Team11

Voting System Application

Software Design Document

Name(s): Arzab Bhattarai, Sam Bakri, Wendell Relacion, and Andy Wang
Section: 001
Date: 02/27/2024

Table of Contents

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the BalletCountPro software. The system will be able to handle OPL and CPL processing. The voting system will be created using the waterfall methodology and will be developed using C++. This document is intended to be used as a blueprint for developers, testers, government officials, and all those involved in the approval or developmental processes of said software.

1.2 Scope

This document contains a complete description of the design of the BalletCountPro system. The primary users of this software will be election officials with proper credentials that will type in the file name into the system to be processed. Its main functionality consists of allocating seats based on the votes for both Open Party List and Closed Party List election types. Additionally, the system will also handle potential ties that may happen amongst candidates through a coin toss process. Once the seat allocation process is finished, the system will produce an audit file and display the winner to the election official. This software aims to account for potential human errors with manually counting votes to ultimately improve efficiency and accuracy of election results.

1.3 Overview

This is a preview of the system details and what it is trying to accomplish. System overview will be given to provide a general summary of the system's functionality. In the next sections, the components of the system will be discussed, then finally the requirements will be listed out and displayed in a matrix that will correspond to the components of the system.

1.4 Reference Material

NONE

1.5 Definitions and Acronyms

1. Aggregation: An object uses another object but it has its own life cycle with ownership to the objects that may have been created
2. Inheritance: allows a class to take properties of another class and use it as its own
3. OPL: Open Party List, a type of election in which a voter places a vote for a specific candidate in a party

4. CPL: Closed Party List, a type of election in which a voter places a vote for the party and the candidate ranked the highest from the party gets the seat
5. Waterfall: Type of software development methodology that behaves in a sequential matter
6. C++: Object Oriented Programming language
7. Component Design: This describes the functionality and also purpose of each component of the overall system: interface, algorithms, and data structures.
8. Human Interface: A type of interface where a human interacts with a system by either inputting data or providing some sort of output
9. GUI: Graphics User Interface
10. Command Line: Interface where users can type in commands, data, and see results from a program
11. CSV: Comma Separated Values
12. Audit File: A file that is produced once the algorithm concludes that contains election data results
13. Pseudocode: Informal overall coding scheme of the overall Voting System

2. SYSTEM OVERVIEW

BallotCountPro is a system that uses the object oriented paradigm to achieve its architecture. Its job is to decide the type of election and determine the winners according to that specific type. It takes in a file that is CSV-formatted which will contain the proper type and its ballot information.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The architectural design of the system is composed of multiple components and are listed and numbered.

1.) Main
2.) Election
3.) Party
4.) OPL_Party
5.) CPL_Party
6.) Candidate

7.) CandidateOPL
8.) CandidateCPL
9.) Ballot
10.) Seat

1.) The main will be used as the function that runs the entire system. This is where the election official will be prompted with writing in the name for the ballot file. The main will introduce a new election object to handle the and parse through the ballot file for the necessary data.
2.) The election component will handle the calculation of the election and map all of the moving components that are inheriting from it accordingly.
3.) The party component will handle the different types of election through the children components OPL_Party and CPL_Party. It will store all of the necessary information so that the component it's under, which is the election component, will be able to do its calculation
4.) Candidate will serve as an abstract class for both CandidateOPL and CandidateCPL. It will have all of the values and methods for the CandidateOPL and CandidateCPL classes to inherit from. CandidateOPL will contain variables and methods from the Candidate abstract class and this applies for CandidateCPL as well.
5.) Ballot object contains getVotes() and SetVotes() which allows the users to get the total votes for a candidate
6.) The Seat object contains getSeats() and setSeats() which allows the users to access the total amount of seats that can be allocated.

3.2 Decomposition Description

To decompose the subsystems in the architectural design, this will be shown in Figure 3.2.1 with the full UML Diagram of the architecture. Figure 3.2.1 will show the whole system which is composed of the CPL and OPL subsystems. Both subsystems will calculate the election according to the type of the election and when that is determined then the appropriate subsystem will handle the allocation of seats.

Figure 3.2.1 UML diagram for system

Figure 3.2.2 CPL Activity Diagram

Figure 3.2.3 OPL Sequence Diagram

3.3 Design Rationale

Design decisions that were made are that parties and candidates will be abstract and that it will be inherited by the specific type of election which are CPL and OPL in this case. This decision was made because we wanted to preserve the SOLID principle.

4. Data Design

4.1 Data Description

The system will store most of the important information in the Election class. We have a Candidate, Ballot, Party, and Seat array where each index in the array will correspond to the indexes in another. For example, The index in Ballot will only correspond to the Indexes of Party. Ballot stores the total votes in a party. Let's say that there are 100 votes for Democrat and 300 votes for Republicans, the first index of Ballot will have 100 votes and the first index of Party will say Democrat while the second index in the ballot array will say 300 and the second index in Party will say Republican.

The Candidate array will be standalone in Election. Having it standalone in elections while being connected to the party will give us a better time in running setRankings(), theoretically.

We will have a dictionary for the candidates in OPL_Party. The name of a candidate will be in the dictionary and it will have a mapped value. As we are getting ballots in, we are adding a vote to the ballot array that corresponds to the Party, and we will also be adding a vote for the candidate that is in the OPL_Party dictionary.

4.2 Data Dictionary

- 4.2.1. Ballot
  - 4.2.1.1. Vote: int
- 4.2.2. Candidate
  - 4.2.2.1. name: string
  - 4.2.2.2. party: Party
- 4.2.3. CandidateCPL (Concrete class)
  - 4.2.3.1. No specific attributes listed
- 4.2.4. CandidateOPL

4.2.4.1. rankingCandidates: int

4.2.5. Election

4.2.5.1. ballots: Ballot[]

4.2.5.2. candidates: Candidate[]

4.2.5.3. electionType: string

4.2.5.4. isTie: Bool

4.2.5.5. sizeOfParties: int

4.2.5.6. numBallots: int

4.2.5.7. numCandidates: int

4.2.5.8. numParties: int

4.2.5.9. numSeats: int

4.2.5.10. parties: Party[]

4.2.5.11. seats: Seat[]

4.2.6. Main

4.2.6.1. election: Election[]

4.2.6.2. fileName: String

4.2.7. OPL_Party

4.2.7.1. votesForCandidate: dict{}

4.2.8. Party <<abstract>>

4.2.9.

4.2.9.1. candidateNames: Candidate[]

4.2.9.2. partyName: String

4.2.10. Seat

4.2.10.1. seats: int

5. Component Design

1.) 5.1 Main

a.) 5.1.1 Main(string[] args): void

i.) args will be used to check for the election filename. If the filename is not given within args then the **promptForFile()** will task the user to enter in a filename

b.) 5.1.2 ReadFile(fileName: string): void

i.) Function that will read in the file once the user types in the file name

c.) 5.1.3 displayResults(): void

i.) Once the program is concluded this function will display the election winner results within the terminal

d.) 5.1.4 produceAuditFile(): void

i.) Writes in the the election data results into a file

e.) 5.1.5 getElection(fileName): Election

i.) Getter method to actually receive the Election object

f.) **5.1.6 setElection(filename): void**
    i.) Sets the Election object based on the filename entered

2.) **5.2 Election**

a.) **5.2.1 getType():string**
    i.) Reads in the file header and sets the election type

b.) **5.2.2 getNumSeats(): int**
    i.) Reads in the number of seats and sets it to numSeats

c.) **5.2.3 getNumBallots(): int**
    i.) Reads in the number of ballots and sets it to numBallots

d.) **5.2.4 getNumParties(): int**
    i.) Reads in the number of parties and sets it to numParties

e.) **5.2.5 getNumCandidates()**
    i.) Reads in the number of candidates and sets it to numCandidates

f.) **5.2.6 getParties(): Party[]**
    i.) Returns the party object array for a given election

g.) **5.2.7 getBallots(): Ballot[]**
    i.) Returns the ballot object array for a given election

h.) **5.2.8 getSeat(): Seat[]**
    i.) Returns the seat object array for a given election

i.) **5.2.9 getCandidate(): Candidate []**
    i.) This function gets the candidate Objects array that is initialized with Candidate objects

j.) **5.2.10 setType(type: string): void**
    i.) This function sets the type of election based on the header information from the ballot file

k.) **5.2.11 setNumSeats(numSeats:int): void**
    i.) This function sets the number of seats that are present in the ballot file

l.) **5.2.12 setNumBallots(numBallots: int): void**
    i.) This function sets the number ballots that are present in the ballot file

m.) **5.2.13 setNumParties(numParties: int): voids**
    i.) This function sets the amount of parties that are present in the ballot file

n.) **5.2.14 setNumCandidates(numCandidates: int): void**
    i.) This function sets the number of candidates that are present in the ballot file

o.) **5.2.15 setParties(ballotLine: string): void**
    i.) This function sets the Party objects

p.) **5.2.16 setBallots(index: string): void**
    i.) This function sets the Ballot objects

q.) **5.2.17 setSeat(): void**
    i.) This function sets the Seat objects

r.) **5.2.18 setCandidate(candidateLine: string): void**
    i.) This function sets the Candidate objects

s.) **5.2.19 createBallots(numParties: int): void**
    i.) This function actively creates the Ballots objects

t.) **5.2.20 allocateSeats()void**
    i.) This function uses the largest remainder algorithm to go through the process of allocating the seats to the respective parties

u.) **5.2.21 setRankings():void**
    i.) This function sets the rankings of the candidates

v.) **5.2.22 tieBreaker(): int**
    i.) This handles the tie using a randomized coin flips and properly allocates the seat to a candidate

w.) **5.2.23 isTieHandler(): bool**
    i.) This function receives a bool value to decide if there is a tie or not

x.) **5.2.24 setSizeOfParties(Parties.size(): int):void**
    i.) This function sets the size of the amount of parties present in the file

3.) **5.3 Seat**
- a.) **5.3.1 getSeats(): int**
    - i.) Gets the amount of seats that are able to be allocated
- b.) **5.3.1 getSeats(): void**
    - i.) Sets the amounts of seats that

4.) **5.4 Ballot**
- a.) **5.3.1 getVotes(): int**
    - i.) Gets the total number of votes for a party
- b.) **5.3.1 setVotes(): void**
    - i.) Sets the total number of votes for a party

5.) **5.5 Party <<abstract>>**
- a.) **OPL_Party**
    - i.) **5.6.5.a getCandidateVote():int**
        - (1) Gets the number of votes per Candidate
    - ii.) **5.6.5.b setCandidateVote():int**
        - (1) Sets the number of votes per Candidate
    - iii.) **5.6.5.b sortCandidateDict(votesForCandidate: dict{}): dict{}**
        - (1) Sorts the Candidate Dictionary by number of votes
    - iiii.) **5.6.5.b setCandidateRankings(): void**
        - (1) Sets Candidate Rankings based on the Candidate Dict
- b.) **CPL_Party**
    - i.) This is a concrete class for Party

6.) **5.6 Candidate <<abstract>>**
- a.) **5.6.1 getName(): string**
    - i.) This function gets the name of the candidate
- b.) **5.6.2 setName(name: string): void**
    - i.) This function sets the name of the candidate
- c.) **5.6.3 getParty(): Party**
    - i.) This function gets the Party of the candidate
- d.) **5.6.4 setParty(): void**

i.) This function sets the Party of the candidate

7.) **5.6.5 CandidateOPL**
   a.) **5.6.5.a getRankingCandidates(): int**
      i.) This gets the Candidates that have been properly ranked
   b.) **5.6.5.b setRankingCandidates(): void**
      (1) This sets the Candidates that have been properly ranked

8.) **5.6.6 CandidateCPL**
   a.) This is a concrete class for Candidate

6. Human Interface Design

6.1 Overview of User Interface

Users will be able to interact with this system through either a Graphical User Interface(GUI), or through the command line on a terminal. In either case, the user will be prompted to type in the name a valid ballot file of csv(Comma-Separated-Values) format that resides within the same directory as the main system. The system will then create a new Audit file with the results of the election. This file will be available for Users to view within the same directory as previously stated.

6.2 Screen Images

Command Line:

```
$ ./BalletCountPro
Please enter the name of a ballot file of CSV format:
```

6.3 Screen Objects and Actions

Election Official Input:

```
$ ./BalletCountPro
Please enter the name of a ballot file of CSV format: filename.csv
```

7. Requirements Matrix

Y-axis numbering reflects the functional requirements listed below whereas the x-axis numbering reflects the ID numbering from section 3.1

REQ-1: Get the file name with 2 mechanisms
REQ-2: Run Open Party List election type
REQ-3: Run Closed Party List election type
REQ-4: Handle tie breaker with a coin flip
REQ-5: Process election file ballots
REQ-6: Use largest remainder approach for seat allocation
REQ-7: Produce the audit file
REQ-8: Extract all info need to run the voting algorithm
REQ-9: Display to the screen the winners and info

	1	2	3	4	5	6	7	8	9	10
REQ-1	X									
REQ-2	X			X						
REQ-3	X				X					
REQ-4		X	X							
REQ-5	X									
REQ-6		X	X	X	X					
REQ-7	X									
REQ-8	X		X			X	X	X	X	X

REQ-9	X									

8. APPENDICES

NONE