. Activity Diagram for IR Algorithm.

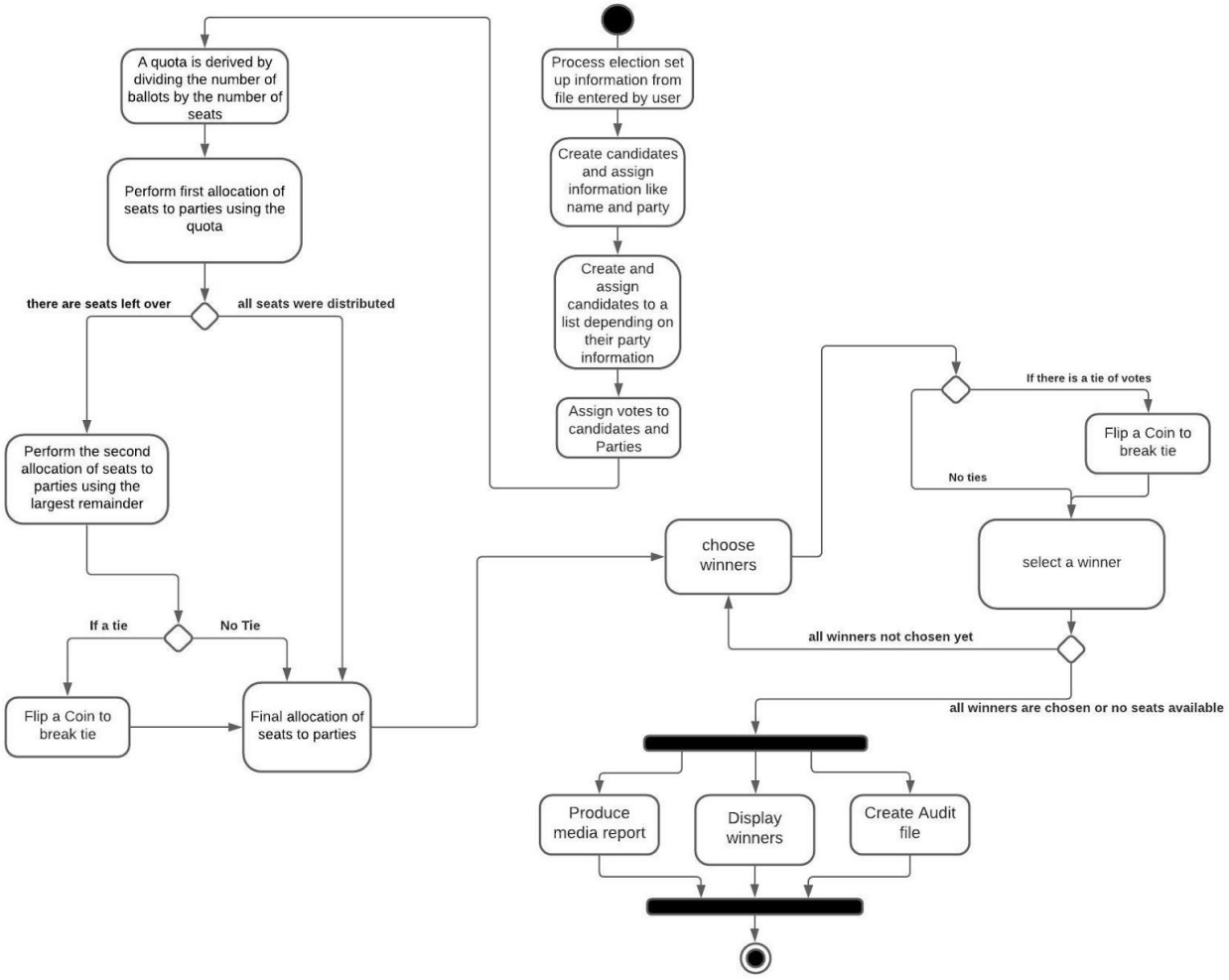

Figure 2. Activity Diagram for OPL Algorithm.

3.2 Decomposition Description

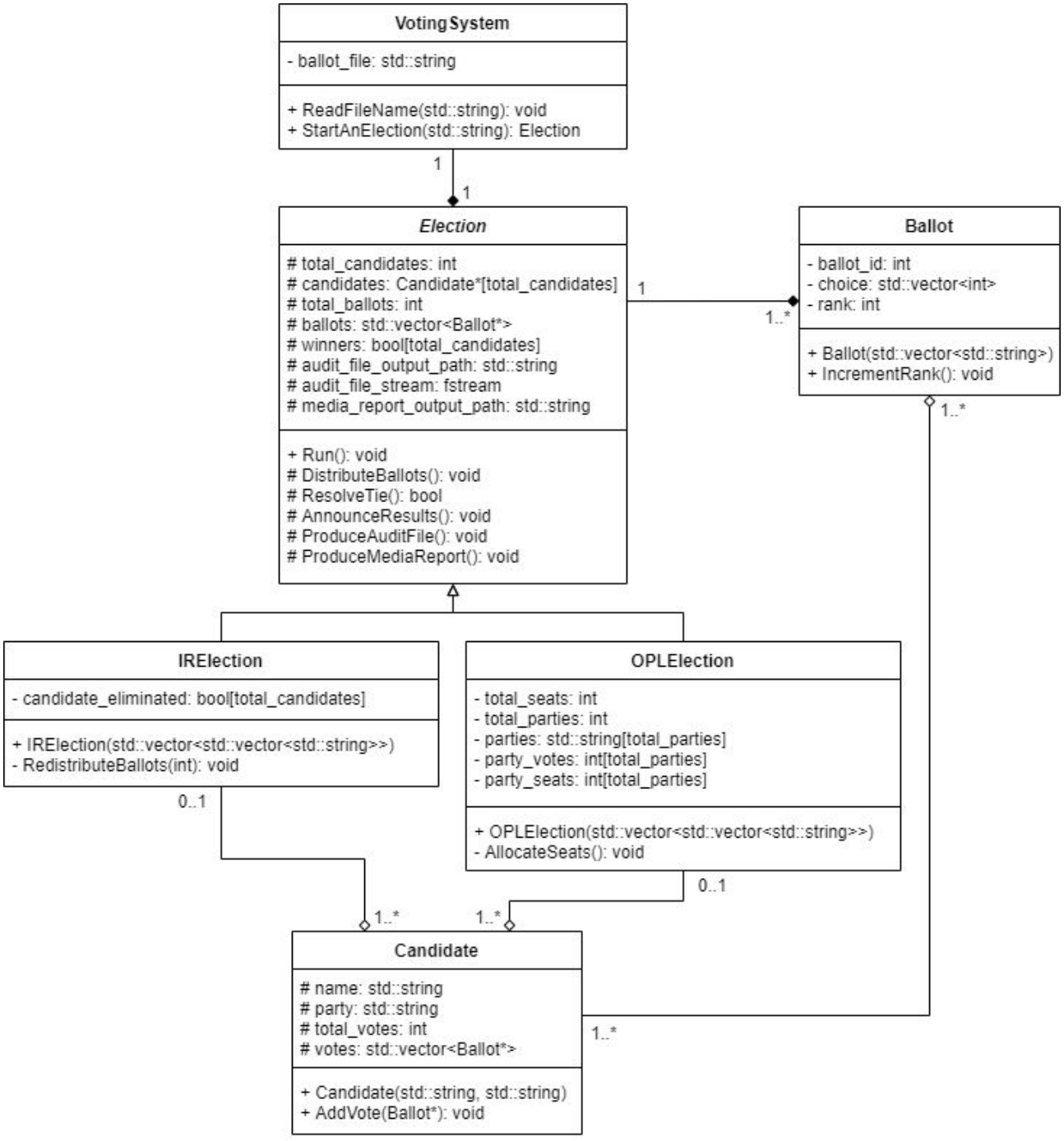

Figure 3. UML Class Diagram.

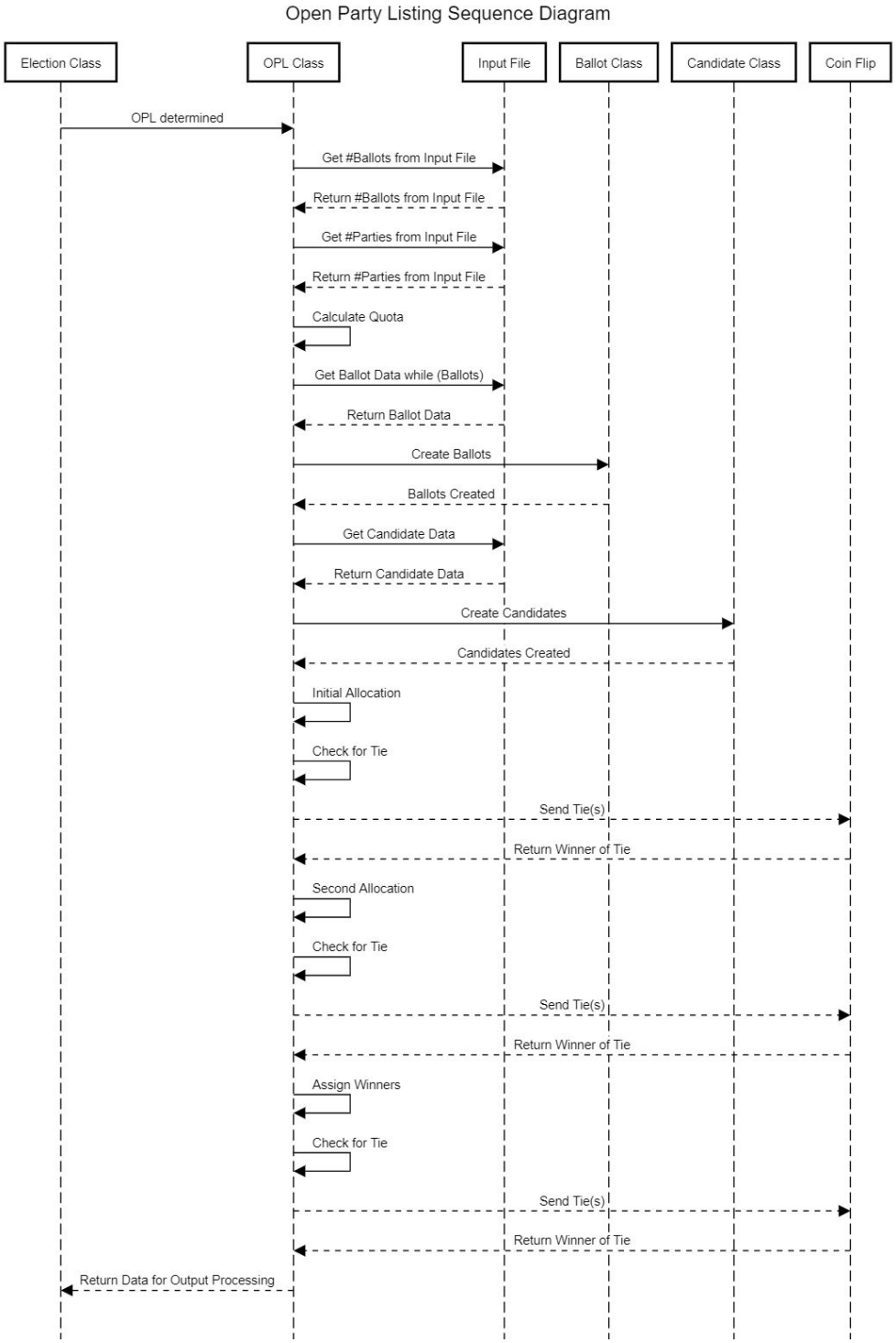

Figure 4. Sequence Diagram for OPL Algorithm.

3.3 Design Rationale

The Election class is abstract, from which there are corresponding inheriting election classes for Instant Runoff voting and Open Party Listing voting. This is because there are aspects of one voting type that would not be used in the other voting type.

Ballots are handled and organized by a derived Election class, but are eventually stored with candidates. This is because in Instant Runoff voting, the losing candidate's ballots must be redistributed. Having to iterate and search through 100,000 ballots to redistribute them would be much more computationally intensive than iterating through only the losing candidate's ballots.

4. Data Design

4.1 Data Description

The system does not interact with any databases. The ballot files are expected to be found in the local machine. The media reports and audit files will be stored locally.

4.2 Data Dictionary

Class Name	Attributes Name	Attribute Type	Description
Ballot	ballot_id	int	A private integer variable to store ballot Id.
	choice	std:vector<int>	A private vector that stores candidates index.
	rank	int	A private integer variable to store candidates rank.
Election	audit_file_output_path	std::string	A protected string variable to store the path where the audit file will be stored.
	audit_file_stream	fstream	A protected file stream to write the audit file.
	ballots	std::vector<Ballot*>	A protected vector list of Ballot objets to be distributed to candidates.

	candidates	Candidate*[total_candidates]	A protected array of pointers to Candidates objects.
	media_report_output_path	std::string	A protected string variable to store the path where the media report file will be stored.
	total_ballots	int	A protected int variable that stores the total number of ballots.
	total_candidates	int	A protected int variable that stores the total number of candidates.
	winners	bool[total_candidates]	A protected boolean array that indicates whether a candidate has won.
Candidate	name	std::string	A protected string variable that stores the candidate name.
	party	std::string	A protected string variable that stores the candidate party.
	total_votes	int	A protected int variable that stores the total number of votes.
	votes	std::vector<Ballot*>	A protected vector of Ballot objects that stores the votes a candidate got.
IRElection	candidate_eliminated	bool[total_candidatets]	A private boolean array that indicates whether a candidate has been eliminated.
OPLElection	parties	std::string[total_parties]	A private string array storing the

			names of each party in the election.
	party_seats	int[total_parties]	A private integer array storing the number of seats each part has won in the election.
	party_votes	int[total_parties]	A private integer array storing the number of votes each party has received.
	total_parties	int	A private integer storing the total number of parties.
	total_seats	int	A private integer storing the total number of seats.
VotingSystem	ballot_file	string	A private string variable to store the filename of the input file.

5. Component Design

VotingSystem

- Type: Class
- Description: This is a class for processing the input ballot file and creating an instance of an Election object.
- Attributes:
    - Ballot file: string
- Operations:
    - Name: *ReadFileName()*
        - Arguments: std::string
        - Returns: None
        - Pre-condition: The user starts the voting system program.
        - Post-condition: The filename is stored in the system as a string.
        - Exceptions: None
        - Flow of Events:
            1. Read the input filename as a string.
            2. Check if the file extension is CSV. If not, return to step 1.
            3. Check if the file exists. If not, return to step 1.
            4. Store the filename in the attribute ballot_file.

    - Name: *StartAnElection()*
        - Arguments: std::string
        - Returns: Election
        - Pre-condition: The ballot filename has been validated.
        - Post-condition: An instance of Election object is created.
        - Exceptions: None
        - Flow of Events:
            1. Open the ballot file in read-mode.
            2. Read the voting algorithm from line 1.
            3. Create the appropriate Election object (IRElection or OPLElection).

Election

- Type: Abstract class
- Description: This is an abstract class from which IRElection and OPLElection inherit. An instance of IRElection or OPLElection is constructed, depending on the type of voting denoted in the ballot file.
- Attributes:
    - Total Candidates: int
    - Candidates: Candidate*[total_candidates]
    - Total Ballots: int
    - Ballots: std::vector<Ballot*>
    - Winners: bool[total_candidates]
    - Audit File Output Path: std::string
    - Audit File Stream: fstream
    - Media Report Output Path: std::string
- Operations:
    - Name: *Run()*
        - Arguments: None
        - Returns: None
        - Pre-condition: An Election instance has been constructed.
        - Post-condition: An election occurs.
        - Exceptions: None
        - Flow of Events: Depends; this is an abstract method.
    - Name: *DistributeBallots()*
        - Arguments: None
        - Returns: None
        - Pre-condition: The ballots vector is nonempty.
        - Post-condition: Depends; this is an abstract method.
        - Exceptions: None
        - Flow of Events: Depends; this is an abstract method.
    - Name: ResolveTie()
        - Arguments: None
        - Returns: bool
        - Pre-condition: There is a tie that must be resolved.
        - Post-condition: The tie is resolved.
        - Exceptions: None
        - Flow of Events
            1. Seed random number generator with current time.
            2. Generate (pseudo-)random int 0 or 1.
            3. Cast randomly-generated int to bool, and return it.
    - Name: AnnounceResults()
        - Arguments: None

        - Returns: None
        - Pre-condition: The winner(s) has/have been determined.
        - Post-condition: A summary of results and the winner(s) are displayed.
        - Exceptions: None
        - Flow of Events
            1. Display winner(s) and other basic election information.
            2. Save the displayed information in the media report using ProduceMediaReport().
    - Name: ProduceAuditFile()
        - Arguments: None
        - Returns: None
        - Pre-condition: The results of the election have been announced.
        - Post-condition: There is an audit log with a unique filename in the assumed directory.
        - Exceptions: None
        - Flow of Events
            1. Close the filestream for the audit file.
    - Name: ProduceMediaReport()
        - Arguments: None
        - Returns: None
        - Pre-condition: The results of the election have been displayed.
        - Post-condition: There is a media report with a unique filename in the assumed directory.
        - Exceptions: None
        - Flow of Events
            1. Open a filestream for the media report under a unique name.
            2. Write basic information about the election into the media report.
            3. Close the filestream for the media report.

IRElection

- Type: Class
- Description: This is a class that is constructed if the first line of the ballot file denotes that it uses Instant Runoff voting.
- Attributes:
    - Candidate Eliminated: bool[total_candidates]
- Operations:
    - Name: IRElection()
        - Arguments: std::vector<std::vector<std::string>>
        - Returns: IRElection
        - Pre-condition: The ballot file denotes that it uses Instant Runoff voting.
        - Post-condition: An election using Instant Runoff voting is constructed.
        - Exceptions: None
        - Flow of Events:
            1. Open a filestream for the audit file under a unique name.
            2. Store information about total candidates, candidates, and total ballots in class variables.
            3. Initialize candidate eliminated array to all false.
            4. Convert parsed ballots into Ballot* instances using Ballot() constructor, and push onto the vector in class variables.
    - Name: Run()
        - Arguments: None

        - Returns: None
        - Pre-condition: An IRElection instance has been constructed.
        - Post-condition: An election using Instant Runoff voting occurs.
        - Exceptions: None
        - Flow of Events:
            1. Distribute ballots using DistributeBallots().
            2. Check whether a candidate has a majority of the votes. If true, go to step 5.
            3. Check whether there are two candidates each with half of the votes. If true, resolve the tie with ResolveTie() and go to step 5.
            4. Find the candidate with the fewest votes, and redistribute their ballots using RedistributeBallots(). If necessary, resolve ties with ResolveTie(). Go to step 2.
            5. Announce results and save a media report using AnnounceResults().
            6. Close the filestream for the audit file using ProduceAuditFile().
    - Name: DistributeBallots()
        - Arguments: None
        - Returns: None
        - Pre-condition: The ballots vector is nonempty.
        - Post-condition: The ballots are distributed to their respective candidates.
        - Exceptions: None
        - Flow of Events:
            1. Associate a Ballot instance with the respective candidate by using the candidate's AddVote() method.
            2. If the ballots vector is still nonempty, go to step 1.
    - Name: RedistributeBallots()
        - Arguments: int
        - Returns: None
        - Pre-condition: There is no candidate with the majority of the votes, and the given candidate has the fewest number of votes.
        - Post-condition: The given candidate is eliminated, and their votes are redistributed.
        - Exceptions: None
        - Flow of Events:
            1. Eliminate the given candidate by indicating it on the candidate eliminated array.
            2. Get a pointer to the candidate's votes using their RemoveVotes() method.
            3. Increment each ballot's rank using IncrementRank() so that it will be redistributed to a non-eliminated candidate.
            4. Redistribute the given candidate's votes to non-eliminated candidates using DistributeBallots().

OPLElection

- Type: Class
- Description: This is a class that is constructed if the first line of the ballot file denotes that it uses Open Party List voting.
- Attributes:
    - Total Seats: int
    - Winners: int[total_seats]
    - Total Parties: int

    - Parties: std::string[total_parties]
    - Party Votes: int[total_parties]
    - Party Seats: int[total_parties]
- Operations:
    - Name: OPLElection()
        - Arguments: std::vector<std::vector<std::string>>
        - Returns: OPLElection
        - Pre-condition: The ballot file denotes that it uses Open Party List voting.
        - Post-condition: An election using Open Party List voting is constructed.
        - Exceptions: None
        - Flow of Events:
            1. Open a filestream for the audit file under a unique name.
            2. Store information about total candidates, candidates, total seats, and total ballots in class variables.
            3. Determine total parties and party names from candidate information.
            4. Initialize party votes and party seats arrays to all 0.
            5. Convert parsed ballots into Ballot* instances using Ballot() constructor, and push onto the vector in class variables.
    - Name: Run()
        - Arguments: None
        - Returns: None
        - Pre-condition: An OPLElection instance has been constructed.
        - Post-condition: An election using Open Party List voting occurs.
        - Exceptions: None
        - Flow of Events:
            1. Distribute ballots using AddBallot().
            2. Allocate seats using AllocateSeats().
            3. Announce results and save a media report using AnnounceResults().
            4. Close the filestream for the audit file using ProduceAuditFile().
    - Name: DistributeBallots()
        - Arguments: None
        - Returns: None
        - Pre-condition: The ballots vector is nonempty.
        - Post-condition: The ballots are distributed to their respective candidates.
        - Exceptions: None
        - Flow of Events:
            1. Associate a Ballot instance with the respective candidate by using the candidate's AddVote() method.
            2. Increment the candidate's party's total votes by 1.
            3. If the ballots vector is still nonempty, go to step 1.
    - Name: AllocateSeats()
        - Arguments: None
        - Returns: None
        - Pre-condition: All ballots have been added and distributed.
        - Post-condition: All seats are allocated.
        - Exceptions: None
        - Flow of Events:
            1. Perform the first allocation of seats to parties.
            2. Perform the second allocation of seats to parties using the largest remainder. If necessary, resolve ties with ResolveTie().

3. Allocate seats to candidates of the respective party. If necessary, resolve ties with ResolveTie().
4. Indicate that a candidate has won a seat by adding their index in the candidates array to the winners array.

Candidate

- Type: Class
- Description: This is a class that is constructed for each candidate in the ballot file.
- Attributes:
    - Name: std::string
    - Party: std::string
    - Total Votes: int
    - Votes: std::vector<Ballot*>
- Operations:
    - Name: Candidate()
        - Arguments: std::string, std::string
        - Returns: Candidate
        - Precondition: A candidate's name and party have been passed in.
        - Postcondition: A Candidate instance is constructed.
        - Exceptions: None
        - Flow of Events:
            1. Store the arguments as the candidate's name and party, respectively.
            2. Initialize total votes to 0, and the votes vector as empty.
    - Name: AddVote()
        - Arguments: Ballot*
        - Returns: None
        - Pre-condition: A Ballot instance is to be distributed to the candidate.
        - Post-condition: The candidate's total votes are incremented by 1, and a pointer to the Ballot instance is associated with the candidate.
        - Exceptions: None
        - Flow of Events:
            1. Increment the candidate's total votes by 1.
            2. Push the pointer to the Ballot onto the votes vector.
    - Name: RemoveVotes()
        - Arguments: None
        - Returns: std::vector<Ballot*>*
        - Pre-condition: The candidate has been eliminated from the running.
        - Post-condition: The candidate's total votes is set to 0, and their votes returned to the inherited Election class.
        - Exceptions: None
        - Flow of Events:
            1. Set the candidate's total votes to 0.
            2. Return a pointer to the votes vector.

Ballot

- Type: Class
- Description: This is a class that is constructed for each ballot string in the ballot file.
- Attributes:
    - Ballot ID: int
    - Choice: std::vector<int>
    - Rank: int

- Operations:
  - Name: Ballot()
    - Arguments: std::vector<std::string>
    - Returns: Ballot
    - Precondition: A parsed ballot is to be converted into a Ballot instance.
    - Postcondition: A Ballot instance is constructed.
    - Exceptions: None
    - Flow of Events:
      1. Initialize a unique ballot ID with a static/global variable, and initialize rank to 0.
      2. Store the ballot's choices as a vector of ints.
  - Name: IncrementRank()
    - Arguments: None
    - Returns: None
    - Precondition: The ballot's vote under the current rank is for an eliminated candidate.
    - Postcondition: The rank is incremented by 1
    - Exceptions: None
    - Flow of Events:
      1. Increment the rank by 1.

6. Human Interface Design

6.1 Overview of User Interface

The system uses a command line interface. The system will prompt the user for inputs when executed. The user is expected to use a keyboard to type in the inputs.

6.2 Screen Images

The figure below shows a prototype of the interface. The system is re-prompting the user to enter the name of the ballot file after the user provides an invalid filename.

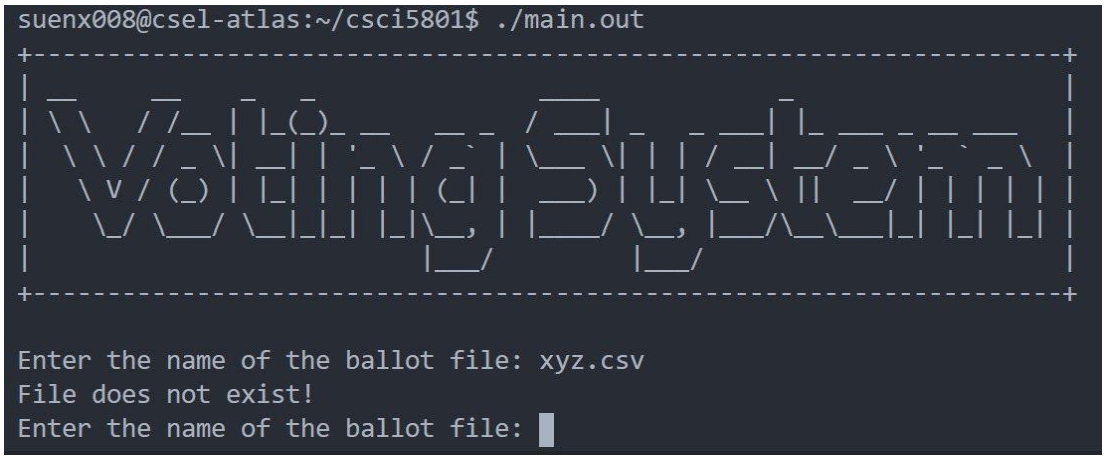

Figure 5. A Prototype of the Interface.

6.3 Screen Objects and Actions

The system does not have a GUI and hence does not have any screen objects. All operations are done through the command line.

7. Requirements Matrix

Use Case ID	Name	System components
UC_01	Read Input Filename	This requirement is fulfilled by the VotingSystem class.
UC_02	Bring File into System	This requirement is fulfilled by the VotingSystem class.
UC_03	Count Votes (Instant Runoff)	This requirement is fulfilled by the IRElection class.
UC_04	Count Votes (Open Party Listing)	This requirement is fulfilled by the OPLElection class.
UC_05	Resolve a tie	This requirement is fulfilled by the Election class.
UC_06	Announce Results	This requirement is fulfilled by the Election class.
UC_07	Produce audit file	This requirement is fulfilled by the Election class.
UC_08	Produce media report	This requirement is fulfilled by the Election class.

8. Appendices

// TODO: *This section is optional.* Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.