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 7)

(Project Aegis)

Software Design Document

Name (s):

Grant Golemo

Donald Huynh

Michael Ly

Lucky Vang

Team 7

Date: 02/28/2021

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This Software Design Document provides the design details of the Aegis voting system project.

The intended audience is election officials, developers, and testers interested in the design and use of a voting system capable of performing Open Party Listing and Instant Runoff voting.

1.2 Scope

This document contains a complete description of the design of Aegis.
The basic architecture is a terminal window. The program will be in C++. The software will either run the Open Party Listing algorithm or the Instant Runoff algorithm. The information about the election will be obtained from csv files that contain the ballot information. The output of the program will be fair in how it decides the winners for each type of election.

1.3 Overview

Section 2: contains a general description of Aegis

Section 3: contains UML and design diagrams that represent the program.

Section 4: contains a general listing of class functions and a description for each class attribute

Section 5: contains descriptions of each class and an thorough description of their functions

Section 6: contains how the user interface would work as well as images that show how it would work.

Section 7: contains a cross reference that traces components and data structures to the requirements in our SRS document.

1.4 Reference Material

No reference material used at this time.

1.5 Definitions and Acronyms

C++	The programming language that is used to write the program.
CSV	A file that has its values separated by commas.
IR	Instant Runoff IR is a voting system that declares the outcome of an election by eliminating unpopular candidates until one candidate has a simple majority. Every ballot has the option of ranking their preferred candidates from most to least, ensuring that everyone's vote is counted.
Italicized text	Used to reference class attributes
Modulo	A math operation that returns the remainder of a division.
OPL	Open Party Listing OPL is a voting system that chooses the outcome of an election by distributing seats among the participating parties according to their popularity. Inside each party, the amount of winners are decided based on the number of seats their party earned, and the winners themselves are determined by their popularity.
Quota	The amount of ballots needed from a party in order to obtain a seat
RNG	Random number generator that will be used to break ties
SDD	Software Design Document. A document that contains the design of a software.
SRS	Software Requirement Specification. A document that contains the requirements
Terminal Window	A program native to all computers that is used to run programs.

UML	Unified Modeling Language.

2. System Overview

Project Aegis is intended to be a program, allowing for the process of elections run on the OPL and IR voting systems. The context of this project is to allow developers and students to explore software engineering and design of a project as well as provide a system to run elections.

3. System Architecture

3.1 Architectural Design

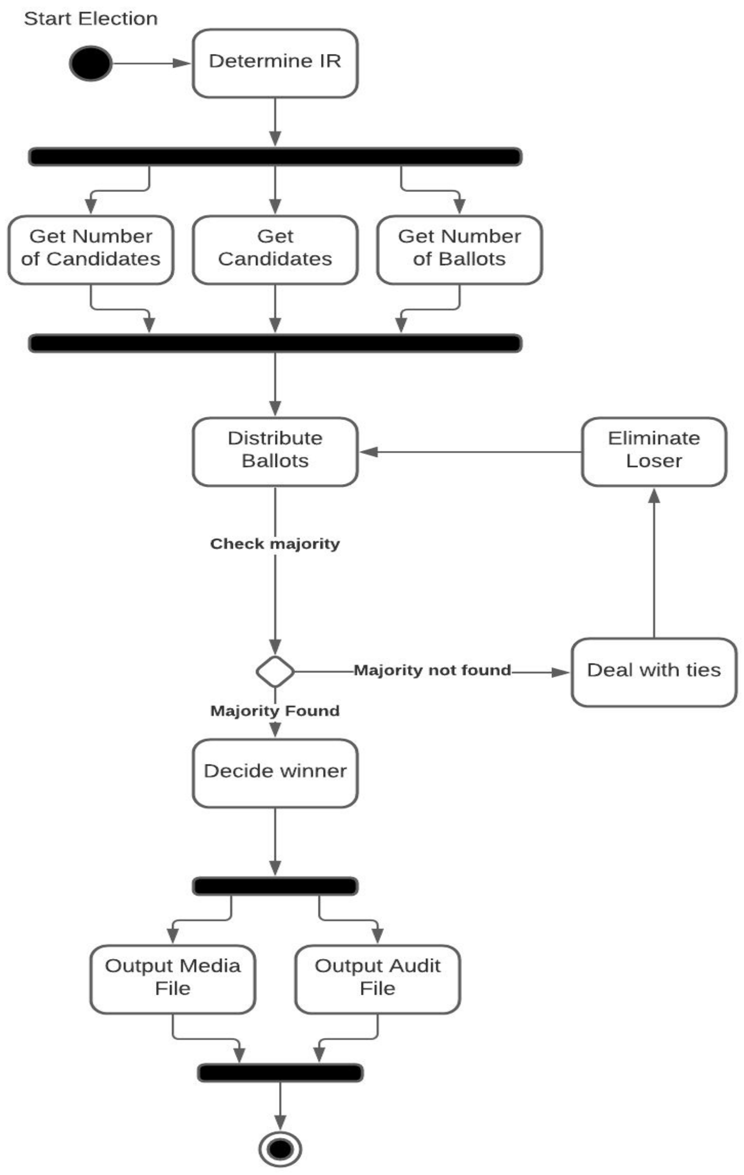


Figure 1: UML IR Activity Diagram

This diagram shows the flow of the IR voting system.

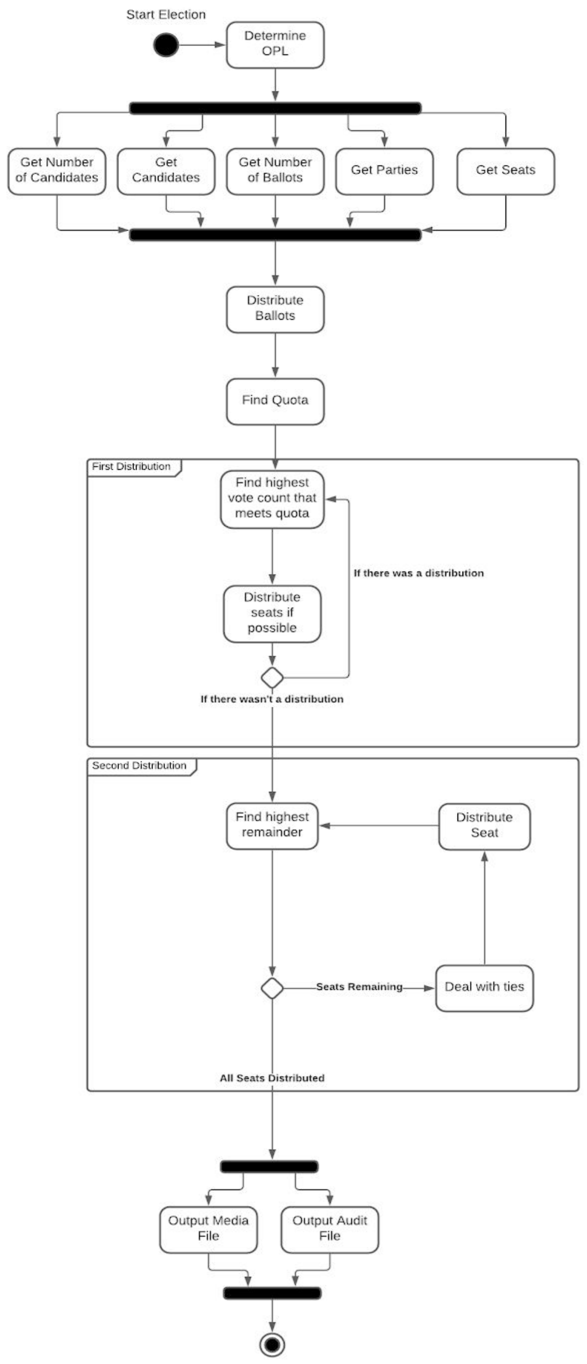


Figure 2: UML OPL Activity Diagram

This diagram shows the flow of the IR voting system.

3.2 Decomposition Description

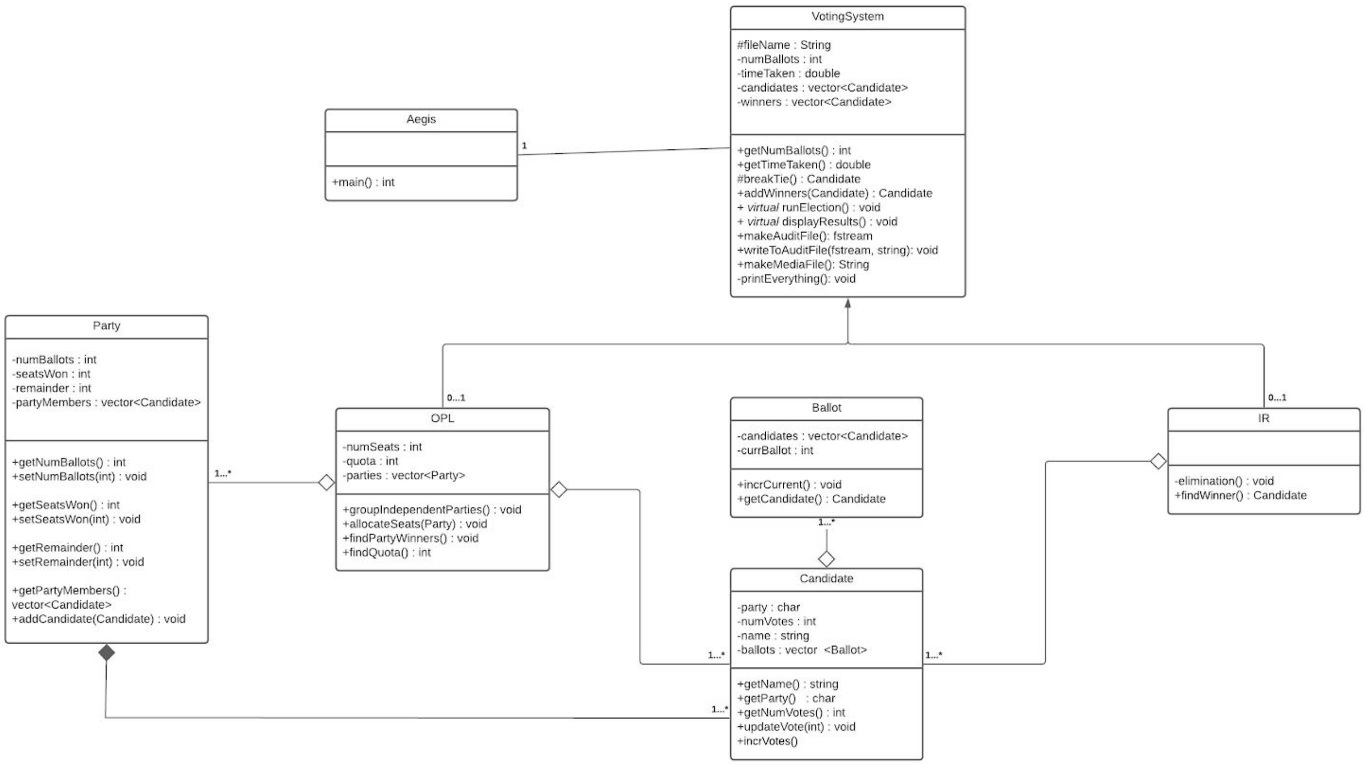
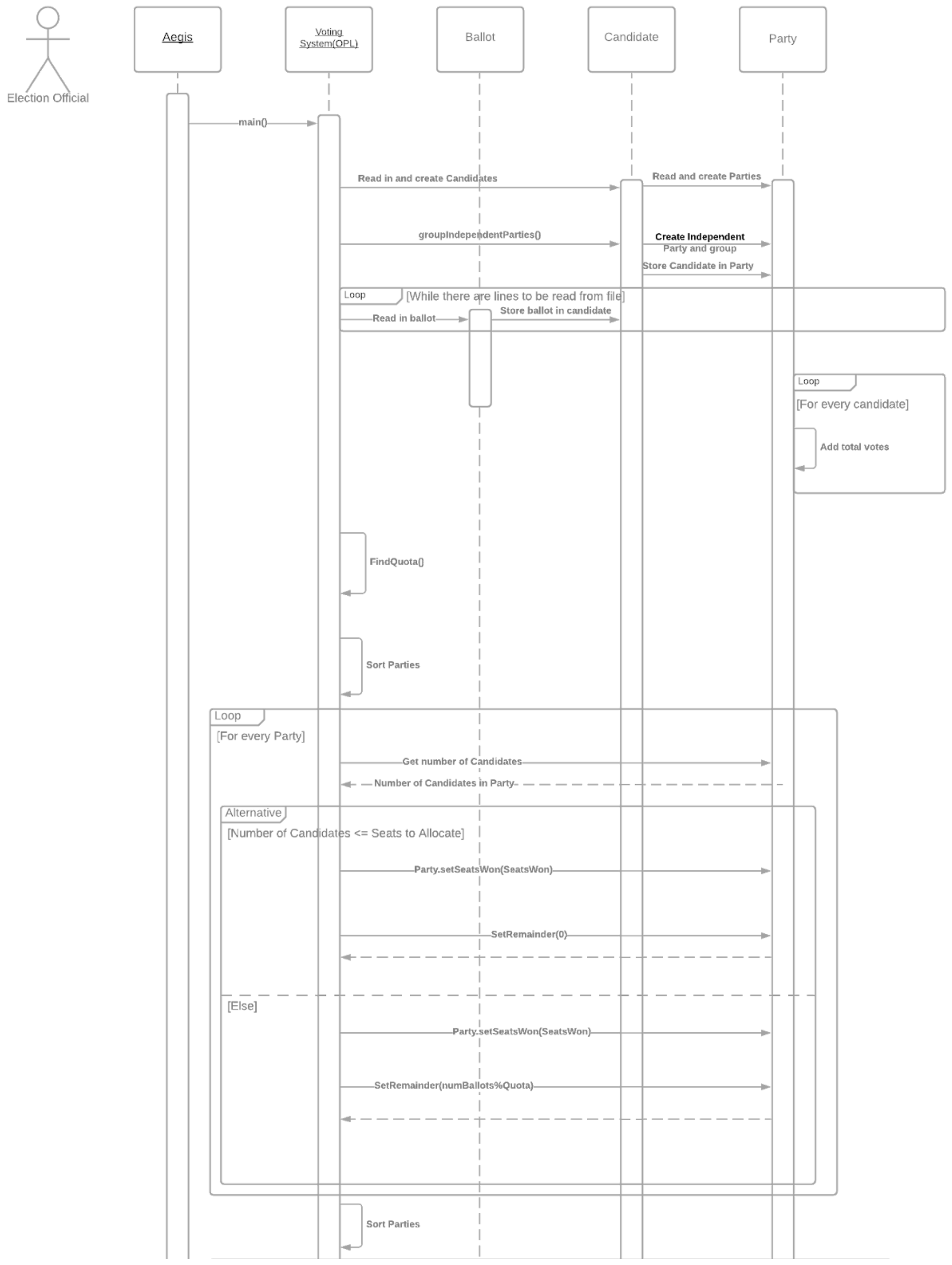


Figure 3: UML Class Diagram

This diagram showcases all the relevant data structures, attributes, functions, and relationships between each class used in the voting system. An in depth description of their attributes and functions can be found in Section 4 and 5, respectively.


Election Official
Aegis
Voting System(OPL)
Ballot
Candidate
Party
main()
Read in and create Candidates
Read and create Parties
groupIndependentParties()
Create Independent Party and group
Store Candidate in Party
Loop
[While there are lines to be read from file]
Read in ballot
Store ballot in candidate
Loop
[For every candidate]
Add total votes
FindQuota()
Sort Parties
Loop
[For every Party]
Get number of Candidates
Number of Candidates in Party
Alternative
[Number of Candidates <= Seats to Allocate]
Party.setSeatsWon(SeatsWon)
SetRemainder(0)
[Else]
Party.setSeatsWon(SeatsWon)
SetRemainder(numBallots%Quota)
Sort Parties

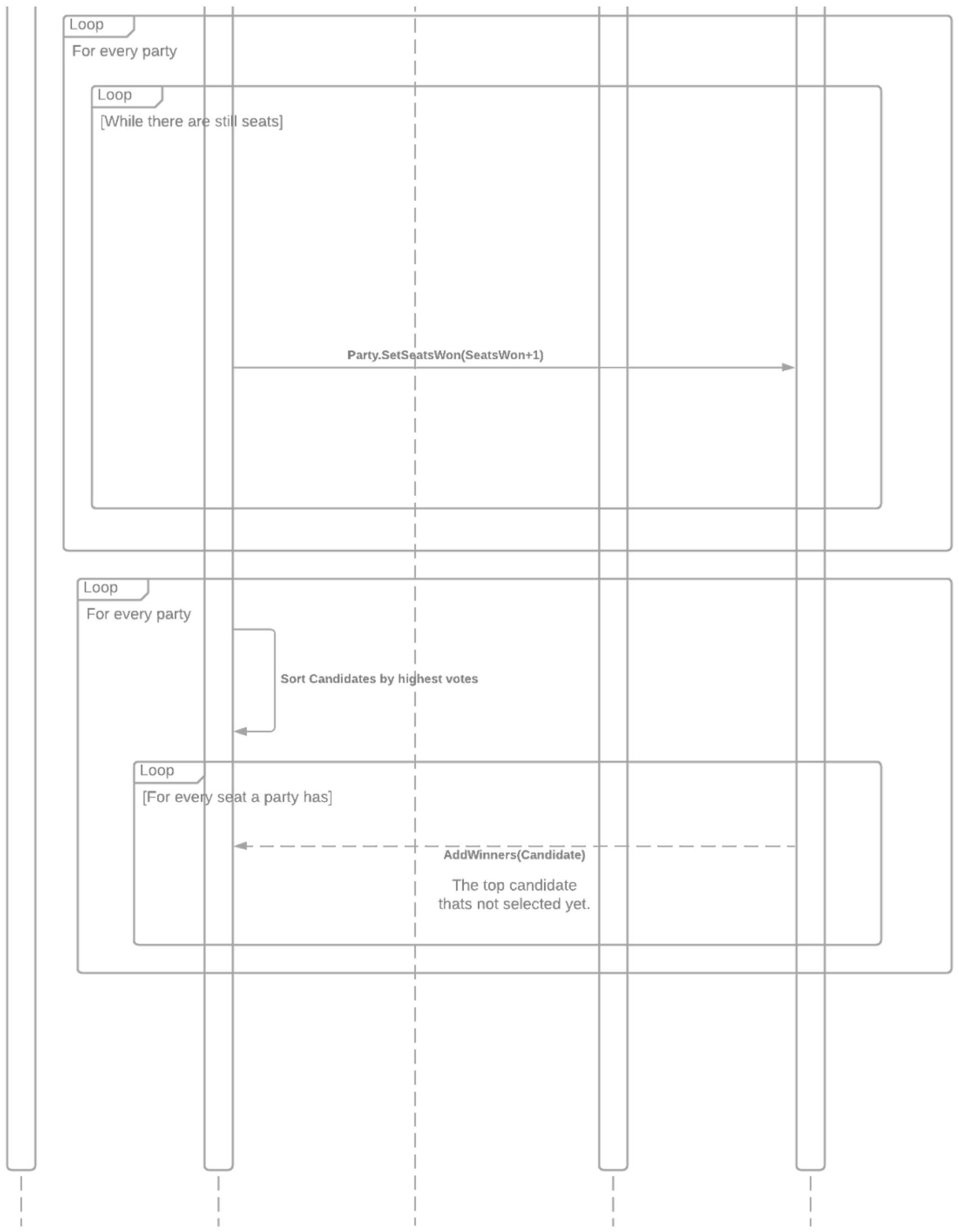

Loop
For every party
Loop
[While there are still seats]
Party.SetSeatsWon(SeatsWon+1)
Loop
For every party
Sort Candidates by highest votes
Loop
[For every seat a party has]
AddWinners(Candidate)
The top candidate
thats not selected yet.

Figure 4 and 5: UML OPL Sequence diagram

1. The user runs the program along with a file argument that reads in OPL.
2. The program goes through all the info in the file and keeps track of the initial variables such as the number of seats and ballots.
3. The candidates are created and then added to their respective parties, if the parties don't exist then create the parties and add the candidate to their party.
4. The independent parties are then grouped together and an independent party is created which encompasses all of the grouped independent parties.
5. The ballots are then read in and created.
6. The ballots will be assigned to their respective candidates.
7. Once ballots are assigned, votes are tallied up for each candidate.
8. All candidates' votes within each party are added up and stored in their respective parties.
9. Information on total ballots and number of seats is now used to determine the quota to receive a seat.
10. Each party's candidate vector is sorted from most to least amount of votes received.
11. For every party, allocate seats as appropriate, if the number of parties is $\leq$ the number of seats to allocate, set party remainder to 0, else set it to the party's number of ballots % the quota.
12. Sort the parties from most to least according to their remainder.
13. While there are still seats available, iterate through the vector of parties and give each party an additional seat.
14. For every party, sort the candidates from most to least according to their votes.
15. For every seat within that party, add the top candidate of that party to the winners vector.

3.3 Design Rationale

We chose to use vectors over arrays, as they are dynamic and allow for easy removal and addition of ballots and candidates.

We chose to design a Party class because it helped with running the OPL algorithm. It stored necessary data such as the remainder, number of seats per party, etc. Using a factory for creating parties was considered, but for the time being we've decided to keep things simple as currently we have no advantages of using a factory for parties.

We chose to have ballots contain candidates instead of the other way around in order to save time and space.

We chose to sort the parties in OPL first because it's easier to allocate the seats when the parties are sorted rather than a search every time we find the next highest ballot total in a party. Similarly, we sorted the candidates so we didn't have to do a search.

4. Data Design

4.1 Data Description

All ballots are taken in through a CSV file. Each line after the first 4 for IR and 5 for OPL represents one ballot. Each of these lines is fed into a constructor that will turn the information into a ballot object. Each candidate is stored inside a vector<Candidate> in the order in which they were ranked. These ballots are all stored in the vector<Ballot> inside a Candidate object.
Each candidate in an election has its own candidate object, along with a vector full of the votes that currently belong to them.

4.2 Data Dictionary

- Aegis
    - main(int, int[])
- Ballot
    - incrCurrent()
    - getCandidate()
- Candidate
    - getName()
    - getParty()
    - getNumVotes()
    - incrVotes()
    - updateVotes(int)
- IR
    - elimination()
    - findWinner()
- OPL
    - groupIndependentParties()
    - allocateSeats()
    - findPartyWinners()
    - findQuota()
- Party
    - getNumBallots()
    - setNumBallots()

    - getSeatsWon()
    - setSeatsWon()
    - getRemainder()
    - setRemainder()
    - getPartyMembers()
    - addCandidate()
- Voting System
    - getNumBallots()
    - getTimeTaken()
    - breakTie()
    - getNumSeats()
    - addWinners(Candidate)
    - virtual runElection()
    - virtual displayResults()
    - makeAuditFile()
    - writeToAuditFile(fstream, string)
    - makeMediaFile()
    - printEverything()

Class	**Attribute Name**	**Attribute Type**	**Description**
Aegis	N/A	N/A	N/A
Ballot	candidates	vector<Candidate>	a private vector containing Candidates ordered by preference
	currBallot	int	a private integer that indicates the current rank that the ballot is on.
Candidate	party	char	a private char containing the party of the Candidate
	name	String	a private String containing the name of the Candidate
	ballots	vector<Ballot>	a private vector containing all the ballots that are currently voting for this Candidate
	numVotes	int	a private integer representing the

			total number of votes belonging to the Candidate
OPL	parties	vector<Party>	a private vector containing Party that stores all parties
	quota	int	a private integers that contains the amount of votes needed to obtain a seat in the OPL type election
	numSeats	int	a private integer that represents the number of seats for the election
IR	N/A	N/A	N/A
Party	numBallots	int	a private integer that stores the current number of ballots that the party has which is the sum of all of the candidate ballots combined.
	seatsWon	int	a private integer which stores the amount of seats that the party has currently obtained from the election.
	remainder	int	a private integer which stores the remaining votes after a first allocation in a OPL based election.
	partyMembers	vector<Candidate>	a private vector of Candidates that indicates which the candidates are in the corresponding party.
Voting System	fileName	string	a private string that contains the name of the file that is being read for the election.
	numBallots	int	a private integer that stores the

			amount of ballots that are present in the file.
	timeTaken	double	a private double variable that stores the amount of time it takes to run the program.
	candidates	vector<Candidate>	a private vector of Candidates that stores all the candidates read from the file
	winners	vector<Candidate>	a private vector of Candidates that stores the winning candidates of the election.

5. Component Design

Aegis

- Name: Aegis
- Type: Class
- Description: A class containing the main function of the program.
- Attributes: None.
- Operations:
  Name: main
  - Description: The main method of Aegis.
  - Arguments:
    - argc: int
    - argv: int[]
  - Return: int
  - Pre-condition: The program has been executed.
  - Post-condition: The results of the election have been computed.
  - Exception:
    - EX1: The file is invalid
      1. The system will inform the user to rerun the program and to input a valid file.
  - Flow of Events:
    1. The user has executed the program and input a valid file name.
    2. The system runs Aegis.

Ballot

- Name: Ballot
- Type: Class
- Description: A class designed to store all necessary information about a ballot.
- Attributes:
    - candidates: vector<Candidate>
    - currBallot: int
- Operations:

  Name: getCandidates
    - Description: Returns the *candidates* vector.
    - Arguments: None.
    - Returns: vector<Candidate>
    - Pre-condition: A Ballot object has been instantiated.
    - Post-condition: The candidate vector has been returned.
    - Exception: None.
    - Flow of Events:
        1. The candidate vector is returned.

  Name: incrCurrent
    - Description: Increments the current ballot choice to the next appropriate position and is used after elimination in IR for vote redistribution.
    - Arguments: None.
    - Returns: void
    - Pre-condition: A Ballot object has been instantiated.
    - Post-condition: *Ballot: currBallot* has been incremented to the next appropriate position.
    - Exceptions: None.
    - Flow of Events:
        1. The *currBallot* is incremented to the next appropriate position.

Candidate

- Name: Candidate
- Type: Class
- Description: A class designed to store all necessary information about a candidate.
- Attributes:
    - party: char
    - name: string
    - ballots: vector<Ballot>
    - numVotes: int
- Operations:

  Name: getParty

- Description: Returns the party.
- Arguments: None.
- Returns: char
- Pre-condition: A Candidate object has been instantiated.
- Post-condition: The party is returned.
- Exception: None.
- Flow of Events:
    1. The party is returned.

Name: getName

- Description: Returns the name.
- Arguments: None.
- Returns: char
- Pre-condition: A Candidate object has been instantiated.
- Post-condition: The name is returned.
- Exception: None.
- Flow of Events:
    1. The name is returned.

Name: getBallots

- Description: Returns the ballots.
- Arguments: None.
- Returns: char
- Pre-condition: A Candidate object has been instantiated.
- Post-condition: The ballots is returned.
- Exception: None.
- Flow of Events:
    1. The ballots is returned.

Name: getNumVotes

- Description: Returns the number of votes.
- Arguments: None.
- Returns: int
- Pre-condition: A Candidate object has been instantiated.
- Post-condition: The number of votes is returned.
- Exception: None.
- Flow of Events:
    1. The number of votes is returned.

Name: incrVotes

- Description: Increment the number of votes by 1.
- Arguments: None.
- Returns: void
- Pre-condition: A Candidate object has been instantiated.
- Post-condition: The number of votes has been increased by 1.

- Exception: None.
- Flow of Events:
    1. The number of votes is incremented by 1.

Name: updateVotes

- Description: Add inputted argument *newVote* number of votes to the current number of votes.
- Arguments:
    - newVote: int
- Returns: void
- Pre-condition: A Candidate object has been instantiated.
- Post-condition: The number of votes has been increased by the inputted argument *newVote*.
- Exception: None.
- Flow of Events:
    1. The number of votes is increased by the inputted argument *newVote*.

IR

- Name: IR
- Type: Class, inherits Voting System
- Description: IR runs the program using the IR algorithm.
- Attributes: None.
- Operations:

Name: elimination

- Description: Removes the candidate with the least amount of votes.
- Arguments: None
- Returns: void
- Pre-condition: No candidate has a majority of the votes.
- Post-condition: One candidate has been eliminated from the election.
- Exceptions: None.
- Flow of Events:
    1. Iterate through *candidates* from the Voting System class and find the candidate with the least amount of votes.
    2. If there is a tie for the least amount of votes, call breakTie() to determine which candidate should be eliminated.
    3. Loop through all of the eliminated candidate's ballots and increment its *currBallot*. Redistribute ballots to its new candidate which is determined by the *currBallot* index of *candidates* in the Ballot class.
    4. Remove the eliminated candidate from *candidates* vector in the Voting System.

Name: findWinner

- Description: Finds the winner of the IR system.
- Arguments: None.
- Returns: Candidate or nullptr
- Pre-condition: The file has been preprocessed and all votes have been distributed to candidates.
- Post-condition: The winner of the position has been chosen.
- Exceptions: None.
- Flow of Events:
    1. Sorts *Voting System: candidates* vector by votes.
    2. Checks for a majority by iterating through each candidate in the *Voting System: candidates* vector and verifying if any candidate has more than 50% of the votes.
    3. If there is a majority, return the zeroth index of the vector.
    4. Otherwise, return nullptr.

OPL

- Name: OPL
- Type: Class, inherits Voting System
- Description: OPL runs the program using the OPL algorithm.
- Attributes:
    - parties: vector<Party>
    - quota: int
- Operations:
  Name: allocateSeats
    - Description: Allocate seats appropriately amongst parties.
    - Arguments: Party
    - Returns: void
    - Pre-condition: The file has been preprocessed, the ballots have been distributed accordingly, and the quota has been calculated.
    - Post-condition: Each party has been given their appropriate seat amount.
    - Exceptions: None
    - Flow of Events:
        1. Sort *parties* by the number of votes each party has received.
        2. Allocate seats to parties by looping through *parties*.
            a. The amount of seats granted to a party is determined by dividing a party's total votes by the quota and flooring the quotient.
            b. Determine the remainder by doing a party's total votes modulo quota and is stored inside the party.

        c. If a party were to receive more seats than available candidates, then the extra seat is granted to another party using the same process as above.
    3. If there are still remaining seats after the first allocation, repeat Step 1 but use the party's remainder in place of its total votes.

Name: findPartyWinners

- Description: Find who gets the seats within each party.
- Arguments: None
- Returns: void
- Pre-condition: Seats have been allocated to parties.
- Post-condition: The members of each party with the most votes get an appropriate number of seats.
- Exceptions: None
- Flow of Events:
    1. Begin a loop that goes through all elements in *parties.*
    2. Sort each candidate in *partyMembers* by number of votes earned.
    3. Assign the number of seats earned by each party to the candidates at the start of the vector (X seats go to first X candidates in vector).

Name: findQuota

- Description: Finds the quota to determine allocation of seats.
- Arguments: None
- Returns: int
- Pre-condition: The file has been preprocessed and all votes have been assigned to candidates.
- Post-condition: The quota required by OPL has been determined.
- Exceptions:
- Flow of Events:
    1. Find the total number of votes by iterating through all instances of the Party class and summing their ballots.
    2. Divide the total number of votes by the total number of seats available.
    3. Return this number.

Name: groupIndependentParties

- Description: Group all the independent parties into a singular party.
- Arguments: None
- Returns: void
- Pre-condition: The file has been preprocessed.
- Post-condition: All independent parties have been grouped together.
- Exceptions: None.
- Flow of Events:

1. When processing the file, while iterating through candidates, add all independent parties to the instance of the Party Class called “I”. If it does not exist yet, create one and add it there.

Party

- Name: Party
- Type: Class
- Description: A class designed to store all the necessary information of a party.
- Attributes:
    - numBallots: int
    - seatsWon: int
    - remainder: int
    - partyMembers: vector<Candidates>
- Operations:

  Name: getNumBallots
    - Description: Returns the number of ballots.
    - Arguments: None.
    - Returns: int
    - Pre-condition: A Party object has been instantiated.
    - Post-condition: Returns the number of ballots.
    - Exceptions: None.
    - Flow of Events:
        1. The number of ballots is returned.

  Name: setNumBallots
    - Description: Sets the number of ballots to the inputted argument *ballots.*
    - Arguments:
        - ballots: int
    - Returns: void
    - Pre-condition: A party object has been instantiated
    - Post-condition: The number of ballots has been set to *ballots*.
    - Exceptions: None.
    - Flow of Events:
        1. The number of ballots is set to the value of the int parameter *ballots*.

  Name: getSeatsWon
    - Description: Returns the seats won.
    - Arguments: None.
    - Returns: int
    - Pre-condition: A Party object has been instantiated.
    - Post-condition: The seats won have been returned.
    - Exceptions: None.

- Flow of Events:
  1. The seats won are returned.

Name: setSeatsWon

- Description: Sets the number of seats won to the inputted argument *seats.*
- Arguments:
  - seats: int
- Returns: void
- Pre-condition: A party object has been instantiated
- Post-condition: The seats won have been set to *seats.*
- Exceptions: None.
- Flow of Events:
  1. The seats won have been set to the value of the int parameter *seats*.

Name: getRemainder

- Description: Returns the remainder.
- Arguments: None.
- Returns: int
- Pre-condition: A Party object has been instantiated.
- Post-condition: The remainder has been returned.
- Exceptions: None.
- Flow of Events:
  1. The remainder is returned.

Name: setRemainer

- Description: Sets the number of remainder to the inputted argument *num*.
- Arguments:
  - num: int
- Returns: void
- Pre-condition: A party object has been instantiated
- Post-condition: The remainder has been set to *num*.
- Exceptions: None.
- Flow of Events:
  1. The remainder is set to the value of the int parameter *num*.

Name: getPartyMembers

- Description: Returns the *partyMembers*.
- Arguments: None.
- Returns: vector<Candidates>
- Pre-condition: A Party object has been instantiated.
- Post-condition: Returns the *partyMembers.*
- Exceptions: None.
- Flow of Events:
  1. The *partyMembers* is returned.

Name: addCandidate

- Description: Adds a candidate to *partyMembers*
- Arguments:
    - candidate: Candidate
- Returns: void
- Pre-condition: A party object has been instantiated
- Post-condition: A candidate has been added to *partyMembers*.
- Exceptions: None.
- Flow of Events:
    1. The *candidate* is added to *partyMembers.*

Voting System

- Name: Voting System
- Type: Class
- Description: The voting system will be the highest super class and an interface. It will provide the methods and attributes for the entire system.
- Attributes:
    - fileName: string
    - numBallots: int
    - timeTaken: double
    - candidates: vector<Candidate>
    - numSeats: int
    - winners: vector<Candidate>
- Operations:

  Name: getNumBallots
    - Description: Returns the number of ballots.
    - Arguments: None
    - Returns: int
    - Pre-condition: None
    - Post-condition: Number of ballots are returned to the method that called it.
    - Exceptions: None.
    - Flow of Events:
        1. The system needs a number of ballots
        2. The system calls the method.
        3. The system receives the number of ballots.

  Name: getTimeTaken
    - Description: Returns the time taken to run the program.
    - Arguments: None
    - Returns: double
    - Pre-condition: The main functions of the system are run.
    - Post-condition: The time taken to run the program is returned

- Exceptions: None.
- Flow of Events:
    1. The program has decided the election results
    2. The program calls for the time taken.
    3. The system returns the time taken to run the program.

Name: getNumSeats

- Description: Returns the total number of available seats
- Arguments: None
- Returns: int
- Pre-condition: The file has been preprocessed.
- Post-condition: The number of seats has been returned.
- Exceptions: None.
- Flow of Events:
    1. The number of seats is returned.

Name: breakTie

- Description: Uses a random number generator to randomly select a winner between 2 or more candidates.
- Arguments: Candidate1, Candidate2, …. , CandidateN
- Returns: Candidate.
- Pre-condition: There is a tie in the number of votes between 2 or more candidates.
- Post-conditions: One of the candidates is isolated from the other candidates.
- Exceptions: None
- Flow of Events:
    1. The system sees that there's a tie.
    2. The system calls breakTie in order to break the tie.
    3. The system then receives a candidate from breakTie.

Name: addWinners

- Description: Adds a candidate to the winners vector
- Arguments: candidate: Candidate
- Returns: void
- Pre-condition: The file has been preprocessed.
- Post-condition: A candidate has been added to the winners vector.
- Exceptions: None.
- Flow of Events:
    1. The *candidate* is added to *winners*.

Name: runElection

- Description: Function to run the election one all necessary objects are made.
- Arguments: None.
- Returns: void
- Pre-condition: relevant data such as fileName, numBallots, and candidates are initialized.
- Post-condition: The election has been won and winners have been determined.
- Exceptions: None.
- Flow of Events:
    1. The system determines the type of election.
    2. Relevant data structures are instantiated.
    3. The system type of voting system is run
    4. The system computes the results and determines winners.
    5. The system outputs relevant information with audit and media files.

Name: displayResults

- Description: Displays to the screen the results of the election.
- Arguments: None.
- Returns: void
- Pre-condition: The election results have been calculated.
- Post-condition: The results of the election have been displayed on the screen.
- Exceptions:
- Flow of Events:
    1. The system prints out to the screen the election results including all of the candidates and their stats and general election information.

Name: makeAuditFile

- Description: Creates an audit file with customized name
- Arguments: None.
- Returns: fstream
- Pre-condition: The program has been preprocessed.
- Post-condition: An audit file has been created and placed into the same directory as the program.
- Exceptions:
    - EX2: The PC is out of storage
        1. An error will show that the PC doesn't have enough storage to save the audit file.
        2. Exit program.
- Flow of Events:

1. The program creates a file stored inside the same directory as the program with the date time as the title.
2. Returns the file.

Name: writeToAuditFile

- ○ Description: Writes to audit file
- ○ Arguments:
  - ■ file: fstream
  - ■ line: string
- ○ Returns: void
- ○ Pre-condition: The program has finished preprocessing
- ○ Post-condition: The audit file has been written to.
- ○ Exceptions: None.
- ○ Flow of Events:
  1. The program writes *line* into the audit file.

Name: makeMediaFile

- ○ Description: Creates and writes to a media file.
- ○ Arguments: None.
- ○ Returns: String
- ○ Pre-condition: The program has finished computing the results of the election.
- ○ Post-condition: A media file has been created and placed into the same directory as the program.
- ○ Exceptions:
  - ■ EX2: The PC is out of storage
    1. An error will show that the PC doesn't have enough storage to save the audit file.
    2. Exit program.
- ○ Flow of Events:
  1. The program creates a file stored inside the same directory as the program with the date time as the title.
  2. The program combines all necessary information and writes it to the file.
  3. Returns the file name.

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

Election Official:

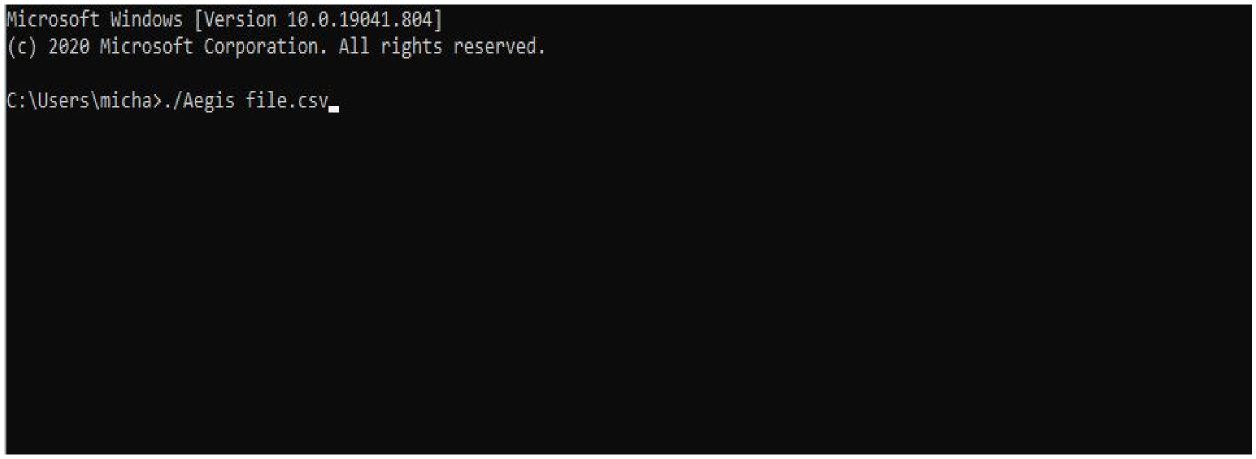
In order to use the Aegis program, the user will have to open the terminal and type the following command without the quotes: "./Aegis file.csv". This will signal the program to execute. No other input from the user is required for the program to function properly. The results of the election will be displayed on the screen and both an audit and media file will be produced and stored inside the same directory as Aegis.

Developer/Tester:

In order to use the Aegis program, the user will have to open the terminal and type the following command without the quotes: "./Aegis file.csv" along with any flags that the user wants to trigger. This will signal the program to execute and run certain commands depending on the inputted flags. The results of the election will be displayed on the screen and both an audit and media file will be produced and stored inside the same directory as Aegis.

6.2 Screen Images

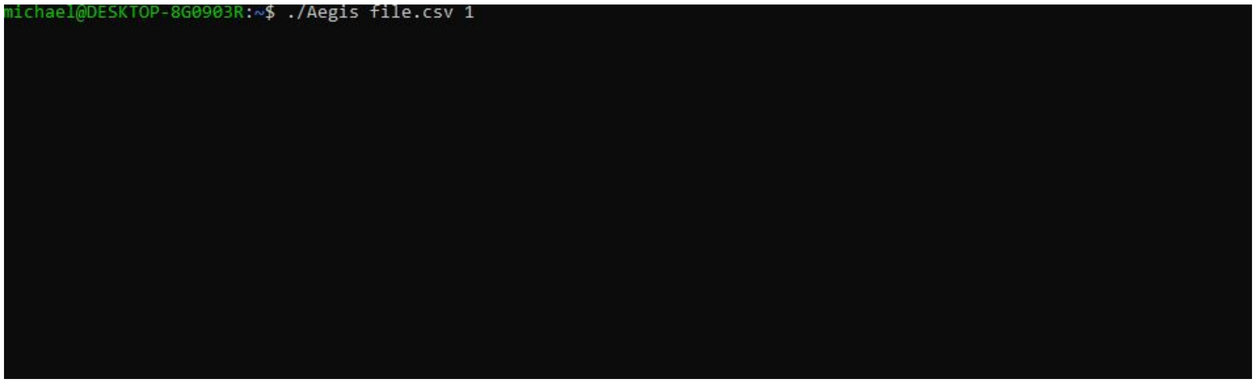
This displays an example of how the program is intended to run.



```
Microsoft Windows [Version 10.0.19041.804]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\micha>./Aegis file.csv
```

Figure 6: Example input for an election official.



```
michael@DESKTOP-8G0903R:~$ ./Aegis file.csv 1
```

Figure 7: Example input for a tester/developer.

6.3 Screen Objects and Actions

No screen objects or actions are provided in this project.

7. Requirements Matrix

7.1 VS_001: Use RNG to Determine Tie Breaker

Description: If there's a tie of votes in the system between two or more candidates, use RNG in order to decide the loser out of those candidates.
Requirement Type: Functional
System: The requirement is created in the VotingSystem class whenever there is a tie between two or more candidates. The requirement then uses RNG to isolate a person out of the group of candidates in order to decide in order to deal with that candidate.

7.2 VS_002: Audit File

Description: Program will compile all information about the decisions made throughout the program and write them to a file. This file will be titled with the date the election was processed.

Requirement Type: Functional

System: This requirement will be fulfilled in the VotingSystem class. The filename will be stored here, as well as the string written to the file.

7.3 VS_003: Run OPL

Description: Run the OPL algorithm to find results of election

Requirement Type: Functional

System: This requirement is created in the VotingSystem class. The system determines the type of the election after preprocessing the input file.

7.4 VS_004: Run IR

Description: Run the IR algorithm to find results of election

Requirement Type: Functional

System: This requirement is created in the VotingSystem class. The system determines the type of the election after preprocessing the input file.

7.5 VS_005: Group Independent Parties into One

Description: A feature in open party listing which groups the independents into a group.

Requirement Type: Functional

System: This requirement is created in the VotingSystem Class. The system will read all Candidates and group the independent parties into a singular party.

7.6 VS_006: Show Election Results

Description: Displays the stats of the candidates that participated in the election which includes the winners and the number of votes/ballots there were and also the type of election that occured.

Requirement Type: Functional

System: The requirement is created in the Voting System class. The system outputs on the terminal the statistics of the election. The statistics include the statistics of the candidates in the election which include the percentage of votes, if they won or lost, and other relevant information. Some other statistics would be general election information such the total amount of ballots in the election as well as the type of the election.

7.7 VS_007: Time Taken

Description: Prints out how long the program took to run from start to finish.

Requirement Type: Non-Functional

System: The requirement is created in the Voting System class. The system outputs the amount of time that it took to process all of the ballots and run the election algorithm.

7.8 VS_008: Media File

Description: On a complete run of an election's results, the results should be provided to the media for reporting. From this, a media file containing the winner(s) information and relevant ballot count will be provided to the media.

Requirement Type: Functional

System: This requirement is created in the VotingSystem Class. After calculating the results of the election, the system will create a file and store the winner(s) information and any other relevant information.

8. APPENDICES

No appendix current