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 2
Voting System

Software Design Document

Name (s): Aahan Tyagi, Alexander Grenier, Dominic Deiman, Jackie Li
Lecture Section: 001
Group Number: 2

Date: (03/03/2023)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This Software Design Document contains details and diagrams showing the design of the software voting system. The system design diagrams are to show the process that the program follows throughout each run.

The intended audience of this document is for the developers who will be working on the project, and the election officials who will be using the system. Along with that, anyone that wants to gain a deeper understanding of how voting systems work will benefit from these design documents.

1.2 Scope

This document contains the description of the design of the voting system. The voting system will take in a CSV file containing data of election ballots, and it will determine the winner of the election. It can follow either CPL or IR voting algorithms. The goal of this voting system is to safely and efficiently count votes. The system should remain untampered and keep track of votes without issue. The objective of this system is to speed up the process of counting votes. The benefits is that votes will no longer need to be manually counted and can be counted extremely quickly by being input into this system.

1.3 Overview

This document contains details and diagrams showing the design of the software voting system. There will be a system overview, and descriptions of the system architecture, the data design, the component design, the human interface design, and the requirements matrix. At the start, there is a brief overview and high view on the whole document. Going into the second section, there is another section that goes into a little more detail about the functionality of the system. Following that, the system architecture and how the program is designed is next. Along with the architecture, the thought process behind the architecture is also in the third section. The fourth section explains how the system stores and processes information and has a dictionary that will provide details about certain definitions. The fifth section goes into more depth about each different component. The sixth section is all about human interface design. The seventh section talks about how this document relates back to the SRS.

1.4 Reference Material

Voting SDD

SDD template

2. SYSTEM OVERVIEW

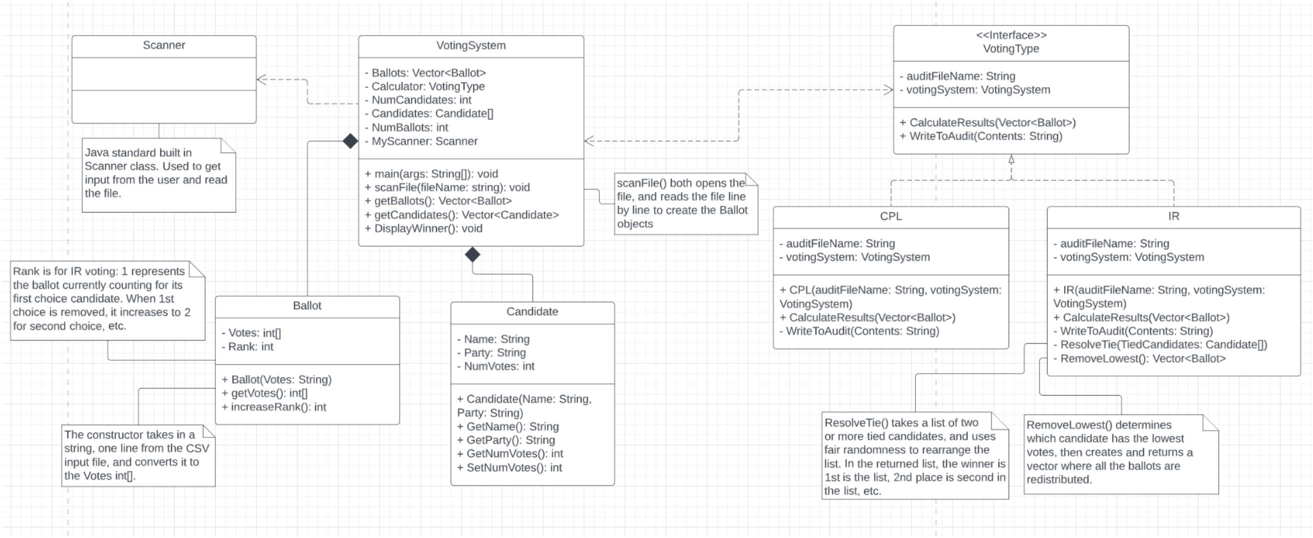
Voting System is a straightforward application that has been designed to give an elected official control over two different forms of voting in which a candidate or party will be elected upon the number of ballots received for each respectively. The two systems that can be used are instant runoff voting (IR) and closed party list (CPL). The IR form of voting tells the voter to choose each candidate in order of superiority with their most preferred candidate being labeled as number 1, their second in command as number 2, and so on. If a candidate receives over 50% of the number 1 votes, they are elected; however, if there is no majority, then the candidate with the fewest first place votes is eliminated. These ballots are then converted to first place votes for their second place candidate. This loops until a candidate has a 50% majority, and a winner is then selected. For CPL the process is different: voters are given a list of candidates and must vote for a candidate rather than a party. The votes for said candidate will then also count for their party. The party with the most votes wins overall, and the candidates in the party are elected based on their own counts. Ties are broken in both forms of voting with the flip of a coin.

To start the count, the user must select which style of voting they prefer. In the case of IR, the ballots are pulled in from a CSV file and are counted. Then it determines if a candidate has 50% or more of the total votes. If so, this person is elected, otherwise a loop goes through and repeats the previous step. In the event of ties, integers are chosen to represent the candidates, and a random integer is selected. The winner is then determined, the system produces an audit file, and the system exits the program. If CPL is selected, the only difference is that the program determines the party as the winner based on the votes. Afterwards, Voting System determines the order of candidates elected by the counts from the ballot, and then it sorts them in order of most voted for to least.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The following UML Class diagram shows all of the classes in the system, the member variables and functions that each class has, and the relationship between the classes.


Scanner
Java standard built in Scanner class. Used to get input from the user and read the file.
VotingSystem
- Ballots: Vector<Ballot>
- Calculator: VotingType
- NumCandidates: int
- Candidates: Candidate[]
- NumBallots: int
- MyScanner: Scanner
+ main(args: String[]): void
+ scanFile(fileName: string): void
+ getBallots(): Vector<Ballot>
+ getCandidates(): Vector<Candidate>
+ DisplayWinner(): void
scanFile() both opens the file, and reads the file line by line to create the Ballot objects
<<Interface>>
VotingType
- auditFileName: String
- votingSystem: VotingSystem
+ CalculateResults(Vector<Ballot>)
+ WriteToAudit(Contents: String)
Rank is for IR voting: 1 represents the ballot currently counting for its first choice candidate. When 1st choice is removed, it increases to 2 for second choice, etc.
Ballot
- Votes: int[]
- Rank: int
+ Ballot(Votes: String)
+ getVotes(): int[]
+ increaseRank(): int
The constructor takes in a string, one line from the CSV input file, and converts it to the Votes int[].
Candidate
- Name: String
- Party: String
- NumVotes: int
+ Candidate(Name: String, Party: String)
+ GetName(): String
+ GetParty(): String
+ GetNumVotes(): int
+ SetNumVotes(): int
CPL
- auditFileName: String
- votingSystem: VotingSystem
+ CPL(auditFileName: String, votingSystem: VotingSystem)
+ CalculateResults(Vector<Ballot>)
- WriteToAudit(Contents: String)
IR
- auditFileName: String
- votingSystem: VotingSystem
+ IR(auditFileName: String, votingSystem: VotingSystem)
+ CalculateResults(Vector<Ballot>)
- WriteToAudit(Contents: String)
- ResolveTie(TiedCandidates: Candidate[])
- RemoveLowest(): Vector<Ballot>
ResolveTie() takes a list of two or more tied candidates, and uses fair randomness to rearrange the list. In the returned list, the winner is 1st is the list, 2nd place is second in the list, etc.
RemoveLowest() determines which candidate has the lowest votes, then creates and returns a vector where all the ballots are redistributed.

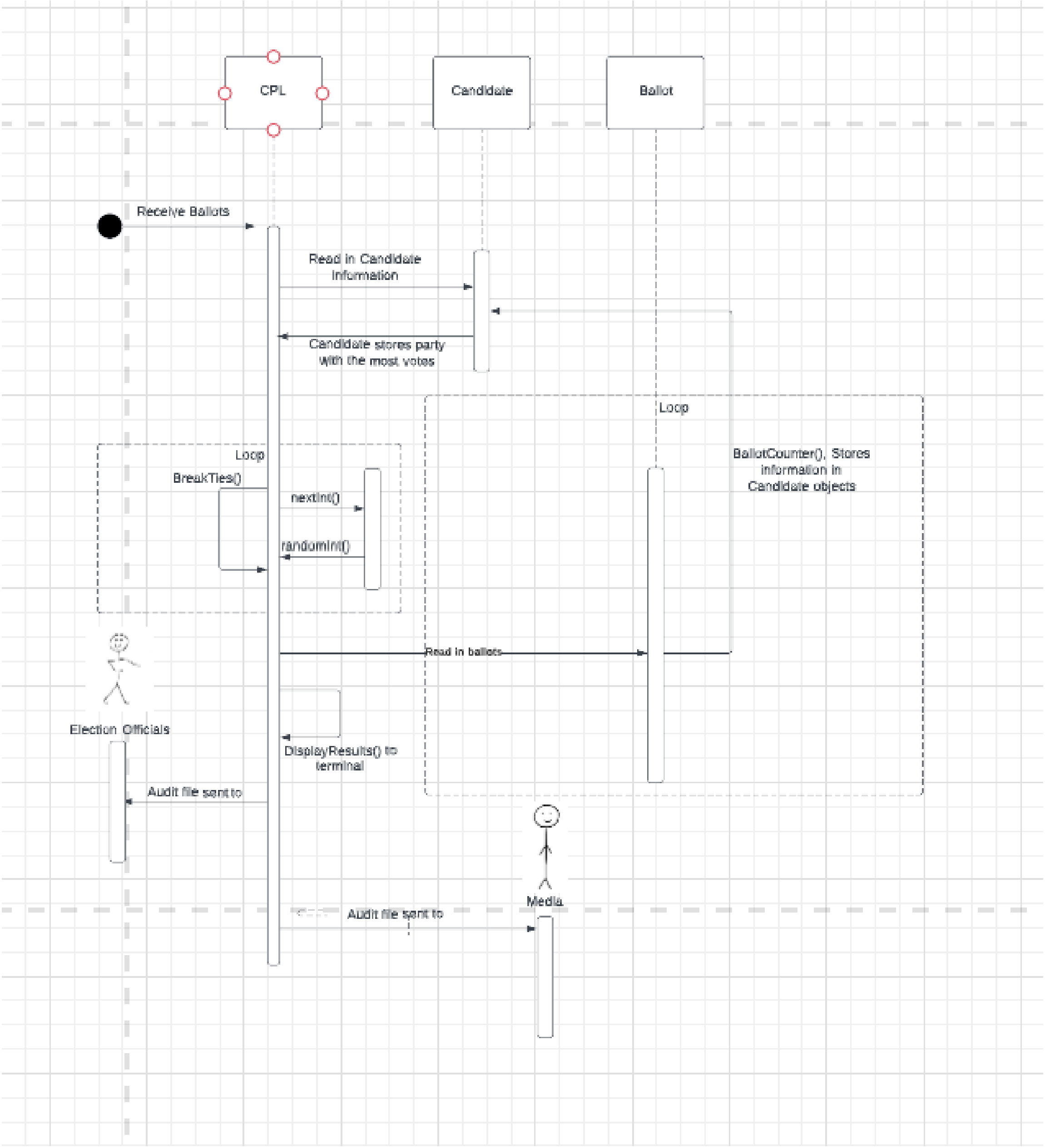
The voting system can be designed using a client-server architecture. The client application can be a web-based application that allows voters to cast their votes securely from anywhere with internet access. The server-side application can handle the vote counting process, store data, and provide a dashboard for administrators to manage the voting process.

Here's a breakdown of the components of the voting system architecture:

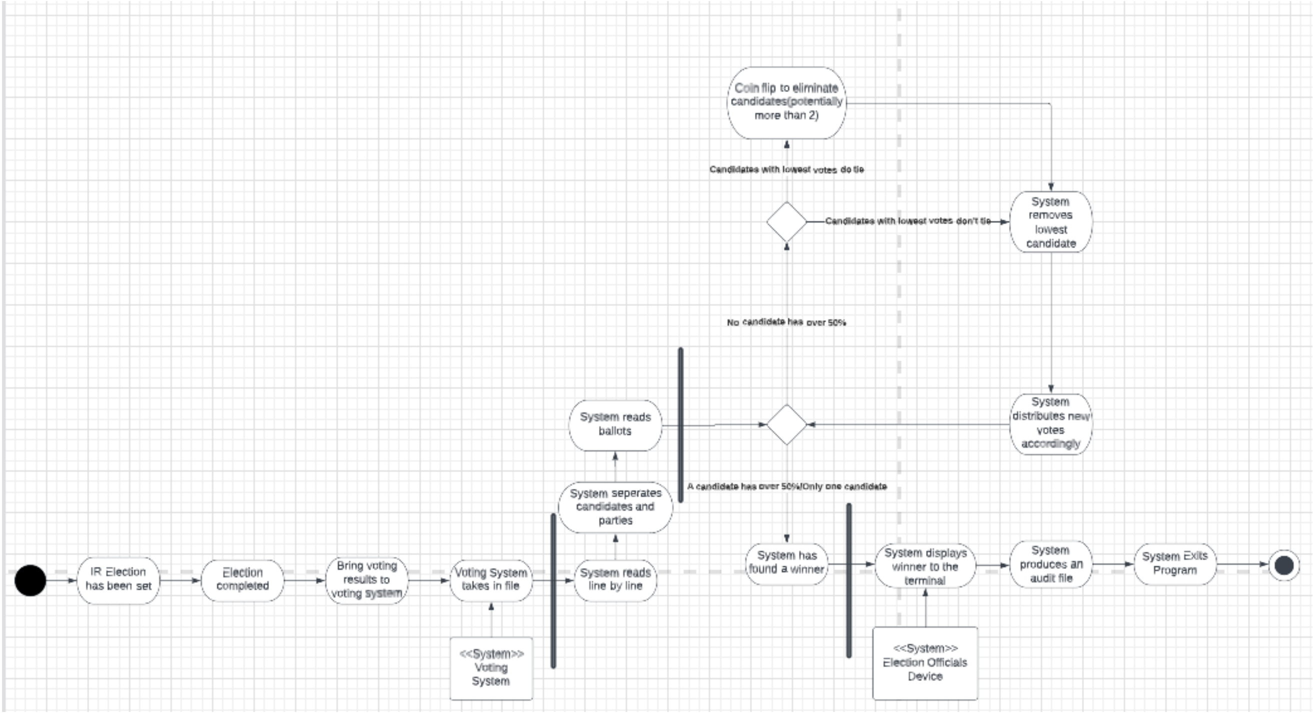
1. Client-side application: This component will be responsible for providing a user interface for voters to cast their vote. The client-side application will be a web-based application that will be accessible from any device with internet access.
2. Authentication and Authorization: This component will handle user authentication and authorization. It will ensure that only authorized users can access the voting system and cast their vote.
3. Communication Channel: This component will manage communication between the client-side application and the server-side application. It will ensure that all data exchanged between the client and server is secure.
4. Server-side application: This component will be responsible for vote counting, data storage, and administration. It will provide a dashboard for administrators to manage the voting process and generate reports.
5. Database: This component will store all data related to the voting process, including voter information and voting results. The database will be secure and will ensure that only authorized users can access it.
6. Reporting and Analytics: This component will be responsible for generating reports on the voting process. It will provide insights into voter behavior, voting trends, and any issues with the voting process.

3.2 Decomposition Description

The decomposition of the subsystems in our architectural design for the CPL voting system can be seen in the following sequence diagram:


CPL
Candidate
Ballot
Receive Ballots
Read in Candidate information
Candidate stores party with the most votes
Loop
BallotCounter(), Stores information in Candidate objects
Loop
BreakTies()
nextInt()
randomInt()
Read in ballots
Election Officials
DisplayResults() to terminal
Audit file sent to
Media
Audit file sent to

The following diagram is an activity diagram for an IR election.


Coin flip to eliminate candidates(potentially more than 2)
Candidates with lowest votes do tie
Candidates with lowest votes don't tie
System removes lowest candidate
No candidate has over 50%
System reads ballots
System distributes new votes accordingly
A candidate has over 50%/Only one candidate
System seperates candidates and parties
IR Election has been set
Election completed
Bring voting results to voting system
Voting System takes in file
System reads line by line
System has found a winner
System displays winner to the terminal
System produces an audit file
System Exits Program
<<System>> Voting System
<<System>> Election Officials Device

3.3 Design Rationale

We discussed and considered which data structures would best represent each component of the system. Some alternatives we considered was having IR and CPL only being functions, instead of classes as in our final design. We also considered having the ballots be stored as integers in an array, or possibly a 2D array of integers, before we decided on having a ballot class and having each ballot be its own object. This allowed us to create a vector of ballot objects to store the voting data.

In decisions like these, we came to our conclusions based on several factors, such as what would be the most efficient to implement, most efficient to run, most secure, and easiest for future developers to understand and work with.

4. DATA DESIGN

4.1 Data Description

The data is entered into the program via a CSV file. The program scans through the CSV file to convert the votes into Ballot objects. They are all held in a vector of Ballot objects managed by the Voting System.

The data is passed between the Voting System, its vector of Ballot objects, and a Voting Type object, which is either CPL or IR, to calculate the results of the election.

4.2 Data Dictionary

Ballot: A class that holds each voter ballot. For IR elections, it also has a rank value, which represents if the ballot is on its first choice, second choice, etc. Its functions are:

+ Ballot(Votes: String)

+ getVotes(): int[]

+ increaseRank(): int

Candidate: A class that represents each candidate person in IR voting, or each party in CPL voting. It keeps track of how many votes it has. Its functions are:

+ Candidate(Name: String, Party: String)

+ GetName(): String

+ GetParty(): String

+ GetNumVotes(): int

+ SetNumVotes(): int

CPL: A class that handles calculating the votes for Closed Party Listing voting. Its functions are:

+ CPL(auditFileName: String, votingSystem: VotingSystem)

+ CalculateResults(Vector<Ballot>)

- WriteToAudit(Contents: String)

IR: A class that handles calculating the votes for Closed Party Listing voting. Its functions are:

+ IR(auditFileName: String, votingSystem: VotingSystem)

+ CalculateResults(Vector<Ballot>)

- WriteToAudit(Contents: String)

- ResolveTie(TiedCandidates: Candidate[])

- RemoveLowest(): Vector<Ballot>

Voting System: The overall system that handles the main steps in the process, and manages the movement of data throughout the system. Its functions are:

+ main(args: String[]): void

+ scanFile(fileName: string): void

+ getBallots(): Vector<Ballot>

+ getCandidates(): Vector<Candidate>

+ DisplayWinner(): void

5. COMPONENT DESIGN

In this section, we take a closer look at what each component does in a more systematic way. If

you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudocode. If you gave an OO description, summarize each object member function for all the objects listed in 3.2 in PDL or pseudocode. Describe any local data when necessary.

```
For ID = UC_001
    while True:
        filename = prompt_user_for_filename()
        try:
            file = open(filename)
            process_file(file)
            file.close()
            print("File processed successfully.")
            break
        except FileNotFoundError:
            print("Invalid filename. Please try again.")

    function prompt_user_for_filename():
```

```
    return input("Please enter a filename to scan: ")

function process_file(file):
    # code to process the file goes here
    pass
```

For ID = UC_002

```
import random

function select_winner(candidates):
    if random.random() < 0.5:
        winner = candidates[0]
    else:
        winner = candidates[-1]
    return winner
```

For ID = UC_003

```
function determine_election_winner(votes):
    max_votes = max(votes.values())
    if max_votes / sum(votes.values()) < 0.5:
        # no clear majority, use popularity to break the tie
        winner = determine_popular_winner(votes)
    else:
        # there is a clear majority
        candidates_with_max_votes = [candidate for candidate in votes if votes[candidate] ==
max_votes]
        if len(candidates_with_max_votes) == 1:
            winner = candidates_with_max_votes[0]
        else:
            # use UC_002 to resolve the tie
            winner = resolve_tie_with_UC_002(candidates_with_max_votes)
    return winner

function determine_popular_winner(votes):
    max_votes = max(votes.values())
    candidates_with_max_votes = [candidate for candidate in votes if votes[candidate] == max_votes]
    if len(candidates_with_max_votes) == 1:
        winner = candidates_with_max_votes[0]
    else:
        # use alphabetical order to break the tie
        winner = sorted(candidates_with_max_votes)[0]
    return winner

function resolve_tie_with_UC_002(candidates):
    # use some tie-breaking procedure to choose a winner
    # this could involve additional input from the user
    # or some other method
    pass
```

For ID = UC_004

```
fucntion parse_election_file(filename):
    with open(filename, 'r') as f:
        lines = f.readlines()

    # scan file to get the type of voting
    voting_type = get_voting_type(lines)
    if not voting_type:
        # prompt the user to input the type of voting
        voting_type = prompt_user_for_voting_type()

    # scan file to get the number of parties
```

```
num_parties = get_num_parties(lines)
if not num_parties:
    # prompt the user to input the number of parties
    num_parties = prompt_user_for_num_parties()

# scan file to get the list of party names
party_names = get_party_names(lines)
if not party_names:
    # prompt the user to input the list of party names
    party_names = prompt_user_for_party_names(num_parties)

# verify that the number of party names matches the expected number of parties
if len(party_names) != num_parties:
    raise ValueError("Number of party names does not match the expected number of parties.")

# scan file to get the number of seats
num_seats = get_num_seats(lines)
if not num_seats:
    # prompt the user to input the number of seats
    num_seats = prompt_user_for_num_seats()

# scan file to get the number of ballots
num_ballots = get_num_ballots(lines)
if not num_ballots:
    # prompt the user to input the number of ballots
    num_ballots = prompt_user_for_num_ballots()

# store the gathered information in variables/lists
election_data = {
    'voting_type': voting_type,
    'num_parties': num_parties,
    'party_names': party_names,
    'num_seats': num_seats,
    'num_ballots': num_ballots
}

return election_data
```

```
For ID = UC_005
    start
     prompt user to enter a filename
        if file doesn't exist:
            display error message and go back to start
        else:
            open file
            read in type of voting
            if type of voting is missing:
                prompt user to enter it
            read in number of parties
            if number of parties is missing:
                prompt user to enter it
            read in list of party names
            if list of party names is missing:
                prompt user to enter it
            check if number of party names matches expected number of parties
            if number of party names doesn't match:
                prompt user to enter it again and go back to start
            read in number of seats
            if number of seats is missing:
                prompt user to enter it
            read in number of ballots
            if number of ballots is missing:
                prompt user to enter it
            calculate election results based on gathered information
            output election results to console
            create audit file with correct title and save in current working directory
        if any errors or exceptions are encountered:
            display error message and prompt user to retry or terminate program
            go back to start
```

For ID = UC_006

```
START

IF file_processed AND audit_file_created THEN
User selects to send file to media
IF media_selected THEN
Send file to selected media
IF file_sent_successfully THEN
Notify User of successful transfer
ELSE
Notify User of transfer error
END IF
END IF
ELSE
Reprompt program and rerun
END IF

END
```

For ID = UC_007

```
START
```

```
INPUT ballots
WHILE number_of_candidates > 1 DO
count the number of first place votes for each candidate
eliminate the candidate with the fewest first place votes
FOR each ballot DO
if the ballot's top preference was eliminated
eliminate that preference and apply the next preference
END FOR
END WHILE
RETURN the candidate remaining

END
```

For ID = UC_008

```
START

PRINT “Which voting style would you like to use (IR/CPL)”
INPUT userChoice
IF userChoice == “IR” THEN
    CALL InstantRunoff()
END IF
ELSE IF userChoice == “CPL” THEN
    CALL ClosedPartyList()
END ELSE IF
ELSE
    PRINT “Invalid choice. Please enter either (IR/CPL)”
    RETURN TO START
END ELSE

END
```

For ID = UC_009

```
    START

    INPUT file FROM user
    IF file extension is not .txt THEN
    DISPLAY error message "Invalid file type."
    ELSE
    READ file
    PARSE information from file
    DISPLAY success message "File successfully read and parsed."
    ENDIF

  END
```

For ID = UC_010

```
START

INPUT filename
IF filename is valid AND filename has correct extension THEN
SET fileContent to read the content of the file
SET firstLine to extract the first line of fileContent
IF firstLine equals "IR" THEN
SET votingType to "Instant Runoff"
ELSE IF firstLine equals "CPL" THEN
SET votingType to "Closed Party List"
ELSE
THROW an error "Invalid voting type"
END IF
ELSE
THROW an error "Invalid filename or extension"
```

```
END IF

END

For ID = UC_011

START

FUNCTION create_IR_audit_file(input_file):
IF input_file is not valid:
        PRINT “No Candidates Found”
        RETURN
END IF
audit_file = initialize_audit_file()

WHILE length of candidates > 1:
        candidate_with_fewest_votes = get_candidate_with_fewest_votes(candidates)
        candidates = remove_candidate_from_ballot(candidates, candidate_with_fewest_votes)
        add_candidate_to_audit_file(audit_file, candidate_with_fewest_votes)
END WHILE

RETURN audit_file

END FUNCTION

END

For ID = UC_012

    START

    CALCULATE_QUOTA(total_votes, seats)
    ALLOCATE_SEATS_TO_PARTIES(parties, quota)
    ALLOCATE_REMAINING_SEATS(parties, seats)
    SELECT_WINNING_CANDIDATES(parties)
    RETURN elected_candidates
END
FUNCTION CALCULATE_QUOTA(total_votes, seats)
    quota = total_votes / seats
END FUNCTION
FUNCTION ALLOCATE_SEATS_TO_PARTIES(parties, quota)
    FOR EACH party IN parties
        party.seats = party.votes / quota
    END FOR
END FUNCTION
FUNCTION ALLOCATE_REMAINING_SEATS(parties, seats)
    remaining_seats = seats - SUM(party.seats)
    WHILE remaining_seats > 0
        party_with_largest_remainder = FIND_PARTY_WITH_LARGEST_REMAINDER(parties)
        party_with_largest_remainder.seats = party_with_largest_remainder.seats + 1
        remaining_seats = remaining_seats - 1
    END WHILE
END FUNCTION
FUNCTION SELECT_WINNING_CANDIDATES(parties)
    FOR EACH party IN parties
        party.elected_candidates = SELECT_CANDIDATES_FROM_LIST(party.candidate_list, party.seats)
    END FOR
END FUNCTION
FUNCTION FIND_PARTY_WITH_LARGEST_REMAINDER(parties)
    party_with_largest_remainder = parties[0]
    FOR i = 1 TO length(parties)-1
        IF parties[i].seats - FLOOR(parties[i].seats) > party_with_largest_remainder.seats - FLOOR(party_with_largest_remainder.seats)
            party_with_largest_remainder = parties[i]
```

```
END IF
END FOR
RETURN party_with_largest_remainder
END FUNCTION
EXCEPTIONS:
IF CALCULATE_QUOTA() fails, throw "failure".
IF the file name is incorrect, throw "name incorrect".

END
```

For ID = UC_013

```
if file_processed_successfully:
    display_results()
else:
    handle_error()

def display_results():
    winner = determine_winner()
    display_winner(winner)
    if user_requests_additional_info():
        additional_info = get_additional_info(winner)
        display_additional_info(additional_info)

def handle_error():
    print("Error processing file.")

def determine_winner():
    # code to determine the winner goes here
    return winner

def display_winner(winner):
    print("Winner: ", winner)

def user_requests_additional_info():
    # code to check if the user requests additional info goes here
    return additional_info_requested

def get_additional_info(winner):
    # code to get additional info about the winner goes here
    return additional_info

def display_additional_info(additional_info):
    print("Additional Information: ", additional_info)
```

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

From a user perspective, the system does everything and the user doesn’t see much. The user will input a csv file in the command line and get back an audit file with a display in the terminal for the winner. The user doesn’t have much else functionality for this system. Since the user wants to know who won the election, this system will properly count the votes and return to the user who won the election. Along with that, the user will have an audit file that will track the votes and the elimination process. The user will also have a display in the terminal that will allow the user to instantly know who won the election.

Describe the functionality of the system from the user’s perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

6.2 Screen Images

```
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
PS C:\Users>
```

6.3 Screen Objects and Actions

The only object that is on the screen of the user is the terminal. The action associated with the terminal is that the user can input a csv that the system will perform the counting on. The terminal will also return the winner from the voting process to the user.

7. REQUIREMENTS MATRIX

Use Case doc

Functional Requirement	System Component
UC_001	Scanner object predefined in Java. Prompt the user for the filename and open the file for reading
UC_002	Resolve tie based on resolveTie() function call
UC_003	One step in the process of calculating the results in both elections
UC_004	Prompt user for more information with Scanner object
UC_005	Function in Voting System that creates an audit file
UC_006	Voting system has a function that enables sharing with media personnel
UC_007	IR class that has multiple functions that aid in instant runoff results

UC_008	Voting system class can create either an IR object or a CPL object, depending on which election type is called
UC_009	Scanner object will read and parse the file
UC_010	Scanner object will read in and voting system will decide which type of voting
UC_011	Within IR voting, methods created in a way to create correct audit file
UC_012	CPL class contains functions to calculate the results of a CPL election
UC_013	Voting System class will have a method that displays results to terminal