Team 5

Sunshine Vote Aggregation System

Software Design Document

Cole Thompson(thom6401), Ioana Munteanu(munte029), Thomas Rooney(roone194), and Trevor Guy(guyxx080)

Team 5

February 28th, 2021

Table of Contents

1. Introduction

1.1. Purpose

This software design document (SDD) describes the architecture and design of the Sunshine Vote Aggregation System. The expected audience includes those present on the Sunshine development team, election officials of entities that make use of the Sunshine system, and any current or future personnel tasked with the maintenance of the voting system.

1.2. Scope

This document contains a complete and whole description of the design of the Sunshine Vote Aggregation System (SVAS). The SVAS will serve as a new system to determine winners in both Instant Runoff (IR) and Open Party List (OPL) type elections.

The GUI architecture will be created using Java Swing. Backend architecture will be described in the document.

1.3. Overview

The document's remaining chapters and their respective contents are highlighted below.

Section 2 contains a more in-depth description of the SVAS system in its entirety.

Section 3 is the Architectural Design that denotes all entities and subsystems present within the Sunshine System. First, there is a UML Class Diagram that visually shows the organization and system architecture.

Section 4 describes the data structures and their organization present within the system. It also contains a wholly complete definition of all classes and methods mentioned within the previous UML diagram.

Section 5 gives an insight into the component design with brief pseudocode examples of methods.

Section 6 gives a preview of the Graphical User Interface.

Section 7 contains the requirements matrix.

1.4. Reference Material

This document references the assignment requirements posted to canvas by Dr. Shana Watters. Additionally, the format was based upon the assignment template also provided by Dr. Watters. The preliminary GUI previews were made via Figma software.

1.5. Definitions and Acronyms

CSE	College of Science and Engineering
CSV	Comma Separated Values
GUI	Graphical User Interface

IR	instant runoff voting (a type of voting where majority wins)
OPL	open party list voting
UML	Unified Modeling Language
SVAS	Sunshine Vote Aggregation System

2. System Overview

The Sunshine Vote Aggregation System is a new software that will assist in the management of both Instant Runoff (IR) and Open Party List (OPL) type elections. The system will take in a CSV formatted file containing all necessary information about the election including election type, candidates, parties, and votes. After this, the system will process the information gained via the file and declare a winner(s). SVAS will prompt the user, most typically an election official, to input the CSV file. Then, all necessary computations will be done without any further input necessary from the user. The output will be delivered via two distinct files. An audit file will be produced with the election staff as the target audience. It will contain the winner(s) as well as intermediate calculations or eliminations. Additionally, a media file will be produced with the intent of notifying the media of the election results. This file will only contain the winner(s) of a given election.

3. System Architecture

3.1. Architectural Design

The SVAS is the only system that handles the aggregation and counting of the votes.

3.2. Decomposition Description

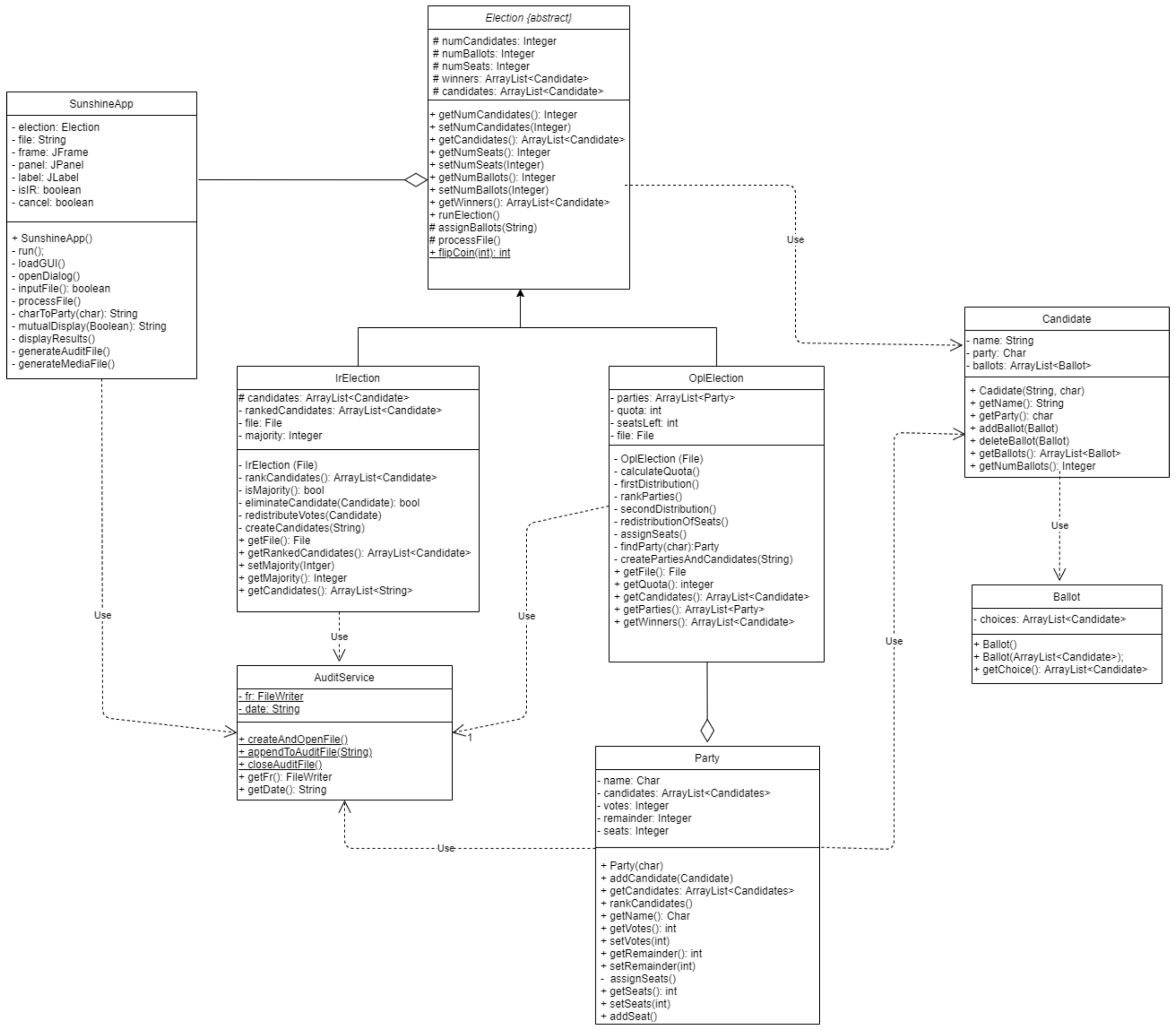

Figure 1: UML Diagram of the system
The UML diagram above describes the Sunshine Voting Aggregation System, including the classes, their attributes, and the relationships between the classes.

3.3. Design Rationale

To handle the aggregation and counting of votes we only used one system. By limiting the use to only one system, we are able to keep things more organized and simplified. Using multiple systems is also not necessary to complete the task.

4. DATA DESIGN

4.1. Data Description

The Sunshine Voting Aggregation System doesn't use any databases. The system and the election file will be stored locally on the user's machine.

4.2. Data Dictionary

4.2.1. Ballot

The ballot class contains information about the ballot, that is stored differently for the IR and the OPL election since people vote for multiple people in the order of their preferences in IR, but only for one person in OPL. Its attributes are:

- choiceIdx: Private integer that will be used only during the IR election. If no majority is reached in the election, then the votes of the least favorite candidate are redistributed based on the voter's next preference. This index is used to keep track of the number of the voter's preferences currently taken into consideration. The first choice on the ballot will have the index 0.
- choices: Array list of integers that will be used only during the IR election, that will store the voter's preference. It will consist of all the integers from 0 to (the number of candidates - 1), in the order of the voter's preference. If a voter's choice is n, then they voted for the n-th candidate in the candidates array list in the IrElection class. The choices array list will be generated during the processing of the input file. Since the first candidate on the physical ballot is numbered with 1, but the first index on the ballot in our system is 0, all of the opinions expressed on the physical ballot will need to be decremented by 1.

The methods of the ballot class are:

- ballot(): Public constructor for the ballot in the OPL election. Its attributes will never be used. A ballot will be created in OPL only for counting purposes.
- ballot(choices): Public constructor for the ballot in the IR election, that has the array list of choices as a parameter. It will initialize the choice attribute.
- getChoice(): Public method that returns the current choiceIdx and then increments the index by 1, to move to the next choice.

4.2.2. Candidate

The candidate class stores information about both the IR and OPL candidates, such as their name, the party they belong to, and the ballots in their favor. Its attributes are:

- ballots: Private array list of ballots that are in favor of the candidate at the current time. For IR, if a ballot is in this array, then it means that the current choice on the ballot is in favor of this candidate. If the candidate is eliminated, all of their ballots will be transferred to the next candidate in order of the preference expressed on the ballot, if there is any preference expressed.

- name: Private string that represents the name of the candidate.
- party: Private character that represents the party the candidate belongs to. D means “democratic”, R means “republican”, I means “independent”, L means “liberal.”

The methods of the candidate class are:

- addBallot(Ballot): Public method that takes a ballot as an argument and adds it to the candidate’s array list of ballots. It doesn’t return anything.
- delete(Ballot): Public method that takes a ballot as an argument and deletes it from the candidate’s array list of ballots. It doesn’t return anything.
- getBallots(): Public method that returns the array list in favor of the candidate.
- getName(): Public method that returns the name of the candidate, as a string.
- getNumBallots(): Public method that returns the number of ballots in favor of the candidate, as an integer.
- getParty(): Public method that returns the name of the candidate’s party, as a char.
- setName(String): Public method that sets the candidate’s name. It takes a string as a parameter and assigns it to the name attribute. It doesn’t return anything.
- setParty(Char): Public method that sets the candidate’s party. It takes a character as a parameter and assigns it to the party attribute. It doesn’t return anything.

4.2.3. Election

The election class is an abstract class that contains information about the election such as the number of candidates, the number of ballots, number of seats, winners. Additionally, it runs both types of elections (IR and OPL), assigns ballots to candidates, and deals with ties. Its attributes are:

- numBallots: Protected integer representing the number of ballots in the election. This attribute will be assigned a value when the input file is processed.
- numCandidates: Protected integer representing the number of candidates in the election. This attribute will be assigned a value when the input file is processed.
- numSeats: Protected integer representing the number of seats in the election. This attribute will be assigned a value when the input file is processed.
- winners: Protected array list of candidates containing the winners of the election. This attribute will be assigned a value when the election is run.

The methods of the election class are:

- addWinner(Candidate): Public method that takes a candidate as a parameter and adds it to the winners array list. It doesn’t return anything.
- allSeatsAssigned(): Public method that returns a boolean: true if all of the seats in the election have been assigned and false otherwise. This will be used in order to know when to stop running the election.
- assignBallots(): Public method that goes through all ballots in the input file, creates a ballot object, and assigns it to the corresponding candidate. It doesn’t return anything.
- flipCoin(Integer): Public method that is used when a tie between two or more candidates happens and a winner or a loser needs to be decided. A tie could happen:
    - In IR

        - When no redistribution can be done because there are two candidates left who have the same number of votes; a coin will be "flipped" once to decide the winner.
        - When two or more candidates have the least number of votes and one of them needs to be eliminated; a coin will be "flipped" once to decide the loser.
    - In OPL
        - When distributing the seats in a party, multiple candidates in the same party have the same number of votes; a coin will be "flipped" multiple times to decide the winners until all of the seats in that party have been occupied.
        - When distributing the seats between parties based on the remainders, multiple parties have the same number of remaining votes; a coin will be "flipped" multiple times to decide the winners until all of the seats in that party have been occupied.

The flipCoin() method will take an integer as an argument that represents how many candidates or parties tied. It will randomly generate an integer between 0 and (the number of candidates or parties -1), corresponding to the winner/loser, depending on the situation. The integer will indicate the offset between the position of the winner/loser and the position of the first candidate/party in the array generated using rankCandidates/rankParties.

- getNumBallots(): Public method that returns the number of ballots in the election, as an integer.
- getNumCandidates(): Public method that returns the number of candidates in the election, as an integer.
- getNumSeats(): Public method that returns the number of seats in the election, as an integer.
- getWinners(): Public method that returns the winners of the election, as an array list of candidates.
- runElection(): Public abstract method that runs the algorithms for the chosen type of election. It doesn't return anything.
- setNumBallots(Integer): Public method that sets the number of ballots of the election. It takes an integer as a parameter and assigns it to the numBallots attribute. This will be used during the file processing. It doesn't return anything.
- setNumCandidates(Integer): Public method that sets the number of candidates of the election. It takes an integer as a parameter and assigns it to the numCandidates attribute. This will be used during the file processing. It doesn't return anything.
- setNumSeats(Integer): Public method that sets the number of seats of the election. It takes an integer as a parameter and assigns it to the numSeats attribute. This will be used during the file processing. It doesn't return anything.
- updateAudit(string): Public method that writes to the audit file. Accepts a string as input that will be written to the audit file. Returns null.
- updateMedia(string): Public method that writes to the media file. Accepts a string as input that will be written to the media file. Returns null.

4.3.4. IrElection

The class IrElection inherits the abstract class Election. Its attributes are:

- candidates: Protected array list of candidates, in the order of the input file.
- rankedCandidates: Private array list of candidates, in descending order of the number of votes they got.

Its methods are:

- eliminateCandidate(): Private method that finds the candidate with the least number of votes, redistributes their ballots based on the following choice expressed on the ballot to the remaining candidates, and then deletes the candidate from the rankedCandidate array. It will return true if a candidate was eliminated and false otherwise.
- getCandidates(): Public method that returns the candidates in the election, as an array list of candidates.
- isMajority(): Private method that will return true if any candidate has reached the majority of the votes and false otherwise.
- rankCandidates(): Private method that will rank the candidates of an IR election based on the number of votes they got, in descending order. It will return an array list of candidate objects.
- redistributeVotes(Candidate): Private method that takes a candidate as a parameter and redistributes their votes based on the following choice expressed on the ballot to the remaining candidates candidates
- setCandidates(): Public method that sets the candidates of the election. It takes an array list of their ballots as a parameter and assigns it to the candidates attribute. This will be used during the file processing. It doesn't return anything.

4.2.5. OplElection

The class OplElection inherits the abstract class Election. Its attributes are:

- parties: Private array list of party objects of all the parties in the election. This attribute will be assigned a value when the input file is processed.
- quota: Private integer representing the quota for the OPL election.

Its methods are:

- assignSeats(): Private method that will loop through all of the parties and will assign seats within each party, using the assignSeats() method in the party class. It will not return anything.
- calculateQuota(): Private method that calculates the quota by taking the total number of votes in the election and dividing it by the total number of seats. The method assigns the results to the quota attribute and also returns it as an integer.
- firstDistribution(): Private method that calculates the remainders by dividing the number of votes each party has by the quota and assigning the quotient to the seats attribute of that specific party. It also assigns the remaining votes to the remainder attribute of that specific party. It will not return anything.
- getQuota(): Public method that will return the quota of the election as an integer.
- rankParties(): Private method that ranks the parties in descending order based on the remaining votes they have. It will return an array list of integers. The integers correspond to the indexes of parties in the candidates array list of the class.
- seatsLeft(): Private method that returns the number of seats in the election that haven't been assigned after the first distribution. It will return an integer.
- secondDistribution(): Private method that will be used only if there are seats left. It assigns one seat to each party as their in the order of the rankParties array list, until there are no more seats left. If two or more parties have the same number of remainders, but the number of seas left is less than the

number of parties, then we will repeatedly use the flipCoin method to determine the winners. It will not return anything.

4.2.6. Party

The class OplElection has a Party class that contains information such as the name of the party, the candidates belonging to the party, and how many votes and seats the party got. The Party class has the following attributes:

- candidates: A private array list of candidate objects. This contains all candidates that are present within the election that belongs to a specific party.
- name: A private char that represents which political party is associated with a specific party class. D means "democratic", R means "republican", I means "independent", L means "liberal."
- remainder: A private integer that denotes the number of votes a party has after the first allocation of seats.
- seats: A private integer that represents the number of seats won by a party
- votes: A private integer that denotes the number of votes awarded to a given party.

Its methods are:

- addCandidate(): Public method that adds a Candidate object into the candidates array list attribute. It takes a Candidate object as input and has a null return.
- getCandidates(): Public method that "gets" the array list of candidates. It will return a candidate object.
- getName(): Public method used to "get" the name of the party object. It will return the same char present within the name attribute.
- getRemainder(): Public method used to "get" the remainder of votes after the first allocation. It will return an integer that corresponds to the integer number of votes remaining following the first allocation.
- getVotes(): Public method used to "get" the number of votes received by a specific party. It will return an integer that is equal to the number of votes a party has received.
- rankCandidates(): Public method that will rank the Candidate objects within the candidates array list object. Afterwards, the candidates object will be sorted. Candidate objects with more votes will be indexed lower than their counterparts with less votes. It will return the sorted candidates object.

4.2.7. SunshineApp

The SubshineApp class is the class that will trigger the election, will display the results, and will generate the audit and media files. Its attributes are:

- election: A private variable that stores the IR or OPL election.
- file: A private variable that stores the name of the input file.

Its methods are:

- displayResults(): A private variable that displays on the screen the results of the election, once the election is done.

- inputFile(): A private method that asks the user to input the file name and then stores it in the file attribute. It will not return anything.
- generateAuditFile(): A private method that generates the audit file of the election, once the election is done. It will not return anything.
- generateMediaFile(): A private method that generates the media file of the election, once the election is done. It will not return anything.
- processFile(): A private method that assigns the number of ballots, number of candidates, number of seats to the election attributes. It also assigns the ballots to the candidates/parties, using the election methods. It will not return anything.

Class	Attribute Name	Attribute Type	Attribute Size	Description
Ballot	choiceIdx	Integer	2	The index of the voter's choice expressed on the ballot. The smaller the index is, the more priority the choice gets. The first index is 0.
	choices	ArrayList<Integer>	100	The voter's choices expressed on the ballot, in the order of their preference.
Candidate	ballots	ArrayList<Ballot>	1000000	The ballots that are in favor of the candidate.
	name	String	20	The name of the candidate.
	party	Char	1	The party the candidate belongs to.
Election	numBallots	Integer	1000000	The number of ballots in the election.
	numCandidates	Integer	100	The number of candidates in the election.
	numSests	Integer	100	The number of seats in the election.
	winners	ArrayList<Candidate>	100	The winners of the election, in descending order of the votes received.
IrElection	candidates	ArrayList<Candidate>	100	The candidates in the IR election, in the order of the input file.
	rankedCandidates	ArrayList<Candidate>	100	The candidates in the IR election, in the descending order of the

				number of votes they got.
OplElection	parties	ArrayList<Party>	20	The parties in the OPL election.
	quota	Integer	1000000	The quota in the OPL election.
Party	candidates	ArrayList<Candidate>	100	The candidates in the IR election, in the order of the input file.
	name	Char	1	The name of the party.
	remainder	Integer	1000000	The remainder of votes of this party after the initial distribution in OPL.
	seats	Integer	100	How many seats this party currently has.
	votes	Integer	1000000	How many votes the party got during the initial allocation.
SunshineApp	election	Election	N/A	The IR/OPL election that the calculations will be made for.
	file	String	100	The name of the file containing the election information.

5. Component Design

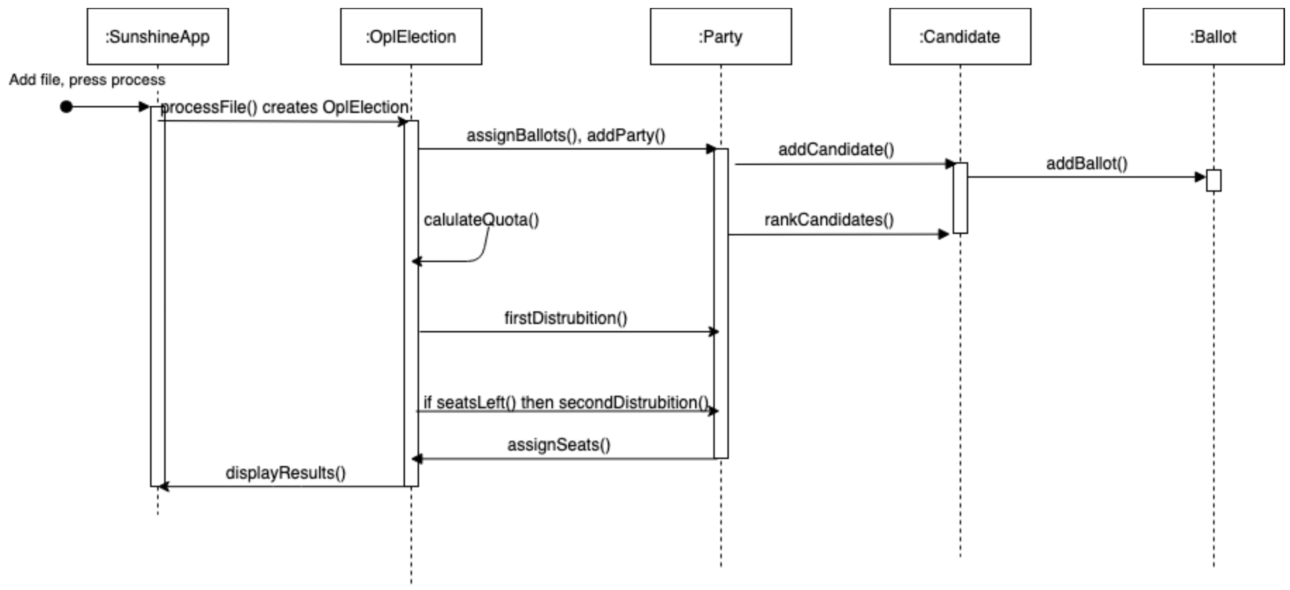

Figure 2: Sequence diagram for Open Party List Voting

This diagram is the sequence diagram for the Sunshine Vote Aggregation System when it runs an Open Party List Election. It includes all the classes and functions involved when an OPL Election has taken place.

- Sunshine App
  - run():
    - Load the screen for the user
    - inputFile()
    - If file != null do processFile() else return
    - displayResults()
  - inputFile():
    - Ask the user for an input file, iFile.
    - If iFile is valid file then file = iFile
  - processFile():
    - election = file.getLine() == “ir” ? new irElection() : new oplElection()
    - generateAuditFile()
    - election.runElection()
    - displayResults()
    - generateMediaFile()
  - displayResults():
    - Load results to the screen for the user
    - Load export buttons for audit and media files
  - generateAuditFile():
    - Create the audit file that will be written to throughout program
  - generateMediaFile():
    - Write media results to new media file

- Election
    - Election is an abstract class.
    - runElection():
        - secondLine = file.getline()
        - secondLine = secondLine.split()
        - For c in secondLine:
            - If opl:
                - Find party
                - If party does not exists do: create party; add to parties;
                - Create candidate; create ballot; assign ballot to candidate; add to party;
            - If ir:
                - assignBallots()
        - If ir:
            - rankCandidates()
            - If isMajority():
                - addWinner(candidates[0])
            - Else if candidates.length > 2:
                - If distinct loser:
                    - eliminateCandidate[candidates.length]
                - Else:
                    - minus = flipCoin(# of candidates tied for last)
                    - eliminateCandidate[candidates.length - minus]
                - reassignVotes()
            - Else:
                - AddWinner(candidates[flipCoin(2)])
        - Else:
            - calculateQuota()
            - firstDistribution()
            - If seatsLeft() do secondDistribution()
            - For party in parties:
                - party.candidates.rankCandidates()
                - i = 0
                - For seats in party:
                    - addWinner(party.candidates[i])
                    - i++
    - assignBallots():
        - While (file.getline() != EOF):
            - Create Ballot
            - If opl:
                - addBallot to candidate; add candidate to Party in parties
            - Else:
                - addBallot to candidate in candidates
    - addWinner(Candidate):
        - winners.add(Candidate)
    - flipCoin(Integer):
        - Return floor(randomNumber(Integer))

- OplElection
    - OplElection is a class that inherits from Election.

- calculateQuota():
    - Return total votes cast / seats that need to be filled
- firstDistribution():
    - For party in parties:
        - party.seats = party.votes / quota
        - party.remainder = party.votes % quota
- seatsLeft():
    - Return numSeats - winners.length
- rankParties():
    - Return indexes corresponding to the sorted array of parties based on the remainder attribute (descending)
- secondDistribution():
    - Assign one seat per party in the order of the array generated by rankParties
    - flipCoin multiple times for ties to determine winners
- assignSeats():
    - Loop through parties; party.assignSeats()

- IrElection
    - IrElection is a class that inherits from Election.
    - isMajority():
        - If Candidates[rankedCandidates[0]].numVotes > election.numVotes/2, then true
    - eliminateCandidate(Candidate):
        - If Candidate in candidates:
            - redistributeVotes(Candidate)
            - Return candidates.remove(Candidate)
        - Return true
    - redistributeVotes(Candidate):
        - For ballot in Candidate.ballots:
            - Find alternative vote
            - Find candidate who received alternate vote
            - Add vote for candidate
- Candidate
    - addBallot(Ballot):
        - ballots.add(Ballot)
    - getNumBallots():
        - Return ballots.length

- Party
    - addCandidate(Candidate):
        - party.candidates.add(Candidate)
    - rankCandidates():
        - Party.candidates.sort by ballots from greatest to least

- Ballot
    - getChoice():
        - choice = choices[choiceIdx]
        - choiceIdx ++
        - Return choice

6. HUMAN INTERFACE DESIGN

6.1. Overview of User Interface

The user will open the system after an election has occurred and be prompted to input the election file. This will be into an input box on the screen. Then, after the votes have been aggregated the results will be shown onto the screen with options for viewing and downloading the audit file and media file.

6.2. Screen Images

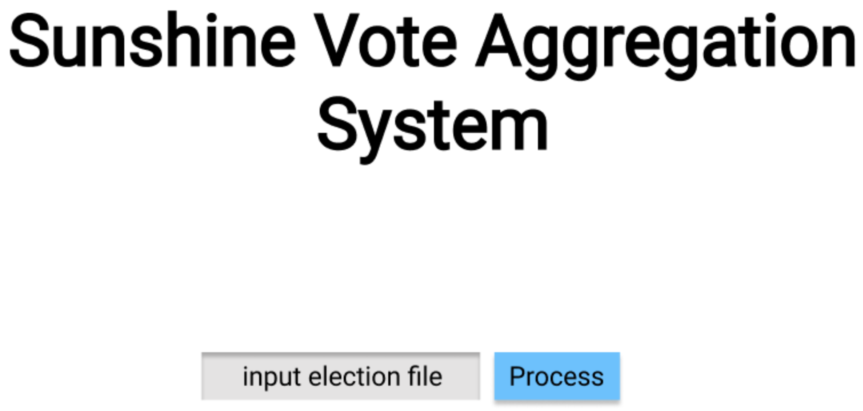

Figure 3: Initial UI that allows a user to input a file

This diagram illustrates the initial user interface that consists of a form with an input field and a process button, which submits the form.

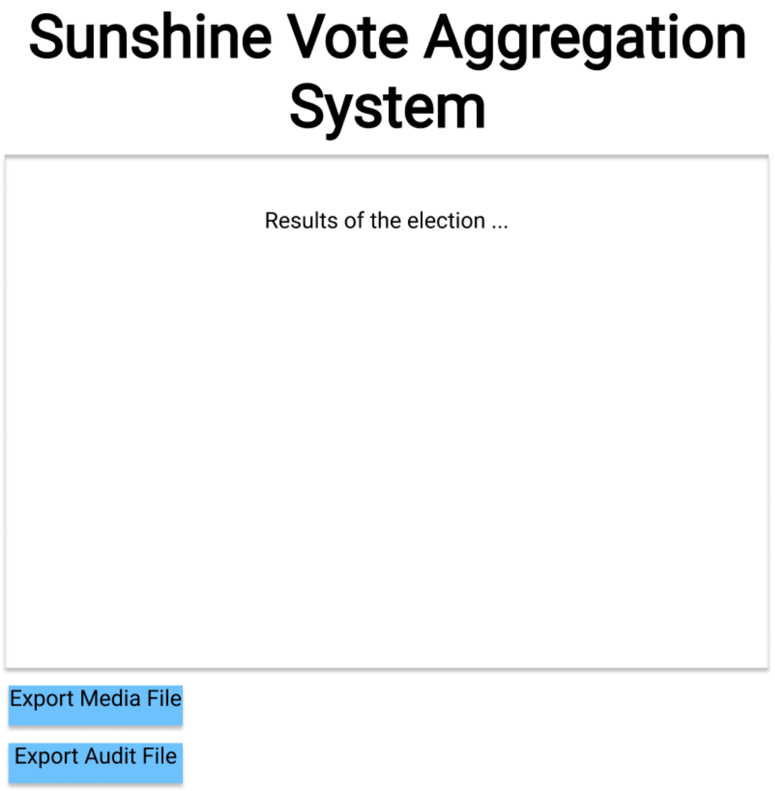

Figure 4: UI that presents the results to the user

This diagram is the final user interface that will come up after the Sunshine Vote Aggregation System has run. It displays the results and shows buttons for exporting the media and audit files.

6.3. Screen Objects and Actions

Figure 1 includes one form with an input box for the file and a process button to submit the form. An election official will only have the option to input the election file and press process. If they input an invalid file then the process button will not let them submit it. In figure 2 the election results are displayed to the official as well as the option to export the media and audit files.

7. Requirements Matrix

7.1. ID: IDENTIFY_FILE_001

Description: The user inputs the name of the CSV file containing election data.

System: When the user opens the SunshineApp, they will be prompted to enter the file name and submit it. Using the method inputFile(), the file name will be associated with the file attribute of the SunshineApp class.

7.2. ID: PROCESS_FILE_IR_001

Description: The input file containing the election data is processed. Candidate objects are created for use with future functions with correct attributes such as name, party, and current votes.

System: Using the processFile() in the SunshineApp class, the number of candidates, ballots, and seats will be assigned to the election attributes. Additionally, ballots will be assigned to candidates, and for the OPL election, the candidates will be assigned to parties.

7.3. ID: CHECK_WINNER_IR_001

Description: An election official checks if there is a winner in an IR election.

System: The system will utilize isMajority() to check if a winner should be declared.

7.4. ID: ELIMINATE_UNPOPULAR_IR_001

Description: An election official checks which candidate has the least number of votes in an IR election and redistributes their votes to the remaining candidates.

System: The system will call eliminateCandidate(Candidate) to eliminate the candidate passed into, this will be the candidate that is least popular or tied for least popular.

7.5. ID: PROCESS_FILE_OPL_001

Description: The input file containing the election data is processed. Candidate objects are created for use with future functions with correct attributes such as name, party, and current votes. Party objects are instantiated containing attributes that track the name of the party and the sum of votes received for all candidates that belong to it.

System: Using the processFile() in the SunshineApp class, the parties, candidates, ballots, and seats will be assigned to the election attributes. Additionally, the candidates will be assigned to parties.

7.6. ID: GROUP_INDEPENDENTS_001

Description: Independents that are present within a given OPL election will be grouped into a single party.

System: Our system will automatically consider independent candidates part of a single party, that will have the name I. This will happen during the file processing. They will be treated as any other party.

7.7. ID: CALCULATE_QUOTA_OPL_001

Description: An election official calculates the quota for OPL.

System: Using the calculateQuota method in the OplElection class, the quota will be determined and assigned to the quota attribute of the class.

7.8. CALCULATE_ALLOCATIONS_AND_REMAINDERS_OPL_001

Description: An election official calculates the allocation of seats and remaining votes for OPL.

System: The method firstDistribution() will be called from the OplElection class. This method will calculate remainders by using a modulo function. The number of votes a party receives will be modded by the previously calculated quota. The answer to this represents the number of seats immediately allocated to a given party. The remainder can be calculated by taking the total number of votes less the quantity represented by the quota multiplied by the number of seats awarded. This remainder will be stored in the Party.remainder attribute. The remainders will be used during the secondDistribution() method to allocate any seats that may remain after the conclusion of the first allocation.

7.9. ID: ID: FLIP_A_COIN_001

Description: A coin is flipped if there is a tie.

System: Whenever a tie occurs that needs to be broken, (see activity diagrams in Appendix) a coin will be flipped to determine a winner or loser. The flipCoin() method from the abstract Election class will be called, a winner/loser will be determined, and the next decision will occur as highlighted in the activity diagrams.

7.10. ID: DISPLAY_WINNER_001

Description: If an IR election takes place, then 1 winner will be displayed to the screen at the end of the counting process. If it's an OPL election, multiple winners will be displayed to the screen equal to the number of seats available. Other information about the election will also be displayed (type of election, number of seats).

System: Following the conclusion of an election run, the winner(s) of the election will be displayed on the screen. The getWinners() method from the abstract Election class will be called. The names of the candidate objects can then be grabbed via the getName() method found within the Candidate object. Finally, the name(s) will be displayed to screen for the user to see.

7.11. ID: PRODUCE_AUDIT_001

Description: After the system is run, an audit file with information on the election is produced.

System: There are several criteria that warrant an update to the audit file. These include, but are not limited to, assigning votes, eliminating candidates, and declaring a winner. Upon one of these criteria being met, updateAuditFile() will be called from the abstract Election class. A string will be inputted containing a message related to the criteria being met. This string will be appended to the audit output file.

7.12. ID: MEDIA_REPORT_001

Description: After the system is run, a media report is produced with information about the election for the media.

System: Following the completion of an election run, the updateMediaFile() function from the abstract Election class will be run. A string will be inputted containing the winner(s) of the election. This string will then be written to the media output file.

8. APPENDICES

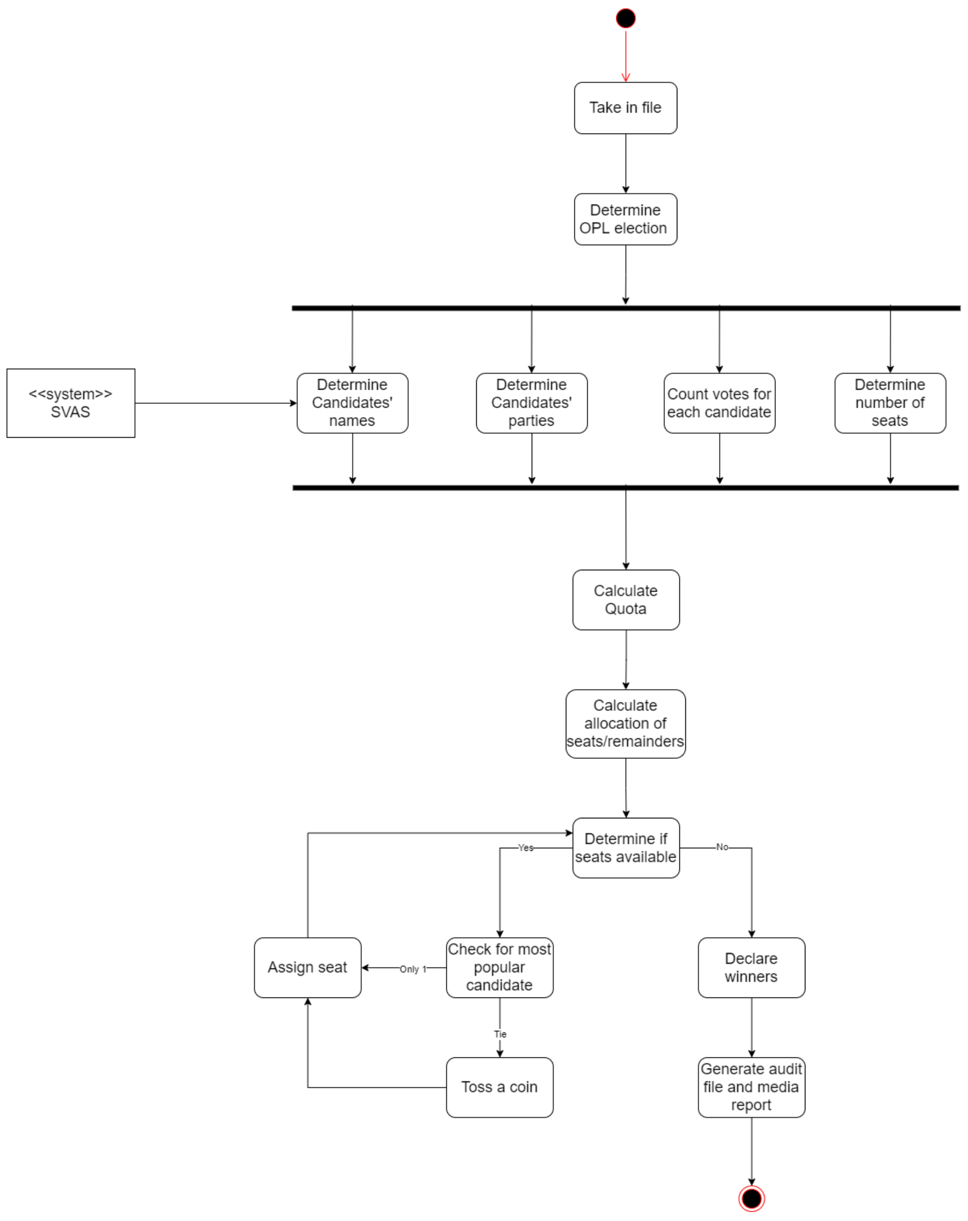
Take in file
Determine OPL election
<<system>> SVAS
Determine Candidates' names
Determine Candidates' parties
Count votes for each candidate
Determine number of seats
Calculate Quota
Calculate allocation of seats/remainders
Determine if seats available
Yes
No
Check for most popular candidate
Only 1
Assign seat
Tie
Toss a coin
Declare winners
Generate audit file and media report

Figure 5: Activity Diagram for OPL Election

This diagram shows the flow of events for an OPL election. It can be seen the file is inputted, and it has been determined that it is an OPL election. Then, the diagram highlights the flow of decisions and shows an overview of events that will prompt function calls.

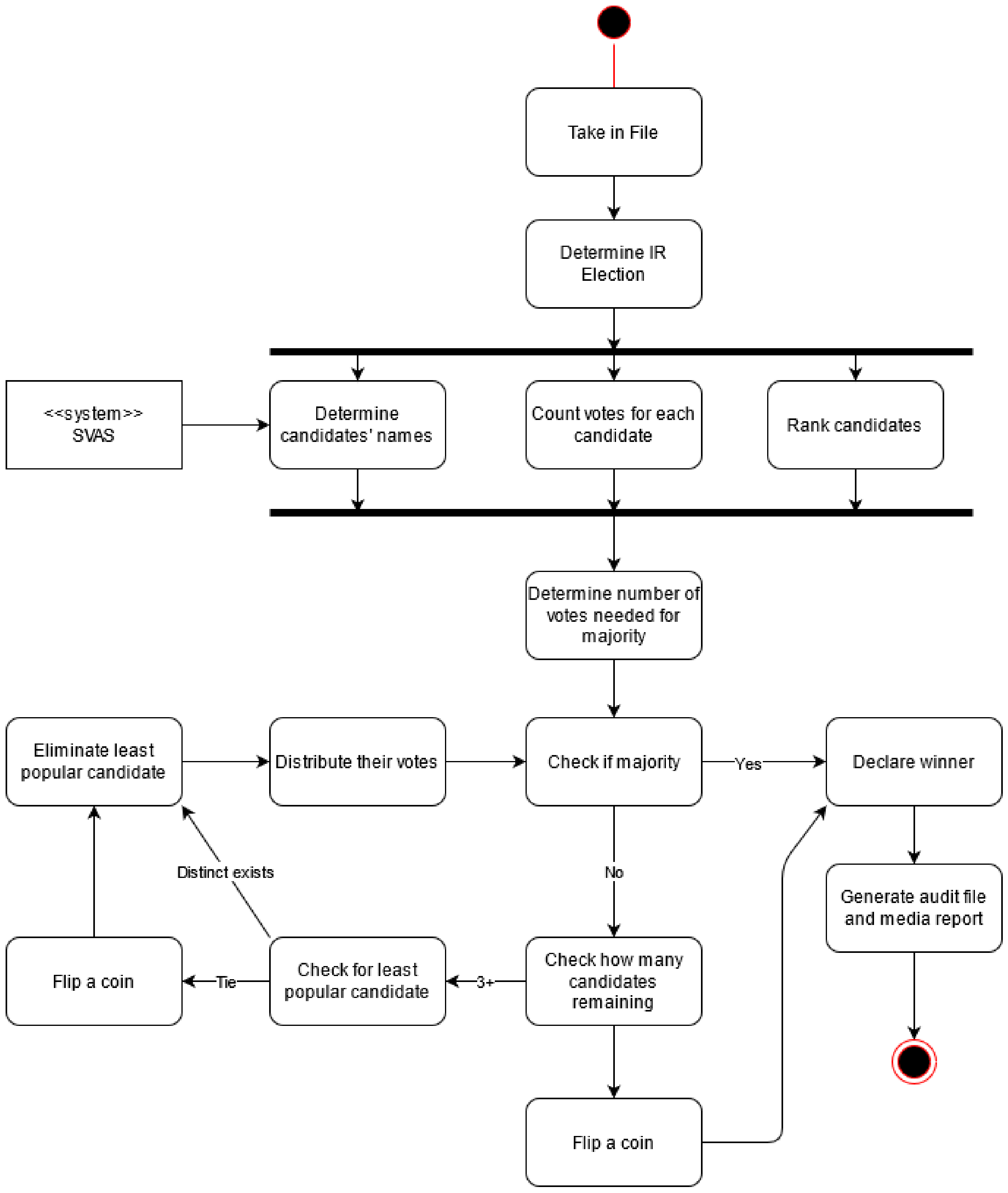
Take in File
Determine IR Election
<<system>> SVAS
Determine candidates' names
Count votes for each candidate
Rank candidates
Determine number of votes needed for majority
Eliminate least popular candidate
Distribute their votes
Check if majority
Yes
Declare winner
Distinct exists
No
Generate audit file and media report
Flip a coin
Tie
Check for least popular candidate
3+
Check how many candidates remaining
Flip a coin

Figure 6: Activity Diagram for IR Election
This diagram shows the flow of events for an IR election. It can be seen the file is inputted, and it has been determined that it is an IR election. Then, the diagram highlights the flow of decisions and shows an overview of events that will prompt function calls.