Team 3

Voting System

Software Design Document

Names:
Hailin Archer - deak0007
Bryan Baker - bake1358
Colin Kluegel - klue0037
Josh Spitzer-Resnick - spitz123

Class: CSCI 5801
Semester: Spring 2020

Date: (03/20/2020)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This Software Design Document (SDD) provides the design details of the Voting System.

The expected audience are our programmers, developers, and testers.

1.2 Scope

This document contains a complete description of the design of the Voting System.

The goal of the Voting System is to provide users with election results of an STV (single transferable vote) algorithm and a plurality algorithm. Users will need to provide the system with the number of seats, select the type election algorithm, and identify the csv (comma separated values) files containing ballot information.

The basic architecture is a stand alone program consisting of following major objects: UserInterface, Election, ElectionRecord, ResultDisplay, Candidate, Ballot, and Logger.

1.3 Overview

The remaining chapters and their contents are listed below.

Section 2 is the system overview.

Section 3 is the Architectural Design that specifies the objects that collaborate to perform all the functions included in the system. Each of these objects has an Abstract Description concerning the services that it provides to the rest of the system.

Section 4 lists the Data Structure Design.

Section 5 describes Components.

Section 6 discusses the User Interface Design.

Section 7 shows the relationship between the VS's components and SRS requirements.

1.4 Reference Material

"Project 1 - Waterfall Methodology Software Design Document (SDD) for Voting System", Watters, Feb 2020: https://canvas.umn.edu/courses/158173/assignments/1022812?module_item_id=3613100

"Software Design Document (SDD) Template", Watters, Feb 2020: https://canvas.umn.edu/courses/158173/assignments/1022812?module_item_id=3613100

"Software Design Description - Web Accessible Alumni Database", Reaves, Dec 8, 2003: https://www.slideshare.net/peny_mg/sdd-software-des-sample

"Software Requirements Specification for Voting System", Archer, Baker, Kluegel, and Spitzer-Resnick, Feb 21, 2020: https://github.umn.edu/umn-csci-5801-002-s20/repo-Team3/blob/master/SRS/SRS_Team3.pdf

"UseCases_Team3", Archer, Baker, Kluegel, and Spitzer-Resnick, Feb 21, 2020: https://github.umn.edu/umn-csci-5801-002-s20/repo-Team3/blob/master/SRS/UseCases_Team3.pdf

1.5 Definitions and Acronyms

Term	Definition
CSV	Comma Separated Values
OOP	Object Oriented Programming
SDD	Software Design Document
SRS	Software Requirements Specification
STV	Single Transferable Vote
VS	Voting System

2. SYSTEM OVERVIEW

The software system being developed is a voting system to be used in local elections. The system will be designed to automate the counting of ballots to simplify the running of elections. The main feature of the software will be to run two types of elections, a plurality voting election and a single transferable voting (STV) election.

In addition to its primary purpose of running an election, the software will need to provide some additional features. The software needs to display detailed information about the election results, that is, it should display the number of ballots, the number of seats, the number of candidates and the winner(s) of the election. The software will also need to create a detailed report that will act as an audit for the election. The report will be saved as a text file and show details about how ballots were assigned to candidates as the election progressed.

The software will also require a diagnostic mode. The diagnostic mode will be entered using a command line option. The diagnostic mode is required to support an option to disable ballot shuffling so the system can be calibrated. The diagnostic mode will also have options for developers to use to debug the software.

To aid the user of the Voting System, a help window will be provided that will give the user information about how to run the program.

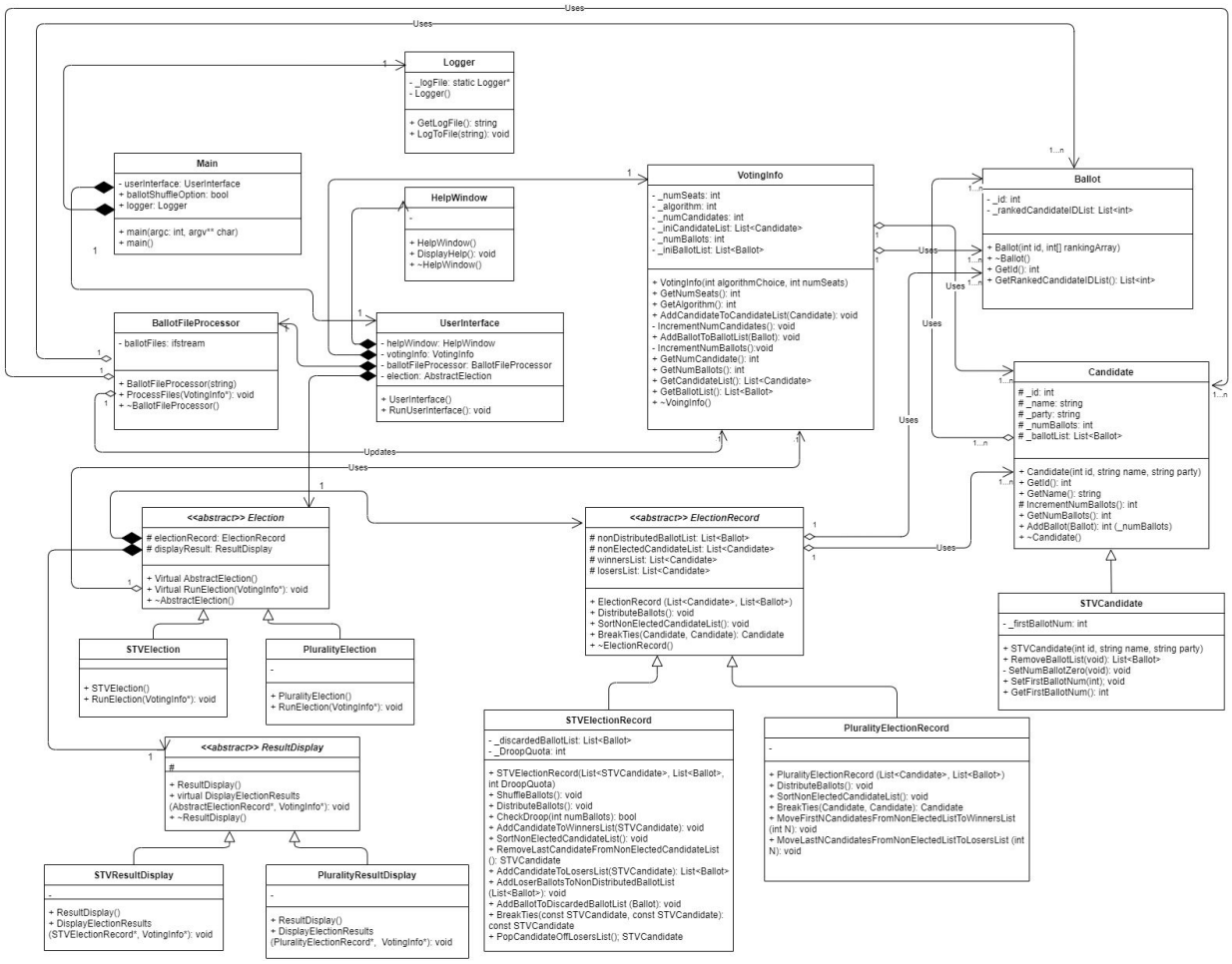
The above information leads to a list of the following subsystems within the Voting System:

- Ballot handling system
- Plurality voting system
- Single transferable voting (STV) system
- Audit reporting system
- Diagnostic system
- Help / display system

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The class diagram shows the overall design of the Voting System:



Main
Name: Main
Type: Class
Description: This is the class the operating system calls upon program execution. Main class will define a global variable ballotShuffleOption and initialize it to true or false based on the command line argument in the main constructor. Main class will also instantiate the Logger class. Then Main class will instantiate the UserInterface class and call its RunUserInterface method.
Data members:

- _userInterface:
  Type: UserInterface class

Access: Private
- ballotShuffleOption:
  Type: boolean
  Access: public
- logger:
  Type: Logger class
  Access: public

Constructors:
- main(argc int, argv** char): int
  Arguments: argc - argument count, argv - arguments strings
  Returns: int (0 - no error, others - error)
  Access: public
- main(): int
  Arguments: none
  Returns: int (0 - no error, others - error)
  Access: public

Member Functions: none
Flow of Events:
1. Program starts running
2. Define public static variable ballotShuffleOption and initialize it to False
3. Check command line arguments. If there is the command line argument '-t', set ballotShuffleOption to True
4. Instantiate logger
5. Instantiate userInterface
6. Call userInterface.RunUserInterface
7. Return 0

Logger
Name: Logger
Type: Class
Description: Logger class is instantiated in main and accessible by all objects of the program.
Data Members:
- _logFile: static Logger*
  Type: static Logger*
  Access: private

Constructor:
- Logger(): void
  Arguments: none
  Returns: none
  Access: private

Member Functions:
- GetLogFile(): string
  Arguments: none

Returns: string
Access: public
Description: Return value of data member _logFile

- LogToFile(string): void
  Arguments: string
  Returns: none
  Access: public
  Description: Write input string in _logFile

UserInterface

Name: UserInterface
Type: Class
Description: This form will launch automatically after the program starts running. The form will have a combination of list boxes and blank fields to be completed. Some fields will be marked as required. There will also be a help option. When a user chooses the help option he/she will be presented with information on how to run the election. When the form is completed the user can click on the run election button.
Data Members:

- helpWindow: HelpWindow
  Type: HelpWindow Class
  Access: Private
- votingInfo: VotingInfo
  Type: VotingInfo Class
  Access: Private
- ballotFileProcessor: BallotFileProcessor
  Type: BallotFileProcessor Class
  Access: Private
- election: AbstractElection
  Type: AbstractElection Class
  Access: Private

Constructors:

- UserInterface()
  Arguments: none
  Returns: none
  Access: public
  Description: Instantiate UserInterface Class

Member Functions:

- RunUserInterface(): void
  Arguments: none
  Returns: none
  Access: public
  Description: Launch the user interface

Flow of Events:

1. Declare data members
2. User is presented with the form
3. If user chooses help option
    a. Run helpWindown.DisplayHelp();
4. User fills in boxes and selects from list boxes
    a. Check validity of input values
    b. Instantiate VotingInfo object with values from the form
5. User clicks run election button
    a. Check if VotingInfo is properly instantiated
6. If the election algorithm is STV, instantiate election to STVElection Concrete Class. If the election algorithm is Plurality, instantiate election to PluralityElection Concrete Class
7. Call ballotFileProcessor.ProcessFiles(votingInfo)
8. Call election.RunElection()

HelpWindow
Name: HelpWindow
Type: Class
Description: This window launches when the user chooses the help option from the user interface. This window displays strings (information on how to use VS to run elections). This window does not have a form for the user to input information. It has an option for the user to return to the user interface.
Data Members: none
Constructors/Destructor:

- HelpWindow()
  Arguments: none
  Returns: none
  Access: public
- ~HelpWindow()

Member Functions:

- DisplayHelp(): void
  Arguments: none
  Returns: none
  Access: public
  Description: This function displays the window and help content

Flow of Events:

1. User views help contents
2. User closes the window
3. Program returns to UserInterface

BallotFileProcessor
Name: BallotFileProcessor
Type: Class
Description: This class is instantiated once in UserInterface class. It reads ballot

file(s) line by line. It extracts candidate information first and creates a candidate list in the VotingInfo object. It then reads ballot information one at a time and updates the ballot list in the VotingInfoObject.
Data Members:

- ballotFiles
  Type: ifstream
  Access: private

Constructors/Destructor:

- BallotFileProcessor(string)
  Arguments: string fileName - Ballot File Name to be processed
  Returns: none
  Access: public
- ~BallotFileProcessor()

Member Functions:

- ProcessFiles(VotingInfo*): void
  Arguments: VotingInfo* votingInfo
  Returns: none
  Access: public
  Description: This function opens Ballot files. It reads in content line by line. The function extracts the candidates' information from the header of the first file, constructs Candidate objects, then updates the candidate list in the VotingInfo object. The function then reads ballots line by line, constructs Ballot objects, then updates the ballot list in the VotingInfo object.

VotingInfo
Name: VotingInfo
Type: Class
Description: VotingInfo class stores information needed for election. It is instantiated once in the UserInterface object. It is passed as a reference argument to BallotFileProcessor object method and Election object method.
Data Members:

- _numSeats
  Type: int
  Access: private
- _algorithm
  Type: int
  Access: private
- _numCandidates
  Type: int
  Access: private
- _iniCandidateList
  Type: List<Candidate>
  Access: private

- _numBallots
  Type: int
  Access: private
- _iniBallotList
  Type: List<Ballot>
  Access: private

Constructors/Destructor:

- VotingInfo (int algorithmChoice, int numSeats)
  Arguments: int algorithmChoice, int numSeats
  Returns: none
  Access: public
- ~VotingInfo()

Member Functions:

- GetNumSeats(): int
  Arguments: none
  Returns: int
  Access: public
  Description: Get value of private member _numSeats
- GetAlgorithm(): int
  Arguments: none
  Returns: int
  Access: public
  Description: Get value of private member _algorithm
- AddCandidateToCandidateList(Candidate): void
  Arguments: Candidate - Candidate object
  Returns: none
  Access: public
  Description: Add a candidate object to private member _iniCandidateList
- IncrementNumCandidates(): void
  Arguments: none
  Returns: none
  Access: private
  Description: This function is called by AddCandidateToCandidateList method. It increments private member _numCandidates when a Candidate object is added to _iniCandidateList
- AddBallotToBallotList(Ballot): void
  Arguments: Ballot - a Ballot object
  Returns: none
  Access: public
  Description: Add a ballot object to private member _iniBallotList
- IncrementNumBallots():void
  Arguments: none
  Returns: none
  Access: public

Description: This function is called by AddBallotToBallotList method. It increments private member _numBallots when a Ballot object is added to _iniBallotList

- GetNumCandidate(): int
  Arguments: none
  Returns: int
  Access: public
  Description: Returns value of private member _numCandidate
- GetNumBallots(): int
  Arguments: none
  Returns: int
  Access: public
  Description: Returns value of private member _numBallot
- GetCandidateList(): List<Candidate>
  Arguments: none
  Returns: List<Candidate>
  Access: public
  Description: Returns values of private member _iniCandidateList
- GetBallotList(): List<Ballot>
  Arguments: none
  Returns: List<Ballot>
  Access: public
  Description: Returns values of private member _iniBallotList

Ballot

Name: Ballot
Type: Class
Description: An object defined by this class stores information from a single ballot. Its data member _id is unique to each Ballot object, assigned sequentially when BallotFileProcessor generates Ballot objects. Its data member _rankedCandidateIDList stores ranked Candidate IDs based on the ranking from original ballots (i.e. if a ballot ranked candidates A,B,C,D,E: 0,1,0,2,3, the _rankedCandidateIDList will be {2,4,5}) .
Data Members:

- _id
  Type: int
  Access: private
- _rankedCandidateIDList
  Type: List<int>
  Access: private

Constructors/Destructor:

- Ballot(int id, int[] rankingArray)
  Arguments: int id, int[] rankingArray

Returns: none
Access: public
Description: This constructor takes an integer and the raw ranking array from ballots, assign integer input to private data member _id, convert the raw ranking array to the rankedCandidateIDList and store it in the corresponding data member

- ~Ballot()

Member Functions:

- GetId(): int
  Arguments: none
  Returns: int
  Access: public
  Description: Returns value of private data member _id
- GetRankedCandidateIDList(): int
  Arguments: none
  Returns: List<int>
  Access: public
  Description: Returns value of private data member _rankedCandidateIDList

Candidate

Name: Candidate
Type: Class
Description: An object defined by this class stores information of a candidate from the original ballot files. It defines a set of associated methods. Its data member _id is a unique integer, assigned sequentially when BallotFileProcessor generates each object. Its data member _ballotList stores ballot objects assigned to the candidate object. This class's data members are protected, so that they can be inherited by STVCandidate class. This class is used by plurality election directly.
Data Members:

- _id
  Type: int
  Access: protected
  Description: Unique integer id assigned at the time of object creation
- _name
  Type: string
  Access: protected
  Description: String name storing a real world candidate's name from the ballot file
- _party
  Type: string
  Access: protected
  Description: Character string stores a candidate's associated party. This member may not be needed depending on what information is available in

the ballot file.

- _numBallots
  Type: int
  Access: protected
  Description: integer storing the number of ballots a candidate had been awarded in ballot distribution process
- _ballotList
  Type: List<Ballot>
  Access: protected
  Description: A list of Ballot objects storing ballots awarded to the candidate object

Constructors/Destructor:

- Candidate(int id, string name, string party)
  Arguments: int id, string name, string party
  Returns: none
  Access: public
  Description: Construct a candidate object and initialize data members with respective input argument values
- ~Candidate()

Member Functions:

- GetId(): int
  Arguments: none
  Returns: int
  Access: public
  Description: This function returns value of protected data member _id
- GetName(): string
  Arguments: none
  Returns: string
  Access: public
  Description: This function returns value of protected data member _name
- IncrementNumBallots(): int
  Arguments: none
  Returns: int
  Access: protected
  Description: This function increments protected data member _numBallots, then returns the value of _numBallots. This function is called by another member function AddBallot when a ballot is awarded to the candidate object
- _GetNumBallots(): int
  Arguments: none
  Returns: int
  Access: public
  Description: This function returns the value of protected data member _numBallots

- AddBallot(Ballot): int _numBallots
  Arguments: Ballot
  Returns: int
  Access: public
  Description: This function add a ballot object to data member _ballotList. It will also call the protected member function IncrementNumBallots(). It then returns the value of protected data member _numBallots

STVCandidate
Name: STVCandidate
Type: Class
Description: This class inherits all members of Candidate class. This class defines one additional data member _firstBallotNum to store the order in which the STVCandidate object received its first ballot (used in tie breaking). This class also defines 3 additional member functions described in below member functions
Data Members:

- _firstBallotNum
  Type: int
  Access: private
  Description: An integer storing the order in which a candidate object gets its first ballot. This information is used in STVElection BreakTie method

Constructors/Destructor: same as parent class Candidate
Member Functions:

- RemoveBallotList(void): List<Ballot>
  Arguments: none
  Returns: List<Ballot>
  Access: public
  Description: This function stores the value of data member _ballotList in a temporary list for return, then sets data member _ballotLIst to empty, and sets data member _numBallots to zero. This function returns the value of data member _ballotList stored in the temporary list. This operation is done when a candidate is moved to loser list in STVElection
- SetNumBallotZero(void):
  Arguments: none
  Returns: none
  Access: private
  Description: This function sets the data member _numBallots to zero. This is a private function only called by member function RemoveBallotList()
- SetFirstBallotNum(int): void
  Arguments: int
  Returns: void
  Access: public
  Description: This function sets the value of private data member _firstBallotNum

- GetFirstBallotNum(): int
  Arguments: none
  Returns: int
  Access: public
  Description: This function returns the value of private data member _firstBallotNum

Election
Name: Election
Type: Abstract Class
Description: This is an abstract class, defining the common interface of election class. It has two protected data members: electionRecord and displayResult. It has two virtual methods: AbstractElection() and RunElection(VotingInfo*). This class is used to declare an election object in the UserInterface class.
Data Members:

- electionRecord
  Type: ElectionRecord Class
  Access: protected
  Description: This data member is an abstract ElectionRecord. It is instantiated by a concrete ElectionRecord class at runtime based on user selection of election algorithm
- displayResult
  Type: ResultDisplay Class
  Access: protected
  Description: This data member is an abstract ResultDisplay. It is instantiated by a concrete ResultDisplay class at runtime based on user selection of election algorithm

Constructors:

- Virtual AbstractElection()
  Arguments: none
  Returns: none
  Access: public
  Description: This is a virtual constructor. Concrete classes will define implementation.
- ~AbstractElection()

Member Functions:

- Virtual RunElection(VotingInfo*): bool
  Arguments: VotingInfo*
  Returns: none
  Access: public
  Description: This is a virtual method defining method inputs and outputs. Its implementation is defined by concrete classes inheriting from this abstract class

STVElection
Name: STVElection
Type: Concrete Class
Description: This concrete class inherits from the abstract Election Class. It defines the member data and member functions in the abstract Election Class. This class is used to instantiate an object declared by the abstract class in UserInterface.
Data Members:

- electionRecord
  Type: STVElectionRecord Class
  Access: private
  Description: This data member keeps all the necessary data structures tracking election progress. Its methods provide means to update the records. Please refer to the description of STVElectionRecord Class for details.
- displayResult
  Type: STVResultDisplay
  Access: private
  Description: This data member instantiates an STVResultDisplay object. This object will display the election result using its member method when the election is complete.

Constructors:

- STVElection()
  Arguments: none
  Returns: none
  Access: public
  Description: This constructor instantiates an STVElection object.

Member Functions:

- RunElection(VotingInfo*): void
  Arguments: VotingInfo*
  Returns: none
  Access: public
  Description: This function implements the STV election algorithm defined in the requirement document. It implements all the control loops and checks. It uses the STVElectionRecord object to store all necessary intermediate results and it uses the STVElectionRecord object methods to update those results.

PluralityElection
Name: PluralityElection
Type: Concrete Class
Description: This concrete class inherits from the abstract Election Class. It defines the member data and member functions in the abstract Election Class. This class is used to instantiate an object declared by the abstract class in UserInterface.
Data Members:

- electionRecord
  Type: PluralityElectionRecord Class
  Access: private
  Description: This data member keeps all the necessary data structures tracking election progress. Its methods provide means to update the records. Please refer to the description of PluralityElectionRecord Class for details.
- displayResult
  Type: PluralityResultDisplay
  Access: private
  Description: This data member instantiates an PluralityResultDisplay object. This object will display the election result using its member method when the election is complete.

Constructors:

- PluralityElection()
  Arguments: none
  Returns: none
  Access: public
  Description: This constructor instantiates an PluralityElection object.

Member Functions:

- RunElection(VotingInfo*): void
  Arguments: VotingInfo*
  Returns: none
  Access: public
  Description: This function implements the STV election algorithm defined in the requirement document. It implements all the control loops and checks. It uses the PluralityElectionRecord object to store all necessary intermediate results and it uses the PluralityElectionRecord object methods to update those results.

ElectionRecord

Name: ElectionRecord
Type: Abstract Class
Description: This is an abstract class defining common interfaces and data members of ElectionRecords. The concrete classes inheriting this class define the implementation. The purpose of ElectionRecord is to define data structures storing intermediate results of an election and provide methods to update those records. This design encapsulates the record management and lower level functions, so that the election object only needs to be concerned with the top level algorithm implementation.
Data Members:

- nonDistributedBallotList
  Type: List<Ballot>
  Access: protected

Description: This list stores Ballot objects to be distributed.

- nonElectedCandidateList
  Type: List<Candidate>
  Access: protected
  Description: This list stores eligible Candidates who are not elected or eliminated.
- winnersList
  Type: List<Candidate>
  Access: protected
  Description: This list stores Candidates who have been declared winners. The list is in the order of winning. I.e. The first element in the list is the first candidate who is put on the list and so forth.
- losersList
  Type: List<Candidate>
  Access: protected
  Description: This list stores Candidates who have been declared losers. The list is in the order of losing. I.e. The first element in the list is the first candidate who is put on the list and so forth.

Constructors/Destructor:

- ElectionRecord (List<Candidate>, List<Ballot>)
  Arguments: List<Candidate>, List<Ballot>
  Returns: none
  Access: public
  Description: This constructor function constructs ElectionRecord and initializes data members with input arguments.
- ~ElectionRecord()

Member Functions:

- DistributeBallots(): void
  Arguments: none
  Returns: none
  Access: public
  Description: This function loops through Ballot objects in nonDistributedBallotList and assigns Ballots to Candidates in the nonElectedCandidateList. This is a virtual function defining input and output parameters. The definition will be done in concrete classes STVElectionRecord and PluralityElectionRecord.
- SortNonElectedCandidateList(): void
  Arguments: none
  Returns: none
  Access: public
  Description: This function sorts Candidates based on the number of votes each candidate had been awarded. If there is a tie, this function will call the BreakTies method to get the winner. This is a virtual function defining input and output parameters. The definition will be done in concrete

classes STVElectionRecord and PluralityElectionRecord.

- BreakTies(Candidate, Candidate): Candidate
  Arguments: (Candidate, Candidate)
  Returns: Candidate
  Access: public
  Description: This function takes in two Candidate objects and returns the winner Candidate in a tie breaking. This is a virtual function defining input and output parameters. The definition will be done in concrete classes STVElectionRecord and PluralityElectionRecord.

STVElectionRecord
Name: STVElectionRecord
Type: Concrete Class
Description: This concrete class defines the implementation of methods in the abstract ElectionRecord class. It adds two additional data members and adds eight more methods. This class defines data structures needed to store intermediate election results and defines methods needed to update those election results.
Data Members (Additional from the ElectionRecord):

- _discardedBallotList
  Type: List<Ballot>
  Access: private
  Description: This list stores ballots that cannot be assigned to a candidate, due to all its ranked candidates being either on the winnersList or the losersList.
- _DroopQuota
  Type: int
  Access: private
  Description: A data member storing Droop Quota value. This value is set at the time of instantiation (by constructor).

Constructors:

- STVElectionRecord(List<STVCandidate>, List<Ballot>, int DroopQuota)
  Arguments: List<STVCandidate>, List<Ballot>, int DroopQuota
  Returns: none
  Access: public
  Description: Construct STVElectionRecord object with initial values.

Member Functions:

- ShuffleBallots(): void
  Arguments: none
  Returns: none
  Access: public
  Description: This function shuffles the ballots in the nonDistributedBallotList.
- DistributeBallots(): void
  Arguments: none

Returns: none
Access: public
Description: This function loops through Ballot objects in nonDistributedBallotList and assigns Ballots to Candidates in the nonElectedCandidateList. For each Ballot object, first the function gets the ranked Candidate list by calling Ballot.GetRankedCandidateIDList(). Next, starting from the first ranked candidate, the function checks if the next ranked candidate is on the nonElectedCandidateList. If the next ranked candidate is on the list, the Ballot object is assigned to that Candidate by calling Candidate.AddBallot(Ballot). Candidate.AddBallot(Ballot) returns the number of ballots the Candidate has. Function then calls CheckDroop(int) with this return value as input. If CheckDroop returns true, then the function will call AddCandidateToWinnersList(Candidate). If all ranked candidates on the ballot had been checked and no candidate is on the nonElectedCandidateList, the function will call AddBallotToDiscardedBallotList(Ballot). The function will continue distributing ballots until all ballots in the nonDestributedBallotList are gone.

- CheckDroop(int numBallots): bool
  Arguments: int numBallots
  Returns: boolean
  Access: public
  Description: This function checks if the input argument is greater than or equal to the data member value _DroopQuota.
- AddCandidateToWinnersList(STVCandidate): void
  Arguments: STVCandidate
  Returns: void
  Access: public
  Description: This function adds a STVCandidate to its data member winnersList.
- SortNonElectedCandidateList(): void
  Arguments: none
  Returns: none
  Access: public
  Description: This function sorts Candidates based on the number of votes each candidate had been awarded. If there is a tie, this function will call the BreakTies method to get the winner.
- RemoveLastCandidateFromNonElectedCandidateList (): STVCandidate
  Arguments: none
  Returns: STVCandidate
  Access: public
  Description: This function takes the last member off the nonElectedCandidateList and returns the object.
- AddCandidateToLosersList(STVCandidate): List<Ballot>
  Arguments: STVCandidate

Returns: List<Ballot>
Access: public
Description: This function adds a STVCandidate object to the loserList, removes the Ballot list from this STVCandidate object and returns the removed Ballot list.

- AddLoserBallotsToNonDistributedBallotList (List<Ballot>): void
  Arguments: List<Ballot>
  Returns: none
  Access: public
  Description: This function adds its input Ballot list to the nonDistributedBallotList.
- AddBallotToDiscardedBallotList (Ballot): void
  Arguments: Ballot
  Returns: void
  Access: public
  Description: This function adds its input Ballot object to the discardedBallotList.
- BreakTies(const STVCandidate, const STVCandidate): const STVCandidate
  Arguments: const STVCandidate, const STVCandidate
  Returns:const STV Candidate
  Access: public
  Description: This function takes two STVCandidate objects with equal number of votes, compares their _firstBallotNum data member value, and returns the STVCandidate object with the smaller _firstBallotNum value. This function is const safe.
- PopCandidateOffLosersList(); STVCandidate
  Arguments: none
  Returns: STVCandidate
  Access: public
  Description: This function removes the last STVCandidate object from the losersList and returns this object.

PluralityElectionRecord

Name: PluralityElectionRecord
Type: Concrete Class
Description: This concrete class defines the implementation of methods in the abstract ElectionRecord class. It adds two more methods. This class defines data structures needed to store intermediate election results and defines methods needed to update those election results.
Data Members: no additional data members from the abstract ElectionRecord class
Constructors:

- PluralityElectionRecord (List<Candidate>, List<Ballot>)
  Arguments: List<Candidate>

Returns: none
Access: public
Description: This constructor function instantiates a PluralityElectionRecord object and initializes its data members nonDistributedBallotList and nonElectedCandidateList.

Member Functions:

- DistributeBallots(): void
  Arguments: none
  Returns: none
  Access: public
  Description: This function loops through Ballot objects in nonDistributedBallotList and assigns Ballots to Candidates in the nonElectedCandidateList. For each Ballot object, first the function gets the ranked Candidate list by calling Ballot.GetRankedCandidateIDList(). For Candidate object, the rankedCandidateList contains only one Candidate member. The function then adds this Ballot object to the Candidate on its rankedCandidateList. The function will go through all Ballot objects on the nonDistributedBallotList until no Ballot object is left on the list.
- SortNonElectedCandidateList(): void
  Arguments: none
  Returns: none
  Access: public
  Description: This function sorts Candidates based on the number of votes each candidate had been awarded. If there is a tie, this function will call the BreakTies method to get the winner.
- BreakTies(Candidate, Candidate): Candidate
  Arguments: Candidate, Candidate
  Returns: Candidate
  Access: public
  Description: This function takes in two Candidate objects with an equal number of votes. It randomly picks one of the Candidates and returns the picked object as the winner.
- MoveFirstNCandidatesFromNonElectedListToWinnersList (int N): void
  Arguments: int N
  Returns: none
  Access: public
  Description: This function moves input N number of Candidate objects from the nonElectedCandidateList to winnersList. This input number is the number of seats to be filled.
- MoveLastNCandidatesFromNonElectedListToLosersList (int N): void
  Arguments: int N
  Returns: none
  Access: public
  Description: This function moves input N number of Candidate objects

from the nonElectedCandidateList to losersList. This input number is the number of Candidates subtract the number of seats to be filled.

ResultDisplay
Name: ResultDisplay
Type: Abstract Class
Description: This is an abstract class defining the user interface for displaying election results. It defines common data members and member functions the concrete class shares.
Data Members: none
Constructors/Destructor:

- ResultDisplay()
  Arguments: none
  Returns: none
  Access: public
  Description: Constructor for ResultDisplay
- ~ResultDisplay()

Member Functions:

- virtual DisplayElectionResults (AbstractElectionRecord*, VotingInfo*): void
  Arguments: AbstractElectionRecord*, VotingInfo*
  Returns: none
  Access: public
  Description: This is a virtual method that specifies inputs and outputs of DisplayElectionResults. Concrete classes will define the implementation of this method.

STVResultDisplay
Name: STVResultDisplay
Type: Concrete Class
Description: This concrete class implements the method defined in the parent abstract class. It takes the STVElectionRecord object and VotingInfo object and displays election results using information from those objects.
Data Members: none

Constructors:

- ResultDisplay()
  Arguments: none
  Returns: none
  Access: public
  Description: Instantiate STVElectionDisplay window

Member Functions:

- DisplayElectionResults (STVElectionRecord*, VotingInfo*): void
  Arguments: STVElectionRecord*, VotingInfo*
  Returns: none

Access: public
Description: Takes in STVElectionRecord object and VotingInfo object after STVElection.RunElection() is completed without exception, and displays the election results specified in SRS document.

PluralityResultDisplay
Name: PluralityResultDisplay
Type: Concrete Class
Description: This concrete class implements the method defined in the parent abstract class. It takes the PluralityElectionRecord object and VotingInfo object and displays election results using information from those objects.
Data Members: none
Constructors:

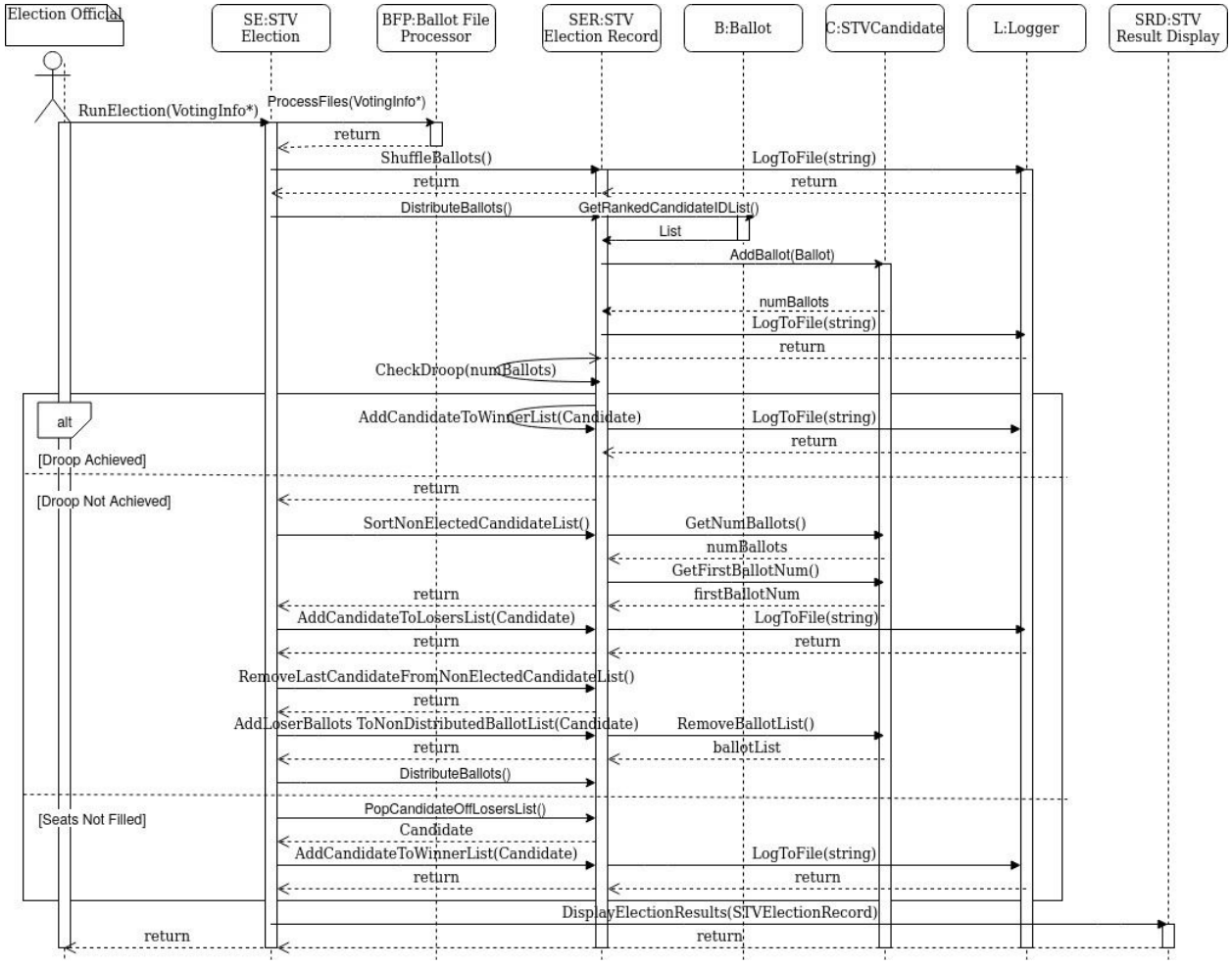
- ResultDisplay()
  Arguments: none
  Returns: none
  Access: public
  Description: Instantiate PluralityElectionDisplay window

Member Functions:

- DisplayElectionResults (PluralityElectionRecord*, VotingInfo*): void
  Arguments: PluralityElectionRecord*, VotingInfo*
  Returns: none
  Access: public
  Description: Takes in PluralityElectionRecord object and VotingInfo object after PluralityElection.RunElection() is completed without exception, and displays the election results specified in SRS document.

3.2 Decomposition Description

The sequence diagram for the SVT system is given below.



To run the STV election, the user first calls the RunElection method from the STV Election object. This method runs the STV election to completion. First, the ballots are read in using the Ballot File Processor. This processor creates a ballot object for each line of the ballot file. It also processes the ballots to create a sorted list in the order of prefered candidates for that particular ballot. The ballots are then shuffled using the STV Election Record object and the shuffle is recorded in the Logger. The STV Election object then calls the DistributeBallot method to start issuing ballots to candidates. This method loops through all of the ballot objects, and for each ballot, calls the GetRankedCandidateIDList method to get the list of candidates for that ballot in order of voter preference. The STV Election Record then determines the prefered candidate by comparing the voter preference to the list of candidates that have not won. The preferred candidate for that ballot is determined and then the addBallot method for the STVCandidate object is called to

add that ballot to that candidate's ballot list. The number of ballots for that candidate is returned and the data is recorded in the Logger. Droop is then checked for this candidate.

If droop is achieved, then the winner is added to the winner list and the data is recorded in the Logger.

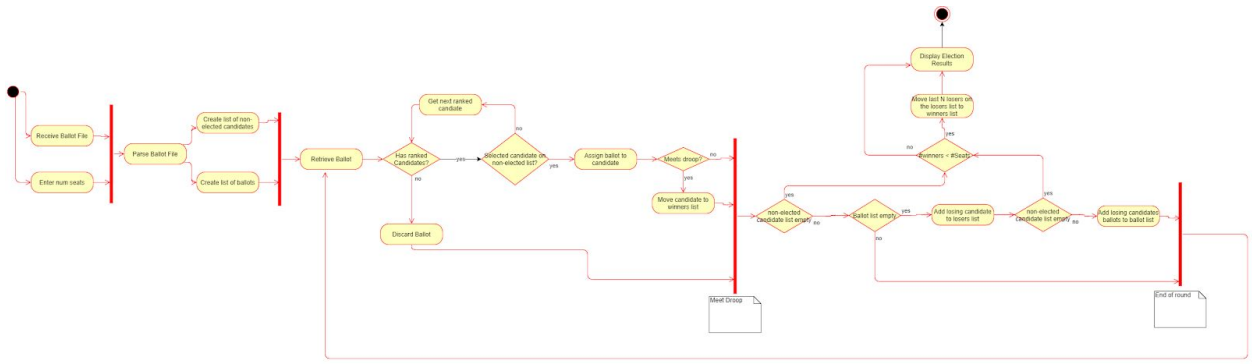
If droop is not achieved and all of the ballots have been distributed, then the process returns to the STV Election method. This method calls the SortNonElectedCandidateList method in the STV Election Record object which gets the number of ballots and the first ballot id for each candidate on that list to determine the loser. That candidate is added to the losers list and the data is captured in the Logger.

The losing candidates ballots are then moved back to the NonDistributedBallotList and the DistributeBallot method is called again to distribute the ballots in the list.

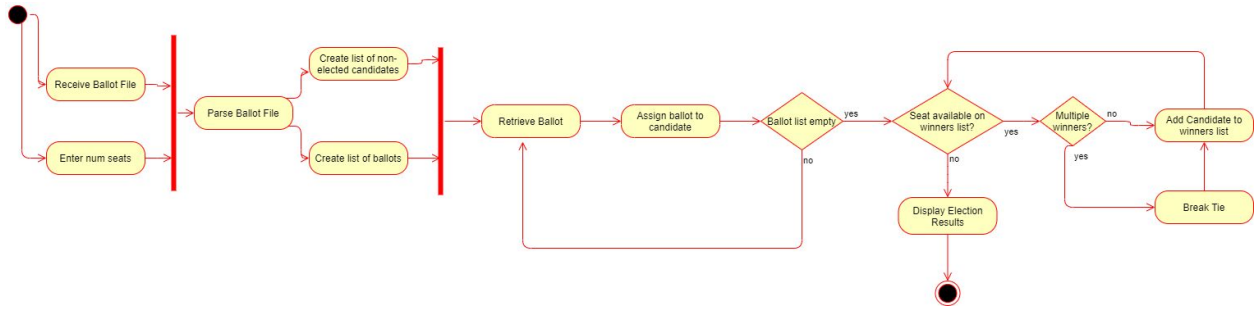
Once the NonElectedCandidatesList is empty, if the seats are still not full, then the PopCandidateOffLosersList method from the STV Election Record object is called to get the last loser. This candidate is then added to the winners list and the data is recorded in the Logger.

Once all seats have been filled, the election results are displayed and the program returns back to the user.

STV Activity Diagram:



Plurality Activity Diagram:



3.3 Design Rationale

We adopted the OOP (object oriented programming) principle in our VS design. Our system consists of following major objects:

- UserInterface
- BallotFileProcesser
- HelpWindow
- Logger
- VotingInfo
- STVElection
- PluralityElection
- Ballot
- Candidate
- STVCandidate
- STVElectionRecord
- PluralityElectionRecord
- STVResultDisplay
- PluralityResultDisplay

All data members in each object are set to private or protected. Methods are provided to access data members. This design allows validity check of data member access, reducing errors and bugs.

In addition, we adopted following design patterns in our VS design:

1. Inheritance
    - STVCandidate inherits from Candidate
    - STVElection and PluralityElection inherit from Election
    - STVResultDisplay and PluralityResultDisplay inherit from ResultDisplay
    - STVElectionRecord and PluralityElectionRecord inherits from ElectionRecord
2. Composition
    - Election object is composed of ElectionRecord object and

ResultDisplay object
    - ElectionRecord object is composed of Candidate objects and Ballot objects
    - VotingInfo object is composed of Candidate objects and Ballot objects
    - UserInterface object is composed of HelpWindow object, VotingInfo object, BallotFileProcessor object and Election object
3. Abstract Factory
   We adopted the abstract factory design pattern to implement Election class and ElectionRecord class. This gives us extendibility in our code design. If we need to add another election algorithm to the system. We only need to add another concrete class and add another runtime choice instead of a total refactor of our code.
4. Singleton Pattern
   We used the singleton pattern in our Logger class design. This pattern allows Logger to be accessible everywhere and there is only one instance of it.

4. DATA DESIGN

4.1 Data Description

Inputs to the Voting System:

- **Shuffle Bool:**
    - This value is a public boolean called BallotShuffleOption, has an initial value of True, and can be changed using an initial command line argument “-t”.
    - This is used in the STVElection class to either run or not run the ShuffleBallots method in the STVElectionRecord class.
- **Num Seats:**
    - This value is given as an integer called _numSeats through the UserInterface class input field.
    - This value is passed to the VotingInfo class upon initialization of the VotingInfo class.
    - The value can be found by using the getNumSeats method in the VotingInfo class.
- **Election Type:**
    - This value is an integer called algorithmChoice in the UserInterface class. This gets passed to the VotingInfo class upon initialization of the class.
    - In the VotingInfo class it gets saved as _algorithm and is also an integer. Other methods can then call this value using the getAlgorithm method in the VotingInfo class.
    - A value of zero corresponds to an STV election and a value of 1

corresponds to a Plurality election.

- **Ballot Files:**
    - The string for the file name of the ballot file is entered into the system through a field in the UserInterface class and is stored in the BallotFileProcessor class.
    - When the ProcessFiles method in the BallotFileProcesser class is called, the method starts to read in each line of the ballot file.
    - The first line of the file reads all the candidates and the ProcessFiles method creates a Candidate class for each of these. The Candidate class is explained later.
    - The rest of the ballot file is a voter's ballot and the ProcessFiles method creates a Ballot class for each of these. The ballot class is explained later.

Internal Data Structures:

- **Ballots:**
    - These objects are all created through the BallotFileProcessor class and upon initialization they create a rankedCandidateIDList where the candidates are placed into a list depending on the preference stated in the voter ballot line.
    - Each ballot is also given an integer id number based on their order in the shuffled ballot list. This integer number is used as a timestamp for the STV election to see the earliest ballot issued to a candidate.
    - The rankedCandidateIDList can be accessed using the getRankedCandidateIDList method of the Ballot class.
    - The ballot id can be accessed using the getID method of the Ballot class.
- **Candidates:**
    - These objects are all created through the BallotFileProcessor class and upon initialization they are passed: an integer called id which is a numeric value unique to each candidate, a string value called name to capture the candidate's name, and a string value called party to capture the party (assuming there and names and parties for each candidate).
    - Ballots can be added to a candidate using the AddBallot method of the candidate class.
    - When an STV election is being run the STVCandidate class is initialized as a subclass of the Candidate class. This class has the added methods required for STV elections such as RemoveBallotList which pulls the ballot list if that candidate has been found to lose the election and their ballots can be redistributed to the rest of the candidates.
- **Droop:**
    - This value is calculated in the RunElection method of the STVElection class. The name of the variable is DroopQuota and is an integer value.
    - This value is calculated using the numSeats integer value and the numBallots integer value from the getNumBallots method of the VotingInfo class.

- The value is used to initialize the STVElectionRecord class and the final droop quota is stored in a variable in the STVElectionRecord called _DroopQuota. This value is never accessed directly. There is a method called checkDroop in the STVElectionRecord class that takes an integer as the number of ballots and checks them against this droop value.

Outputs of the Voting System:

- **Log File:**
  - This log file is created from an internal string in which the value “audit.txt” is the resulting file name.
  - This creates a stream using the Logger class where the election information can be saved.
  - Election information can be saved to this file using the LogToFile method of the Logger class. A string of information is passed into this method and this will be written to the audit file.

4.2 Data Dictionary

Object	**Attribute**	**Method**
Ballot	_id - int _rankedCandidateIDList - List<int>	GetId(): int GetRankedCandidateIDList(): List<int>
BallotFileProcessor	ballotFiles - ifstream	ProcessFiles(): void BallotFileProcessor()
Candidate	_id: int _name: string _party: string _numBallots: int _ballotList: List<Ballot>	Candidate GetId(): int GetName(): string IncrementNumBallots(): int GetNumBallots(): int AddBallot(Ballot): int (_numBallots)
HelpWindow		HelpWindow() DisplayHelp(): void
Logger	_logFile: static	GetLogFile(): string

	Logger*	LogToFile(string): void
Main	userInterface: UserInterface ballotShuffleOption: bool logger: Logger	main(argc: int, argv** char)
PluralityElection		PluralityElection() RunElection(VotingInfo*): void
PluralityElectionRecord		PluralityElectionRecord (List<Candidate>, List<Ballot>) DistributeBallots(): void SortNonElectedCandidateList(): void BreakTies(Candidate, Candidate): Candidate MoveFirstNCandidatesFromNonElectedListToWinnersList (int N): void MoveLastNCandidatesFromNonElectedListToLosersList (int N): void
PluralityResultDisplay		ResultDisplay() DisplayElectionResults (PluralityElectionRecord*, VotingInfo*): void
STVCandidate	_firstBallotNum: int	STVCandidate(int id, string name, string party) RemoveBallotList(void): List<Ballot> SetNumBallotZero(void): void SetFirstBallotNum(int); void GetFirstBallotNum(): int
STVElection		STVElection() RunElection(VotingInfo*): void

STVElectionRecord	_discardedBallotList: List<Ballot> _DroopQuota: int	STVElectionRecord(List<STVCandidate>, List<Ballot>, int DroopQuota) ShuffleBallots(): void DistributeBallots(): void CheckDroop(int numBallots): bool AddCandidateToWinnersList(STVCandidate): void SortNonElectedCandidateList(): void RemoveLastCandidateFromNonElectedCandidateList (): STVCandidate AddCandidateToLosersList(STVCandidate): List<Ballot> AddLoserBallotsToNonDistributedBallotList (List<Ballot>): void AddBallotToDiscardedBallotList (Ballot): void BreakTies(const STVCandidate, const STVCandidate): const STVCandidate PopCandidateOffLosersList(); STVCandidate
STVResultDisplay		ResultDisplay() DisplayElectionResults (STVElectionRecord*, VotingInfo*): void
UserInterface	helpWindow: HelpWindow votingInfo: VotingInfo ballotFileProcessor: BallotFileProcessor election: AbstractElection	UserInterface() RunUserInterface(): void
VotingInfo	_numSeats: int _algorithm: int	VotingInfo(int algorithmChoice, int numSeats)

	_numCandidates: int _iniCandidateList: List<Candidate> _numBallots: int _iniBallotList: List<Ballot>	GetNumSeats(): int GetAlgorithm(): int AddCandidateToCandidateList(Candidate): void IncrementNumCandidates(): void AddBallotToBallotList(Ballot): void IncrementNumBallots():void GetNumCandidate(): int GetNumBallots(): int GetCandidateList(): List<Candidate> GetBallotList(): List<Ballot> VoingInfo()

5. COMPONENT DESIGN

The section shows implementations of each of the methods and constructors in each class written in pseudocode.

main

```
int main(argc int, argv** char) {
        static bool ballotShuffleOption = True;
        if argv[1] == '-t'{
                ballotShuffleOption = False;}
        initialize class Logger;
        initialize class userInterface;
        userInterface.RunUserInterface();
        return 0;
}
```

Logger

```
Logger()
{
```

```
        // Check if an instance of Logger already exists, exit if an instance exists
        // construct logger object using file io stream
        // throw error if file io stream initialization fails
}

String GetLogFile(void) {
        return _logFile*;
}

void LogToFile(String message) {
        //write string to _logFile*;
        //throw error if write to file fails
}
```

UserInterface

```
UserInterface()
{
}

void RunUserInterface(void){
        declare variables;
        output form;
        if help.click == True{
                HelpWindow.DisplayHelp();
        }
        check data boxes for correct format:
        initialize VotingInfo class (VotingInfo*);
        runelectionbtn.click == True {
                if VotingInfo* correct {
                        if electionType == 0 {
                                ballotFileProcessor.ProcessFiles(VotingInfo*);
                                STVElection.RunElection(VotingInfo*);
                        } elif electionType == 1 {
                                ballotFileProcessor.ProcessFiles(VotingInfo*);
                                PluralityElection.RunElection(VotingInfo*);
                        } else {
                                throw Exception(Please specify either STV or Plurality);
                        }
                } else {
                        the Exception(Please use correct data);
                }
        }
}
```

HelpWindow

```
HelpWindow()
{
}

void DisplayHelp()
{
   print(windowContents);
   wait for user to close window;
   return;
}
```

BallotFileProcessor

```
BallotFileProcessor(String ballotfiles)
{
    import ballotfiles to this->ballotFiles
}

void ProcessFiles(VotingInfo*) {
        int linecnt = 1
        if (ballotFiles.is_open()) {
                while (getline(ballotFiles, line)) {
                        if linecnt == 1 {
                                parse line for list of candidates;
                                for( i = 0, i < length(candidates), i++) {
                                        initialize candidate;
                                }
                                update VotingInfo[CandidateList];
                        } else {
                                read line;
                                initialize Ballot;
                                update VotingInfo[BallotList];
                        }
                }
                ballotFiles.close();
        } else {
                print("Cannot open file");
        }
}
```

VotingInfo

```
VotingInfo(int algorithmChoice, int numSeats)
{
   //0 = STV, 1 = Plurality
   if((algorithmChoice < 0) or (algorithmChoice > 1))
   {
      throw Exception(Algorithm choice must be 0 or 1);
   }
   if(numSeats <= 0)
   {
     throw Exception(Need to have at least 1 seat);
  }

   this->_algorithm = algorithmChoice;
   this->_numSeats = numSeats;
}

int GetNumSeats(void)
{
   return this->_numSeats;
}

int GetAlgorithm(void)
{
   return this-> _algorithm;
}

void IncrementNumCandidates(void)
{
   this->_numCandidates += 1;
   return;
}

void AddCandidateToCandidateList(Candidate)
{
   this->_iniCandidateList.append(Candidate);
   this->IncrementNumCandidates();
   return;
}

void IncrementNumBallots(void)
{
   this->_numBallots += 1;
   return;
```

```
}

void AddBallotToBallotList(Ballot)
{
   this->_iniBallotList.append(Ballot);
   this->IncrementNumBallots();
   return;
}

int GetNumCandidate(void)
{
   return this->_numCandidate;
}

int GetNumBallots(void)
{
   return this->_numBallot;
}

List<Candidate> GetCandidateList(void)
{
   return this->_iniCandidateList;
}

List<Ballot> GetBallotList(void)
{
   return this->_iniBallotList;
}
```

Ballot

```
Ballot(int id, int[] rankingArray)
{
   if(id < 0)
  {
      throw exception(ID must be positive int);
      return;
   }
   if(rankingArray contains duplicates)
   {
     throw exception(Ranking list contains duplicates);
     return;
   }
   this->id = id
```

```
    for each element in rankingArray
    {
        this->ranked_candidateIDList.add(element);
    }
}

int GetId()
{
     return this->_id;
}

List<int> GetRankedCandidateIDList()
{
    return this->_ranked_candidateIDList;
}
```

Candidate

```
Candidate(int id, String name, String party)
{
    if(id < 0)
   {
        throw exception(ID must be positive int);
        return;
    }
    this->_id = id;
    this->_name = name;
    this->_party = party;
    this->_numBallots = 0;
    this->_ballotList = emptyList;
}

int GetId()
{
    return this->_id;
}

String GetName()
{
    return this->_name;
}

int IncrementNumBallots()
{
```

```
    this->numBallots++;
    return this->_numBallots;
}

int GetNumBallots()
{
    return this->_numBallots;
}

int AddBallot(Ballot ballot)
{
    this->_ballotList.add(ballot);
    IncrementNumBallots();
    return this->_numBallots;
}
```

STVCandidate

```
STVCandidate(int id, String name, String party)
{
    //error checking done in Candidate constructor
    super(id, name, party);

    this->_firstballotNum = 0;
}

List<Ballot> RemoveBallotList()
{
    List<Ballot> temp_ballot_list = this->_ballotList;
    clear this->_ballotList;
    this->_numBallots = 0;
    return temp_ballot_list;
}

void SetNumBallotZero()
{
    this->_numBallots = 0;
}

int GetFirstBallotNum()
{
    return this->_firstBallotNum;
}
```

```
override
int AddBallot(Ballot ballot)
{
   if(this->_firstBallotNum == 0)
   {
      this->_firstBallotNum = ballot.GetId();
   }
   this->_ballotList.add(ballot);
   IncrementNumBallots();
   return this->_numBallots;
}
```

PluralityElectionRecord

```
PluralityElectionRecord(List<Candidate> candidates, List<Ballot> ballots)
{
   this->nonElectedCandidateList = candidates;
   this->nonDistributedBallotList = ballots;
}

void DistributeBallots()
{
    for each ballot in this->nonDistributedBallotList
    {
       List<int> selected_candidate_list =  ballot.GetRankedCandidateIDList();
       if(selected_candidate_list is empty)
       {
          throw Exception(Ballot did not rank a candidate);
          return;
       }
      int candidate_selected = selected_candidate_list[0];
      this->nonElectedCandidateList[candidate_selected].AddBallot(ballot);
      this->nonDistributedBallotList.remove(ballot);
   }
}

void SortNonElectedCandidateList()
{
   std::sort(this->nonElectedCandidateList.begin(), this->nonElectedCanidateList.end());
}

Candidate BreakTies(List<Candidate> candidates)
{
    return (randomly select candidate in candidates);
```

```
}

void MoveFirstNCandidatesFromNonElectedListToWinnersList(int N)
{
  //Check that we don't add too many winners in RunElection
   int i;
   for (i = 0; i < N; i++)
   {
      winnersList.append(nonElectedCandidateList[i]);
   }
}

void MoveLastNCandidatesFromNonelectedListToLosersList(int N)
{
   int i;
   for (i = 0; i < N; i++)
  {
      losersList.append(nonElectedCandidateList[i]);
  }
}

Candidate BreakTies(Candidate, Candidate)
{
   return(//randomly pick one Candidate);
}
```

STVElectionRecord

```
STVElectionRecord(List<Candidate> candidates, List<Ballot> ballots)
{
   this->nonElectedCandidateList = candidates;
   this->nonDistributedBallotList = ballots;
}

void ShuffleBallots()
{
   randomize ballots in this->nonDistributedBallotList
}

void DistributeBallots()
{
   this->ShuffleBallots();
   for(each ballot in this->nonDistributedBallotList)
   {
```

```
        List<int> candidate_id_list = ballot.GetRankedCandidateIDList();
        int candidate_id = -1;
        while(candidate_id_list is not empty)
        {
            //remove first candidate id
            candidate_id = candidate_id_list.pop(0)
            if(nonElectedCandidateList contains candidate with id candidate_id)
            {
                break;
            }
        }
        //no valid candidate on ballot
        if(candidate_id == -1)
        {
             AddBallotToDiscardedBallotList(ballot);
             continue;
        }
        Candidate selected_candidate = nonElectedCandidateList[candidate_id]
        int num_ballots = selected_candidate.AddBallot(ballot);
        if(this->checkDroop(num_ballots))
        {
            this->AddCandidateToWinnersList(selected_candidate);
        }
    }
}

bool CheckDroop(int numBallots)
{
    if(numBallots >= this->DroopQuota)
    {
        return true;
    }
    return false;
}

void AddCandidateToWinnersList(Candidate candidate)
{
     this->winnersList.append(candidate);
}

void SortNonElectedCandidateList()
{
    std::sort(this->nonElectedCandidateList.begin(), this->nonElectedCanidateList.end());
}
```

```
void RemoveLastCandidateFromNonElectedCandidateList()
{
   this->nonElectedCandidateList.pop_back();
}

List<Ballot> AddCandidateToLosersList(Candidate candidate)
{
   List<Ballot> losers_ballots = candidate.RemoveBallotList();
   this->losersList.append(candidate);
   return losers_ballots;
}

void AddLoserBallotsToNonDistributedBallotList(List<Ballot> ballots)
{
   this->nonDistributedList.append(ballots);
}

STVCandidate BreakTies(STVCandidate *c1, STVCandidate *c2)
{
   return (c1.GetFirstBallotNum<c2.GetFirstBallotNum? c1:c2);
}

Candidate PopCandidateOffLosersList()
{
   return losersList.pop(0);
}
```

PluralityElection

```
PluralityElection()
{
   this->displayResult =PluralityResultDisplay();
}

void RunElection(VotingInfo *votinginfo)
{
   List<Candidate> candidates_list = votinginfo.GetCandidateList()
   List<Ballot> ballots_list = votinginfo.GetBallotList()
   int num_seats = votinginfo.GetNumSeats();
   this->electionRecord = PluralityElectionRecord(candidates_list, ballots_list);
   this->electionRecord.DistributeBallots();
   this->electionRecord.SortNonElectedCandidateList();
   this->electionRecord.MoveFirstNCandidatesFromNonElectedListToWinnersList(
```

```
num_seats)
   this->electionRecord.MoveFirstNCandidatesFromNonElectedListToLosersList((size
of candidates_list) -num_seats);

   this->displayResult.DisplayElectionResults(this->electionRecord, votinginfo);
}
```

STVElection

```
STVElection()
{
   this->displayResult =STVResultDisplay();
}

RunElection(VotingInfo *votinginfo)
{
   List<Candidate> candidates_list = votinginfo.GetCandidateList()
   List<Ballot> ballots_list = votinginfo.GetBallotList()
   int num_seats = votinginfo.GetNumSeats();
   this->electionRecord = STVElectionRecord(candidates_list, ballots_list);
   while(this->electionRecord.nonElectedCandidateList is not empty)
   {
      this->electionRecord.DistributeBallots();
      if(this->electionRecord.nonElectedCandidateList is empty)
         break;
      this->electionRecord.SortNonElectedCandidateList();
      STVCandidate loser_candidate =
this->electionRecord.RemoveLastCandidateFromNonElectedCandidateList();
     List<ballot> loser_ballots =
this->electionRecord.AddCandidateToLosersList(loser_candidate);
     this->electionRecord.AddLoserBallotsToNonDistributedBallotList(loser_ballots);
   }
   while( (size of this->electionRecord.winnerslist) < votinginfo.numSeats)
   {

this->electionRecord.winnerslist.append(this->electionRecord.PopCandidateOffLosersLi
st)
   }
   this->displayResult.DisplayElectionResults(this->electionRecord, votinginfo);
}
```

STVResultDisplay

```
STVResultDisplay()
```

```
{
}

void DisplayElectionResults(STVElectionRecord electionRecord, VotingInfo *votinginfo)
{
   parse electionRecord;
   parse votinginfo;
   display results to screen;
}
```

PluralityResultDisplay

```
PluralityResultDisplay()
{
}

void DisplayElectionResults(PluralityElectionRecord electionRecord, VotingInfo
*votinginfo)
{
   parse electionRecord;
   parse votinginfo;
   display results to screen;
}
```

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The user interface will consist of 3 separate windows: a startup/information gathering window, an election results window, and a help window.

Startup/information gathering: The startup/information gathering window will be the first window that the user sees when starting the Voting System program. The window will have fields where the user can enter the necessary information to run the election, which includes the election type (STV/plurality), the number of seats, and a field to list the ballot files to be used. The window will also have a button to run the election and will show the status of the election (Not Running / Running / Complete). The window will also have a menu bar where the user can access the help menu.

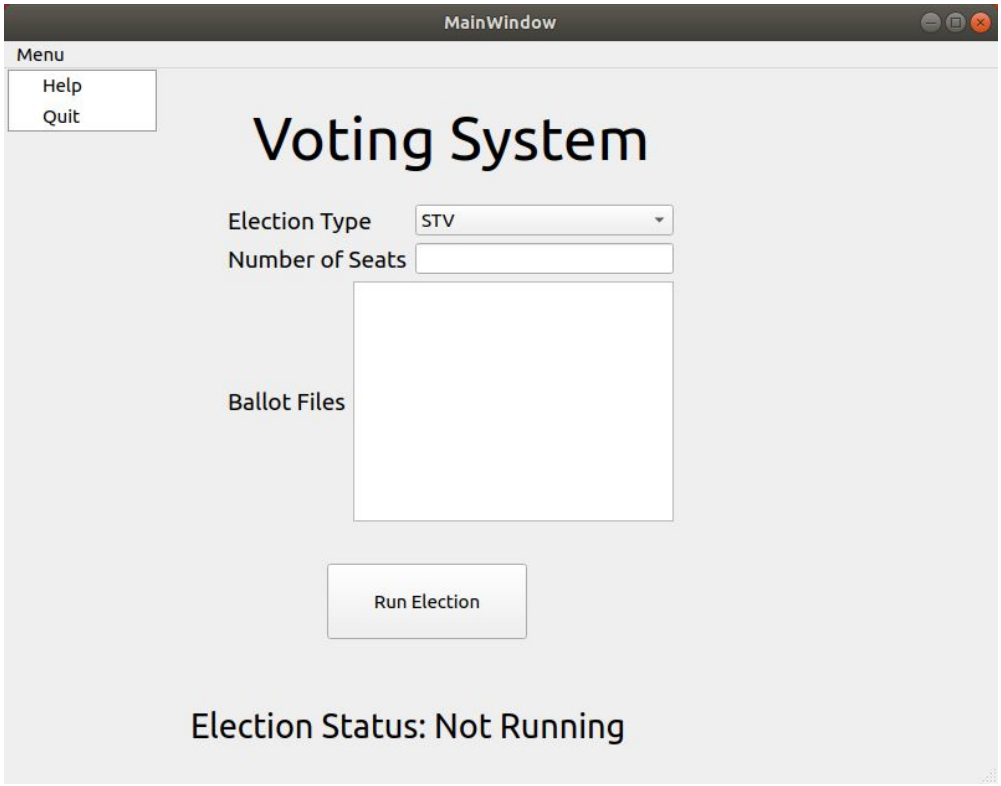
Election results window: The election results window will display information about the election after it is run. It will display the results of the election as well as additional information about the election. This additional information will include election type (STV/plurality), number of ballots, number of seats, number of

candidates and the droop quota (STV election only). There will also be a menu bar where the user can access the help menu.

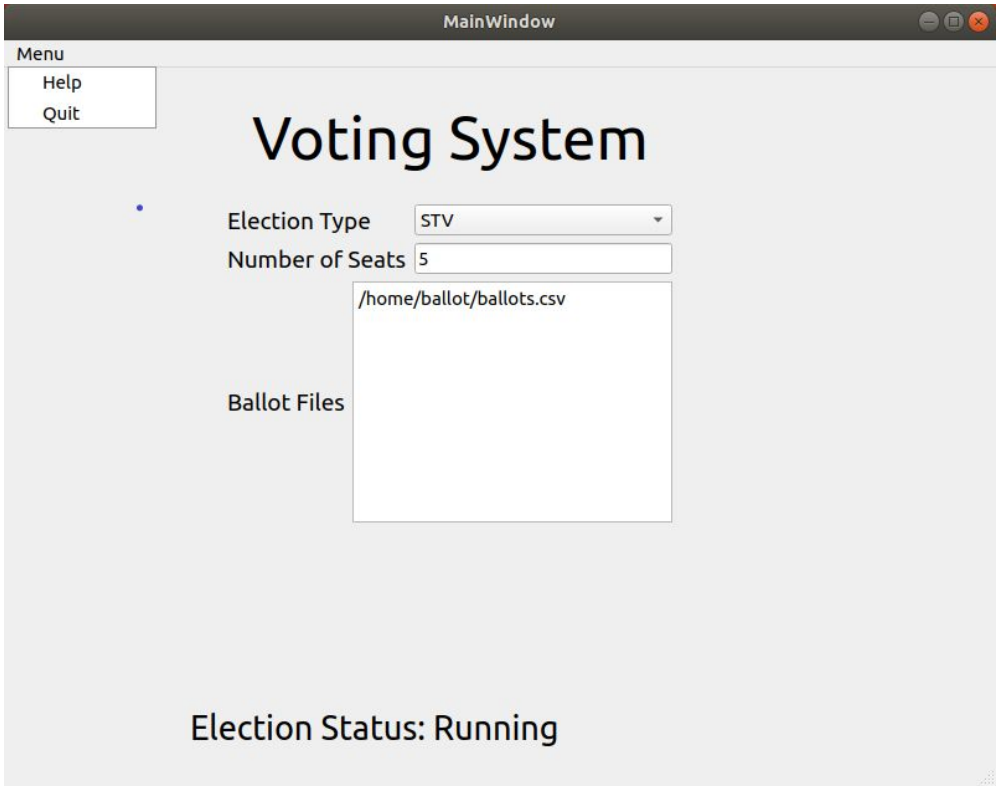
Help Window: The help window will display helpful information / user guide to the user.

6.2 Screen Images

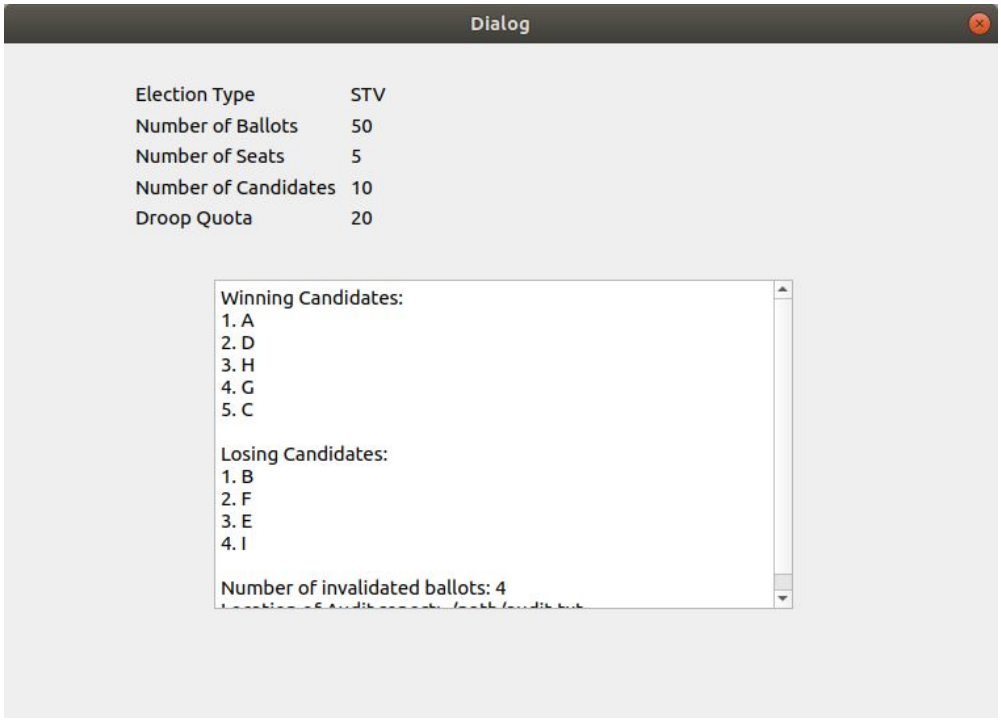
Startup/election configuration screen



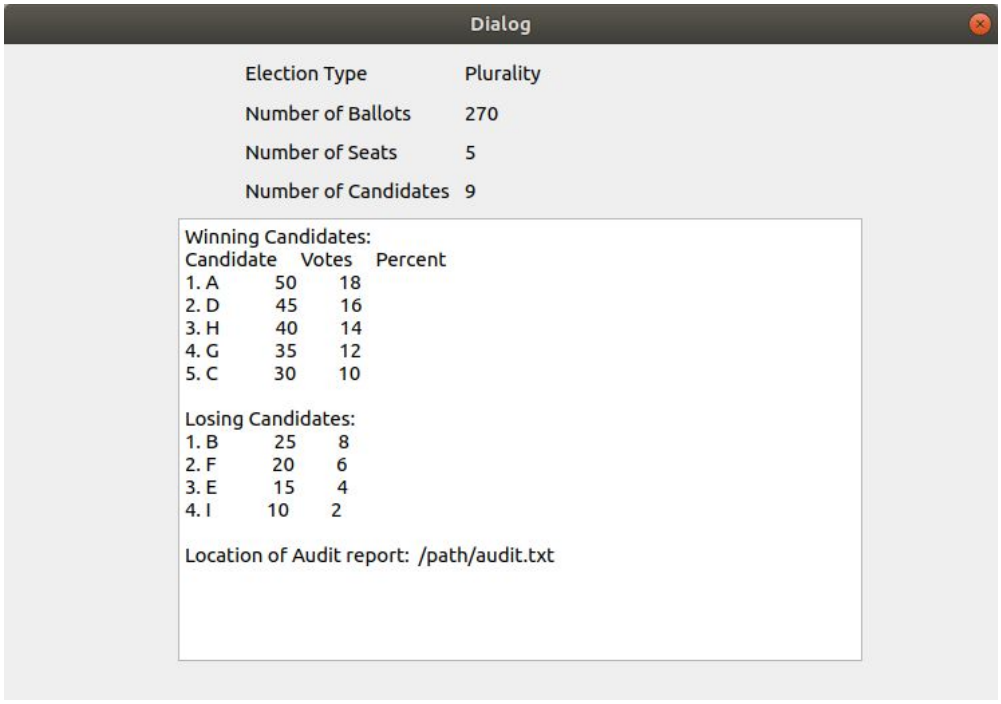
Screen while election is running



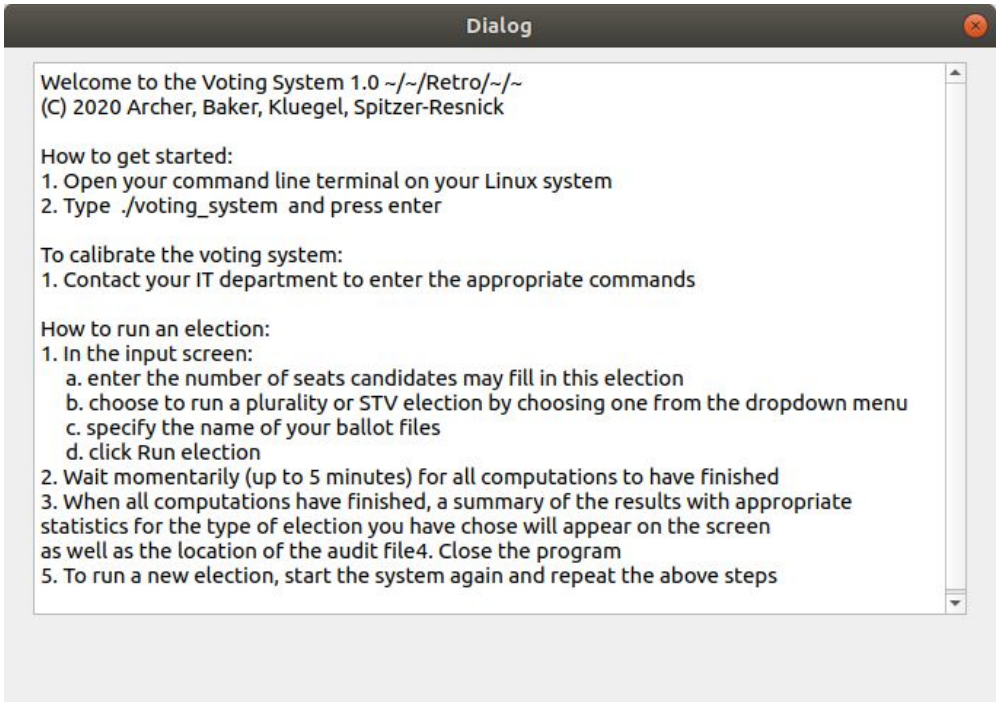
STV election results screen



Plurality election results screen



Help screen



6.3 Screen Objects and Actions

Startup/Information gathering window: The Startup/information gathering window will have the following screen objects:

- Dropdown menu to allow the user to select the election type, STV or plurality.
- Free form text area where the user enters the number of seats in the election.
- Larger free form text area where the user can list the names of the ballot files to use.
- “Run Election” button that when clicked will start running an election from the user given inputs.
- Election status field that will show when an election is not running(has not been started), is running, or is complete.
- Menu bar that will allow the user to access the help menu.

Election results window: The election election results window will have the following screen objects:

- Fields listing information about the election such as Election type, number of ballots, number of seats, number of candidates, and droop quota.
- Non-editable text field that will display the results of the elections.
- Menu bar that will allow the user to access the help menu.

Help menu: The help menu will have the following screen objects:

- Non-editable text field that will display the help text.
- Menu bar with an exit option to allow the user to exit the window and return to the main program.

7. REQUIREMENTS MATRIX

The following is the list of use cases from the SRS document with details on how they are handled by the Voting System program:

Use Case	UC_1 Input required voting information
Actors	Election Officials, Ballot handling system
Description	An election official enters the information necessary to run the election. This information includes election type (STV or plurality), number of seats, and the ballot files.
Data	Election type(STV or plurality), number of seats, list of ballot files.
Stimulus	User starts the Voting System program.
Response	Voting System is ready to run STV or plurality election.
Comments	Number of seats entered should be a positive integer.

Use Case	UC_2 Run STV election
Actors	Election Officials, Single transferable voting (STV) system
Description	All required election information has been entered and an election official now wishes to run an STV election.
Data	From the input data Candidate and ballot objects are created to run the election.
Stimulus	User clicks ‘Run election’ button.
Response	Ballots are processed by the STV election algorithm.
Comments	Results will be displayed when algorithm is done running.

Use Case	UC_3 Run plurality election
Actors	Election Officials, Plurality voting system
Description	All required election information has been entered and an election official now wishes to run a plurality election.
Data	From the input data Candidate and ballot objects are created to run the election.
Stimulus	User clicks ‘Run election’ button.
Response	Ballots are processed by the plurality election algorithm.
Comments	Results will be displayed when the algorithm is done running.

Use Case	UC_4 Run test files(s)
Actors	Developers/testers, Diagnostic system
Description	A developer/tester may wish to run test files through the voting system either for calibration purposes or to perform unit testing.
Data	Election type (STV or plurality), number of seats, list of ballot files (passed in through command line args). The ballot files are used to create candidate and ballot objects to run the election with.
Stimulus	User starts the Voting System program with command line arguments.
Response	Voting System runs an election based on command line arguments and displays the result.
Comments	None.

Use Case	UC_5 Turn off ballot shuffle
Actors	Developers/testers, Diagnostic system
Description	A developer or tester may wish to check that the STV election is being run correctly. By disabling the ballot shuffle the results of the STV election will be predictable.
Data	boolean passed in to disable shuffle
Stimulus	User starts the Voting System program with a command line argument.
Response	Voting system runs as it normally would but ballots will not be shuffled for the STV election.
Comments	Ballot shuffle cannot be turned back on, program needs to be restarted to turn shuffle back on.

Use Case	UC_6 Display election results
Actors	Election Officials/testers/developers, Plurality voting system or Single transferable voting (STV) system
Description	After an election has been run the results of the election will be displayed. The election results are encoded into a String format and printed to the screen.
Data	Election type (STV or plurality), number of seats, number of candidates, number of ballots, winners list, losers list, droop quota (STV only).
Stimulus	The STV or plurality algorithm has completed.
Response	The results are displayed.
Comments	None.

Use Case	UC_7 Display help window
Actors	Election Officials/developers/testers, Help/display system.
Description	A user of the Voting System wants more information about how to run the Voting System.
Data	File object loads in text of help file.
Stimulus	Users click the ‘help’ button on the menu bar.
Response	Help document displayed.
Comments	Help window can be displayed while the user is running the election.

Use Case	UC_8 Create election audit report
Actors	Election Officials/developers/testers, Audit reporting system
Description	An audit file needs to be produced for each election. The report shows how the election progressed (how ballots were assigned to candidates). This report is not returned to the screen, it is saved to a text file.
Data	File object created to save audit file.
Stimulus	When the election is started the audit file will be created.
Response	The audit file is updated while the election is running.
Comments	None.

8. APPENDICES

“Software Requirements Specification for Voting System”, Archer, Baker, Kluegel, and Spitzer-Resnick, Feb 21, 2020: https://github.umn.edu/umn-csci-5801-002-s20/repo-Team3/blob/master/SRS/SRS_Team3.pdf

“UseCases_Team3”, Archer, Baker, Kluegel, and Spitzer-Resnick, Feb 21, 2020: https://github.umn.edu/umn-csci-5801-002-s20/repo-Team3/blob/master/SRS/UseCases_Team3.pdf

“ClassDiagram_Team3”, Archer, Baker, Kluegel, and Spitzer-Resnick, Mar 20, 2020: https://github.umn.edu/umn-csci-5801-002-s20/repo-Team3/blob/master/SDD/ClassDiagram_Team3.pdf

“SequenceDiagram_STV_Team3”, Archer, Baker, Kluegel, and Spitzer-Resnick, Mar 20, 2020: https://github.umn.edu/umn-csci-5801-002-s20/repo-Team3/blob/master/SDD/SequenceDiagram_STV_Team3.pdf

“ActivityDiagram_Plurality_Team3”, Archer, Baker, Kluegel, and Spitzer-Resnick, Mar 20, 2020: https://github.umn.edu/umn-csci-5801-002-s20/repo-Team3/blob/master/SDD/ActivityDiagram_Plurality_Team3.pdf

“ActivityDiagram_STV_Team3”, Archer, Baker, Kluegel, and Spitzer-Resnick, Mar 20, 2020: https://github.umn.edu/umn-csci-5801-002-s20/repo-Team3/blob/master/SDD/ActivityDiagram_STV_Team3.pdf