PPA Assignment 12

Overview

Do you think there is a problem with any of the content below? Let us know immediately at programming@kcl.ac.uk.

Read through this brief carefully before starting your attempted solution. Also ensure that you comment your code.

You are not advised to print this assignment, but instead to always access the latest version of this brief through KEATS, which will evolve with minor clarifications and corrections throughout the assessment period. Students will be notified of any major modifications to the brief by email.

A partner from your lab session should be selected for this piece of work at the next available opportunity, typically your next lab session. You must not complete any of the assignment below without your chosen partner present, as doing so is likely to jeopardise your grade.

You must select a new partner for this piece of work, to ensure that there is no impact on your mark.

Aims

The aims of this piece of coursework are as follows:

Once completed, both you and your partner must submit your assignment using the link marked `Assignment 12: Nexus Submission Link' on KEATS. However, this is not enough to receive a mark for this piece of work. You must also attend the lab session following your submission, so that one of the teaching assistants can mark your work with you present, and ask you detailed questions about it. Revisit the `Lab Assessment and Pair Programming Q&A' guide on KEATS for more information.

Any submitted code or documentation that is found to be unduly similar to the code or documentation submitted by any other pair(s) of students, will result in a penalty for those involved.

Provisional marks for your code will be released on KEATS within one week of the final lab assessment. Final assignment grades will be submitted to the exam board at the end of the semester.

For all other queries, see the Support section on KEATS, specifically the `Lab

- To explore the roles dfferent components can have in a GUI;
- To explore the use of GUI layouts;
- To employ simple event handlers;
- To understand how to layout and structure GUI code;
- To solve basic problems in Java.

As this assignment is designed to test the knowledge acquired over a two week period, you may not be able to complete all of this assignment until after the PPA lecture following its release, where further language concepts will be presented to you.

Task Overview

In this piece of work, we are going to build a very simple Flight Simulator, the graphical component of which will be constructed using the basic UI components offered to us by Java Swing:

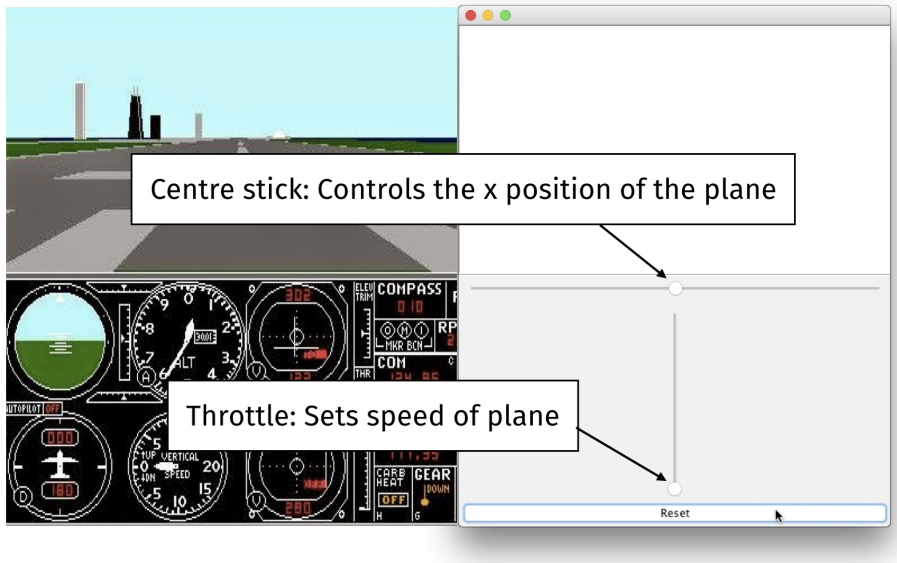
Assessment and Pair Programming Q&A' guide.

As you can see from the left image above, a typical flight simulator places the player at the viewpoint of a pilot in a plane. Therefore, the screen is separated into two parts: the top of the screen shows the view the pilot has out of the window of the cockpit, facing forward, and the bottom of the screen shows a view of the plane controls in the cockpit, which the pilot (the player) can interact with.

As can also be seen from the image above, the plane starts on a runway, facing forward. In this assignment, we will only consider the phase of the flight simulation in which the plane takes off.

The right image above shows how we will replicate the flight simulator UI to the left in Java. Rather than displaying a graphic of the world outside the plane, our UI will simply describe this world using a series of printed statements. Details of what these statements are to say will follow.

In addition, rather than a comprehensive set of controls, as shown in the image on the left, we will only consider two main controls, as shown in an image on the right, designed to set the x position and the speed of the plane, as shown below:



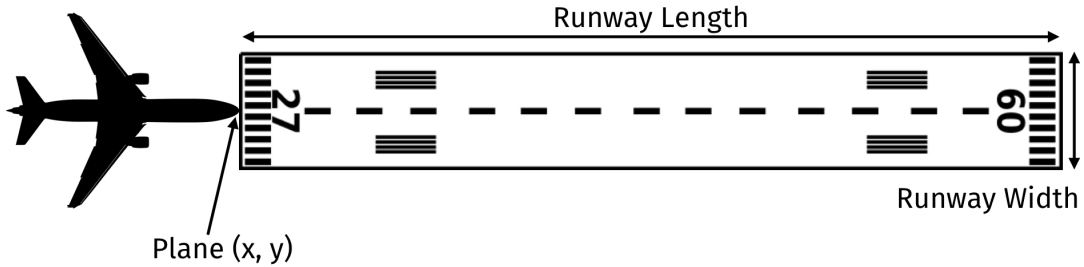
We will also consider a reset button, that restarts the simulation.

Simulation Timeline

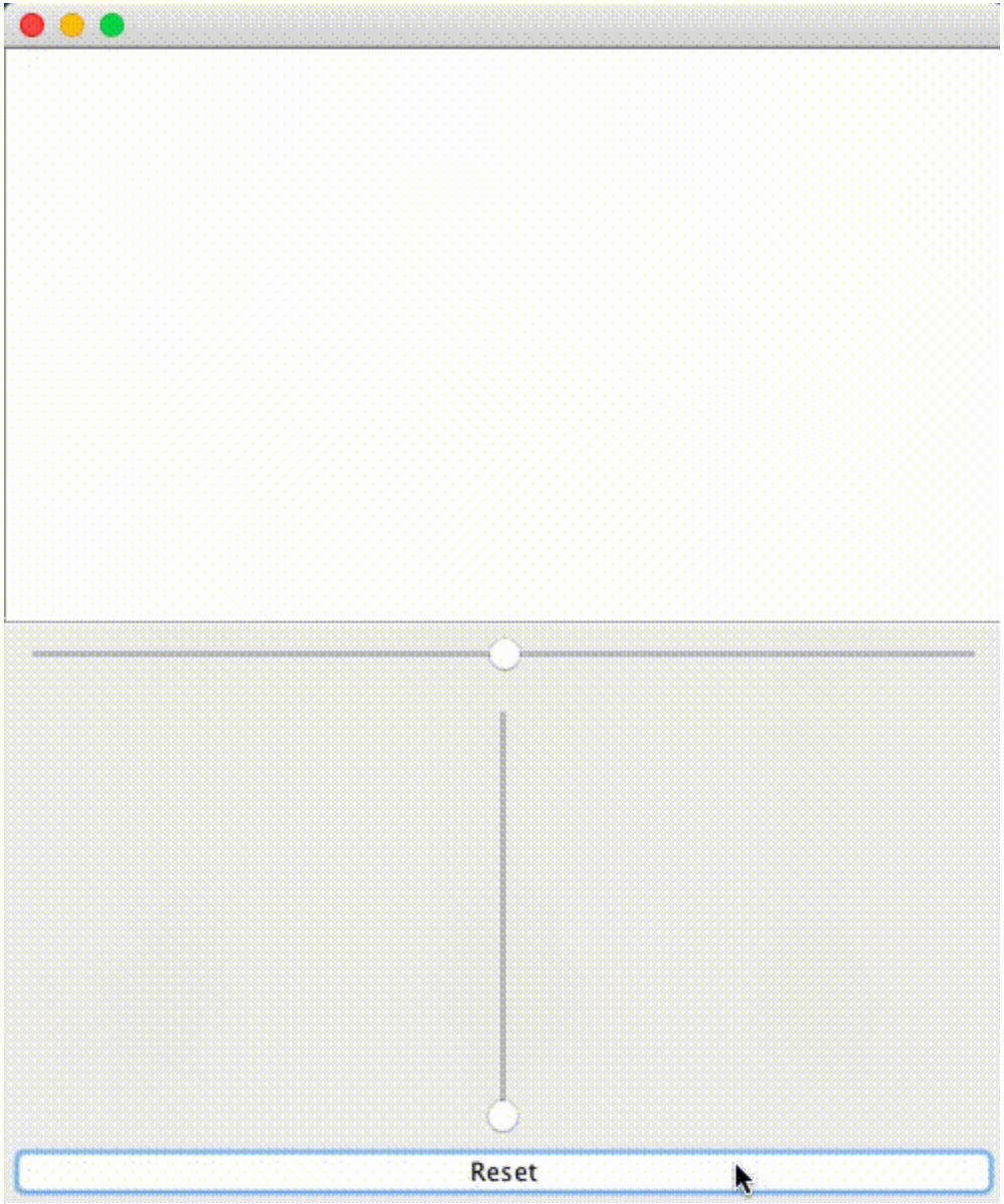
When the program starts, the simulation should start.

The passage of time in the simulation is quantified by updating the state of the aircraft every second, based upon the current state of the aircraft controls. Details of the current state of the aircraft should be output, at each timestamp, to the UI. This information should include the current time, the x and y position of the plane, the plane's speed and its elevation (i.e. how high it is in the air).

At the start of the simulation, the aeroplane has not yet taken off, so it is at the start of the runway, in the centre. For formality, we will measure the position of the plane from the nose, but this should not have a significant impact on how we construct our simulator:



So, if the user does not interact with the controls during the simulation, time passes with the airplane staying in this position at the start of the runway. This, and the general passage of time, can be seen below:



The aim of the player is to make the plane take off. In order for the plane to take off, it must have exceeded a certain elevation by the end of the runway, and must remain in the centre of the runway. If it does not exceed this elevation

by the end of the runway, or deviates from its central position, the take off should fail. Planes may take off before the end of the runway.

In order to lift off from the ground (i.e. gain elevation), and climb towards a given elevation, the plane must be going at a certain speed for a given number of seconds. When the plane reaches this speed, and has been travelling at this speed for the given number of seconds, then the plane's elevation increases by 1 for each second that passes.

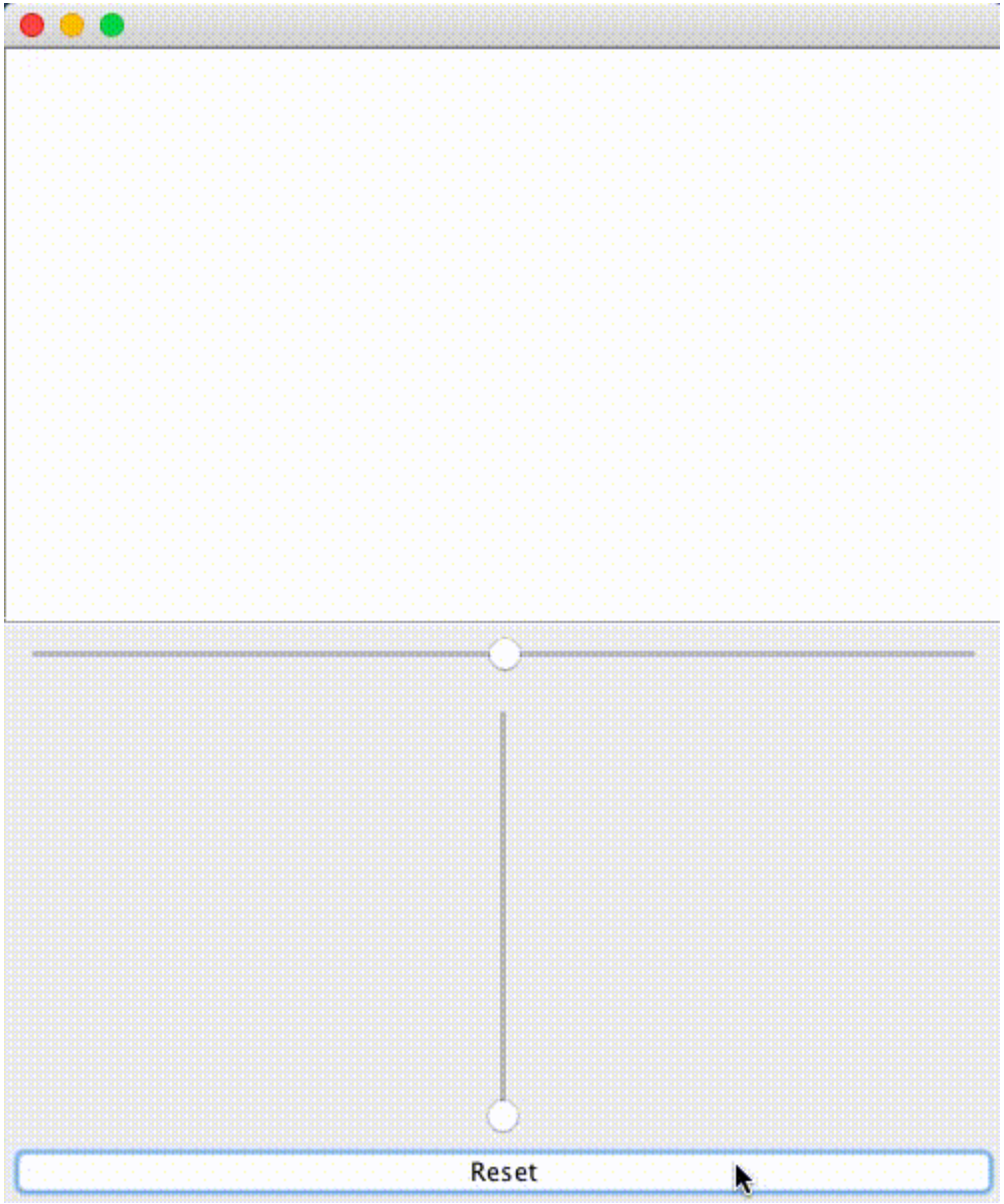
Simulation Parameters

We will assume the following about the state of the world for the flight simulation:

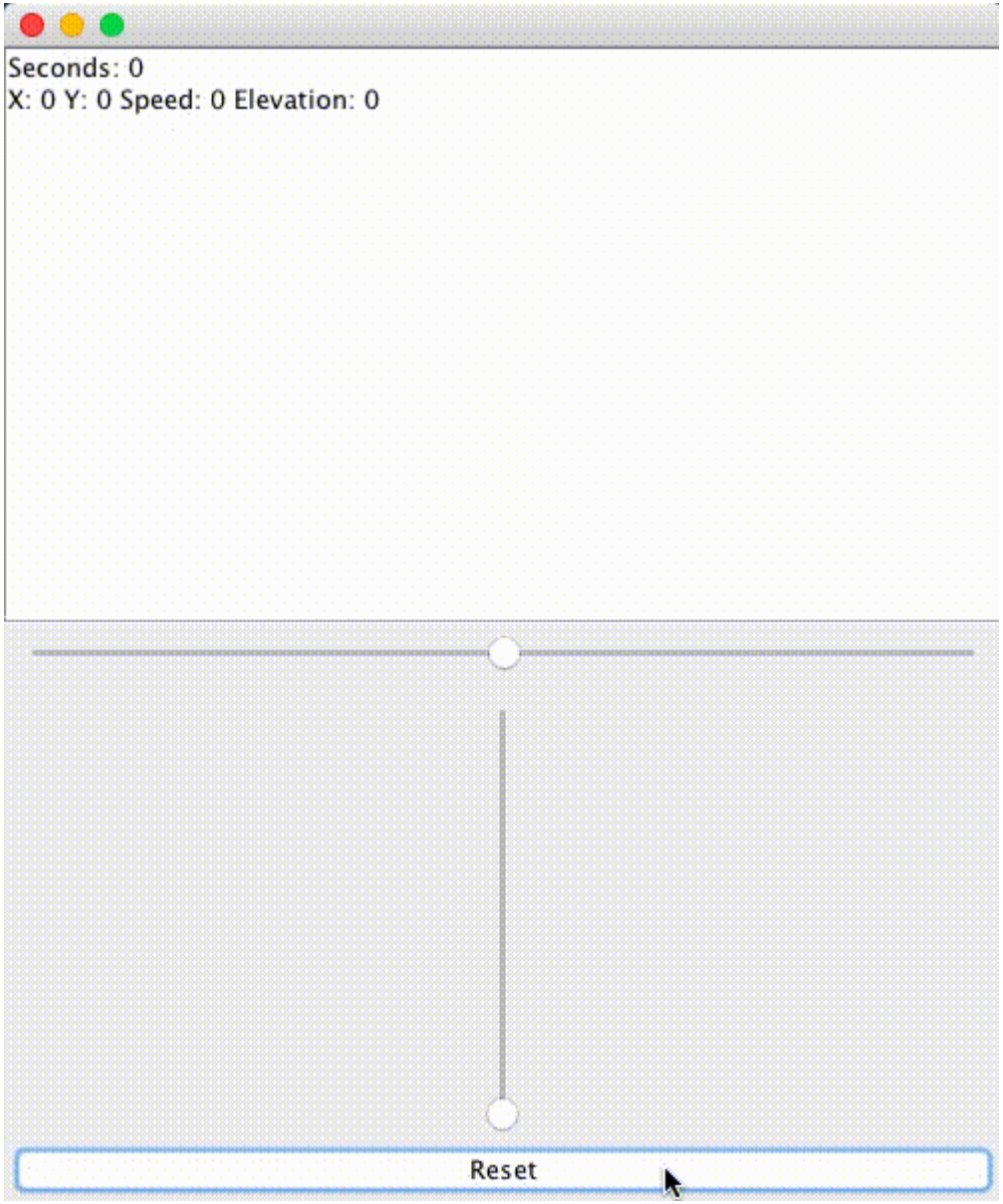
1. Plane minimum and maximum speed: 0 and 10
2. Plane minimum and maximum x position: 0 and 10
3. Runway width: 10
4. Runway length: 100
5. Plane initial x: 5
6. Plane initial y: 0
7. Speed required to increase elevation: 10
8. Time at speed required to increase elevation in order to increase elevation: 5
9. Elevation required for a successful take off: 5

Demonstrations

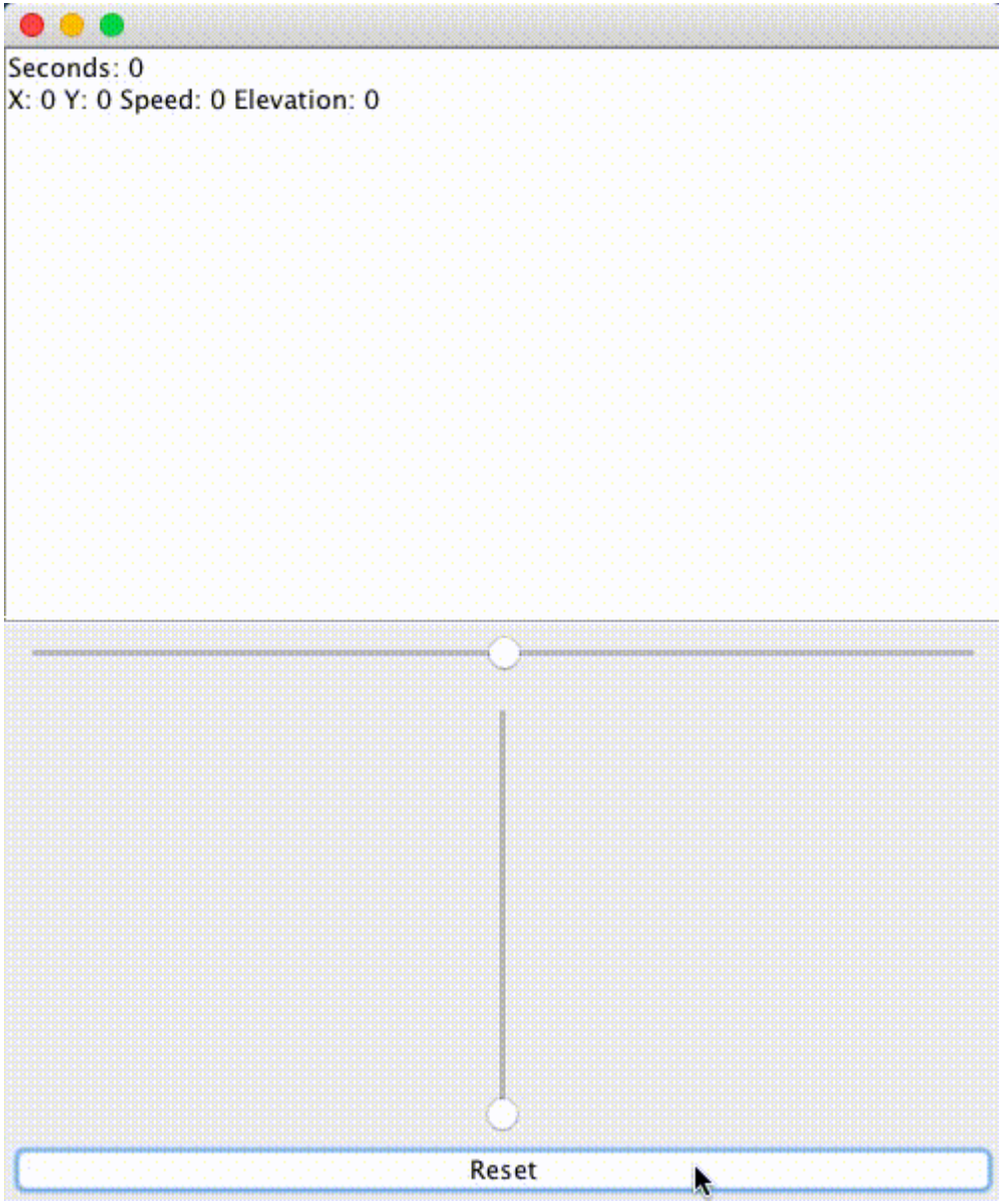
A successful takeoff is therefore as follows...



...while this is an unsuccessful takeoff due to a lack of speed (i.e. the front of the plane exceeds the end of the runway, but it is not moving quick enough)...



..and this is an unsuccessful takeoff due to incorrect x positioning...



As shown in these graphics, the simulation ends when the plane is either in flight, or the take off has failed.

Considerations

You will need to consider the following when building your simulator:

1. You might like to start by building your frame. Which components will you use? How will you get these components to appear in the correct positions on the frame? How can you neatly arrange your components so that when the window is sized, they remain in appropriate positions?

2. Which classes can we reuse from previous assignments? If we can reuse any, we should, in order to avoid duplicating work. If you do reuse classes, they should be stored in a separate package in your solution.

3. How will you represent the runway in your program? Are there any existing library classes that can help us with this?

4. How can we control the passage of time in Java? You may wish to look at the static method `sleep` in the `Thread` class. How can what we saw in the Tube example help us deal with the idea of an *exception* here (specifically the `InterruptedException`), which we are yet to formally discuss?

5. You will need to update the position of the plane every second, based upon the state of the plane controls in the simulator. For the X position of the plane this involves simply translating the position of the centre stick to the position of the plane. For the Y position of the plane, you will need to combine the position of the throttle (the speed of the plane) with the time the plane has been moving, in order to derive an incremented y position. For simplicity, we will not consider the time it takes for a plane to reach a certain speed, but

instead consider it to reach that speed immediately. The graphics above may be useful in observing this fact.

6. You will need to update the elevation of the plane every second, based upon whether it is going at the required speed, and whether it has been going at that required speed for the required duration.

7. How will you determine whether the plane is in flight or when a take off has failed, and how will this information be used to stop the progression of the simulation if necessary?

8. Consider how you might formally separate the key elements of your program into different classes, and into different packages. We may not have yet covered enough in the course in order for you to be aware of, or to achieve, the optimum separation, so this will not affect your marks, but try and achieve the most natural separation of your code as you can.

Optional tasks

1. As an optional task, you may like to implement the reset button functionality, which may not be as trivial as it seems.

    1. When clicked, the position of the plane, and all of its associated attributes, should be reset, as should the position of the controls in the UI. Then the main simulation loop should begin again.

    2. Depending on how you've separated your code you may need to consider different techniques for inter-object communication.

    3. Do you experience any unusual behaviour from your GUI when trying to implement this functionality? If so, consider why this might be, and what a proposed solution may be.

4. Regardless of whether you complete this task, your UI should still display the reset button.

2. The current controls are very simple in appearance. Try adding some extra information to them, such as labels for the throttle and centre stick. You may even wish to add an extra component to display speed information.

1. Any modifications you make to the UI for this additional task should not affect the basic functionality you have implemented.