

Team 25

Party Voting Systems

Software Design Document

Names: Michael Ung
x500s: ungxx028

YongFeng Ji
ji000017

Jing Wu
wu000163

Zhouran Bi
bi000001

Date: 10/31/2019

TABLE OF CONTENTS

1. INTRODUCTION	1
1.1 Purpose	1
1.2 Scope	1
1.3 Overview	2
1.4 Reference Material	2
1.5 Definitions and Acronyms	2
2. SYSTEM OVERVIEW	2
3. SYSTEM ARCHITECTURE	6
3.1 Architectural Design	6
3.2 Decomposition Description	8
3.3 Design Rationale	11
4. DATA DESIGN	12
4.1 Data Description	12
4.2 Data Dictionary	13
5. COMPONENT DESIGN	14
6. HUMAN INTERFACE DESIGN	19
6.1 Overview of User Interface	19
6.2 Screen Images	25
6.3 Screen Objects and Actions	27
7. REQUIREMENTS MATRIX	28
8. APPENDICES	28

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to provide a clear description of the internals of the voting system and a path to how to create the application.

This document is intended for the programmers of the voting system, Team 25, which consists of YongFeng Ji, Jing Wu, Zhouran Bi, and Michael Ung.

1.2 Scope

The ultimate goal of the software is to determine which candidates are the winners in a given election and to generate an audit file that corresponds to the resulting data. A user will be able to input a properly formatted file and process the votes for the input data.

By creating this piece of software, we try to create a way to quickly and accurately analyze the results of an election, lessening the need to manually inspect voting results.

1.3 Overview

The remaining chapters and their contents are listed below:

Section 2 concerns the system overview, explain how the system looks like

Section 3 concerns the system architecture, it specifies how each design entities combined together to perform all the functions included in the system. In addition, this section provides comprehensive descriptions for each design entity, and its design rationale.

Section 4 concerns the data design

Section 5 concerns the component design

Section 6 concerns the human interface design, it comprises of user interface, and screen images, and screen objects and actions. It would give information about how the interface looks.

1.4 Reference Material

No references were used for planning tests.

1.5 Definitions and Acronyms

No additional definitions and acronyms are needed to interpret this document.

2. SYSTEM OVERVIEW

Background information:

Two different Voting Systems: OPL (Open Party List) and CPL (Closed Party List).

OPL is developed for voting for the candidate, CPL is developed for voting parties. It is like the calculation program, given the format of voting information, then calculate the final result of voting based on the type of voting system, and it provides two functions: create an audit file that comprises all original information plus the result of voting, and create voting result text file that only include the voting result based on the type of voting system, for example, after executing command line of creating voting result text file in CPL voting system, the winning party would be written in the voting result text file.

This software contains several functions, as listed below:

- Specify file name: Type the file path. System will check if the file exists or not. If not, it would return an error. If it exists, systems will be prepared to open it. Users would use it when they need to search for a file.
- Run Voting System for OPL: System would read the input OPL file, and process all the data in that file and save some data into local database. Users would use it when they need to generate the result from OPL by using the system.
- Run Voting System for CPL: System would read the input CPL file, and process all the data in that file and save some data into local database. Users would use it when they need to generate the result from CPL by using the system.
- Create Audit file for OPL: After running the OPL system, all the OPL information will be generated in a text file that will be downloaded into a local disk. Users would use it when they need to see an Audit file for OPL.
- Create Audit file for CPL: After running the CPL system, all the CPL information will be generated in a text file that will be downloaded into a local disk. Users would use it when they need to see an Audit file for CPL.
- Display voting results: display the voting results of the election onto the user's screen, which include winner, and how many votes they obtained and how many votes the party obtained, and the total amount of votes cast. Users would use it when they need to see the results, or check the results on the screen.
- Create voting result text file for OPL: It will create a text file that comprise the result of voting for OPL. Users would use it when they need to have a text file that contains the result of voting for OPL.

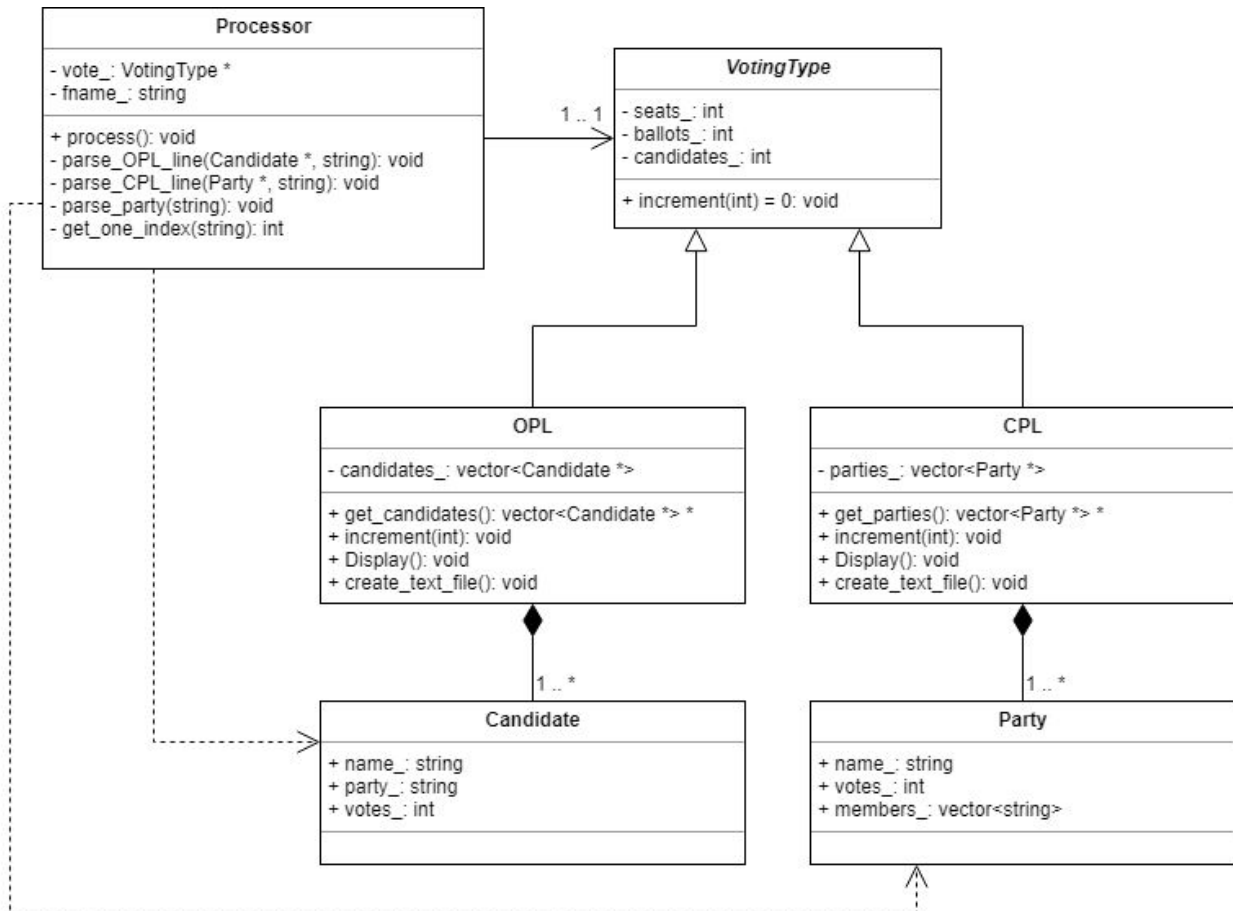
- Create voting result text file for CPL: It will create a text file that comprise the result of voting for CPL. Users would use it when they need to have a text file that contains the result of voting for CPL.

The System might have some error cases:

- Cannot locate file with the filename.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design



The pipe and filter architecture was selected to model the voting system. Data is read in through the file, which serves as the pipe, filtered through the Processor, and stored in our data structures.

In the UML diagram above, a Processor object will be created by a driver and read in a file name. The Processor object will then call the `process()` function to analyze the data. An abstract class was created to allow for polymorphism, so the Processor can run on different types of voting methods. The `process()` function will also create the audit file required procedurally.

3.2 Decomposition Description

Party

Name: Party

Type: Class

Descriptions: It is a data structure storing the information about a party.

Attributes:

 Name: string

 Votes: Int

 Members: Vector<string>

Operations: None

Candidate

Name: Candidate

Type: Class

Descriptions: It is a data structure storing the information about a candidate

Attributes:

 Name: String

 Party: String

 Votes: Int

Operations: None

Voting Type

Name: VotingType

Type: Class

Descriptions: It is an abstract class which has derived types corresponding to two different types of voting.

Attributes:

 Seats: Int

 Ballot: Int

 Candidate: Int

Operations:

Name: Increment(int)

Arguments: Int

Returns: No return value

Pre-condition: File are opened, party and candidate must be written already

Post-condition: number will be increased depends on each party or each candidate

Exceptions: File error

Flow of Events:

1. Party and candidate are written
2. Systems use this function
3. Systems get the numbers

OPL

Name: OPL

Type: Class

Descriptions: One of the voting types, calculate the voting for candidates.

Attributes:

Candidates_: vector<Candidate*>*

Operations:

Name: get_candidates()

Arguments: None

Returns: vector of all candidates

Pre-condition: System determines the OPL

Post-condition: vector of all candidates will be obtained

Exceptions: None

Flow of Events:

1. Systems determines OPL
2. Systems uses this function
3. Systems obtain vector of all candidates

Name: increment(int)

Arguments: Int

Returns: No value returned

Pre-condition: System determine the OPL

Post-condition: number will be increased for candidates

Exceptions: File error

Flow of Events:

1. Systems determines the OPL
2. Systems uses this function
3. Systems get the numbers of voting for each candidates

Name: Display()

Arguments: None

Returns: None

Pre-condition: all results calculated

Post-condition: An interface will be shown

Exceptions: No results given

Flow of Events:

1. System determines the OPL
2. Users Clicks display
3. Interface shown

Name: create_text_file()

Arguments: None

Returns: None

Pre-condition: System has finished processing data

Post-condition: Text file will be created of the results

Exceptions: None

Flow of Events:

User clicks button to create text file

Function is called

Text file is created and written to

CPL

Name: CPL

Type: Class

Descriptions: One of the voting types, calculate the voting for Party.

Attributes:

Parties_: vector<Party*>*

Operations:

Name: get_parties()

Arguments: None

Returns: vector of all parties

Pre-condition: Systems determines the CPL

Post-condition: vector of all parties will be obtained

Exceptions: None

Flow of Events:

Systems determines the CPL

Systems uses this function

Systems obtain vector of all parties

Name: increment(int)

Arguments: Int

Returns: No value returned

Pre-condition: Systems determines the CPL

Post-condition: numbers of voting will be increased for parties

Exceptions: File error

Flow of Events:

1. Systems determines the CPL
2. Systems uses this function
3. Systems get the numbers of voting for party

Name: Display()

Arguments: None

Returns: None

Pre-condition: all results calculated

Post-condition: An interface will be shown

Exceptions: No results given

Flow of Events:

1. Systems determines the CPL

2. Users Clicks display
3. Interface shown

Name: create_text_file()

Arguments: None

Returns: None

Pre-condition: System has finished processing data

Post-condition: Text file will be created of the results

Exceptions: None

Flow of Events:

User clicks button to create text file

Function is called

Text file is created and written to

Processor:

Name: Processor

Type: Class

Description: Processor is designed to parsing the file uploaded by specifying the file type and process the content of the file to generate the voting results to the audit file.

Attributes:

vote_: VotingType*

fname_: string

Operations:

Name: process()

Argument: None

Returns: No value returned

Pre-condition: All the candidates or parties should be ready to count

Post-condition: Results would be calculated

Exceptions: No results given

Flow of Events:

1. Systems gather the data
2. Systems calculate the voting
3. Results generated

Name: parse_OPL_line(Candidate*, string)

Argument: Candidate*, string

Returns: No value returned

Pre-condition: System determines OPL

Post-condition: A new candidate will be created, and pushed to candidate vector

Exceptions: No results given

Flow of Events:

1. Systems determines OPL
2. Systems parse the candidate
3. New candidate created, and pushed to candidate vector

Name: parse_CPL_line(Party*, string)

Argument: Party*, string

Returns: No value returned

Pre-condition: System determines CPL

Post-condition: A new party will be created, and pushed to party vector

Exceptions: No results given

Flow of Events:

1. Systems determines OPL
2. Systems parse the candidate
3. New party created, and pushed to party vector

Name: parse_party(string)

Argument: string

Returns: No value returned

Pre-condition: System determines CPL

Post-condition: new party created, and pushed inside of party vector

Exceptions: No results given

Flow of Events:

1. Systems determines OPL
2. Systems parse the candidate
3. Candidate created

Name: get_one_index(string)

Argument: string

Returns: an integer of that index

Pre-condition: All candidates or parties must be written in.

Post-condition: System will get an integer of index

Exceptions: No results given

Flow of Events:

1. Systems execute the function
2. Systems get the index

3.3 Design Rationale

It was decided that the pipe and filter architecture will be used. The pipe and filter architecture provides a flow of events similar to what is needed to process votes. The pipe portion of the design parallels how the voting system or the driver will read in a file that contains data. The Processor class serves as a filter to reduce the amount of data in the end, which is stored in the data structures that are created.

The Model-View-Controller (MVC) model was also considered, but the pipe and filter design was better suited for the system. The MVC model is similar to how the system's

design is made, but there is one key difference between them. The flow of a MVC model is generally circular, whereas the flow of the pipe and filter design goes in one direction. Since we only had to process data and display it, it was decided that the MVC model did not suit our purposes. There was no need for additional views and a robust controller when the functionality needed is not rigorous. A tradeoff that was considered was that the MVC provides more extensibility in terms of what can be done to the data after it is processed, whereas in the pipe and filter model, once the data is done being processed, everything is complete.

4. DATA DESIGN

4.1 Data Description

Overall, we create 6 classes for this whole system, including Processor, VotingType, OPL, CPL, Candidate and Party.

The Processor class does most of the functionalities of the whole system, and the constructor will be called by the main function and cooperate with other classes. The Processor class will parse the input file specified by the user and store it in the OPL/CPL class, which derives from VotingType. We separate our VotingType class into two sub-class, one is OPL class, the other is CPL class, which both inherits from the VotingType and shares some variables.

For OPL, there is a Candidate class associated with OPL, which consists of information of each candidate, for example, the candidates' name, which party they are in and how many ballots they have gotten.

For CPL, there is a Party class associated with CPL. It is different from Candidate class because in CPL, people only vote for parties rather than individual candidates.

The Processor instance created in the main function will parse the input file to determine if it is OPL or CPL. Next, the Processor will keep reading the file and store information depending on which type of voting system is used. If it is OPL, then an OPL instance and Candidate instances will be generated to count the votes each candidate gets. If it is CPL, then a CPL instance and Party instances will be generated to count the votes each party gets. Finally, after the processor encounters the end of the file, the processor will display the results of the election which were stored in the data structures described above.

4.2 Data Dictionary

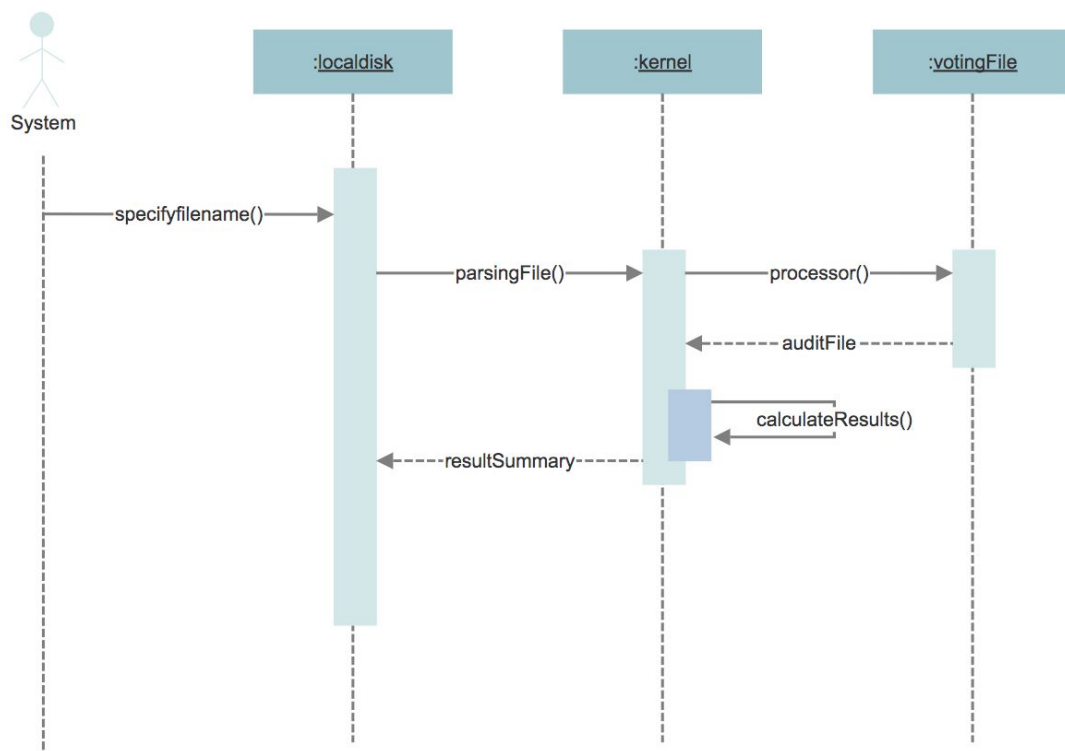
Class	Attribute Name	Method	Method parameters
Processor	int vote_ string fname_	void processor	void
		void parse_OPL_line	Candidate *, string
		void parse_CPL_line	Party *, string
		void parse_party	string
		int get_one_index	string
VotingType	int seats_ int ballots_ int candidates_	void increment	int
OPL	vector<Candidate*> * candidates_	vector<Candidate*> * get_candidates	void
		void increment	int
		void Display	void
		void create_txt_file	void
CPL	vector<Party*> * parties_	vector<Party*> * get_parties	void
		void increment	int
		void Display	void
		void create_txt_file	void
Candidate	string name_ string party_ int votes_	None	None

Party	string name_ string votes_ vector<string> members_	None	None
--------------	---	------	------

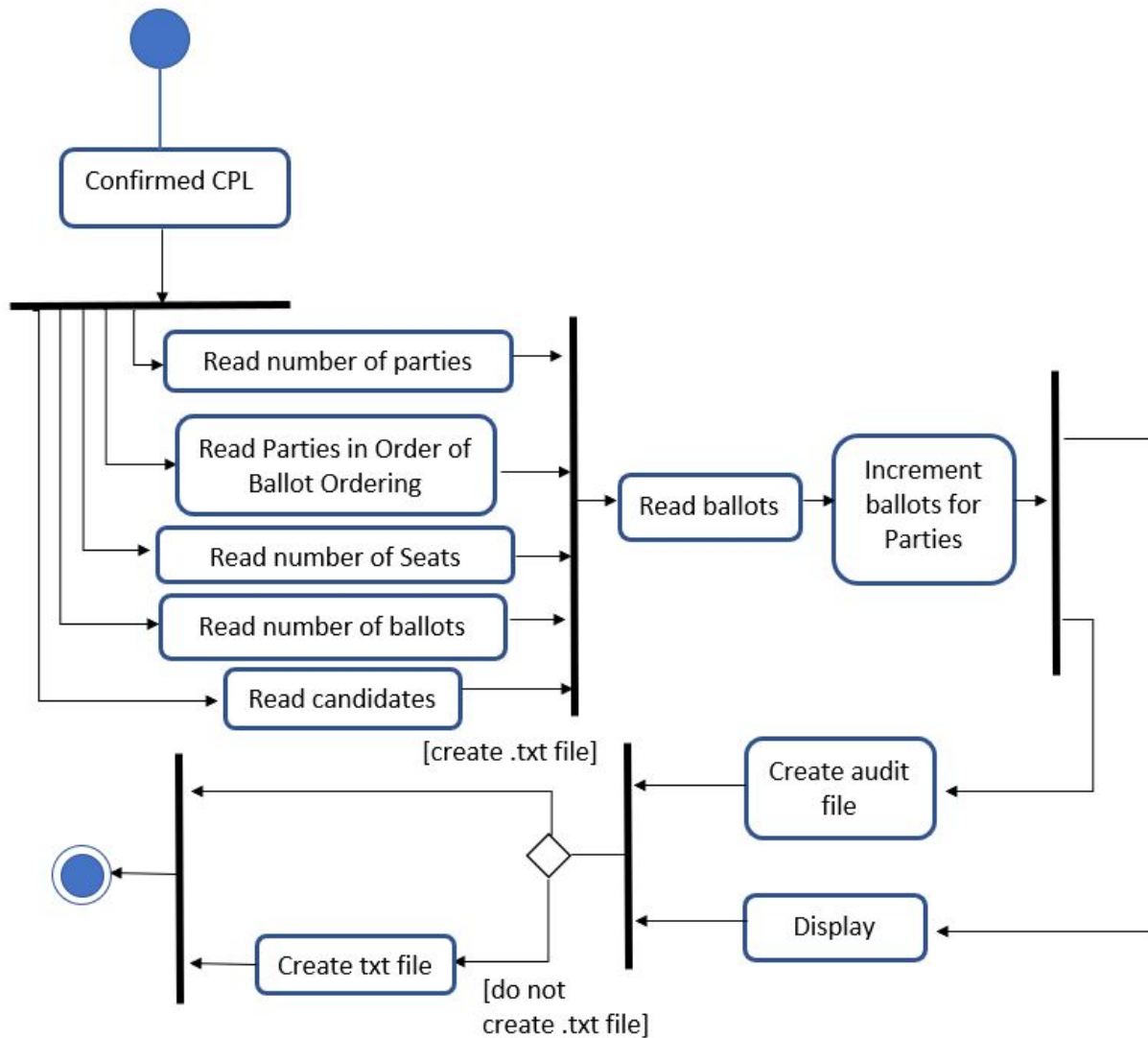
5. COMPONENT DESIGN

5.1 Sequence Diagram : Closed Party List Voting

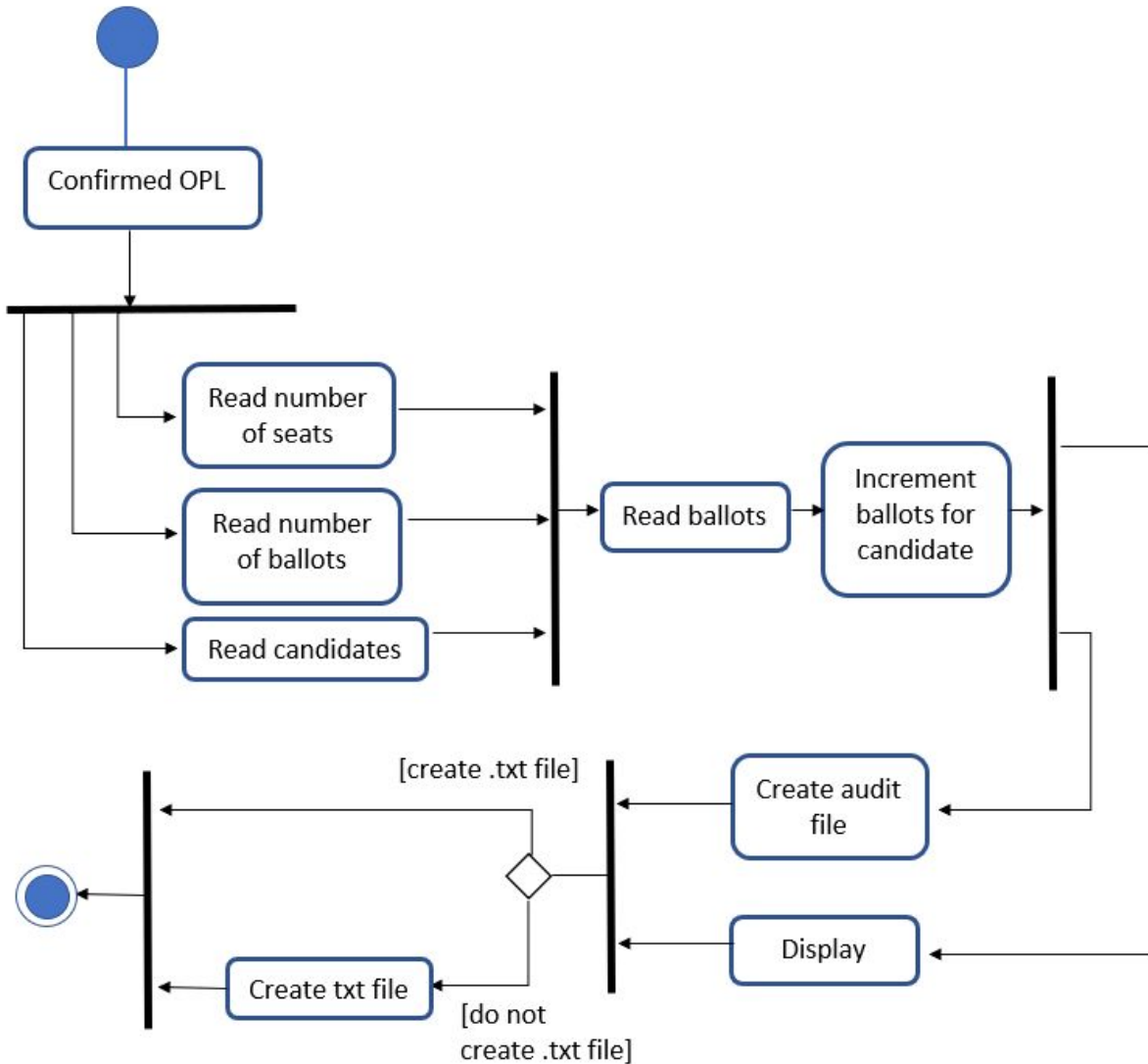
Sequence Diagram: Closed Party List Voting



5.2 Activity Diagram : Closed Party List Voting



5.3 Activity Diagram : Open Party List Voting



5.4 Pseudocode

1. Increment function for the party class


```
void increment(int index){
    party[index]->votes_++;
}
```
2. `vector<Party *> * get_parties() {`
 `return &parties_;`
 `}`
3. Increment function for the Candidate class


```
void increment(int index) {
```

```

    candidates[index]->votes_++;
}

```

```

4. vector<Candidates *> * get_candidates() {
    return &candidates_;
}

```

5. Display the results.

```

void Display() {
    create_UI(); // For all UI elements
    get_data_and_format_ui();
    setVisible(UI);
}

```

6. Process the file to create the objects of OPL or CPL.

```

void process() {
    int fp = open(fname_, "R");
    string inp;
    get_line(inp, fp);
    if (inp == OPL) {
        int v, b, c;
        v = get_line();
        b = get_line();
        c = get_line();
        vote_ = new OPL(v, b, c);
        for (int i = 0; i < c; i++) {
            Candidate * cand = new Candidate();
            get_line(inp, fp);
            parse_OPL_line(cand, inp);
            get_candidates()->push_back(cand);
        }
    } else if (inp == CPL) {
        int pc, v, b, c;
        vector<Party *> * parties;
        pc = get_line();
        get_line(inp, fp);
        parse_party(inp);
        v = get_line();
        b = get_line();
        c = get_line();
        vote_ = new CPL(v, b, c, parties);
        for (int i = 0; i < c; i++) {
            Party * party = new Party();
            get_line(inp, fp);

```

```

    parse_CPL_line(party, inp);
  } else { error here }
  for (num votes) {
    get_line(inp, fp);
    vote_ ->increment(get_one_index(inp));
  }
  Display();
}

```

7. Get the index of the position of one.

```

int get_one_index(char* ballots){
  char arr[numpartiesorcand];
  tokenize_string_into_array(arr);
  for (int i = numpartiesorcand) {
    if (arr[i] == 1) {
      return i;
    }
  } return -1;
}

```

8. Parses a candidate line for OPL

```

void parse_OPL_line(Candidate * cand, string inp) {
  char * arr[2];
  tokenize_string(arr);
  cand->name_ = arr[0];
  cand->party_ = arr[1];
}

```

9. Parses a candidate line for CPL

```

void parse_CPL_line(Party * party, string inp) {
  char * arr[3];
  tokenize_string(arr);
  party->members->push_back( arr[0]);
}

```

10. Creates a text file for the results of the processing

```

void create_text_file() {
  int fp = open(filename);
  write_all_members_and_data_into_file();
  close(fp);
}

```

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

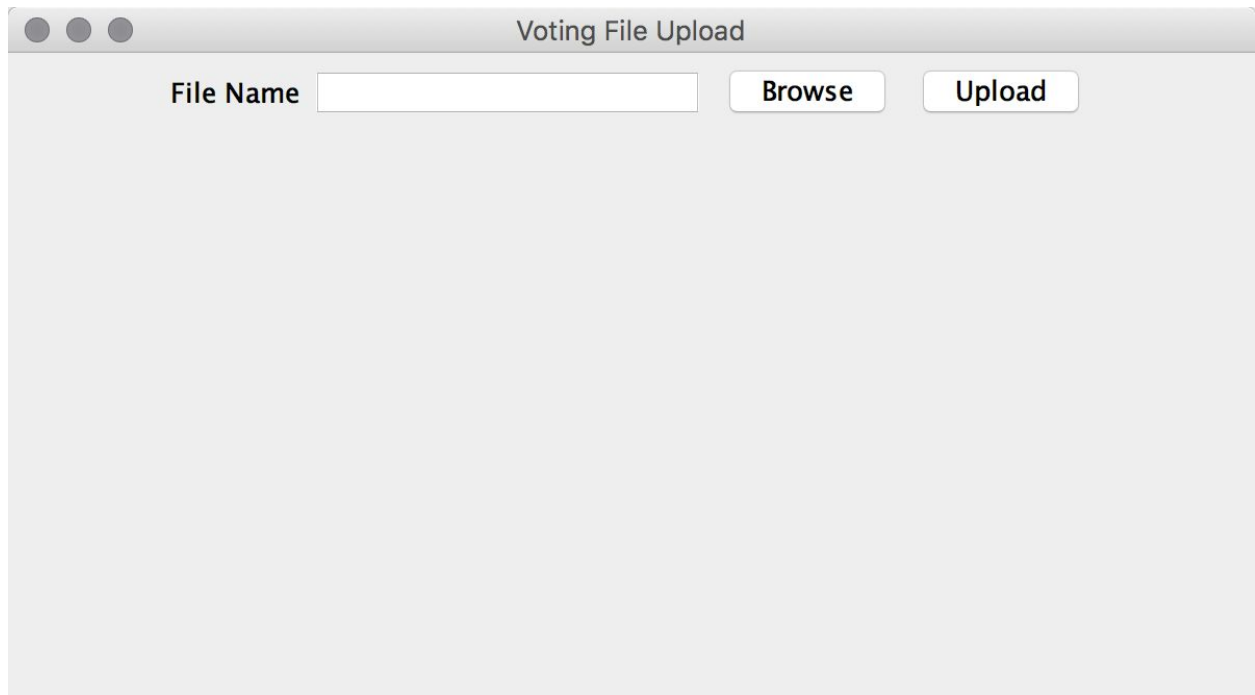
The user of voting system can specify the filename by browsing the local file system to upload the specific file from disk. The file must be csv format exported from Excel. After the user uploads the file successfully, the audit file will be generated automatically. The audit file will be txt format and can be downloaded locally. The user can also choose to display the explicit result to the screen and then download it in the local disk.

6.2 Screen Images

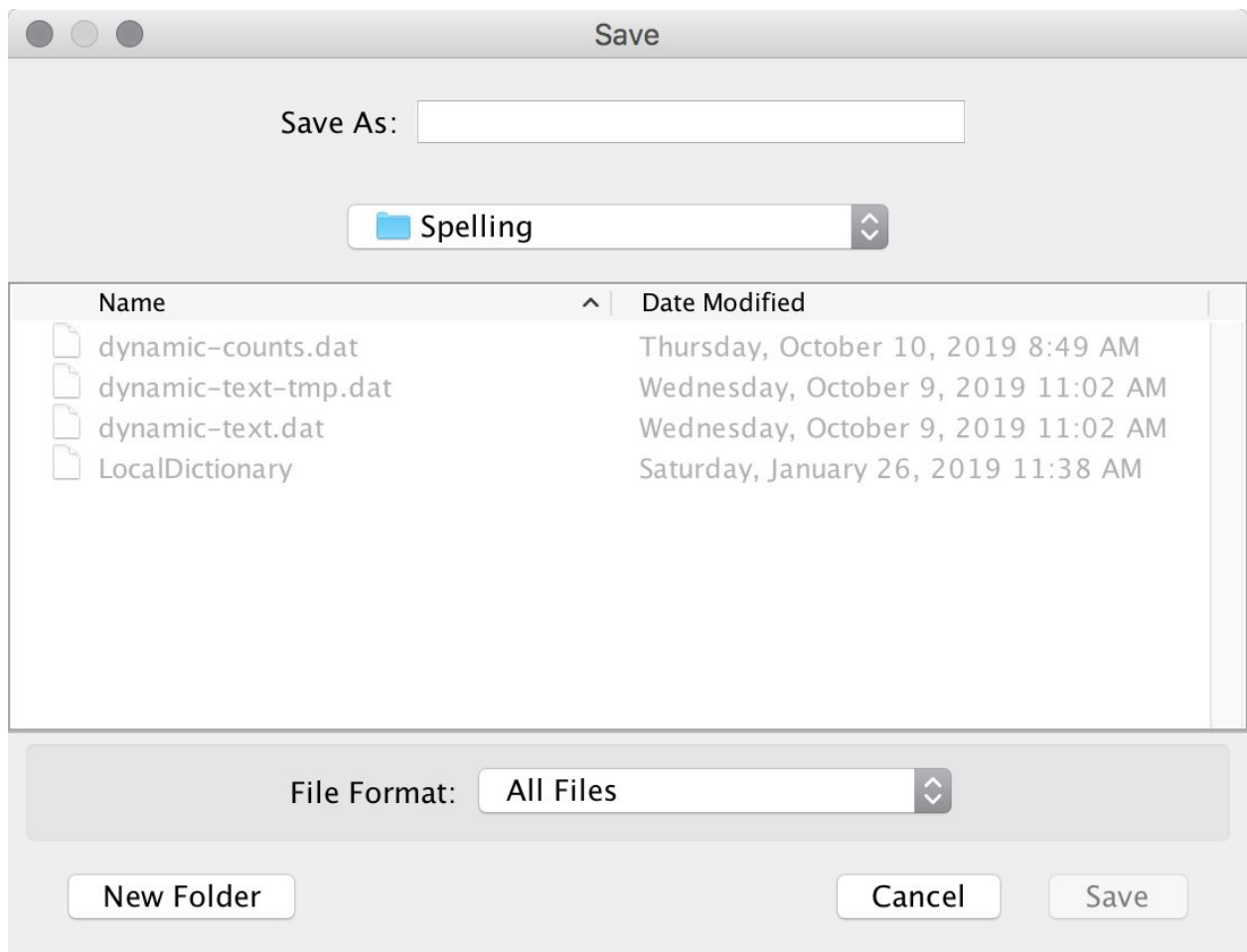
1. Main page and menu



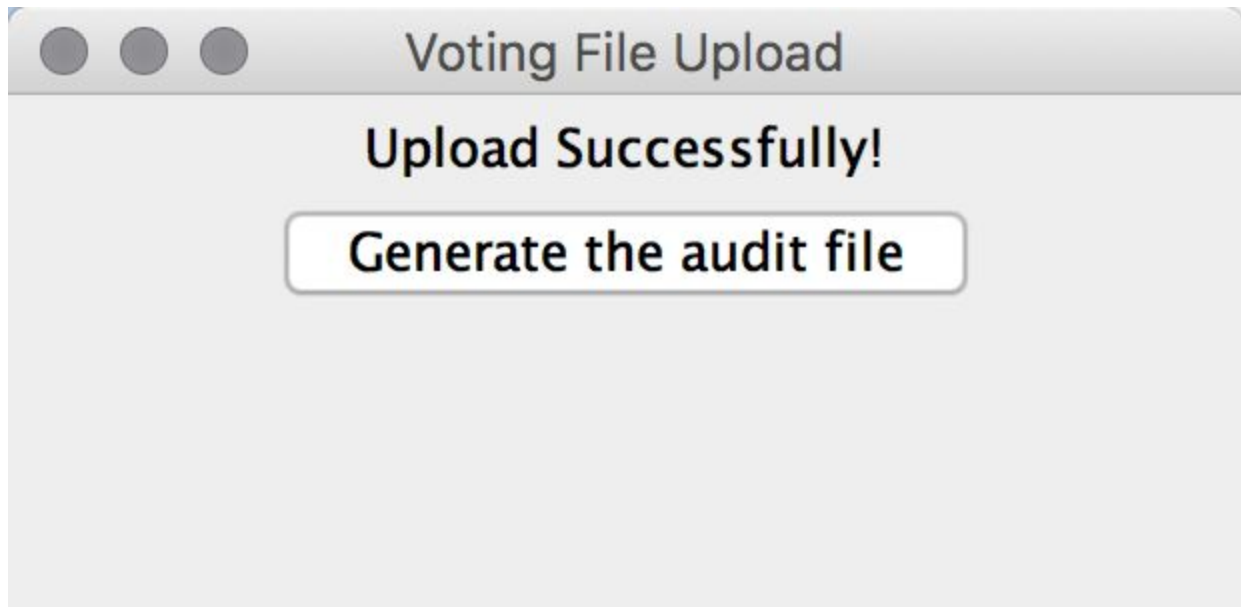
2. Prompt users to specify filename Screen:



3. Upload CSV format file exported from Excel:



4. Prompt users “Upload Successfully”:



5. Generate the audit file:
 - (1) Generate CPL audit file

● ● ●

Generate Audit File

Audit File Information:

Audit File Name:

auditFile2.txt

Save:

/User/Desktop/CSCI5801/SRS

Generated Time

Oct 10, 2019

CPL

4

[D,R,G,I]

7

100

16

[Pike,D,1]

[Foster,D,2]

[Floyd,D,3]

[Jones,D,4]

[Mallory,D,5]

[Deutsch,R,1]

[Wong,R,2]

[Walters,R,3]

[Keller,R,4]

[Borg,R,5]

[Jones,G,1]

[Smith,G,2]

[Lewis,G,3]

[Smith,G,4]

[Li,G,5]

[Perez,I,1]

I,,

I,,

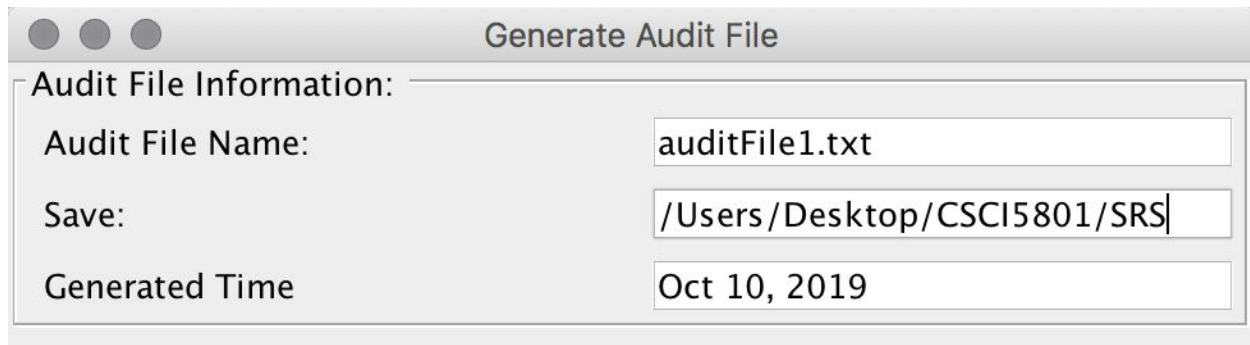
,I,,

,I,,

,,I,

,,

(2) Generate OPL audit file



Generate Audit File

Audit File Information:

Audit File Name: auditFile1.txt

Save: /Users/Desktop/CSCI5801/SRS

Generated Time Oct 10, 2019

OPL
3
9
6
[Pike,D]
[Foster,D]
[Deutsch,R]
[Borg,R]
[Jones,R]
[Smith,I]
I,,,,
I,,,,
,I,,,,
,,,I,
,,,I
,,I,,
,,I,,

6. Display results to screen:
 - (1) Display results for CPL

Voting Results:

Winner Name: Pike
Winner Party: Democratic
Election Type: CPL

Number of Parties : 3

Parties in Order of Ballot Ordering:

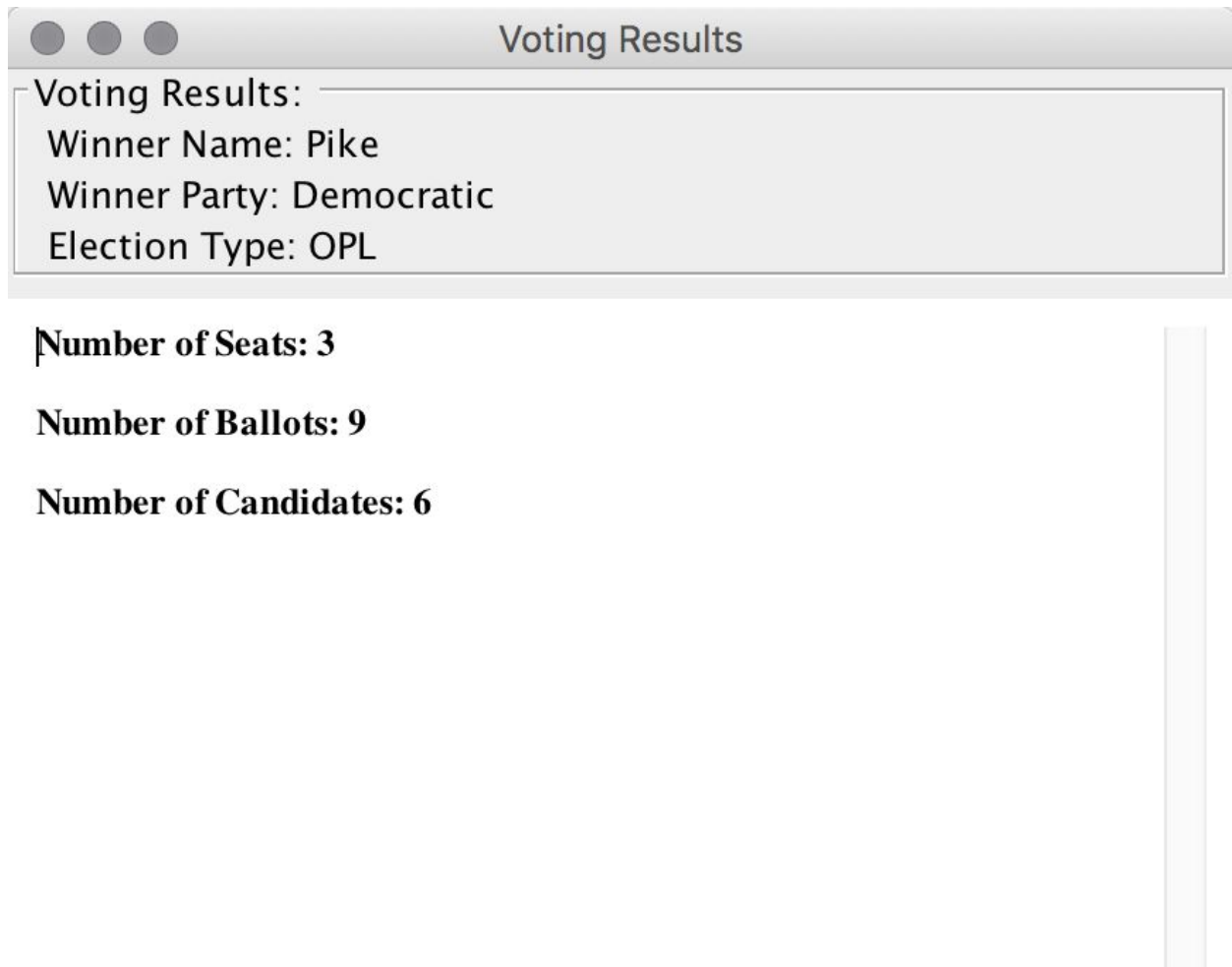
[1] D e m o c r a t i c
[2] R e p u b l i c a n
[3] G r e e n
[4] I n d e p e n d e n t

Number of Seats: 7

Number of Ballots: 1 0 0

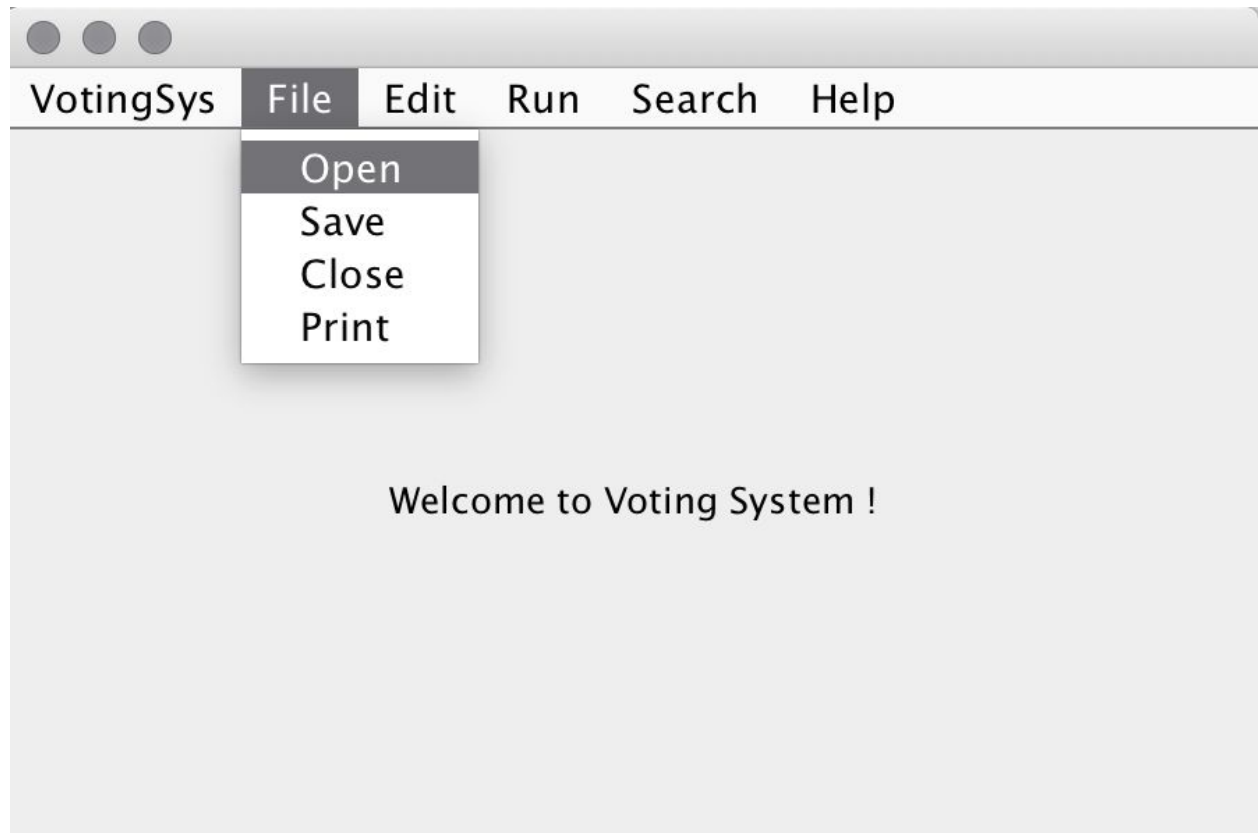
Number of Candidates: 1 6 v|

(2) Display results for OPL

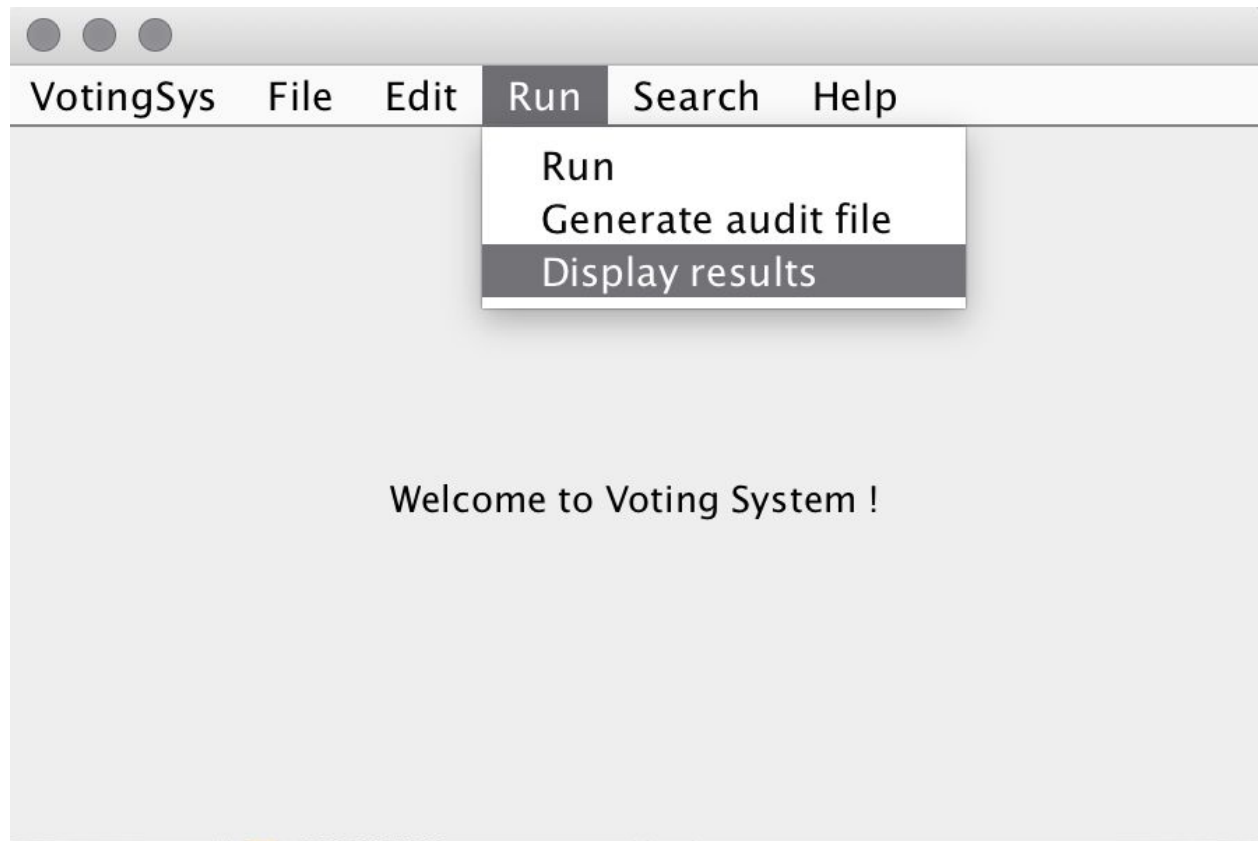


6.3 Screen Objects and Actions

The main page of the system when the user login has a menu bar at the top of the screen. User clicks the file button and then clicks the open button in the sub menu. Then the system goto 6.3.2.



The user clicks the “browser” button to choose the file from the local and clicks the “upload” button to upload the file. After submitting the file, the system prompts the user “Upload successfully” and the user clicks the “generate the audit file” button then the system goto 6.2.5.



The user downloads the audit file as txt format. The user clicks the “Display results” and the system goto 6.2.6. The user can choose to download the file as txt format for the results displayed on the screen.

7. REQUIREMENTS MATRIX

Please reference section 2.2 of the SRS for the functional requirements and section 4 for the system components.

Requirements Matrix					
Project: Voting System Application					
Business Requirements		Functional Requirements			
ID	Feature	ID	Requirement	Priority	Components
001	Analyzing voting files for OPL	001	Specify file name	High	Main
		002	Run OPL voting analysis		Processor, VotingType, OPL, Candidate
002	Analyzing voting files for CPL	001	Specify file name	High	Main
		003	Run CPL voting analysis		Processor, VotingType, CPL, Party
003	Displaying voting results	006	Display voting result	Medium	OPL, Candidate, CPL, Party
004	Exporting results to a file	004	Create OPL audit file	Low	Processor, OPL
		005	Create CPL audit file		Processor, CPL
		007	Create OPL text file		OPL, Candidate
		008	Create CPL text file		CPL, Party

8. APPENDICES

An appendix is not included for this document.