Project Name: Project1 - Voting System
Team#1
Testing Log - Voting System (Task #3)

Jake Waro (warox001)
Allison Miller (mill7079)
Sami Frank (fran0942)
Declan Buhrsmith (buhrs001)

System Tests

Test Stage: Unit ___ System **X** **Test Case ID#**: testSystemVoteTypeInput **Test Description:** This test will check for the UI prompt to confirm either Plurality or STV was selected to determine what type of election the system should process. **Automated:** Yes ____ No **X**	**Test Date**: 4/2/2020 **Name(s) of Testers**: Sami Frank **Indicate where you are storing the test (what file) and the name of the method/functions being used:** Test uses UI.java and focuses on its promptVoteType method

Results: Pass ___X___ Fail _____

Preconditions for Test: The class files must compile, and UI.java has started

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Testing for Plurality vote type. Enter the number 1.	1	Move to next prompt.	Move to next prompt	Confirmation screen at last prompt will be checked against "Plurality"
2	Enter data to get to confirmation screen	numSeats = 1 File name = "/testing/tesBallot1.csv"	Confirmation screen: " Algorithm type: Plurality Number of seats to fill: 1 file(s) entered: testBallot1.csv "	Algorithm type: "Plurality"	

			Expect Algorithm type to be “Plurality”		
3	Restart program (java UI.java)				
4	Testing for STV vote type. Enter the number 2.	2	Move to next prompt	Move to next prompt	Confirmation screen at last prompt will be checked against “STV”
5	Enter data to get to confirmation screen	numSeats = 1 File name = “/testing/tesWriteFile2.csv”	Confirmation screen: “ Algorithm type: STV Number of seats to fill: 1 file(s) entered: testWriteFile.csv “ Expect Algorithm type to be “STV”	Algorithm type: “STV”	
6	Restart program (java UI.java)				
7	Testing for incorrect input. (Anything but 1 or 2)	Try: *return*	java.lang.IllegalArgumentException: is not a valid argument. Try again.	java.lang.IllegalArgumentException: is not a valid argument. Try again.	
8	Testing for incorrect input. (Anything but 1 or 2)	Try: f	java.lang.IllegalArgumentException: f is not a valid argument. Try again.	java.lang.IllegalArgumentException: f is not a valid argument. Try again	
9	Testing for incorrect input. (Anything but 1 or 2)	Try: 3	java.lang.IllegalArgumentException: 3 is not a valid argument. Try again.	java.lang.IllegalArgumentException: 3 is not a valid argument. Try again.	

Post condition(s) for Test: Steps 1-5 should get the confirmation screen. Steps 6-9 should show that you're looping through the same prompt until you give valid input (either 1 or 2, 'h' to enter the help window or 'x' to quit the program).

Test Stage: Unit ___ System **X**	**Test Date**: 4/2/2020
Test Case ID#: testSystemFilesInput	**Name(s) of Testers**: Jake Waro
Test Description: This test checks that the files input prompt can handle different kinds of inputs.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:**
Automated: Yes ____ No **X**	UI.promptFiles method

Results: Pass _____ Fail **X**

Preconditions for Test: UI.java exists. Vote type and number of seats have been collected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run "java UI" command				
2	At the vote type prompt: input 2				
3	At the number of seats prompt: input 2				
4	At the files prompt, press enter with no other input		Illegal Argument Exception	Moves to confirm Input prompt	
5	Input 2 to re-enter data				
6	Repeat steps 2 & 3				
7	Input a file that doesn't exist at file prompt	"sadfadsbab.txt"	Illegal Argument Exception	Illegal Argument Exception	

8	Input a file that does exist.	"/testing/testWriteFile2.csv"	Program moves to Confirm Input prompt	Program moves to Confirm Input prompt	
9	Repeat steps 5 & 6				
10	At the files input prompt, input a file that does exist followed by a space and a file that doesn't exist	"/testing/testWriteFile2.csv sadfadsbab.txt "	Illegal Argument Exception	Illegal Argument Exception	
11	At the files prompt, input two acceptable file paths separated by a space.	"/testing/testWriteFile1.csv /testing/testWriteFile2.csv "	Program moves onto the confirm input prompt.	Program moves onto the confirm input prompt.	

Post condition(s) for Test: Voting System has collected files to use for the election and moved onto the confirm input prompt.

Test Stage: Unit ___ System **X**

Test Case ID#: testSystemSTVelection

Test Description: this test runs several STV elections, and looks to confirm the output results are correct.

Automated: Yes ____ No **X**

Test Date: 4/2/2020

Name(s) of Testers: Jake Waro

Indicate where you are storing the test (what file) and the name of the method/functions being used:

Files used: everything besides Plurality.java

Results: Pass **X** Fail _____

Preconditions for Test: UI.java, Election.java, Ballot.java, Candidate.java, VotingAlgorithm.java, and STV.java exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run "java UI"				

	command				
2	At the vote type prompt, input 2				
3	For the number of seats, input 0				
4	For the files, input "/testing/testWriteFile 2.csv"				
5	Confirm the input by inputting 1				
6	Confirm the results displayed to screen are correct.		Election Type: STV Number of Ballots: 9 Number of Seats: 0 Number of Candidates: 4 Winner(s): Loser(s): Declan Allison Sami Jake	Election Type: STV Number of Ballots: 9 Number of Seats: 0 Number of Candidates: 4 Winner(s): Loser(s): Declan Allison Sami Jake	
7	Open the ElectionResults_Audit File.txt file and confirm the results match the expected results from step 6.				
8	Repeats steps 1 - 5 using number of seats = 1 for step 3				
9	Confirm the results displayed to screen are correct.		Election Type: STV Number of Ballots: 9 Number of Seats: 1 Number of Candidates: 4 Winner(s): Jake Loser(s): Sami Allison Declan	Election Type: STV Number of Ballots: 9 Number of Seats: 1 Number of Candidates: 4 Winner(s): Jake Loser(s): Sami Allison Declan	
10	Open the ElectionResults_Audit File.txt file and				

	confirm the results match the expected results from step 9.				
11	Repeat step 8 using number of seats = 2.				
12	Confirm the results displayed to screen are correct.		Election Type: STV Number of Ballots: 9 Number of Seats: 2 Number of Candidates: 4 Winner(s): Jake Declan Loser(s): Sami Allison	Election Type: STV Number of Ballots: 9 Number of Seats: 2 Number of Candidates: 4 Winner(s): Jake Declan Loser(s): Sami Allison	
13	Open the ElectionResults_Audit File.txt file and confirm the results match the expected results from step 12.				
14	Repeat step 8 using number of seats = 3.				
15	Confirm the results displayed to screen are correct.		Election Type: STV Number of Ballots: 9 Number of Seats: 3 Number of Candidates: 4 Winner(s): Jake Declan Sami Loser(s): Allison	Election Type: STV Number of Ballots: 9 Number of Seats: 3 Number of Candidates: 4 Winner(s): Jake Declan Sami Loser(s): Allison	
16	Open the ElectionResults_Audit File.txt file and confirm the results match the expected results from step 12.				
17	Repeat step 8 using number of seats = 4.				
18	Confirm the results		Election Type: STV	Election Type: STV	

	displayed to screen are correct.		Number of Ballots: 9 Number of Seats: 4 Number of Candidates: 4 Winner(s): Jake Declan Sami Allison Loser(s):	Number of Ballots: 9 Number of Seats: 4 Number of Candidates: 4 Winner(s): Jake Declan Sami Allison Loser(s):	
19	Open the ElectionResults_AuditFile.txt file and confirm the results match the expected results from step 12.				
20	Repeat steps 1 - 19, while using two input files	"/testing/testWriteFile1.csv /testing/testWriteFile2.csv"	Winners hierarchy goes: (1) Allison (2) Jake (3) Sami (4) Declan	Winners hierarchy goes: (1) Allison (2) Jake (3) Sami (4) Declan	
21	Run an STV election, with 2 seats, using a test ballot file with over 100,000 votes	"test100000Ballots.csv"	Election Type: STV Number of Ballots: 100053 Number of Seats: 2 Number of Candidates: 4 Winner(s): Allison Jake Loser(s): Declan Sami	Election Type: STV Number of Ballots: 100053 Number of Seats: 2 Number of Candidates: 4 Winner(s): Allison Jake Loser(s): Declan Sami	

Post condition(s) for Test: Voting System Properly runs an STV election.

Test Stage: Unit ___ System **X**	**Test Date**: 4/2/2020
Test Case ID#: testSystemPluralityElection	**Name(s) of Testers**:
Test Description: This test runs several Plurality elections, and looks to confirm the output results are correct.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:**

Automated: Yes ____ No **X**	All files are used besides the STV.java file.

Results: Pass **X** Fail _____

Preconditions for Test: UI.java, Election.java, Ballot.java, Candidate.java, VotingAlgorithm.java, and Plurality.java exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run "java UI" command				
2	At the vote type prompt, input 1				
3	For the number of seats, input 0				
4	For the file input, enter a valid Plurality styled file.	"/testing/electionPluralityTest1.csv"			
5	Confirm Input is correct, enter 1				
6	Verify the results displayed to screen are correct.		Election Type: Plurality Number of Ballots: 183 Number of Seats: 0 Number of Candidates: 8 Winner(s): Loser(s): Kevin Hart Micky Mouse Sharon Joe Exotic Joy Henry Ford Karen Amelia Earhart	Election Type: Plurality Number of Ballots: 183 Number of Seats: 0 Number of Candidates: 8 Winner(s): Loser(s): Kevin Hart Micky Mouse Sharon Joe Exotic Joy Henry Ford Karen Amelia Earhart	
7	Confirm the results in the audit file match step 6.				

8	Repeat Steps 1 - 5, using number of seats = 1				
9	Verify the results displayed to screen are correct.		Election Type: Plurality Number of Ballots: 183 Number of Seats: 1 Number of Candidates: 8 Winner(s): Kevin Hart Loser(s): Micky Mouse Sharon Joe Exotic Joy Henry Ford Karen Amelia Earhart	Election Type: Plurality Number of Ballots: 183 Number of Seats: 1 Number of Candidates: 8 Winner(s): Kevin Hart Loser(s): Micky Mouse Sharon Joe Exotic Joy Henry Ford Karen Amelia Earhart	
10	Repeat step 8 using number of seats = 2, 3, 4, 5, 6, 7, and 8. For each additional seat, verify the correct candidate moved from losers to winners.		2 seats - Micky Mouse moves to winners 3 seats - Sharon moves to winners 4 seats - Joe Exotic moves to winners 5 & 6 seats - Joy or Henry Ford moves to winners as they share the same number of votes 7 seats - Karen moves to winners 8 seats - Amelia Earhardt moves to winners. There are no losers.	2 seats - Micky Mouse moves to winners 3 seats - Sharon moves to winners 4 seats - Joe Exotic moves to winners 5 & 6 seats - Joy or Henry Ford moves to winners as they share the same number of votes 7 seats - Karen moves to winners 8 seats - Amelia Earhardt moves to winners. There are no losers.	

Post condition(s) for Test: Voting System Properly runs a plurality election.

Test Stage: Unit ___ System **X** **Test Case ID#**: testSystemConfirmInput **Test Description:** Tests for correct input to confirm previous entered data (indicated by 1) or to restart prompts because wrong data was inputted (indicated by 2). Anything else should reprompt. **Automated:** Yes ____ No **X**	**Test Date**: 4/2/2020 **Name(s) of Testers**: Sami Frank **Indicate where you are storing the test (what file) and the name of the method/functions being used:**

Results: Pass __X___ Fail _____

Preconditions for Test: Java UI is ran, correct input was given to lead up to the confirmation prompt.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Test that data is accurate.	1	Program begins to process	Program begins to process	
2	Reset test (rerun program and meet preconditions)				
3	Test that data was inaccurate and something needs to be reentered.	2	Program brings you back to the first prompt (entering vote type)	Program brings you back to the first prompt (entering vote type)	
4	Reset test (rerun program and meet preconditions)				
5	Testing for incorrect	Try:	java.lang.IllegalAr	java.lang.IllegalAr	

	input. (Anything but 1 or 2)	r	gumentException: r is not a valid argument. Try again.	gumentException: r is not a valid argument. Try again.	
6	Testing for incorrect input. (Anything but 1 or 2)	Try: 10	java.lang.IllegalArgumentException: 10 is not a valid argument. Try again.	java.lang.IllegalArgumentException: 10 is not a valid argument. Try again.	
7	Testing for incorrect input. (Anything but 1 or 2)	Try: *return*	java.lang.IllegalArgumentException: is not a valid argument. Try again.	java.lang.IllegalArgumentException: is not a valid argument. Try again.	

Post condition(s) for Test: Either the election should start processing, the user should be brought back to the first prompt to refill information that was deemed incorrect by the user, or you're looping through the same prompt until you give valid input (either "h", "x", 1, 2)

STV Class - Unit Tests

Test Stage: Unit **X** System __ **Test Case ID#**: test_constructor **Test Description:** Checks that the STV object is constructed successfully and that all of the instance variables are initialized properly. **Automated:** Yes **X** No __	**Test Date**: 4/1/2020 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, addBallotToList, getBallotList, getNumSeats. From STV: constructor, getDroop, getVoteOrder.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 4, new File[0], false);			
2	Add ballots to the election	50 ballots with vote “1,2,3,4”			
3	Create an STV object using the created Election	stv			
4	Check if ballots were set correctly	test_ballots	true	true	
5	Check if the number of seats is correct	4	true	true	
6	Check if the droop quota was set correctly	11	true	true	
7	Check if the voteOrder stack was initialized	voteOrder != null	true	true	

Post condition(s) for Test: An STV object has been successfully initialized.

Test Stage: Unit **X** System __	**Test Date**: 4/1/20
Test Case ID#: test_calcDroopQuota_50	**Name(s) of Testers**: Allison Miller
Test Description: Checks if the Droop quota is correctly calculated for a low number of ballots and a variety of seat numbers.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:**
Automated: Yes **X** No __	STVTest.java in /src directory. From Election: constructor, addBallotToList. From STV: constructor, calcDroopQuota.

Results: Pass ___**X**_ Fail _____

Preconditions for Test: STV.java, Election.java, and Ballot.java exist in /src.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Add ballots to the election	50 ballots			
3	Create an STV object with the created Election	stv			
4	Check if the Droop quota is correct for one seat	26	true	true	
5	Check if the Droop quota is correct for five seats	9	true	true	
6	Check if the Droop quota is correct for ten seats	5	true	true	
7	Check if the Droop quota is correct for zero seats	1	true	true	

Post condition(s) for Test: Droop quotas have been successfully calculated for different numbers of seats.

Test Stage: Unit **X** System __	**Test Date**: 4/1/20
Test Case ID#: test_calcDroopQuota_max	**Name(s) of Testers**: Allison Miller

Test Description: Checks if the Droop quota is correctly calculated for the maximum number of ballots across a variety of seat numbers. **Automated:** Yes **X** No __	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, addBallotToList. From STV: constructor, calcDroopQuota.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Add ballots to the election	100,000 ballots			
3	Create an STV object with the created Election	stv			
4	Check if the Droop quota is correct for one seat	50001	true	true	
5	Check if the Droop quota is correct for five seats	16667	true	true	
6	Check if the Droop quota is correct for ten seats	9091	true	true	

Post condition(s) for Test: Droop quotas have been successfully calculated for different numbers of seats.

Test Stage: Unit **X** System __ **Test Case ID#**: test_calcDroopQuota_zero **Test Description:** Checks if the Droop quota is correctly calculated for zero ballots across a variety of seat numbers. **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, addBallotToList. From STV: constructor, calcDroopQuota.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Create an STV object with the created Election	stv			
3	Check if the Droop quota is correct for one seat	1	true	true	
4	Check if the Droop quota is correct for five seats	1	true	true	
5	Check if the Droop quota is correct for ten seats	1	true	true	

Post condition(s) for Test: Droop quotas have been successfully calculated for different numbers of seats (though with zero ballots the value is always the same).

Test Stage: Unit **X** System __ **Test Case ID#**: test_shuffle_true **Test Description:** Checks if the shuffle function shuffles the ballots and does not create patterns while doing so. **Automated:** Yes__ No **X**	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, setCandidates, addBallotToList. From STV: constructor, shuffle, getBallots.

Results: Pass ___**X**_____Fail _____

Preconditions for Test: STV.java, Election.java, and Ballot.java exist in /src.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], true);			
2	Set the candidates of the election	"jake,allison,sami,declan,test5"			
3	Add ballots to the election	10 ballots			
4	Create an STV object with the created Election	stv			
5	Write the original ballot order to the file using the ballot numbers	testing/test_shuffle_true.txt			
6	Shuffle the ballots ten times, writing the order to the file with each shuffle	stv.getBallots(), testing/test_shuffle_true.txt			

7	Manually check the results in the file for shuffle correctness.		true	true	

Post condition(s) for Test: Ballots have been successfully shuffled and the shuffle displays no identifiable patterns.

Test Stage: Unit **X** System __

Test Case ID#: test_shuffle_false

Test Description: Checks that the shuffle function does not shuffle the ballots when the shuffle switch in Election is turned off.

Automated: Yes__ No **X**

Test Date: 4/1/20

Name(s) of Testers: Allison Miller

Indicate where you are storing the test (what file) and the name of the method/functions being used:

STVTest.java in /src directory. From Election: constructor, addBallotToList. From STV: constructor, calcDroopQuota.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Set the candidates of the election	"jake,allison,sami,declan,test5"			
3	Add ballots to the election	10 ballots			
4	Create an STV object with the created	stv			

	Election				
5	Write the original ballot order to the file using the ballot numbers	testing/test_shuffle_false.txt			
6	Shuffle the ballots ten times, writing the order to the file with each shuffle	stv.getBallots(), testing/test_shuffle_false.txt			
7	Manually check the results in the file to ensure all orders were the same.		true	true	

Post condition(s) for Test: No ballots have been shuffled.

Test Stage: Unit **X** System __

Test Case ID#: test_distributeVotes_noWinners

Test Description: Checks to ensure ballots are distributed properly with no candidate reaching Droop.

Automated: Yes **X** No __

Test Date: 4/1/20

Name(s) of Testers: Allison Miller

Indicate where you are storing the test (what file) and the name of the method/functions being used:

STVTest.java in /src directory. From Election: constructor, setCandidates, addBallotToList. From STV: constructor, distributeVotes. From Candidate: getVoteCount.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, Candidate.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv",			

		1, new File[0], false);			
2	Set the candidates of the election	“test1, test2, test3”			
3	Add ballots to the election	6 ballots, with each candidate ranked first on two of them			
4	Create an STV object with the created Election	stv			
5	Distribute the votes	stv			
6	Check that there are no winners	0	true	true	
7	Check that there are no losers	0	true	true	
8	Check that there are still three candidates	3	true	true	
9	Check that the vote count for each candidate is 2	2	true	true	

Post condition(s) for Test: Ballots have successfully been distributed among all three candidates and no candidate has been removed from the race.

Test Stage: Unit **X** System __	**Test Date**: 4/1/20
Test Case ID#: test_distributeVotes_oneWinner	**Name(s) of Testers**: Allison Miller
Test Description: Checks to ensure ballots are distributed properly with one candidate reaching Droop. Ensures the candidate who reached Droop is properly declared a winner.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, setCandidates, addBallotToList. From

Automated: Yes **X** No __	STV: constructor, distributeVotes. From Candidate: getVoteCount.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, Candidate.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Set the candidates of the election	“test1, test2, test3”			
3	Add ballots to the election	9 ballots, with each candidate ranked first twice across six of them, and one candidate ranked first on three others			
4	Create an STV object with the created Election	stv			
5	Distribute the votes	stv			
6	Check that there is one winner	1	true	true	
7	Check that there are no losers	0	true	true	
8	Check that two candidates remain	2	true	true	
9	Check that the vote count for each	2	true	true	

	remaining candidate is 2				
10	Check that the vote count for the winning candidate equals the Droop quota	stv.getDroop()	true	true	

Post condition(s) for Test: Ballots have successfully been distributed among all three candidates and one candidate has been removed from the race as a winner.

Test Stage: Unit **X** System __ **Test Case ID#**: test_distributeVotes_secondRank **Test Description:** Checks to ensure ballots are distributed properly with one candidate reaching Droop. Ensures the candidate who reached Droop is properly declared a winner. Ensures ballots with that candidate as the top ranked candidate distributed after the candidate is declared a winner are given to the second rank candidate. **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, setCandidates, addBallotToList. From STV: constructor, distributeVotes. From Candidate: getVoteCount.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, Candidate.java, and Ballot.java exist in /src.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Set the candidates of the election	“test1, test2, test3”			
3	Add ballots to the	Same ballots			

	election	as previous test with two additional ballots that go to second-ranked candidates			
4	Create an STV object with the created Election	stv			
5	Distribute the votes	stv			
6	Check that there is one winner	1	true	true	
7	Check that there are no losers	0	true	true	
8	Check that two candidates remain	2	true	true	
9	Check that the vote count for each remaining candidate is 3	3	true	true	
10	Check that the vote count for the winning candidate equals the Droop quota	stv.getDroop()	true	true	

Post condition(s) for Test: Ballots have successfully been distributed among all three candidates and one candidate has been removed from the race as a winner.

Test Stage: Unit **X** System __	**Test Date**: 4/1/20
Test Case ID#: test_distributeVotes_twoWinners	**Name(s) of Testers**: Allison Miller
Test Description: Checks to ensure ballots are distributed properly with two candidates reaching Droop. Ensures the candidates who reached Droop are properly declared winners.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:**

Automated: Yes **X** No __	STVTest.java in /src directory. From Election: constructor, setCandidates, addBallotToList. From STV: constructor, distributeVotes. From Candidate: getVoteCount.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 2, new File[0], false);			
2	Set the candidates of the election	“test1, test2, test3”			
3	Add ballots to the election	Same ballots as previous test with two additional ballots that go to second-ranked candidates			
4	Create an STV object with the created Election	stv			
5	Distribute the votes	stv			
6	Check that there are two winners	2	true	true	
7	Check that there are no losers	0	true	true	
8	Check that one candidate remains	1	true	true	
9	Check that the vote	3	true	true	

	count for the remaining candidate is 3				
10	Check that the first winning candidate is test2 with a vote count equal to the Droop quota	test_cans.get(1), stv.getDroop()	true	true	
11	Check that the second winning candidate is test1 with a vote count equal to the Droop quota	test_cans.get(0), stv.getDroop()	true	true	

Post condition(s) for Test: Ballots have successfully been distributed among all three candidates and two candidates have been removed from the race as winners.

Test Stage: Unit **X** System __ **Test Case ID#**: test_distributeVotes_thirdRank **Test Description:** Checks to ensure ballots are distributed properly with two candidates reaching Droop. Ensures the candidates who reached Droop are properly declared winners. Ensures ballots with those candidates as the top ranked candidates distributed after they are declared as winners are given to the ballots' third rank candidate(s). **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, setCandidates, addBallotToList. From STV: constructor, distributeVotes. From Candidate: getVoteCount.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, Candidate.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election	elect = new			

	object	Election("stv", 2, new File[0], false);			
2	Set the candidates of the election	“test1, test2, test3”			
3	Add ballots to the election	Same ballots as previous test with one additional ballot for the third-ranked candidate			
4	Create an STV object with the created Election	stv			
5	Distribute the votes	stv			
6	Check that there are two winners	2	true	true	
7	Check that there are no losers	0	true	true	
8	Check that one candidate remains	1	true	true	
9	Check that the vote count for the remaining candidate is 4	4	true	true	
10	Check that the first winning candidate is test2 with a vote count equal to the Droop quota	test_cans.get(1), stv.getDroop()	true	true	
11	Check that the second winning candidate is test1 with a vote count equal to the Droop quota	test_cans.get(0), stv.getDroop()	true	true	

Post condition(s) for Test: Ballots have successfully been distributed among all three candidates and two candidates have been removed from the race as winners.

Test Stage: Unit **X** System __ **Test Case ID#**: test_getLoser_singleLoser **Test Description:** Checks that getLoser returns the proper candidate with a single potential loser. **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, setCandidates. From STV: constructor, getLoser, getVoteOrder. From Candidate: addBallot. From Ballot: constructor.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, Ballot.java, and Candidate.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Set the candidates of the election	“test1,test2,test3,test4,test5”			
3	Add ballots to candidates	Ballots equal to # in candidate name test#			
4	Create an STV object with the created election	stv			
5	Push the candidates onto the voteOrder	voteOrder			

	stack				
6	Check that the loser is test1 (with one ballot)	test_cans.get(0)	true	true	

Post condition(s) for Test: Candidate test1 has been returned from getLoser as a loser.

Test Stage: Unit **X** System __ **Test Case ID#**: test_getLoser_twoLosers **Test Description:** Checks that getLoser returns the correct candidate as the loser when there are two losers. **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, setCandidates. From STV: constructor, getLoser, getVoteOrder. From Candidate: addBallot. From Ballot: constructor.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, Ballot.java, and Candidate.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Set the candidates of the election	“test1,test2,test3,test4,test5”			
3	Add ballots to candidates	Ballots equal to # in candidate name test#			
4	Create an STV object	stv			

	with the created election				
5	Push the candidates onto the voteOrder stack	voteOrder			
6	Add one more ballot to test1 for a total of two ballots	test_cans.get(0)			
7	Check that the loser is test2 (with two ballots)	test_cans.get(1)	true	true	

Post condition(s) for Test: Candidate test2 has been returned from getLoser as a loser.

Test Stage: Unit **X** System __ **Test Case ID#**: test_getLoser_allLosers **Test Description:** Checks that getLoser returns the correct candidate when all candidates could be losers. **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, setCandidates. From STV: constructor, getLoser, getVoteOrder. From Candidate: addBallot. From Ballot: constructor.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, Ballot.java, and Candidate.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			

2	Set the candidates of the election	“test1,test2,test3,test4,test5”			
3	Add ballots to candidates	3 ballots for each candidate			
4	Create an STV object with the created election	stv			
5	Push the candidates onto the voteOrder stack	voteOrder			
6	Check that the loser is test5 (with three ballots - added last to voteOrder)	test_cans.get(test_cans.size()-1)	true	true	

Post condition(s) for Test: Candidate test5 has been returned from getLoser as a loser.

Test Stage: Unit **X** System __ **Test Case ID#**: test_breakTie_zeroCandidates **Test Description:** Checks that breakTie properly returns null if attempting to break a tie between zero candidates. **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor. From STV: constructor, breakTie.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Candidate.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv"			

		, 1, new File[0], false);			
2	Create an STV object with the created Election	stv			
3	Create an empty LinkedList<Candidate>	losers			
4	Check that breakTie called on losers returns null	null	true	true	

Post condition(s) for Test: breakTie returned null with no tie to break.

Test Stage: Unit **X** System __ **Test Case ID#**: test_breakTie_oneCandidate **Test Description:** Checks that breakTie properly returns the candidate if attempting to break a tie with a single candidate. **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor. From STV: constructor, breakTie. From Candidate: constructor.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, and Candidate.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv" , 1, new File[0], false);			
2	Create an STV object	stv			

	with the created Election				
3	Create an empty LinkedList<Candidate>	losers			
4	Create a new Candidate and add it to losers	test1			
5	Check that breakTie called on losers returns test1	test1	true	true	

Post condition(s) for Test: test1 returned as the winner (loser?) of the tie-break.

Test Stage: Unit **X** System __ **Test Case ID#**: test_breakTie_twoCandidates **Test Description:** Check that breakTie returns the candidate with the latest first vote given two candidates with the same positive number of votes. **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor. From STV: constructor, breakTie. From Candidate: constructor.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Candidate.java exist in /src.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Create an STV object with the created	stv			

	Election				
3	Create an empty LinkedList<Candidate>	losers			
4	Create two new Candidates and add them to losers	test1, test2			
5	Push the candidates on to voteOrder	test1, test2			Pushed in this order
6	Check that breakTie called on losers returns test2 (highest on stack)	test2	true	true	

Post condition(s) for Test: test2 returned as the overall loser from breakTie.

Test Stage: Unit **X** System __

Test Case ID#: test_breakTie_threeCandidates

Test Description: Check that breakTie returns the candidate with the latest first vote given three candidates with the same positive number of votes.

Automated: Yes **X** No __

Test Date: 4/1/20

Name(s) of Testers: Allison Miller

Indicate where you are storing the test (what file) and the name of the method/functions being used:

STVTest.java in /src directory. From Election: constructor. From STV: constructor, breakTie. From Candidate: constructor.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Candidate.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			

2	Create an STV object with the created Election	stv			
3	Create an empty LinkedList<Candidate>	losers			
4	Create three new Candidates and add them to losers	test1, test2, test3			
5	Push the candidates on to voteOrder	test3, test1, test2			Pushed in this order
6	Check that breakTie called on losers returns test2 (highest on stack)	test2	true	true	

Post condition(s) for Test: test2 returned as the overall loser from breakTie.

Test Stage: Unit **X** System __ **Test Case ID#**: test_breakTie_fourCandidates **Test Description:**Check that breakTie returns the candidate with the latest first vote given three candidates with the same positive number of votes and one candidate with a higher number of votes (checking usage of voteOrder). **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor. From STV: constructor, breakTie. From Candidate: constructor.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Candidate.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv",			

		1, new File[0], false);			
2	Create an STV object with the created Election	stv			
3	Create an empty LinkedList<Candidate>	losers			
4	Create three new Candidates and add them to losers	test1, test2, test3			
5	Push the candidates on to voteOrder	test1, test2, test3			Pushed in this order
6	Push a new Candidate on to voteOrder	"winner4"			Not a loser because not in losers list
7	Check that breakTie called on losers returns test3	test3	true	true	

Post condition(s) for Test: test3 returned as the overall loser from breakTie.

Test Stage: Unit **X** System __	**Test Date**: 4/1/20
Test Case ID#: test_breakTie_zeroVotes	**Name(s) of Testers**: Allison Miller
Test Description: Check that a candidate is returned randomly given three candidates with zero votes.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor. From STV: constructor, breakTie. From Candidate: constructor.
Automated: Yes **X** No __	

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Candidate.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Create an STV object with the created Election	stv			
3	Create an empty LinkedList<Candidate>	losers			
4	Create three new Candidates and add them to losers	test1, test2, test3			
5	Push three other candidates on to voteOrder	“winner1”, “winner2”, “winner3			
6	Check that breakTie called on losers returns one of the three losers	test1, test2, test3	true	true	Loser is returned randomly from the three

Post condition(s) for Test: Either test1, test2, or test3 is returned as the overall loser from breakTie.

Test Stage: Unit **X** System __ **Test Case ID#**: test_breakTie_maxCandidates **Test Description:** Checks that breakTie returns the candidate with the latest first vote given three candidates with the same low positive number of votes and seven candidates with some higher number of votes (check usage of voteOrder). **Automated:** Yes **X** No __	**Test Date**: 4/1/20 **Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor. From STV: constructor, breakTie. From Candidate: constructor.

Results: Pass ___**X** Fail _____

Preconditions for Test: STV.java, Election.java, and Candidate.java exist in /src.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Initialize an Election object	elect = new Election("stv" , 1, new File[0], false);			
2	Create an STV object with the created Election	stv			
3	Create an empty LinkedList<Candidate>	losers			
4	Create three new Candidates and add them to losers	test1, test2, test3			
5	Push the loser candidates on to voteOrder interspersed with several other candidates	test3, “winner1”, “winner2”, test2, “winner3”, “winner4”, “winner5”, “winner6”, test1, “winner7”			Pushed in this order
7	Check that breakTie called on losers returns test1	test1	true	true	

Post condition(s) for Test: test1 returned as the overall loser from breakTie.

Test Stage: Unit **X** System __	**Test Date**: 4/1/20

Test Case ID#: test_evaluate **Test Description:** Check that evaluate calculates the proper results given an election with four candidates, one seat, and six ballots (one winner and three losers). **Automated:** Yes **X** No __	**Name(s) of Testers**: Allison Miller **Indicate where you are storing the test (what file) and the name of the method/functions being used:** STVTest.java in /src directory. From Election: constructor, addBallotToList, setCandidates. From STV: constructor, evaluate.

Results: Pass **X** Fail _____

Preconditions for Test: STV.java, Election.java, Candidate.java, and Ballot.java exist in /src.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an Election object	elect = new Election("stv", 1, new File[0], false);			
2	Set the candidates of the election	"test1,test2,test3 ,test4"			
3	Add ballots to the election	6 ballots, 2 for each of test1, test2, and test3			
4	Create an STV object with the created Election	stv			
5	Check that there is one winner	1	true	true	
6	Check that there are three losers	3	true	true	
7	Check that test1 is the winner	test_cans.get(0)	true	true	
8	Check that losers	test_cans.get(1),	true	true	

	contains all three of the other candidates	test_cans.get(2), test_cans.get(3)			
9	Check that the vote count of the winner is the Droop quota	test_cans.get(0)	true	true	

Post condition(s) for Test: Election has been run successfully. Test1 is declared a winner and test2, test3, and test4 are declared losers.

Candidate Class - Unit Tests

Test Stage: Unit **X** System __ **Test Case ID#**: testConstructorGetName **Test Description:** This test checks that the candidate object is instantiated, and that its respective fields get set correctly (the candidate's name, voteCount, isWinner, and ballots). **Automated:** Yes **X** No __	**Test Date**: 3/29/2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** CandidateTest.java in /src directory. Test method: testConstructorGetName This file is using the Candidate class and its following methods: Candidate (constructor), getName, getVoteCount, isWinner, getBallots methods.

Results: Pass **X** Fail _____

Preconditions for Test: Candidate.java class file exists in /src directory.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(strings = data)	data = { "", "J", "Jake", "Samantha", "Peter Shii", "James Gorch" }			Define test as a parameterized test.

					@ParameterizedTest
2	Set the test method to accept a String value	testValue			
3	Initialize a new candidate object, passing testValue to its constructor.	testValue			
4	assertEquals(testValue, candidate.getName());	testValue	true	true	
5	assertEquals(0, candidate.getVoteCount());	0	true	true	
6	assertEquals(false, candidate.isWinner());	false	true	true	
7	assertEquals(new LinkedHashSet<Ballot>(), candidate.getBallots());	Empty LinkedHashSet<Ballot>	true	true	

Post condition(s) for Test: Candidate object is created, and its fields are initialized with values.

Test Stage: Unit **X** System __ **Test Case ID#**: testConstructorGetBigNames **Test Description:** This test specifically checks that the Candidate constructor can correctly initialize the name field to larger names. This test checks that the candidate object is instantiated, and that its name field is initialized to the correct value. **Automated:** Yes **X** No __	**Test Date**: 3/29/2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** CandidateTest.java in /src directory. Test method: testConstructorGetBigNames This file is using the Candidate class and its following methods: Candidate (constructor) and getName methods.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Candidate.java class file exists in /src directory.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Create a new String, name999, and set it to a string of length 999.	"jjjj…" where name999.length() == 999			Use helper function makeString(999)
2	Create a new candidate, passing name999 to its constructor.	name999			
3	assertEquals(name999, candidate.getName());	name999	true	true	
4	Repeat steps 1 - 3 using values of string length 1000, 999999, 1000000	name1000 name999999 name1000000	true	true	

Post condition(s) for Test: Candidate object is created, and its fields are initialized with values.

Test Stage: Unit **X** System __	**Test Date**: 3/29/2020
Test Case ID#: testIsWinnerFalse	**Name(s) of Testers**: Jake Waro
Test Description: This test verifies that a candidate is not a winner right after it is initialized.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** CandidateTest.java in /src directory. Test method: testIsWinnerFalse This file is using the Candidate class and its following methods: Candidate (constructor) and isWinner methods.
Automated: Yes **X** No __	

Results: Pass ___X___ Fail _____

Preconditions for Test: Candidate.java class file exists in /src directory.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new candidate object, passing its constructor a String value of length 1 or larger.				Candidate cand = new Candidate("Jake");
2	Verify the candidate's isWinner field is false.		false	false	assertFalse(candidate.isWinner());

Post condition(s) for Test: Candidate's isWinner field is set to false.

Test Stage: Unit **X** System __	**Test Date**: 3/29/2020
Test Case ID#: testIsWinnerTrue	**Name(s) of Testers**: Jake Waro
Test Description: Verifies candidate.setWinner() sets the candidate's isWinner field to true.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** CandidateTest.java in /src directory. Test method: testIsWinnerTrue This file is using the Candidate class and its following methods: Candidate (constructor), isWinner, and setWinner methods.
Automated: Yes **X** No __	

Results: Pass ___X___ Fail _____

Preconditions for Test: Candidate.java class file exists in /src directory.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new candidate				Candidate cand = new

	object, passing its constructor a String value of length 1 or larger.				Candidate(“Jake”);
2	Call Candidate’s setWinner method				candidate.setWinner();
3	Verify candidate’s isWinner field is set to true.		true	true	assertTrue(candidate.isWinner());

Post condition(s) for Test: Candidate’s isWinner field is set to true.

Test Stage: Unit **X** System __ **Test Case ID#**: testIsWinnerMultipleSets **Test Description:** This test tests that calling setWinner method multiple times does not corrupt the isWinner field i.e. once setWinner is called, any subsequent calls should not change the value of isWinner, and it should stay true for the rest of the program. **Automated:** Yes **X** No __	**Test Date**: 3/29/2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** CandidateTest.java in /src directory. Test method: testIsWinnerMultipleSets This file is using the Candidate class and its following methods: Candidate (constructor), isWinner, and setWinner methods.

Results: Pass **X** Fail _____

Preconditions for Test: Candidate.java class file exists in /src directory.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new candidate object, passing its constructor a String value of length 1 or larger.				Candidate cand = new Candidate(“Jake”);

2	Call Candidate's setWinner method twice.				candidate.setWinner(); candidate.setWinner();
3	Verify the candidate's isWinner field is set to true.		true	true	assertTrue(candidate.isWinner());
4	Call Candidate's setWinner method				candidate.setWinner();
5	Verify the candidate's isWinner field is set to true.		true	true	assertTrue(candidate.isWinner());

Post condition(s) for Test: Candidate's isWinner field is set to true.

Test Stage: Unit **X** System __

Test Case ID#: testGetVoteCount

Test Description: Test that a candidate's voteCount field matches the number of ballots they have assigned to them.

Automated: Yes **X** No __

Test Date: 3/29/2020

Name(s) of Testers: Jake Waro

Indicate where you are storing the test (what file) and the name of the method/functions being used:

CandidateTest.java in /src directory
Test method: testGetVoteCount
This file is using the Candidate and Ballot classes and the following methods: Candidate constructor, Ballot constructor, Candidate's addBallot and getVoteCount methods.

Results: Pass **X** Fail _____

Preconditions for Test: Candidate.java class file exists in /src directory.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test.	data = { 0, 1, 2, 4, 11, 999, 1000,			Define test as a parameterized test.

	@ValueSource(ints = data)	999999, 1000000 }			@ParameterizedTest
2	Set the test method to accept a int value	testValue			
1	Create a new *candidate* object, passing its constructor a String value of length 1 or larger.	"Jake"			
2	Create a new, empty LinkedHashSet<Candidate>, *cands*.	*cands*			
3	Add the candidate object to *cands*.				cands.add(candidate);
4	Add testValue ballots to the candidate object	new Ballot(cands, "1")			Use a loop to add testValue ballots.
5	Verify that the voteCount of the candidate matches the testValue		true	true	assertEquals(testValue, candidate.getVoteCount());
6	Verify the voteCount is not equal to an arbitrary number not in the ValueSource (e.g. 14)		false	false	assertFalse(14 == candidate.getVoteCount());

Post condition(s) for Test: Candidate's voteCount matches the number of ballots they have.

Test Stage: Unit **X** System __	**Test Date**: 3/29/2020
Test Case ID#: testAddGetBallots	**Name(s) of Testers**: Jake Waro
Test Description: Test that we can add ballots to a candidate and that we can retrieve ballots from them.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** CandidateTest.java in /src directory

Automated: Yes **X** No __	Test method: testAddGetBallots This file is using the Candidate and Ballot classes and the following methods: Candidate constructor, Ballot constructor, Candidate's addBallot and getBallot methods.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Candidate.java class file exists in /src directory.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(ints = data)	data = { 0, 1, 2, 4, 7, 10, 11, 999, 1000, 99999, 100001, 999999, 1000000 }			Define test as a parameterized test. @ParameterizedTest
2	Set the test method to accept a int value	testValue			
1	Create a new *candidate* object, passing its constructor a String value of length 1 or larger.	"Jake"			
2	Create a new, empty LinkedHashSet<Candidate>, *cands*.	*cands*			
3	Add the candidate object to *cands*.				cands.add(candidate);
4	Create a new LinkedHashSet<Ballot>, *ballots*.	*ballots*			
5	Add testValue ballots to the candidate and *ballots* object. They should be adding the same ballot object respectively.	new Ballot(cands, "1")			Use a loop to add testValue amount of ballots.

6	Verify that the *ballots* object is equal to the candidate's ballot list.		true	true	assertEquals(ballots, candidate.getBallots());

Post condition(s) for Test: Candidate has a populated LinkedHashSet of ballots and it is retrievable.

Test Stage: Unit **X** System __ **Test Case ID#**: testRemoveBallots **Test Description:** Test that we can set a candidate's ballots to an empty list. This should set their voteCount to 0, and we should be able to retrieve the candidate's list of ballots before the previous steps occur. **Automated:** Yes **X** No __	**Test Date**: 3/29/2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** CandidateTest.java in /src directory Test method: testRemoveBallots This file is using the Candidate and Ballot classes and the following methods: Candidate constructor, Ballot constructor, Candidate's addBallot, removeBallots, getVoteCount, and getBallots.

Results: Pass **X** Fail _____

Preconditions for Test: Candidate.java class file exists in /src directory.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(ints = data)	data = { 0, 1, 2, 4, 7, 10, 11, 999, 1000, 99999, 100001, 999999, 1000000 }			Define test as a parameterized test. @ParameterizedTest
2	Set the test method to accept a int value	testValue			
1	Create a new *candidate* object, passing its constructor a String value of	"Jake"			

	length 1 or larger.				
2	Create a new, empty LinkedHashSet<Candidate>, *cands*.	*cands*			
3	Add the candidate object to *cands*.				cands.add(candidate);
4	Create a new LinkedHashSet<Ballot>, *ballots*.	*ballots*			
5	Add testValue ballots to the candidate and *ballots* object. They should be adding the same ballot object respectively.	new Ballot(cands, "1")			Use a loop to add testValue amount of ballots.
6	Verify that *ballots* is equal to the value returned by calling candidate's removeBallots method.	*ballots*	true	true	assertEquals(ballots, candidate.removeBallots());
7	Check that the candidate's voteCount field is set to 0.	0	true	true	assertEquals(0, candidate.getVoteCount());
8	Create a new, empty LinkedHashSet<Ballot>, *empty*.				
9	Check that the candidate's ballots field is now empty.	*empty*	true	true	assertEquals(empty, candidate.getBallots());

Post condition(s) for Test: Candidate's voteCount field is 0, their ballot list is empty, and we have the value of their ballots field prior to it getting cleared.

Plurality Class - Unit Tests

Test Stage: Unit **X** System __ **Test Case ID#**: testPluralityConstructor **Test Description:** This test method will test for the main attributes that have been assigned in creating an Election object that are used by the plurality class. (numSeats, LinkedHashSet of candidates, ballotList). **Automated:** Yes **X** No __	**Test Date**: 3/30/2020 **Name(s) of Testers**: Sami Frank **Indicate where you are storing the test (what file) and the name of the method/functions being used:** PluralityTest.java in /src directory. Test method: testConstructor This file is using the Plurality class and Election class (numSeats, candidates, getBallotList())

Results: Pass **X** Fail _____

Preconditions for Test: The election and plurality class must compile with no errors. An Election object must be initialized with the "Plurality" vote type and ballots must be put into elections BallotList attribute, candidates must also be set.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new instance of Election	Election e = new Election("Plurality", 2, new File[0], false); Where "Plurality" is the voteType, 2 is the number of seats to fill, new File[0] is a list of files to read (not testing here), false determines that the shuffle flag is off(not needed for plurality).	numSeats = 2	numSeats = 2	
2	Create 4 new candidats and add them to Election 'e'	[Sami, Jake, Declan, Allison]	[Sami, Jake, Declan, Allison]	[Sami, Jake, Declan, Allison]	
3	Add ballots to the election	String votes = "1,,,,"; String votes1 = ",1,,,"; String votes2 = ",,,1,";	List of 5 ballot objects	List of 5 ballot objects	

		String votes3 = ",,1,,";
		String votes4 = "1,,,,";
		e.addBallotToList(votes);
		e.addBallotToList(votes1);
		e.addBallotToList(votes2);
		e.addBallotToList(votes3);
		e.addBallotToList(votes4);			

4	Create a new instance of PluralityTest with the Election object 'e'	e.numSeats = 2 e.candidates e.BallotList = list of 5 ballot objects that coorelate to the test data in step 3	numSeats = 2 candidates = [Sami, Jake, Declan, Allison] BallotList = [Ballot, Ballot1, Ballot2, Ballot3, Ballot4]	numSeats = 2 candidates = [Sami, Jake, Declan, Allison] BallotList = [Ballot, Ballot1, Ballot2, Ballot3, Ballot4]	

Post condition(s) for Test: An object of election class and its data has been properly set in the Plurality class to use for the remainder of its functionality.

Test Stage: Unit **X** System __

Test Case ID#: testPluralityDistributeVotes

Test Description:
This test will review the plurality classes functionality when it comes to taking a list of ballots from the election object and distributing them to their respective candidate. After the function is called, the candidates vote count will be called to check against what the expected vote count should be.

Automated: Yes **X** No __

Test Date: 3/30/2020

Name(s) of Testers: Sami Frank

Indicate where you are storing the test (what file) and the name of the method/functions being used:

PluralityTest.java in /src directory.
Test method: testDistributeVotes
This file is using the Plurality class and Election class (numSeats, candidates, getBallotList())

Results: Pass **X** Fail _____

Preconditions for Test: Election class must be initialized with proper data (linked hash set of candidates, list of ballots). Plurality class must compile without error.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize new plurality class	Plurality plur = new Plurality(Election e)			
2	DistributeVotes() is called	Plur.distributeVotes()	Sami = 2 votes Jake = 1 vote Declan = 1 vote Allison = 1 vote Tester = 0 votes	Sami = 2 votes Jake = 1 vote Declan = 1 vote Allison = 1 vote Tester = 0 votes	

Post condition(s) for Test: After this function was called, each candidate in the election should have a list of ballots assigned to them if they won a ballot.

Test Stage: Unit **X** System __

Test Case ID#: testPluralityBreakTie

Test Description:
This test will check the functionality of the plurality classes breakTie() function. This test simulates a tie between 3 candidates in one instance where there are 2 seats to be filled, and one instance where there are 3 seats to be filled.

Automated: Yes **X** No __

Test Date: 3/30/2020

Name(s) of Testers: Sami Frank

Indicate where you are storing the test (what file) and the name of the method/functions being used:

PluralityTest.java in /src directory.
Test method: testBreakTie
This file is using the Plurality class and Election class (numSeats, candidates, getBallotList())

Results: Pass **X** Fail _____

Preconditions for Test: 2 elections must be initialized, one with 2 seats to fill, another with 3 seats to fill. Candidates and ballots must be added to this election and ballots must be distributed to the correct candidate.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes

1	Create an arrayList of Candidates that would expect to have a tie	3 candidates each with 2 votes	Tie = [c1,c2,c3]	[c1,c2,c3]	
2	Call plurality.breakTie(tie) when there are **2** seats available	ArrayList tie	Winner = c1 && c2 || c1&&c3 || c2&&c3	One of the 3 expected results	
3	Create an arrayList of Candidates that would expect to have a tie	3 candidates each with 2 votes	Tie1 = [c1,c2,c3]	[c1,c2,c3]	
4	Call plurality.breakTie(tie1) when there are **3** seats available	Tie1	Winner = [c1,c2,c3]	[c1,c2,c3]	

Post condition(s) for Test: After breakTie determines who should win the tie, or if there are enough seats for all candidates in the tie list to win, those candidates will be added to the winners list.

Test Stage: Unit **X** System __

Test Case ID#: testPluralityEvaluate

Test Description:
This test will review the functionality of the plurality classes evaluate function. This function will look through a candidate list sorted by how many total votes the candidate received. This function will determine if and when a tie should be broken, eventually adding winners to the Election winners queue and losers to the Election losers stack.

Automated: Yes **X** No __

Test Date: 3/30/2020

Name(s) of Testers: Sami Frank

Indicate where you are storing the test (what file) and the name of the method/functions being used:

PluralityTest.java in /src directory.
Test method: testEvaluate
This file is using the Plurality class and Election class (numSeats, candidates, getBallotList())

Results: Pass ___X___ Fail _____

Preconditions for Test: An election instance must be created will proper data (LinkedHashSet of candidates, list of ballots), a new instance of plurality must be created and distributeVotes() must have already been called.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	The evaluate() function is called directly after distributeVotes() has finished. This test should look for the expected winners and losers	Num seats = 2 Candidates = [sami,jake,Declan, Allison,tester] Vote counts = [sami: 2,jake:1,Declan:1, Allison:1,tester:0]	Winner = sami && (jake||Declan||Allison) Losers = tester && ((jake&&Declan)||(jake&&Allison)|| (Declan&&Allison)	Winner = sami && (jake||Declan||Allison) Losers = tester && ((jake&&Declan) ||(jake&&Allison)|| (Declan&&Alliso n)	

Post condition(s) for Test: The winners and losers have been sent to the election class to handle results.

Ballot Class - Unit Tests

Test Stage: Unit **X** System __	**Test Date**: 3/30/2020
Test Case ID#: testBallot	**Name(s) of Testers**: Sami Frank
Test Description: This test case will test the initialization of multiple ballots, of both STV and Plurality type. The test will then check the ballots voteMap that was created against expected output (which comes from the vote strings that are passed into each new ballot instance)	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** BallotTest.java in /src directory. Test method: testBallot Methods used: Ballot constructor

Automated: Yes **X** No __	

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Ballot and Candidate class must compile with no errors. A LinkedHashSet of candidates has been created. A list of vote Strings has been created.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Instantiate a new BallotTest object.	From setup(): A LinkedHashSet [Sami, Jake, Declan, Allison] From parameterized values: And an ArrayList<String> of votes [“1,,,”,”2,1,3,4”, “,,,1”, “,,1,”, “4,2,1,3”]	5 new ballots created respective to the votes list that was passed in; [1: Sami] [2: Sami, 1: Jake, 3: Declan, 4: Allison] [1: Allison] [1: Declan] [4: Sami, 2: Jake, 1: Declan, 3:Allison]	[1: Sami] [2: Sami, 1: Jake, 3: Declan, 4: Allison] [1: Allison] [1: Declan] [4: Sami, 2: Jake, 1: Declan, 3:Allison]	

Post condition(s) for Test: 5 new ballots have been created.

Test Stage: Unit **X** System __ **Test Case ID#**: testBallotGetVotes **Test Description:** This test will look at the getVotes functionality in the Ballot class. It will compare the expected candidates name to the actual name the candidate has in the voteMap that is returned by this function call. **Automated:** Yes **X** No __	**Test Date**: 3/30/2020 **Name(s) of Testers**: Sami Frank **Indicate where you are storing the test (what file) and the name of the method/functions being used:** BallotTest.java in /src directory. Test method: testGetVotes

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Ballot and Candidate class must compile with no errors. A LinkedHashSet of candidates has been created. Ballots have been created.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Call testGetVotes( )	7 ballots that have been created where the linkedHashSet of candidates corresponds to the vote strings A LinkedHashSet [Sami, Jake, Declan, Allison] And an ArrayList<String> of votes [“1,,,”,”2,1,3,4”, “,,,1”, “,,1,”, “4,2,1,3”, “1,,3,2”, “,1,2,3”]	[1: Sami] -> “Sami” [2: Sami, 1: Jake, 3: Declan, 4: Allison] -> “Jake” -> “Sami” -> “Declan” -> “Allison” [1: Allison] -> “Allison” [1: Declan] -> “Declan” [4: Sami, 2: Jake, 1: Declan, 3:Allison] -> “Sami” -> “Jake” -> “Declan” -> “Allison” [1: Sami, 3: Declan, 2: Allison] -> “Sami” -> “Declan” -> “Allison” [1: Jake, 2: Declan, 3: Allison]-> “Jake”->”Declan” -> “Allison”	“Sami” “Jake” -> “Sami” -> “Declan” -> “Allison” “Allison” “Declan” “Sami” -> “Jake” -> “Declan” -> “Allison” “Sami” -> “Declan” -> “Allison” “Jake”->”Declan” -> “Allison”	

Post condition(s) for Test: voteMaps for each respective ballot has returned the correct output given the test data.

Test Stage: Unit **X** System __	**Test Date**: 3/30/2020
Test Case ID#: testBallotGetBallotNum	**Name(s) of Testers**: Sami Frank
Test Description: This test will ensure that the Ballot class is keeping track of each ballot that is being created by giving	**Indicate where you are storing the test (what file) and the name of the method/functions being used:**

it a number. This number starts at 1 and increments every time a new ballot is created. **Automated:** Yes **X** No __	BallotTest.java in /src directory. Test method: testGetBallotNum

Results: Pass ___X___ Fail _____

Preconditions for Test: Ballot and Candidate class must compile with no errors. A LinkedHashSet of candidates has been created. 5 ballots have been created.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call testGetBallotNum()	7 ballots created from BallotTest constructor	1 2 3 4 5 6 7	1 2 3 4 5 6 7	

Post condition(s) for Test: Each ballot created has a unique number assigned to it, indicating in what order the ballot was created.

UI Class - Unit Tests

Test Stage: Unit **X** System __ **Test Case ID#**: testVoteTypePlurality **Test Description:** This test will test when you enter 1 in for the voteType prompt to set vote type to Plurality. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testVoteTypePlurality. The following method is being used: UI.promptVoteType

Results: Pass ___X___ Fail _____

Preconditions for Test: UI.java exists.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Set the UI.testing to true	UI.testing = true;			
2	Pass in a String of 1 into promptVoteType	UI.promptVoteType("1")			
3	AssertEquals on UI.voteType and "Plurality"		true	true	

Post condition(s) for Test: UI.voteType will be set to the string Plurality

---

Test Stage: Unit **X** System __	**Test Date**: April 1, 2020
Test Case ID#: testVoteTypeSTV	**Name(s) of Testers**: Declan Buhrsmith
Test Description: This test will test when you enter 1 in for the voteType prompt to set vote type to STV.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testVoteTypeSTV. The following method is being used: UI.promptVoteType
Automated: Yes **X** No __	

Results: Pass ___X___ Fail _____

Preconditions for Test: UI.java exists.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**

1	Set the UI.testing to true	UI.testing = true;			
2	Pass in a String of 1 into promptVoteType	UI.promptVoteType(“2”)			
3	AssertEquals on UI.voteType and “STV”		true	true	

Post condition(s) for Test: UI.voteType will be set to the string STV

Test Stage: Unit **X** System __ **Test Case ID#**: testNumSeats **Test Description:** This test will test when you enter a number into promptNumSeats that it is logged and set to the String UI.numSeats. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testNumSeats. The following method is being used: UI.promptNumSeats

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	@ParameterizedTest @ValueSource(strings = data	data = { "0", "1", "2", "4", "11", "999", "1000", "999999", "1000000" })			
2	UI.testing is set to true	UI.testing =			

		true;			
3	Pass in a numeric string into promptNumSeats	UI.promptNumSeats(value)			
4	AssertEquals that UI.numSeats = value		true	true	

Post condition(s) for Test: UI.numSeats will be set to value

Test Stage: Unit **X** System __ **Test Case ID#**: testFiles **Test Description:** This test will test when you enter a file, into the promptFiles method. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testFiles. The following method is being used: UI.promptFiles

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	UI.testing is set to true	UI.testing = true;			
2	Create a new File linking to a path in the project.	File testFile = new File("./testing/testWriteFile1.csv");			
3	UI.promptFile using	UI.promptFiles			

	the same path as the file you created in step 2.	("./testing/testWriteFile1.csv");			
4	AssertEquals on the UI.files[0] and, the test file you used to check with in this case.		true	true	

Post condition(s) for Test: UI.files will be set to an array containing testFile from step 2.

Test Stage: Unit **X** System __ **Test Case ID#**: testFilesMultiple **Test Description:** This test will test when you enter files into the promptFiles method. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testFilesMultiple. The following method is being used: UI.promptFiles File.getName

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Create a new string called uiFileNames	String uiFileNames = "";			
2	Create a new string called testFileNames	String testFileNames = "";			
3	UI.testing is set to true	UI.testing =			

		true;			
4	Create a new File linking to a path in the project.	File testFile1 = new File("./testing/testWriteFile1.csv");			
5	Create a new File linking to a path in the project.	File testFile2 = new File("./testing/testWriteFile2.csv");			
6	Set the string testFileNames to the File.getName() of each file.	testFileNames = testFile1.getName() + " " + testFile2.getName() + " ";			
7	Loop through the UI.Files Array and get the names of those files and set them to uiFileNames	for (File f : UI.files) { uiFileNames = uiFileNames + f.getName() + " "; }			
8	AssertEquals on the names of uiFileNames, and testFileNames	assertEquals(uiFileNames, testFileNames);	true	true	

Post condition(s) for Test: UI.files will contain an array of the two files passed in above.

Test Stage: Unit **X** System __ **Test Case ID#**: testFilesMultipleExsitsFileMismatch **Test Description:** This test will test when you	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions**

enter files into the promptFiles method. **Automated:** Yes **X** No __	**being used:** UITest.java in /src directory. Test Method: testFilesMultipleExsitsFileMismatch. The following method is being used: UI.promptFiles

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	UI.testing is set to true	UI.testing = true;			
2	Create a new File linking to a path in the project.	File testFile1 = new File("./testing/testWriteFile1.csv");			
3	Create a new File linking to a path in the project.	File testFile2 = new File("./testing/testWriteFile2.csv");			
4	Add the two new files to a File Array	File[] fileList = new File[]{testFile1,testFile2};			
5	UI.prompt on a string that is different to the path of the files you created in step 2, and 3.	UI.promptFiles("./testing/testWriteFile1.csv ./testing/testWriteFile2.csv");			
6	Assert Not Equals on UI.files and the fileList array created in step 4.	assertNotEquals(UI.files, fileList);	true	true	

Post condition(s) for Test: UI.files will contain an array of different files to the ones testing for.

Test Stage: Unit **X** System __ **Test Case ID#**: testFilesExsitsFileMismatch **Test Description:** This test will test when you enter a file into the prompt if it is the same as another file object you create with a different path. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testFilesMultipleExsitsFileMismatch. The following method is being used: UI.promptFiles

Results: Pass **X** Fail _____

Preconditions for Test: UI.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	UI.testing is set to true	UI.testing = true;			
2	Create a new File linking to a path in the project.	File testFile = new File("./testing/testWriteFile2.csv");			
5	UI.prompt on a string that is different to the path of the file you created in step 2	UI.promptFiles("./testing/testWriteFile1.csv");			
6	Assert Not Equals on UI.files[0] and the file created in step 2	assertNotEquals(UI.files[0],	true	true	

		testFile);			

Post condition(s) for Test: UI.files[0] will contain a different file then the one created in step 2.

Test Stage: Unit **X** System __ **Test Case ID#**: testTypeQuitProgram **Test Description:** This test if the program is to be closed. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testTypeQuitProgram. The following method is being used: UI.quitProgram

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set the UI.testing to true	UI.testing = true;			
2	UI.quitProgram is passed in with a true boolean	UI.quitProgram(true)			
3	Assert Equals on the UI.quitprogram boolean and true		true	true	In order to get around having a return type in this context because if you quit the program the test will also exit so I just have it take a

					variable that if it wasn't in the test the program would do System.exit(0) which quits the program.

Post condition(s) for Test: UI.quitProgram will be set to true and the program, if run actually, would quit.

Test Stage: Unit **X** System __ **Test Case ID#**: testDisplayResultsPass **Test Description:** This test is to test once you enter all the right information that displays the information as well. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testDisplayResultsPass. The following method is being used: UI.confirmInput UI.displayResults

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set the UI.testing boolean to true	UI.testing = true;			
2	Set the UI.voteType string to STV	UI.voteType = "STV";			
3	Set the UI.numSeats string to 1	UI.numSeats = "1";			

4	Create a new File linking to a path in the project.	File file = new File("./testing/testWriteFile2.csv");			
5	Add the two new files to a File Array	File[] fileList = new File[]{testFile};			
6	Set the UI.files to the temp File var in step 3	UI.files = temp;			
7	Run UI.confirmInput("1"), 1 means the input is confirmed.	UI.confirmInput("1");			
8	AssertEquals on UI.displayResults() and true	assertEquals(UI.displayResults(), true);	true	true	UI.displayResults() returns a string just for the sake if the file runs.

Post condition(s) for Test: UI.voteType will be STV, UI.numSeats will be 1, UI.files will be the file in step 4. displayResults will return true.

Test Stage: Unit **X** System __	**Test Date**: April 1, 2020
Test Case ID#: testDisplayResultsFails	**Name(s) of Testers**: Declan Buhrsmith
Test Description: This test that you did not enter the required information so it will return false.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testDisplayResultsFails. The following method is being used: UI.displayResults
Automated: Yes **X** No __	

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Set the UI.testing boolean to true	UI.testing = true;			
2	AssertEquals on UI.displayResults() and false	assertEquals(UI.displayResults(), false);	true	true	UI.displayResults() returns a string just for the sake if the file runs. Sense nothing is set with regards to election and what not from confirmInput() it would throw an error and displayResults would return false.

Post condition(s) for Test: UI.testing is set to true.

Test Stage: Unit **X** System __	**Test Date**: April 1, 2020
Test Case ID#: testDisplayHelp	**Name(s) of Testers**: Declan Buhrsmith
Test Description: This will test if you want to display help and if it works properly.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testDisplayHelp. The following method is being used: UI.displayHelp
Automated: Yes **X** No __	

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Set the UI.testing boolean to true	UI.testing = true;			
2	Run UI.displayHelp passing in x because that what you need to exit the program	UI.displayHelp ("x");			
3	AssertEquals on UI.restartProgram and true	assertEquals(UI.restartProgram, true);	true	true	UI.restartProgram is set if you press x in UI.displayHelp because after you displayHelp you restart the UI.

Post condition(s) for Test: UI.testing is set to true. UI.restartProgram is set to true.

Test Stage: Unit **X** System __	**Test Date**: April 1, 2020
Test Case ID#: testIsNumericPass	**Name(s) of Testers**: Declan Buhrsmith
Test Description: This will test if the string you pass in is a numeric string.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testIsNumericPass. The following method is being used: UI.displayHelp
Automated: Yes **X** No __	

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	@ParameterizedTest @ValueSource(strings = data	ata = { "0", "1", "2", "4", "11", "999", "1000", "999999", "1000000" })			
2	AssertEquals UI.isNumeric if it's true.	assertEquals(UI.isNumeric(value), true);	true	true	checks if the string value is a string that contains strictly numbers.art the UI.

Post condition(s) for Test:

Test Stage: Unit **X** System __ **Test Case ID#**: testIsNumericPass **Test Description:** This will test if the string you pass in is a numeric string. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testIsNumericPass. The following method is being used: String.allmatch

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	@ParameterizedTest @ValueSource(strings = data	data = { "0", "1", "2", "4", "11", "999", "1000", "999999", "1000000" })			
2	AssertEquals UI.isNumeric if it's true.	assertEquals(UI.isNumeric(value), true);	true	true	checks if the string value is a string that contains strictly numbers.art the UI.

Post condition(s) for Test:

Test Stage: Unit **X** System __ **Test Case ID#**: testIsNumericFail **Test Description:** This will test if the string you pass in is a numeric string. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testIsNumericFail. The following method is being used: String.allmatch

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes

1	@ParameterizedTest @ValueSource(strings = data	data = { "asd", "f1f1", "fasg2", "4a2f", "1ff1", "99as9", "100ff0", "999212gh999", "100hfd0000" }			
2	AssertEquals UI.isNumeric if it's false.	assertEquals(UI.isNumeric(value), false);	true	true	checks if the string value is a string that contains strictly numbers.art the UI.

Post condition(s) for Test:

Test Stage: Unit **X** System __ **Test Case ID#**: testConfirmInputRestart **Test Description:** This test is to test once you enter all the right information that displays the information as well. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testConfirmInputRestart. The following method is being used: UI.confirmInput

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	**Test Step Description**	**Test Data**	**Expected**	**Actual Result**	**Notes**

			Result		
1	Set the UI.testing boolean to true	UI.testing = true;			
2	Set the UI.voteType string to STV	UI.voteType = "STV";			
3	Set the UI.numSeats string to 1	UI.numSeats = "1";			
4	Create a new File linking to a path in the project.	File file = new File("./testing/testWriteFile2.csv");			
5	Add the two new files to a File Array	File[] fileList = new File[]{testFile};			
6	Set the UI.files to the temp File var in step 3	UI.files = temp;			
7	AssertEquals on UI.confirmInput("2"), and true	assertEquals(UI.confirmInput("2"), true);			2 means the input is not confirmed and the program will restart.

Post condition(s) for Test: UI.voteType will be STV, UI.numSeats will be 1, UI.files will be the file in step 4. confirmInput will return true.

Test Stage: Unit **X** System __ **Test Case ID#**: testConfirmInputRestart **Test Description:** This test is to test once you enter all the right information that displays the information as well.	**Test Date**: April 1, 2020 **Name(s) of Testers**: Declan Buhrsmith **Indicate where you are storing the test (what file) and the name of the method/functions being used:** UITest.java in /src directory. Test Method: testConfirmInputRestart.

Automated: Yes **X** No __	The following method is being used: UI.confirmInput

Results: Pass ___**X**___ Fail _____

Preconditions for Test: UI.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Set the UI.testing boolean to true	UI.testing = true;			
2	Set the UI.voteType string to STV	UI.voteType = "STV";			
3	Set the UI.numSeats string to 1	UI.numSeats = "1";			
4	Create a new File linking to a path in the project.	File file = new File("./testing/testWriteFile2.csv");			
5	Add the two new files to a File Array	File[] fileList = new File[]{testFile};			
6	Set the UI.files to the temp File var in step 3	UI.files = temp;			
7	AssertEquals on UI.confirmInput(“1”), and true	assertEquals(UI.confirmInput("1"), true);			1 means the input is confirmed

Post condition(s) for Test: UI.voteType will be STV, UI.numSeats will be 1, UI.files will be the file in step 4. confirmInput will return true.

Election Class - Unit Tests

Test Stage: Unit **X** System __ **Test Case ID#**: testConstructor **Test Description:** This test checks that the Election constructor is properly initialized, having set voteType, numSeats, files, and shuffle. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testConstructor. The following methods are being used: Election's constructor, getVoteType, getNumSeats, getFiles, and getShuffle methods.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize an array of File objects, using 1+ file objects initialized with a string value.	File[] files = { new File("file1.csv ") };			
2	Create a new election object, passing its constructor values for voteType, numSeats, files, and shuffle	voteType = "STV" numSeats = 5 files = files shuffle = false			
3	Check that election.getVoteType is equal to "STV"		"STV"	"STV"	
4	Check that election.getNumSeats is equal to 5		5	5	
5	Check that election.getFiles is equal to files		files	files	
6	Check that		false	false	

	election.getShuffle is equal to false				

Post condition(s) for Test: Election object is initialized with properly set fields.

Test Stage: Unit **X** System __ **Test Case ID#**: testSetCandidates **Test Description:** This test confirms that the candidates get properly set. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testSetCandidates The following methods are being used: Election's constructor, setCandidates, getCandidates and Candidate's getName

Results: Pass **X** Fail _____

Preconditions for Test: Election.java exists. Candidate.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(ints = data)	Data = { "", "Jake Waro", "A,B,C,D,E,F", "11,12,13,14", "Allison, Declan, Sami, Jake" }			Define test as a parameterized test. @ParameterizedTest
2	Initialize a new "STV" election.				
3	Create a String array, *expected*, of the				

	testHeader values split on commas.				
4	Call election.setCandidates passing the testHeader parameter value	testHeader			
5	Get the candidates using election.getCandidates ()	testHeader			
6	Check that all of the candidates from expected are in the retrieved list of candidates.		True	True	

Post condition(s) for Test: Election's candidates are properly set.

Test Stage: Unit **X** System __ **Test Case ID#**: testAddBallotToListSizeOne **Test Description:** This test confirms that we can add at least one ballot to the ballot list. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testAddBallotToListSizeOne The following methods are being used: Election's constructor, setCandidates, addBallotToList, and getBallotList.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java, Ballot.java, and Candidate.java exist.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**

1	Define value source for test. @ValueSource(ints = data)	Data = { "1,,,", "1,2,3,4", "3,,2,1", "1,2,,", ",,,1" }			Define test as a parameterized test. @ParameterizedTest
2	Create a new Election object				
3	Set the candidates on the election object passing the testValues string				
4	Create a new ballot object = election.addBallotToList(testValues)				
5	Retrieve the ballot list from election and verify its size is 1.		true	true	
6	Verify the ballot list contains the ballot object.		true	true	

Post condition(s) for Test: Successfully added a ballot to Election's ballot list.

Test Stage: Unit **X** System __ **Test Case ID#**: testAddBallotToListSizeMany **Test Description:** This tests that we can add many ballots to the ballot list. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testAddBallotToListSizeMany Methods used are Election's constructor, addBallotToList, setCandidates, and getBallotList.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java, Ballot.java, and Candidate.java exist.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Initialize an STV Election object and set its Candidates.	Candidates = "Jake,Allison, Declan,Sami"			
2	Create a test String record	"1,2,3,4"			
3	Initialize an empty LinkedHashSet<Ballot>, *expectedList*				
4	Assert *expectedList* is empty		true	true	
5	Add the result of calling election.addBalotToList(record) to *expectedList*				
6	Assert *expectedList* is equal to election.getBallotList		true	true	
7	Repeat step 5, 4 times, and repeat step 6.		true	true	
8	Repeat step 7.		true	true	

Post condition(s) for Test: Successfully added multiple ballots to Election's ballot list.

Test Stage: Unit **X** System __	**Test Date**: April 1, 2020
Test Case ID#: testWritingHeaders	**Name(s) of Testers**: Jake Waro

Test Description: Test that the election information section header and the candidates are properly being written to the audit file. **Automated:** Yes **X** No __	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testWritingHeaders The following methods are being used: Election's constructor, setTesting, and setCandidates.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java and Candidate.java exist.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Define value source for test. @ValueSource(ints = data)	Data = { "Jake,Declan,Sami,Allison", "Bob,Pam,Sia", "A", "Amy,Ben,Carl,Dean,Erin,Frank" }			Define test as a parameterized test. @Parameterized Test
2	Create a new election object				
3	Call election.setTesting() passing it a test audit file to write to.	./testing/testOutputs/testWritingHeaders_output.txt			
4	Create a new string, *expected*, that reflects the correct line that should be written for the candidates.				
5	Call election.setCandidates (testHeaders)				

6	Read in from the test audit file and verify the first three lines match the Election header format.		Line1 = "---------- ---------- ---------- --" Line2 = "---------- ELECTION INFORMATI ON ----------" Line3 = "---------- ---------- ---------- --"	Line1 = "---------- ---------- ---------- ----------" Line2 = "---------- ELECTION INFORMA TION ---------- --"Line3 = "---------- ---------- ---------- ----------"	
7	Assert that the fourth line read from the file is equal to *expected*		true	true	

Post condition(s) for Test: Election's candidates are properly set.

Test Stage: Unit **X** System __ **Test Case ID#**: testWriteToAudit **Test Description:** This test checks that we correctly write the parameterized number of ballots to the audit file. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testWriteToAudit Methods used are Election's constructor, setCandidates, setTesting, addBallotToList, writeToAudit, getCandidates, and Candidate's getName and getVotes.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java, Ballot.java, and Candidate.java exist.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Define value source for test. @ValueSource(ints = data)	numBallots = { 1, 2, 4, 7, 11, 99, 101, 999, 1001, 99999, 100001}			Define test as a parameterized test. @ParameterizedTest

2	Create a String array of test ballot values	{
		"1,2,3,4",
		"4,3,2,1",
		",3,2,1",
		"1,,2,",
		"4,3,2,1",
		",3,2,1",
		"1,,2,",
		",,,1",
		"4,1,2,3",
		"3,,1,2",
		}			

3	Setup the test doing the following: A new election object, Set the candidates, Set election to testing with a test audit file path				
4	Add numBallots amount of ballots to the election object, using the value found at numBallot % 10 index of the ballot values array.				
5	Read the ballots back from the test audit file				
6	Verify that the		true	true	

	numBallots line of the file is equal to the expected ballot num and votes				

Post condition(s) for Test: Ballots and descriptions are correctly written to the audit file.

Test Stage: Unit **X** System __ **Test Case ID#**: testWriteToAuditOverloaded **Test Description:** This test checks the writeToAudit overloaded method that only takes a String description correctly write numLines descriptions to the audit file. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testWriteToAuditOverloaded Methods used are Election's constructor, setTesting, and writeToAudit.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(ints = data)	numLines = { 1, 2, 4, 7, 11, 99, 999, 1001, 99999 }			Define test as a parameterized test. @ParameterizedTest
2	Create a new election object and set testing to a test audit file path				
3	Initialize a test String value	"thisIsATestString"			
4	Call				

	election.writeToAudit(testTring) numLines amount of times.				
5	Assert that the audit file has numLines amount of lines and they all read “thisIsATestString”		true	true	

Post condition(s) for Test: Each line read from the test overload file is equal to the test string, and there are numLines amount of lines in the test file.

Test Stage: Unit **X** System __ **Test Case ID#**: testProcessElection **Test Description:** This test verifies the block that gets called during processing an election. If it's an STV election is processed, "STV" is returned, if it's a Plurality election, "Plurality" is returned, any other election type should cause an illegal argument exception. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testProcessElection Methods used are Election’s constructor, and processElection.

Results: Pass **X** Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(ints = data)	testValue = { "STV", "Plurality", "aslfjasd#@%", "Neither STV nor Plurality" }			Define test as a parameterized test. @ParameterizedTest

2	Create new Election object, using testValue as the value passed to Election for voteType.				
3	If testValue is “STV” or “Plurality”, assert that election.processElection() is equal to testValue.		testValue	testValue	
4	If testValue is not “STV” or “Plurality”, assert that election.processElection() causes an IllegalArgumentException		IllegalArgumentException thrown	IllegalArgumentException thrown	

Post condition(s) for Test: the string returned matches the testValue if STV or Plurality and throws an exception otherwise.

Test Stage: Unit **X** System __

Test Case ID#: testCreateBallotList

Test Description: This test checks that a ballot list is correctly created from the input files.

Automated: Yes **X** No __

Test Date: April 1, 2020

Name(s) of Testers: Jake Waro

Indicate where you are storing the test (what file) and the name of the method/functions being used:

ElectionTest.java in /src directory.
Test Method: testCreateBallotList
Methods used are Election’s constructor, setTesting, createBallotList, getBallotList, and Ballot’s getBallotNum, getVotes, and Candidate’s getName.

Results: Pass **X** Fail _____

Preconditions for Test: Election.java, Ballot.java, and Candidate.java exist. And testWriteFile1.csv, testWriteFile2.csv, and testWriteFile1.csv are all populated with STV style votes.

Step #	**Test Step Description**	**Test Data**	**Expected Result**	**Actual Result**	**Notes**
1	Initialize an STV election object using testing files for the files object.	File[] files = { new File("./testing/testWriteFile1.csv"), new File("./testing/testWriteFile2.csv"), new File("./testing/testWriteFile3.csv") };			
2	Set testing to use a test audit file path.				
3	Call election.createBallotList()				
4	Read the first 4 lines from the audit file				
5	Verify each ballot in election.getBallotList() matches the line read from subsequent lines in the audit file.		true	true	
6	Assert that the numBallots == the size of the ballot list.		true	true	

Post condition(s) for Test: each ballot in the ballot list matches the respective ballot read from the test file.

Test Stage: Unit **X** System __	**Test Date**: April 1, 2020

Test Case ID#: testWriteFinalResults **Test Description:** This test checks that the final results of an election are correctly written to the audit file. **Automated:** Yes **X** No __	**Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testWriteFinalResults Methods used are Election's constructor, and processElection.

Results: Pass ___X___ Fail _____

Preconditions for Test: Election.java exists. Test file testWriteFile2.csv is populated with STV styled votes.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new STV election, and pass it a files object using a testing file. Set testing to use a test audit file.	Test file = new File("./testing/test WriteFile2.csv")			
2	Call election.processElection()				
3	Calculate Droop	Droop = 5			
4	Create a list of strings, *expected*, that reflect how the audit file should format before processing.				
5	Read each line in the test audit file and assert each line equals the respective string in *expected*.		true	true	Stop checking when you reach processing section.

Post condition(s) for Test: each string in the expected list matches the respective line read from the test audit file.

Test Stage: Unit **X** System __ **Test Case ID#**: testSetAuditWriterFalse **Test Description:** This test checks that the audit write value is not set to a new value when testing is not in process and the current process does not require the audit writer to write to a different file. **Automated:** Yes **X** No __	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testSetAuditWriterFalse Methods used are Elections's constructor and setAuditWriter.

Results: Pass **X** Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize a new Election STV object.				
2	Create a String variable equal to the return value of election.setAuditWriter(false)	result	""		
3	AssertEquals(result, "");		true	true	

Post condition(s) for Test: the setAuditWriter does not set the auditWriter to a new value.

Test Stage: Unit **X** System __	**Test Date**: April 1, 2020

Test Case ID#: testSetAuditWriterTrue **Test Description:** This test checks that when we call the audit writer with a true value, the audit writer gets set to write to the temporary audit file. **Automated:** Yes **X** No __	**Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testSetAuditWriterTrue Methods used are Elections's constructor and setAuditWriter.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize a new Election STV object.				
2	Create a String variable equal to the return value of election.setAuditWriter(true)	result	"temp"		
3	AssertEquals(result, "temp");		true	true	

Post condition(s) for Test: auditWriter writes to the temporary audit file.

Test Stage: Unit **X** System __ **Test Case ID#**: testSetAuditWriterTestSet **Test Description:** This test checks that when testing is in process, the auditWriter writes to the test audit file, and the audit file is overwritten as this is the first time the audit writer is set to write	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:**

to this audit file. **Automated:** Yes **X** No __	ElectionTest.java in /src directory. Test Method: testSetAuditWriterTestSet Methods used are Elections's constructor, setTesting, and setAuditWriter.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize a new Election STV object.				
2	Call election.setTesting() passing a test audit file path				
3	Create a String variable equal to the return value of election.setAuditWriter(false)	result	"testSet"		
4	AssertEquals(result, "testSet");		true	true	

Post condition(s) for Test: auditWriter writes to the test audit file, and the audit test file will be overwritten.

Test Stage: Unit **X** System __ **Test Case ID#**: testSetAuditWriterTest **Test Description:** This test checks that when testing is in process, the auditWriter writes to the test audit file by appending to it.	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory.

Automated: Yes **X** No __	Test Method: testSetAuditWriterTest Methods used are Elections's constructor, setTesting, and setAuditWriter.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize a new Election STV object.				
2	Call election.setTesting() passing a test audit file path				
3	Call election.setAuditWriter(false);				
4	Create a String variable equal to the return value of election.setAuditWriter(false)	result	"test"		
5	AssertEquals(result, "test");		true	true	

Post condition(s) for Test: auditWriter is set to append to the test audit file.

Test Stage: Unit **X** System __ **Test Case ID#**: runSTVelection **Test Description:** This is a manual test. This test runs an STV election using the provided test files in files variable. The results are saved to	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:**

VotingSystem_AuditFile_Group1.txt and should be visually inspected to verify the functionality is working. **Automated:** Yes ____ No **X**	ElectionTest.java in /src directory. Test Method: runSTVelection Methods used are Elections's constructor and processElection.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java and STV.java exist. Test files: testWriteFile1.csv, testWriteFile2.csv, and testWriteFile3.csv are populated with stv styled votes.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new File array using the test files.	File[] files = { new File("./testing/testWriteFile1.csv"), new File("./testing/testWriteFile2.csv"), new File("./testing/testWriteFile3.csv") };			
2	Initialize a new STV Election using the files variable.				
3	Call election.processElection()				
4	Manually inspect the VotingSystem_AuditFile_Group1.txt audit file.		Droop Quota: 1372 Winner(s): 1. Allison 2. Jake Loser(s): 1. Declan 2. Sami	Droop Quota: 1372 Winner(s): 1. Allison 2. Jake Loser(s): 1. Declan 2. Sami	

Post condition(s) for Test: STV election processed and a full audit file is available.

Test Stage: Unit **X** System __ **Test Case ID#**: runPluralityElection **Test Description:** This is a manual test. This test runs a Plurality election using the provided test file in the files variable. The results are saved to VotingSystem_AuditFile_Group1.txt and should be visually inspected to verify the functionality is working. **Automated:** Yes ____ No **X**	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: runPluralityElection Methods used are Elections's constructor and processElection.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java and Plurality.java exist. Test file electionPluralityTest1.csv is populated with plurality styled votes.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new File array using the test files.	File[] files = { new File("./testing/electionPluralityTest1.csv")};			
2	Initialize a new Plurality Election using the files variable.				
3	Call election.processElection()				
4	Manually inspect the VotingSystem_AuditFile_Group1.txt audit file.		Winner(s): 1. Kevin Hart (36 votes) 2. Micky Mouse	Winner(s): 1. Kevin Hart (36 votes) 2. Micky Mouse	

			(26 votes) Loser(s): 1. Sharon (23 votes) 2. Joe Exotic (22 votes) 3. Henry Ford (21 votes) 4. Joy (21 votes) 5. Karen (20 votes) 6. Amelia Earhart (14 votes)	(26 votes) Loser(s): 1. Sharon (23 votes) 2. Joe Exotic (22 votes) 3. Henry Ford (21 votes) 4. Joy (21 votes) 5. Karen (20 votes) 6. Amelia Earhart (14 votes)	

Post condition(s) for Test: Plurality election processed and a full audit file is available.

Test Stage: Unit **X** System __ **Test Case ID#**: testGetVoteType **Test Description:** This test checks that the voteType variable is properly retrieved from the election object. **Automated:** Yes **X** No ____	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testGetVoteType Methods used are Elections's constructor and getVoteType.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source	voteType = {			Define test as a

	for test. @ValueSource(strings = data)	"STV", "Plurality", "aslfjasd#@%", "" }			parameterized test. @ParameterizedTest
2	Initialize a new election object, passing the voteType parameter as the voteType to the election.				
3	Set the value of String *result* to a call to election.getVoteType( )	result	voteType		
4	assertEquals(voteType , result)		true	true	

Post condition(s) for Test: the vote type matches the vote type value passed to the election constructor.

Test Stage: Unit **X** System __ **Test Case ID#**: testGetNumSeats **Test Description:** This test checks that the numSeats variable is properly retrieved from the election object. **Automated:** Yes **X** No ____	**Test Date**: April 1, 2020 **Name(s) of Testers**: Jake Waro **Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testGetNumSeats Methods used are Elections's constructor and getNumSeats.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(ints = data)	testValue = { 0, 1, 2, 3, 4, 7, 11, 99, 999 }			Define test as a parameterized test. @ParameterizedTest
2	Initialize a new election object, passing the testValue parameter as the numSeats to the election.				
3	Set the value of int *result* to a call to election.getVoteType( )	result	testValue		
4	assertEquals(testValue , result)		true	true	

Post condition(s) for Test: the number of seats matches the number of seats passed to the election constructor.

Test Stage: Unit **X** System __

Test Case ID#: testGetNumCandidates

Test Description: This test checks that the number of candidates matches the number of candidates found in the first line of the files.

Automated: Yes **X** No ____

Test Date: April 1, 2020

Name(s) of Testers: Jake Waro

Indicate where you are storing the test (what file) and the name of the method/functions being used:

ElectionTest.java in /src directory.
Test Method: testGetNumCandidates
Methods used are Elections's constructor, setCandidates, and getCandidates.

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java and Candidate.java exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(strings = data)	testValue = { "a,b,c,d,e,f", "ab,cd", "Jake, Sami, Declan, Allison", "a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p" }			Define test as a parameterized test. @ParameterizedTest
2	Set the variable *expected* equal to the length of the array of the testValue string split on a comma.	*expected*			
3	Call election.setCandidates (testValue)				
4	assertEquals(expected, election.getCandidates ().size())		true	true	

Post condition(s) for Test: the number of candidates matches the number of candidates in the comma delimited string passed to election.setCandidates().

Test Stage: Unit **X** System __	**Test Date**: April 1, 2020
Test Case ID#: testGetShuffle	**Name(s) of Testers**: Jake Waro
Test Description: This test checks that the shuffle value matches the value passed to the election constructor.	**Indicate where you are storing the test (what file) and the name of the method/functions being used:** ElectionTest.java in /src directory. Test Method: testGetShuffle Methods used are Elections's constructor and getShuffle.
Automated: Yes **X** No ____	

Results: Pass ___**X**___ Fail _____

Preconditions for Test: Election.java exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Define value source for test. @ValueSource(booleans = data)	testValue = { true, false }			Define test as a parameterized test. @ParameterizedTest
2	Create a new election object, passing testValue as the value for the shuffle argument.	testValue			
3	assertEquals(testValue, election.getShuffle())		testValue	testValue	

Post condition(s) for Test: election.getShuffle() matches the value passed to election's constructor.