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.

Team 4

Voting System

Software Design Document

Name (s): Annabelle Coler (coler018), Anwesha Samaddar (samad037), Hilton Nyguen (nguy4798), Zoe Sepersky (seper011)

Course Section: 002

Date: 03/03/2025

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document (SDD) describes the architecture, data design, component design, and user interface of a voting system. This document is meant to serve as a guide to the software development team as to how to implement the system.

1.2 Scope

The voting system will count ballots and run either Single Transferable Voting (STV) or Plurality elections. Users will input ballots in a comma-separated values (CSV) file, as well as the number of seats up for election and the type of election. Ballots will then be counted using the algorithm selected by the user. Once the election is complete, the results will be output in a text file serving as an audit report and displayed on the screen. This system will automate two election styles, resulting in more efficient ballot counting and faster election results.

1.3 Overview

This document follows the conventions of a simplified template of the IEEE Recommended Practice for Software Design Descriptions. This SDD begins with an introduction of the document, followed by an overview of the voting system. The project's architecture, data design, component design, and human interface design are then described in technical detail. A requirements matrix is also provided to demonstrate all requirements outlined in the Software Requirements Specification (SRS) are met.

1.4 Reference Material

This document is based on the following template:
https://canvas.umn.edu/courses/483118/assignments/4495497#submit.
The voting system is based on the following project description:
https://canvas.umn.edu/courses/483118/files/50177539?module_item_id=13993087.
Use cases mentioned in part five are described in more detail in the following software requirements specification:
https://github.umn.edu/umn-csci-5801-02-S25/repo-Team4/blob/main/SRS/SRS_Team4.pdf

1.5 Definitions and Acronyms

- Ballot: A representation of voter preferences for Plurality and STV elections. “1” denotes the top choice in candidate selection.
- Candidate: An individual applying for a position through election.
- Election official: An individual responsible for entering ballots into the voting system and running elections.
- Plurality Voting: Each voter is allowed to vote for only one candidate, and the candidate who polls the most votes is elected.
- STV: Single transferable voting. Voters rank candidates according to their preferences. Their vote can be transferred according to alternate preferences if their preferred candidate is eliminated
- Droop Quota: The minimum number of votes a party or candidate needs to earn to win at least one seat in a district.

2. System Overview

The voting system will be used by election officials and testers to run elections. It will take in as user input a number of seats up for election, a CSV file containing ballots, and a type of election algorithm (i.e. STV or Plurality). After counting ballots and determining a winner, the system will provide an election audit report as a text file. It will also display the election results on the screen. Testers will be able to toggle on or off the shuffle functionality of the STV algorithm so that algorithm can be tested. The gathering and validating of ballots is outside the scope of the system, as is election security.

3. System Architecture

3.1 Architectural Design

The voting system program has a modular structure to maintain its clarity and extendibility. The design uses object-oriented programming (OOP), with key classes containing different parts of the election system. The system can be broken down into subsystems that are responsible for running elections, ballots, user interface interactions, and candidates.

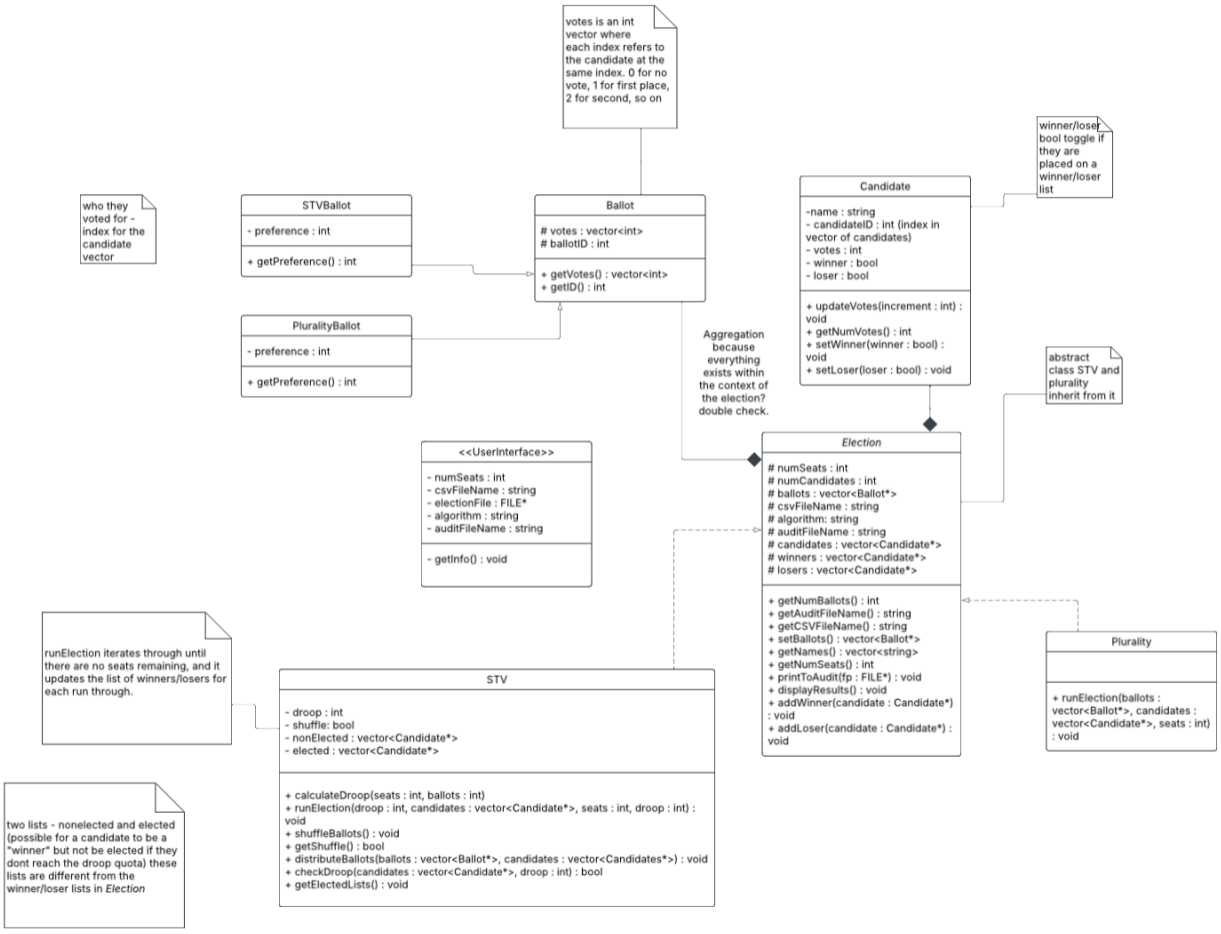
The systems and responsibilities of each system are as follows:

Name	Classes	Responsibilities
Ballot Subsystem	Ballot, STVBallot, PluralityBallot	Represents individual ballots in the election and stores voter preferences
Candidate Subsystem	Candidate	Represents individual candidates in the election
Election Management Subsystem	Election, STV, Plurality	Manages election execution, supports different elections algorithms (STV or Plurality), and handles the output/display of election results
User Interface Subsystem	UserInterface	Interface for a user to interact with the system and to display election results

The UserInterface class collects election information from the user. This information is then passed into the Election class, which creates Ballot and Candidate objects. If the user specified the election as STV, then STVBallots are made, and the STV class is used to run the election. STVBallot objects inherit from Ballot, and the STV class inherits from Election. If the user specified plurality, then PluralityBallots are created, and the Plurality class is used to run the election. PluralityBallots inherit from Ballot, and the Plurality class inherits from Election. Once the election is complete, the election algorithm classes will print the results to a text file and display them on the screen.

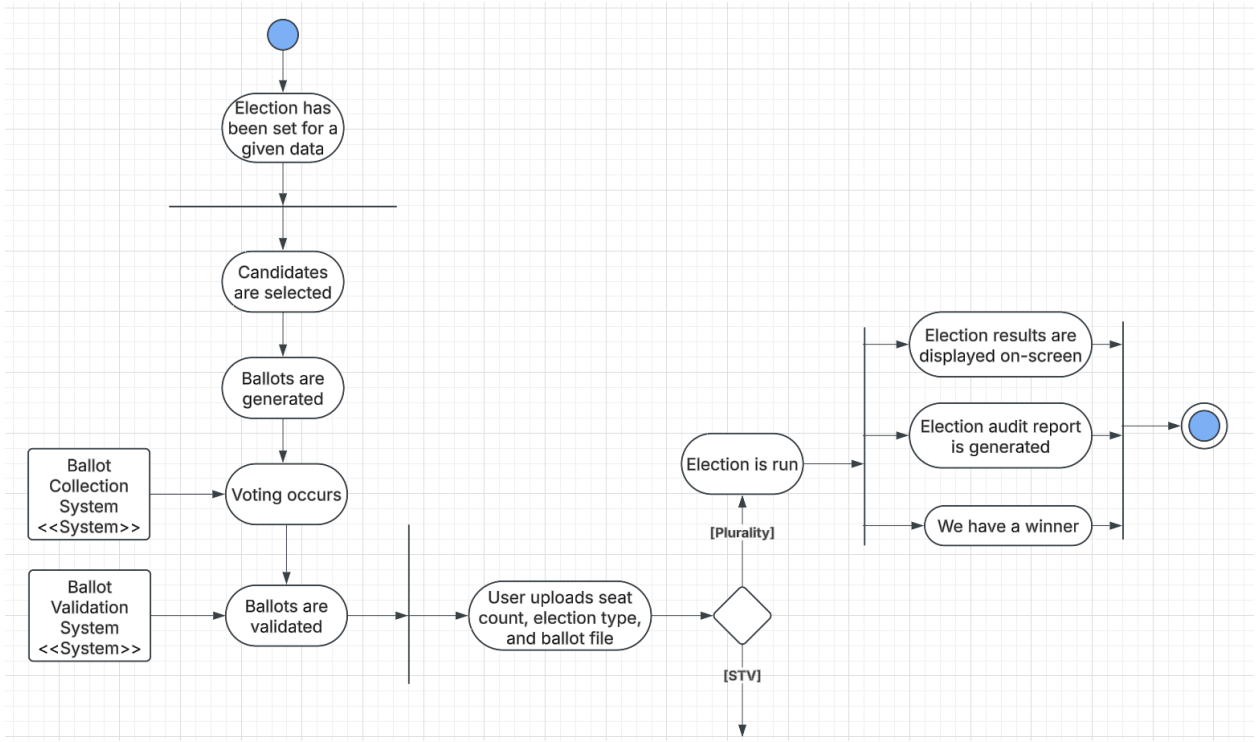
3.2 Decomposition Description

UML Class Diagram:



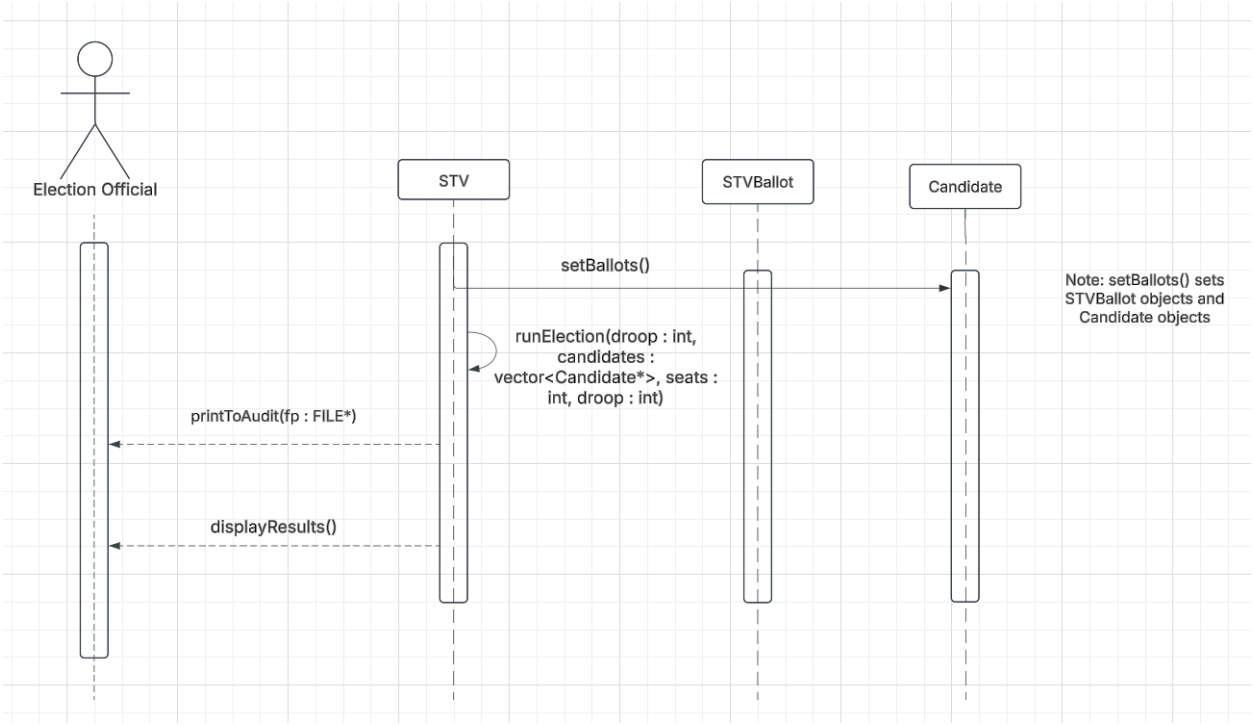
The STV and Plurality classes inherit from the abstract class Election. The Election class stores variables and methods pertinent to all election algorithms, while the STV and Plurality classes are specific to the algorithms for which they are named. The same is true for STVBallot and PluralityBallot, which inherit from the class Ballot. The Candidate and UserInterface classes will be used for all election types.

Activity Diagram for Plurality:



Once the election is set, candidates are selected and ballots are generated. Voting occurs via a Ballot Collection System, and ballots are validated via a Ballot Validation System. The user is then able to upload pertinent information to the voting system and run the election. Once a winner has been found, results are displayed on-screen and in an audit report.

Sequence Diagram for STV:



The sequence diagram for the STV election process begins with the assumption that the ballot file has been read and user inputs (such as candidates, voting algorithm, and number of seats) have been correctly received. The STV object, which inherits from the Election class, sets up STVBallot and Candidate objects using the setBallots() method. It then runs the election through the runElection() method, processing ballots based on the droop quota, candidate list, and number of seats. After the election concludes, the printToAudit() method logs the election details to an audit report, and the displayResults() method shows the final results to the user. This sequence clearly outlines the flow from processing ballots to displaying the results, capturing the essential steps of the STV election process. **There is no alternative process because we assume the election is conducted accurately and the STV algorithm works as intended.**

3.3 Design Rationale

One of the biggest reasons for the modular structure of the system is to handle two election algorithms, STV and plurality. These two algorithms have different rules for counting votes and determining the winners, so they cannot both be represented by a single election class. Instead, they both inherit from an abstract class Election. This also makes it easier to add new election types to the system in the future. The Election Management System handles the election process. It applies the correct algorithm, while the Ballot Subsystem makes sure that the voters' preferences are stored correctly based on the election type.

This design choice also allows code to be reused. The STV and Plurality classes both inherit from the abstract Election class, and the STVBallot and PlurarilyBallot classes inherit from the Ballot class. Key functions do not need to be rewritten for each election/ballot type.

Lastly, the User Interface Subsystem ensures that the way the users interact with the system is independent of how elections are managed. This allows the possibility to improve or change the user interface without changing the core election logic. Additionally, the display and output functions are separate so that the result can easily be formatted and presented in different ways.
We did not consider using a monolithic (one large program) design for this system. This is because it would make making changes harder, since modifying one part of a class could affect the whole program. Adding a new election type could potentially harm multiple sections of the program, increasing the risk of breaking other functions.

4. DATA DESIGN

4.1 Data Description

The information domain of the system will be primarily tracked through three classes: Ballot, Candidate, and Election. Ballots will be tracked through a Ballot class, and those ballots will hold an integer vector corresponding to the votes for a candidate. Each index of the vector will correspond to a specific candidate. Within the Election class, there will be a vector of Candidate pointers and a vector of Ballot pointers, each representing the candidates and the ballots of the election. Using vectors allows for dynamic handling of votes and efficient updating and retrieval of candidate information. Election results are maintained in separate vectors of winners and losers within the Election class. While most data is stored in memory for fast access, the system ensures persistence by writing election results to an external audit file using the printToAudit method.There are no external databases or data storage systems. All data is stored locally.

4.2 Data Dictionary

A - E

Ballot

- **Type:** Class
- **Description:** Represents an individual vote, storing a vector of ranked candidate preferences and a unique ballot ID.
- **Attributes:**
  - votes - vector<int>
  - ballotID - int
- **Methods:**
  - int getID() - returns the ID of a ballot

- vector<int> getVotes() - returns the votes.

Candidate

- **Type:** Class
- **Description:** Represents a candidate in the election, storing details such as name, ID, vote count, and winner/loser status.
- **Attributes:**
  - name - string
  - candidateID - int
  - votes - int
  - winner - bool
  - loser - bool
- **Methods:**
  - void updateVotes(int) - update vote counts of a candidate
  - int getNumVotes() - returns the amount of votes a candidate has
  - void setWinner(bool) - sets the winner boolean if the candidate is a winner
  - void setLoser(bool) - sets the loser boolean if the candidate is a loser.

Election

- **Type:** Abstract Class
- **Description:** Serves as a base class for the two different election types, managing seats, candidates, ballots, winners, and losers.
- **Attributes:**
  - numSeats - int
  - numCandidates - int
  - ballots - vector<Ballot*>
  - csvFileName - string
  - algorithm - string
  - candidates - vector<Candidate*
  - winners - vector<Candidate*>
  - losers - vector<Candidate*>
- **Methods:**
  - int getNumSeats() - returns number of seats
  - string getAuditFileName() - returns the name of the audit file
  - string getCSVFileName() - returns the name of the CSV file
  - vector<Ballot*> setBallots() - sets the ballots of the election and returns a vector of Ballot pointers
  - int getNumBallots() - returns the number of ballots
  - void printToAudit(FILE* fp) - prints election information to the audit file
  - vector<string> getNames() - returns the names of the candidates

- void displayResults() - display intermediate results to the terminal for the user
- void addWinner(Candidate* candidate) - adds candidate to winner list
- void addLoser(Candidate* candidate) - adds candidate to loser list

---

P - U

Plurality

- **Type:** Class (inherits from Election)
- **Description:** Implements the Plurality voting system, where the candidate with the highest number of votes wins.
- **Attributes:**
  - None
- **Methods:**
  - void runElection(vector<Ballot*> ballots, vector<Candidate* candidates, int seats) - runs the plurality algorithm on an election.

PluralityBallot

- **Type:** Class (inherits from Ballot)
- **Description:** Represents a ballot used in a Plurality election, storing a single candidate preference.
- **Attributes:**
  - preference - int
- **Methods:**
  - int getPreference() - returns the preference of the vote

STV (Single Transferable Vote)

- **Type:** Class (inherits from Election)
- **Description:** Implements the STV voting system, handling ranked-choice voting, vote redistribution, and Droop quota calculations.
- **Attributes:**
  - droop - int
  - shuffle - bool
  - elected - vector<Candidate*>
  - nonElected - vector<Candidate*>
- **Methods:**
  - int calculateDroop(int seats, int ballots) - calculates the droop quota based on number of votes and number of seats

- void runElection(vector<Ballot*> ballots, vector<Candidate*> candidates, int seats, int droop) - runs the election on the STV algorithm.
- bool getShuffle() - returns if the shuffle has been turned off or on for this election
- void shuffleBallots() - shuffles the ballots
- void distributeBallots(vector<Ballot*> ballots, vector<Candidate*> candidates) - distributes the leftover ballots after a round of STV to the remaining candidates
- bool checkDroop(vector<Candidate*>, int droop) - checking if any candidate's vote count has met or surpassed the Droop quota, meaning they are a winner of the election
- void getElectedLists() - print elected/non-elected lists to the user

STVBallot

- **Type:** Class (inherits from Ballot)
- **Description:** Represents a ballot used in an STV election, storing ranked preferences.
- **Attributes:**
  - preference - int
- **Methods:**
  - int getPreference() - returns the preference of the vote

UserInterface

- **Type:** Class
- **Description:** Takes in and stores initial user input regarding the ballot file, number of seats, election type, and audit file name.
- **Attributes:**
  - numSeats - int
  - csvFileName - string
  - electionFile - FILE*
  - algorithm - string
  - auditFileName - string
- **Methods:**
  - void getInfo() - Prints the information of the election to the console

5. COMPONENT DESIGN

1. UserInterface Class:
Created to take input from the user.

Public Functions:
//UC_001, UC_002, UC_003, UC_006, UC_008, Get needed information from user
Function getInfo() -> void

- Print “Enter CSV file name:”
- csvFileName = input csvFileName
- Print “Upload CSV file:”
- electionFile = input electionFile
- Print “Enter number of seats:”
- numSeats = input numSeats
- Print “Select algorithm (STV or Plurality):”
- algorithm = input algorithm
- Print “Disable ballot shuffle? (true/false)”
- shuffle_stv = input shuffle_stv
- Print “Enter audit file name:”
- auditName = input auditName

2. Election Class
The election class manages election parameters, loads ballots, stores candidate names and ids, generates the audit file report, and displays the election results to the screen.

Public Functions:
// Get CSV file name
Function getCSVFileName() -> string

- Return csvFileName

//Get audit file name
Function getAuditFileName() -> string

- Return auditFileName

//Return the number of seats
Function getNumSeats() -> Integer

- Return the number of seats up for election

//Get number of Ballots
Function getNumBallots() -> Integer

- Return the number of ballots in the ballots list

//Get voting algorithm
Function getAlgorithm() -> String

- Return algorithm selection (i.e. the string “STV” or “Plurality”)

//Get Candidate names

Function getNames() -> vector<Candidate*>

- Create an empty vector called names
- For each candidate in the candidates vector:
    - Add the candidate's name to the names vector

Return names

//Load ballots from CSV
Function setBallots() -> vector<Ballot*>

- Open fileName
- Read the first line of the file to get Candidate names
    - set candidates vector
- Read For each subsequent line in the file:
- If the algorithm is "STV":
    - Create a new STVBallot object using the line data.
    - Else:
    - Create a new PluralityBallot object using the line data.
- Add the new ballot to the ballots list.
- Validate the ballots to ensure they follow the ballot rules for STV or Plurality.

//Get number of ballots in ballots vector
Function getNumBallots() -> int

- Return length of ballots vector

//UC_009: Generate audit file report
Function printToAudit(File*) -> void

- Open the file specified by auditFileName
- Write the type of election selected by the user
- Write the number of Seats filled
- Write the number of candidates
- Write the list of winners using winners vector
- Write the list of losers using losers vector
- Close the file

//UC_007: Display election results to screen
Function displayResults() -> void

- Print type of election, number of seats, number of ballots, number of candidates, list of winners, and list of losers to the screen

// Add Candidate to Winners List
Function addWinner(candidate: Candidate*) -> void

- Add the candidate to the winners list.
- Mark the candidate as a winner using candidate->setWinner(true).
- Record the order of election
- Remove those ballots permanently
- Surplus ballots for the winning candidate will go to the next candidate.

// Add Candidate to Losers List

Function: addLoser(candidate: Candidate*) -> void

- Add the candidate to the losers list.
- Mark the candidate as a loser using candidate->setLoser(true).
- Redistribute their votes to the next preferred candidates.

3. STV Class:

Implements the STV algorithm.

Public Functions:

//Calculate Droop Quota

Function calculateDroop(seats: int, ballots: int) -> int

- Calculate the Droop Quota using the formula:

  droopQuota = $\left\lfloor \left(\frac{numberOfBallots}{numberOfSeats+1} \right) \right\rfloor + 1$

- Return the calculated droopQuota

//Check if droop quota has been reached

Function checkDroop(candidates : vector<Candidate*>, droop : int) -> bool

- For each candidate, if candidate.getNumVotes() >= droop:
  - Return true.

  Else:
  - Return false.

//Get shuffle decision

Function getShuffle() -> bool

- Return true or false for whether shuffle is toggled on or off

//Shuffle Ballots

Function shuffleBallots() -> void

- If shuffle is enabled, shuffle the ballots randomly.
- If shuffle is disabled, leave the ballots in their original order.

// Distribute Ballots

Function distributeBallots(ballots: vector<Ballot*>, candidates: vector<Candidate*>) -> void

- For each ballot in ballots:
  - Get the next preference using: preference = ballot->getPreference().
  - Assign the ballot to the corresponding candidate using candidate[preference] -> updateVotes().

// Run STV election

Function runElection(droop : int, candidates : vector<Candidate*>, seats : int) -> winners : vector<Candidate*>, losers : vector<Candidate*>

- Call shuffleBallots()
- Calculate the Droop Quota using calculateDroop(seats, ballots.size()).
- Initialize empty lists for **winners and losers.**

- While the number of winners <= seats:
    - Distribute Ballots using distributeBallots()
    - Create winners list:
      For each candidate:
        - If checkDroop(candidate, droop) returns true:
            - Call addCandidateToWinnersList(candidate).

    - Create losers list:
        - If no candidate meets the droop:
            - Find the candidate with the lowest votes.
            - If there is a tie, eliminate the candidate who was last to receive their first ballot.
            - Call addCandidateToLosers(candidate).

- Return winners and losers lists.

//Returns nonElected and Elected vectors of candidates
Function getElectedLists() -> void

- Return nonElected and Elected vectors of candidates

4. Plurality Class:
Implements the Plurality election algorithm.

Public Functions:
//Runs election
Function runElection(ballots : vector<Ballot*>, candidates : vector<Candidate*>, seats : int) -> winners : vector<Candidate*>, losers : vector<Candidate*>

- Loop through ballots vector and maintain a count of votes for each candidate
- Compare the count for each candidate
- If no tie, the candidate with the greatest count wins
- Else, randomly select winner for a given seat
- Return winners and losers lists

5. Ballot Class:
Maintains information about an individual ballot.

Public Functions:
//Gets the votes on a ballot
Function getVotes() -> vector<int>

- Return a list of the votes

//Gets the unique ID of a ballot
Function getID() -> int

- Return ballot ID

6. PluralityBallot Class:
Maintains information about an individual plurality ballot.

Public Functions:

//Gets the index of a vote on a ballot
Function getPreference() -> int

- Returns the index of the vote on the ballot

7. STVBallot Class:

Maintains information about an individual STV ballot.

Public Functions:

//Gets the index of the highest-ranked vote on a ballot
Function getPreference() -> int

- Returns the index of the highest-ranked vote on the ballot

8: Candidate Class:

Maintains information about an individual candidate.

Public Functions:

//Gets the count of votes for a candidate
Function getNumVotes() -> int

- Returns the number of votes for a candidate

//Increments the number of votes assigned to a candidate
Function updateVotes(increment : int) -> void

- Increments the number of votes a candidate has

//Used to signal a candidate has been placed on a winner list
Function setWinner(winner : bool) -> void

- Toggle to true if a candidate has been placed on a winner list

//Used to signal a candidate has been placed on a loser list
Function setLoser(loser : bool) -> void

- Toggle to true if a candidate has been placed on a loser list

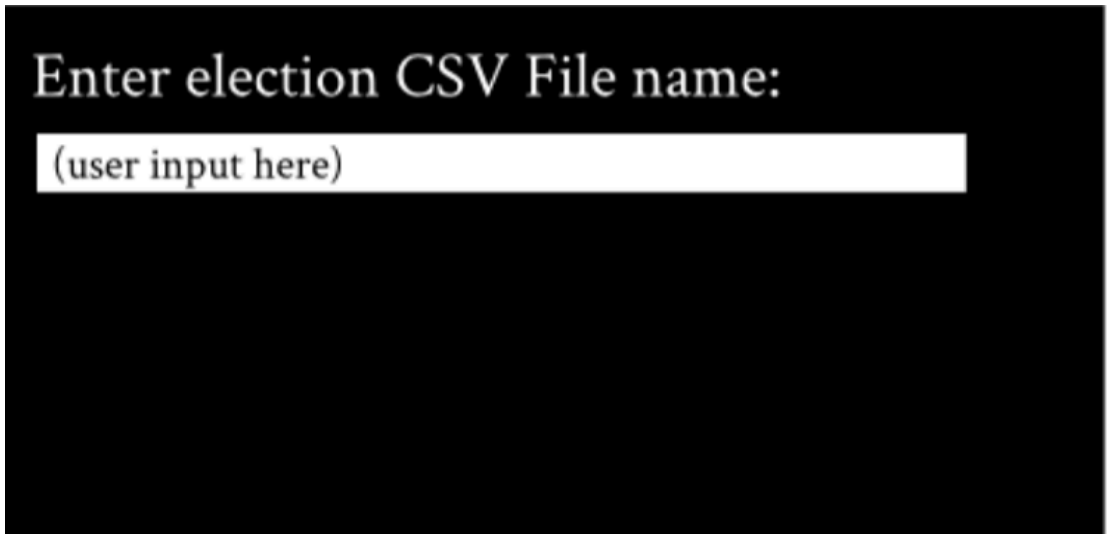
6. HUMAN INTERFACE DESIGN

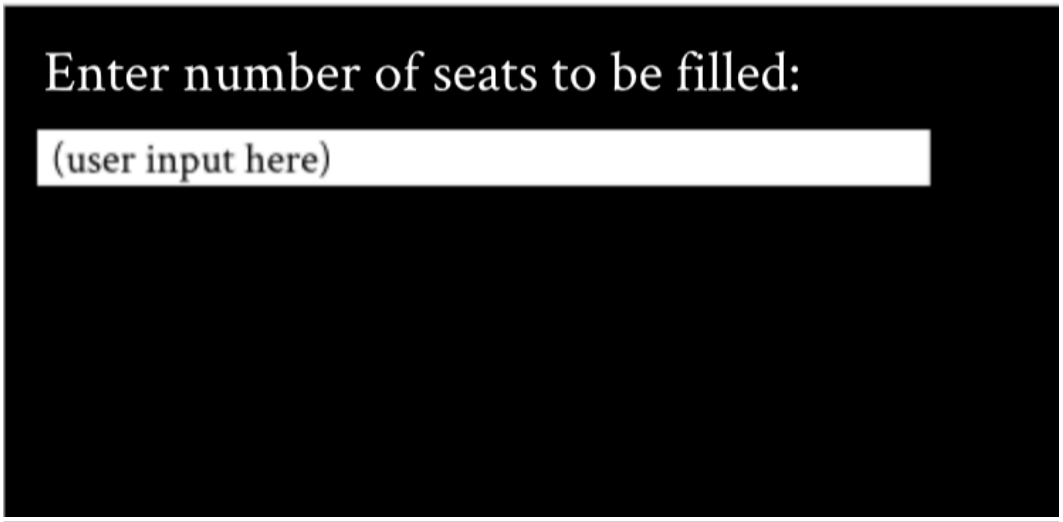
6.1 Overview of User Interface

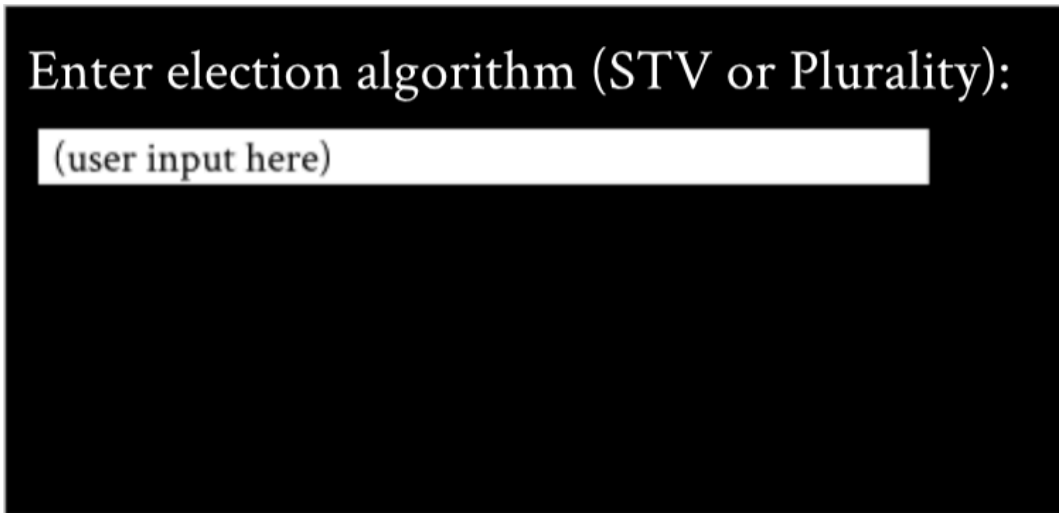
The user shall input information through the graphical user interface (GUI). The user shall be prompted to enter the comma-separated value file name, number of seats to be filled, and the type of election algorithm that they wish to use. If the user inputs invalid information, they shall be alerted that their input is invalid and prompted if they would like to try again. They can attempt to input information up to two times, after which the system shall notify the user they have used up their attempts and gracefully exit.

6.2 Screen Images

The following are images of the interface from the user's perspective.







After inputting the election information and running the election, election results will be printed onto an audit file, which the user can then view. Election results will also be displayed on the screen.

6.3 Screen Objects and Actions

The UserInterface object accepts and stores user input. The Election object displays the election results to the screen.

7. Requirements Matrix

The following table shows the mapping of functional requirements from the SRS document to the system components and data structures in the Voting System design. The unique use case ID identifies each requirement in the SRS.

Use Case ID	Name	Description	System Component	Data Structures
UC_001	Input CSV file name	The user is prompted to input the file name that contains the ballots. The name should refer to a CSV file.	UserInterface, Election	1. ballots : vector<Ballot*> 2. candidates : vector<Candidate*> 3. numCandidates: int
UC_002	Input number of seats	The user is prompted to input the number of seats that are to be filled.	Election, UserInterface	1. numSeats : int
UC_003	User selects voting algorithm	The user is prompted to enter the method of election, either STV or Plurality.	Election, UserInterface	1. algorithm: string
UC_004	Run Single Transferable Vote (STV) algorithm	The user runs a STV algorithm for a given election.	Election, STV, Ballot, STVBallot, Candidate	1. numSeats : int 2. ballots : vector<Ballot*> 3. candidates : vector<Candidate*> (loaded from CSV) 4. winners : vector<Candidate*> 5. losers : vector<Candidate*> 6. droop: int

UC_005	Run Plurality voting algorithm	The user runs the Plurality voting algorithm for a given election.	Election, Plurality, Ballot, PluralityBallot, Candidate,	1. numSeats : int 2. ballots : vector<Ballot*> 3. candidates : vector<Candidate*> (loaded from CSV)
UC_006	Toggle shuffle option for STV	Allows the user to turn off ballot shuffle for testing purposes of the STV election algorithm.	UserInterface, Election, STV	1. shuffle: bool
UC_007	Display election result	After an election has been run, the system displays the details of the election results on the screen. This will include the type of election (i.e., STV/plurality), the number of ballots, the number of seats, the number of candidates, the list of winners and losers, the percentage of votes obtained if plurality, and the order of winning and losing for STV.	UserInterface, Election	1. numSeats : int 2. algorithm : string 3. ballots : vector<Ballot*> 4. vector<Candidate*> 5. winners : vector<Candidate*> 6. losers : vector<Candidate*>
UC_008	Input audit file name	User inputs the name of the election audit file.	UserInterface, Election	1. auditFileName : string
UC_009	Generate audit file report	Show the ballots were assigned to a candidate as the election progressed. Reports include all election data, such as type, seats, candidates, winners, and losers.	Election	1. numSeats : int 2. algorithm : string 3. ballots : vector<Ballot*> 4. vector<Candidate*> 5. winners : vector<Candidate*> 6. losers : vector<Candidate*> 7. auditFileName : string

UC_010	Run test file	The user has decided to run a test file to test the election system.	UserInterface, Election, see UC_004 and UC_005	See UC_004 and UC_005
UC_011	User interface	There is a user interface where the user can input file, seat, and election choice information.	UserInterface	1. numSeats : int 2. csvFileName : string 3. electionFile : FILE* 4. algorithm : string 5. auditFileName : string