Group 4

Voting Count System (VCS)

Software Design Document

Name (s): Tanner Skluzacek, Tobias Moszer, Jonathan Leibovich, Varun Reddy

Lab Section: CSCI 5801 Section 001

Date: (2/28/2021)

TABLE OF CONTENTS

1. INTRODUCTION **3**
1.1 Purpose 3
1.2 Scope 3
1.3 Overview 3
1.4 Reference Material 4
1.5 Definitions and Acronyms 4

2. SYSTEM OVERVIEW **4**

3. SYSTEM ARCHITECTURE **5**
3.1 Architectural Design 5
3.2 Decomposition Description 8
3.3 Design Rationale 9

4. DATA DESIGN **10**
4.1 Data Description 10
4.2 Data Dictionary 10

5. COMPONENT DESIGN **16**

6. HUMAN INTERFACE DESIGN **22**
6.1 Overview of User Interface 22
6.2 Screen Images 23
6.3 Screen Objects and Actions 28

7. REQUIREMENTS MATRIX **28**

1. INTRODUCTION

1.1 Purpose

The purpose of this Software Design Document is to describe the design details of the Voting Count System or VCS. The intended audience for this document is the programmers and testers that will be creating and maintaining this system.

1.2 Scope

The VCS is a product that is capable of quickly calculating the results of an election, given the voting data. It is able to do this for both Instant Runoff Voting and Open Party List Voting. The goal of this product is to accurately and quickly calculate the results of a given election for two types of voting formulas. The intended use is for election officials to conclude the outcome of an election. Files will also be created to share the results with the media and to audit the validity of the election outcome.

1.3 Overview

This document was written by the creators of the Voting Count System (VCS) to show the design and flow for its main processes.

Section 1 contains the background information required to understand the reasoning behind our design choices.

Section 2 will provide a general overview and description of the VCS. This will include a general description of the program's functionality, user interface, and program limitations.

Section 3 contains a UML class diagram for the VCS, an activity diagram for Open Party List (OPL) voting, an activity diagram for Instant Run-Off (IR) voting, and a sequence diagram for OPL voting. Along with these diagrams will be written explanations that cover everything from the attributes to the flow of the program.

Section 4 will describe how data is transferred, processed, and stored throughout the VCS. It will go into detail about the data structured used relationships between them. It will also contain a section that specifically defines and describes the various types of structures and attributes used in the VCS.

Section 5 will contain pseudo code for all of our methods. This will act as a detailed overview for the VCS's implementation.

Section 6 will have an in-depth look at the user-interface component of the VCS.

Section 7 will serve as a cross-referencing tool for our SRS. It will show that we have met all of the functional requirements listed previously using other components of this Software Design Document.

1.4 Reference Material

<1> - IEEE Template for System Requirement Specification Documents: https://goo.gl/nsUFwy

<2> - FairVote Explanation for Instant Runoff Voting: https://www.fairvote.org/rcv#how_rcv_works

<3> - FairVote Explanation for Party List Voting: https://www.fairvote.org/how_proportional_representation_elections_work

1.5 Definitions and Acronyms

In this document, IR refers to Instant Runoff and OPL refers to Open-Party Listing voting. Along with this, CSV refers to a .csv file or comma separated value file. VCS refers to our software's name which stands for Voting Count System.

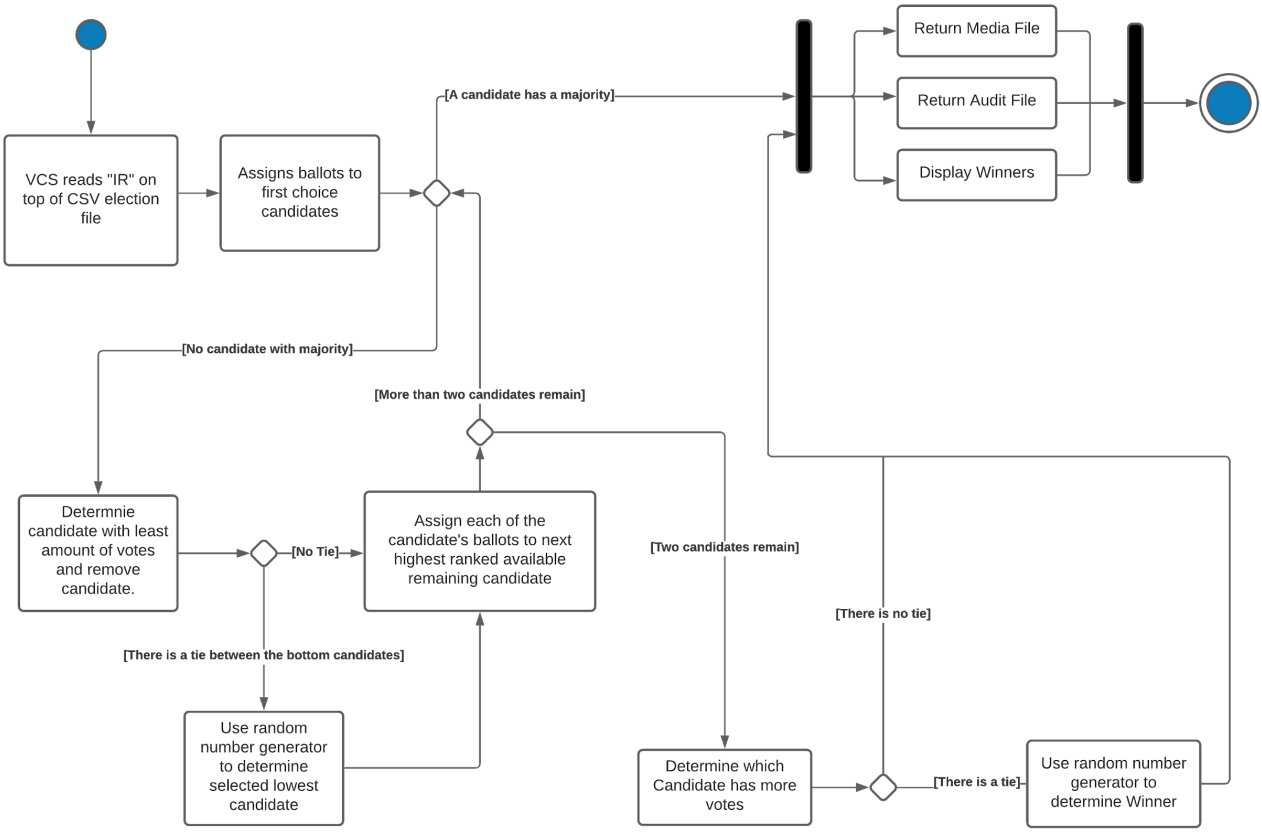
2. System Overview

The Voting Count System (VCS) was developed to be used as a standalone software that is part of a much bigger process. The software requires that the user is comfortable with using the command line and that a CSV formatted ballot file already be in the same directory. The system will then proceed to prompt the user and begin counting the ballots. Currently, the system can run both an Open Party List and an Instant Run-Off election. The program is designed without any security or safety features as that will have already been handled. Once the VCS is done tallying up all of the ballots, it will release three things. First, immediately on the terminal screen the program will display the winners and/or how the seats were allocated. On top of that, the program will create a text file designed to show the media the overall result of the election. Last, the program will release an audit file which can be used to trace how the election was processed, manually. The program is built to handle any tie internally. It uses a random number generator to ensure a fair outcome. Although VCS can be used multiple times, it can only run one ballot at a time. The VCS was designed to process 100,000 ballots in under eight minutes.

3. System Architecture

3.1 Architectural Design

3.1.1 IR Activity Diagram



Description: This will occur when an Instant Run-Off election ballot is selected to be run in the program. There are no specific actions that the Election Officials or the Media members can take during this process except collect the files generated throughout it. The program will run independently and without any errors.

Actors: System, Election Officials, Media Members

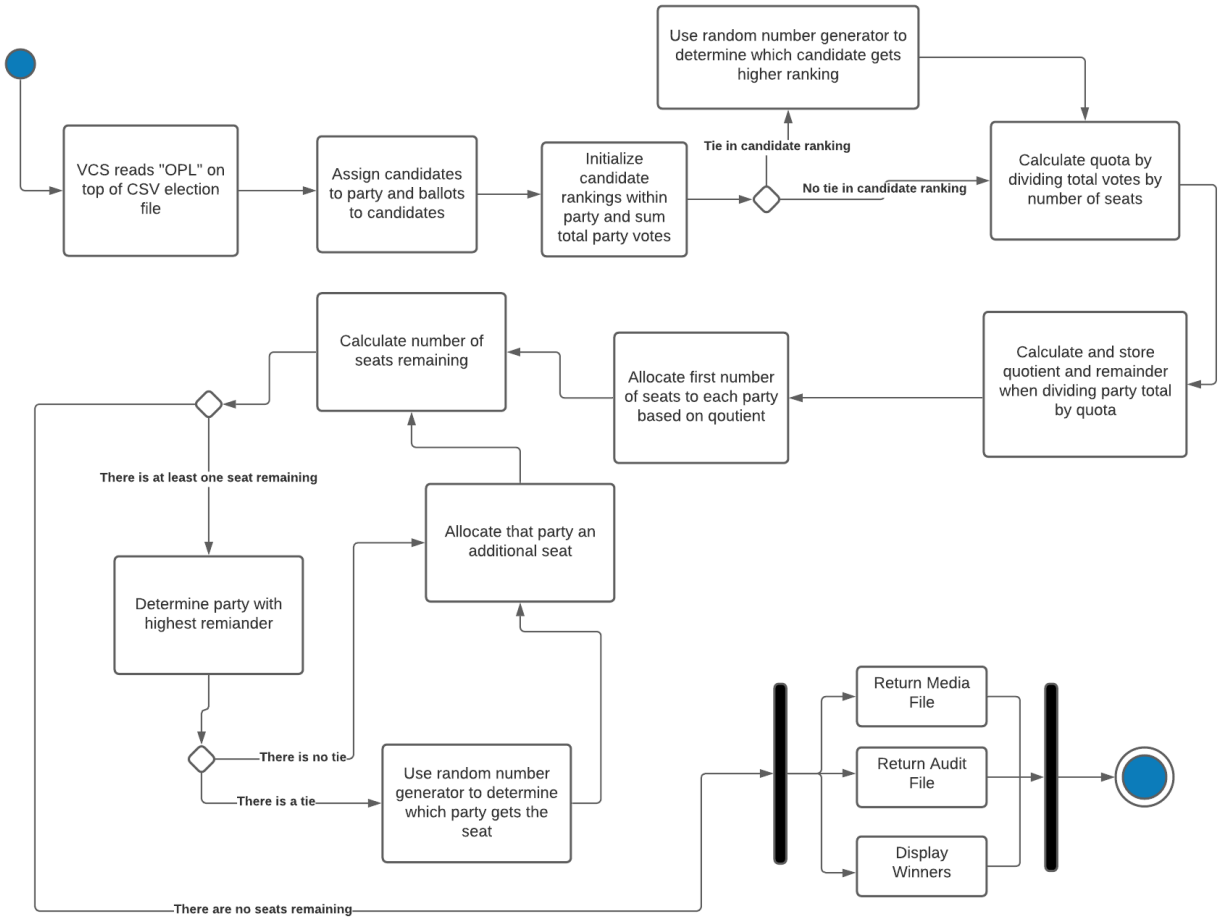
Type: Program

More Detailed Flow of Events:

1. Program initializes if the ballot file indicates it is an IR election.
2. The program will create a candidate object for each candidate in the election
3. All ballots will be allocated into the ballot array and looped through to assign each candidates vote count attribute to the correct value
4. After all votes have been initially tallied, check if any candidate has a majority of votes

5. If a candidate has a majority, proceed to step 13
6. Determine which remaining candidate has the least amount of votes
7. If there is a tie, use the random number generator to determine which candidate gets removed
8. Allocate all of the removed candidates ballots to the loserBallots ballot array
9. Re-increment the votes in the loserBallots to available candidates
10. If there are only two candidates left, determine which candidate has more votes and proceed to step 13
11. If there are only two candidates left and they have the same number of votes, use the random number generator to determine which candidate gets removed and proceed to step 13
12. It there are more than two candidates left, proceed to step 5
13. Return the generated files created and display the winning candidate on the screen

3.1.2 OPL Activity Diagram



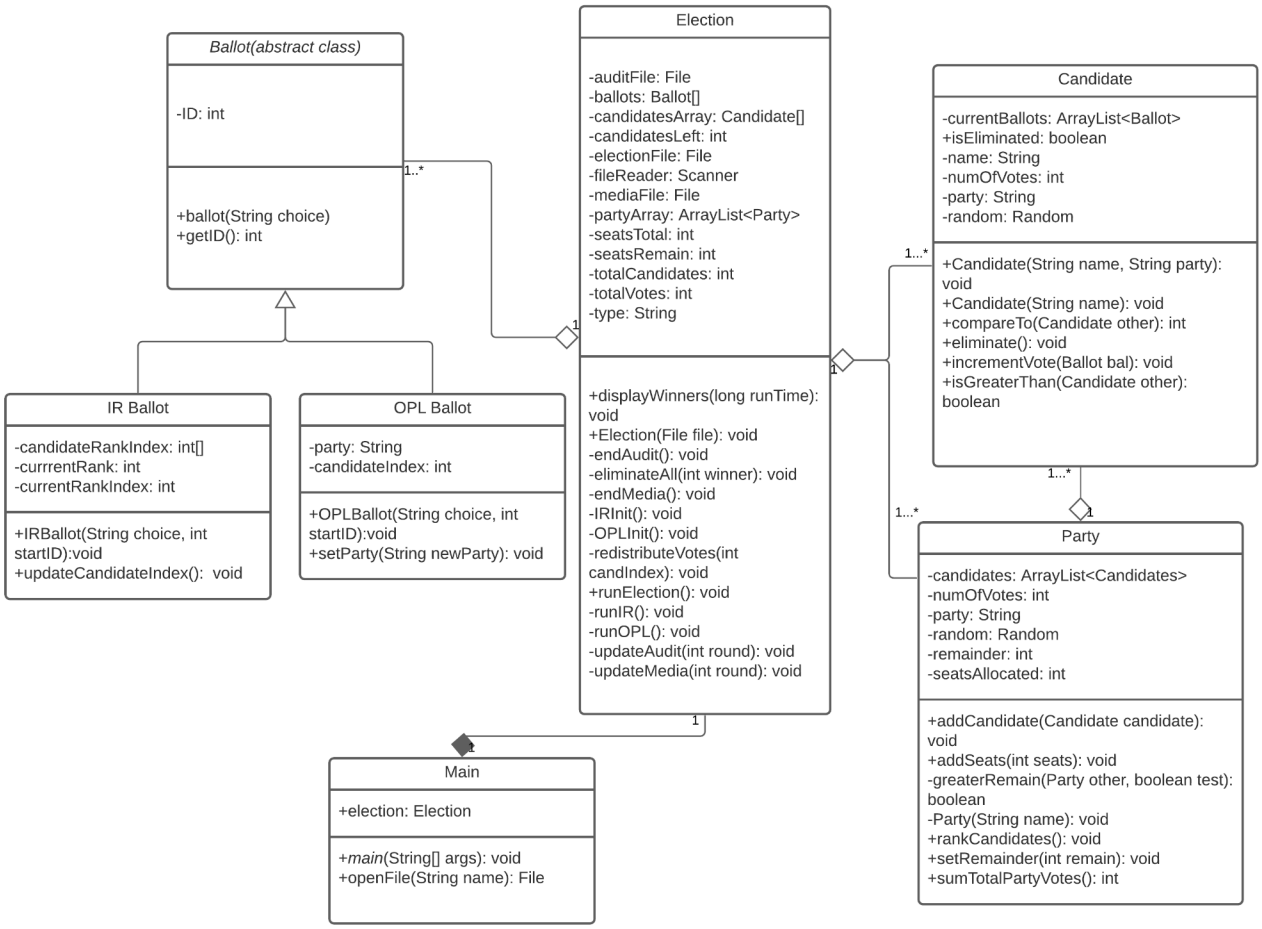
Description: This activity will occur when an Open Party List election ballot is selected to be run in the program. There are no specific actions that the Election Officials or the Media members can take during this process except collect the files generated throughout it. The program will run independently and without any errors.

Actors: System, Election Officials, Media Members
Type: Program
More Detailed Flow of Events:

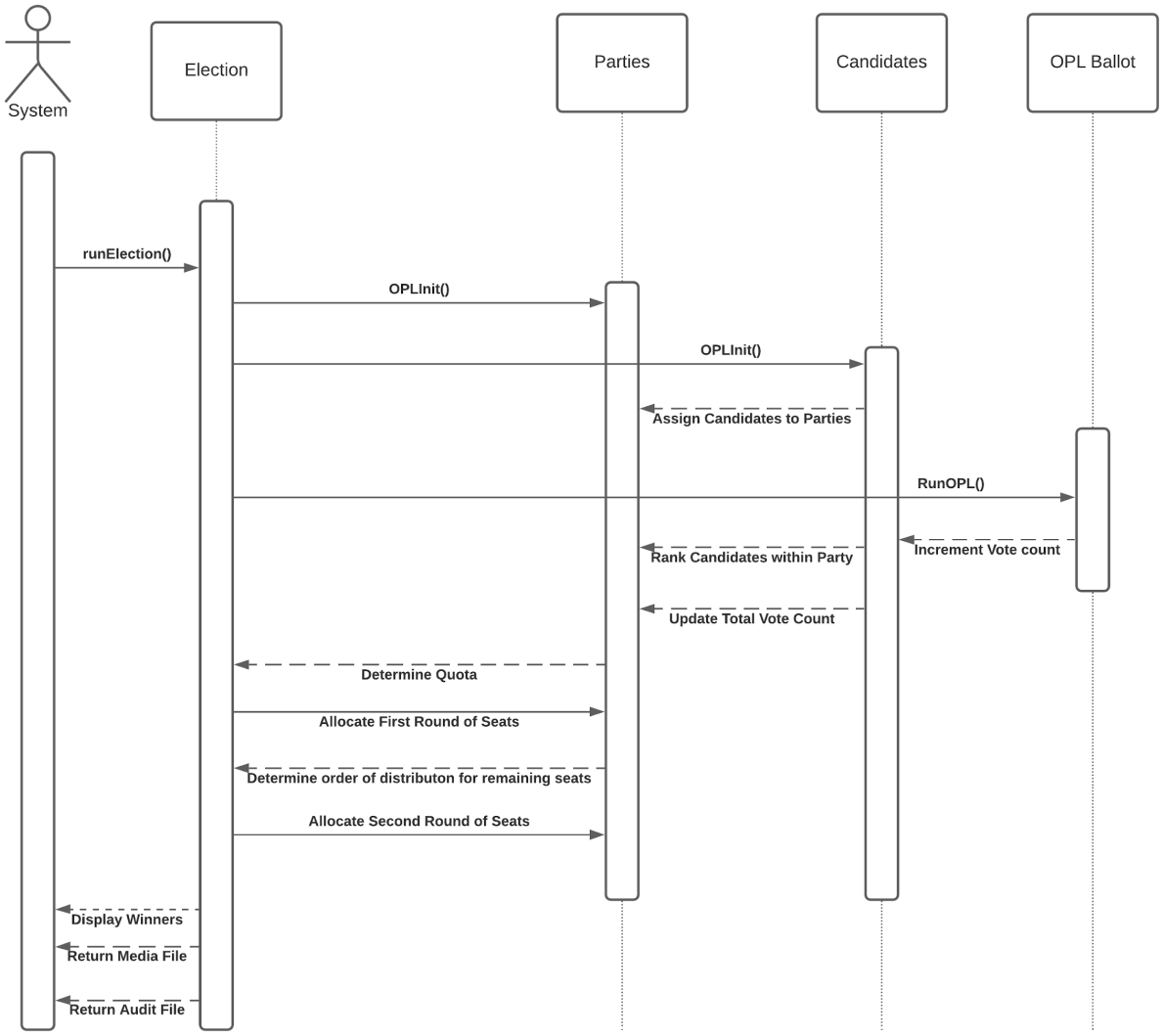
1. Program initializes that it is an OPL election by reading the first line of the election results CSV file.
2. Program will read in the candidates and create the relevant party classes and place the candidates within their corresponding party class.
3. Program will go line by line through the file to assign ballots to the candidate that the ballot was for.
4. Within each party, the candidates will be ranked in order of most ballots to least.
5. If there is a tie in candidate ranking within a party, a coin flip or random number generation will determine the higher candidate.
6. The quota(Q) will be determined by dividing the total number of votes by the number of seats that are open.
7. For each party, the total number of votes for candidates in that party will be divided by Q and the quotient and remainder will be stored for that party.
8. Each party will initially be allocating the number of seats corresponding to the quotient of Step 7.
9. The remaining number of seats will be calculated by taking the initial number of seats subtracted by the number of seats already allocated in Step 8.
10. If there is at least one seat remaining the party with the highest remainder from step 7 will be given an additional seat. If there is not a seat remaining proceed to Step 13.
11. If there is a tie for the highest party, a coin flip will determine the selected party.
12. The selected party will then be removed from the race, remaining seats will decrement by 1 and return to step 10 until there are no more seats remaining.
13. For a party that was allocated n seats, the top n candidates of that party will be elected and this information will be displayed to the election official.

3.2 Decomposition Description

UML Class Diagram:



OPL Sequence Diagram:



3.3 Design Rationale

UML Class Diagram:

We decided to have the Election class take care of all major functionality, instead of for instance having the candidates handle their own ballots. This simplifies things greatly for when ballots must be reallocated if the first choice was eliminated in IR. The ballot, candidate, and party classes just contain information about themselves, they don't actually do any of the hard work. One possible downside for structuring the election this way is that the election class could get very convoluted, as the runIR and runOPL methods will contain most of the code.

OPL Sequence Diagram:

This diagram shows the overall flow of how the VCS goes through the Open Party List Election. There are no other actors involved in running the counting algorithm other than the system itself. You can read the diagram as follows:

1. The system calls runElection() on an instantiated Election file
2. The Election would then call OPLInit() and instantiate the Parties
3. OPLInit() would then instantiate the Candidates
4. Candidates would then be assigned to their respective parties
5. RunOPL() would then be called and look at all of the ballots
6. All of the votes would be correctly incremented to for each candidate
7. All of the candidates would be ranked within the party, if there is a tie a random number generator will be used to determine the outcome
8. The total number of votes for a party would be updated
9. The quota would be determined for the election
10. The first round of seats would be allocated accordingly
11. Using the remainders, the order of distribution for the second round would be determined. If there is a tie a random number generator will be used to determine the outcome
12. The second round of seats would be allocated accordingly
13. The election would display the winners to the System
14. The election would return the media file to the System
15. The election would return the media file to the System

4. DATA DESIGN

4.1 Data Description

All major data is stored within the Election class. Ballots are initialized and stored as ballot objects within an array, and the type of ballot depends on the type of election. Copies of the ballots are also stored within the candidates they represent, with the possibility of moving to a different candidate if their first choice is eliminated (IR only). Candidates and parties are stored in arrays similar to ballots within the election class. Note that this party array is only used for OPL elections.

4.2 Data Dictionary

Election Class:

Attribute Table

Name	Type	Description
auditFile	File	The file the audit information will be written to

ballots	Ballot[]	All the ballots in the election
candidatesArray	Candidate[]	All the candidates in the election
candidatesLeft	int	The number of candidates that have not been eliminated
electionFile	File	The given file containing all election information
fileReader	Scanner	The scanner that is reading in the election file
mediaFile	File	The file that will be given to the media containing election result information
partyArray	ArrayList<Party>	All the parties represented in the election
seatsTotal	int	The number of seats to be filled
seatsRemain	int	The number of seats remaining to be filled
totalCandidates	int	The total number of candidates in the election
totalVotes	int	The total number of votes in the election
type	String	The type of election

Method Table

Name and Parameters	Return Type	Description
displayWinners(long runTime)	void	Displays final election information to terminal using print, takes in parameter runTime that shows how long the election was running
Election(File file)	void	Constructor that takes in a opened file containing election data

endAudit()	void	Writes final election information into the audit file
eliminateAll(int winner)	void	Eliminates all candidates except the winner
endMedia()	void	Writes final election information into the media file
IRInit()	void	Preprocesses data from the Election file and initializes attributes necessary for an IR election
OPLInit()	void	Preprocesses data from the Election file and initializes attributes necessary for an OPL election
redistributeVotes(int candidateIndex)	void	Transfers ballots from eliminated candidate to next candidate for each respective ballot in the candidates' array, uses parameter candidateIndex to find the eliminated candidate
runElection()	void	Starts election process by reading whether the election is IR or OPL
runIR()	void	Facilitates the simulation of a instant runoff election
runOPL()	void	Facilitates the simulation of a open party list election
updateAudit(int round)	void	Writes into audit file with election progression information
updateMedia(int round)	void	Writes into media file with election progression information

Candidate:

Attributes:

Name	Type	Description
currentBallots	ArrayList<Ballot>	The ballots that voted for this candidate
isEliminated	boolean	If this candidate has been eliminated or not
name	String	The name of the candidate
numOfVotes	int	The number of votes that this candidate has
party	String	The name of the party this candidate is associated with
random	Random	Instance of Random to determine a tie

Methods:

Name and Parameters	Return Type	Description
Candidate(String name, String party)	void	Constructor that initializes the name and party of the candidate
Candidate(String name)	void	Constructor that initializes only the name of the candidate
compareTo(Candidate other)	int	Returns the difference between the other candidate and this candidate, if a tie, a random choice of the two
eliminate()	void	Eliminates the candidate by updating the isEliminated attribute and setting numOfVotes to -1
incrementVote(Ballot bal)	void	Increments the numOfVotes attribute and adds the ballot to currentBallots

isGreaterThan(Candidate other)	boolean	Overrides > operator for checking if one candidate has more votes than another

Party:

Attributes:

Name	Type	Description
candidates	ArrayList<Candidate>	The candidates associated with the party
numOfVotes	int	The number of votes within the party
party	String	The name of the party
random	Random	An instance of Random to determine a tie
remainder	int	The remaining votes the party has after the quota
seatsAllocated	int	The number of seats allocated to the party

Methods:

Name and Parameters	Return Type	Description
addCandidate(Candidate candidate)	void	Adds a candidate to the candidatesArray
addSeats(int seats)	void	Add seats to the number of seats allocated
greaterRemain(Party other)	boolean	Calculate if this party's number of votes is greater than the other's
Party(String name)	void	Constructor that sets the name of the Party to name
rankCandidates()	void	Ranks the order of the of candidates based on the number of votes they have

setRemainder(int remain)	void	Set reminder to remain
sumTotalPartyVotes()	int	Sum the total votes from every candidate in the party

Ballot:

Attribute:

Name	Type	Description
ID	int	Unique Ballot ID

Methods:

Name and Parameters	Return Type	Description
ballot(String choice)	void	Ballot Constructor with input of the choices directly from the file
getID()	int	Returns ID of the ballot

IR Ballot:

Attribute:

Name	Type	Description
candidateRankIndex	int[]	Array with a ballots rankings of all the candidates
currentRank	int	What choice the ballot is on
currentRankIndex	int	Index where the best current candidate is in the candidate array

Methods:

Name and Parameters	Return Type	Description
IRballot(String choice, int startID)	void	Constructor that sets the ballots ID and the ballots candidates preference
updateCandidateIndex()	void	Updates currentRank and

		currentRankIndex

OPL Ballot:

Attributes:

Name	Type	Description
candidateIndex	int	The index of the candidate that the ballot voted for
party	String	The name of the party that the candidate voted for represents

Methods:

Name and Parameters	Return Type	Description
OPLBallot(String choice, int startID)	void	Constructor that parses the choice string for the vote, and sets the candidate index to the index of the vote
setParty(String newParty)	void	Sets party to newParty

Main:

Attributes:

Name	Type	Description
election	Election	The election object used to run the election

Methods:

Name and Parameters	Return Type	Description
main(String [] args)	void	The executable that can determine the outcome of an

		election
openFile(String name)	File	Opens a file and ensures the file exists

5. Component Design

5.2.1 Election Class

5.2.1.1 Election(file)

```
electionFile = file
this.fileReader = new Scanner(election file)
```

5.2.1.2 redistributeVotes(candIndex)

```
candidate eliminated state is set to true
for(i = 0 to (size of # of ballots in the eliminated candidate's current ballot arraylist))
        v = index of next ranked choice candidate of ballot that is being transferred
        Using i as an index a ballot from the eliminated candidate is added to the
        next choice candidate's currentBallots array list using incrementVote
this.candidatesLeft--
set numOfVotes of eliminated candidate to 0
```

5.2.1.3 displayWinners(time_taken)

```
print("Election Completed in %ld seconds\n", time_taken)
print("Elected Official(s):\n")
for i=0 to length of totalCandidates:
        If !candidatesArray[i].eliminated:
                If (first line of election file == "IR"):
                        print("[%s]",  candidatesArray[i].name)
                Else:
                        (print("[%s, %s]",  candidatesArray[i].name,
                    candidatesArray[i].party)
print("The media file will be saved under ./%s", mediaName)
print("The audit file will be saved under ./%s", auditName)
```

5.2.1.4 updateAudit(round)

```
if (first line of election file == "IR")
        If (round ==1)
                write headers for candidate information to audit file, including round
                of vote distribution on the audit file
        for(i = 0 to candidatesArray.length)
                write(candidatesArray[i].name)
                write("Votes: %d", candidatesArray[i].numOfVotes)
```

```
                For (j=0 to candidatesArray[i].currentBallots.length):
                        write(candidatesArray[i].currentBallots[j].ID)
        write("Summary:\n")
        for(i = 0 to candidatesArray.length)
                write("Candidate   Total Votes")
                write(candidatesArray[i].name)
                write(candidatesArray[i].numOfVotes)
else if(first line of election file == "OPL")
        If (round ==1)
                write headers for candidate information to audit file, including round
                of vote distribution on the audit file
        for(i = 0 to candidatesArray.length)
                write(candidatesArray[i].name)
                write("Votes: %d", candidatesArray[i].numOfVotes)
                For (j=0 to candidatesArray[i].currentBallots.length):
                        write(candidatesArray[i].currentBallots[j].ID)
        write(Summary:)
        for(i = 0 to partyArray.length)
                write(partyArray[i].party)
                write("Votes: %d", partyArray[i].numOfVotes)
        for(i = 0 to candidatesArray.length)
                write(candidatesArray[i].name)
                write("Votes: %d", candidatesArray[i].numOfVotes)
```

5.2.1.5 runElection()

```
S = use fileReader to read first line
this.auditFile = new file()
this.mediaFile = new file()
if(s = "OPL")
        OPLInit()
        runOPL()
else
        IRInit()
        runIR()
```

5.2.1.6 runIR()

```
for(int i = 0 to totalVotes-1)
        int currentRank = ballots[i].getCurrentRankIndex()
        Index into candidateArray using currentRank and use incrementVote to add
        ballot to the candidate
Write headers for candidate information to audit file, including round of vote
distribution
for(i = 0 to totalCandidates-1)
```

```
        Write candidateArray[i]'s name, then party, then votes on the audit file
    while(no candidate has majority or there are more than two candidates left)
        find the candidate or candidates with lowest votes using a find min algorithm
        if there is a tie for lowest use random number generator to decide loser
        store old vote counts for each candidate in an array to use for audit file
        redistributeVotes(candidate object with lowest votes)
        see if any candidate has majority by looping the candidates
        updateAudit(old vote count array, # of while loops +1)
    if  (there are two candidates and neither has majority)
        the candidate with more votes is the winner
    else if (there are two candidates left and there is a tie)
        use random number generator to decide the victor
    else
        candidate with majority is victor
    endMedia()
```

5.2.1.7 runOPL()

```
    quota = this.totalVotes/this.seatsTotal
    for(i = 0 to size of this.partyArray-1)
        this.partyArray[i].addSeats(this.partyArray[i].sumTotalPartyVotes/quota)
        this.partyArray[i].setRemainder(this.partyArray[i].sumTotalPartyVotes%
        quota)
        this.seatsRemain -= this.partyArray[i].sumTotalPartyVotes/quota
    while(this.seatsRemain > 0)
        use find maximum algorithm to find the index (maxInd) of the party with
        max remainder by looping through the parties use greaterRemain() function
        to compare
        this.partyArray[maxInd].addSeats(1)
        this.partyArray[maxInd].setRemainder(0)
        this.seatsRemain --
    updateAudit()
    endMedia()
```

5.2.1.8 endMedia()

```
    output = new file
    if election_type is OPL:
        write election type, candidates, total votes, seats available to top lines
        write total votes per party to file
        write number seats for each party to file
        write candidate vote totals for each party to file
        write elected candidates to file
    else:
        write election type, candidates, total votes to top lines
        for each count round:
            write candidate totals before and after transfer to file
```

```
            write winning candidate to file
        save file under MediaFile_[date].txt
```

5.2.1.9 IRInit()

```
        numCandidates = read second line of election file
        this.totalCandidates = numCandidates
        this.candidatesLeft = numCandidates
            candLine = read third line of election file
        cands = split candLine by ","
        this.candidatesArrray = new Candidate[numCandidates]
        for(i = 0 to numCandidates-1)
            this.candidatesArray[i] = new Candidate(
        this.totalVotes = read fourth line of election file
        this.ballots = new Ballot[this.totalVotes]
        for(i = 0 to numBallots-1)
            this.ballots[i] = new IRBallot(read next line of election file, i)
```

5.2.1.10 OPLInit()

```
        numCandidates = read second line of election file
        pairedArray = split third line of election file by ","
        this.candidatesArrray = new Candidate[numCandidates]
        this.partyArray = new ArrrayList<Party>
        for(i = 0 to (size of pairedArray-1))
            currPair = split pairedArray by ","
            currCand= currPair[0]
            currParty = currPair[1]
            this.candidatesArray[i] = new Candidate(currCand, currParty)
            create a new party object only if it is a new party
            add the candidate to the party
        this.seatsTotal = read fourth line
        this.totalVotes = read fourth line of election file
        this.ballots = new Ballot[this.totalVotes]
        for(i = 0 to numBallots-1)
            this.ballots[i] = new IRBallot(read next line of election file, i)
        for(i = 0 to size of this.partyArray-1)
            use quicksort on this.partyArray[i].candidates array list using isGreater than
            function
```

5.2.2 IRBallot class

5.2.2.1 IRBallot(string choices, startID)

```
            splitchoices = split the choices string by "," into an array
            candidateRankIndex = new int[size of splitchoices]
            for(i = 0 to (size of splitchoices-1))
                if(splitchoice[i] == "")
```

```
                candidateRankIndex[i] = -1
            else if (splitchoice[i] == “1”)
                candidateRankIndex[i] = typecast splitchoice[i] to int
                currentRankIndex = i
            else
                candidateRankIndex[i] = typecast splitchoice[i] to int
        this.currentRank = 1
        this.ID = startID
```

5.2.2.2 getNextCandidate()

```
        this.currentRank+=1
        for(i = 0 to (size of candidateRankIndex - 1))
            if(candidateRankIndex[i] == currentRank)
                currentRank = i;
                return i
        currentRank = -1;
        return -1
```

5.2.2.3 getCurrentRankIndex()

```
        return currentRankIndex
```

5.2.3 Candidate class

5.2.3.1 Candidate(String name, String party)

```
        this.name = name
        this.party = party
```

5.2.3.2 Candidate(String name)

```
        this.name = name
```

5.2.3.3 isGreaterThan(Candidate other)

```
        if(this.numOfVotes > other.numOfVotes)
            return 1
        else if (this.numOfVotes = other.numOfVotes)
            using random number generator determine winner
            if(this.numOfVotes wins)
                return 1
            else
                return -1
        else
            return -1
```

5.2.3.4 incrementVote(Ballot bal)

```
        this.numOfVotes += 1
        this.currentBallots.add(bal)
```

5.2.4 Party Class

5.2.4.1 greaterRemain(Party other)

```
if(this.numOfVotes > other.numOfVotes)
    return true
else if(this.numOfVotes = other.numOfVotes)
    Use random number generator to select winner
    if(this.numOfVotes wins)
        return true
    else
        return false
else
    return false
```

5.2.4.2 sumTotalPartyVotes()

```
this.numOfVotes = 0
for(i = 0 to size of (candidates-1)
    this.numOfVotes += candidate[i].numOfVotes
return numOfVotes
```

5.2.4.3 addSeats(int seats)

```
this.seatsAllocated += seats
```

5.2.4.4 setRemainder(remain)

```
this.remainder+=remain
```

5.2.5 OPLBallot Class

5.2.5.1 OPLBallot(String choice, startID)

```
splitchoices = choice.split(split wherever “,” appears)
for(i = 0 to splitchoices.size)
    if(splitchoices[i] == “1”)
        this.candidateIndex = i
this.ID = startID
```

5.2.6 Main class

5.2.6.1 main(string[] args)

```
startTime = current time in milliseconds
fileName = args[argument index]
file = openFile(fileName)
election = new Election(file)
election.runElection()
endTime = current time in milliseconds
runTime = endTime - startTime
```

```
election.displayWinners(runTime)
```

5.2.6.2 openFile(String fileName)

```
file = open File(fileName)
Catch (fileNotFoundException)
        return fileNotFound
return file
```

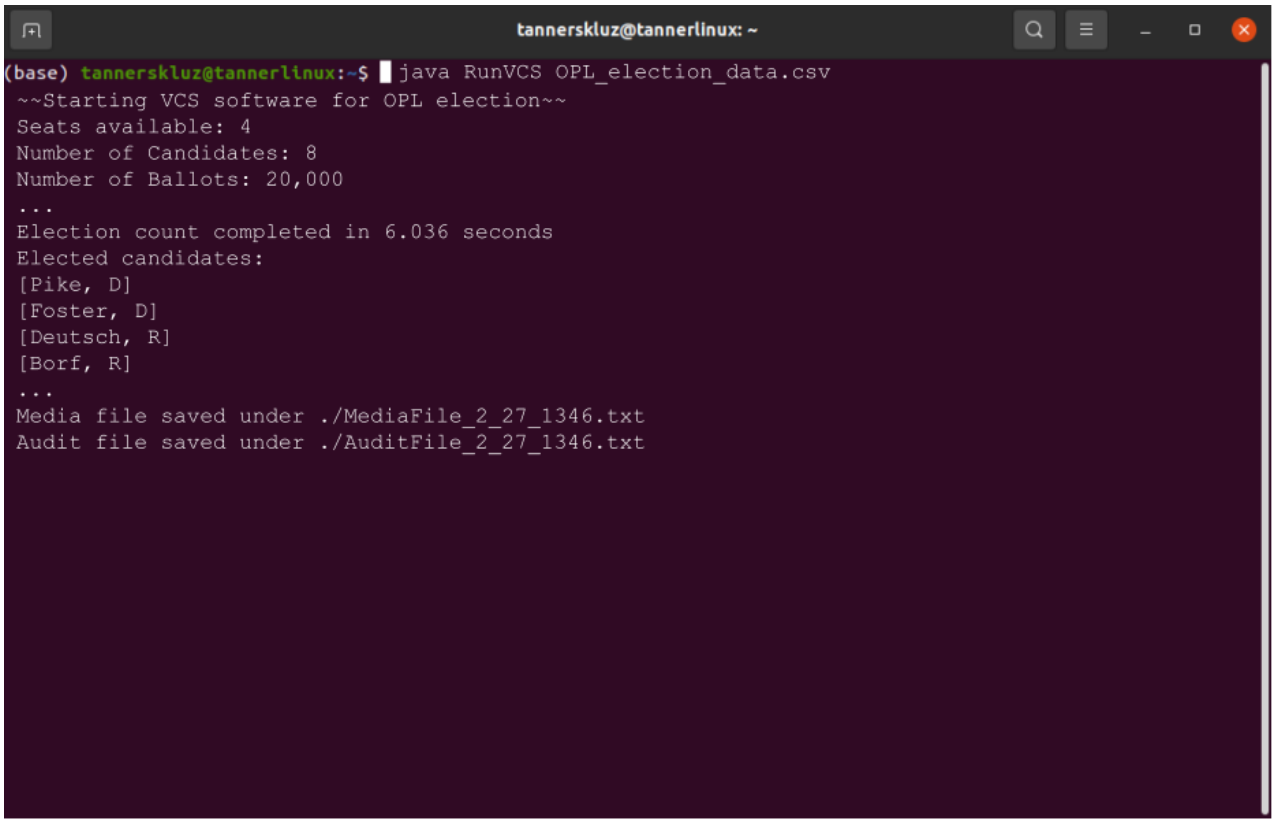
6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

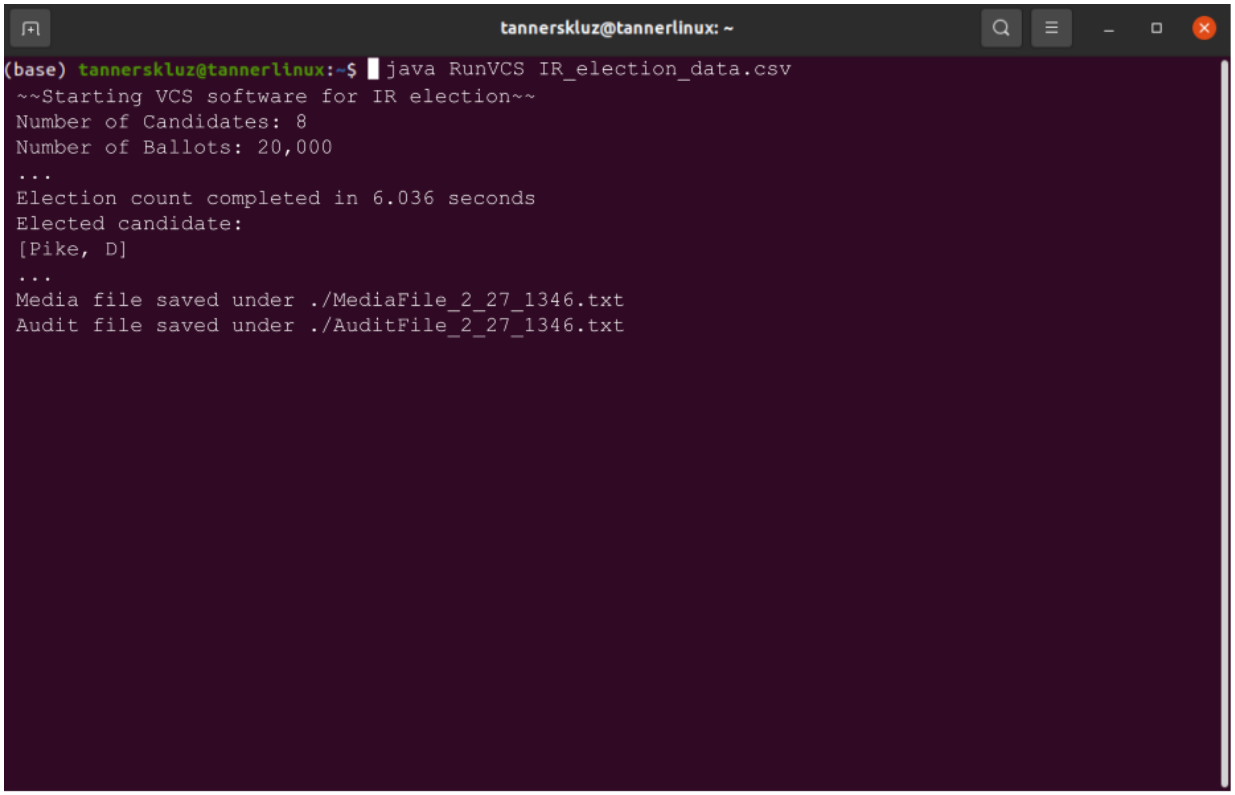
Users will interact with the VCS software through the terminal for linux and MacOS and the command line or powershell for Windows. The user will be able to run the VCS software on an election data file by placing that file in the same directory and passing its file name into the command line arguments. Feedback will be displayed to the user if the file is correctly found. After the algorithm has run, feedback will also be displayed regarding the election summary. Along with this, an audit and media file will be created by the software that can be accessed by the user to get a more in-depth result summary.

6.2 Screen Images

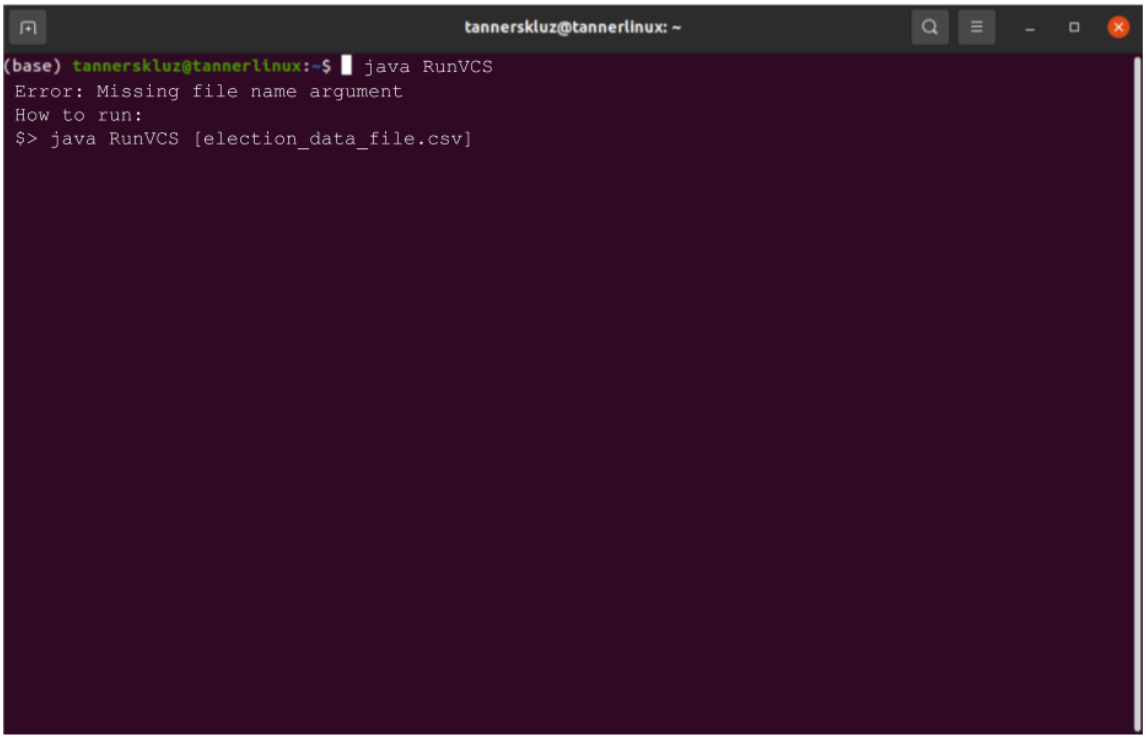
Example Interface for Successful OPL Election:



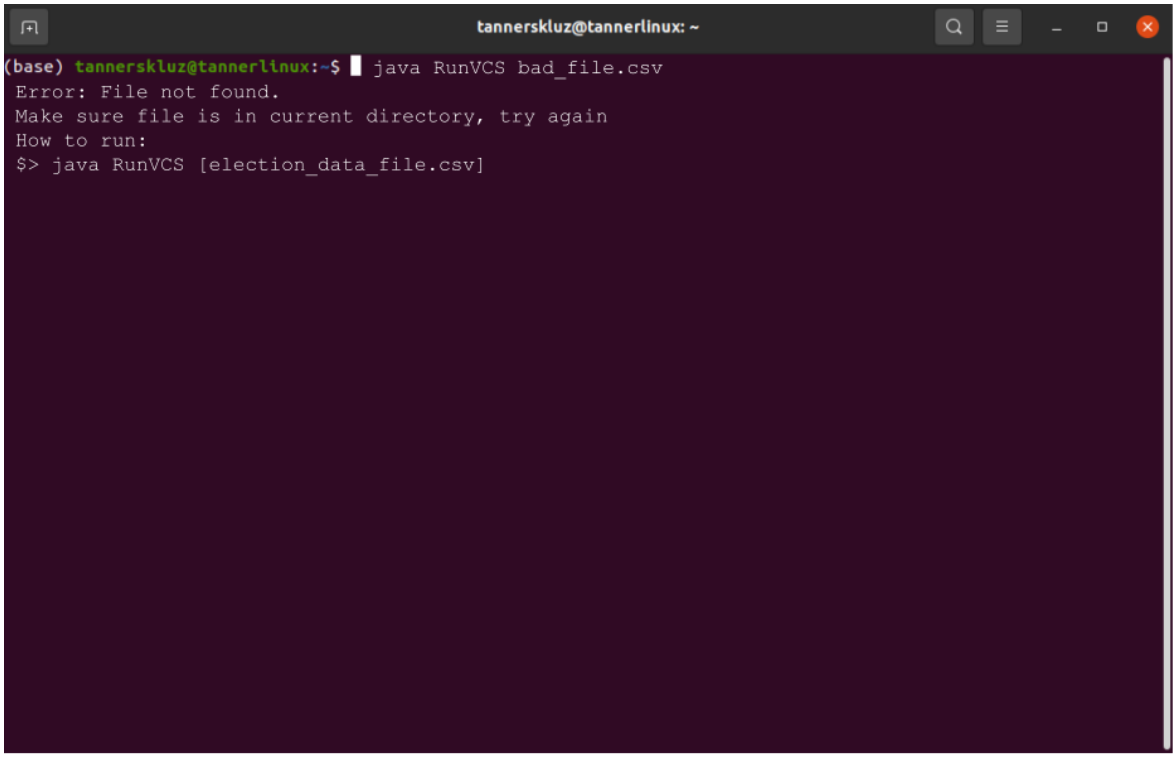
Example Interface for Successful IR Election:



Example User Interface for Running VCS with no File Argument:



Example User Interface for Running VCS with File Not Found:



Media File Example:

MediaFileExample.txt

```
Media File for OPL election on 2/27/2021
Candidates: [Pike,D], [Foster,D],[Deutsch,R], [Borg,R], [Jones,R], [Smith,I]
Total Votes: 20,000
Seats Available: 4

Election Results:
Party           Votes           Seats Allocated
Republican      10,000          2
Democrat        9,000           2
Independent     1,000           0

Candidate Rank Within Party:
Republican
    Borg (6,000 votes)
    Jones (3,000 votes)
    Deutsch (1,000 votes)
Democat
    Pike (5,000 votes)
    Foster (4,000 votes)
Independent
    Smith (1,000 votes)

Elected Candidates:
[Borg,R]
[Pike,D]
[Jones,R]
[Foster,D]
```

Example Audit File:

Audit File fo IR election on 2/27/2021

```
Audit File fo IR election on 2/27/2021
Candidates: [Pike, D], [Borg, R], [Smith, I]
Total Votes: 11

>Assigning votes to the first choice on each ballot

>Listing candidates and the associated ballot ID's:

Pike(D)
    ID#0
    ID#4
    ID#7
    ID#2

Borg(R)
    ID#1
    ID#9
    ID#3
    ID#10

Smith(I)
    ID#5
    ID#6
    ID#8

First Count Results:
Candidates              Votes
Pike(D)                 4
Borg(R)                 4
Smith(I)                3

>Smith(I) had the lowest vote count for this round

>Redistribute Smith(I)'s ballots to second choice rank if available:

Pike(D)
    ID#0
    ID#4
    ID#7
    ID#2

Borg(R)
    ID#1
    ID#9
    ID#3
    ID#10
    ID#5
    ID#6

Secound Count Results:
Candidates                  Total Votes
Pike(D)                     4
Borg(R)                     6
Smith(I)                    0

>Borg(R) has reached a majority so they are elected.
>IR election winner:
    Borg(R)
```

6.3 Screen Objects and Actions

Since the VCS software is run through the command line/terminal, there will not be any object or actions to interact with to run the software. Instead, the user will simply type in the file name that stores the election data they would like to run the software on as a command line argument.

7. Requirements Matrix

Functional Requirement	Use Case Reference on SRS	Functions
Bring File into System	4.1	main() and openFile(String name) in Election class
Preprocessing Data from Ballot File	4.2	The election class constructor reads first line and calls either OPLInit() or IRInit()
Run an Instant Runoff Election with Given Ballots	4.3	runIR() facilitates the instant runoff election
Run an Open Party List Election with Given Ballots	4.4	runOPL() facilitates the instant runoff election
Determine a tie	4.5	Inside the runIR(), runOPL(), greaterRemain(), and isGreatThan() functions a random number generator is used to determine ties
Create Audit File	4.6	The audit file is instantialized at the beginning of the election class constructor, audit file is updated in updateAudit()
Display Winners and Information	4.7	displayWinners() called at the end of the run functions
Create Media File	4.8	The media file is instantialized at the beginning of the election class constructor, media file is updated in endMedia()