Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit **Test Date: 03/09/2021**

Test Case ID#: Can_001 **Name(s) of Testers: Noah Park**

Test Description: Tests normal Candidate constructor.

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's constructor. The test is called testCandidateConstructor().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create Candidate object	Name: "John Doe" Party: "test" CandidateID: 1	Candidate object with its name as John Doe, party as test, candidateID as 1, and curNumVotes as 0.	Candidate object with its name as John Doe, party as test, candidateID as 1, and curNumVotes as 0.	

Post condition(s) for Test: Candidate object successfully initialized.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can002

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from constructor on null name argument

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's constructor. The test is called testCandidateConstructorThrowsExceptionNullName().

Results: Pass

Preconditions for Test: Candidate class must exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Try to create a Candidate object	Name: null Party: "test" CandidateID: 1	Throws IllegalArgumentException	Throws IllegalArgumentException	
2	Catch IllegalArgumentException thrown from the constructor call.		Caught IllegalArgumentException	Caught IllegalArgumentException	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_003

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from constructor on empty name argument

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's constructor. The test is called testCandidateConstructorThrowsExceptionEmptyName().

Results: Pass

Preconditions for Test: Candidate class must exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Try to create a Candidate object	Name: null Party: "test" CandidateID: 1	Throws IllegalArgumentException	Throws IllegalArgumentException	
2	Catch IllegalArgumentException thrown from the constructor call.		Caught IllegalArgumentException	Caught IllegalArgumentException	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_004

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from constructor on null party argument

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's constructor. The test is called testCandidateConstructorThrowsExceptionNullParty().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Try to create a Candidate object	Name: "John Doe" Party: null CandidateID: 1	Throws IllegalArgumentException	Throws IllegalArgumentException	
2	Catch IllegalArgumentException thrown from the constructor call.		Caught IllegalArgumentException	Caught IllegalArgumentException	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_005

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from constructor on empty party argument.

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's constructor. The test is called testCandidateConstructorThrowsEmptyParty().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Try to create a Candidate object	Name: null Party: "" CandidateID: 1	Throws IllegalArgumentException	Throws IllegalArgumentException	
2	Catch IllegalArgumentException thrown from the constructor call.		Caught IllegalArgumentException	Caught IllegalArgumentException	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_006

Name(s) of Testers: Noah Park

Test Description: Tests get name method

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's getName() method. The test is called testGetName().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a Candidate object.	Name: "John Doe" Party: "test" CandidateID: 1	Candidate object created.	Candidate object created.	
2	Ensure candidate's name from getName() is "John Doe"		John Doe	John Doe	

Post condition(s) for Test: Name of the candidate is returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_007

Name(s) of Testers: Noah Park

Test Description: Tests get Candidate ID method.

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's getCandidateID() method. The test is called testGetCandidateID().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a Candidate object.	Name: "John Doe" Party: "test" CandidateID: 1	Candidate object created.	Candidate object created.	
2	Ensure candidate's ID from the getCandidateID() method is 1.		1	1	

Post condition(s) for Test: Candidate ID is returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_008

Name(s) of Testers: Noah Park

Test Description: Tests get party method.

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's getParty() method. The test is called testGetParty().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a Candidate object.	Name: "John Doe" Party: "test" CandidateID: 1	Candidate object created.	Candidate object created.	
2	Ensure candidate's party from getParty() method is test		test	test	

Post condition(s) for Test: Candidate party is returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_009

Name(s) of Testers: Noah Park

Test Description: Tests set number of votes and get number of votes methods.

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's getCurNumVotes() and setCurNumVotes() methods. The test is called testSetAndGetCurNumVotes().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a Candidate object.	Name: "John Doe" Party: "test" CandidateID: 1	Candidate object created.	Candidate object created.	
2	Update candidate votes using setCurNumVotes() method.	1000	Candidate votes are 1000	Candidate votes are 1000	
3	Obtain candidate votes using getCurNumVotes() method.		1000	1000	

Post condition(s) for Test: Candidate number of votes is updated and returned properly.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_010

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from set number of votes method on negative votes argument.

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's setCurNumVotes() method. The test is called testSetCurNumVotesException().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a Candidate object.	Name: "John Doe" Party: "test" CandidateID: 1	Candidate object created.	Candidate object created.	
2	Try to set the number of votes to –1 using setCurNumVotes() method.	-1	Exception thrown	Exception thrown	
3	Catch IllegalArgumentException		Caught IllegalArgumentException	Caught IllegalArgumentException	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_011

Name(s) of Testers: Noah Park

Test Description: Tests increment num votes method.

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's incrementCurNumVotes() method. The test is called testIncrementCurNumVotes().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a Candidate object.	Name: "John Doe" Party: "test" CandidateID: 1	Candidate object created.	Candidate object created.	
2	Set then Increment Candidate votes using incrementCurNumVotes method.	1000	Candidate votes would be at 2000	Candidate votes are at 2000	
3	Ensure Candidate votes are at 2000.		2000	2000	

Post condition(s) for Test: Candidate votes were incremented correctly.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/09/2021

Test Case ID#: Can_012

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from increment number of votes method on negative votes argument.

Automated: yes

This test is stored in CandidateTest.java and uses the Candidate class's incrementCurNumVotes() method. The test is called testIncrementCurNumVotesException().

Results: Pass

Preconditions for Test: Candidate class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a Candidate object.	Name: "John Doe" Party: "test" CandidateID: 1	Candidate object created.	Candidate object created.	
2	Try to increment curNumVotes by –1	-1	Exception thrown	Exception thrown	
3	Catch IllegalArgumentException		Catch exception	Caught exception	
4					

Project Name: Project 1: Voting System **Team#6**

Post condition(s) for Test: IllegalArgumentException caught.

Test Stage: Unit **Test Date: 03/10/2021**

Test Case ID#: Par_001 **Name(s) of Testers: Noah Park**

Test Description: Tests normal Party constructor.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's constructor. The test is called testPartyConstructor().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize Party object.	Name: "test"	Party object with "test" as its name, and 0 for all of its other attributes.	Party object with "test" as its name, and 0 for all of its other attributes.	

Post condition(s) for Test: Party object initialized correctly.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_002

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from constructor on null name argument.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's constructor. The test is called testPartyConstructorThrowsExceptionNullName().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize Party object.	Null name	Exception thrown	Exception thrown	
2	Catch Exception		Catch exception	Caught exception	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_003

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from constructor on empty name argument

Automated: yes

This test is stored in PartyTest.java and uses the Party class's constructor. The test is called testPartyConstructorThrowsExceptionEmptyName().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize Party object.	Name: ""	Exception thrown	Exception thrown	
2	Catch Exception		Catch exception	Caught exception	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_004

Name(s) of Testers: Noah Park

Test Description: Tests get name method.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's getName() method. The test is called testGetName().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Ensure party name from getName() is "test"		"test"	"test"	

Post condition(s) for Test: Name of party is returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_005

Name(s) of Testers: Noah Park

Test Description: Tests add candidate method to ensure candidates are added to the candidates list.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's addCandidate() and getCandidates() methods. The test is called testAddCandidate().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Create and add three candidates to the party	3 Candidate objects	Successful addition	Successful addition	
3	Ensure candidates were added to the party		True	True	

Post condition(s) for Test: Candidates are added to the party.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_006

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from add candidate method on null candidate argument.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's addCandidate() method. The test is called testAddCandidateThrowsException().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Try to add null candidate	null	Exception thrown	Excpetion thrown	
3	Catch Exception		Catch exception	Caught exception	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_007

Name(s) of Testers: Noah Park

Test Description: Tests get candidates method.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's getCandidates() method. The test is called testGetCandidates().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Create and add three candidates to the party	3 Candidate objects	Successful addition	Successful addition	
	Ensure candidates were added to the party		True	True	

Post condition(s) for Test: Candidates are returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_008

Name(s) of Testers: Noah Park

Test Description: Tests set total votes method and get total votes method.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's setTotalVotes() and getTotalVotes() methods. The test is called testSetAndGetTotalVotes().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Set total votes of the party to 1000	1000	Updated votes to 1000	Updated votes to 1000	
3	Ensure votes were set to 1000 using getter		1000	1000	

Post condition(s) for Test: Total votes are updated and returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_009

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from set total votes method on negative votes argument.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's setTotalVotes() method. The test is called testSetTotalVotesException().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Try to set total votes to –1	-1	Exception thrown	Exception thrown	
3	Catch exception		Catch exception	Caught exception	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_010

Name(s) of Testers: Noah Park

Test Description: Tests set remaining votes method and get remaining votes method.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's setRemainingVotes() and getRemainingVotes() method. The test is called testSetAndGetRemainingVotes().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Set remaining votes to 1000	1000	Remaining votes is 1000	Remaining votes is 1000	
3	Ensure remaining votes is 1000		1000	1000	

Post condition(s) for Test: Remaining votes is updated and returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_011

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from set remaining votes method on negative votes argument.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's setRemainingVotes() method. The test is called testSetRemainingVotesException().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Set remaining votes to –1	-1	Exception thrown	Exception thrown	
3	Catch exception		Exception caught	Exception caught	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_012

Name(s) of Testers: Noah Park

Test Description: Tests set number of seats method and get number of seats method.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's setNumberOfSeats() and getNumberofSeats() methods. The test is called testSetAndGetNumberOfSeats().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Sets number of seats to 10.	10	Updated number of seats to 10	Updated number of seats to 10	
3	Ensure number of seats is 10		10	10	

Post condition(s) for Test: Number of seats is updated and returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_013

Name(s) of Testers: Noah Park

Test Description: Tests exception thrown from set number of seats method on negative votes argument.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's setNumberOfSeats() method. The test is called testSetNumberOfSeatsException().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Sets number of seats to -1	-1	Exception thrown	Exception thrown	
3	Catch exception		Catch exception	Caught exception	

Post condition(s) for Test: IllegalArgumentException caught.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_014

Name(s) of Testers: Noah Park

Test Description: Tests get number of candidates method to ensure the correct of candidates are added to the candidates list.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's getNumCandidates() and addCandidate() method. The test is called testGetNumCandidates().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Adds three Candidates	3 Candidate objects			
3	Ensure number of candidates is 3.		3	3	

Post condition(s) for Test: Number of candidates is returned.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: Par_015

Name(s) of Testers: Noah Park

Test Description: Tests get candidate names to ensure all candidate names are output correctly.

Automated: yes

This test is stored in PartyTest.java and uses the Party class's getCandidateNames() and addCandidate() methods. The test is called testGetCandidateNames().

Results: Pass

Preconditions for Test: Party class must exist.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Initialize party object.	Name: "test"	Party object initialized	Party object initialized	
2	Adds three Candidates	3 Candidate objects			
3	Obtain candidate names.		Candidate names as a list	Candidate names as a list	

Post condition(s) for Test: Candidate names are returned as a list.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/10/2021**

Test Case ID#: OPLUnit_001

Name(s) of Testers: Michael Markiewicz

Test Description: Tests the handleTie function that the OPL class uses from Election for illegal arguments.

Automated: yes

This test is stored in OPLTest.java and uses the Election class's handleTie function. This test is called testHandleTieIllegalArguments().

Results: Pass

Preconditions for Test: handleTie function must exist in Election.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call handleTie on 4 illegal arguments (nonpositive numbers).	illegalArguments: {0, -1, -10, -100}	handleTie throws an IllegalArgumentException for each illegal argument.	An IllegalArgumentException was thrown for each illegal argument.	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_002

Name(s) of Testers: Michael Markiewicz

Test Description: Tests normal OPL Constructor

Automated: yes

This test is stored in OPLTest.java and uses the OPL class's constructor. This test is called testOPLConstructor().

Results: Pass

Preconditions for Test: The OPL, Candidate, and Party class exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 10 basic parties with 10 candidates each.	Parties: parties in the OPL Election	Parties are created without error.	Parties are created without error.	
2	Create new OPL instance.	totalNumBallots: 5520 numSeatsAvailable: 3	OPL instance should be created without error.	OPL instance was created without error.	
3	Check that quota was set correctly.	quota: 1840	The quota from the OPL instance matches the expected quota.	Quotas matched.	
4	Check that the number of seats available was set correctly.	numSeatsAvailable: 3	The number of seats from the OPL instance matches the number of seats passed to constructor.	Number of seats available matched.	
5	Check that the total number of ballots was set correctly.	totalNumBallots: 5520	The number of ballots from the OPL instance matches the total number of ballots passed to constructor.	Number of ballots matches.	

Post condition(s) for Test: OPL instance was created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_003

Name(s) of Testers: Michael Markiewicz

Test Description: Tests the OPL constructor for a null party list to see if an IllegalArgumentException is thrown.

Automated: yes

This test is stored in OPLTest.java and uses the OPL class's constructor. This test is called testOPLConstructorThrowsExceptionNullParties().

Results: Pass

Preconditions for Test: The OPL, Candidate, and Party class exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new OPL instance with the parties parameter noninitialized (null).	totalNumBallots: 10 numSeatsAvailable: 10 parties: null	IllegalArgumentException is thrown.	IllegalArgumentException is thrown.	
2	Catch IllegalArgumentException and compare error message to expected error message.	None	The exception thrown has the expected error message: "totalNumBallots must be positive, numSeatsAvailable must be positive, and parties must not be null and contain at least one party".	The exception was thrown with the correct error message.	

Post condition(s) for Test: IllegalArgumentException is thrown which means no instance of OPL is created. Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_004

Name(s) of Testers: Michael Markiewicz

Test Description: Tests the OPL constructor for an empty party list to see if an IllegalArgumentException is thrown.

Automated: yes

This test is stored in OPLTest.java and uses the OPL class's constructor. This test is called testOPLConstructorThrowsExceptionEmptyParties().

Results: Pass

Preconditions for Test: The OPL, Candidate, and Party class exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new OPL instance with the parties parameter initialized but empty with no parties.	totalNumBallots: 10 numSeatsAvailable: 10 parties: initialized	IllegalArgumentException is thrown.	IllegalArgumentException is thrown.	
2	Catch IllegalArgumentException and compare error message to expected error message.	None	The exception thrown has the expected error message: "totalNumBallots must be positive, numSeatsAvailable must be positive, and parties must not be null and contain at least one party".	The exception was thrown with the correct error message.	

Post condition(s) for Test: IllegalArgumentException is thrown which means no instance of OPL is created. Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_005

Name(s) of Testers: Michael Markiewicz

Test Description: Tests the OPL constructor for an nonpositive number of ballots to see if an IllegalArgumentException is thrown.

Automated: yes

This test is stored in OPLTest.java and uses the OPL class's constructor. This test is called testOPLConstructorThrowsExceptionNonPositiveNumBallots().

Results: Pass

Preconditions for Test: The OPL, Candidate, and Party class exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new OPL instance multiple times with the numBallots being nonpositive.	totalNumBallots: {-100, -50, -1, 0} numSeatsAvailable: 10 parties: initialized	IllegalArgumentException is thrown.	IllegalArgumentException is thrown.	
2	Catch IllegalArgumentException and compare error message to expected error message for each OPL instance that is created.	None	The exception thrown has the expected error message: "totalNumBallots must be positive, numSeatsAvailable must be positive, and parties must not be null and contain at least one party".	The exception was thrown with the correct error message.	

Post condition(s) for Test: IllegalArgumentException is thrown which means no instance of OPL is created. Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/10/2021**

Test Case ID#: OPLUnit_006

Name(s) of Testers: Michael Markiewicz

Test Description: Tests the OPL constructor for a nonpositive number of seats available to see if an IllegalArgumentException is thrown.

Automated: yes

This test is stored in OPLTest.java and uses the OPL class's constructor. This test is called testOPLConstructorThrowsExceptionNonPositiveSeats().

Results: Pass

Preconditions for Test: The OPL, Candidate, and Party class exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new OPL instance multiple times with the numSeats being nonpositive.	totalNumBallots: 10 numSeatsAvailable: {-100, -50, -1, 0} parties: initialized	IllegalArgumentException is thrown.	IllegalArgumentException is thrown.	
2	Catch IllegalArgumentException and compare error message to expected error message for each OPL instance that is created.	None	The exception thrown has the expected error message: "totalNumBallots must be positive, numSeatsAvailable must be positive, and parties must not be null and contain at least one party".	The exception was thrown with the correct error message.	

Post condition(s) for Test: IllegalArgumentException is thrown which means no instance of OPL is created. Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/10/2021**

Test Case ID#: OPLUnit_007

Name(s) of Testers: Michael Markiewicz

Test Description: Determines whether the audit and media files are created when the OPL algorithm runs.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLRunVotingAlgorithmCheckFilesCreated().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 5 parties with different number of candidates.	numParties: 5 numCandidatesPerParty: {3, 2, 6, 3, 1}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Create a new OPL instance.	numBallots: 10000 numSeats: 2	New instance of OPL is created.	The instance was created without errors.	
3	Run the voting algorithm.	OPL instance	Algorithm runs without errors.	No errors occurred.	
4	Check if the audit file and media report were created with correct filenames.	None	The audit file has the expected name of "OPLAuditFile_<year>_<month>_<day>_<time>.csv" and the media file has the expected name of "OPLMediaReport_<year>_<month>_<day>_<time>.csv"	These files were found and correctly named.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_008

Name(s) of Testers: Michael Markiewicz

Test Description: Tests that the number of seats each party has earned after the first allocation matches how many seats they should have.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLRunVotingAlgorithmFirstAllocation().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {3, 2, 6, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 3000, 4000, 100}	The party votes are populated.	The parties were successfully populated.	
3	Create a new OPL instance.	numBallots: 9100 numSeats: 3	New instance of OPL is created.	The instance was created without errors.	No ties should occur with these votes.
4	Run the voting algorithm.	OPL instance	Algorithm runs without errors.	No errors occurred.	
5	Get the first allocation votes from the OPL instance.	None	The first allocation for each party is obtained without any issues.	The first allocation for each party was successfully obtained.	

6	Compare the expected number of seats for each party and the expected remaining votes after the first allocation with the actual.	firstAllocation statistics	The number of seats and remaining number of votes for each party matches the expected.	For each party, the number of remaining votes and number of seats earned matched the expected for each.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_009

Name(s) of Testers: Michael Markiewicz

Test Description: Tests that the final number of seats each party has earned after the second allocation matches how many seats they should have.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLRunVotingAlgorithmSecondAllocationNumSeats().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {3, 2, 6, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 3000, 4000, 100}	The party votes are populated.	The parties were successfully populated.	
3	Create a new OPL instance.	numBallots: 9100 numSeats: 5	New instance of OPL is created.	The instance was created without errors.	No ties should occur with these votes.
4	Run the voting algorithm.	OPL instance	Algorithm runs without errors.	No errors occurred.	
5	Compare the number of seats earned by each party to the	Expected number of seats: {1, 2, 2, 0}	The expected number of seats for each party will match the actual.	The expected matches the actual.	

	expected.				

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_010

Name(s) of Testers: Michael Markiewicz

Test Description: Tests that the winners of a normal OPL election are the expected winners.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLRunVotingAlgorithmCheckWinners().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {3, 2, 3, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 3000, 4000, 100}	The party votes are populated.	The parties were successfully populated.	
3	Populate the candidates with a certain number of votes.	candidateVotes: {{1200, 500, 300}, {2000,1000}, {500,1000,2500}, {300,200}}	The candidates were populated.	The candidates were successfully populated.	This was designed in such a way where no ties should occur.
4	Create a new OPL instance.	numBallots: 9100 numSeats: 5	New instance of OPL is created.	The instance was created without errors.	No ties should occur with these votes.

5	Run the voting algorithm.	OPL instance	Algorithm runs without errors.	No errors occurred.	
6	Compare the winners of the election to the expected winners.	Expected winners: c0_p0, c0_p1, c1_p1, c1_p2, c2_p2	The winners of the election match the expected winners.	The expected winners matched the actual winners.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_011

Name(s) of Testers: Michael Markiewicz

Test Description: Checks that the output to the console, the media report, and the audit file match the expected formatting for a regular election.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLOutput().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {3, 2, 3, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 3000, 4000, 100}	The party votes are populated.	The parties were successfully populated.	
3	Populate the candidates with a certain number of votes.	candidateVotes: {{1200, 500, 300}, {2000,1000}, {500,1000,2500}, {300,200}}	The candidates were populated.	The candidates were successfully populated.	This was designed in such a way where no ties should occur.
4	Create a new OPL instance.	numBallots: 9100 numSeats: 5	New instance of OPL is created.	The instance was created without errors.	No ties should occur with these votes. This means the output is the same each time.

5	Run the voting algorithm.	OPL instance	Algorithm runs without errors.	No errors occurred.	
6	Compare the output to the terminal to the expected output.	Expected Output: ExampleOPLOutput.txt	The actual output string matches the expected output string.	The output matches.	
7	Compare the audit output to the expected audit output.	Expected Audit Output: ExampleOPLAudit.txt	The actual audit string matches the expected audit output string.	The output matches.	
8	Compare the media output to the expected media output.	Expected Media Output: ExampleOPLMedia.txt	The actual media string matches the expected media output string.	The output matches.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_012

Name(s) of Testers: Michael Markiewicz

Test Description: Checks that the output to the console, the media report, and the audit file match the expected formatting when the number of total candidates is less than the number of seats available in the election.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLLessCandidatesThanSeats().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {3, 2, 3, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 3000, 4000, 100}	The party votes are populated.	The parties were successfully populated.	
3	Populate the candidates with a certain number of votes.	candidateVotes: {{1200, 500, 300}, {2000,1000}, {500,1000,2500}, {300,200}}	The candidates were populated.	The candidates were successfully populated.	This was designed in such a way where no ties should occur.
4	Create a new OPL instance.	numBallots: 9100	New instance of OPL is created.	The instance was created without errors.	There are 11 seats up for grabs and only 10 available candidates.

		numSeats: 11			Every candidate should win a seat.
5	Run the voting algorithm.	OPL instance	Algorithm runs without errors.	No errors occurred.	
6	Compare the output to the terminal to the expected output.	Expected Output: ExampleOPLOutput_MoreSeatsThanCandidates.txt	The actual output string matches the expected output string.	The output matches.	
7	Compare the audit output to the expected audit output.	Expected Audit Output: ExampleOPLAudit_MoreSeatsThanCandidates.txt	The actual audit string matches the expected audit output string.	The output matches.	
8	Compare the media output to the expected media output.	Expected Media Output: ExampleOPLMedia_MoreSeatsThanCandidates.txt	The actual media string matches the expected media output string.	The output matches.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_013

Name(s) of Testers: Michael Markiewicz

Test Description: Checks that the output to the console, the media report, and the audit file match the expected formatting when there are ties between Candidates for a seat.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLTiedCandidatesExpectedOutput().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {4, 2, 3, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 3000, 4000, 100}	The party votes are populated.	The parties were successfully populated.	
3	Populate the candidates with a certain number of votes.	candidateVotes: {{600,600,600,200}, {2000,1000}, {500,1000,2500}, {300,200}}	The candidates were populated.	The candidates were successfully populated.	A tie is going to occur between candidates 0-2 in party 1.
4	Create a new OPL instance.	numBallots: 9100 numSeats: 5	New instance of OPL is created.	The instance was created without errors.	

5	Run the voting algorithm 100 different times and compare the output for each run-through.	OPL instance	Algorithm runs without errors.	No errors occurred.	
6	Compare the output to the terminal to the possible expected output.	Expected Output: ExampleOPLOutput_TiedCandidates0.txt ExampleOPLOutput_TiedCandidates1.txt ExampleOPLOutput_TiedCandidates2.txt	The actual output string matches one of the expected output string.	The output matches.	
7	Compare the audit output to the possible expected audit output.	Expected Audit Output: ExampleOPLAudit_TiedCandidates0.txt ExampleOPLAudit_TiedCandidates1.txt ExampleOPLAudit_TiedCandidates2.txt	The actual audit string matches one of the expected audit output string.	The output matches.	
8	Compare the media output to the possible expected media output.	Expected Media Output: ExampleOPLMedia_TiedCandidates0.txt ExampleOPLMedia_TiedCandidates1.txt ExampleOPLMedia_TiedCandidates2.txt	The actual media string matches one of the expected media output string.	The output matches.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_014

Name(s) of Testers: Michael Markiewicz

Test Description: Checks that the output to the console, the media report, and the audit file match the expected formatting when there are ties between Parties for a seat.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLTiedPartiesExpectedOutput().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {1, 1, 1, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 2000, 2000, 100}	The party votes are populated.	The parties were successfully populated.	A tie is going to occur between parties 0-2.
3	Create a new OPL instance.	numBallots: 6100 numSeats: 1	New instance of OPL is created.	The instance was created without errors.	
4	Run the voting algorithm 100 different times and compare the output for each run-through.	OPL instance	Algorithm runs without errors.	No errors occurred.	

5	Compare the output to the terminal to the possible expected output.	Expected Output: ExampleOPLOutput_TiedParties0.txt ExampleOPLOutput_TiedParties1.txt ExampleOPLOutput_TiedParties2.txt	The actual output string matches one of the expected output string.	The output matches.	
6	Compare the audit output to the possible expected audit output.	Expected Audit Output: ExampleOPLAudit_TiedParties0.txt ExampleOPLAudit_TiedParties1.txt ExampleOPLAudit_TiedParties2.txt	The actual audit string matches one of the expected audit output string.	The output matches.	
7	Compare the media output to the possible expected media output.	Expected Media Output: ExampleOPLMedia_TiedParties0.txt ExampleOPLMedia_TiedParties1.txt ExampleOPLMedia_TiedParties2.txt	The actual media string matches one of the expected media output string.	The output matches.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_015

Name(s) of Testers: Michael Markiewicz

Test Description: Checks that the output to the console, the media report, and the audit file match the expected formatting when there are ties between Parties and candidates for a seat.

Automated: yes

This test is stored in OPLTest.java and uses the OPL runVotingAlgorithm function. This test is called testOPLTiedPartiesAndTiedCandidatesExpectedOutput().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 3 parties with different number of candidates.	numParties: 3 numCandidatesPerParty: {2, 2, 1}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 2000, 100}	The party votes are populated.	The parties were successfully populated.	A tie is going to occur between parties 0 and 1.
3	Populate the candidates with a certain number of votes.	candidateVotes: {{1000, 1000}, {1000, 1000}, {100}}	The candidates were populated.	The candidates were successfully populated.	No matter whether party 0 or 1 wins the tie for earning a seat, the two candidates in each party is going to tie in popular votes.
4	Create a new OPL instance.	numBallots: 4100	New instance of OPL is	The instance was created without	

		numSeats: 1	created.	errors.	
5	Run the voting algorithm 100 different times and compare the output for each run-through.	OPL instance	Algorithm runs without errors.	No errors occurred.	
6	Compare the output to the terminal to the possible expected output.	Expected Output: ExampleOPLOutput_TiedCandidatesAndParties0.txt ExampleOPLOutput_TiedCandidatesAndParties0.txt ExampleOPLOutput_TiedCandidatesAndParties2.txt ExampleOPLOutput_TiedCandidatesAndParties3.txt	The actual output string matches one of the expected output string.	The output matches.	
7	Compare the audit output to the possible expected audit output.	Expected Audit Output: ExampleOPLAudit_TiedCandidatesAndParties0.txt ExampleOPLAudit_TiedCandidatesAndParties1.txt ExampleOPLAudit_TiedCandidatesAndParties2.txt ExampleOPLAudit_TiedCandidatesAndParties3.txt	The actual audit string matches one of the expected audit output string.	The output matches.	
8	Compare the media output to the possible expected media output.	Expected Media Output: ExampleOPLMedia_TiedCandidatesAndParties0.txt ExampleOPLMedia_TiedCandidatesAndParties1.txt ExampleOPLMedia_TiedCandidatesAndParties2.txt ExampleOPLMedia_TiedCandidatesAndParties3.txt	The actual media string matches one of the expected media output string.	The output matches.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the OPL class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_016

Name(s) of Testers: Michael Markiewicz

Test Description: Checks that the handleTie function is determining ties in a fairly uniform fashion.

Automated: no

This test is stored in OPLManualTest.java and uses Election's handleTie function. This test is called testHandleTie().

Results: Pass

Preconditions for Test: The Election class with the handleTie function exists.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	For a tie of size 5, run the handleTie function 100000 times and keep track of the frequencies of return values.	numPartiesInTie: 5	The handleTie function is called 100000 times without errors.	The handleTie function did not raise any errors for 100000 function calls.	
2	Print out the frequencies of each tie member.	Frequencies: Party 0 won the tie 19924 times. Party 1 won the tie 20040 times. Party 2 won the tie 19919 times. Party 3 won the tie 20073 times. Party 4 won the tie 20044 times.	The frequencies are fairly uniform.	The frequencies were fairly uniform and seem to be fair.	The uniformity of the results is up to the discretion of the tester. These results seemed fairly uniform and fair to me.

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_017

Name(s) of Testers: Michael Markiewicz

Test Description: Checks that the results from the OPL algorithm when a party ties results in a fairly fair coin toss.

Automated: no

This test is stored in OPLManualTest.java and uses OPL's runVotingAlgorithm function. This test is called testOPLRunVotingAlgorithmCheckPartyTie().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {3, 2, 6, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 2000, 2000, 100}	The party votes are populated.	The parties were successfully populated.	A tie is going to occur between parties 0-2.
3	Create a new OPL instance.	numBallots: 4100 numSeats: 1	New instance of OPL is created.	The instance was created without errors.	
4	Run the voting algorithm 1500 different times.	OPL instance	Algorithm runs without errors.	No errors occurred.	
5	Print the frequency in which each party won a seat (i.e., won the tie to earn the seat).	Frequencies: Party 1 won the seat 474 times.	The frequencies are fairly uniform.	The frequencies were fairly uniform and seem to be fair.	The uniformity of the results is up to the discretion of the tester.

		Party 2 won the seat 499 times. Party 3 won the seat 527 times.			These results seemed fairly uniform and fair to me.

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: OPLUnit_018

Name(s) of Testers: Michael Markiewicz

Test Description: Checks that the results from the OPL algorithm when a candidate ties results in a fairly fair coin toss.

Automated: no

This test is stored in OPLManualTest.java and uses OPL's runVotingAlgorithm function. This test is called testOPLRunVotingAlgorithmCheckCandidateTie().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create 4 parties with different number of candidates.	numParties: 4 numCandidatesPerParty: {4, 2, 3, 2}	The expected number of parties are created with the expected number of candidates.	The parties attribute is created successfully.	
2	Populate the party instances with a certain number of votes.	partyVotes: {2000, 3000, 4000, 100}	The party votes are populated.	The parties were successfully populated.	No ties should occur between parties. Party 0 is always going to get only 1 seat.
3	Populate the candidates with a certain number of votes.	candidateVotes: {{600,600,600,200}, {2000, 1000}, {500,1000,2500}, {300,200}}	The candidates were populated.	The candidates were successfully populated.	The first three candidates in party 0 are always going to tie for a seat.
4	Create a new OPL instance.	numBallots: 9100 numSeats: 5	New instance of OPL is created.	The instance was created without errors.	

5	Run the voting algorithm 1500 different times.	OPL instance	Algorithm runs without errors.	No errors occurred.	
6	Print the frequency in which each candidate won a seat (i.e., won the tie to earn the seat).	Frequencies: Candidate 1 won the seat 529 times. Candidate 2 won the seat 485 times. Candidate 3 won the seat 486 times.	The frequencies are fairly uniform.	The frequencies were fairly uniform and seem to be fair.	The uniformity of the results is up to the discretion of the tester. These results seemed fairly uniform and fair to me.

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: 03/11/2021

Test Case ID#: OPLSystem_001

Name(s) of Testers: Noah Park

Test Description: Tests that the example OPL file given to us outputs the correct results.

Automated: Yes

This test is stored in OPLSystemTest.java and uses Eligere's main() function. The test is named givenOPLTest().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, Eligere, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the givenOPL.csv filepath.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the example output, audit, and media files in a non-null string.	File paths for each text file.	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: 03/11/2021

Test Case ID#: OPLSystem_002

Name(s) of Testers: Noah Park

Test Description: Tests the 100,000 ballot OPL file outputs the correct results.

Automated: Yes

This test is stored in OPLSystemTest.java and uses Eligere's main() function. The test is named bigOPLTest().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, Eligere, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the bigOPL.csv filepath.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the example output, audit, and media files in a non-null string.	File paths for each text file.	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: 03/11/2021

Test Case ID#: OPLSystem_003

Name(s) of Testers: Noah Park

Test Description: Tests that the OPL file with a Candidate tie outputs the correct results.

Automated: Yes

This test is stored in OPLSystemTest.java and uses Eligere's main() function. The test is called candidateTieOPLTest().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, Eligere, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the candidateTieOPL.csv filepath.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the example output, audit, and media files in a non-null string.	File paths for each text file.	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: 03/11/2021

Test Case ID#: OPLSystem_004

Name(s) of Testers: Noah Park

Test Description: Tests that the OPL file with a Party tie outputs the correct results.

Automated: Yes

This test is stored in OPLSystemTest.java and uses Eligere's main() function. The test is named partyTieOPLTest().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, Eligere, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the partyTieOPL.csv filepath.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the example output, audit, and media files in a non-null string.	File paths for each text file.	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: 03/11/2021

Test Case ID#: OPLSystem_005

Name(s) of Testers: Noah Park

Test Description: Tests that a 100,000 ballot election runs in under 8 minutes.

Automated: No

This test is stored in OPLSystemTest.java and uses Eligere's main() function. The test is called bigOPLTest().

Results: Pass

Preconditions for Test: The OPL, Candidate, Party, Eligere, and OPLTestHelpers class exists. In particular, the constructor for the OPL class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the giveOPL.csv filepath.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the example output, audit, and media files in a non-null string.	File paths for each text file.	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	
5	Test runs in under 8 minutes.		Runs in under 8 minutes.	Ran in 167 ms.	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/10/2021**

Test Case ID#: IRUnit_001

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests the handleTie function that the OPL class uses from Election for illegal arguments.

Automated: yes

This test is stored in IRTest.java and uses the Election class's handleTie function

Results: Pass

Preconditions for Test: handleTie function must exist in Election

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Call handleTie on 4 illegal arguments (nonpositive numbers).	illegalArguements: {0, -1, -10, -100}	handleTie throws an IllegalArguementException for each illegal argument.	An IllegalArguementException was thrown for each illegal argument.	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_002

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests the correct IR Constructor

Automated: yes

This test is stored in IRTest.java and uses the IR class's Constructor

Results: Pass

Preconditions for Test: handleTie function must exist in Election

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an arrayList of candidates with 1 candidate in it.	Votes: Votes received by the candidate	Candidate is created without an error	Candidate is created without an error	
2	Create IR Instance	Candidates: candidate created in step 1 Ballots: hashmap containing a single ballot for the single candidate totalNumBallots: 20	IR instance is created without error	IR instance is created without error	
3	Check that totalCounts was set correctly	totalNumBallots: 20	IR.TotalCounts is equal to totalNumBallots	IR.TotalCounts is equal to totalNumBallots	
4	Check that ballots was set correctly	Ballots: ballots used as parameter to IR constructor	IR.Ballots is equal to Ballots	IR.Ballots is equal to Ballots	
5	Check that Candidates was set correctly	Candidates: arraylist used as a parameter to constructor	IR.candidates is equal to candidates	IR.candidates is equal to candidates	

Post condition(s) for Test: IR instance was created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_003

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests the IR constructor throws an exception when given null parameters.

Automated: yes

This test is stored in IRTest.java and uses the IR class's constructor. This test is called testNullParametersIRConstructor

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance with the ballots parameter set to null	totalNumBallots: 5520 candidates: arrayList of 5 candidates ballots:null	IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	An IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	
2	Create an IR instance with the candidates parameter set to null	totalNumBallots: 5520 candidates: null Ballots: hashmap containing ballots	IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	

Post condition(s) for Test: IllegalArgumentException is thrown which means no instance of IR is created. Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_004

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests the IR constructor throws an exception when given empty inputs.

Automated: yes

This test is stored in IRTest.java and uses the IR class's constructor. This test is called testEmptyParametersIRConstructor

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance with the empty ballots parameter	totalNumBallots: 5520 candidates: arrayList of 5 candidates ballots: empty hashmap	IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	An IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	
2	Create an IR instance with the candidates parameter set to null	totalNumBallots: 5520 candidates: empty arrayList Ballots: hashmap containing 1 ballot	IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	

Post condition(s) for Test: IllegalArgumentException is thrown which means no instance of IR is created. Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/10/2021**

Test Case ID#: IRUnit_005

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests the IR constructor throws an exception when given a nonpositive input

Automated: yes

This test is stored in IRTest.java and uses the IR class's constructor. This test is called testNonPositiveInputIRConstructor

Results: Pass

Preconditions for Test: IR, Candidate, and IRTestHelpers class exist. IR constructor functions correctly.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance with a nonpositive totalNumBallots	totalNumBallots: 0 candidates: arrayList of 5 candidates ballots: non-empty hashmap	IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	An IllegalArguementException is thrown with the message totalNumBallots must be positive, candidates and ballots must be non-null and non-empty	

Post condition(s) for Test: IllegalArgumentException is thrown which means no instance of IR is created. Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_006

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Determines whether the audit and media files are created when the IR algorithm runs

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called testIRRunVotingAlgorithmCheckFilesCreated().

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 100 candidates: arrayList of 4 candidates ballots: HashMap of ballots for each candidate	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Check if the audit file and media file were created with correct file names	None	T he audit file has the expected name of "IRAuditFile_<year>_<month>_<day>_<time>.csv" and the media file has the expected name of "IRMediaReport_<year>_<month>_<day>_<time>.csv"	T he audit file has the expected name of "IRAuditFile_<year>_<month>_<day>_<time>.csv" and the media file has the expected name of "IRMediaReport_<year>_<month>_<day>_<time>.csv"	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_007

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests that the algorithm is cut short if a majority is present

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called CheckMajorityPresent().

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 130 candidates: arrayList of 4 candidates with 10,20,30,and 70 votes respectively ballots: HashMap of ballots for each candidate	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Check that the ballots hashmap was not modified	BallotsCopy: Copy of the original Ballots	Ballots attribute of the IR instance is equal to the copy of the original ballots	Ballots attribute of the IR instance is equal to the copy of the original ballots	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_008

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests that the algorithm is not cut short if a majority is not present

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called CheckMajorityNotPresent().

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 120 candidates: arrayList of 4 candidates with 10,20,30,and 60 votes respectively ballots: HashMap of ballots for each candidate	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Check that the ballots hashmap was modified	BallotsCopy: Copy of the original Ballots	Ballots attribute of the IR instance is not equal to the copy of the original ballots	Ballots attribute of the IR instance is not equal to the copy of the original ballots	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_009

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests that the algorithm is not cut short if only two candidates are present.

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called CheckMajorityWhenOnlyTwoCandidates

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 20 candidates: arrayList of 2 candidates with 10 votes each ballots: HashMap of ballots for each candidate	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Check that the ballots hashmap was not modified	BallotsCopy: Copy of the original Ballots	Ballots attribute of the IR instance is equal to the copy of the original ballots	Ballots attribute of the IR instance is equal to the copy of the original ballots	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_010

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests that the ballots hashmap changes correctly when there is a candidate who wins without a tie but after the first round

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called CheckBallotsChangedToExpected().

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 120 candidates: arrayList of 4 candidates. ballots: { <"(p0)", 10>, <"(p1)(p2)(p0)", 10>, <"(p3)(p2)", 20>, <"(p2)(p1)(p0)", 10>, <"(p1)(p2)", 5>, <"(p2)(p1)(p3)", 20> }	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Check that the ballots hashmap contains expected modifications	ExpectedBallots: {< "(p2)", 25>, <"(p3)(p2)", 20>, <(p2)(p3)", 20>	Ballots attribute of the IR instance is expected ballots at the end of the algorithm	Ballots attribute of the IR instance is expected ballots at the end of the algorithm	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_011

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Checks that the expected winner of an election with no ties is declared.

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called CheckWinnerOfRegularElection().

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 120 candidates: arrayList of 4 candidates. ballots: { <"(p0)", 10>, <"(p1)(p2)(p0)", 10>, <"(p3)(p2)", 20>, <"(p2)(p1)(p0)", 10>, <"(p1)(p2)", 5>, <"(p2)(p1)(p3)", 20> }	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Check that the expected winner matches the real winner	Expected winner: candidate in p1	Winner of the election is the same as the expected winner	Winner of the election is the same as the expected winner	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_012

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Checks that a candidate wins if they are the only ones in the election.

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called CheckWinnerOfElectionWithOneCandidate().

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 10 candidates: arrayList of 1 candidate. ballots: { <"(p0)", 10>, }	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Check that the expected winner matches the real winner	Expected winner: candidate in p0	Winner of the election is the same as the expected winner	Winner of the election is the same as the expected winner	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/10/2021**

Test Case ID#: IRUnit_013

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Ensures that the output to the console, audit file, and media report are expected outputs. This works by running the algorithm, and comparing the outputs to example output files. (ExampleIROutput.txt, ExampleIRAudit.txt, ExampleIRMedia.txt).

Automated: **yes**

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called TestIROutput().

Results: **Pass**

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 120 candidates: arrayList of 4 candidates. ballots: { <"(p0)", 10>, <"(p1)(p2)(p0)", 10>, <"(p3)(p2)", 20)>, <"(p2)(p1)(p0)", 10>, <"(p1)(p2)", 5>, <"(p2)(p1)(p3)", 20> }	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Checks for the correct output on the screen	expectedOutput: Load expected output in screen to this variable	Expected output is equal to the actual output by the IR algorithm	Expected output is equal to the actual output by the IR algorithm	
4	Checks for the correct output in the audit file	expectedAudit: : Load expected text in Audit file to this variable	ExpectedAudit is equal to the actual audit file by the IR algorithm	Expected Audit is equal to the actual Audit file by the IR algorithm	

5	Checks for the correct output in the media file	expectedMedia: : Load expected text in Media file to this variable	ExpectedMedia is equal to the actual media file by the IR algorithm	ExpectedMedia is equal to the actual media file by the IR algorithm	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_014

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests to see if the output from running the IR algorithm with a candidate tie when being eliminated matches the expected output from a tie.

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called testIRTiedCandidatesExpectedOutput()

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 153 candidates: arrayList of 4 candidates. ballots: {<"(p0)(p1)(p3)", 15>, <("(p0)", 3)> ,<"(p0)(p2)", 5)>, <("(p0)(p2)(p1)(p3)", 5)>, <"(p0)(p3)(p1)",5>, <("(p1)(p2)(p0)", 20>, <("(p1)(p2)", 15)> <("(p2)(p1)(p0)", 20)>, <("(p2)(p1)(p3)", 20)>, <("(p3)(p2)", 25)>, <("(p3)", 20)>	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Checks for the correct output on	expectedOutputs: Load all expected files	Output is equal to one of the expected files	Output is equal to one of the expected files	

	the screen				
4	Checks for the correct output in the audit file	expectedAudits : Load all expected Audits	Audit File is equal to one of the expected files	Audit File is equal to one of the expected files	
5	Checks for the correct output in the media file	expectedMedias: : Load all expected Medias	Media file is equal to one of the expected files	Media File is equal to one of the expected files	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/10/2021**

Test Case ID#: IRUnit_015

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests to see if the output from running the IR algorithm with a candidate tie when being eliminated matches the expected output from a tie.

Automated: yes

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called testIRTiedCandidatesForWinnerExpectedOutput().

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create a new IR instance	totalNumBallots: 153 candidates: arrayList of 4 candidates. ballots: {<"(p0)(p1)(p3)", 15>, <("(p0)", 3)> ,<"(p0)(p2)", 5)>, <("(p0)(p2)(p1)(p3)", 5)>, <"(p0)(p3)(p1)",5>, <("(p1)(p2)(p0)", 20>, <("(p1)(p2)", 15)> <("(p2)(p1)(p0)", 20)>, <("(p2)(p1)(p3)", 20)>, <("(p3)(p2)", 25)>, <("(p3)", 20)>	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm	IR instance	Algorithm runs without errors	No errors occurred	
3	Checks for the correct output on	expectedOutputs: Load all expected files	Output is equal to one of the expected files	Output is equal to one of the expected files	

	the screen				
4	Checks for the correct output in the audit file	expectedAudits : Load all expected Audits	Audit File is equal to one of the expected files	Audit File is equal to one of the expected files	
5	Checks for the correct output in the media file	expectedMedias: : Load all expected Medias	Media file is equal to one of the expected files	Media File is equal to one of the expected files	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/10/2021

Test Case ID#: IRUnit_016

Name(s) of Testers: Michael Markiewicz, Mohammad Essawy

Test Description: Tests the handleTie function that is used by IR when inputted valid arguments. Creates situation where tiw will exist in every round of Ir in order to make sure the function works when multiple people tie for last place and when two people tie for first place.

Automated: No

This test is stored in IRTest.java and uses the IR class's runVotingAlgorithm function. This test is called testHandleTie().

Results: Pass

Preconditions for Test: IR and Candidate class exist

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create the same IR instance 1000 times	totalNumBallots: 40 candidates: arrayList of 4 candidate. ballots: { <"(p0)", 10>, <"(p1)", 10>, <"(p2)", 10>, <"(p3)", 10>, }	New instance of IR is created	New instance of IR is created	
2	Run the voting Algorithm for each instance	IR instances	Algorithm runs without errors	No errors occurred	
3	Record and print the number of times the election was one for each candidate	None	Each candidate wins the election roughly yhe same number of ties.	Each candidate wins the election roughly yhe same number of ties.	

Post condition(s) for Test: Nothing in the system has changed. An instance of the IR class has been created.

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_001

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns null and prints the correct error message if the file on the filePath does not exist.

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testFileNotFound().

Results: Pass

Preconditions for Test: System.out is captured by the test

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with a fault file path a verify the Election is null	CreateElectionFileName: "/not/a/real/file/path"	The election object passed back by createElection is NULL	The election object passed back by createElection is NULL	
2	Verify the error message printed to the terminal is correct	actualOutput: systemOut testOutput:String	String "Error: File Not Found"	String "Error: File Not Found" is captured from System.out	

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_002

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns null and prints the correct error message if the file given to createElection() has an invalid election type

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testInvalidElectionType().

Results: Pass

Preconditions for Test: System.out is captured by the test. The file invalidElectionType.csv exists and its path is set in the string invalidElectionTypeFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file invalidElectionType.csv path and verify the Election is null	InvalidElectionTypeFilePath: "./testing/testFiles/invalidElectionType.csv "	The election object passed back by createElection is NULL	The election object passed back by createElection is NULL	
2	Verify the error message printed to the terminal is correct	actualOutput: systemOut testOutput:String	String "Error: Invalid Election Type"	String "Error: Invalid Election Type" is captured from System.out	

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_003

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns an IR object when given a ballot file of election type IR

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testCreateIRElectionObject().

Results: Pass

Preconditions for Test: The file givenIR.csv exists and its path is set in the string givenIRFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file givenIR.csv and verify the Election is of type IR	givenIRFilePath: "./testing/testFiles/givenIR.csv "	The election object returned by createElection is an instance of IR	The election object returned by createElection is an instance of IR	The givenIR.csv file is the example IR file given to us in the write up

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_004

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns an OPL object when given a ballot file of election type OPL

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testCreateOPLElectionObject().

Results: Pass

Preconditions for Test: The file givenOPL.csv exists and its path is set in the string givenOPLFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file givenOPL.csv and verify the Election is of type OPL	givenOPLFilePath: "./testing/testFiles/givenOPL.csv "	The election object returned by createElection is an instance of OPL	The election object returned by createElection is an instance of OPL	The givenOPL.csv file is the example OPL file given to us in the write up

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_005

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns IR object with the correct parameters based on the givenIR.csv

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testParametersIRGivenFile().

Results: Pass

Preconditions for Test: The file givenIR.csv exists and its path is set in the string givenIRFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file givenIR.csv and verify the Election is of type IR	givenIRFilePath: "./testing/testFiles/givenIR.csv "	The election object returned by createElection is an instance of IR This object is called testIR	The election object returned by createElection is an instance of IR	The givenIR.csv file is the example IR file given to us in the write up
2	Iterate through the fields in IR to verify the IR object created by createElection is correct	SpyIR that is an instance of testIR Array that contains the fields of data of test IR	No errors in this test step.	No errors in test step	
3	Verify that the “ballots” field in testIR is correct	testBallotCounter is the expected hashmap in IR generated by the test	testBallotCounter is equal to the “ballots” field in testIR	testBallotCounter is equal to the “ballots” field in testIR	

4	Verify that the “candidates” field in testIR is correct	realCandidates is the expected arraylist of candidates in IR generated by the test	realCandidates is equal to the “candidates” field in testIR	realCandidates is equal to the “candidates” field in testIR	CanidateArraylistIsEqual() is used to verify equality
5	Verify the “totalCount” field in testIR is correct	The expected total votes in the IR object is 6	“totalCount” field in testIR is equal to 6	“totalCount” field in testIR is equal to 6	

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_006

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns OPL object with the correct parameters based on the givenOPL.csv

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testParametersOPLGivenFile()

Automated: Yes

Results: Pass

Preconditions for Test: The file givenOPL.csv exists and its path is set in the string givenOPLFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file givenOPL.csv and verify the Election is of type OPL	givenOPLFilePath: "./testing/testFiles/givenOPL.csv "	The election object returned by createElection is an instance of OPL	The election object returned by createElection is an instance of OPL	The givenOPL.csv file is the example OPL file given to us in the write up
2	Iterate through the fields in OPL to verify the OPL object created by createElection is correct	spyOPLthat is an instance of testOPL Array that contains the fields of data of testOPL	No errors in this test step.	No errors in test step	

3	Verify the "numSeatsAvaible" field in testOPL is correct	The expected number of seats available is 3	The "numSeatsAvaible" field in testOPL is equal to 3	The "numSeatsAvaible" field in testOPL is equal to 3	
4	Verify the "numSeatsLeft" field in testOPL is correct	The expected number of seats left is 3	The "numSeatsLeft" field in testOPL is equal to 3	The "numSeatsLeft" field in testOPL is equal to 3	
5	Verify the "parties" field in testOPL is correct	A party arraylist called testParty is created with the expected parmeters for each party created and inserted	The "testParty" arraylist is equal to the "parties": field in testOPL	The "testParty" arraylist is equal to the "parties": field in testOPL	The function partyArraylistIsEqual() is used for equlity
6	Verify the "quota" field in testOPL is correct	The expected number of seats available is 3	The "quota" field in testOPL is equal to 3	The "quota" field in testOPL is equal to 3	

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_007

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns IR object with the correct parameters based on the bigIR.csv. It is different than CreateElection_005 in that this CSV contains 100,000 ballots.

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testReadBigIRFile().

Results: Pass

Preconditions for Test: The file bigIR.csv exists and its path is set in the string bigIRFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file bigIR.csv in under 30 seconds	bigIRFilePath: "./testing/testFiles/bigIR.csv "	CreateElection() runs in under 30 seconds	CreateElection() runs in under 30 seconds	Trying to keep the processing time down. This test is only truly useful on a CSE lab machine, but is still interesting else were.
2	verify the Election is of type IR	testIR	testIR is an instance of IR	testIR is an instance of IR	
3	Iterate through the fields in IR to verify the IR object created by createElection is correct	SpyIR that is an instance of testIR Array that contains the fields of data of test IR	No errors in this test step.	No errors in test step	

4	Verify that the “ballots” field in testIR is correct	testBallotCounter is the expected hashmap in IR generated by the test	testBallotCounter is equal to the “ballots” field in testIR	testBallotCounter is equal to the “ballots” field in testIR	
5	Verify that the “candidates” field in testIR is correct	realCandidates is the expected arraylist of candidates in IR generated by the test	realCandidates is equal to the “candidates” field in testIR	realCandidates is equal to the “candidates” field in testIR	CanidateArraylistIsEqual() is used to verify equality
6	Verify the “totalCount” field in testIR is correct	The expected total votes in the IR object is 100,000	“totalCount” field in testIR is equal to 100,000	“totalCount” field in testIR is equal to 100,000	

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_008

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns OPL object with the correct parameters based on the bigOPL.csv. It is different than CreateElection_006 in that this CSV contains 100,000 ballots.

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testReadBigOPL()

Results: Pass

Preconditions for Test: The file bigOPL.csv exists and its path is set in the string bigOPLFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file bigOPL.csv in under 30 seconds	bigOPLFilePath: "./testing/testFiles/bigOPL.csv "	CreateElection() runs in under 30 seconds	CreateElection() runs in under 30 seconds	Trying to keep the processing time down. This test is only truly useful on a CSE lab machine, but is still interesting else were.
2	verify the Election is of type OPL	testOPL	testOPL is an instance of OPL	testOPL is an instance of OPL	
3	Iterate through the fields in OPL to verify the OPL	spyOPLthat is an instance of testOPL	No errors in this test step.	No errors in test step	

	object created by createElection is correct	Array that contains the fields of data of testOPL			
4	Verify the “numSeatsAvaible” field in testOPL is correct	The expected number of seats available is 3	The “numSeatsAvaible” field in testOPL is equal to 3	The “numSeatsAvaible” field in testOPL is equal to 3	
5	Verify the “numSeatsLeft” field in testOPL is correct	The expected number of seats left is 3	The “numSeatsLeft” field in testOPL is equal to 3	The “numSeatsLeft” field in testOPL is equal to 3	
6	Verify the “parties” field in testOPL is correct	A party arraylist called testParty is created with the expected parmeters for each party created and inserted	The “testParty” arraylist is equal to the “parties”: field in testOPL	The “testParty” arraylist is equal to the “parties”: field in testOPL	The function partyArraylistIsEqual() is used for equlity
7	Verify the “quota” field in testOPL is correct	The expected number of seats available is 33333	The “quota” field in testOPL is equal to 33333	The “quota” field in testOPL is equal to 33333	

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: **03/13/2021**

Test Case ID#: CreateElection_009

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns IR object with the correct parameters based on the OneCandidate.csv. This test tests the edge case of having a single candidate

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testParametersIROneCandidate().

Results: Pass

Preconditions for Test: The file oneCandidateIR.csv exists and its path is set in the string oneCandidateIRFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file givenIR.csv and verify the Election is of type IR	oneCandidateIRFilePath: "./testing/testFiles/oneCandidateIR.csv "	The election object returned by createElection is an instance of IR This object is called testIR	The election object returned by createElection is an instance of IR	
2	Iterate through the fields in IR to verify the IR object created by createElection is correct	SpyIR that is an instance of testIR Array that contains the fields of data of test IR	No errors in this test step.	No errors in test step	
3	Verify that the “ballots” field in testIR is correct	testBallotCounter is the expected hashmap in IR generated by the test	testBallotCounter is equal to the “ballots” field in testIR	testBallotCounter is equal to the “ballots” field in testIR	

Project Name: Project 1: Voting System **Team#6**

4	Verify that the “candidates” field in testIR is correct	realCandidates is the expected arraylist of candidates in IR generated by the test	realCandidates is equal to the “candidates” field in testIR	realCandidates is equal to the “candidates” field in testIR	CanidateArraylistIsEqual() is used to verify equality
5	Verify the “totalCount” field in testIR is correct	The expected total votes in the IR object is 1	“totalCount” field in testIR is equal to 1	“totalCount” field in testIR is equal to 1	

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: Unit

Test Date: 03/13/2021

Test Case ID#: CreateElection_010

Name(s) of Testers: John Foley

Test Description:

Test that createElection() returns OPL object with the correct parameters based on the oneCandidateOPL.csv. This test tests the edge case of having a single candidate

Automated: Yes

This test is stored in CreateElectionTest.java and uses createElection(). The test is called testParametersOPLOneCandidate()

Results: Pass

Preconditions for Test: The file oneCandidateOPL.csv exists and its path is set in the string oneCandidateOPLFilePath.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Create an Election with the file oneCandidateOPLFilePath.csv and verify the Election is of type OPL	oneCandidateOPLFilePath: "./testing/testFiles/oneCandidateOPL.csv "	The election object returned by createElection is an instance of OPL	The election object returned by createElection is an instance of OPL	
2	Iterate through the fields in OPL to verify the OPL object created by createElection is correct	spyOPLthat is an instance of testOPL Array that contains the fields of data of testOPL	No errors in this test step.	No errors in test step	

3	Verify the “numSeatsAvaible” field in testOPL is correct	The expected number of seats available is 1	The “numSeatsAvaible” field in testOPL is equal to 1	The “numSeatsAvaible” field in testOPL is equal to 1	
4	Verify the “numSeatsLeft” field in testOPL is correct	The expected number of seats left is 1	The “numSeatsLeft” field in testOPL is equal to 1	The “numSeatsLeft” field in testOPL is equal to 1	
5	Verify the “parties” field in testOPL is correct	A party arraylist called testParty is created with the expected parmeters for each party created and inserted	The “testParty” arraylist is equal to the “parties”: field in testOPL	The “testParty” arraylist is equal to the “parties”: field in testOPL	The function partyArraylistIsEqual() is used for equlity
6	Verify the “quota” field in testOPL is correct	The expected number of seats available is 1	The “quota” field in testOPL is equal to 1	The “quota” field in testOPL is equal to 1	

Post condition(s) for Test: System out is no longer captured

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: 03/13/2021

Test Case ID#: IRSystem_001

Name(s) of Testers: Justin Lam

Test Description: Tests that the example IR CSV file given to us outputs the correct results.

Automated: Yes

This test is stored in IRSystemTest.java and uses Eligere's main() function. The test is named givenIRTest().

Results: Pass

Preconditions for Test: The IR, Candidate, Party, Eligere, and IRTestHelpers class exists. In particular, the constructor for the IR class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the givenIR.csv file path.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the expected output, audit, and media files in a non-null string.	File paths for each text file: ExpectedGivenIROutput.txt ExpectedGivenIRAudit.txt ExpectedGivenIRMedia.txt	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: **03/13/2021**

Test Case ID#: IRSystem_002

Name(s) of Testers: Justin Lam

Test Description: Tests that the large example IR given to us outputs the correct results.

Automated: Yes

This test is stored in IRSystemTest.java and uses Eligere's main() function. The test is named givenBigIRTest().

Results: Pass

Preconditions for Test: The IR, Candidate, Party, Eligere, and IRTestHelpers class exists. In particular, the constructor for the IR class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the givenBigIR.csv file path.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the expected output, audit, and media files in a non-null string.	File paths for each text file: ExpectedBigIROutput.txt ExpectedBigIRAudit.txt ExpectedBigIRMedia.txt	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: 03/13/2021

Test Case ID#: IRSystem_003

Name(s) of Testers: Justin Lam

Test Description: Tests that the system outputs the correct results when there is only one candidate.

Automated: Yes

This test is stored in IRSystemTest.java and uses Eligere's main() function. The test is named oneCandidateTest().

Results: Pass

Preconditions for Test: The IR, Candidate, Party, Eligere, and IRTestHelpers class exists. In particular, the constructor for the IR class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the oneCandidateIR.csv file path.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the expected output, audit, and media files in a non-null string.	File paths for each text file: ExpectedOneCandidateIROutput.txt ExpectedOneCandidateIRAudit ExpectedOneCandidateIRMedia.txt	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: 03/13/2021

Test Case ID#: IRSystem_004

Name(s) of Testers: Justin Lam

Test Description: Tests that the system outputs the correct results when there is a CSV file with many candidates and 100000 ballots.

Automated: Yes

This test is stored in IRSystemTest.java and uses Eligere's main() function. The test is named bigIRTest().

Results: Pass

Preconditions for Test: The IR, Candidate, Party, Eligere, and IRTestHelpers class exists. In particular, the constructor for the IR class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Begin a timer.	start: indicating the current time.			
2	Run Eligere's main with the bigRandomIR.csv file path.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
3	End the timer and check if the duration is less than 8 minutes.	endtime: indicating the end time. duration: calculating the total time spent.	The total duration is less than 8 minutes.	The total duration is less than 8 minutes.	
3	Store the expected output, audit, and media files in a non-null string.	File paths for each text file. ExpectedBigIROutput.txt ExpectedBigIRAudit.txt ExpectedBigIRMedia.txt	Each string is stored correctly.	Each string populated correctly.	
4	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
5	Ensures the expected and actual outputs, audit files,	Each expected and actual string.	True	True	

	and media files are all identical.				

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: **03/13/2021**

Test Case ID#: IRSystem_005

Name(s) of Testers: Justin Lam

Test Description: Tests that the system outputs the correct results when there is a tie for the most number of votes.

Automated: Yes

This test is stored in IRSystemTest.java and uses Eligere's main() function. The test is named winnerTieTest().

Results: Pass

Preconditions for Test: The IR, Candidate, Party, Eligere, and IRTestHelpers class exists. In particular, the constructor for the IR class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the winnerTieIR.csv file path.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the expected output, audit, and media files in a non-null string.	File paths for each text file. ExpectedTieIROutput_1.txt ExpectedTieIRAudit_1.txt ExpectedTieIRMedia_1.txt ExpectedTieIROutput_2.txt ExpectedTieIRAudit_2.txt ExpectedTieIRMedia_2.txt	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.

Project Name: Project 1: Voting System **Team#6**

Test Stage: System

Test Date: **03/13/2021**

Test Case ID#: IRSystem_006

Name(s) of Testers: Justin Lam

Test Description: Tests that the system outputs the correct results when there is a tie for the least number of votes and a candidate must be eliminated.

Automated: Yes

This test is stored in IRSystemTest.java and uses Eligere's main() function. The test is named loserTieTest().

Results: Pass

Preconditions for Test: The IR, Candidate, Party, Eligere, and IRTestHelpers class exists. In particular, the constructor for the IR class works as expected.

Step #	Test Step Description	Test Data	Expected Result	Actual Result	Notes
1	Run Eligere's main with the winnerTieIR.csv file path.	Args: String array with the file path for the string.	The main function runs to completion and outputs the necessary files.	The main function runs to completion and outputs the necessary files.	
2	Store the expected output, audit, and media files in a non-null string.	File paths for each text file. ExpectedLoserTieIROutput_1.txt ExpectedLoserTieIRAudit_1.txt ExpectedLoserTieIRMedia_1.txt ExpectedLoserTieIROutput_2.txt ExpectedLoserTieIRAudit_2.txt ExpectedLoserTieIRMedia_2.txt	Each string is stored correctly.	Each string populated correctly.	
3	Store the actual output, audit, and media files in a non-null string.		Each string is stored correctly.	Each string is stored correctly.	
4	Ensures the expected and actual outputs, audit files, and media files are all identical.	Each expected and actual string.	True	True	

Post condition(s) for Test: Nothing in the system has changed.