CSCI5801 Team5

Voting System

Software Design Document

Name (s): Pengyin Chen, Jian Wang, Shuai Hao, Junren Huang

Lab Section: 002

Workstation: CSE-LAB-4250

Date: (03/10/2020)

TABLE OF CONTENTS

1. INTRODUCTION 2

1.1 Purpose 2

1.2 Scope 2

1.3 Overview 2

1.4 Reference Material 2

2. SYSTEM OVERVIEW 2

3. SYSTEM ARCHITECTURE 3

3.1 Architectural Design 3

3.2 Decomposition Description 3

3.3 Design Rationale 14

4. DATA DESIGN 14

4.1 Data Description 14

4.2 Data Dictionary 14

5. COMPONENT DESIGN 14

6. HUMAN INTERFACE DESIGN 19

6.1 Overview of User Interface 19

6.2 Screen Images 19

6.3 Screen Objects and Actions 22

7. REQUIREMENTS MATRIX 22

8. APPENDICES 22

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the Voting System. The expected audience is electoral officials for all conditions. This voting system is suitable for plurality and droop quota voting.

1.2 Scope

Voting System is a tool that users can use to input information, run desired election algorithms with the shuffle option, and produce audit reports. Users can use it to manage voting information and interact with them. They can produce reports with statistical information of each candidate, testing and so on so that it could improve management efficiency and achieve fairness of the campaign process.

1.3 Overview

Section 2 provides a basic description of the functionality, context and design of the system. This provides a detailed explanation of how systems process in different modules. Section 3 is Architectural Design. In this section, we provide a detailed explanation of all functions included in the system. Each function has an abstract description that demonstrates what the function looks like. Section 4 is Data Design. This section shows how the function reads data and stores it. Section 5 is Component Design. In this function, we will illustrate each component with pseudocode. Section 6 is Interface Design. Mainly this section is discussing how to improve user experience of using this system. Section 7 provides a cross reference that traces components and data structures to the requirements in the SRS document.

1.4 Reference Material

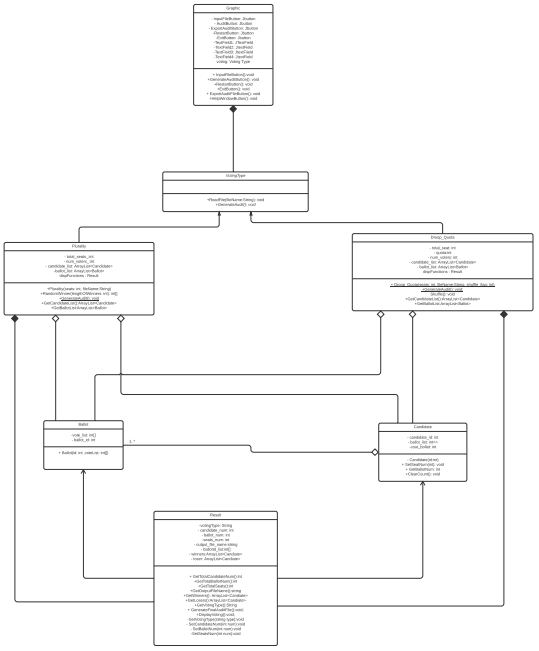
IEEE Standard for Information Technology--Systems Design--Software Design Descriptions - Redline," in IEEE Std 1016-2009 (Revision of IEEE Std 1016-1998) - Redline, vol., no., pp.1-58, 20 July 2009

2. SYSTEM OVERVIEW

The voting system was developed to ensure the fairness and correctness of the voting count. Mainly the voting system is separate from the actual voting. Therefore, all voting data is collected before the system runs. The function of this system is to simulate the counting process and provide interface to users.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design



The voting system is divided into several subsystems, including ballots, candidates, voting type (Plurality & Droop Quota), result and graph. The ballots and candidate subsystem are the data structure to store and keep track of the information of each voter and candidates. Voting type could achieve two different algorithms in the voting system. Result system could catch the details of the voting process and display necessary output to users. Graphic provides users an interface so that they could easily input information and get corresponding reports. Ballots and candidates are related with the voting type subsystem so that different algorithms could access data and generate results. Voting type subsystem is connected with result subsystem and graphic system. The graphic subsystem could interface with different algorithms including input information and display result. Result system could catch the result from the voting type subsystem and deliver to the graphic subsystem to achieve the interface.

3.2 Decomposition Description

Ballot
Name: Ballot
Type: Data Structure
Description: This is a data structure for ballots so that the algorithm and result subsystem could access the needed data and generate reports for users

Attributes: None
Resources: None
Operations:
Name: Ballot ()
Argument: id, vote list
Return: void
Precondition: all necessary information to create a ballot (ballot ID and vote list)
Postcondition: a ballot object generated
Exception: None
Flow of Event:
1. Algorithm reads files
2. generate ballots based on content of the file

Candidate
Name: Candidate
Type: Data Structure
Description: This is a data structure for candidate so that the algorithm and result subsystem could access the needed data and generate reports for users
Attributes: None
Resources: None
Operations:
Name: Candidate ()
Argument: id
Return: void
Precondition: voting system has read the file and there exists necessary info for candidates
Postcondition: a candidate object generated
Exception: None
Flow of Event:
1. Algorithm reads files
2. generate candidate based on content of the file

Voting type
Name: Voting Type
Type: Abstract Class
Description: This is an abstract class for plurality and droop quota. Thus, the two algorithms could inherit this abstract class and get shared methods.
Attributes: None
Resources: None
Operations:
Name: ReadFile()
Argument: fileName
Return: void
Precondition: A voting type object (either plurality or droop quota has been created)
Postcondition: file content has been read and corresponding objects are created
Exception: None

Flow of Event:
1. Create voting type (plurality, droop quota) objects
2. call ReadFile method
3. read file content and generate objects for storing data

Name: GenerateAudit()
Argument: None
Return: void
Precondition: ReadFile method has been called and necessary lists are initialized by ReadFile.
Postcondition: report generate and display corresponding details.
Exception: None

Flow of Event:
1. Create voting type (plurality, droop quota) objects
2. call Readfile method
3. read file content and generate objects for storing data
4. call GenerateAudit()
5. details displayed to users
6. audit file generated

Plurality
Name: Plurality
Type: Voting type
Description: This is a child class of voting type. It inherits ReadFile() and GenerateAudit() functions so that it could achieve the basic functionality. Also, it adds a random winner select method for the case that could happen in plurality that several candidates has the same amount of ballots.
Attributes: None
Resourses: None
Operations:
Name: ReadFile()
Argument: fileName
Return: void
Precondition: A voting type object (either plurality or droop quota has been created)
Postcondition: file content has been read and corresponding objects are created
Exception: None
Flow of Event:
1. Create voting type (plurality, droop quota) objects
2. call ReadFile method
3. read file content and generate objects for storing data

Name: GenerateAudit()
Argument: None

Return: void
Precondition: ReadFile method has been called and necessary lists are initialized by ReadFile.
Postcondition: report generate and display corresponding details.
Exception: None

Flow of Event:
1. Create voting type (plurality, droop quota) objects
2. call readfile method
3. read file content and generate objects for storing data
4. call GenerateAudit()
5. details displayed to users
6. audit file generated

Name: RandomWinner()
Argument: candidate length
Return: int list with the order of each candidate
Precondition: several candidates have the same amount of ballot
Postcondition: the order list of candidates generated
Exception: None

Flow of Event:
1. after running algorithm, several candidates have the same amount of ballot
2. run random Winner to decide the order for each candidate
3. pick any candidates as we need.

Name: GetAllBallots()
Argument: None
Return: a list of ballots with their ID
Precondition: algorithm starts
Postcondition: ballots returned
Exception: None

Flow of Event:
1. run algorithm
2. access ballots list
3. return and display

Name: GetAllCandidates( )
Argument: None
Return: a list of candidates with their ID
Precondition: algorithm starts
Postcondition: candidates list returned
Exception: None

Flow of Event:
1. run algorithm
2. read file content
3. create ballots and candidates objects and lists
2. access candidates list

Droop_Quota
Name: Droop_Quota
Type: Voting type
Description: This is a child class of voting type. It inherits ReadFile and GenerateAudit() functions so that it could achieve the basic functionality. Also, it adds a shuffle method to shuffle ballots before running the algorithm.
Attributes: None
Resourses: None
Operations:
Name: ReadFile()
Argument: fileName
Return: void
Precondition: A voting type object (either plurality or droop quota has been created)
Postcondition: file content has been read and corresponding objects are created
Exception: None
Flow of Event:
1. Create voting type (plurality, droop quota) objects
2. call ReadFile method
3. read file content and generate objects for storing data

Name: GenerateAudit()
Argument: None
Return: void
Precondition: ReadFile method has been called and necessary lists are initialized by ReadFile.
Postcondition: report generate and display corresponding details.
Exception: None
Flow of Event:
1. Create voting type (plurality, droop quota) objects
2. call readfile method
3. read file content and generate objects for storing data
4. call GenerateAudit()
5. details displayed to users
6. audit file generated

Name: Shuffle ()
Argument: none
Return: void
Precondition: there exists a ballot list
Postcondition: ballot list is shuffled
Exception: None

Flow of Event:
1. User choose droop quota algorithm
2. Create droop quota objects
2. call readfile method
3. read file content and generate objects for storing data
4. shuffle ballots list

Name: GetAllBallots()
Argument: None
Return: a list of ballots with their ID
Precondition: algorithm starts
Postcondition: ballots returned
Exception: None
Flow of Event:
1. run algorithm
2. access ballots list
3. return and display

Name: GetAllCandidates( )
Argument: None
Return: a list of candidates with their ID
Precondition: algorithm starts
Postcondition: candidates list returned
Exception: None
Flow of Event:
1. run algorithm
2. read file content
3. create ballots and candidates objects and lists
2. access candidates list

Result
Name: Result
Type: report generator
Description: This is report generator class for algorithms. As the algorithm is running, it could access data stored in them and display details of the voting process.
Attributes: None
Resourses: None
Operations:
Name: GetTotalCandidateNum( )
Argument: None
Return: total candidate number (int)
Precondition: candidate list created
Postcondition: total candidate number returned
Exception: None
Flow of Event:

1. run algorithm
2. read file content
3 build ballots and candidates objects and lists
4. access total candidate number as we need to display information

Name: GetTotalBallotNum( )
Argument: None
Return: total ballot number (int)

Precondition: ballot list created
Postcondition: total ballot number returned
Exception: None

Flow of Event:
1. run algorithm
2. read file content
3 build ballots and candidates objects and lists
4. access total ballots number as we need to display information

Name: GetTotalSeats ()
Argument: None
Return: total seats number
Precondition: user has input seats and run the algorithm
Postcondition: seats number returned
Exception: None

Flow of Event:
1. run algorithm
2. read file content
3 build ballots and candidates objects and lists
4. access total seats number as we need to display information

Name: GetOutputFileName()
Argument: None
Return: output file name
Precondition: output file generated
Postcondition: return output file name
Exception: None

Flow of Event:
1. run algorithm
2. audit file generated and stored in a location
2. access output file name

Name: GetWinners()
Argument: None
Return: a list of winners with their ID
Precondition: algorithm starts
Postcondition: winner returned

Exception: None
Flow of Event:
1. run algorithm
2. access winners and display it to the users as we need

Name: GetLosers()
Argument: None
Return: a list of losers with their ID
Precondition: algorithm starts
Postcondition: losers returned
Exception: None
Flow of Event:
1. run algorithm
2. access losers and display it to the users as we need

Name: GetVotingType( )
Argument: None
Return: current voting type as a string
Precondition: user input algorithms to use
Postcondition: voting type returned
Exception: None
Flow of Event:
1. user input algorithm
2. access voting type

Name: GenerateFinalAuditFile( )
Argument: None
Return: void
Precondition: algorithm finished
Postcondition: final audit file generated
Exception: None
Flow of Event:
1. user input information
2. run algorithm
3. algorithm finished and get result
4. Generate final audit file

Name: DisplayVoting( )
Argument: None
Return: void
Precondition: algorithm ran
Postcondition: display detail to users
Exception: None
Flow of Event:

1. run algorithm
2. in plurality algorithm, display the final result
2. in droop quota algorithm, for each round of the voting, display details

Name: SetVotingType( )
Argument: String
Return: void
Precondition: None
Postcondition: voting type confirmed
Exception: None

Flow of Event:
1. user input voting type
2. set voting type for display

Name: SetCandidateNum( )
Argument: int
Return: void
Precondition: None
Postcondition: candidate number confirmed
Exception: None

Flow of Event:
1. algorithm reads file information
2. based on the content, set candidate number

Name: SetBalletNum( )
Argument: int
Return: void
Precondition: None
Postcondition: ballot number set
Exception: None

Flow of Event:
1. algorithm reads file information
2. based on the content, set ballot number

Name: SetSeatsNum( )
Argument: None
Return: void
Precondition: None
Postcondition: seats number confirmed
Exception: None

Flow of Event:
1. user input the seats number
2. set the seats number for display

Graphic
Name: Graphic
Type: User interface
Description: This is a user interface class for the voting process. As the algorithm is running, it passes data to the algorithm and receives data from the result class for display.
Attributes: None
Resourses: None
Operations:
Name: InputFileButton( )
Argument: None
Return: void
Precondition: File name inputted
Postcondition: a file has been inport
Exception: None
Flow of Event:
1. user input file name
2. import the file
3. display “IMPORT FILE SUCCESS”

Name: GenerateAuditButton( )
Argument: None
Return: void
Precondition: File Imported
Postcondition: result generated, detail displayed, final audit file location shown
Exception: None
Flow of Event:
1. user input file name, seats number, algorithm and shuffle option
2. click the GenerateAudit button
3. call algorithms to run with parameters
4. display each rounds detail if choose droop quota, display final result if choose plurality.
Name: RestartButton( )
Argument: None
Return: void
Precondition: None
Postcondition: clear all input, shut down the running program
Exception: None
Flow of Event:
1. user click the Restart button
2. terminate running program if its running, clear all input
3. clear display pane

Name: ExitButton( )
Argument: None
Return: void
Precondition: None

Postcondition: exit the system
Exception: None

Flow of Event:
1. user click the Exit button
2. Exit the program and close everything

Name: ExportAuditFileButton( )
Argument: None
Return: void
Precondition: algorithm finished and result created
Postcondition: audit file exported
Exception: None

Flow of Event:
1. user click the ExportAuditFile button
2. user could choose a location to save or choose default location export the audit file as user indicates

Name: HelpWindowButton( )
Argument: None
Return: void
Precondition: None
Postcondition: help window shown
Exception: None

Flow of Event:
1. user click the HelpWindow button
2. help window pop up

3.3 Design Rationale

This architecture is built based on the logical flow of our voting system. First, it begins with reading files and storing ballots and candidate's data. Then, algorithms will process data and generate reports. During the process, we also need to interface with users. By considering this architecture, we can easily access the result of each round and deliver it to the graphic class to achieve the interface.

4. DATA DESIGN

4.1 Data Description

All data of the ballot is collected and stored in multiple existing files. These files contain 2 kinds of information --- candidates and ballots. After the system has been executed, the system will read the input file and generate a data structure for each candidate and ballot. Then, the system will store them in an int list that each index corresponds with their IDs. There will be a list for ballots and a list for candidates. Also, for each candidate, we maintain a vote list for ballots this candidate receives.

4.2 Data Dictionary

Attribute Name	Attribute Type	Attribute Size
candidate id	int	10
candidate list	int[]	unknow
ballot id	int	10
ballot list	int[]	unknow
vote list	int[]	unknown

5. COMPONENT DESIGN

In Voting Type Class, we have two methods: Readfile and GenerateAudit(). ReadFile is already written for inheritance. GenerateFile is an virtual function to be override in different algorithms.

```
Void ReadFile(string fileName) {
        initialCandidateList();
        initialBallotList();
```

```
        initialWinnerList();

        while(getline() ) {
                if(it is a ballot)
                        createBallot();
                        add to the ballot list;
                if (it is a candidate)
                        createCandidate();
                        add to the ballot list;
        }
}
```

In Plurality class:

```
void GenerateAudit() {
        For each ballot in the ballot list
                assign this ballot to his desire candidate
        sort the candidate list by the ballot count
        add candidate to the winner list until the seats is all filled
        DisplayVote();
        GenerateAuditFiles();
}

int[] RandomWinner(length: int){
        result = new int[length];
        for (i =0 to length-1){
                val= rand(1, length);
                if(result doesnt contain val)
                        result[i] = val;
        }
        return result;
}

GetAllBallots() {
        return ballotsList;
}

GetAllCandidates() {
```

```
        return candidatesList;
}
```

In Droop Quota class, we have GenerateAudit(), Shuffle() GetBallot(), and GetCandidates()

```
void GenerateAudit() {
        Calculate the droop number
        For each ballot in the ballot list
                assign this ballot to his first choice
            sort the candidate list by the ballot count
            for each candidate in the candidate list
                if his ballot count is more than the droop number
                    add to the winner list
                add the others to the loser list
            eliminate the candidate with the least candidate in the loser list
            DisplayVote();

            while(the seat is not filled)
                for each candidate in the loser list
                    for each ballot in this candidate's ballot list
                        choose the next available candidate with higher desire
                sort the loser list with the ballot count
                for candidate in the loser list
                if (it has more ballots than the droop number)
                add candidate to the winner list
                DisplayVote();
        GenerateAuditFiles();
}

Shuffle(){
        random(ballot list);
}

GetAllBallots() {
        return ballotsList;
```

```
}

GetAllCandidates() {
        return candidatesList;
}
```

In the Result Class:

```
GetTotalCandidateNum() {
        return candidateList.size();
}

GetTotalBallotNum() {
        return ballotList.size();
}

GetTotalSeats() {
        return seats.size();
}

GetOutPutFileName() {
        return filename;
}

GetWinners() {
        return winnerList;
}

GetLosers() {
        return losersList;
}

GetVoltingType() {
        return this.votingType;
}

GenerateFinalAuditFile() {
        auditFile.close();
}
DisplayVoting() {
        print(voting type\n number of seats \n number of candidates \n number of balltos\n)
        if(in plurality algorithm)
                for each candidate
                        calculate the percentage of balltos
                        for each candidate in the winner list{
```

```
                        print(candidate info)
                        print(candidate_ballot_count);
                        print (percentage)
                    }

        if(in droop quota algorithm)
            print (winner list)
            for each candiate in the winner list
                print(candidate info)
                print(candidate_ballot_count);
                print (percentage)
            }

            print (loser list)
}

SetVotingType(String v) {
        this.votingType = v;
}
SetCandidateNum(int n) {
        this.candidateNum = n;
}
SetBallotNum(int n) {
        this.ballotNum = n;
}
SetSeatsNum(int n ) {
        this.seatsNum = n;
}
```

In Graphic Class:

```
Graphic:
InputFileButton() {
        open(inputFile);
        readFile(intputFile);
        display("successful import" )
}

GenerateAuditButton() {
        generateAudit();
}
restartButton() {
        stop(generateAudit);
        clear(candidateList);
        clear(BallotsList);
```

```
        clear(seatsList);
        clear(winners)
        clear(losers)
}

HelpWindowButton(){
        display(help window info);
}
```

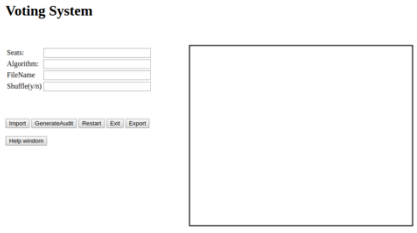
6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The user interface of this system will help users better understand and use the system. To use the system, users have to input the information first. At the left side of the interface, there are four types of information users have to input. First one is the number of candidates who will win this voting, second one is which algorithm user wants to use, third one is the filename that contains all the data and forth one is to determine shuffle or no shuffle. The buttons below are different operations users can choose. Input button use to input the file to the system, if the file is successfully imported, correct information will be displayed at the information window (right side the interface). After the user presses the GenerateAudit button, the system will begin to run the algorithm and display the information for each round. System will be restarted by pressing the Restart button. The user can exit the system by pressing the Exit button. If the user wants to generate the file to store the information, the user can press the Export button. The last button is the help window, if the user is confused about the system, the user can click the help window button to get more information.

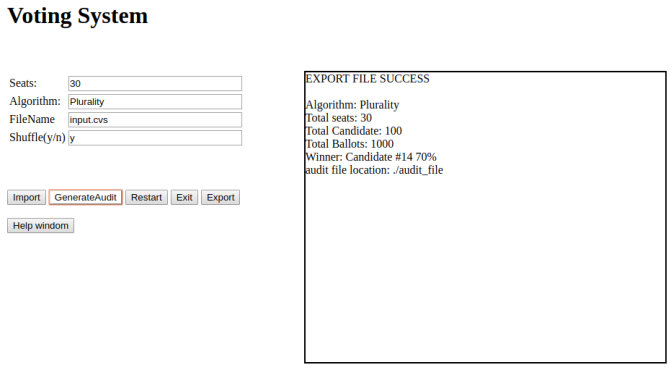
6.2 Screen Images

Voting System Overview

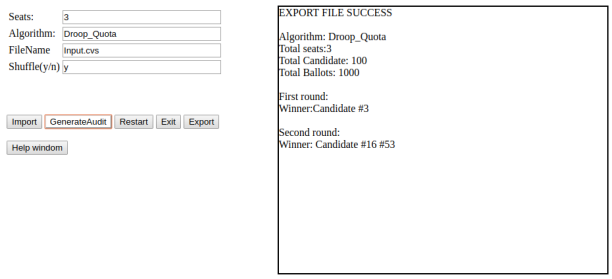


Input File

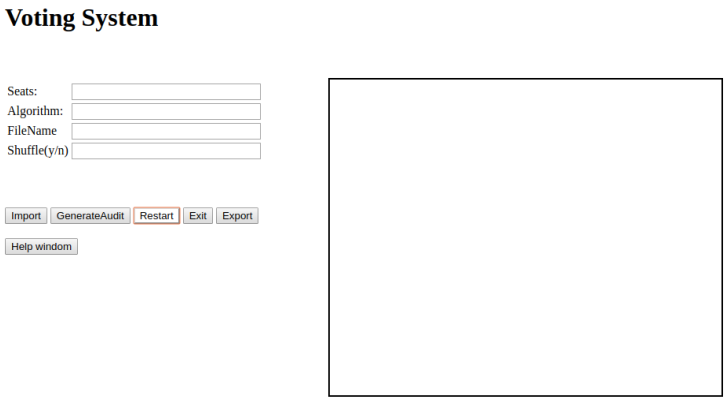
Choose Plurality as the algorithm and press Generate Audit Button



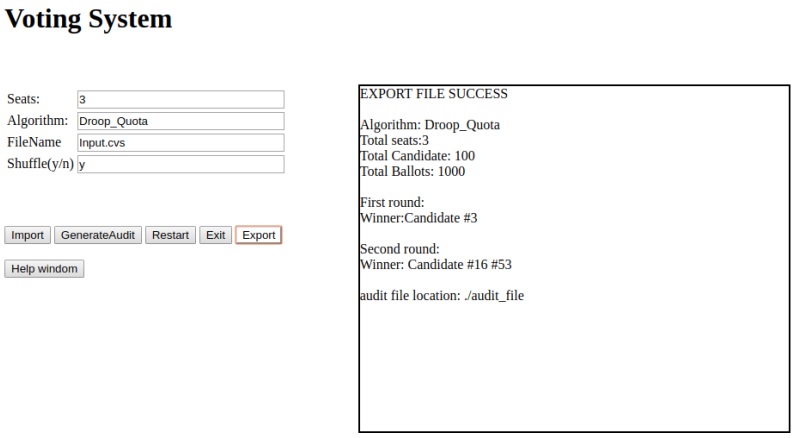
Choose Droop_Quota as the algorithm:



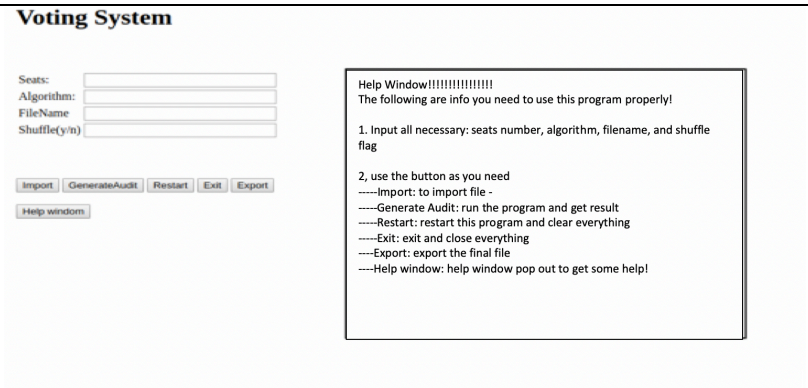
Restart the program



Export the result file



Help window



6.3 Screen Objects and Actions

There are three types of objects on the screen, and they are input object, button object and display objects. input object allows the user to input the information that is required by the system. After the user input all the information that is required and press the button object, the system will take actions based on the input information. The result will be displayed on the right side of the window.

.

7. REQUIREMENTS MATRIX

User input seats and choose algorithm	In Graphic, Textfield1: JTextField, TextField2: JTextField
User input filename	In Graphic, Textfield3: JTextField
Shuffle/Shuffle off	In Graphic, Textfield4: JTextField
System generate audit report	In Graphic, GenerateAuditFileButton (): Jbutton
User uses help windows	In Graphic, HelpWindowButton (): Jbutton
Run Plurality Algorithm	In Graphic, TextField2: JTextField, GenerateAuditFileButton (): Jbutton
Run STV Algorithm	In Graphic, TextField2: JTextField, GenerateAuditFileButton (): Jbutton
Display election result on the UI	In Graphic, GenerateAuditFileButton (): Jbutton

8. APPENDICES

None