Andrew Lin, Jeffrey Tao, Vinay Senthil

Period 3

Preliminary 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. Basic classes

Candidate is a class that stores information about each candidate. It includes the name of the candidate as well as a short blurb and the number of votes that candidate has. The list of candidates is the pool that the voters will pick from during the voting process.

Each Ballot is represented by a LinkedList of candidates, where the head is the most chosen candidate and the remaining candidates are linked by preference. Why is this a good design choice? In the Alternative Voting system, at any given time, only the top choice matters for a voter. Therefore, only the first node of the list is important for calculations.

2. Algorithm

The user is first prompted to enter the names of the candidates. As each candidate is being entered into a program, a Candidate object will be created and added to an ArrayList of Candidates. Then, the user will be prompted to enter the ballots. After each ballot (String of rankings) has been entered into the program, it will be converted into a Ballot object, which store a LinkedList with the list of Candidates in the order indicated on the ballot. The Ballots will be stored in an HashSet of Ballots to assist with tallying up the votes. During each round of processing the ballots, the first entry of each Ballot LinkedList will be tallied up and stored in the appropriate Candidate’s voteCount field. Then, the ArrayList of Candidates will be sorted by the value of their voteCount fields. The candidate with the least amount of votes will be eliminated by removing it from the head of the ArrayList of candidates and deleting it from the LinkedList of each Ballot. This process will continue until a Candidate has attained a majority or when there is only one Candidate left.

3. Frontend - GUI

In order to make this voting system user-friendly, we will need separate displays for the user and for the result tally. When each user joins to cast a vote / ballot, they will be using a user class currently called ElectionScreen. ElectionScreen consists of several different panels (classes that extend JPanel). Of course, all of this vote-casting comes after we enter the names of all available candidates.

The VisualBallot class 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.

The ActionButtons are “clear”, “vote”, and “tutorial”. Each of these is intended to make the voting process easier. “Clear” clears all ballot choices up to this point and allows the user to revote. “Vote” casts the ballot and presents the voter with a loading screen until all votes have been tallied. “Tutorial” explains the voting process with a short tuorial.

The CandidateInfo class stores all information about candidates and blurbs, as well as the ballot list for calculation. This will be displayed in the middle of the user screen, so the users can make an educated decision.

The Results class 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. Statistics like margin of error will also be displayed after a winner is chosen.

4. Backend

Database

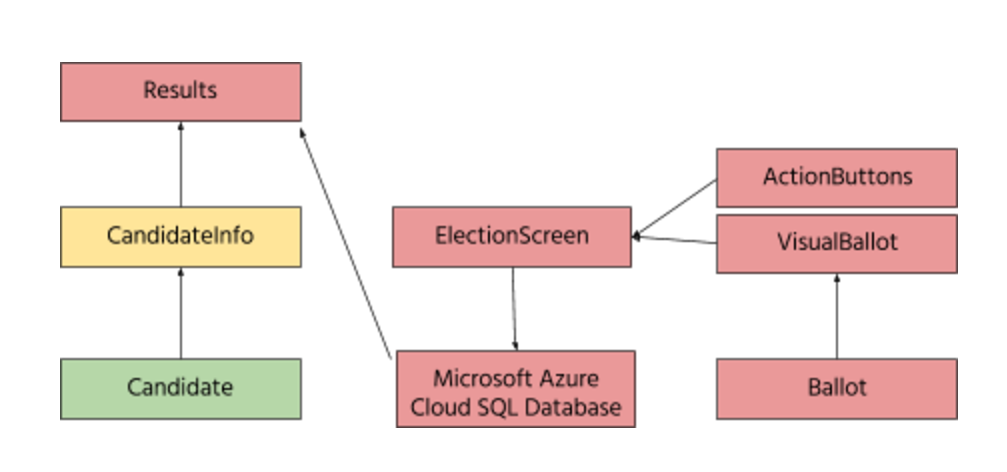
The typical database for a java application is a structure using mySQL. MySQL databases are basically huge tables that have applications for quickly querying and storing various data types. Initially when Warowac implemented a mySQL database, the database was locally stored on the machine it ran on using 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.

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 Microsoft Azure, 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 SQL database on Microsoft 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 SQLServer module which is an addon for JDBC. Finally, we need to import the java.sql library which allows java to interact with SQL 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.