Team-12

Voting System

Software Design Document

Author(s):

Rex Zhu(zhu00100), Sunny Qin(qing0002),
Yingjin Zhang(zhan4943), Xiaohui Chao(chao0070)

Workstation: CSE LAB MACHINE LIND40-02

Date: (11/04/2019)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This Software Design Document comprises of concrete information about the Voting System which includes its intended use, architecture design, deployment method, and other relevant resources.

The expected audience is official staff, developers(programmers), and the people who will test this system (testers).

1.2 Scope

Countries and regions require a more and more transparent, efficient, and scientific method to hold elections. Under this circumstance, the Voting System is designed and put into use. The Voting System has encapsulated algorithms for two types of election: OPL (Open Party List) and CPL (Closed Party List).

The document contains a complete description of the design of OPL and CPL voting system.

The basic architecture is a java program which executes a voting file in terminal. The basic requirements are a well formed voting result file and terminal in CSE labs.

The designated faculty member in charge of the system will have full access to make changes, as he/she deems necessary. The changes could include the design and refinement of the system.

1.3 Overview

The remaining chapters and their contents are listed below.

Chapter 2 is an overview of the voting system, which includes the main components in the system and how the system should behave in the context.

Chapter 3 is the architectural design that specifies the design entities that collaborate to perform all the functions included in the system. Also, the details will be included in each design entity that collaborate to perform the services.

Chapter 4 concerns the data structure design.

Chapter 5 contains the components design. Each function will be in pseudocode format to show how they will implement the use cases.

Chapter 6 discusses the user interface design and the screen objects that will be included in the design.

Chapter 7 discusses the requirement matrix, which provides a cross reference that traces components and data structures to the requirements in the SRS document.

Chapter 8 is the appendix.

This document was created based on the IEEE template for Software Design Documents. Reading the document, readers can follow the sequence of Chapter 1-> Chapter 2 -> Chapter 3 -> Chapter 4 -> Chapter 5 -> Chapter 6 -> Chapter 7.

1.4 Reference Material

Textbook: Software Engineering, Ian Sommerville, Tenth Edition, ISBN 10: 0-13-394303-8, ISBN 13: 978-0-13-394303-0

System Design Document:

https://canvas.umn.edu/courses/134519/assignments/822290?module_item_id=3014484

Software Design Document example :
https://www.slideshare.net/peny_mg/sdd-software-des-sample

1.5 Definitions and Acronyms

See Appendix A in Chapter 8.

2. SYSTEM OVERVIEW

The Voting System is a useful, self-contained product. It is designed to count the voting ballots and decide the winners and winning parties. This system is supposed to be used after the actual voting is finished and a voting file is provided to the election officials. It allows election officials to transfer the original voting file into a final election result with the election information and output the result. Also, this system helps the election officials send results to media and let the public know the election results. All the functions mentioned above can be accomplished by this system itself.

Figure 2.1 shows the main components of the Voting System.

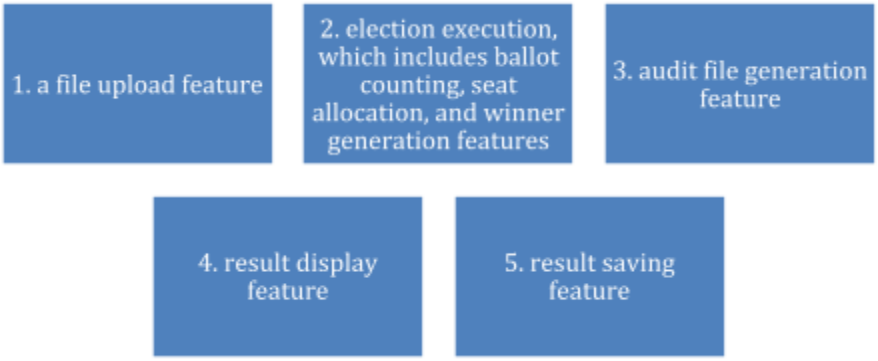


Figure 2.1 The main components of the Voting System.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

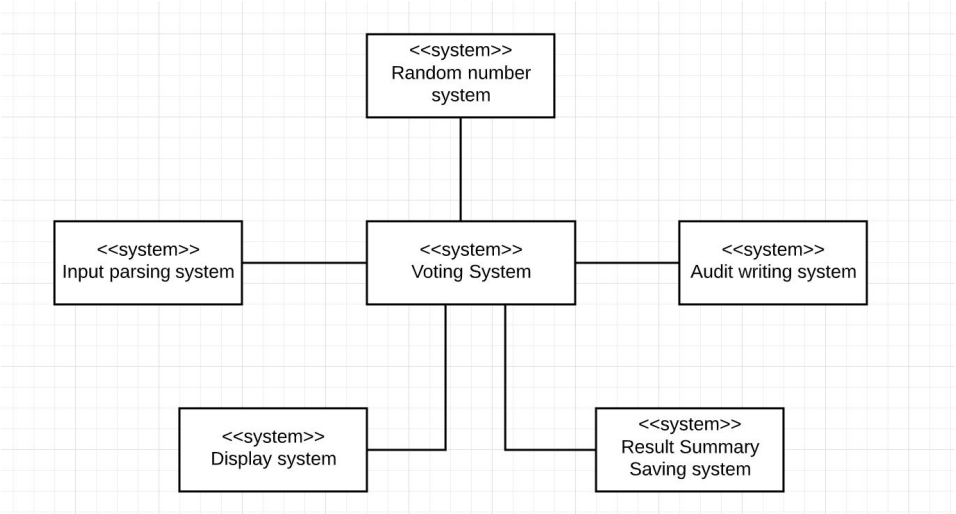


Figure 3.1. Architectural Design

The Voting System is connected with the Operating System and Random Number System. The Operating System includes operations like reading input, saving results, writing to a file, displaying to screen, which use java operation functions to complete the actions. The Random Number System is used when there is a tie occuring in seats allocation.

3.2 Decomposition Description

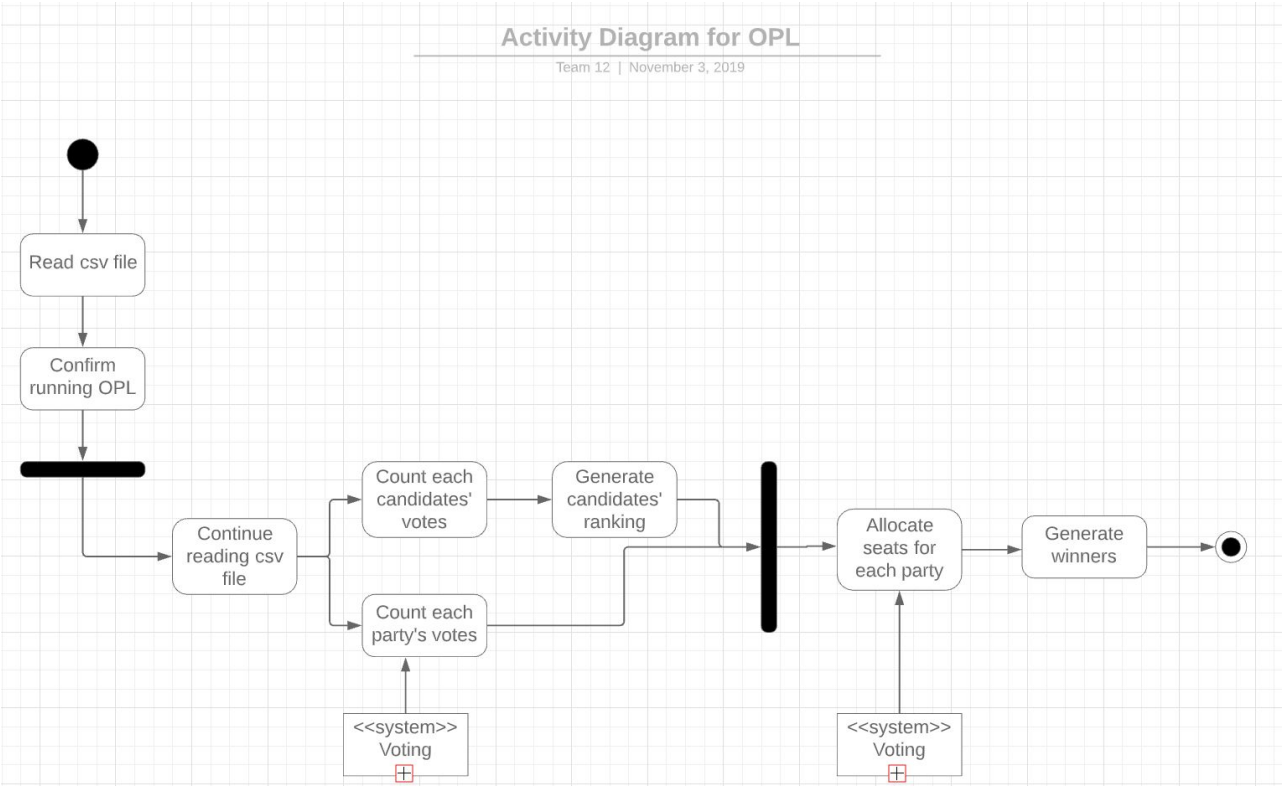


Figure3.2 Activity Diagram for OPL.

Figure 3.2 is the activity diagram for OPL. OPL starts from reading a csv file, confirming to run on OPL, then goes into counting ballots for both parties and candidates. After generating the candidates' ranking, it goes into allocating seats and generates winners.

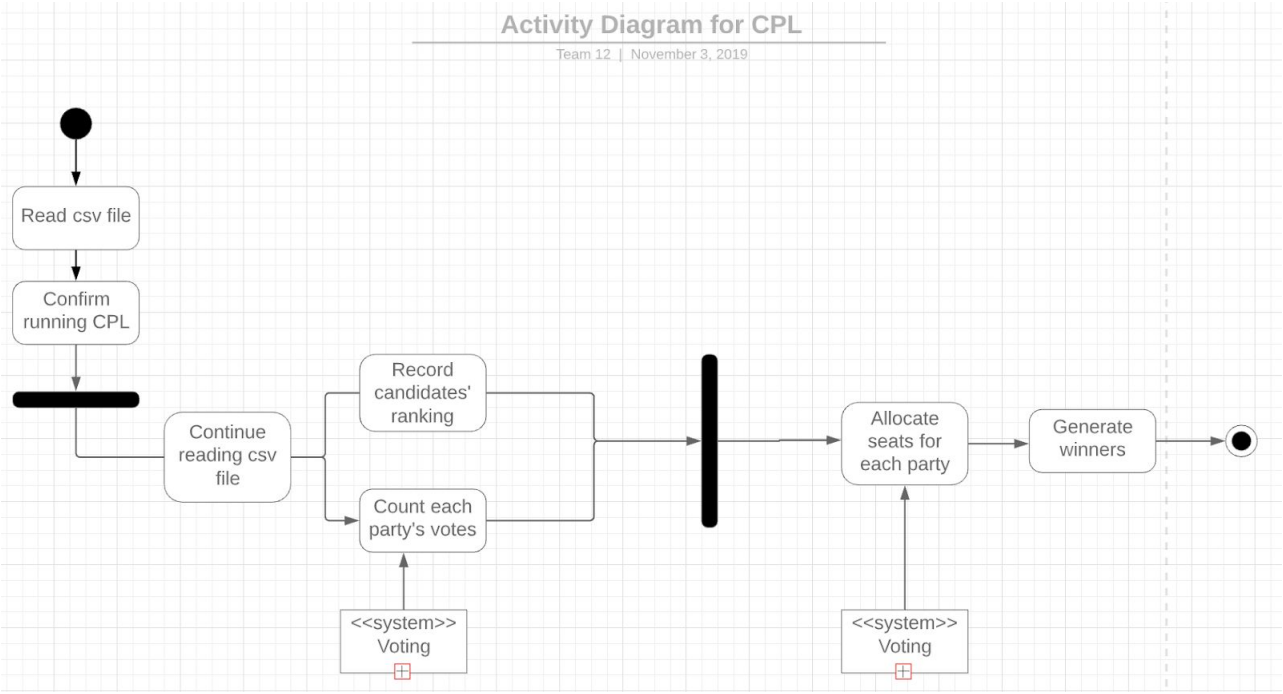


Figure3.3. Activity Diagram for CPL.

Figure 3.3 is the activity diagram for CPL. CPL starts from reading csv file. Confirm to run on CPL, then goes into counting ballots for party and record the candidates' ranking. Then, it goes into allocating seats and generates winners.

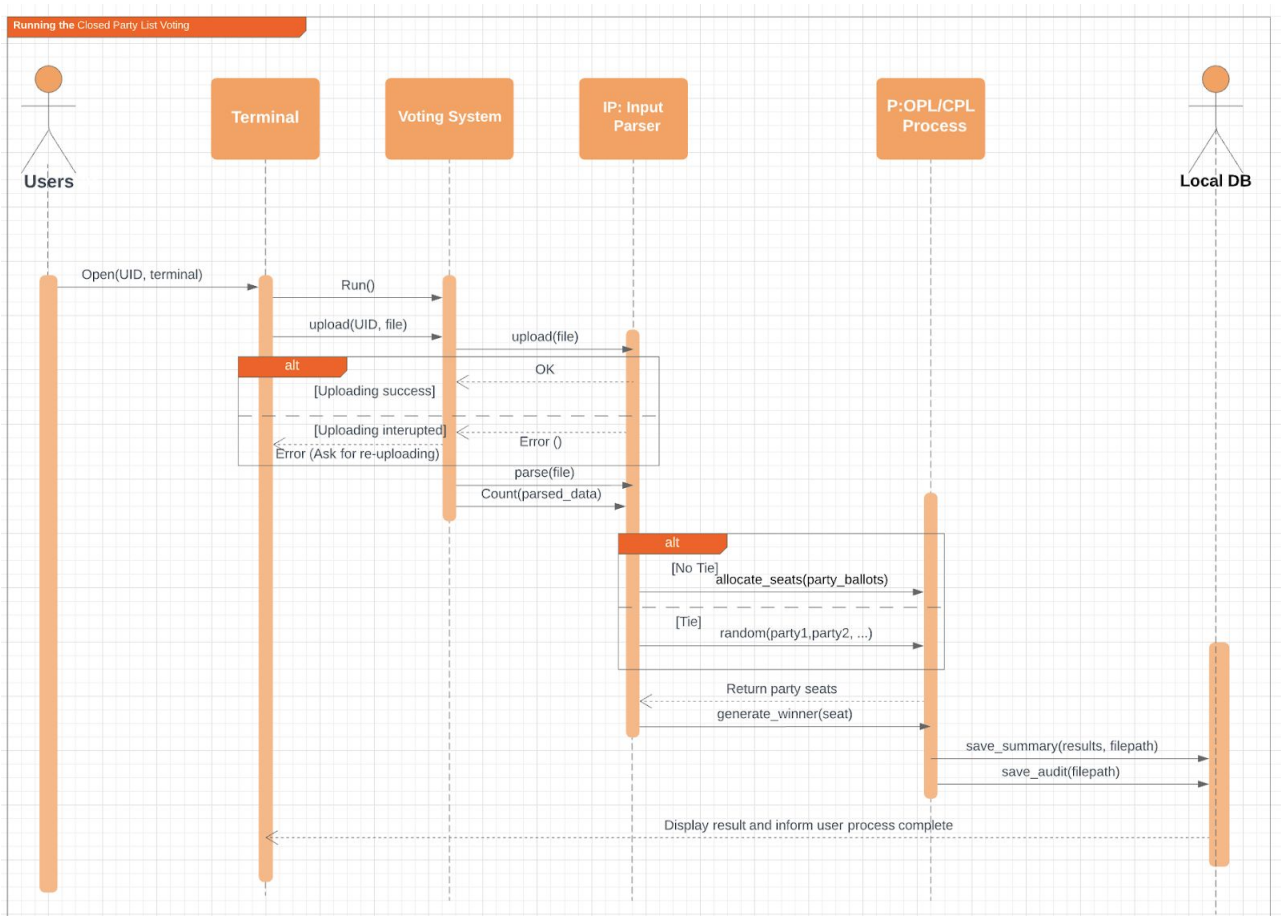


Figure 3.4 Sequence diagram for running the closed party list voting.

Figure 3.4 is the sequence diagram for running the closed party list voting. The user interacts with the Voting System through a terminal window. After the user runs the System, it prompts the user to enter the input csv file name. After the user enters the file name, the System uploads the file from the data storage item, i.e. the CSE lab machine, and parses the data. The system first checks the voting type and confirms it is CPL, then it counts the ballots using the corresponding algorithm. After ballots counting, the System allocates seats to parties. If there is a tie in the allocation result, the System needs to flip a coin to decide which party wins. Then, the System generates winners based on the seat allocation result and save an audit file and a summary report to the CSE lab machine.

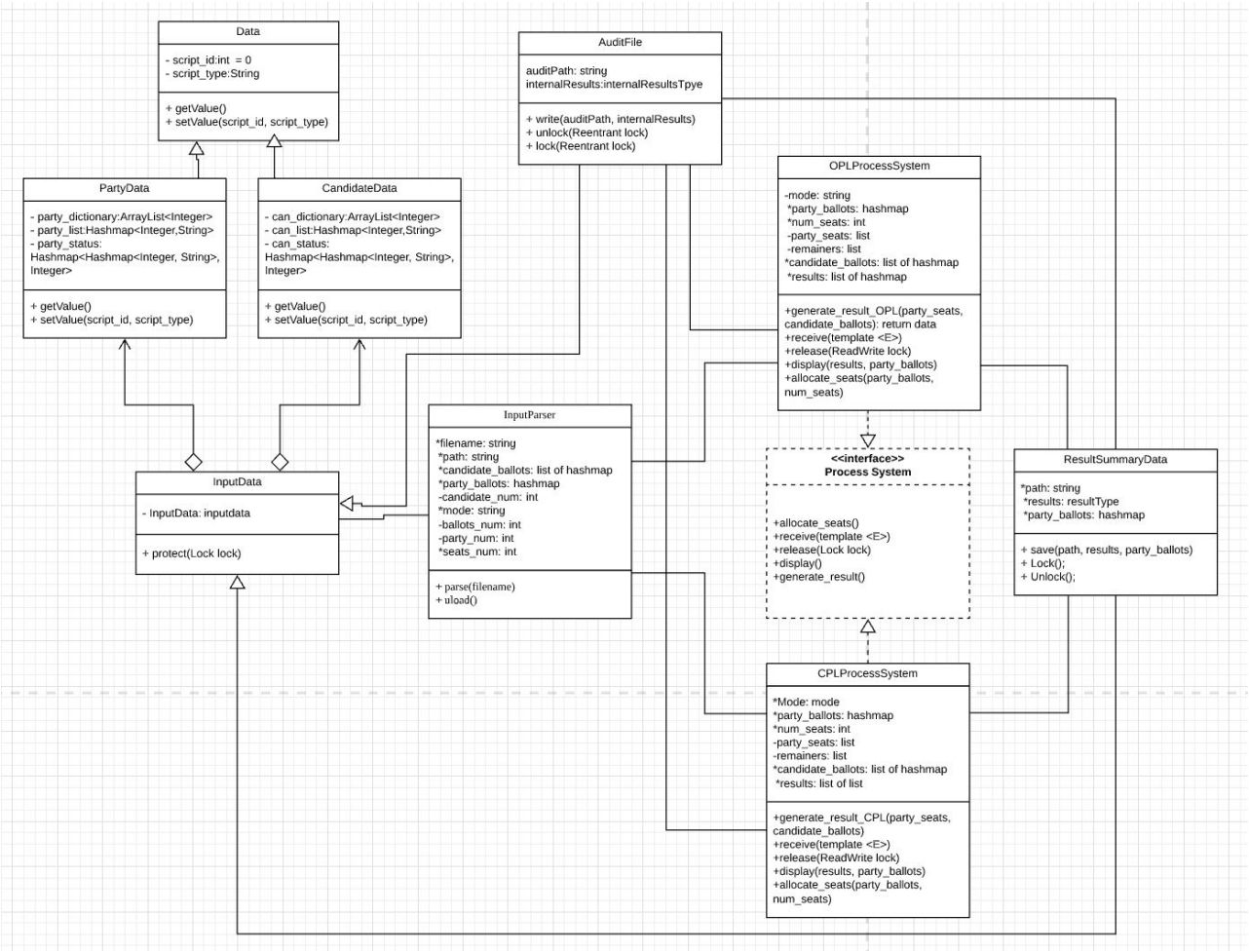


Figure 3.5 UML class diagram for the entire system.

Figure 3.5 is the UML class diagram for the entire system. The design of this system is fully penetrated with the Object-oriented design. In the Data art, Data is an abstract class for all the data-related classes. The Input data is a InputData class. As in the figure 3.5, the InputData is constructed with two parts: CandidateData (stores data of candidates) and PartyData(stores data of parties). The input parser is for receiving the input data and parsing the data into a programmable format. The input parser is also a classifier that can send the parsing results to expected mode(OPL / CPL). After successfully parsing the input data, the process system will extract those data and does a series of operations to generate the election result. As the process system processes the data, the audit data will be generated and kept updating. After processing the data, the result will be stored locally in ResultSummaryData as a record. When the data is localizing, the data will also be printed into output device.

3.3 Design Rationale

In developing the specification for this system, we have to decide whether the system should focus on developing a brand new method for voting process or whether use other external system to do this. The advantage of relying on random number generation system is to execute the

coin flipping only when a tie occurs. The major disadvantage, however, is that using other systems may make it slower to access information, and if these systems are unavailable, then it may be impossible to use the Voting System.

4. DATA DESIGN

4.1 Data Description

The input and output data are stored in the CSE lab computers. The original input data file is in the csv format. After the system loads the input data file from data storage item, it starts to process the input data file. The Voting System reads the first line of the input data file and then identifies whether the string in the first line is “opl” or “cpl.” Then the System reads the rest of the file by input parsing process and then calculates seats allocation for each party. Finally, the System generates two output files, i.e., an audit file and a summary report, and saves them in the CSE lab machine. These two files are text files. The internal results may use list, list of hashmap and hashmap to store the data.

4.2 Data Dictionary

Attribute Name	Attribute type	Attribute Size(Bytes)
auditPath	String	200
ballots_num	int	10000000
candidate_num	int	10000
candidates_ballots	ArrayList<HashMap<String, Integer>>	2000
filename	String	50
internalResult	InternalResult type	1000
mode	String	10
party_ballots	HashMap<String, Integer>	1000000
party_num	int	1000
party_seats	ArrayList<Integer>	1000
path	String	200

remainers	ArrayList<Integer>	200
results	ArrayList<HashMap<String, Integer>>	100000
seats_num	int	1000000

5. COMPONENT DESIGN

Input Parser:
Name: Input Parser
Type: Input data
Description: This is the input parser to parse input file into the useful information for further OPL/ CPL.
Attributes:
filename: string
path: string
candidate_ballots: list of hashmap
party_ballots: hashmap
candidate_num: int
mode: string
ballots_num: int
party_num: int
seats_num: int
Resources: None
Operations:
Name: upload
Arguments: None
Return: None
Pre-condition: Start running the system
Post-condition: The input data is sent to parser
Exceptions: None

Name: parse
Arguments: inputData
Returns: voting_type, party_ballots, candidate_ballots
Pre-condition: The input file has been loaded
Post-condition: The input data is parsed.
Exceptions: None
Flow of Events:
1. User upload file
2. Parser parses the file

Algorithm 1: upload()

Result: Filename,path
filename:=[];
print ”Please enter your filename:”;
filename:= scanner(System.in);
path = get path;
create a new audit file in this path;

Algorithm 2: parse(filename)

Result: Party_ballots, Candidate_ballots, ballots_num, seats_num, voting_type
open(filename);
String voting_type = Files.lines(filename).findFirst().get();
if *voting*$_type == \acute{C}PL\acute{}$ **then**
    party_num := read next line;
    party_ballots{}:= read next line to key and set value to 0;
    seats_num := read next line;
    ballots_num := read next line;
    candidate_num := read next line;
    while *i ¡ candidates_num* **do**
        candidates$_b allots :=$
        $add the candidate name to the dictionary of the party she he belongs to,$
        set the name as key, the position as value;
        i++;
    end
    while *reach the end of the file* **do**
        read next line, find the position of 1, add 1 to corresponding value in party_ballots;
    end
else
    seats_num := read next line;
    ballots_num := read next line;
    candidate_num := read next line;
    while *i ¡ candidates_num* **do**
        candidates_ballots := add the candidate name to the dictionary of the party she he belongs to,
        set the name as key, set the value to 0;
        i++;
    end
    while *reach the end of the file* **do**
        read next line, find the position of 1, add 1 to corresponding value in candidate_ballots and to party_ballots;
    end
end

OPL Process System:
Name: OPL Process System
Type: System
Description: This is a system that can process OPL voting, including allocating seats, generating winner and displaying results.
Attributes: Mode: string
party_ballots: hashmap
num_seats: int
party_seats: list
remainers: list
candidate_ballots: list of hashmap
results: list of hashmap

Resources: None
Operations: Name: allocate_seats
Arguments: Inputdata
Returns: party_seats
Pre-condition: The ballotes has been counted for each party.
Post-condition: The seats will be allocated to each party.
Exceptions: None

Name: generate_result
Arguments: Mode, Data
Returns: voting results
Pre-condition: The input file has been loaded and the input data has been parsed.
Post-condition: The voting results has been generated.
Exceptions: None

Name: display
Arguments: voting result
Returns: None
Pre-condition: The result has been generated
Post-condition: The results display on the screen
Exceptions: None
Flow of Events:

1. Based on the voting type, get into the process
2. run the process
3. generate result

Algorithm 3: allocate_seats(party_ballots, num_seats)

```
Result: party_seats
party_seats = [];
remainers = [];
for party in party_ballots do
    party_seat:= party / num_seats;
    party_seats.append(party_seat);
    remainers.append(party % num_seats);
end
remain_seats = num_seats - sum(party_seats);
write_audit(auditPath,[party_seats, remainers]);
while remain_seats != 0 do
    ind = find_the_greatest_rem(remainers);
    party_seats[ind]++;
    remainers[ind] = 0;
    remain_seats- -;
end
write_audit(auditPath,[party_seats]);
```

Algorithm 4: generate_result_OPL(party_seats, candidate_ballots)

```
Result: results
results = [];
for p in len(party_seats) do
    party = ;
    seats = party_seats[p] while seats != 0 do
        find candidate with largest ballots in p;
        party[”candidate”] = candidate_ballots[p][”candidate”];
        candidate.ballots = 0;
        seats- -;
    end
    results.append(party);
    write_audit(auditPath, results);
end
```

Algorithm 5: display(results, party_ballots)

```
Result: None
for p in len(party_ballots) do
    partyName = party_ballots.keys();
    print(partyName[p], results[p])
end
```

CPL Process System:
Name: CPL Process System
Type: System
Description: This is the system processing CPL type voting, including allocating seats, generating winner and displaying result
Attributes: Mode: mode ,
party_ballots: hashmap
num_seats: int
party_seats: list
remainers: list
candidate_ballots: list of hashmap
results: list of list

Resources:None
Operations:
Name: allocate_seats
Arguments: Inputdata
Return: party_seats
Pre-condition: The input file has been loaded and the input data has been parsed.
Post-condition: The seats for each party are allocated.
Exceptions: None

Name: generate_result
Arguments: Mode, Data
Return: voting results
Pre-condition: The seats have been allocated.
Post-condition: The voting results has been generated.
Exceptions: None

Name: display
Arguments: Result
Returns: None
Pre-condition: The result has been generated
Post-condition: The results display on the screen
Exceptions: None
Flow of Events:
1. Based on the voting type, get into the process
2. run the process
3. generate result

Algorithm 3: allocate_seats(party_ballots, num_seats)

```
Result: party_seats
party_seats = [];
remainers = [];
for party in party_ballots do
    party_seat:= party / num_seats;
    party_seats.append(party_seat);
    remainers.append(party % num_seats);
end
remain_seats = num_seats - sum(party_seats);
write_audit(auditPath,[party_seats, remainers]);
while remain_seats != 0 do
    ind = find_the_greatest_rem(remainers);
    party_seats[ind]++;
    remainers[ind] = 0;
    remain_seats- -;
end
write_audit(auditPath,[party_seats]);
```

Algorithm 4: generate_result_CPL(party_seats, candidate_ballots)

```
Result: results
results = [];
for p in len(party_seats) do
    party = [];
    seats = party_seats[p] while seats != 0 do
        find candidate with highest ranking in p;
        party.append(candidate);
        candidate.ranking = inf;
        seats- -;
    end
    results.append(party);
    write_audit(auditPath, results);
end
```

Algorithm 5: display(results, party_ballots)

```
Result: None
for p in len(party_ballots) do
    partyName = party_ballots.keys();
    print(partyName[p], results[p])
end
```

ResultSummaryData:
Name: ResultSummaryData
Type: a text file
Description: This is an output file, which contains the voting results generated by process system.
Attributes:
path: string
results: ResultType
party_ballots: hashmap
Resources:None
Operations:
Name: save
Arguments: ResultData
Return: None
Pre-condition: The results of the voting has been generated.
Post-condition: A file includes the result summary of the voting process has been written.
Exceptions: None
Flow of Events:
1. Read the result data
2. write the result data to the result summary file

Algorithm 7: save(path, results, party_ballots)

```
Result: None
new = newFile(path);
for p in len(party_ballots) do
    partyName = party_ballots.keys();
    new.write(partyName[p], results[p]);
end
```

AuditFile:
Name: AuditFile
Type: a text file
Description: This is the audit file, which includes every detail about the voting process, such as input data, counting ballots, allocated seats and voting results
Attributes:
AuditPath: String
InternalResults: InternalResultsTpye
Resources:None
Operations:
Name: write_audit
Arguments: AuditData
Return: None
Pre-condition: The internal result has been generated

Post-condition: The internal result is written in the file
Exceptions: None
Flow of Events:

1. One operation is finished
2. Write the result into the audit file

Algorithm 8: write_audit(auditPath, internalResults)

Result: None
audit = auditPath.load();
audit.write(internalResults);

6. HUMAN INTERACTION DESIGN

6.1 Overview of User Interface

To start, the users need to add their CSV file path in the command line to run the program. After uploading the CSV file, the system starts to read the first line of the file. Based on the information included in the first line, the file will be the input source for corresponding election methods. The system should print out the information they read from the first line to let the user know the progress.

When the system starts to count the ballots, there should be a line printed out to inform the user the progress of the counting and allocating process. After the system is finished, the audit file is already generated in the same directory, so there should be information about that file to indicate where it is and what it includes. The summary report of the election will be printed after the program finishes, and there will also be a text file describing the result saved in the same directory. The user can send it to media later.

6.2 Screen Images

Figure. 6.1, 6.2, and 6.3 are examples of the user interface screen images. The Voting System is operated through a terminal window by users. Users will need to input all the required information into the terminal and receive all the feedback from the terminal. In Figure 6.1, the Voting System prompts a message to the user for the csv file name. After the user enters the csv file name, the system will upload the corresponding csv file and count the ballots.

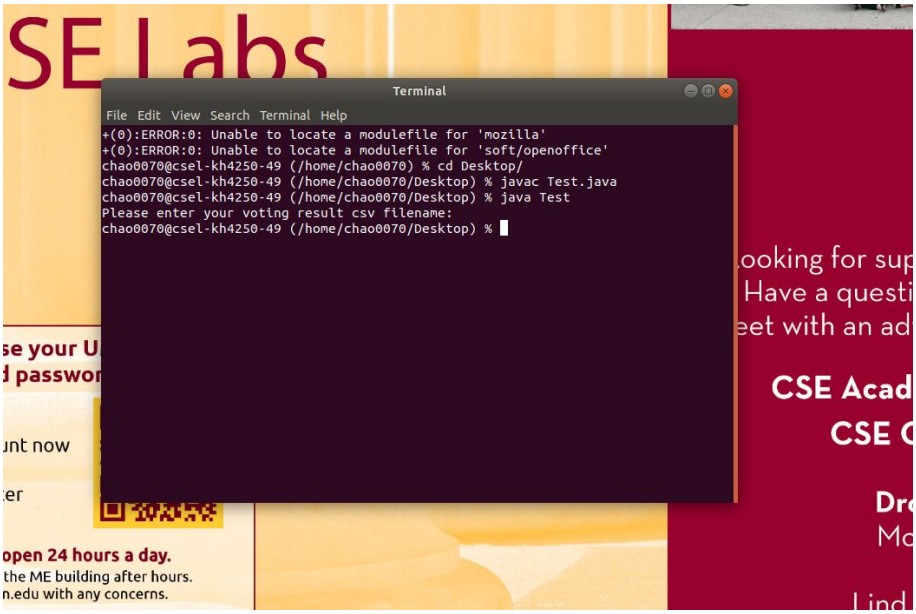
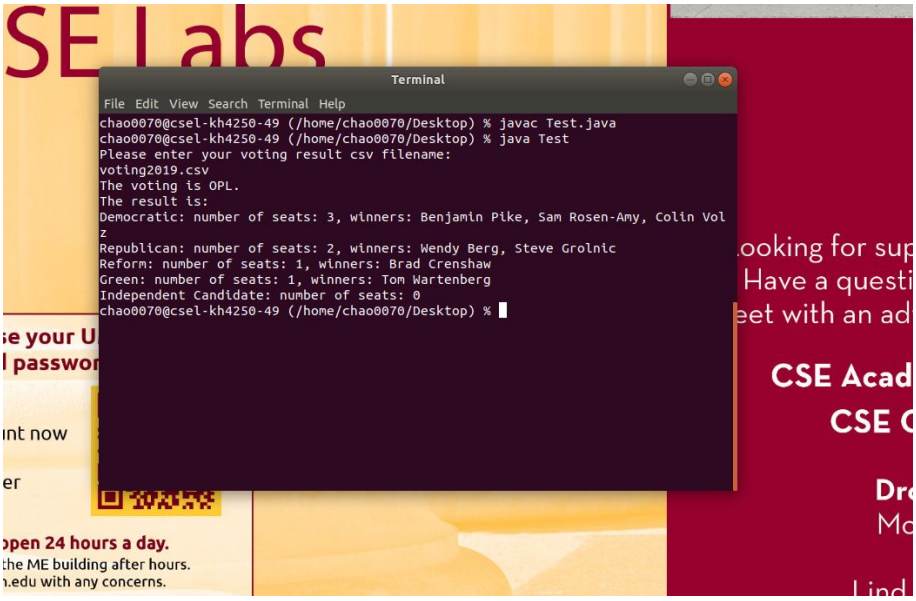
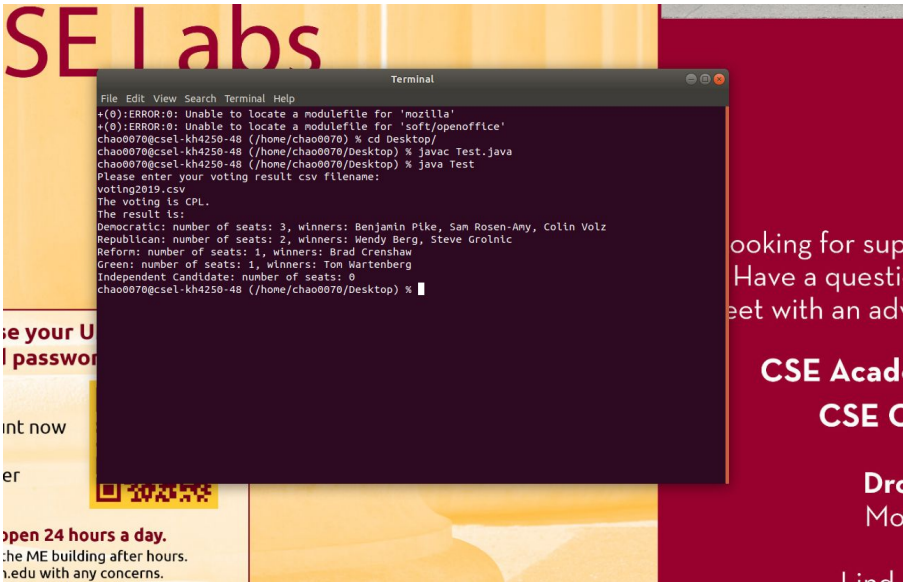


Figure 6.1 An example of the Voting System user interface. It prompts the user to enter the csv file name.

Before the System starts the ballot counting process, it needs to identify whether the voting type is CPL or OPL through the first line of the csv file. Then, the System will print the voting type to the screen. Based on different voting types, the System will count the ballots using different algorithms. Then, it will print the election results to the screen, as shown in Figure 6.2.



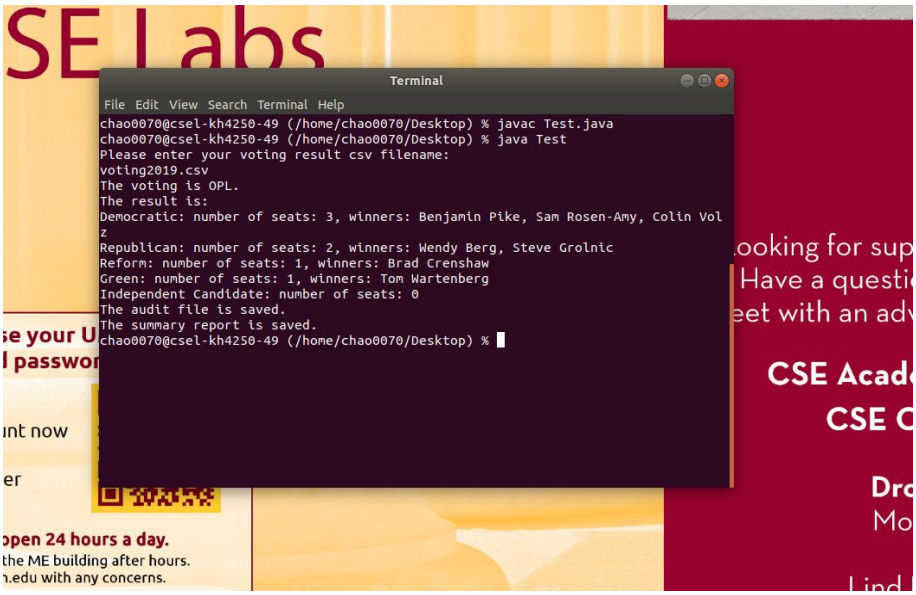
(a)



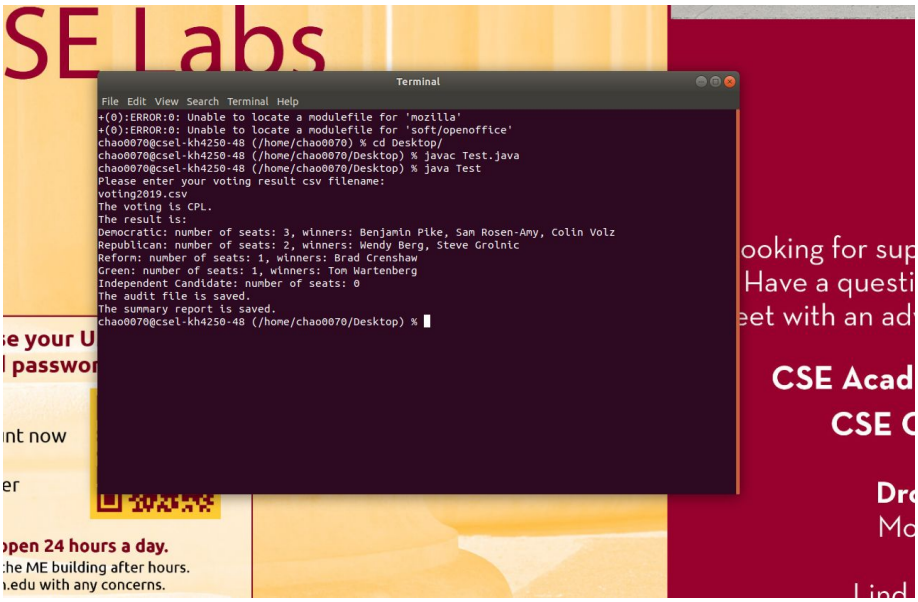
(b)

Figure 6.2 An example of the Voting System user interface after the ballots are counted. It shows the election results. (a) User interface for OPL voting. (b) User interface for CPL voting.

After the System counts the ballots, it will generate an audit file and a summary report. They are saved in the directory where the csv file is saved. The audit file contains every detail, such as the voting type, number of candidates, candidates, number of ballots, calculations, how many votes a candidate or party has. The audit file should also list the winner(s), and show how the election progressed so that the audit could replicate the election itself. It should also show who got what ballot and its order of being received if applicable. The summary report concludes the results of voting and will be sent to media personnel.



(a)



(b)

Figure 6.3 An example of the Voting System user interface after the summary report and audit file are generated and saved. (a) User interface for OPL voting. (b) User interface for CPL voting.

6.3 Screen Objects and Actions

The system will prompt the user to enter their csv file name. After the user enters the file name, the system will recognize the filename and run the program.

Furthermore, there will be an info section, which shows the results and saved files' detailed information after running the program.

7. REQUIREMENT MATRIX

Table 1.1 shows the system components and each of the functional requirements they satisfy in the SRS document.

System Components	Functional Requirements
InputParser	Upload a voting result file
OPLProcessSystem	Count ballots (CPL)
CPLProcessSystem	Count ballots (OPL)

OPLProcessSystem/ CPLProcessSystem	Allocate seats to parties
OPLProcessSystem/ CPLProcessSystem	Flip a coin
OPLProcessSystem/ CPLProcessSystem/ ResultSummaryData	Generate winners
AuditFile/ OPLProcessSystem/ CPLProcessSystem	Write in the audit file (OPL)
AuditFile/ OPLProcessSystem/ CPLProcessSystem	Write in the audit file (CPL)
ResultSummaryData/ OPLProcessSystem/ CPLProcessSystem	Display and save the results

Table 1 The system components and their corresponding functional requirements

8. APPENDICES

Appendix A: Glossary

- ***Voting System****: Election Voting System, which is used by election officials during the election.*
- ***CSV****: a kind of file format, which can be generated by Excel*
- ***CPL****: stands for "closed party list". In a closed party list system--the original form of party list voting--the party fixes the order in which the candidates are listed and elected, and the voter simply casts a vote for the party as a whole. Voters are not able to indicate their preference for any candidates on the list but must accept the list in the order presented by the party.*
- ***OPL****: stands for "open party list". In an open party list system, voters are allowed to express a preference for particular candidates, not just parties. It is designed to give voters some say over the order of the list and thus which candidates get elected.*
- ***Allocate Seats****: After voting, each party receives the number of seats they win. There are lots of different formulas for the allocation of seats to the parties. One is called the "largest remainder formula," which is also used in this voting system. In this approach, the first step is to calculate a quota, which is determined by taking the number of valid votes and dividing it by the number of seats. The party wins one seat for each whole number produced. After the first allocation of seats is complete, the remainder numbers for the parties are compared and the parties with the largest remainders are allocated the remaining seats. Finally, all the parties win the number of seats that as closely as possible approximate their percentage of the vote.*

Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.