Testing of INI

Introduction

For assessment 3 we once again decided to use Test-driven development this is due to the fact that we have had previous experience using this method in past assessments and as we only had a couple of features to add we could discuss what each feature was meant to provide then write a unit or functional tests to see if the implemented code achieved its goal. When doing unit tests we either used a JUnit[1] or where this would be difficult two members of the team worked through the code .

The testing of project that we took for assessment 3 was quite comprehensive but modifications and additions to the tests already in place had to be made to allow for code that we wanted to introduce to work. The changes we had to make to existing tests were very minor such as adding one line of code to select a plane from a hash table or initialising new images that were required in a test.

Testing Requirements

As you can see in the tables below there is a requirements satisfied column, while every test may not satisfy a requirement, each requirement should be satisfied by at least one test. We decided to test non-functional requirements as they still have goals which tests can be written for. All functional requirements and the majority of non-functional requirements appear in the table below. The reason some non-functional requirements are not satisfied is due to the time constraints we were working under. The non-functional requirements we could not test were: The game should be easy to learn (User 3.1), the game must be enjoyable (User 3.2) and the user must view an aesthetically pleasing GUI (User 3.3) as they would require a batch of beta tests and a resulting questionnaire to find if they had been achieved. The requirements can be found in the appendix they have all of the requirements from assessment 1 as well as new requirements that were added in assessment 3.

Test Table No.1

The table below is a testing table for all new features we have added to the project, we have used a mixture of Unit, Integration and Functional/Acceptance tests. We have chosen not to test very simple methods like Getters and Setters due to the JVM already having tests in place for methods like this.

Test ID	**Test Type**	**Class**	**What is under test?**	**How was it tested?**	**Tester /s**	**Est. run time**	**Success Condition**	**Requirements Satisfied**
1.1.1 (fig. 4)	Unit	State	Reset()	A JUnit test was created that reset the scores and timer	David	< 10 ms	The scores and timer should be equal to 0	
1.1.2 (fig. 5)	Unit	EndScreen	EndScreen()	The code was read through by two members of the team to check that it functioned as expected	Dan	-	The results predicted at the beginning should be equal to those produced when walking through the code.	
1.1.3	Unit	State	changeScore()	The difficulty was set and a JUnit test was written that passed a float to the changeScore() method.	David	< 10 ms	The value predicted for the score to be should match what is returned from the changeScore() method	G.U.I User 1.1.4 Mathematical System 2.1.3
1.1.4	Unit	Art	preLoad(), load()	The code was read through by two members of the team checking that result calculated from the walkthrough are equal to results that the program gets	David	-	The walked through result should be equal to that produced by the program	G.U.I System 1.1.2 G.U.I System 1.1.3 G.U.I System 1.1.4 G.U.I System 1.1.5
1.1.5	Unit	AircraftController	AircraftController()	The code was read through by two team members	David	< 10 ms	The game will be able to handle all of the planes spawned without performance issues.	G.U.I System 1.1.8

1.1.6	Unit	Airport		A JUnit test was written that creates 4 planes they are then instructed to land.	Jamaal	< 10 ms	The game should let the first 3 planes land and create a new flight path for the plane that is rejected.	G.U.I System 1.1.9 G.U.I System 1.1.10

1.2.1	Integration	Leaderboard	addLeaderBoardEntries()
			addLeaderboardEntry()
			sortLeaderboard()
				A JUnit test was written that created an array of scores which were then passed to the addLeaderBoardEntries() method	David	< 200 ms	The values in the leaderboard array should be equal to those in the leaderboard	G.U.I User 1.1.5

1.3.1	Functional	State, SidebarController	incScore(), update()	The game was run and hard difficulty was selected	Sam	< 5000 ms	The sidebar when playing the game should update with 2 being added to the score every second and 50 being added for every plane successfully navigated	
1.3.2	Functional	AircraftController, AircraftType	AircraftController(), AircraftType()	The game was played allowing for many planes to be spawned	Paul	< 5000 ms	The game was played for a while waiting for a snake plane to spawn, it was also recorded at what frequency compared to other planes the snake planes came onto the screen. Which should be around 1/7.	G.U.I User 1.1.8 G.U.I User 1.2.4
1.3.4	Functional	Waypoint	RandomFlightPaths	Two planes were spawned from the same entry point	Jamaal	< 5000 ms	The planes flightpaths were then compared to make sure that they were actually different	
1.3.5	Functional	Aircraft	increaseSpeed(), decreaseSpeed()	The fastest and slowest planes in the game were spawned	Paul	< 5000 ms	The planes speed were changed making sure they stay in the bounds that they should do	
1.3.6	Functional	Cloud	Cloud()	The game was run to	Dan	<	The clouds should slowly	G.U.I System 1.2.5

				check that clouds had been implemented correctly		5000 ms	move from one side of the screen to the other without interfering with any features of the game	
1.3.7	Functional	Map	Map()	The map was loaded from the textures hash table.	Sam	< 5000 ms	A map depending on the theme chosen should be displayed to the user, behind all planes, waypoints and exit points.	G.U.I User 1.1.2 G.U.I System 1.1.2
1.3.8	Functional	Aircraft	Aircraft()	The game was loaded and an aircraft was spawned.	David	< 5000 ms	The aircraft will be visible to the user including its flight path through the waypoints all the way to the exit point.	G.U.I User 1.1.3 G.U.I User 1.1.6 G.U.I System 1.2.1 Mathematical System 2.1.5
1.3.9	Functional	Aircraft	additionalDraw()	A plane was created and then selected, its flight was then interrupted by pressing an arrow key.	Paulius	< 5000 ms	The additionalDraw() method should redraw the line from the plane straight to the exit point for that plane.	G.U.I User 1.1.6
1.3.10	Functional	MenuController	addLeaderboard()	The game was loaded	Paul	< 5000 ms	A leaderboard should be visible on the menu screen	G.U.I User 1.2.3 G.U.I System 1.2.2
1.3.11	Functional	SidebarController	changed()	A non-snake plane was spawned and the land button was pressed	Dan	< 5000 ms	The selected plane should head towards the airport and disappear on arrival, the number of planes in the airport should also be incremented	Interaction User 2.1.1
1.3.12	Functional	SidebarController	changed()	A non-snake plane was spawned and the up button was pressed	Sam	< 5000 ms	The altitude of the selected plane should increase if the current altitude is lower than 15000	Interaction User 2.1.2 G.U.I System 1.1.6

1.3.13	Functional	MenuController	changed()	The game was played three times each time selecting a new theme	Dan	< 5000 ms	Each time a theme is selected the game should load with a the corresponding textures.	Interaction User 2.2.1

Test Table No.2

The second test table (below) is for testing that all requirements from assessment 1 have been met by the project we have taken up.

Test ID	**Test Type**	**Class**	**What is under test?**	**How was it tested?**	**Tester /s**	**Est. run time**	**Success Condition**	**Requirements Satisfied**
2.1.1	Unit (fig. 6)	Aircraft	increaseSpeed(), decreaseSpeed()	A JUnit test was written that created two default planes. One plane was selected and the accelerated once, the other was decelerated.	David	< 10 ms	The plane that was accelerated should increase in speed and the one that was decelerated should decrease in speed. Written by previous engineering team, but modified to work for new code.	Interaction User 2.2.2 G.U.I System 1.2.4
2.1.2	Unit	Aircraft	act()	A pair of testers read through the code to make sure that there were no errors present.	Paulius, Dan		The predicted results should match those of the calculated results.	G.U.I System 1.1.1 Mathematical System 2.1.4 Mathematical System 2.2.1
2.2.1	Functional	Entrypoint	Entrypoint()	Four entry points were created and then drawn.	Jamaal	< 5000 ms	The entry points that were created should appear on screen in the correct locations.	G.U.I User 1.1.1 G.U.I System 1.1.3 G.U.I System 1.1.5
2.2.2	Functional	Exitpoint	Exitpoint()	Four exit points were instantiated and then printed to the screen in	Jamaal	< 5000 ms	The exit points should be visible to the user in the exact location that they were	G.U.I User 1.1.1 G.U.I System 1.1.3 G.U.I System 1.1.5

				different locations.			passed during creation.	
2.2.3	Functional	EndScreen	EndScreen()	Two planes were instantiated and then intentionally collided	Paulius	< 5000 ms	The game should be stopped and the end screen displayed.	G.U.I User 1.1.7 G.U.I System 1.1.7 Mathematical System 2.1.2
2.2.4	Functional	AircraftController, Aircraft	update(), separationRulesBreached()	Two planes were created, one was selected and then directed towards the other plane.	Sam	< 5000 ms	The plane which is clicked on should have a red circle appear around it, when they are steered towards each other a red ring should be visible around both of the planes.	G.U.I User 1.1.9 Mathematical System 2.1.1 Non-functional System 3.2
2.2.5	Functional	SidebarController	update()	A plane was created and then selected.	Paul	< 5000 ms	The speed label in the sidebar should update with the speed of the selected plane.	G.U.I User 1.2.1
2.2.6	Functional	MenuController, MenuScreen	menuController(), render()	The game was loaded	Dan	< 5000 ms	The first screen that appears to the user should be the menu screen	G.U.I User 1.2.2
2.2.7	Functional	MenuController, MenuScreen	menuController(), render()	The game was loaded and a game was played	David	< 5000 ms	After the game over screen the menu screen should appear	G.U.I User 1.2.2
2.2.8	Functional	SidebarController	changed()	A non-snake plane was spawned and the down button was pressed	David	< 5000 ms	The instant the down button is pressed the selected planes altitude should decrease unless the current altitude is 5000	Interaction User 2.1.2 G.U.I System 1.1.6
2.2.9	Functional	SidebarController	changed()	A plane was spawned and selected, the assign waypoint was then pressed and a waypoint was clicked	Sam	< 5000 ms	The plane should add the selected waypoint to the front of its waypoint list and then head straight towards it	Interaction User 2.1.3

2.2.10	Functional	SidebarController	changed()	The game was loaded and played. The Main Menu button was then pressed	Paulius	< 5000 ms	The game should exit to the main menu the instant the button was pressed	Interaction User 2.1.4
2.2.11	Functional	Waypoint	Waypoint()	Ten waypoints were instantiated at static locations	Dan	< 5000 ms	The waypoint that were created should appear in the correct locations	G.U.I System 1.2.3
2.2.12	Functional	GameScreen	update(), render()	The game was loaded and a fps counter was drawn on screen.	Jamaal	< 5000 ms	The fps shown by the counter should be higher than 30 most of the time.	Non-functional System 3.1
2.2.13	Functional	-	-	The game was loaded and played on 3 systems installed with Mac, Linux and windows OS	Dan	< 5000 ms	The game should run on all platforms without any change in performance or game play	Non-functional System 3.3 Non-Functional System 3.6
2.2.14	Functional	-	-	The games texture files were opened	Paul	< 5000 ms	The textures should have consistent colours that fit in with the game. All objects like waypoints and planes should also be clearly visible.	Non-functional System 3.4
2.2.15	Functional	-	-	The game was played.	Paulius	< 5000 ms	The time taken from the game to be opened and the game to be interactable should be less than 5 seconds	Non-functional System 3.5
2.2.16	Functional	Waypoint	deleteWaypoint()	The game was loaded, a user way point was created and then right clicked to remove	Jamaal	< 5000 ms	The waypoint should be removed from the airspace	Interaction User 2.1.7

Acceptance Testing

The use cases can be found in the appendix these are the exact use cases from assessment 1. An acceptance test will be passed if a user can interact with a game how the main or secondary success scenario stipulates.

Use Case 1:

Main success scenario: The system indicates that the operation has completed.
Preconditon: We loaded the game, and waited for a plane to spawn.
Trigger: We selected a plane by clicking on it.
Scenario: To control the plane we used the arrow keys to move it in the direction we wanted.
Post-condition: The plane changed direction to the desired heading.
Acceptance Test Successful?: Yes

Use Case 2:

Main success scenario: The leaderboard is shown on the main menu.
Trigger: We loaded the game.
Post-condition: We found the information we wanted from the leaderboard.
Acceptance Test Successful?: Yes

Use Case 3:

Main success scenario: Player quits the game.
Precondition: The game is being played.
Trigger: We pressed the ‘Menu' button or the Esc key.
Scenario: The game returned to the main menu.
Post-condition: The game stopped executing.
Acceptance Test Successful?: Yes, but we did not want scores being recorded when the game was quit so we just showed the main menu.

Use Case 4:

Main success scenario: The User sees the game over screen. (
Precondition: We loaded the game, selected the earth theme and started playing the game, we then waited for two planes to spawn. (fig. 1)
Trigger: The planes collide.
Scenario: When two planes spawned, we selected both and changed their heading so they would collide with each other at a similar altitude. (fig. 2)
Post-condition: The game over screen appears and the game is stopped (fig. 3)
Acceptance Test Successful?: Yes.

Use Case 5:

Main success scenario: User wants to land/exit a plane.
Precondition: We started playing the game and waited for a plane to finish moving through their designated way points.
Trigger: The plane flies through the way point.
Scenario: We selected a plane and pressed the 'land' button.
Post-condition: The plane is removed from the screen.
Acceptance Test Successful?: Yes.

Testing Examples

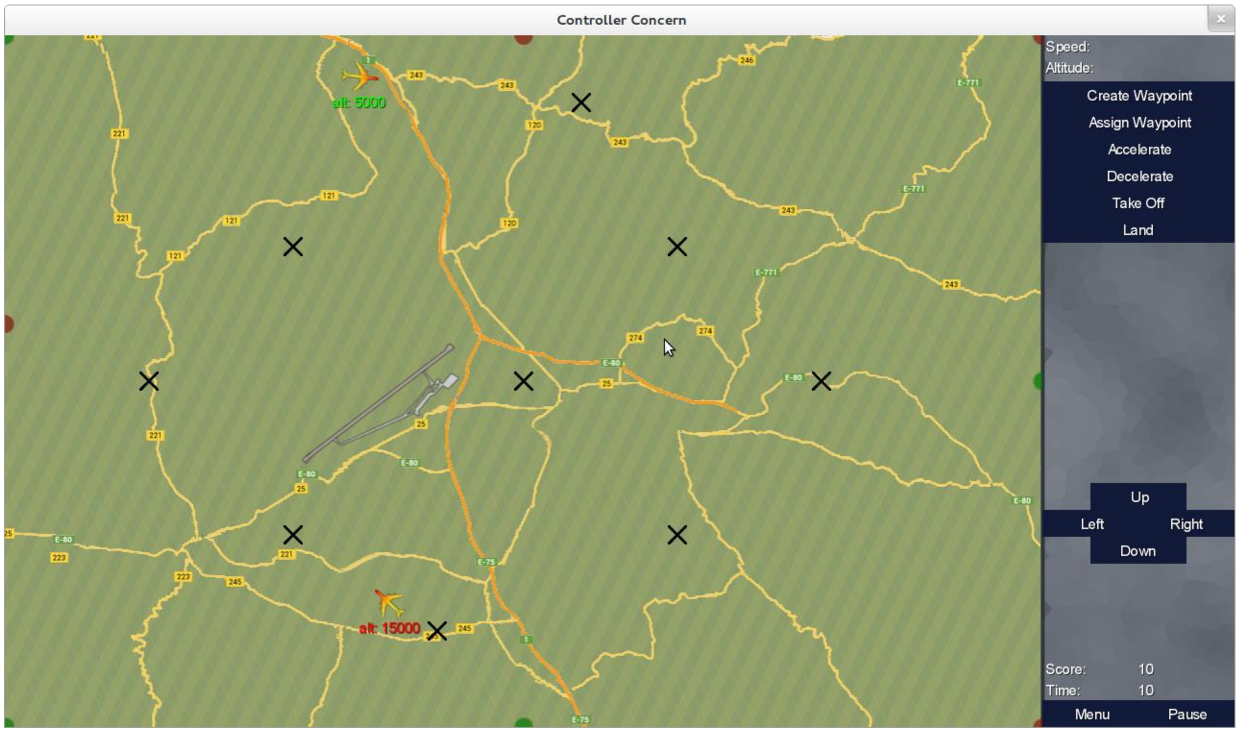


Fig. 1: Case 4 Pre-condition

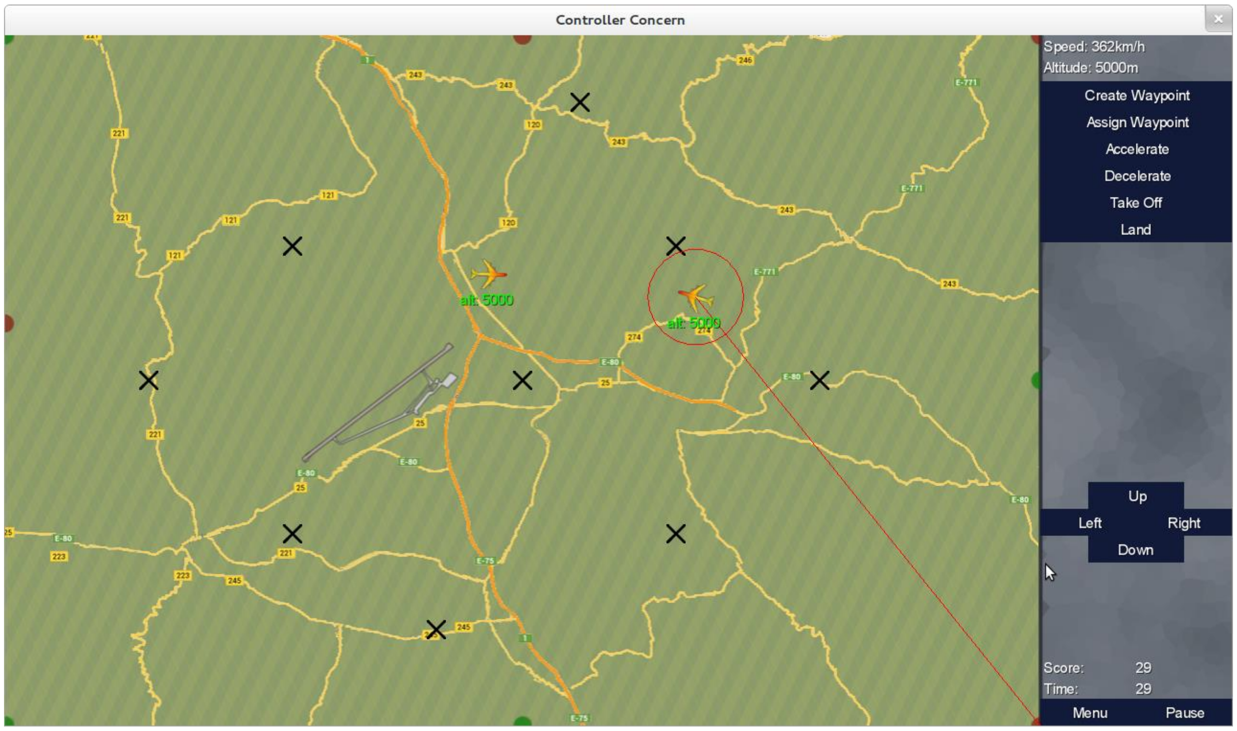
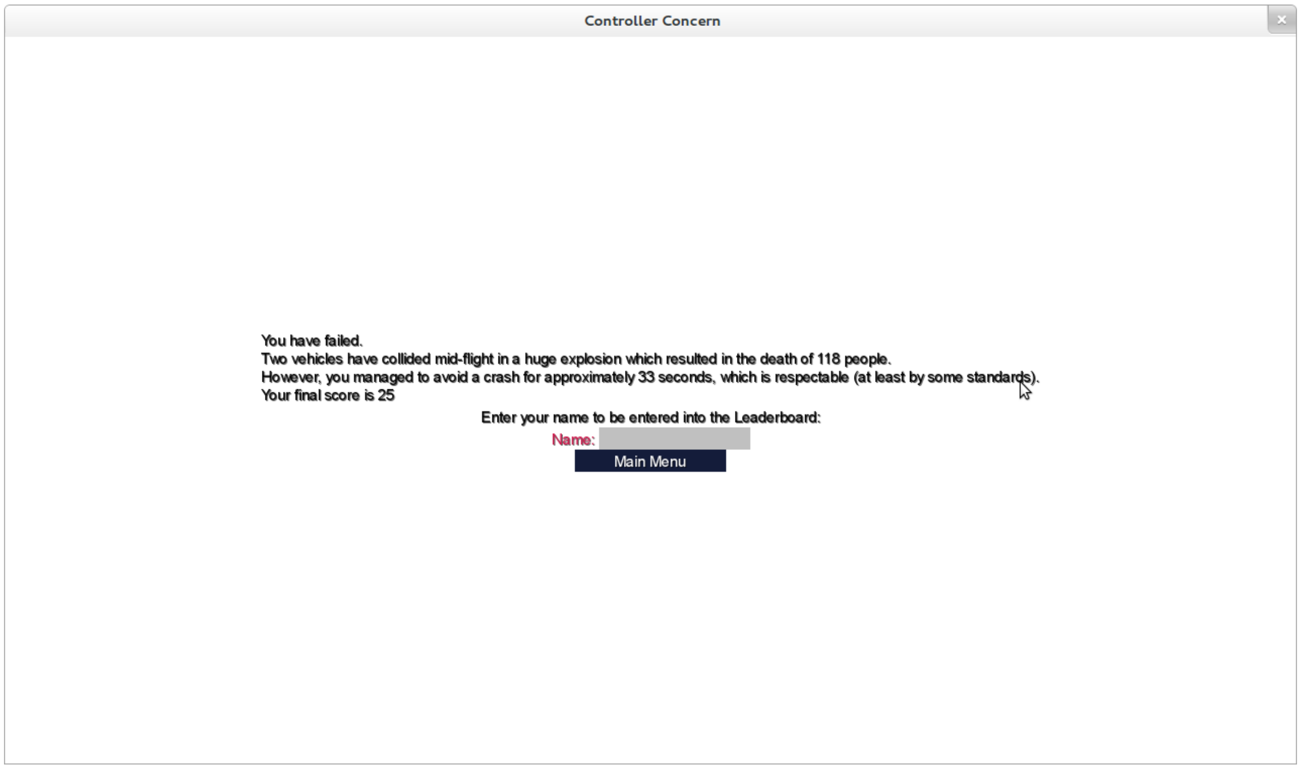


Fig. 2: Case 4 Scenario

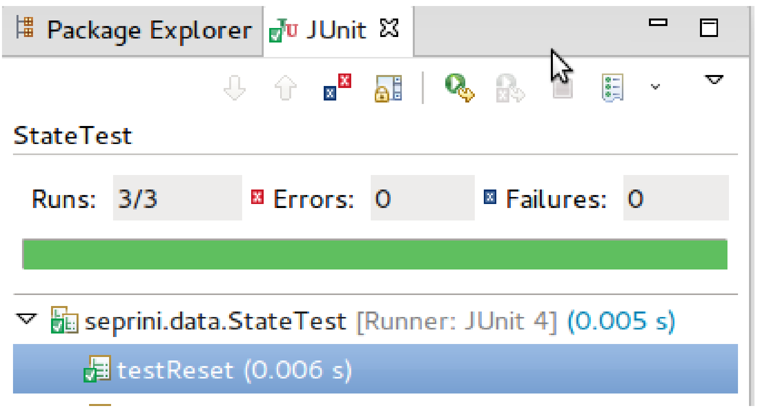
Fig. 3: Case 4 Main Success Scenario and Post-condition



Controller Concern

You have failed.
Two vehicles have collided mid-flight in a huge explosion which resulted in the death of 118 people.
However, you managed to avoid a crash for approximately 33 seconds, which is respectable (at least by some standards).
Your final score is 25

Enter your name to be entered into the Leaderboard:
Name:
Main Menu

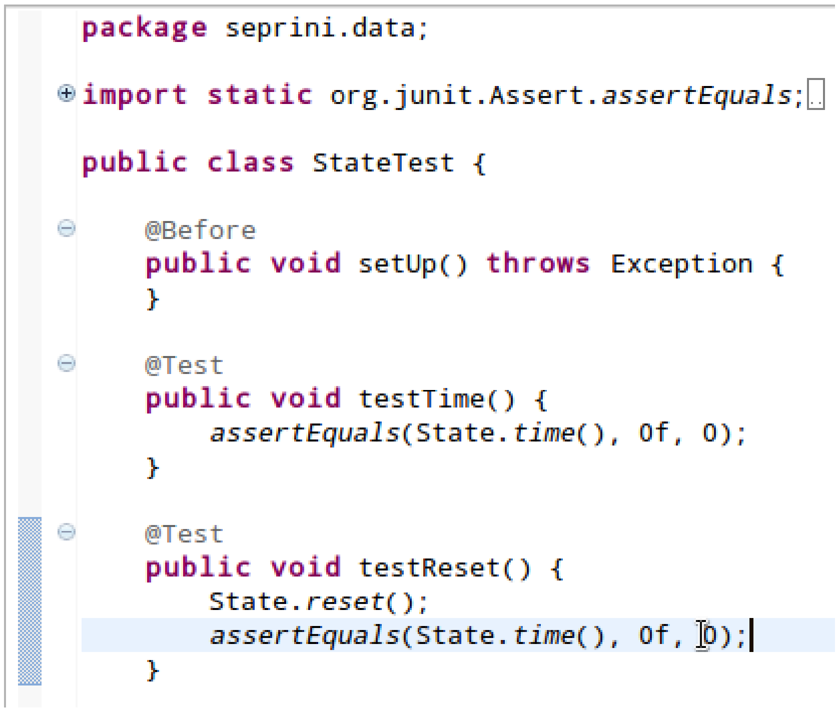


StateTest

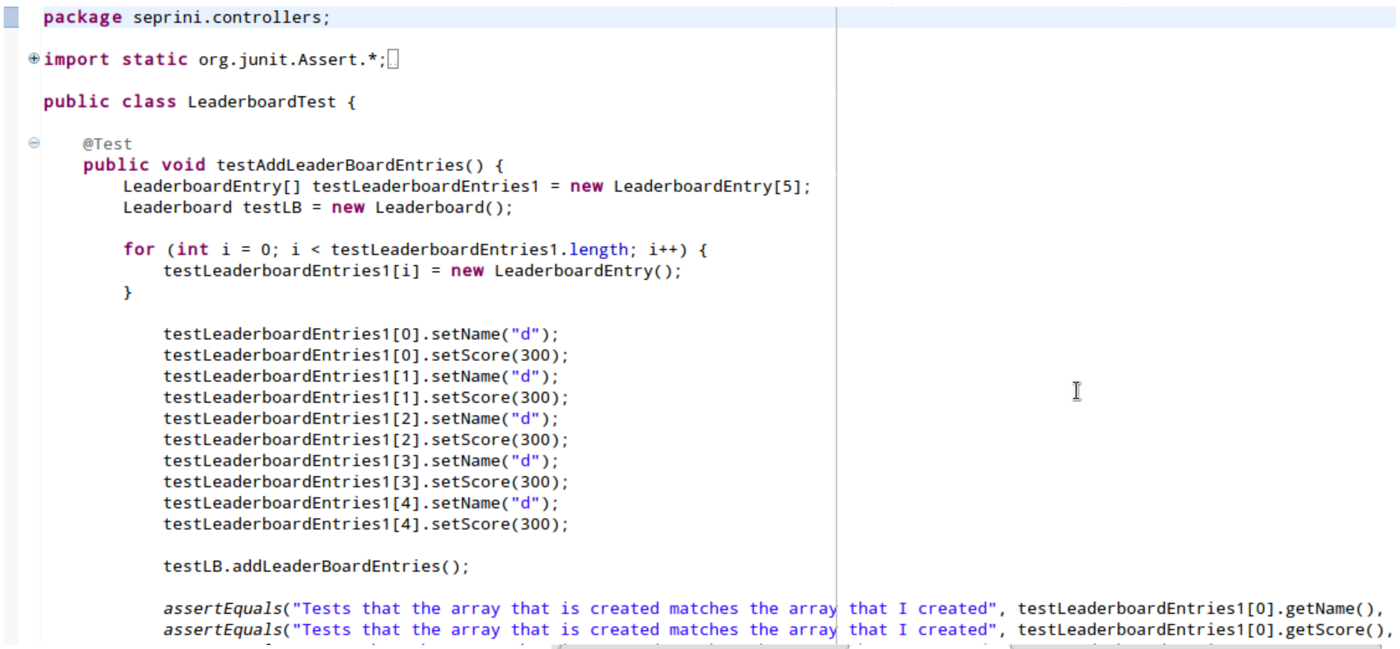
Runs: 3/3 Errors: 0 Failures: 0

seprini.data.StateTest [Runner: JUnit 4] (0.005 s)
testReset (0.006 s)

Fig. 4: Test 1.1.1 Test reset() and JUnit

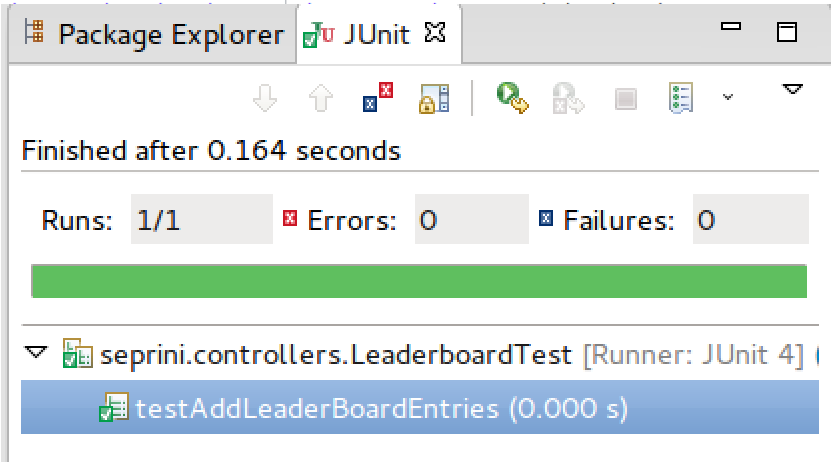


```
package seprini.data;

import static org.junit.Assert.assertEquals;

public class StateTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void testTime() {
        assertEquals(State.time(), 0f, 0);
    }

    @Test
    public void testReset() {
        State.reset();
        assertEquals(State.time(), 0f, 0);
    }
```

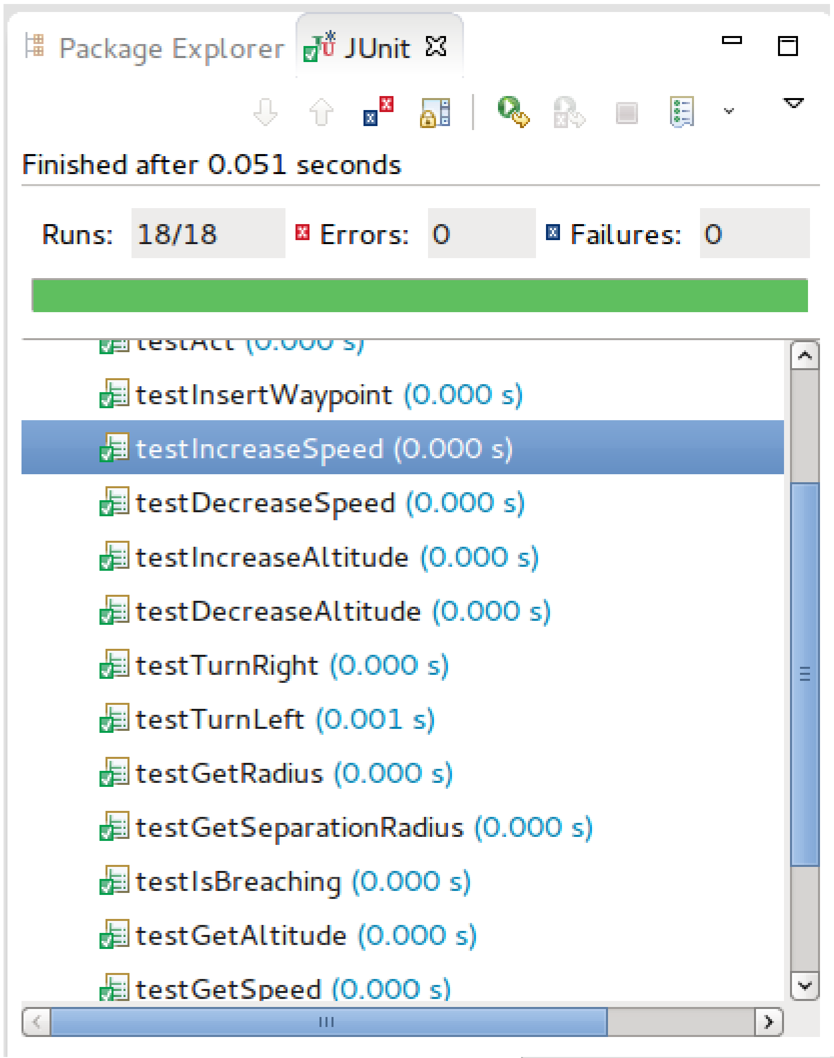


```
package seprini.controllers;

import static org.junit.Assert.*;

public class LeaderboardTest {

    @Test
    public void testAddLeaderBoardEntries() {
        LeaderboardEntry[] testLeaderboardEntries1 = new LeaderboardEntry[5];
        Leaderboard testLB = new Leaderboard();

        for (int i = 0; i < testLeaderboardEntries1.length; i++) {
            testLeaderboardEntries1[i] = new LeaderboardEntry();
        }

            testLeaderboardEntries1[0].setName("d");
            testLeaderboardEntries1[0].setScore(300);
            testLeaderboardEntries1[1].setName("d");
            testLeaderboardEntries1[1].setScore(300);
            testLeaderboardEntries1[2].setName("d");
            testLeaderboardEntries1[2].setScore(300);
            testLeaderboardEntries1[3].setName("d");
            testLeaderboardEntries1[3].setScore(300);
            testLeaderboardEntries1[4].setName("d");
            testLeaderboardEntries1[4].setScore(300);

            testLB.addLeaderBoardEntries();

            assertEquals("Tests that the array that is created matches the array that I created", testLeaderboardEntries1[0].getName(),
            assertEquals("Tests that the array that is created matches the array that I created", testLeaderboardEntries1[0].getScore(),
```

Fig. 5: Test 1.1.2 Leaderboard addLeaderBoardEntries() addLeaderboardEntry() sortLeaderboard() and JUnit



```
Finished after 0.164 seconds

Runs: 1/1    Errors: 0    Failures: 0

seprini.controllers.LeaderboardTest [Runner: JUnit 4]
    testAddLeaderBoardEntries (0.000 s)
```

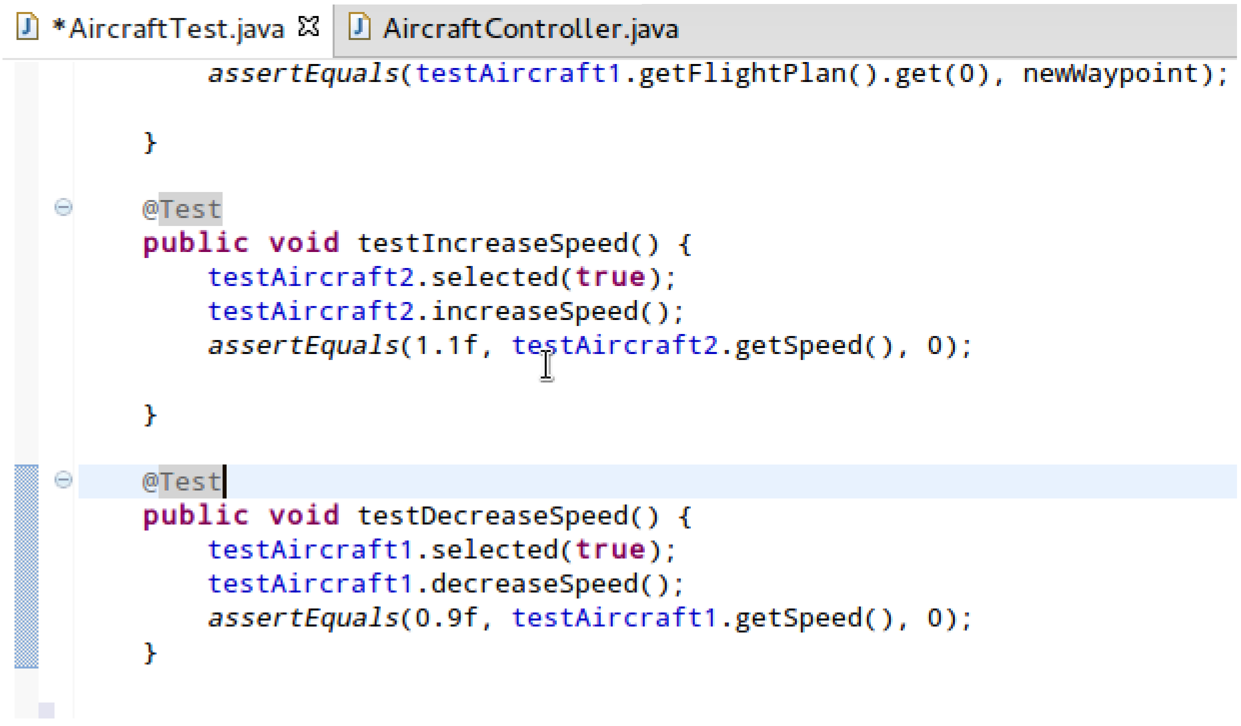


Finished after 0.051 seconds

Runs: 18/18 Errors: 0 Failures: 0

```
testAct (0.000 s)
testInsertWaypoint (0.000 s)
testIncreaseSpeed (0.000 s)
testDecreaseSpeed (0.000 s)
testIncreaseAltitude (0.000 s)
testDecreaseAltitude (0.000 s)
testTurnRight (0.000 s)
testTurnLeft (0.001 s)
testGetRadius (0.000 s)
testGetSeparationRadius (0.000 s)
testIsBreaching (0.000 s)
testGetAltitude (0.000 s)
testGetSpeed (0.000 s)
```

Fig. 6: Test 2.1.1 Aircraft increaseSpeed(), decreaseSpeed()



```
        assertEquals(testAircraft1.getFlightPlan().get(0), newWaypoint);

    }

    @Test
    public void testIncreaseSpeed() {
        testAircraft2.selected(true);
        testAircraft2.increaseSpeed();
        assertEquals(1.1f, testAircraft2.getSpeed(), 0);

    }

    @Test
    public void testDecreaseSpeed() {
        testAircraft1.selected(true);
        testAircraft1.decreaseSpeed();
        assertEquals(0.9f, testAircraft1.getSpeed(), 0);
    }
```

APPENDIX

Use Cases

Use Case 1

“Change heading of an aeroplane”

1. Primary Actor: Player
2. Supporting Actors: Computer
3. Precondition:
    3.1. Player is playing the game.
    3.2. At least one aeroplane is in the airspace.
    3.3. At least one waypoint is remaining for the aeroplane.
4. Trigger: The player clicks on the aeroplane.
5. Scenario:
    5.1. Arrow appears pointing from aeroplane to mouse pointer. The heading and altitude of the aeroplane is shown.
    5.2. Player moves the mouse until it points in the desired direction and then lets go of the mouse.
    5.3. The system saves the heading selected by the user. The system displays the altitude editor.
    5.4. The player adjusts target altitude using either the scroll wheel, altitude bar, or by typing in an altitude.
    5.5. The system indicates that the operation has completed.
6. Post-condition: The heading and altitude are set as targets for the aeroplane. The aeroplane begins changing direction.

Use Case 2

“Player checks the leaderboard”

1. Primary Actor: Player
2. Supporting Actors: Computer
3. Precondition:
    3.1. Player has a computer.
    3.2. Player has the ATC game installed.
4. Trigger: Player loads up the game.
5. Main success scenario:
    5.1. The player accesses main screen.
    5.2. The leaderboard is shown.
6. Post-condition: Player gets the information they wanted from the leaderboard.

Use Case 3

“Player quits the game”

1. Primary Actor: Player
2. Supporting Actors: Computer
3. Precondition:
    3.1. The game is being played.
4. Trigger: Player presses the quit button.
5. Main Success Scenario:
    5.1. Game Over screen is shown.
    5.2. Game exits.
6. Post-condition: The game is no longer running.

Use Case 4

“User crashes a plane”
1. Primary Actor: Player
2. Supporting Actors: Computer
3. Precondition:
    3.1. The game is being played.
    3.2. There are aeroplanes under the user’s control.
4. Trigger: Aeroplanes collide.
5. Scenario:
    5.1. User sees an explosion at the position of the plane.
    5.2. User is shown game over screen.
6. Post-condition: Main menu is shown and the game has stopped.

Use Case 5

“User wants to land/exit a plane”
1. Primary Actor: Player
2. Supporting Actors: Computer
3. Precondition:
    3.1. The game is being played
    3.2. There are aeroplanes in the airspace
4. Trigger: Plane flies through final waypoint.
5. Main success scenario:
    5.1. The player manoeuvres the plane so that it flies through its exit point.
    5.2. Plane is removed.
6. Post-condition: Player notices score increment after the plane has exited through its exit point.

Requirements

(GUI) (1) User shall see: (1.1)	**User Requirements**	**How to meet User Requirements?**

1.1.1	Airplane entry and exit points on the edges of the game screen that planes will arrive at and leave from on their flight path.	Place entry and exit points at specific places on the screen edge. Planes will then spawn and exit at these points.
1.1.2	A GUI showing an airspace (including waypoints, entry/exit points and planes).	When the player starts a game the GUI will be rendered on-screen with all waypoints, entry and exit points in place. Planes will then spawn on-screen.
1.1.3	Feedback on which plane is currently selected.	When selected a red circle will appear around a plane including the plane flight path as a red line to an exit point.
1.1.4 (Ass.3)	Their score based on: 1.1.4.1: Passing through way points and exit points. 1.1.4.2: How long the player has been playing (proportionally to time in seconds). 1.1.4.3: Separation violations 1.1.4.4 How many planes are in the Airport	1.1.4.1: Will include a score incrementer that when a plane has been detected as leaving the airspace the user will gain points. Bonus points will be awarded if the plane passes through its designated way points. Points will be deducted if plane leaves the screen boundaries not through its exit point 1.1.4.2: A timer has been implemented and a score multiplier will allow time based scores to increment based on these values. 1.1.4.3: For each second two or more planes are violating their separation rules the player will lose a set amount of points. 1.1.4.4 Player will lose certain amount of points per second based on the number of planes docked at the airport.
1.1.5 (Ass.3)	Their final score entered into the leaderboard if they qualify to be placed on the leaderboard with a high enough score by entering their name after a game ends (not if they voluntarily exit).	Compare leaderboard scores with the score just received; if the new score is high enough then prompt the player for their score and place the score in the leaderboard file and return to the ‘Main Menu’. The score will not be asked for if the user ends the game voluntarily.
1.1.6 (Ass.3)	A selected Plane’s flight plan projection indicated by a red line path through all relevant waypoints and an exit point.	Edit the code that shows the path to the currently selected planes exit point to also point to the waypoints the plane travels through before it leaves through its exit point.
1.1.7	The “Game Over” screen appears when the player loses.	After a plane has crashed the ‘Game Over’ screen will appear showing score and time played with the possibility of asking for a leaderboard entry.
1.1.8	Planes with different characteristics including speed and colour.	Alter code to allow planes to have different base speeds e.g. faster planes will have a faster base speed and slower planes will have slower base speed and replicate plane images with different hues and colours.
1.1.9	Red circles around the planes where the separation rules are being violated or if the planes are close together.	The violating planes will have a red circle appear around them showing the separation violation for the difficulty. These circles will appear when planes are calculated to be close enough to each other.
User should: (1.2.1)	**User Requirements**	**How to meet User Requirements?**

1.2.1	See the speed and altitude of a plane in the 'Side Menu' and altitude on the airspace GUI.	Selected planes will have a calculated speed and altitude which will be shown in the 'Side Menu' GUI. Altitude will also be shown near to each plane on the GUI.
1.2.2	Be able to see a 'Main Menu' screen to start the game and after exiting the 'Game Over' screen.	After initialising the 'Main Menu' will appear and after the 'Game Over' screen or exiting a game voluntarily the 'Main Menu' will also be shown.
1.2.3 (Ass.3)	Be able to view a leaderboard on the 'Main Menu'.	Leaderboard will be displayed onto the 'Main Menu'.
1.2.4 (Ass.3)	See a rogue plane that cannot be controlled.	Create a different sprite for the rogue plane and prevent the player from controlling the plane. The plane must have a set flight path.
(Interaction) (2) User shall be able to: (2.1)	**User Requirements**	**How to meet User Requirements?**
2.1.1 (Ass.3)	Command a plane to land at the airport via a 'landing' button on the 'Side Menu'.	Place a 'Land' button on the 'Side Menu' which sends a plane to the Airport when pressed.
2.1.2	Alter a plane's altitude.	Allow player to press the 'W' and 'S' or '↑' and '↓' keyboard buttons and use the 'Up' and 'Down' buttons on the 'Side Menu' to alter the altitude.
2.1.3	Change a plane's direction by assigning new waypoints.	Press the 'Assign Waypoint' then click on a specific waypoint while a plane is selected. This will redirect a plane. A selected plane will move when pressing certain arrow keys in the specified direction.
2.1.4	Exit game whenever they want by pressing the 'Main Menu' button in-game.	Include a 'Main Menu' button that will return the user to the 'Main Menu' and end the current game.
2.1.5	Change plane's direction manually.	Allow player to press 'A' and 'D' or '←' and '→' keyboard buttons or 'Left' and 'Right' side menu buttons to change plane's direction and turn off autopilot.
2.1.6	Create new waypoints for the planes.	Allow player to create new waypoints in the airspace by pressing the 'Create waypoint' button on side menu and then pressing left mouse button on the desired location.
2.1.7	Delete user-created waypoints.	Allow player to delete waypoints that were created by pressing the right mouse on the desired waypoint.
User should be able to: (2.2)	**User Requirements**	**How to meet the User Requirements?**
2.2.1 (Ass.3)	Play the game with different skins.	The GUI file will be copied and changed to include different skins e.g. Space and underwater theme that the user can choose between on the 'Main Menu'. These reskins will not change the gameplay only the appearance.
2.2.2	Change speed of planes	Pressing 'Q', 'E', 'Accelerate' or 'Decelerate' will allow the user to change a planes speed to a maximum and

		minimum value (specific to each plane).
Non-functional (3)	**User Requirements**	**How to meet the User Requirements?**
3.1	Game should be easy to learn.	The user should be fluent with the use of the controls within two games with the game manual provided.
3.2	The game must be enjoyable and leaderboard scoring should be addictive.	The user must be happy having played the game and enjoyed the competitive aspect (leaderboard scoring).
3.3	View an aesthetically pleasing GUI.	‘Main Menu’ includes new buttons to change between old ‘Earth’, Space and Ocean skin. New skins designed to be fun and easy on the eye. ‘Side Menu’ redesigned to use gentler colours.
(GUI) (1) System shall: (1.1)	**System Requirements**	**How to meet System Requirements?**
1.1.1	Move planes using translation animations.	Use small increments of movement to simulate smooth transitions and continuous movement.
1.1.2	Have a backdrop GUI image for the airspace.	Load an image from a file to be displayed on the canvas.
1.1.3 (Ass.3)	Place an invisible airport entry/exit point.	An invisible entry/exit point will be placed at specific coordinates that will cause planes to be removed when the ‘Land’ button is pressed and take off when the ‘Take off’ button is pressed.
1.1.4	Show at least 10 intermediate waypoints.	Replicate at least 10 waypoints images from a file on the canvas on screen using coordinates.
1.1.5	Display at least 3 entry points and 3 exit points.	Load and display the entry/exit points images from a file and display them on the canvas GUI at specific coordinates around the edge of screen.
1.1.6	Have an interface to adjust plane’s altitude.	Map the ‘W’, ‘S’, ‘↑’,‘↓ or ‘Increase Altitude’ and ‘Decrease Altitude’ to adjust the planes altitude.
1.1.7	Display “Game Over” screen when user loses a game.	After a plane crashes the game screen is removed and a ‘Game Over’ screen is displayed.
1.1.8 (Ass.3)	Game should allow at least 10 flights on-screen.	The number of planes generated can be manipulated by changing the integer variables for each difficulty where hard mode will spawn more planes (with 10 being the minimum).
1.1.9 (Ass.3)	Planes must be able to land and take off from the airport.	When the ‘Land’ button is pressed, the selected plane will travel to the airport where it will detect a collision and be removed from the screen and stored in the airport. A maximum number of planes can be stored and are given a new flight plan when they take off and return to the

		game. Excess planes will reach the airport and continue on their flight path.
1.1.10 (Ass.3)	Limit the number of planes allowed to land and take off from an airport simultaneously.	A maximum of 3 planes can land simultaneously at the airport. Only one plane can take off every 2 seconds to prevent them from colliding with each other if the player pressed the ‘Take Off’ button in rapid succession.
System should: (1.2)	**System Requirements**	**How to meet System Requirements?**
1.2.1	Display plane’s flight plan and direction.	Generate a red line path to show where the planes waypoint path is, ending with the exit point.
1.2.2 (Ass.3)	Display the leaderboard on the ‘Main Menu’ at startup.	When the game is initialised the ‘Main Menu’ will be displayed with graphics and sound loaded.
1.2.3	Place waypoints on the GUI at the start of gameplay.	Use static waypoint placements before the planes start spawning.
1.2.4	Change speed of planes	Map ‘Q’, ‘E’ buttons and use ‘Accelerate’ and ‘Decelerate’ sidebar buttons to increase/decrease the speed by a set amount.
1.2.5 (Ass.3)	Allow clouds to pass by on ‘Earth’ mode skin.	Clouds will be designed in the GUI file and move like planes across the screen but won’t interact with planes or the environment. The clouds will be semi-transparent so players can see through the clouds to prevent obscured views.
(Mathematical) (2) System shall: (2.1)	**System Requirements**	**How to meet System Requirements?**
2.1.1	Monitor plane separation rules.	The game will check for the difference in altitude and the horizontal 2D space plane and check whether the planes have violated the separation rules returning a pair of the planes that have violated the rules to be displayed on screen using a separation circle.
2.1.2	End the game if two or more planes crash.	If the planes are close enough in altitude and horizontal 2D space then they will be declared as crashed and the ‘Game Over’ screen will appear.
2.1.3 (Ass.3)	Track and calculate score based on: 2.1.3.1: If the plane passes through its way points and exit point. 2.1.3.2: How long the player has been playing (time in seconds). 2.1.3.3: Whether two or more planes are violating their separation rules. 2.1.3.4 For every second plane is docked at the airport	2.1.3.1: When a plane exits through its designated exit point the player will receive a scaled score increase. If the plane passes through its way points the player will receive bonus points. 2.1.3.2: The score is incremented with a multiplier for each second the game has been played. 2.1.3.3: Decrement the score by a set amount for each second one or more planes are violating their separation rules (decrementing stacks per plane violating another plane). 2.1.3.4 Decrement the score every second by a set

		amount based on the number of planes currently docked in the airport.
2.1.4	Calculate updated positions for each plane regularly based on velocities and angles.	The coordinates of the plane will be updated based on the velocity and the current rotation of the plane. The plane will move to specific waypoints at specific angles at the velocity it currently has.
2.1.5	Specify a static bearing (ignoring the flight plan) for a selected plane.	When a plane is selected a red line path will be drawn to show its flight path which can be changed by taking manual control of the plane and turning it left or right or assigning a new way point.
System should: (2.2)	**System Requirements**	**How to meet System Requirements?**
2.2.1	Calculate realistic movements in response to changes in flight path.	When a player changes the flight path of a plane manually the plane should move smoothly (and in a curve if necessary) in the direction of the new waypoint/exit point designated.
Non-functional (3)	**Non-functional Requirements**	**How to meet Non-functional Requirements?**
3.1	GUI should appear to be real-time.	GUI must be updated at least 30 times per second (30fps).
3.2	Collidable objects (planes) shall not pass through each other without consequences.	Check for breaches of the separation rules every frame (30fps).
3.3	Consistent Design.	Similar colour scheme is used across all skins for the ‘Main Menu’ and design of the game title. Reskinned vehicles use different colours to be consistent between versions and contrast with background GUI’s.
3.4	The game has to load quickly.	User can play game in less than 5 seconds from loading.
3.5	Game must be cross-platform and run on multiple Operating Systems.	The ATC game must run on Windows, Linux and OS X.