# 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 | Succes |
|---|---|---|---|---|---|---|---|
| 1.1.1 | Unit | State | Reset() | A JUnit test was created that reset the scores and timer | David | < 10 ms | The sc be equa |
| 1.1.2 | Unit | EndScreen | EndScreen() | The code was read through by two members of the team to check that it functioned as expected | Dan | - | The res beginni those p through |
| 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 val score t what is change |
| 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 wa should produc |
| 1.1.5 | Unit | AircraftController | AircraftController() | The code was read through by two team members | David | < 10 ms | The ga handle spawne perform |
| 1.1.6 | Unit | Airport | | A JUnit test was written that creates 4 planes they are then instructed to land. | Jamaal | < 10 ms | The ga 3 plane new flig that is r |
| 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 val array sl those in |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.3.1 | Functional | State, SidebarController | incScore(), update() | The game was run and hard difficulty was selected | Sam | < 5000 ms | The sid game s being a every s added f succes |
| 1.3.2 | Functional | AircraftController, AircraftType | AircraftController(), AircraftType() | The game was played allowing for many planes to be spawned | Paul | < 5000 ms | The ga while w plane t recorde compar snake p screen. around |
| 1.3.4 | Functional | Waypoint | RandomFlightPaths | Two planes were spawned from the same entry point | Jamaal | < 5000 ms | The pla then co that the differen |
| 1.3.5 | Functional | State | Reset() | The fastest and slowest planes in the game were spawned | Paul | < 5000 ms | The pla change stay in should |
| 1.3.6 | Functional | Cloud | Cloud() | The game was run to check that clouds had been implemented correctly | Dan | < 5000 ms | The clo move fr screen interfer the gam |
| 1.3.7 | Functional | Map | Map() | The map was loaded from the textures hash table. | Sam | < 5000 ms | A map theme displaye all plan points. |
| 1.3.8 | Functional | Aircraft | Aircraft() | The game was loaded and an aircraft was spawned. | David | < 5000 ms | The air the use path th the way |
| 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 ad should the plan point fo |
| 1.3.10 | Functional | MenuControll | addLeaderboard() | The game was loaded | Paul | < | A leade |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | er | | | | 5000 ms | visible |
| 1.3.11 | Functional | SidebarController | changed() | A non-snake plane was spawned and the land button was pressed | Dan | < 5000 ms | The sel head to disappe number airport increme |
| 1.3.12 | Functional | SidebarController | changed() | A non-snake plane was spawned and the up button was pressed | Sam | < 5000 ms | The alti plane s current 15000 |
| 1.3.13 | Functional | MenuControlller | changed() | The game was played three times each time selecting a new theme | Dan | < 5000 ms | Each ti selecte load wi textures |

***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?** | **Teste r/s** | **Est. run time** | **Succe** |
|---|---|---|---|---|---|---|---|
| 2.1.1 | Unit | 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 pla accele in spee was de decrea<br><br>Written engine modifie code. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2.1.2 | Unit | Aircraft | act() | A pair of testers read through the code to make sure that there were no errors present. | Pauliu s, Dan | | The pr match results |
| 2.2.1 | Functional | Entrypoint | Entrypoint() | Four entry points were created and then drawn. | Jamaa l | < 5000 ms | The en created screen locatio |
| 2.2.2 | Functional | Exitpoint | Exitpoint() | Four exit points were instantiated and then printed to the screen in different locations. | Jamaa l | < 5000 ms | The ex visible locatio passed |
| 2.2.3 | Functional | EndScreen | EndScreen() | Two planes were instantiated and then intentionally collided | Pauliu s | < 5000 ms | The ga and the display |
| 2.2.4 | Functional | AircraftContr oller, Aircraft | update(), separationRulesBreached () | Two planes were created, one was selected and then directed towards the other plane. | Sam | < 5000 ms | The pla should appear are ste other a visible planes |
| 2.2.5 | Functional | SidebarContr oller | update() | A plane was created and then selected. | Paul | < 5000 ms | The sp sideba the spe plane. |
| 2.2.6 | Functional | MenuControll er, MenuScreen | menuController(), render() | The game was loaded | Dan | < 5000 ms | The firs to the u menu s |
| 2.2.7 | Functional | MenuControll er, MenuScreen | menuController(), render() | The game was loaded and a game was played | David | < 5000 ms | After th the me appear |
| 2.2.8 | Functional | SidebarContr oller | changed() | A non-snake plane was spawned and the down button was pressed | David | < 5000 ms | The ins is pres planes decrea altitude |
| 2.2.9 | Functional | SidebarContr oller | changed() | A plane was spawned and selected, the assign waypoint was then | Sam | < 5000 ms | The pla selecte of its w |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | pressed and a waypoint was clicked | | | head s |
| 2.2.10 | Functional | SidebarController | changed() | The game was loaded and played. The Main Menu button was then pressed | Pauliu s | < 5000 ms | The ga main m button |
| 2.2.11 | Functional | Waypoint | Waypoint() | Ten waypoints were instantiated at static locations | Dan | < 5000 ms | The wa created correct |
| 2.2.12 | Functional | GameScreen | update(), render() | The game was loaded and a fps counter was drawn on screen. | Jamaa l | < 5000 ms | The fps should most o |
| 2.2.13 | Functional | - | - | The game was loaded and played on 3 systems installed with Mac, Linux and windows OS | Dan | < 5000 ms | The ga platfor in perf |
| 2.2.14 | Functional | - | - | The games texture files were opened | Paul | < 5000 ms | The tex consist with th waypoi also be |
| 2.2.15 | Functional | - | - | The game was played. | Pauliu s | < 5000 ms | The tim game t game t should seconc |

***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.

Fig. 1: Case 4 Pre-condition


***Testing Examples***

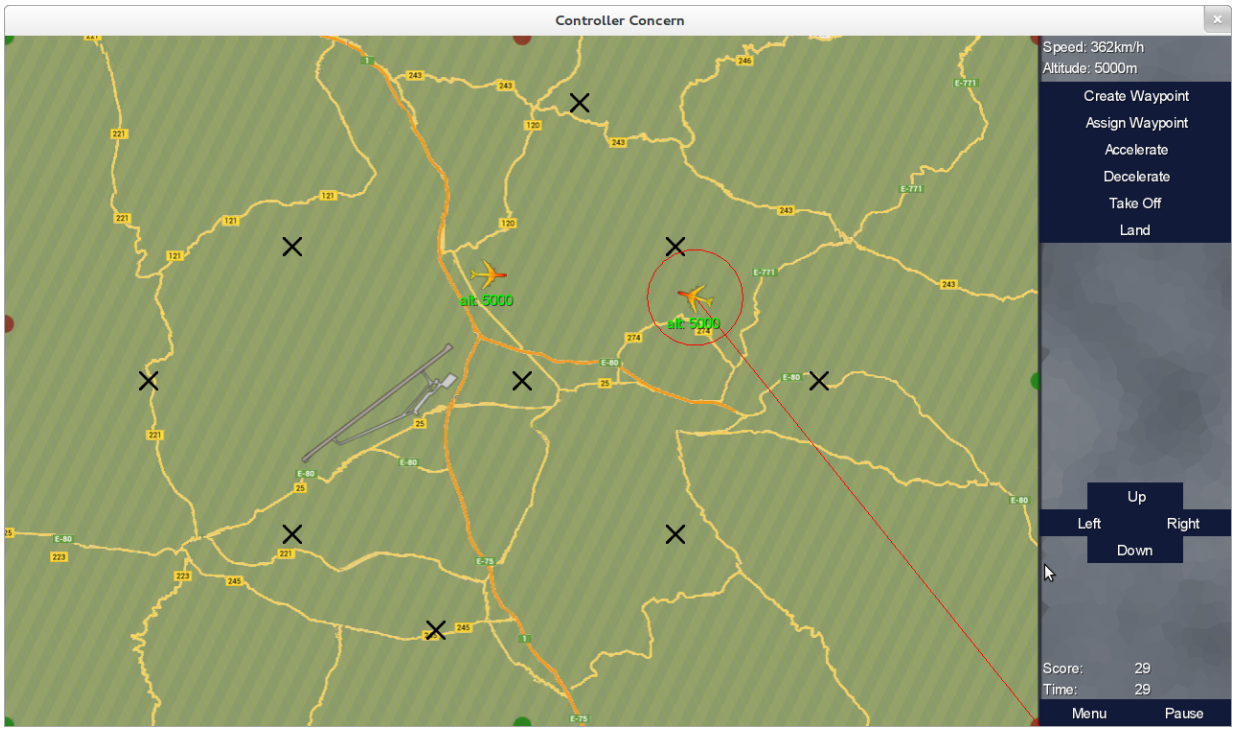
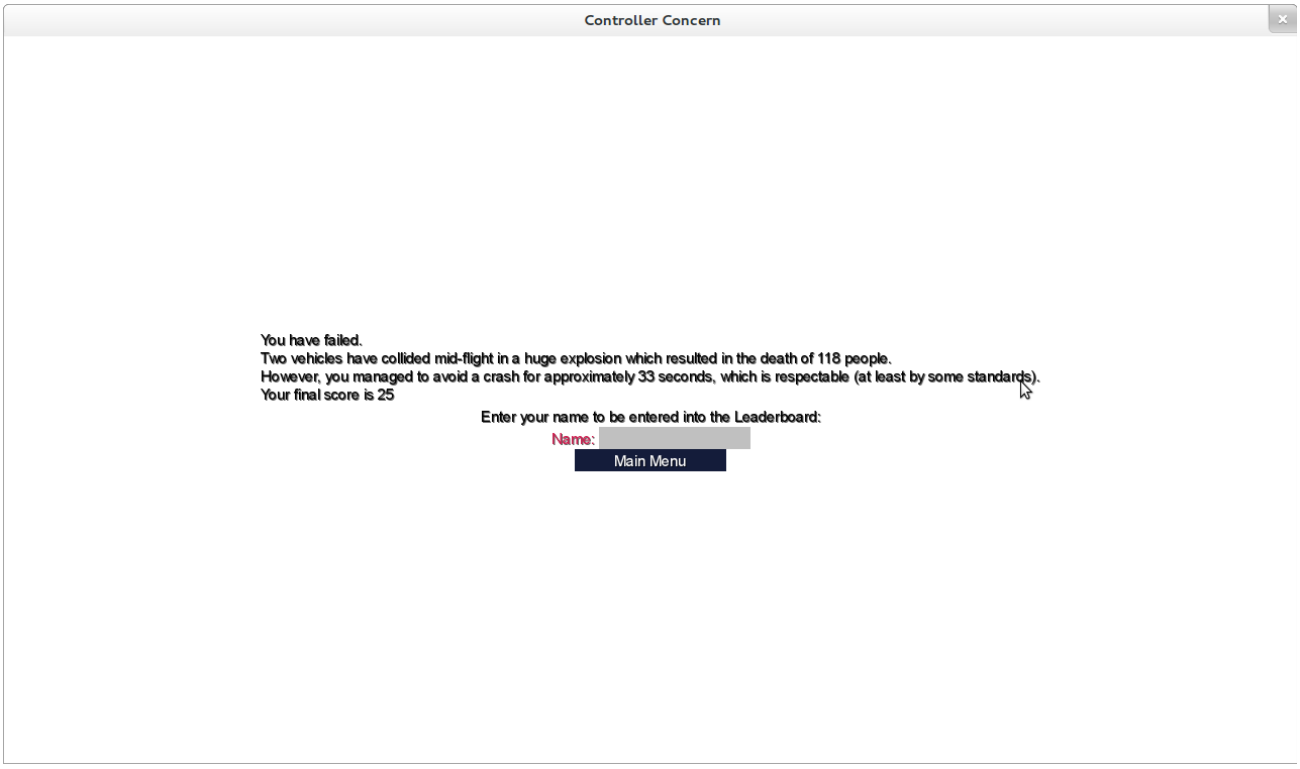


Fig. 2: Case 4 Scenario

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



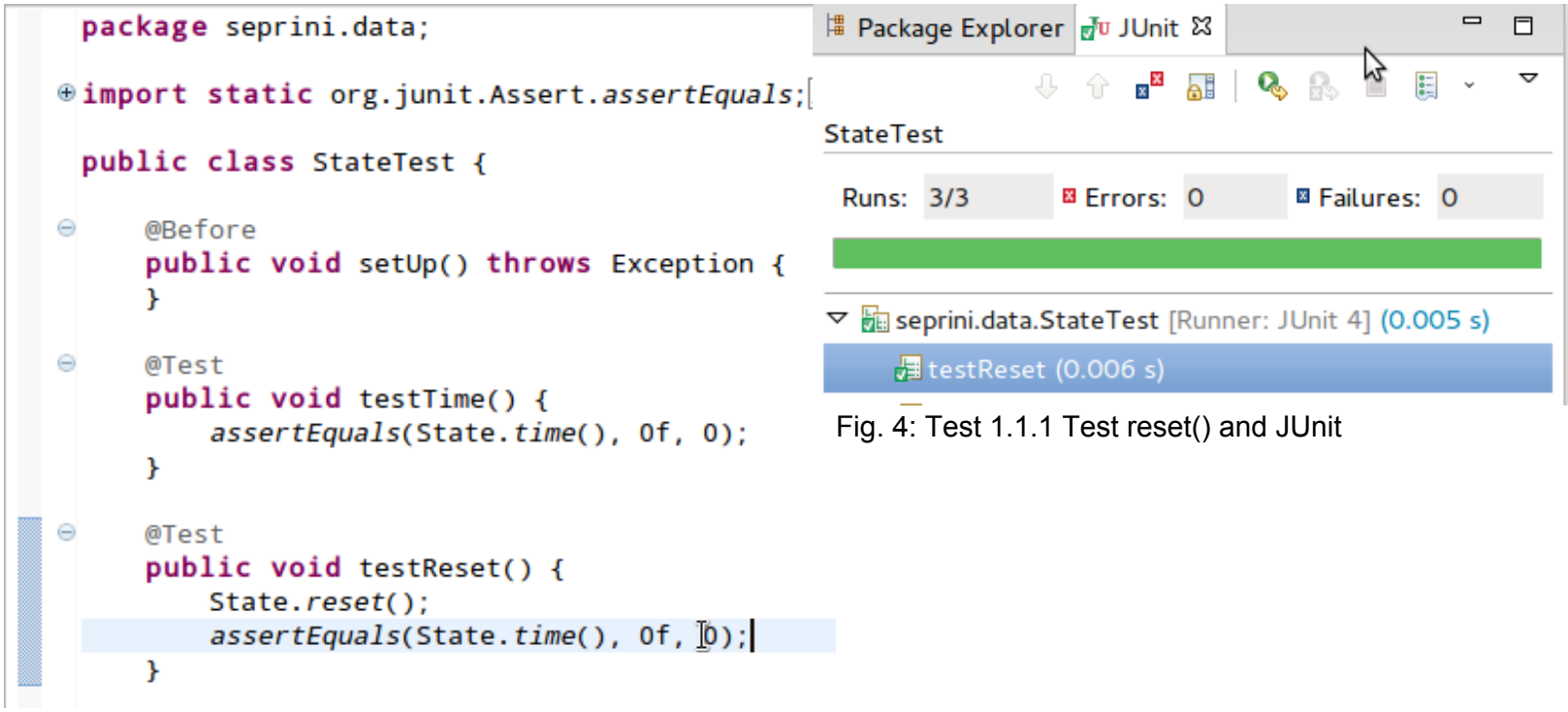




Fig. 4: Test 1.1.1 Test reset() and JUnit

```java
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(),
```

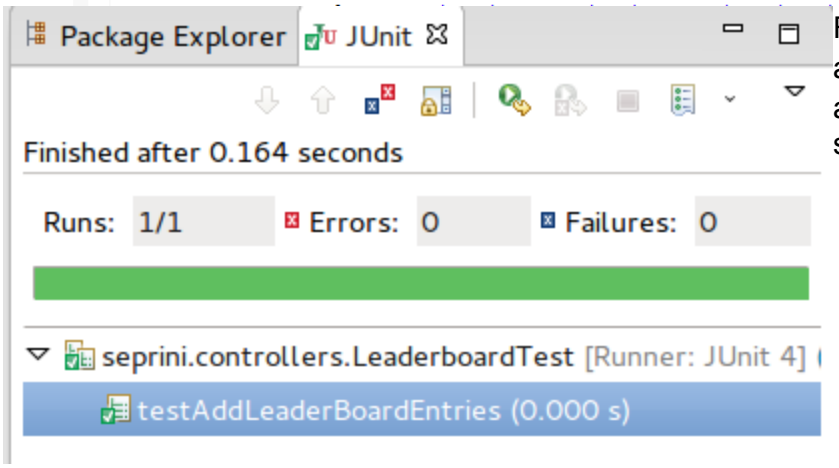



Fig. 5: Test 1.1.2 Leaderboard addLeaderBoardEntries() addLeaderboardEntry() sortLeaderboard() and JUnit

*AircraftTest.java  AircraftController.java

```
        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);
    }
```

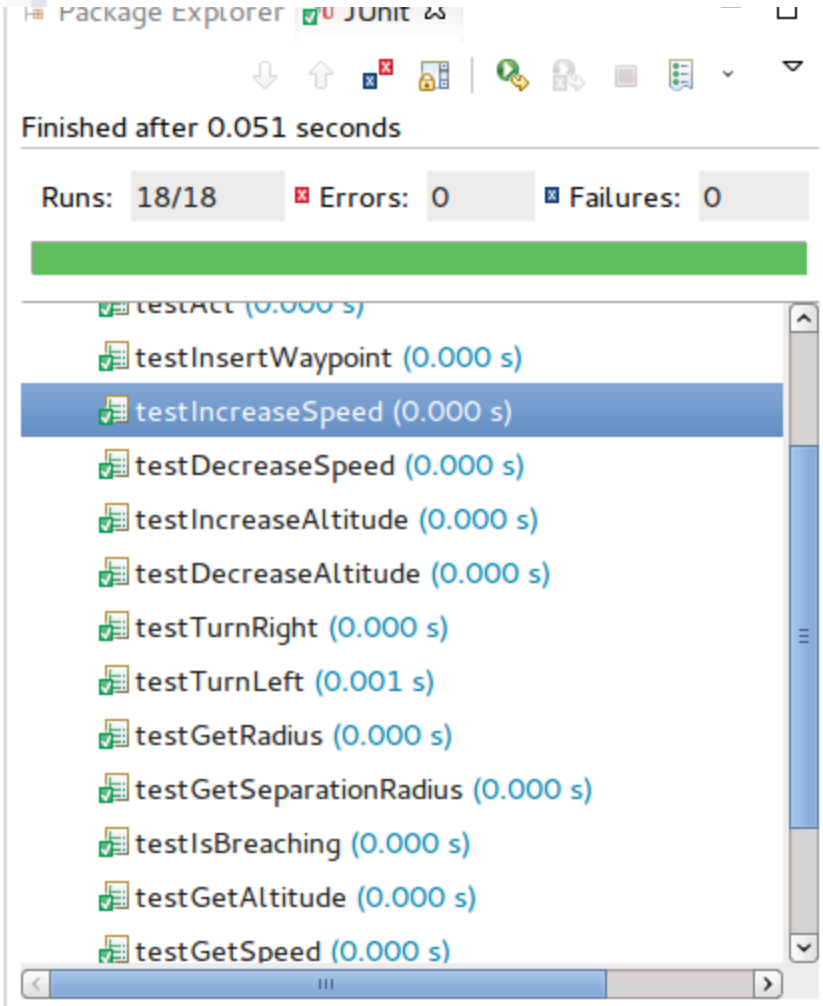



Fig. 6: Test 2.1.1 Aircraft increaseSpeed(), decreaseSpeed()

# 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 maneuvers 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.