Test plan

Unit tests will be done for each component of the MVC architecture. A series of unit tests will be produced to test the components.

Unit test: Model

Who will do them: Brian (maybe), Oscar, Kelvin.

What will be done: The tests below will be written as a main method in separate Java wrapper classes (one for each test) and stored as .java files in the SVN folder /src/unitTests/Model.

1. Instantiate an airport (“Heathrow”) with a physical runway called 09L/27R. Add to it a runway called 09L with TORA = 3902; TODA = 3902; ASDA = 3902; LDA = 3595.

Expected result: The variable’s values are printed using system.out.println() to show that they are assigned to the correct values.

1. Add to “Heathrow” physical runway 09L/27R a runway called 27R with TORA = 3884; TODA = 3962; ASDA = 3884; LDA = 3884.

Expected result: The variable’s values are printed using system.out.println() to show that they are assigned to the correct values.

1. Instantiate an obstacle (Height = 25). Expected result: An instance of Obstacle is created with height set to 25 printed using system.out.println().
2. Calculate new values for runway in 09L with obstacle in 3 and 500 from the 09L threshold value. Expected result: TORA = 2092, TODA = 2092, ASDA = 2092, LDA = 2795.
3. Calculate new values for runway in 27R with obstacle in 3 and 27R from the threshold value. Expected result: TORA = 3084, TODA = 3162, ASDA = 3084, LDA = 2074.
4. Save airport in 1 with the runways declared in 1 and 2 to an XML file. Expected result: Heathrow airport is created with 1 physical runway and within this runway there are 2 runways with the TORA, TODA, ASDA and LDA values set.
5. Save obstacle in 3 to an XML file. Expected result:
6. Load an airport (saved in 6) from XML file. Expected result: All values stored in the file are correctly loaded into variables.
7. Load an obstacle (obstacle 5 saved in 13) from XML file. Expected result: All values stored in the file are correctly loaded into variables.

Unit test: View

Who will do them: Edward, Kristian, Brian (maybe).

What will be done: The tests below will be written as a main method in separate Java wrapper classes (one for each test) and stored as .java files in the SVN folder /src/unitTests/View.

1. Open the application. Expected result: The window should open with everything in place.
2. Save airport/obstacle pop up.
3. Load airport/obstacle pop up.
4. Edit airport/obstacle pop up.
5. Visualisation of the runway – obstacles should seem like they are in the correct place with respect to the runway.
6. Calculation breakdown – numbers should be correct and labeled correctly.

Unit test: Controller

Who will do them: Brian.

What will be done: The tests below will be written as a main method in separate Java wrapper classes (one for each test) and stored as .java files in the SVN folder /src/unitTests/Controller.

1. Clicking on menu items should result in the correct action being performed. E.g. clicking Save Airport should open the save dialogue box etc.
2. Clicking on the buttons on the pop ups in View should yield the correct actions being taken. E.g. clicking “place obstacle” should calculate new values and update the visualization.

Integration testing

Who will do them: All.

What will be done: we will make M, V and C mingle and win. I don’t know because the unit tests for controller is kind of like integration testing?!?