Team26

Voting System

Software Design Document

Name (s):
Hongyu Zhu(zhu00091)
Songyan Wu(wuxx1331)
Ge Yu(yuxx0851)
Date: (10/30/2019)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the Voting system. This document is intended for both developers and software users including voters, who are involved to express their preference for candidates or parties in such election process; candidates, who are nominated in the party lists of the election with their parties; party members, who belong to the parties that are in the election; general public, who is awe of such election and want to acknowledge the out come of this election.

1.2 Scope

Voting system is a tool that implements the ability to perform two types of voting: open party list voting and closed party list voting. Open party list voting allows voters 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.In a closed 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. This voting system is designed to replicate the election process and display the outcomes. The intended audience which stated above will be able to learn about the election through this voting system.

1.3 Overview

The remaining chapters and their contents are listed below.

Section 2 is a Deployment Diagram that shows the physical locations where the system actually exists. This allows a clear explanation of where each design entity will reside. Each part will work in unison to accomplish each requested task.

Section 3 is the Architectural Design that specifies the design entities that collaborate

to perform all the functions included in the system. Each of these entities has an Abstract description concerning the services that it provides to the rest of the system. In turn, each design entity is expanded into a set of lower-level design operations that collaborate to perform its services.

Section 4 concerns the Data Structure Design.

Section 5 contains the Use Case Realizations. Each Use Case stated in the SRS Document can be traced by the given design objects.

Section 6 discusses the User Interface Design, and how it can be created with maximum user efficiency and ease of use.

Section 7 covers the help system.

Section 8 includes any additional appendices

1.4 Reference Material

Voting System requirement:

https://canvas.umn.edu/courses/134519/files/8682408/download?wrap=1

1.5 Definitions and Acronyms

- UI: User interface
- API:Application programming interface
- Voters: who are involved to express their preference for candidates or parties in such election process
- Candidates: who are nominated in the party lists of the election with their parties
- Party members: who belong to the parties that are in the election
- General public: who is awe of such election and want to acknowledge the out come of this election

- OPL: Open party list
- CPL: Closed party list
- EOF: End of file
- Voting Process: How the votes are distributed
- Display: Print a diagram/text as a result to the screen

2. SYSTEM OVERVIEW

Voting System is developed for everyone that would like to know the voting process and results, such as the voters, the analytical staff or even the party representatives. It has some functions like prompt, read the file, and also have options to choose based on different Open Party List or Closed Party List. With this software, the process and result can be displayed directly through the screen, guarantees fairness and efficiency. This software can run on different platforms such as Windows, Mac Os and Linux.

The description of functionalities are listed as following:

System:

- Read: System can read the voting result based on the text file.
- Prompt: System is able to prompt the user with filename.
- Audit File: System is able to produce an audit file for future analyzing based on the voting type
- Display result: System can display the election result
- Display voting process:System is able to display the process of voting

Voter:

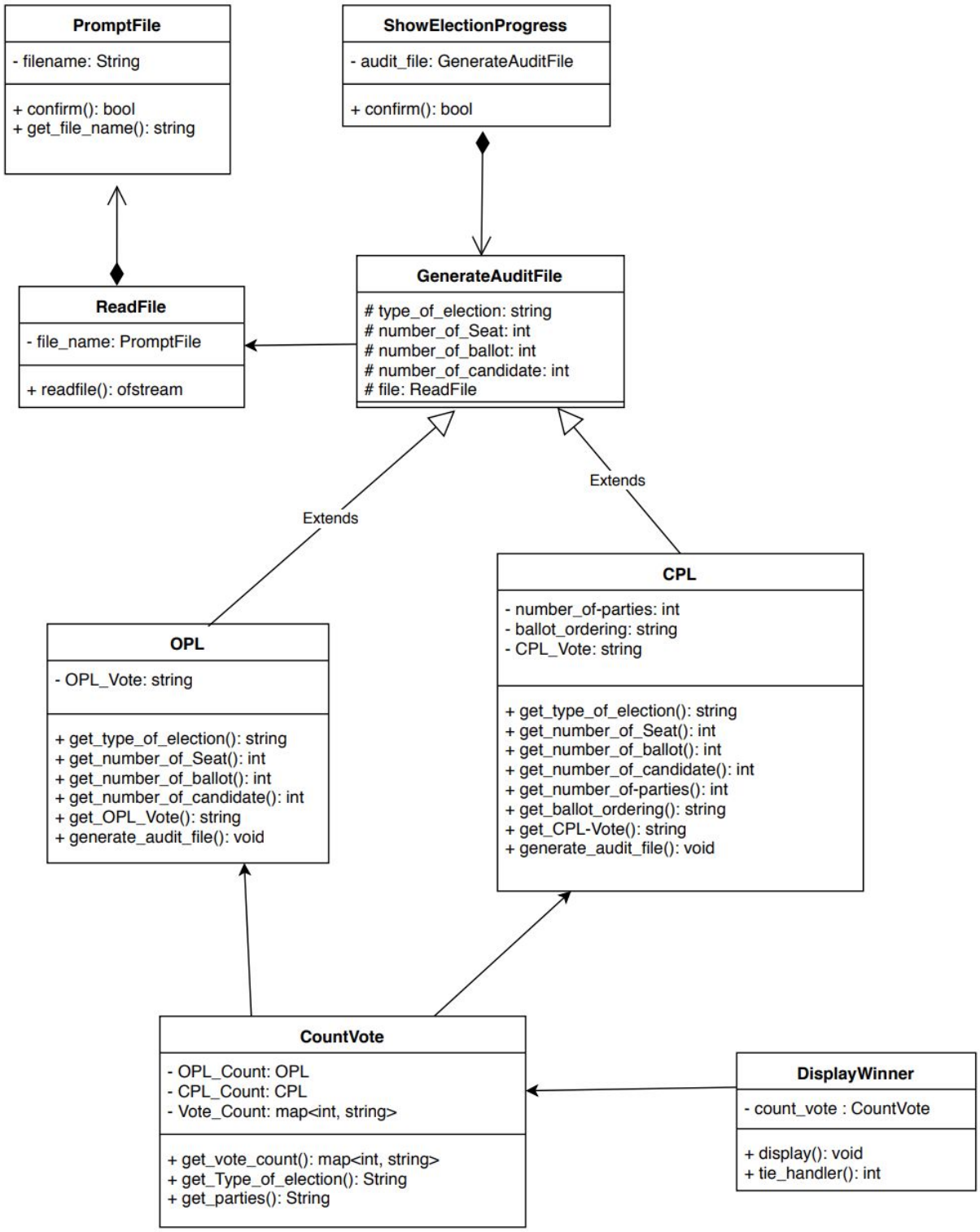
- Run: User can run the software

The Design and Implementation Constraints:

Voting System is developed in C++ and uses Visual Studio for its visualization interaction. It uses a modular design where every feature is wrapped into a separate module, and the modules depend on each other through well-written APIs. There are several APIs available to make plugin development easy.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

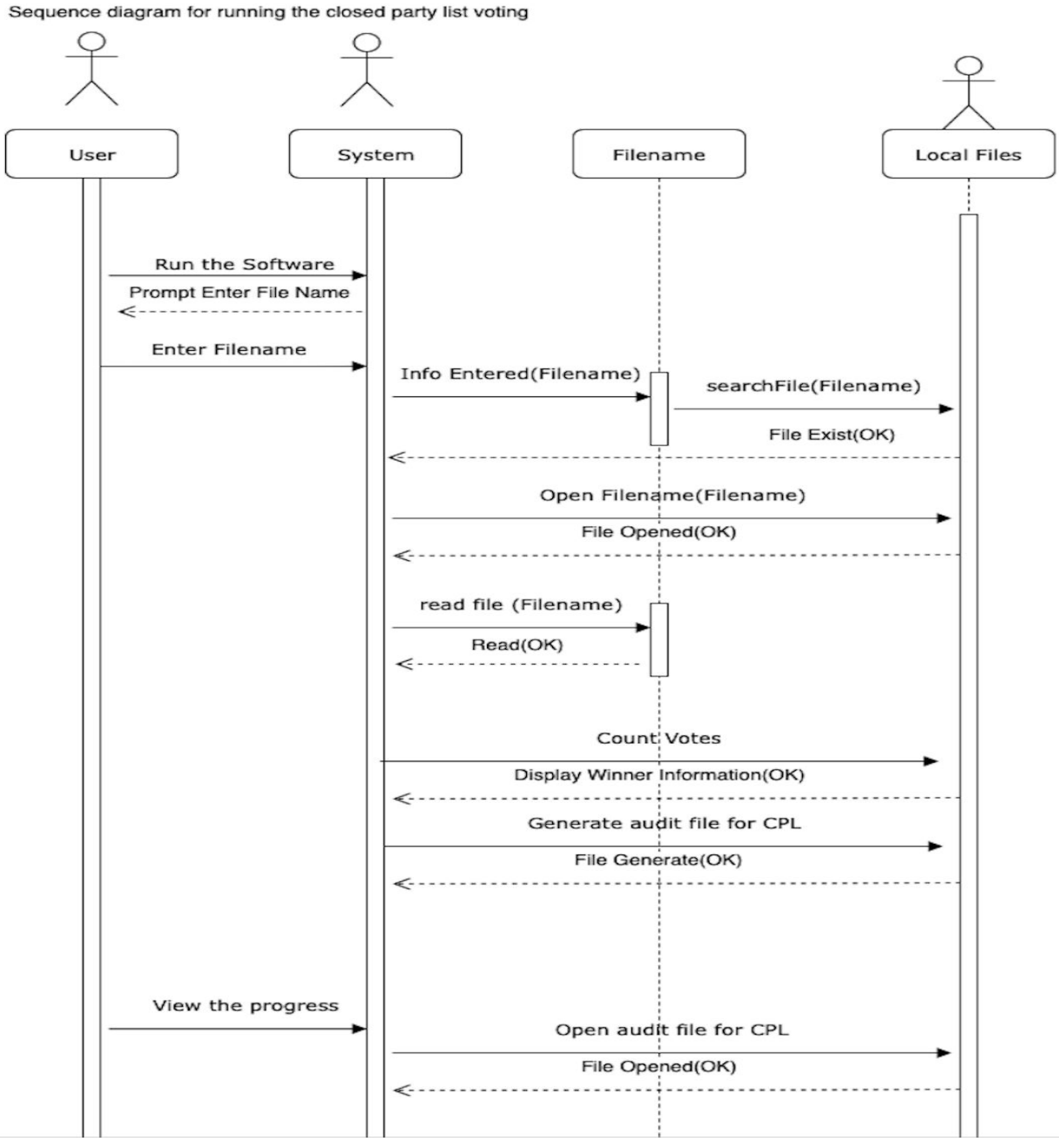


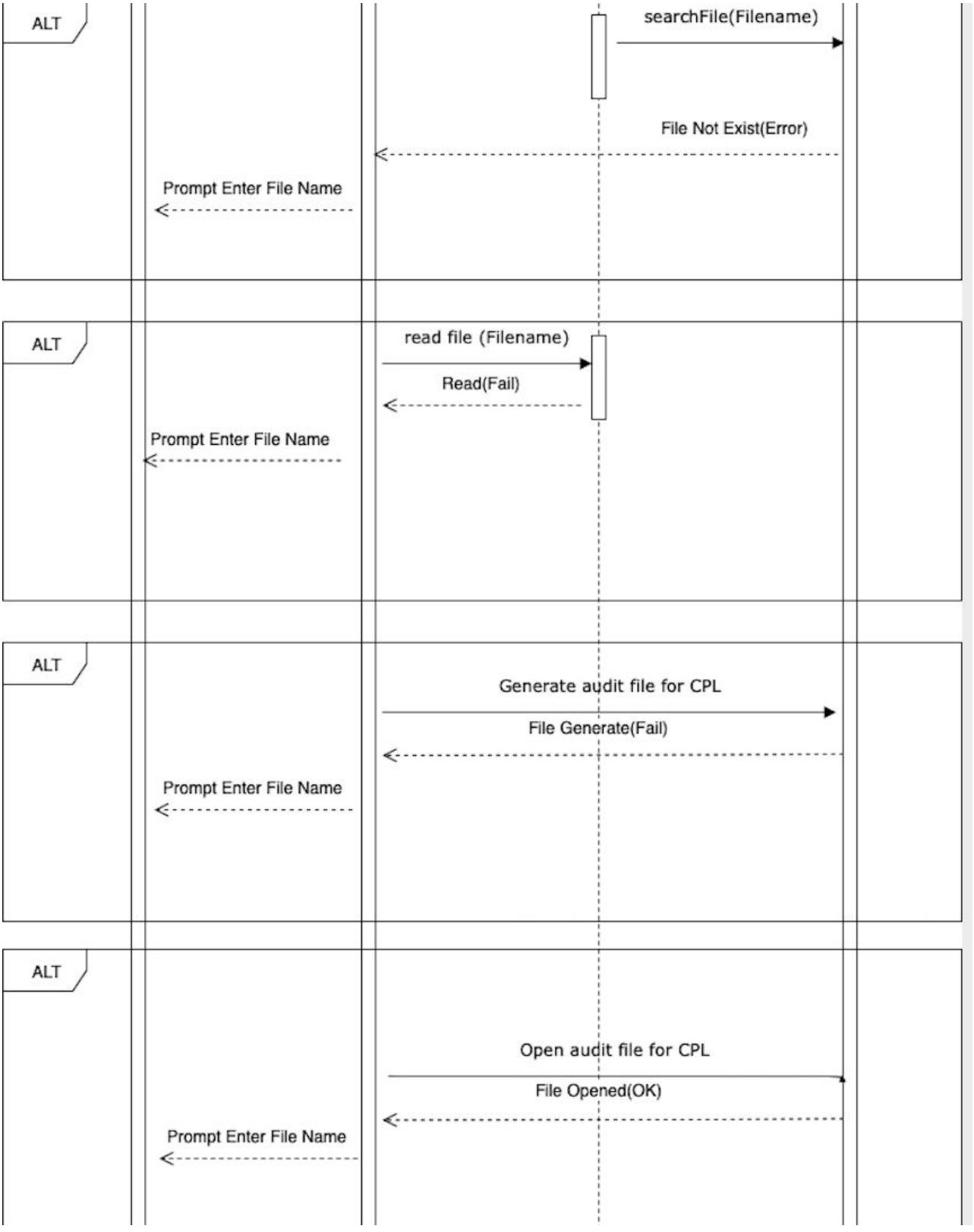
The UML-CLASS diagram has 8 classes. The first class is PromptFile which records the filename corresponding to the user's input. The second class is ReadFile, it has a member variable

from PromptFile class, since ReadFile cannot run if there is user input, the relationship should be aggregation. The third class GenerateAuditFile, it uses the file pointer from ReadFile to read the file, and it stores file information into protected member variables. OPL and CPL are inherited form GenerateAuditFile class, it also stores the file information due to the type of election. CountVote class uses the information from OPL and CPL class and arrange the votes to the candidate in a map. DisplayWinner class will traverse the CountVote variable and find the winner, it also has tilehander() if two candidates have the same vote. Finally, the ShowElectionProgress class will open the audit file if the user wants to see the election progress.

3.2 Decomposition Description

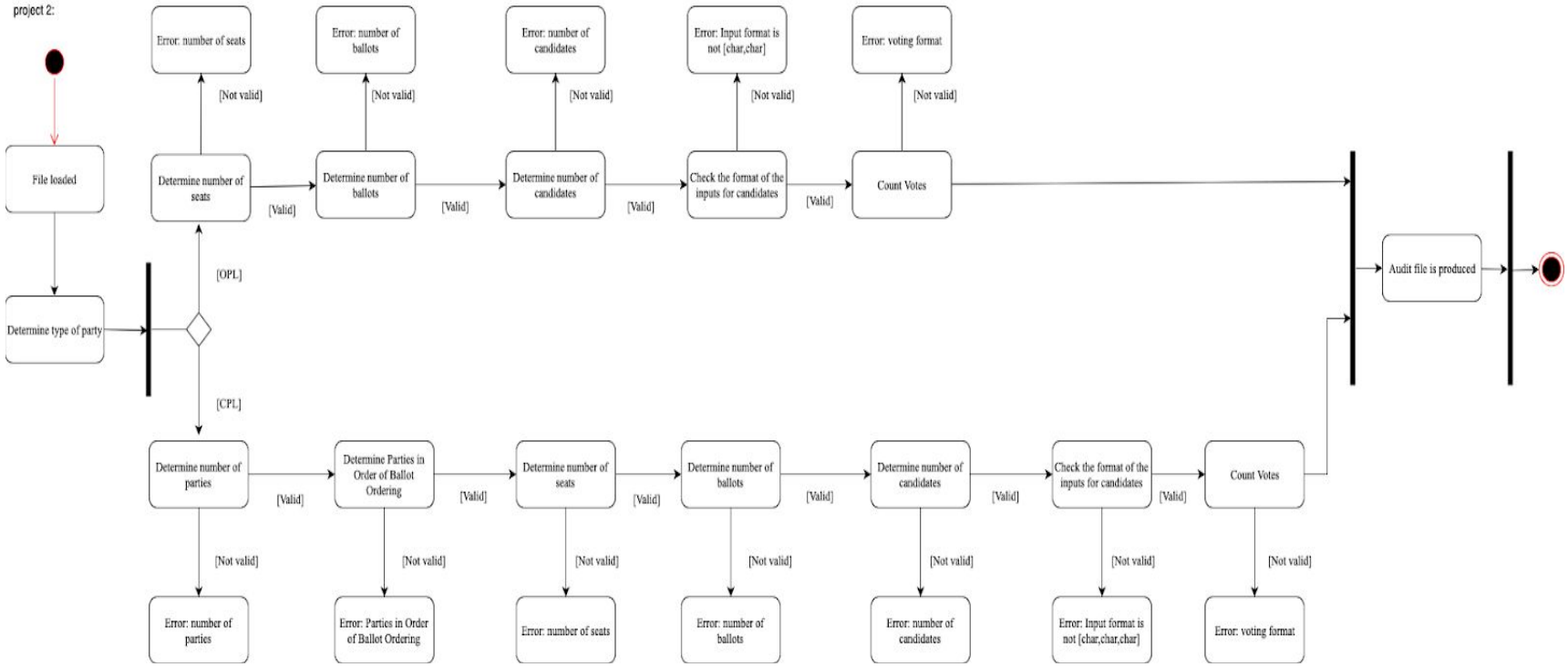
Sequence Diagram for CPL





This sequence diagram describes the general process for a closed party list voting from the process when the user runs the program, then it will prompt the user to enter the filename, and the system will check if the filename exists, if it exists, system will start to read the file and count the votes. Aslo, it shows the alternative courses depend on different cases.

UML Activity diagram for both OPL and CPL


project 2:
File loaded
Determine type of party
[OPL]
[CPL]
Determine number of seats
[Valid]
Determine number of ballots
[Valid]
Determine number of candidates
[Valid]
Check the format of the inputs for candidates
[Valid]
Count Votes
[Not valid]
Error: number of seats
[Not valid]
Error: number of ballots
[Not valid]
Error: number of candidates
[Not valid]
Error: Input format is not [char,char]
[Not valid]
Error: voting format
Audit file is produced
Determine number of parties
[Valid]
Determine Parties in Order of Ballot Ordering
[Valid]
Determine number of seats
[Valid]
Determine number of ballots
[Valid]
Determine number of candidates
[Valid]
Check the format of the inputs for candidates
[Valid]
Count Votes
[Not valid]
Error: number of parties
[Not valid]
Error: Parties in Order of Ballot Ordering
[Not valid]
Error: number of seats
[Not valid]
Error: number of ballots
[Not valid]
Error: number of candidates
[Not valid]
Error: Input format is not [char,char,char]
[Not valid]
Error: voting format

This UML activity diagram shows the process for both open party list voting and closed party list voting. The process will kick off once a file is loaded. Then the system will check which type of election it belongs to and therefore separate from there. For both party lists, the system will check if each line of inputs is in the correct format. If any input is incorrect, the corresponding error message will notice the user. In the end, an audit file will be generated.

Functional Requirements

PromptFile

Name: Prompt File
Type: PromptFile
Description: Ask user to type the filename to read.
Attributes: filename: String
Resources: None
Operations:

Name: confirm()
Arguments: None
Returns: bool
Pre-condition: None
Post-condition: Get user's input as the filename
Exceptions: Filename is empty

Flow of Events:

1. The user type the filename
2. The user click the confirm button

Name: get_file_name()

Arguments: None
Returns: string
Pre-condition: None
Post-condition: return the filename
Exceptions: None

Flow of Events:

1. Return the file name whenever the ReadFile needs to open a file

ReadFile

Name: Read File
Type: ReadFile
Description: System reads the file
Attributes: file_name : PromptFile
Resources: None
Operations:

Name: readfile()
Arguments: None
Returns: ofstream
Pre-condition: prompt file is complete
Post-condition: file is successfully read
Exception: File cannot be found

Flow of Events:

1. The user confirms file name

2. The system searches the file
3. The system reads the file

GenerateAuditFile
Name: Generate Audit File
Type: GenerateAuditFile
Description: This class is used to generate the audit file for further use.
Attributes:
    type_of_election: string
    number_of_Seat: int
    number_of_ballot: int
    number_of_candidate: int
    file: ReadFile
Resource: None
Operations:
    Precondition: File is read successfully
    Postcondition: None
    Exception: Input file is not correct
Flow of event:
1. The system determines if the file format is excel
2. The system determines if the type is OPL or CPL
3. The system reads the attributes relate to OPL or CPL

OPL
Name: OPL
Type: GenerateAuditFile
Description: Generate the audit file for OPL, stores all the information in the variables
Attributes: OPL_Vote: String
Resources: GenerateAuditFile
Operations:
    Name: get_type_of_election()
    Arguments: None
    Returns: string
    Pre-condition: information has been stored correctly
    Post-condition: Get the type of election
    Exception: None
Flow of event:
1. Store type of election information correctly
2. return the type of election
Operations:
    Name: get_number_of_seat()
    Arguments: None
    Returns: int
    Pre-condition: information has been stored correctly
    Post-condition: Get the number of seat
    Exception: None

Flow of event:

1. Stored the information correctly
2. return the the number of seat

Name: get_number_of_ballot()
Arguments: None
Returns: int
Pre-condition: information has been stored correctly
Post-condition: Get the number of ballot
Exception: None

Flow of event:

1. Stored the information correctly
2. return the the number of ballot

Name: get_number_of_candidate()
Arguments: None
Returns: int
Pre-condition: information has been stored correctly
Post-condition: Get the number of candidate
Exception: None

Flow of event:

1. Stored the information correctly
2. return the the number of candidate

Name: get_OPL_Vote()
Arguments: None
Returns: string
Pre-condition: information has been stored correctly
Post-condition: Get the OPL vote
Exception: None

Flow of event:

1. Stored the information correctly
2. return the the OPL vote

Name: generate_audit_file()
Arguments: None
Returns: void
Pre-condition: all information has been stored correctly and file has been read till the end
Post-condition: Get the number of candidate
Exception: None

Flow of event:

1. Stored the information correctly
2. meet the EOF signal
3. generate the audit file

CPL

Name: CPL
Type: GenerateAuditFile
Description: Generate the audit file for CPL, stores all the information in the variables
Attributes: CPL_Vote: String
number_of_parties: int
ballot_ordering: string
Resources: GenerateAuditFile

Operations:

Name: get_type_of_election()
Arguments: None
Returns: string
Pre-condition: information has been stored correctly
Post-condition: Get the type of election
Exception: None

Flow of event:

1. Store type of election information correctly
2. return the type of election

Operations:

Name: get_number_of_seat()
Arguments: None
Returns: int
Pre-condition: information has been stored correctly
Post-condition: Get the number of seat
Exception: None

Flow of event:

1. Stored the information correctly
2. return the the number of seat

Name: get_number_of_ballot()
Arguments: None
Returns: int
Pre-condition: information has been stored correctly
Post-condition: Get the number of ballot
Exception: None

Flow of event:

1. Stored the information correctly
2. return the the number of ballot

Name: get_number_of_candidate()
Arguments: None
Returns: int
Pre-condition: information has been stored correctly
Post-condition: Get the number of candidate
Exception: None

Flow of event:

1. Stored the information correctly
2. return the the number of candidate

Name: get_CPL_Vote()
Arguments: None
Returns: string
Pre-condition: information has been stored correctly
Post-condition: Get the CPL vote
Exception: None

Flow of event:

1. Stored the information correctly
2. return the the CPL vote

Name: generate_audit_file()
Arguments: None

Returns: void
Pre-condition: all information has been stored correctly and file has been read till the end
Post-condition: Get the number of candidate
Exception: None

Flow of event:

1. Stored the information correctly
2. meet the EOF signal
3. generate the audit file

Name: get_number_of_parties()
Arguments: None
Returns: int
Pre-condition: information has been stored correctly
Post-condition: Get the number of parties
Exception: None

Flow of event:

1. Stored the information correctly
2. return the number of parties

Name: get_ballot_ordering()
Arguments: None
Returns: string
Pre-condition: information has been stored correctly
Post-condition: Get the ballot ordering
Exception: None

Flow of event:

1. Stored the information correctly
2. return the ballot ordering

CountVote

Name: Count Vote
Type: Count Vote
Description: system counts the vote of CPL/OPL
Attributes: OPL_Count: OPL
CPL_Count: CPL
Vote_Count: map<int, string>
Resources: OPL/CPL
Operations:

Name: Get_vote_count()

Arguments: None

Returns:map<int, string>

Pre-condition: Audit file is generated

Post-condition: Return the votes

Exceptions: Vote inputs are not valid

Flow of Events:

1. The system reads the audit file
2. The system counts the total votes

Name: Get_Type_of_election()

Arguments: None

Returns: string

Pre-condition: Audit file is generated

Post-condition: return the election voting type

Exceptions: Not CPL or OPL voting

Flow of Events:

1. System read the first line of the audit file

Name: Get_parties()

Arguments: None

Returns: string

Pre-condition: Audit file is generated

Post-condition: return the party name

Exceptions: Invalid party name

Flow of Events:

1. System reads the audit file
2. System determines party name

DisplayWinner

Name: Display Winner
Type: DisplayWinner
Description: The system display the winner's information based on CPL/OPL
Attributes: - count_vote : CountVote
Resources: CountVote
Operations:

Name:display()
Arguments: None
Returns: None
Pre-condition: The votes are counted
Post-condition: Votes are displayed on the screen.
Exceptions: File doesn't belong to OPL or CPL

Flow of Events:

1. System determines the type of voting
2. System display all the information based on OPL or CPL

Operations:

Name: tie_handler()
Arguments: None

Returns: int
Pre-condition: The vote results is a tie, no winner
Post-condition: A winner is decided using flipping coin
Exceptions: None

Flow of Events:

1. The voting result is same
2. System randomly select a winner in a fair coin toss
3. System determines if a winner has a highest votes
4. System decides the winner
5. System displays all the information based on CPL/OPL

ShowElectionProgress

Name: Show Election Progress
Type: PromptFile
Description: Open the audit file to the user if user wants to see the election progress
Attributes: audit_file : GenerateAudiFile
Resources: GenerateAudiFile
Operations:

Name: confirm()
Arguments: None
Returns: bool
Pre-condition: Audit file is generated
Post-condition: Election progress of the audit file is shown
Exceptions: Invalid audit file

Flow of Events:

1. The audit file has been generated
2. The user clicks the confirm button
3. System opens the corresponding audit file

3.3 Design Rationale

We design separate workflows for open party list voting and closed party list voting. Due to the the differences in these two types of files, we think it's rational to break them into different classes to generate audit files. Our design heavily depends on the correctness of input files since they are the only source to proceed the workflow. Moreover, Security is another issue to think about, while the system is running locally without any database. However, we don't need to consider security and database issues since the project requirement does not require us to do such things.

4. DATA DESIGN

4.1 Data Description

The data is stored in a CSV file in a comma separated format. The data can be stored locally with no internet access.

4.2 Data Dictionary

The functions and function parameters are listed in the following table:

Function name	Function parameters
confirm()	None
get_file_name()	None
readfile()	None
get_type_of_election()	None
get_number_of_seat()	None
get_number_of_ballot()	None
get_number_of_candidate()	None
get_OPL_Vote()	None
get_number_of-parties()	None
get_ballot_ordering()	None
get_CPL-Vote()	None
generate_audit_file()	None
tie_handler(): int	None
Get_Type_of_election()	None
display(): void	None
Get_parties(): String	None
Get_vote_count(): map<int, string>	None

The attributes for transmitting to and from the voting system are given in the following table:

Attribute Name	Attribute Type	Attribute Size
Type of election*#	String	3

Number of parties#	Int	4
Number of seats*#	Int	4
Number of ballots*#	Int	4
Number of candidates*#	Int	4
Name of candidates*#	String	30
Name of parties#	String	30
Votes*#	String	30
filename *#	String	30
CPL_Vote ^	String	3
CPL_Vote ^	String	3
OPL_Count ^	OPL	100
CPL_Count ^	CPL	100
count_vote ^	CountVote	100

Fields marked with an '*' are required fields. Fields marked with a '#' can be visible or not visible and is determined by the voting system. Fields marked with a '^' are never visible to anyone other than the voting system.

5. COMPONENT DESIGN

pseudocode for each function listed in section 3.2:

Confirm()

```
if(user_input = yes)
{
  return true;
}
return false;
```

get_file_name()

```
{
if(user_input = filename)
{
  return filename;
}else{
```

```
  return exception;
}
}
```

readfile()

```
{
   filePath = userInput;
   fp = read(userInput);
   return fp;
}
```

get_number_of_candidate()

```
{
    return number_of_candidate;
}
```

get_type_of_election()

```
{
        return type_of_election;
}
```

get_number_of_Seat()

```
{
        return number_of_seat;
}
```

get_number_of_ballot()

```
{
        return number_of_ballot;
}
```

get_number_of-parties()

```
{
        return number_of-parties;
}
```

get_ballot_ordering()

```
{
        return ballot_ordering;
}
```

get_OPL_Vote()

```
{
        return OPL_vote;
}
```

get_CPL_Vote()

```
{
        return CPL_vote;
}

generate_audit_file()
{
        ofstream audit ("aduitfile.CVS");
        audit<<type_of_election;
        audit<<number_of_parties;
        audit<<number_of_ballot;
        audit<<number_of_candidate;
        for(int i=0;i<candidate_name.size();i++)
        {
                audit<<number_of_candidate;
        }
        for(int i=0;i<vote.size();i++)
        {
                audit<<vote;
        }
        audit.close()
}

Get_parties()
{
        return party_name;
}

display()
{
     int max = getMaxCandidate();
     if(max>1)
    {
      tie_handler();
      return;
    }
    else
   {

    cout<< winnderInfo;
   }
}
Get_vote_count()
{
        return vote_count;
}
```

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

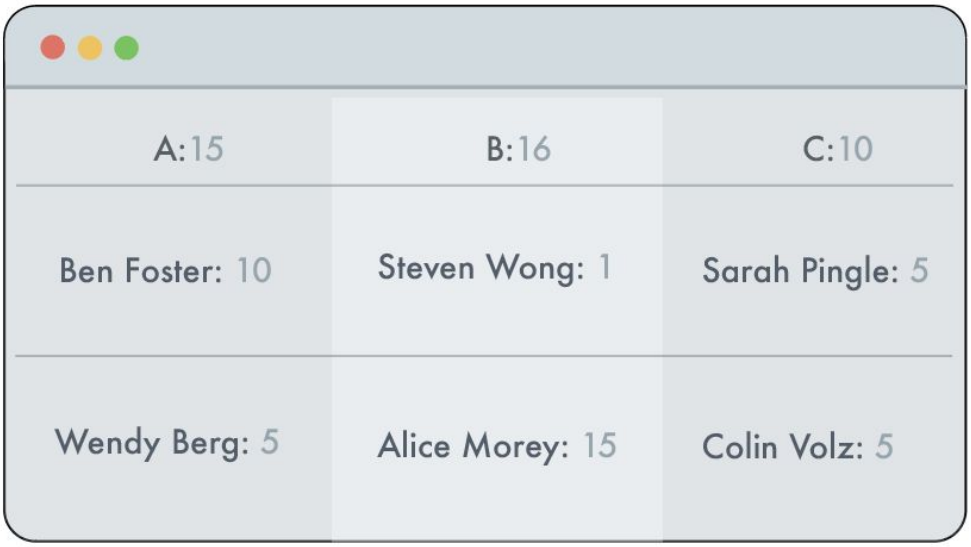
Voting system requires users to provide with their excel file about the voting results, users are able to decide their way of providing files based on their preferences. If they would like to download the file online, they might need Internet connection to download the file, or they can use a USB drive. However, the software itself doesn't need Internet connection to run, in other words, no other communications are required. The Interface includes Prompt filename, generated audit file, show election progress and display winner.

6.2 Screen Images

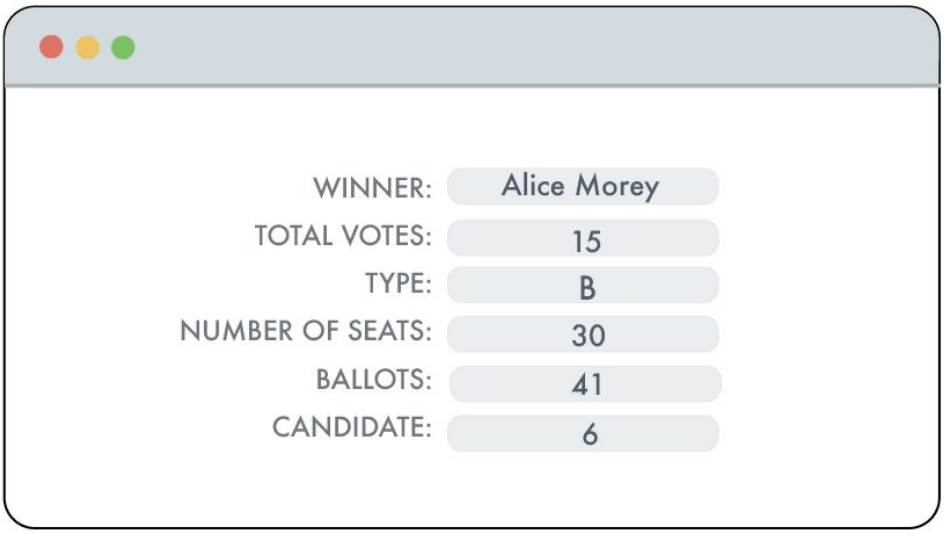
1. Prompt filename

2. Generated audit file

3. Show election progress



4. Display winner



6.3 Screen Objects and Actions

1. Prompt file name

   Users can put in filename in the search text area. Then they can click the “confirm” button to start searching for this file.

2. Generated audit file

   Users can click the “start” button to start the program.

3. Show election progress

   The table will display the information for the election progress, the first row records the parties’ name and votes, the second and third row records the candidates’ name and votes.

4. Display winner

   The result of the file will be displayed to the user.

7. REQUIREMENTS MATRIX

This table shows the cross reference on how our functions satisfy the functional requirements from SRS:

Function name	Functional Requirement
confirm()	Prompt User for filename Show election progress
get_file_name()	Prompt User for filename
readfile()	Read file
get_type_of_election()	Produce audit file for Closed Party List Produce audit file for Open Party List
get_number_of_seat()	Produce audit file for Closed Party List Produce audit file for Open Party List
get_number_of_ballot()	Produce audit file for Closed Party List Produce audit file for Open Party List
get_number_of_candidate()	Produce audit file for Closed Party List Produce audit file for Open Party List
get_OPL_Vote()	Produce audit file for Open Party List
get_number_of-parties()	Produce audit file for Closed Party List
get_ballot_ordering()	Produce audit file for Closed Party List
get_CPL-Vote()	Produce audit file for Closed Party List
generate_audit_file()	Produce audit file for Closed Party List Produce audit file for Open Party List
tie_handler(): int	Display for closed party list Display for open party list
Get_Type_of_election()	Display for closed party list Display for open party list
display(): void	Display for closed party list Display for open party list
Get_parties(): String	Display for closed party list Display for open party list
Get_vote_count(): map<int, string>	Display for closed party list Display for open party list

8. APPENDICES

N/A