Testing

Assessment 3

Introduction

This document describes the complete testing process carried out by our team during the course of assessment 3. The first part of this document discusses the objectives and roles involved behind the whole process. It also gives a brief description of the research, plan and the schedule our team abided to during the entire testing process. The second part of the document consists of an overview and justification of every methodology we implemented in assessment 3, while extending the product of team FLR. It also consists of the testing evidence for each methodology. The last part of the document consists of an independent risk table for testing and a brief conclusion.

1. Testing Strategy and Plan

1.1 Objectives

Testing is one of the most crucial phases of any software project. It defines the quality of the software developed. Our main objective in assessment 3 was to test the extensions added to the product we adopted, in order to make it an ideal product for our client. We believe that the definition of an ideal product for a client would be a product with no bugs and errors, a product that meets all his requirements and a product that is reliable and fast.

To achieve an ideal product for our client, we had to achieve 4 sub-objectives from testing:

- Finding and fixing all the bugs in the code adopted by our team; before we add new functionality.
- Finding and fixing all the bugs in the code after our team has added the new functionality.
- Making sure that the software consists of all the additional functionality requested by the client.
- Making sure that the software meets the functional, performance and reliability requirements.

A secondary objective of our team in this assessment was to also fix or change the existing testing of the team's initial product because we felt there was scope for improvement.

1.2 Plan

During the initial planning of Assessment 3, we had a team consensus of continuing with the same testing team as assessment 2. The team consisted of Jaidev, Joseph and Leslie. The basis for this decision was that the previous knowledge of these testing team members would help in developing a better testing report for the product in the short span of time of assessment 3. Jaidev was assigned as the team leader for assessment 3. As a leader his responsibilities included:

- Making the test plan for assessment 3.

- Making a test schedule for assessment 3.
- Assigning, conducting and keeping track of the research done.
- Communicating regularly with the main programmers during parallel unit testing.
- Managing and allocating the responsibilities given to the other two members.
- Keeping track of the progress made every day.

The team leader, Jaidev decided to abide to the protocol of equal participation of every member towards each testing methodology that was formed in assessment 3. The reason behind this decision was that the combination of the knowledge gained in assessment 3 and the ideas of every team member would immensely benefit the quality of testing. A brief overview of the division of tasks between members is given below:

Task	Team Member
Unit, Integration and Regression Testing	Jaidev, Joseph and Leslie
Functional Testing	Jaidev, Joseph and Leslie
Acceptance Testing	Jaidev, Joseph and Leslie
Deployment Testing	Jaidev, Joseph and Leslie
Alpha Testing	Jaidev, Joseph and Leslie
Beta Testing	Jaidev, Joseph and Leslie
Testing Strategy and Plan	Jaidev and Joseph
Risk Table and conclusion	Jaidev
Methodology overviews and Justifications	Jaidev, Leslie and Joseph
Review	1st Phase - Jaidev, Leslie and Joseph 2nd Phase – Philip, Paulius and Miguel

The research phase of our testing process lasted for 2 days and was relatively shorter than assessment 2. The idea of independent research by every member worked extremely well in assessment 2 and helped us develop a strong testing plan and hence the same strategy was followed. The aim this time was to gather maximum knowledge about regression testing and give a glimpse to a few other testing methodologies that would be helpful for this assessment. The strategy was to do this individually and, then share and combine our outputs with the outputs of others. The key resource that we used for our research was the 9th Edition of the book 'Software Engineering' by Ian Sommerville[1].

Our rough plan this time was to complete the testing research over the weekend after the submission and then develop a solid testing plan on the basis of the research and the lessons learnt from the testing phase of the previous assessment. The Monday following the weekend was the day when we developed our test plan. The approach of our testing plan to fulfil the objectives was different than last time. Firstly, we decided to carry out unit testing in parallel to the programming phase. After the previous assessment, we all strongly believed that this would make the bug detection process more precise and would also make the fixing process easier and better. Secondly, the key thing to keep in mind this time was that we were extending a testing phase for a product and were not developing a completely new test report. Hence, it was very important for us to be very careful about making changes and extending the testing phase originally developed by Team FLR. During this extension we introduced some new methodologies or extended some old ones. The table below gives a brief description of how we have modified or introduced each and every testing methodologies in line with the testing done by team FLR:

Testing Method	**Notes about Changes**
Unit Testing	Extended to test the new functionality.
Integration Testing	Extended the original testing to test new functionality.
Regression Testing	New methodology implemented to test that the addition of new functionality hasn't caused any damage to anything.
Acceptance Testing	Extension of their report. The formatting was changed.
Functional Testing	New methodology only implemented to test the functionality added by our team.

[1] I. Sommerville, Software Engineering. Boston: Pearson, 2010

Alpha Testing	New Methodology.
Beta testing	New methodology.
Deployment Testing	Added to test that the performance and reliability of the system is high

The key objectives that each methodology is fulfilling is given in the table below:

Testing Method	**Objective**
Unit Testing	Finding and Fixing bugs as a whole
Integration Testing	Finding and Fixing bugs as a whole
Regression Testing	Finding and Fixing bugs as a whole
Functional Testing	Requirements of the System are met
Acceptance Testing	Requirements of the client are met
Alpha Testing	Finding and fixing unusual bugs
Beta Testing	Finding and fixing unusual bugs
Deployment Testing	Performance and reliability of the system is high

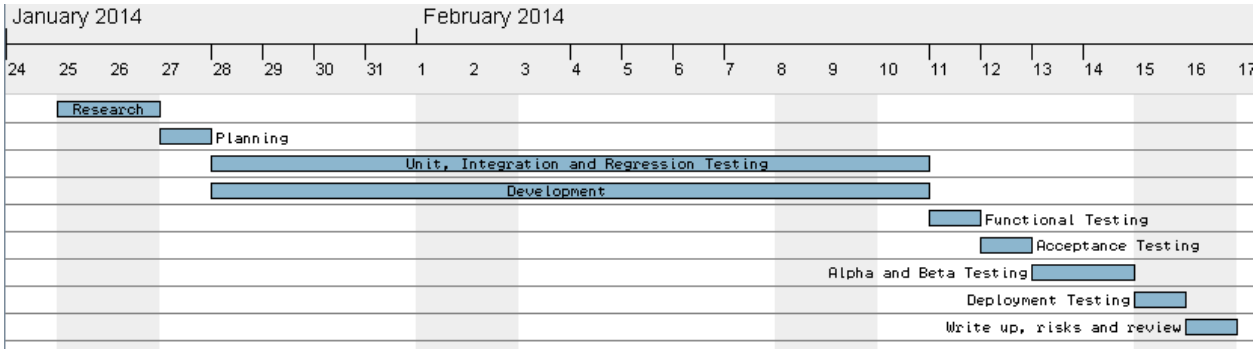
The overviews and justification of every methodology are present in the second part of the document.

1.3 Schedule

Unlike assessment 2 the testing process of this assessment took place throughout the course of the development. Spring week 4/5 were spent in Unit testing the additional functionalities in parallel with the development. The reason for doing this is justified above.

Spring week 6 was spent in carrying out testing for all the other methodologies that we have included. Spring week 7 was spent in reviewing the document.

The Gantt chart for our entire testing process is given below:



For a better understanding the Table below consists of the days spent on every task within the team:

Task	**Days Spent**
Research	2 days
Planning	1 day
Unit, Integration and Regression testing	14 days
Functional Testing	1 day
Acceptance Testing	1 day
Alpha and Beta Testing	2 days
Deployment testing	1 day
Small write-ups, risks and review	2 days

2.1 Unit, Integration and Regression Testing

Our changes to the code made during this assessment had left many of the existing unit tests broken. We, therefore, decided to create new unit tests and copy over any old tests that we felt were still appropriate. We also added new tests for full coverage of new and existing features that we felt hadn't been covered sufficiently in the previous teams unit and integration testing documents.

We tested only public methods using unit testing techniques due to the restrictions of JUnit testing which was our chosen methodology to perform these tests. All other methods will be indirectly covered during the integration testing phase.

For this assessment, we decided that the previous team's approach to integration testing was not the most efficient so, for our integration testing phase, we used JUnit tests as we felt this was the most systematic approach and would provide us with the most confidence in extensive coverage.

We followed a big bang approach which means all modules are coupled together for integration testing. We chose this for efficiency.

Our regression testing was effectively carried out within both of these phases as tests required modification to be made appropriate for changes in the code.

2.1.1 HoldingWaypoint.java (ID - JUnit1)

HoldingWaypoint is a new class implemented during this assessment. Therefore, we had to create all new JUnit tests.

Test Name	**Method Under Test**	**Description**	**Pass/Fail**
testHoldingWaypoint	HoldingWaypoint(x, y)	Tests if the holding waypoint is Initialized correctly.	Pass
testSetNextWaypoint	setNextWaypoint(Waypoint nextWaypoint)	Testing if the next waypoint is appropriately set	Pass
testGetNextWaypoint	getNextWaypoint()	Testing if the value of the next waypoint is correct	Pass

Tests not conducted:

There was 2 public methods in the HoldingWaypoint class that we believed would be better tested under functional testing. The methods is listed below:

- testDraw()
- testDrawDoubleDouble()

2.1.2 Airport.java (ID: JUnit2)

Airport is a new class implemented during this assessment. Therefore, we had to create all new JUnit tests.

Test Name	**Method Under Test**	**Description**	**Pass/Fail**
testAirport()	Airport()	Tests if the coordinates of the airport are initialized correctly.	Pass
testTakeoff()	takeoff()	Tests if the takeoff is happening properly. Sends error messages if	Pass

		something is wrong	
testInsertAircraft	InsertAircraft()	Tests if the insertion of the aircraft in the airspace is appropriate.	Pass

Tests not conducted:

There was 1 public method in the Airport class that we believed would be better tested under functional testing. The methods are listed below:

- testDrawAirportInfo

2.1.3 Waypoint (ID: JUnit3)

Waypoint was an existing class. All methods that were unaltered from the previous testing report are in **Bold**.

Test Name	**Method Under Test**	**Description**	**Pass/Fail**
testWaypointDoubleDoubleWaypointTypeString()	Waypoint(double x, double y, WaypointType type, String name)	Tests if a new waypoint is initialized problem with type given.	Pass
testWaypointDoubleDouble()	Waypoint(double x, double y)	Tests if a new waypoint is initialized problem without type given.	Pass
testPosition()	position()	Tests if position is correct.	Pass
testGetType()	getType()	Tests if type is correct.	Pass
testGetName()	getName()	Tests getter gets correct	Pass

		name.	
testGetCost()	*getCost()*	*Tests to check correct cost is returned.*	*Pass*
testGetCostBetween()	*getCostBetween()*	*Tests to check correct cost is returned.*	*Pass*
testIsMouseOver()	*isMouseOver()*	*Tests to see if the mouse is over a position*	*Pass*

Tests not conducted:

There were 2 public methods in the Waypoint class that we believed would be better tested under functional testing. The methods are listed below:

- testDrawDoubleDouble()
- testDraw()

2.1.4 Aircraft (ID: JUnit4)

Aircraft was an existing class. All methods that were unaltered from the previous assessment are in **Bold**.

Test Name	**Method Under Test**	**Description**	**Pass/Fail**
testPosition()	position()	Test that this returns a vector with the same coordinates as the location of the aircraft.	Pass
testName()	name()	Test that this returns a string which matches the aircraft's name.	Pass
testOriginName()	originName()	Test that this returns a string which matches the spawning waypoint's name.	Pass
testDestinationName()	destinationName()	Test that this returns a string which matches	Pass

		the aircraft's exit waypoint's name.	
testIsFinished()	isFinished()	Test that this returns false as the aircraft is in the airspace. Forcing a collision with the aircraft should make this method return true.	Pass
testAtAirport()	atAirport()	Tests that this returns false whenever the aircraft is not going to the airport, or is going to the airport but has not yet arrived.	Pass
testIsManuallyControlled()	isManuallyControlled()	Tests that this returns false when the aircraft is flying autonomously, and true when the user assumes direct control.	Pass
testIsLanding()	isLanding()	Test that this returns correct boolean value of the isLanding variable.	Pass
testOutOfBounds()	outOfBounds()	Force the aircraft to fly to a waypoint outside of the bounds of the game, then test that this returns true when the aircraft goes out of bounds.	Pass
testBearing()	bearing()	Checks that the value of the bearing is calculated correctly.	Pass
testSpeed()	speed()	Checks that the value of the speed is calculated correctly.	Pass

testIsAt()	isAt()	Test that this returns true if the aircraft is within 16 units of the passed in vector.	Pass
testIsTurningLeft()	isTurningLeft()	Tests that this teturns true if the plane is turning left.	Pass
testIsTurningRight()	isTurningRight()	Test that this returns true if the plane is turning right.	Pass
testFlightPathContains()	flightPathContains()	A waypoint is passed into the method, and the method returns an integer denoting the position of that waypoint if the aircraft's flight plan. It will return a -1 if the waypoint is not in the flight plan.	Pass
testAlterPath()	alterPath()	A waypoint and integer is passed in to the method. The new waypoint should replace the original waypoint in the flight plan at the position of the passed in integer.	Pass
testIsMouseOverIntInt()	isMouseOverIntInt()	Tests that the returned true when the passed pair of integers are within 256 units of the aircraft's position.	Pass
testTurnLeft()	turnLeft()	Test that the aircraft turns left, so the next position of the aircraft is to the left of a straight line projection from the aircraft.	Pass

testTurnRight()	turnRight()	Test that the aircraft turns right, so the next position of the aircraft is to the right of a straight line projection from the aircraft.	Pass
testFindGreedyRoute()	findGreedyRoute()	Tests that the returned flight plan matches the calculated flight plan of waypoints from entry point to the exit point.	Pass
testUpdateCollisions()	updateCollisions()	A new aircraft is generated and spawned in the same location and altitude as the original aircraft. This should result in the original aircraft registering a collision, resulting in the isFinished() method to return true.	Pass
testToggleManualControl()	toggleManualControl()	Test that the boolean value of isManuallyContolled() is toggled when this method is called.	Pass
testSetManualControl()	setManualControl()	Test that the value of isManuallyControlled() is forced to the boolean value passed into this method.	Pass
testToggleLand()	toggleLand()	Test that the boolean value of isLanding is toggled when this method is called.	Pass
testClimb()	climb()	Tests that it changes the velocity to include a z value, which makes the altitude of the	Pass

		aircraft to increase with time.	
testFall()	fall()	Tests that it changes the velocity to include a z value, which makes the altitude of the aircraft to decrease with time.	Pass
testDecreaseTargetAltitude()	decreaseTargetAltitude()	Tests that the target altitude is set to the value of the layer below iit.	Pass
testIncreaseTargetAltitude()	increaseTargetAltitude()	Tests that the target altitude is set to the value of the layer above it.	Pass
testSetAltitude()	setAltitude()	Tests that the z component of the aircraft's position is set to the value passed into it by the method.	Pass
testGetPoints()	getPoints()	Tests that the returned value of the method is number of points assigned to the aircraft.	Pass
testGetDestination()	getDestination()	Test that the returned waypoint is the destination waypoint of the current aircraft.	Pass

Tests not implemented:

There were 7 public methods in the Aircraft class that we believed would be better tested under functional testing. The methods are listed below:

- testIsMouseOver()
- testUpdate()
- testDraw()

- testDrawCompass()
- testDrawFlightPath()
- testDrawModifiedPath()
- testSetBearing()

2.1.5 Vector (ID: JUnit5)

Vector was an existing class. All methods that were unaltered from the previous assessment are in **Bold**.

Test Name	**Method Under Test**	**Description**	**Pass/Fail**
testX()	X()	Tests correct x value is returned.	Pass
testY()	Y()	Tests correct y value is returned.	Pass
testZ()	Z()	Tests correct z value is returned.	Pass
testEqualsObject()	equalsObject()	Checks that two vectors containing different values are not equal.	Pass
testMagnitude()	magnitude()	Checks that the correct magnitude value is returned.	Pass
testMagnitudeSquared()	magnitudeSquared()	Checks that the correct magnitude value is returned.	Pass
testNormalise()	normalise()	Tests that a normalised vector is returned.	Pass
testScaleBy()	scaleBy()	Tests that a scaled vector is returned.	Pass
testAdd()	add()	Tests that the correct values are returned when two vectors are added together.	Pass

testSub()	sub()	Tests that the correct values are returned when one vector is subtracted from another.	Pass
testAngleBetween()	angleBetween()	Checks that the correct angle between two vectors is returned	Pass
testSetZ()	setZ()	Checks that Z is being set to the correct value when that method is called.	Pass

2.2 Functional Testing

Functional testing in this assessment was only done on the additional features and changes we made to the software. Any existing functionality was not tested, as it was already tested in the previous assessment. Functional testing will ensure that the classes in the game communicate properly, and work together in order to correctly function as the game is played. If classes are not communicating across each other correctly, it will be caught here, and can be fixed before moving forward. Some functions which could not be testing in unit testing (such as draw functions, sections which require direct user input etc.) will be tested here to ensure they are working correctly.

Our testing plan will be to define a test action, then compare the result of performing that action against the expected result. If the predicted result and actual result are the same, the test has passed, as the software is doing what we expected it to do. The test actions are a series of inputs into the software to simulate actual usage.

Tests were designed by going through each feature and change we added to the software, and breaking them down into discrete functions to be tested. This allows individual expected results to be tested, ensuring the tests are not too generalised in which function they are testing.

Landing:

Action	**Condition**	**Expected Result**	**Actual Result**	**Pass/Fail**
Select an aircraft.	Aircraft must have a destination of the airport and have an altitude of over 5000ft.	The land button will show "lower altitude" until the altitude of the aircraft is 5000ft.	The land buttons displays "lower altitude" until the aircraft is at 5000ft or below.	Pass
Select an aircraft.	Aircraft must have a destination of the	The landing button should appear next to the take control button	Landing button appears.	Pass

	airport.	at the top of the screen.		
Select an aircraft.	Aircraft must not have a destination of the airport.	The landing button should not appear.	Landing button does not appear.	Pass
Allow the game to generate an aircraft.	The aircraft must have a destination of the airport.	As the aircraft reaches the airport, it should circle around the airport following a series of holding waypoints, until it is ordered to land.	Aircraft follows a series of holding waypoints which encircle the airport.	Pass
Select an aircraft and click the land button.	Aircraft must be circling around the airport and have an altitude of 5000ft.	The aircraft should cause the aircraft to land, moving toward the airport.	The aircraft stops circling the airport, and goes to the airport.	Pass
Select an aircraft and click the land button.	Aircraft must be circling around the airport and have an altitude of 5000ft.	The aircraft should slow its speed as it goes to land.	The aircraft's speed is reduced.	Pass
Select an aircraft and click the land button.	Aircraft must be circling around the airport and have an altitude of 5000ft.	The aircraft should lower its altitude as it is landing.	The aircraft's altitude is lowered.	Pass
Select an aircraft and click the land button.	An aircraft must've landed within five seconds of ordering the current aircraft to land.	The aircraft should not land, until five seconds has elapsed after the previous aircraft has landed.	The aircraft does not land, until five seconds after the previous aircraft has landed.	Pass
Select an aircraft and press the "f" key.	Aircraft must be circling around the airport and have an altitude of 5000ft.	The aircraft should cause the aircraft to land, moving toward the airport.	The aircraft lands at the airport.	Pass
Select an	Aircraft must be	The aircraft should not	The aircraft does not	Pass

aircraft and press the "f" key.	circling around the airport and have an altitude of over 5000ft.	be allowed to land.	land, continues circling the airport.	
Select an aircraft and press the "f" key.	Aircraft must be circling around the airport and have an altitude of 5000ft.	The aircraft should slow its speed as it goes to land.	The aircraft slows down and moves to the airport.	Pass
Select an aircraft and press the "f" key.	Aircraft must be circling around the airport and have an altitude of 5000ft.	The aircraft should lower its altitude as it is landing.	The aircraft's altitude is lowered as it moves to the airport.	Pass

Take off:

Action	**Condition**	**Expected Result**	**Actual Result**	**Pass/Fail**
Click on the airport waypoint.	Airport must have at least one aircraft in it.	An aircraft should spawn next at the airport.	An aircraft is spawned to the left of the airport.	Pass
Click on the airport waypoint multiple times.	Airport must have several aircraft in it.	Every aircraft spawned should begin moving in the same direction.	All aircraft are spawned moving to the left.	Pass
Click on the airport waypoint multiple times.	Airport must have several aircraft in it.	aircraft should only be allowed to spawn five seconds after the previous aircraft was spawned.	The game only allows aircraft to be spawned five seconds apart.	Pass
Click on the airport waypoint.	Airport must have at least one aircraft in it.	Aircraft should spawn at an altitude of 100ft and immediately start increasing its altitude.	aircraft are spawned at 100ft, and increase altitude to a random target altitude.	Pass

Airport:

Action	Condition	Expected Result	Actual Result	Pass/Fail
Land an aircraft in the airport.	Airport must have ten aircraft in it before the new aircraft attempts to land.	The game should end, and present the user with the game over screen.	The game ended.	Pass
Land an aircraft in the airport.	Airport must have less than ten aircraft in it before the new aircraft attempts to land.	The listed number of aircraft in the airport should go up by one when the airport successfully lands.	When an aircraft lands at the airport. The displayed number of aircraft goes up by one, and the number of aircraft that can take off from the airport goes up by one.	Pass
Start a game.	Any.	The airport must display the number of aircraft currently in the airport next to the airport.	The airport has a number next to it, which indicates the number of aircraft in the airport. This number goes down as aircraft take off, and goes up as aircraft land.	Pass
Lead an aircraft to its exit point without letting it breach any separation rules.	Game must be on easy difficulty.	The score should increase by 10 when the aircraft successfully reaches its exit point.	Score increases by 10.	Pass
Lead an aircraft to its exit point without letting it breach any separation rules.	Game must be on medium difficulty.	The score should increase by 15 when the aircraft successfully reaches its exit point.	Score increases by 15.	Pass

Lead an aircraft to its exit point without letting it breach any separation rules.	Game must be on hard difficulty.	The score should increase by 20 when the aircraft successfully reaches its exit point.	Score increases by 20.	Pass
Lead an aircraft to its exit point forcing it to breach its separation rules.	Any.	The number of points added to the score (determined by difficulty) should be reduced by 5 for each time the aircraft breaches the separation rules.	Points added to the score is reduced by 5 every time the aircraft breaches its separation rules.	Pass
Click on the airport waypoint.	Airport must have at least one aircraft in it.	A countdown should appear next to the waypoint, counting down 5 seconds before the next aircraft is allowed to take off.	When an aircraft takes off from the airport, a countdown appears at the airport, and aircraft cannot take off until the countdown is finished.	Pass

Keybindings:

Action	**Condition**	**Expected Result**	**Actual Result**	**Pass/Fail**
Press the "a" or "s" keys.	An aircraft must be selected.	The aircraft should stop automatically following its flight plan, and the player will have direct control over its movement.	Pressing "a" causes the aircraft to move left, ignoring its flight plan. Pressing "s" causes it to move right, ignoring its flight plan.	Pass
Press the space bar.	An aircraft must be selected, and under manual control.	The aircraft should return to automatically following its flight plan.	The aircraft stops manual control and returns to following its flight plan.	Pass
Press the "w" key.	An aircraft must be selected, and have an altitude	The aircraft should increase its altitude.	The aircraft's altitude increases.	Pass

	below 15000ft.			
Press the "s" key.	An aircraft must be selected, and have an altitude over 5000ft.	The aircraft should decrease its altitude.	The aircraft's altitude decreases.	Pass

High Score:

Action	**Condition**	**Expected Result**	**Actual Result**	**Pass/Fail**
Start the application.	Any.	There should be an option to open the high scores screen on the main menu.	A high scores option appears on the main menu.	Pass
Finish a game.	The end score must be larger than 0.	The score should be saved and visible on the high scores screen.	The score appears in the high scores screen.	Pass
Open high scores screen.	There must be at least two saved scores of different values.	The scores should be listed in descending order.	The highest scores appear at the top of the high scores screen, and lower scores appear at the bottom of the screen.	Pass

2.3 Acceptance Testing

Introduction

This section tests that each system requirement added in assessment 3 by our team works properly. It also consists of the testing evidence of the system requirements and tests defined by team FLR in assessment 2 **(Noting the formatting of their tests has been modified in a manner that our team finds best)**.

Running through the system requirements in this manner allowed missing features to be found, so they could be implemented. Such requirements will be listed as a pass if the feature has been implemented, however a note will be added to inform the reader which features were implemented after the initial run of acceptance testing.

Requirements Tests

Each system requirement has a test written to test that it has been implemented. The tests all have an ID number associated with them to allow a trace matrix to be drawn showing the overall link between the tests and the requirement they test. The first sub-section consists of acceptance tests conducted on the new system requirements and the second sub-section

consists of acceptance testing evidence of the function requirements defined by team FLR in assessment 2.

2.3.1 Acceptance Testing Evidence for NEW System Requirements

System Requirement 1

"The game should display the score of the user at every point of time"

Test (ID: N1)

Start new game. Play.

Result: Pass

The score will be displayed on the top right corner at all times.

System Requirement 2

"The initial score should be different in the different difficulties"

Test (ID: N2)

Start game in 3 different difficulties and see the score

Result: Pass

The initial score displayed will be different in the 3 difficulties.

System Requirement 3

"Score should decrease when the separation rules are breached"

Test (ID: N3)

Star game and play. Breach separation rules voluntarily.

Result: Pass

The score would decrease every time the separation rules are breached

System Requirement 4

"The game should contain an airport. The airport should store aircraft and display the number of aircraft in it at any point of time"

Test (ID: N4)

Start new game. Play.

Result: Pass

The airport should be displayed on the game screen. The number of aircraft should be displayed on the airport.

System Requirement 5

"The airports should spawn aircraft in the game"

Test (ID: N5)

Start new game. Play.

Result: Pass

You would notice flight generation from the airport.

System Requirement 6

“The game should allow a voluntary aircraft take off feature for the users”

Test (ID: N6)

Start new game. Simply click on the airport waypoint (The red dot on the airport).

Result: Pass

You would notice a flight taking off from the airport. The number of aircraft currently at the airport would decrease.

System Requirement 7

“The game should allow the users to land aircraft at the airport”

Test (ID: N7)

Start new game. Select a Flight. Lower the altitude of the flight to 5000. The lower altitude button on the Centre top of the game screen would change to a Land button. Press Land button.

Result: Pass

The flight would land at the airport. Number of aircraft at the airport should increase by

System Requirement 8

“The game should allow at least 10 aircraft to be on the game screen”

Test (ID: N8)

Start game. Press the left control key (Ctrl) multiple times in a row - This would spawn aircraft. (Noting this key was only kept for debugging)

Result: Pass

This would spawn as many aircraft as you want.

2.3.2 Acceptance Testing Evidence for OLD System Requirements

System Requirement 9

“The game shall initialise aircraft entering the airspace with static flight plans. The flight plan will be a route passing through waypoints from the planes entry to it's exit point”

Test (ID: O1)

Start new game. Select an aircraft.

Result: Pass

The blue line going from the aircraft to an exit point is its initialised flight plan.

System Requirement 10

“The game GUI shall display the game airspace and active flights”

Test (ID: O2)

Start new game. Select a difficulty and start.

Result: Pass

The game screen would display airspace and moving aircraft

System Requirement 11

"The game shall allow the player to send orders to aircraft to immediately alter their flight plan"

Test (ID: O3)

Start new game. Select an aircraft and send orders (for e.g. turn left)

Result: Pass

The aircraft would respond to the order immediately.

System Requirement 12

"There shall be at least 3 entry and 3 exit points and the exit points will correspond to given destinations"

Test (ID: O4)

Start new game. See the points on the border of the game screen from which aircraft enter and exit.

Result: Pass

The flights are generated from these points.

System Requirement 13

"The game shall check aircraft separation regularly."

Test (ID: O5)

Start new game. Play game and voluntarily bring to aircraft close enough.

Result: Pass

When 2 aircraft are close enough (depending on difficulty) the circles around the plane check and indicate that the separation rules are breached.

System Requirement 14

"The game shall end when the distance between one aircraft and another is lower than the specified separation rules."

Test (ID: O6)

Start new game. Select and aircraft and voluntarily breach the separation rules for 2 aircraft to crash.

Result: Pass

The game would end.

System Requirement 15

"The game shall track an aircraft's position and speed while it is within the airspace"

Test (ID: O7)

Start new game. Play.

Result: Pass
The speed of the aircraft is displayed at the bottom left corner and position is displayed next to the selected aircraft.

System Requirement 16
"The game shall calculate the score as a function of time played and successful flights"

Test (ID: O8)
Start new game. Play the game for a while.

Result: Pass
Notice the score would be a function of time and successful flights. (Noting the exact mechanism can only be tested in the code and unit testing)

System Requirement 17
"The game shall run on PCs provided in the Computer Science labs"

Test (ID: O9)
Look at Deployment Testing.

Result: Pass
Proved in deployment testing

System Requirement 18
"The game shall allow the use of the keyboard and mouse as input devices"

Test (ID: O10)
Start new game. Read User manual. Play the game with the instructions provided.

Result: Pass
Keyboard and mouse would be functional

System Requirement 19
"The game shall allow for separation rules of aircraft to scale against difficulty"

Test (ID: O11)
Start new game. Play game in all 3 difficulties. Select a plane. The circle around the plane

Result: Pass
The size of the circle around the plane defines the separation radius and would vary in the 3 different difficulties

System Requirement 20
"The game shall allow aircraft to vary the rate at which they climb / descend"

Test (ID: O12)
Start new game. Play.

Result: Pass
The climb/descend rate would vary with difficulty. In Easy altitude changes quickest whereas in hardest it takes time.

System Requirement 21

"The game shall provide at least ten fixed waypoints within the airspace"

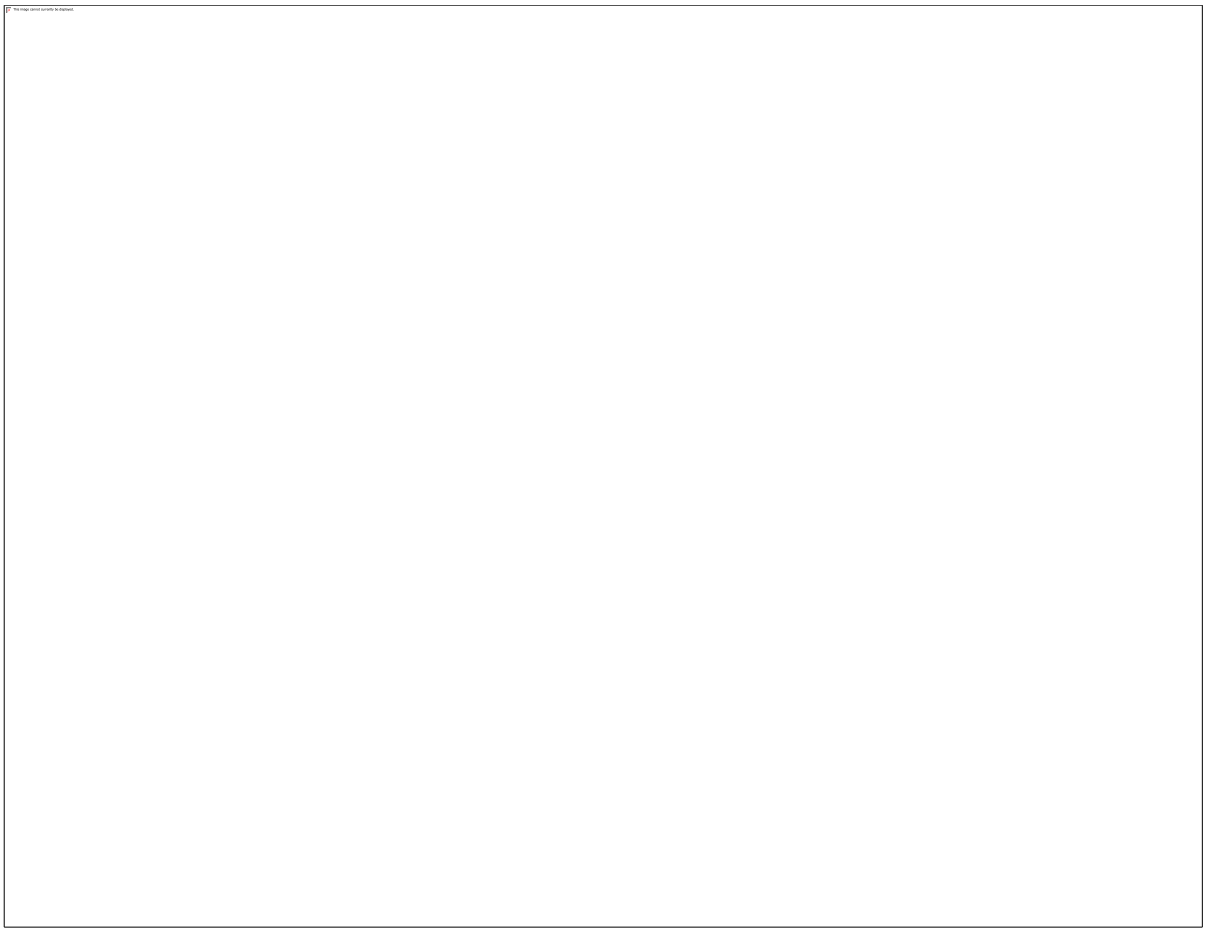
Test (ID: O13)

Start new game.

Result: Pass

The 7 normal waypoints, 1 airport, 4 entry and exit points and 4 surrounding the airport.

Trace Table



2.4 Deployment Testing

Deployment testing is checking that the game runs on the correct platforms. Although we were aware that Java is platform independent, we decided to test it on different operating systems to make sure nothing unexpected occurred. We chose to test it on the two most common operating systems: Windows and Linux. For Windows and Linux, we used the machines in the software laboratories.

To do this, we simply ran the game on each different operating system and checked that the window and all the different screens appeared as expected. It was unnecessary to run all the functionality tests on each different OS due to the cross platform nature of Java. Deployment testing evidence is available in Appendix A.

2.5 Alpha and Beta testing

Alpha testing and beta testing are two new methodologies implemented by our team in Assessment 3.

The key reason to include alpha testing was to find bugs that were very unusual and didn't come to our attention. One of the secondary reasons to do this testing was to check the overall impression that our game created on users and the way they perceived it. Alpha testing was basically conducted in 3 phases. The first phase being the unit, integration, functional and acceptance testing shown previously in the report. The second phase was when all the team members of our testing team played the game and tried to take notes of the bugs that we detected and the unusual behaviours that were noticed in the game.

The third and the main phase of alpha testing was to make people who are not part of our team play the game. The audience that seemed appropriate for alpha testing was an audience who was not part of our project but had a good idea about what we were doing. Hence the best fit audience for this were the current 2nd year SEPR students. We choose 5 SEPR students as our test cases and made them play the game. We gave them access to the user manual and gave them 1 trial chance to get acquainted to the game. Then we gave them 3 chances to play the game and post that asked them to fill a small questionnaire created by us. We developed this questionnaire and we believe it covers all the aspects that we want to know and get out of this testing process.

Beta testing was our last stage of the testing process. We believed that the game now with all the new functionalities is ready for actual audiences to play. We think it is fun and playable. Hence, this final testing was just conducted to receive feedback about our game and also discover some unusual bugs that a real-audience would detect. We conducted this in the same way as we conducted the third phase of alpha testing, just the audience this time was 5 non-computer science students but avid game players.

While conducting these test phases, we unfortunately neglected to officially document the ethical guidelines we followed. However, every participant was fully aware of the nature of the questionnaires, was under no harm and was able to stop participating at any time. Every participant has also been granted anonymity.

2.5.1 Questionnaire, Test Cases and Results

Below is the questionnaire that we developed to conduct Alpha and Beta testing:

Question No.	Question	Rating	Comments
1	Did you discover any bugs in the Game? If so please specify in the comments box.	Yes/No	
2	Did all the buttons work properly? If not please specify in the comments box.	Yes/No	
3	Any missing functionality? If so	Yes/No	

	please specify in the comments box.		
4	Did you have fun while playing the game?	1/2/3/4/5 (1 Bad, 5 Great)	
5	Did you find it engaging?	1/2/3/4/5 (1 Bad, 5 Great)	
6	Was it challenging?	1/2/3/4/5 (1 Not at all, 5 Very)	
7	Was the User Manual helpful?	1/2/3/4/5 (1 Not at all, 5 Very)	
8	Is the User Interface Intuitive?	1/2/3/4/5 (1 Not at all, 5 Very)	
9	Did you find the scoring system appropriate?	1/2/3/4/5 (1 Not at all, 5 Very)	
10	What was your highest score?		
Any Suggestions?			

Below are the test cases used for alpha and beta testing:

Test Case ID (ALPHA)	Alpha Test Case	Test Case ID (BETA)	Beta Test Case

A1		B1	
A2		B2	
A3		B3	
A4		B4	
A5		B5	

Below are the average test results of the questionnaire:

Question No.	Alpha Test Results	Beta Test Results	Comments
1	3 Yes's 2 No's	4 Yes's 1 No	- Aircraft spawning not appropriate - Aircraft landing at 5000ft could not land - Airport full crash bug
2	2 Yes's 3 No's	4 Yes's 1 No	- Take control button didn't do anything
3	3 Yes's 2 No's	5 No's 0 Yes	- Changing speed - Altitude colours
4	Average of 3.6	Average of 3.6	
5	Average of 3.8	Average of 3.6	
6	Average of 4	Average of 4.2	
7	Average of 3.2	Average of 4	
8	Average of 3.4	Average of 3.4	

9	Average of 3.6	Average of 3.8	
10	Average of 175	Average of 162	
Suggestions	- Add keyboard shortcut for remove control - Add scoring in the manual - Make the text size for the altitude bigger - Add flight plans even when the plane is not selected - Improve the explosion animation		

2.5.2 Reported bugs and actions take Post Alpha and Beta Testing

The bugs that were flagged up by these two phases were: issues with aircraft spawning, aircraft landing issues and the take control button. The last of these three appeared to be a problem with the unresponsiveness of the mouse touchpad that the player was using but the other two issues were fed back to our programming team to be fixed. We also added an explanation of the scoring system in the manual.

3. Risk Table and Conclusion

3.1 Risk Table

The testing phase is like a big sub-project within the main project. Hence we believe that there are various independent risks that are attached to it. The complete description of the risks in provided below in the risk table:

Risk Type	**Risk**	**Risk ID**	**Probability**	**Impact**	**Mitigation**	**Contingency**
Project	The concept of extending a testing for a project is misunderstood.	1.1	3	4	The test report is considered to be inappropriate.	Clarify with the module tutors in the team checks
Project	Tester Falling ill	1.2	2	2	Testing cannot be completed	Other members of the team

						help in testing.
Product	Not Appropriate testing methodologies used.	2.1	2	4	Using methodologies are not appropriate for the current Assessment and making the testing process moot.	Conduct adequate research on testing methodologies
Project	Testers falling short of time	1.3	4	4	During testing the testers realize that the final week allotted to testing is not adequate and fail to conduct the desired amount of testing	The plan made for testing should be thorough and should allow space to handle mitigations to an extent.
Product	Inefficient testing	2.2	2	3	More time is spent on testing methodologies that are less important than the ones that are more important.	Have a better plan and know your priorities before starting.

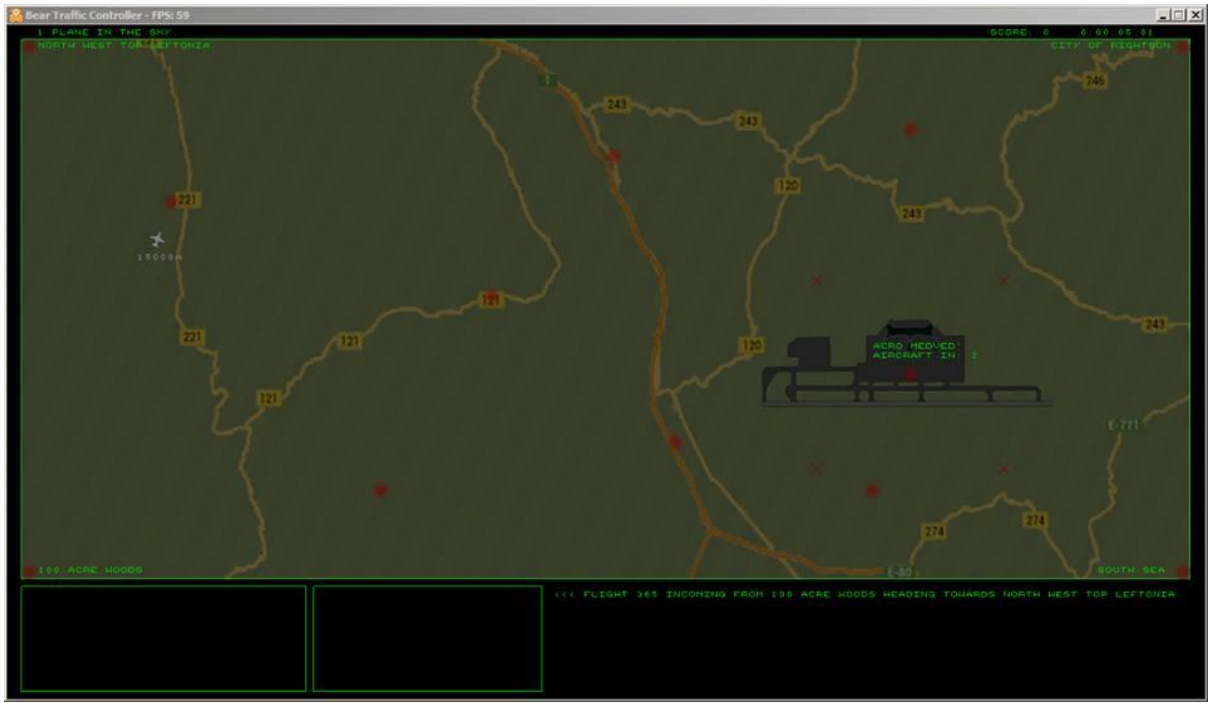
Key:

Score	Probability	Impact
1	0.25	Very negligible impact
2	0.5	Considerable impact
3	0.75	Highly impactful
4	1	Enormous Impact

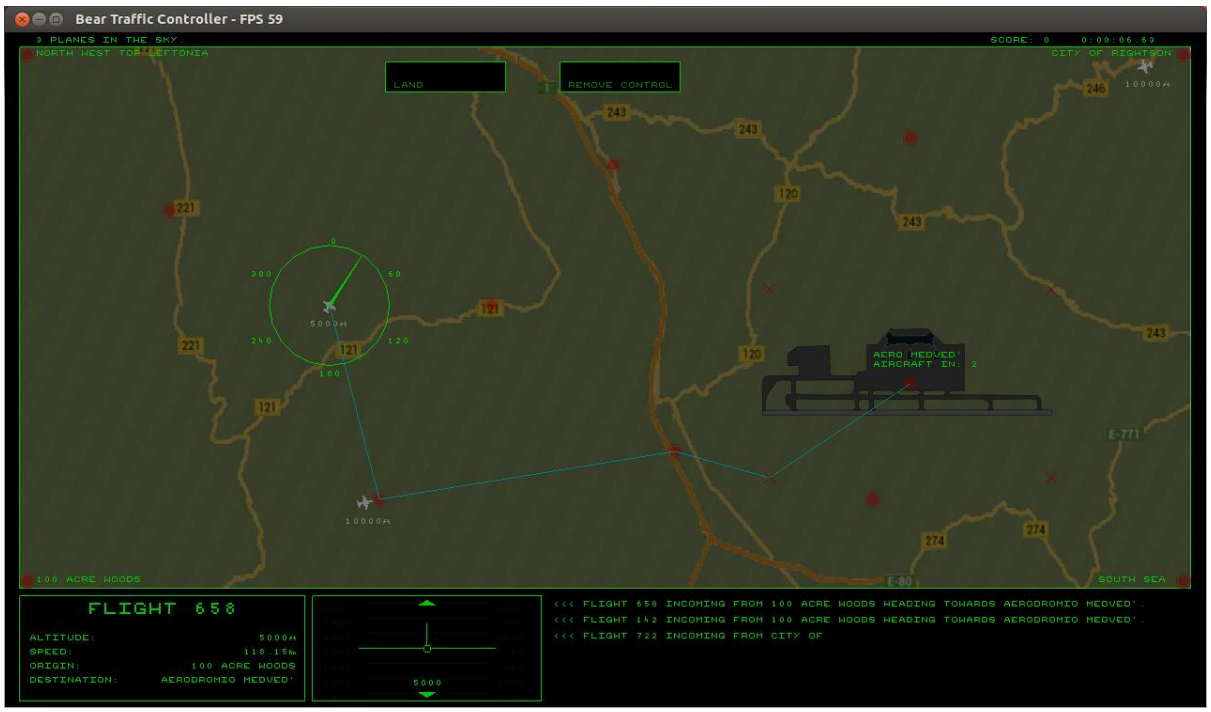
3.2 Conclusion

We believe that the testing process in assessment 3 is more extensive, thorough and detailed than assessment 2. We believe the introduction of a few of new testing methodologies and strategies gave us an edge to find bugs more efficiently. The use of various testing methodologies: Unit, integration, regression, functional, acceptance, deployment, alpha and beta and the immense effort put in by every testing team member gives us confidence that we have achieved our goal of developing an ideal product as we said in the introduction.

Appendix A



Game running in windows



Game running in Linux