Andrew Lin, Jeffrey Tao, Vinay Senthil

Period 3

Final Specification: Alternative Voting System

0. Introduction

In today’s world, most election results are determined using the **first-past-the-post** system, where each voter is only allowed to mark their most preferred candidate on their ballot.

However, this voting system is flawed because it suffers from the spoiler effect. The spoiler effect occurs when the majority of votes are split between two candidates with similar ideologies, allowing a third candidate to take the plurality and win the election, even though the majority would have preferred either of the first two candidates over the third. The alternative voting system solves this problem by allowing voters to rank the candidates.

Initially, only the first-choice candidate of each ballot is considered. After tallying up the counts for the first-choice candidates of each ballot, the candidate with the least number of votes is eliminated. For all voters who chose this candidate as their first choice, their second-choice preferences will now be considered and added to the count. Then, the candidate with the least amount of votes is again eliminated. This process continues until either one candidate reaches a majority, or all candidates have been eliminated.

1. Classes

Candidate

Requirements:

The Candidate class stores information about each candidate: the name of the candidate and the number of votes that candidate has. It also has methods to reset the number of votes of a Candidate to 0 and to increment the vote count by 1

Functional specifications:

* Receive name of Candidate in constructor
* Include vote count variable
* Include method to increment vote count by 1
* Include method to reset vote count to 0
* Include compareTo method that compares two Candidates by vote count

AlternativeElection

Requirements:

The constructor of the AlternativeElection class receives the ballots and candidate names, in the form of a LinkedList of Queues of Candidates ordered according to candidate preferences and an ArrayList of Candidates, respectively. The primary method of the AlternativeElection class, eliminateCandidate(), processes one round of votes and returns an ArrayList of Candidates. More specifically, it tallies up the first choice votes of the ballots by incrementing the votes variable of each Candidate and eliminates the least popular Candidate after sorting the ArrayList of Candidates. This least popular candidate is added to a HashSet that will be checked when processing ballots in future rounds, so that this candidate will be ignored when processing ballots in these future rounds (since it has already been eliminated). The new ArrayList of Candidates excluding the eliminated Candidate is then returned by the method. Should the method detect that a Candidate has reached a majority, the ArrayList returned will be of size 1 and only contain that Candidate.

Functional Specifications:

* Receives ballots and candidate names in the form of a LinkedList of Queues of Candidates ordered according to candidate preferences and an ArrayList of Candidates, respectively
* Tallies up the first choice of each ballot and eliminates the least popular Candidate(s)
* Adds least popular Candidate to HashSet to avoid processing this Candidate in following rounds.

BallotStats

Requirements: BallotStats provides mathematical analysis on the data; more specifically, tables of how many voters preferred a certain Candidate over another, and percentage of voters preferred a certain Candidate over another

Functional Specifications:

* Receives ballots via a LinkedList of Queues of Candidates
* Includes a method to return a 2-dimensional int array counting voter preferences for all pairs of Candidates
* Includes a method to return a 2-dimensional int array of voter preferences by percentage for all pairs of Candidates

Heroku is a class that establishes a connection with our Postgres database and provides methods of setting up a table within it. In addition, the classes functions are also called to upload the votes of individual.

Requirements: Heroku should provide methods that can be used in other classes to retrieve, request, and push values to the database. Any interaction with the database must be routed through the Heroku class. The class checks to make sure duplicate elections are prevented and provides other classes with values from the database.

Functional Specifications:

* createTable: creates an election based on the given name and list of candidates
* Retrieve: returns the list of candidates
* Request: returns the ballots in a 2D array
* Push: pushes an individual’s ballot
* Clear: checks if an inputted election name is being used

3. Frontend - GUI

JSPs

In order to make this voting system user-friendly, we will need separate displays for the user and for the result tally. To do so we’ve used Java Server Pages or JSPs to create webpages so individuals can load our project on their phones and are not limited by the desktop GUI. By doing so, our project utilizes a live URL hosted by Heroku. When each user joins to cast a vote / ballot, they will be using a JSP file currently called voter.jsp. Voter.jsp consists of a servlet that prompts the user with their votes. Of course, all of this vote-casting comes after we enter the names of all available candidates.

The welcome.jsp page open up with greeting text and has two options: vote or create. The user should click vote when an election has been made for them and by clicking the button, a text field will appear, prompting the user to enter the election name. From here they are routed to the voter.jsp page. If they click the create button, the user is prompted to enter the name of the election they would like to create as well as the number of candidates. After clicking submit they are routed to the candidateEntry.jsp page.

The candidateEntry.jsp page is generated by using a for loop to create n number of input divs to enter each of the candidate's’ name. N is requested from welcome.jsp using the requests library. The for loop creates a p tag to state the current candidate number, a input field, and a break line.

The voter.jsp consists of text fields where the candidates are ranked (from 1 to N, where N is at most the number of candidates). The user can choose between clicking the candidates’ buttons in order (which automatically ranks candidates by order of click), or manually entering the choices in the text boxes. This form is presented as a servlet in order to have quick access to the database.

The Results.jsp will display statistics and winners after all votes have been cast, using the Alternative Vote algorithm. This will use a BarChart class, which will display graphically the top chosen candidates at any time. The Plotly API will be used to draw the bar chart. Statistics like margin of error will also be displayed after a winner is chosen.

Servlets

ServletTest: Processing for the create button.

Requirement: Process get and post requests from candidateEntry.jsp

Functional Specifications:

* Creates the database in the Postgres db.
* Prints a specific message on the screen notifying the admin that their election has been created
* Button back to home

Voter: Processing for the voter screen

Requirements: Processes get / post requests from voter.jsp

Functional Specifications:

* Gets the election name, voter name, list of candidates and list of rankings from the jsp file
* Pushes them to the database in the Postgres db.
* Prints a message notifying the voter that the vote was successful
* Button back to home

4. Backend

Database

The typical database for a java application is a structure using PostGres, a flavor of mySQL. Postgres databases are basically huge tables that have applications for quickly querying and storing various data types. Initially when Warowac implemented a mySQL database, the database was locally stored on the machine it ran on using the localhost protocol. To add multiple device, one must create users and approve access to each IP address of the devices, which is not feasible for . This meant it worked perfect for gathering an individual’s vote or if everyone voted on one laptop, but was not usable for multiple devices. The database is hosted on Heroku.

In order to make the desktop java application executable on any laptop, Warowac needed something that connected multiple. We use a cloud database that runs on Heroku, allowing the database to have a unique url. After an individual inputs there vote in the GUI, candidate choices are sent to the database as an arraylist. A Postgres database on Heroku The database is structured as a key pair value with the key the candidate’s name and the value, the three choices. The data is consolidated and then the program loops through the voters querying the rankings for the candidates and creates the percentages for each candidate. The results are presented as a bar graph that represents the percentage of voters for each of the candidates.

In order to make the connection, we used the JDBC module. JDBC stands for the Java Database Connectivity module. In addition, Warowac uses the Heroku Server module which is an addon for JDBC. Finally, we need to import the java.sql library which allows java to interact with Postgres objects. After importing these libraries, we can attempt to make a connection with the server’s username and password. The program uses a try and catch statement to account for the possible exceptions. If the connection is successful, a statement with querying and inserting is created to update the database with changes.

Warowac uses a cloud database to implement real time changes. Event listeners allows the program to update the graphs of percentages after an individual votes.

5. Class Diagram



6. Testing

A JUnit test class will be written, with additional Javadoc documentation for each method to make it easier to understand what is going on.