Team 12

CSCD 350

Group Project Part 3 System Test and Evaluation

6/3/2021

**Airplane Tests**

**Test 1:** Airplane Straight-and-Level Flight

1.

This test tests if a created plane can fly at a constant altitude in a straight line without changing directions

2.

A bomb daBomb is created, from there we define an airplane plane, then we create daPlane from plane and give it the coordinates 45\*30'15#/110\*30'10#/200 with course 0 speed 5. We set the altitude to 1000 so the plane is flying off the ground.

3.

define munition bomb daBomb

define airplane plane with munition (daBomb)

create actor daPlane from plane at 45\*30'15#/110\*30'10#/1000 with course 0 speed 5

4.

The expected results are a plane traveling in a straight horizontal line at a constant altitude

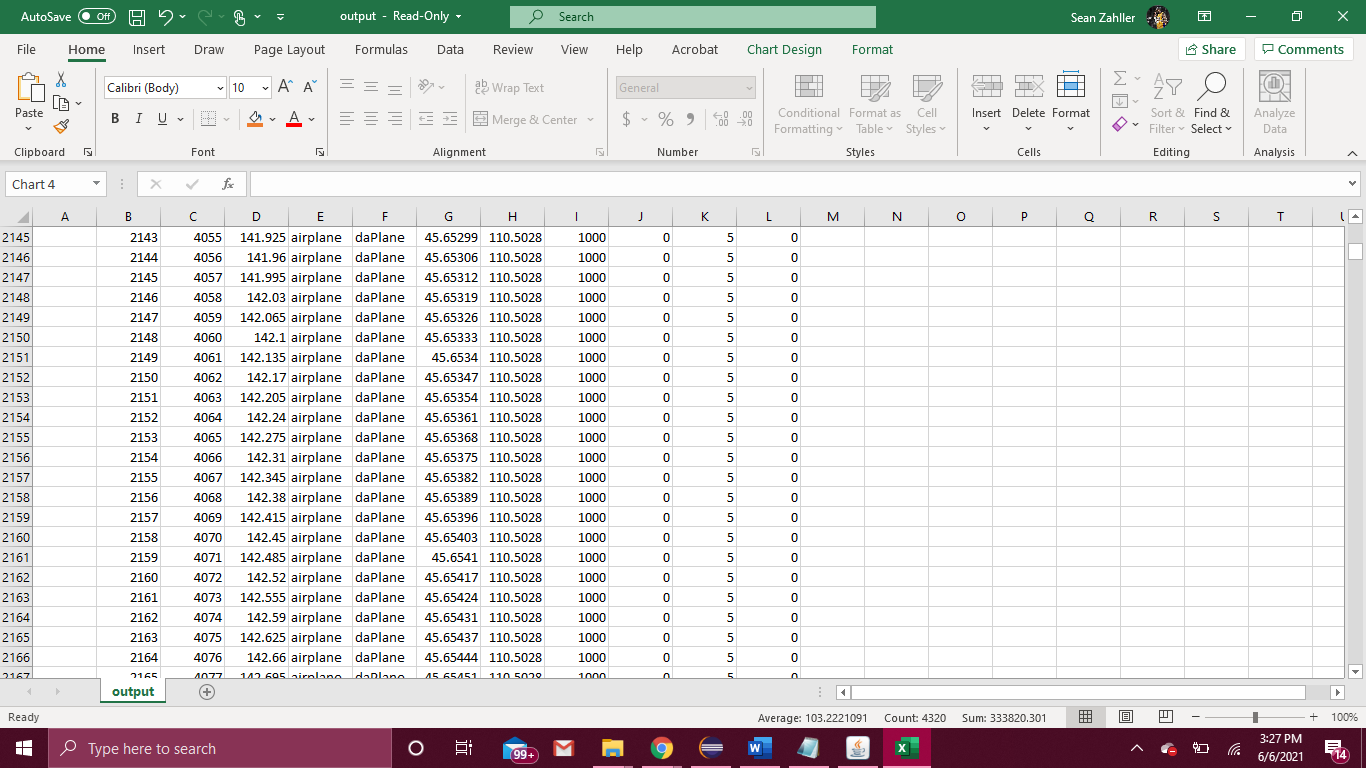
5.

The results were a plane traveling in a straight horizontal line at a constant altitude

Graphical user interface, text, timeline

Description automatically generated

6.



Graphical user interface, application, table, Excel

Description automatically generated

In the first screenshot we can see in event number 2143 the plane is traveling slowly at longitude 110.5028, the latitude is increasing from 45.63299, with an altitude of 1000.

In the next screen shot we are at event 33184, here we can see the longitude is still at 110.5028, the altitude is still 1000, and the latitude has now increased to 47.80847 and is still increasing.

This shows how the plane is flying on a straight horizontal path

7.

The results and the expected results did not differ

8.

A way to extend this test is to create another plane and see if it can travel in a horizontal line in the opposite direction at the same time and get the same expected results.

**Test 2:** Airplane Climbing

1.

This test tests if a plane can travel in a constant upwards direction gaining elevation without changing directions.

2.

A bomb daBomb is created, from there we define an airplane plane, then we create daPlane from plane and give it the coordinates 45\*30'15#/110\*30'10#/200, so it starts with an altitude of 200 and has course 0 speed 10. We then set the altitude for daPlane 60,000.

3.

define munition bomb daBomb

define airplane plane with munition (daBomb)

create actor daPlane from plane at 45\*30'15#/110\*30'10#/200 with course 0 speed 5

set daPlane altitude 60000

4.

The expected results are that a plane will be created and fly at a constant rate and direction while gaining elevation at a constant rate

5.

The actual results were that the plane flew in a constant rate and direction while gaining elevation at a constant rate

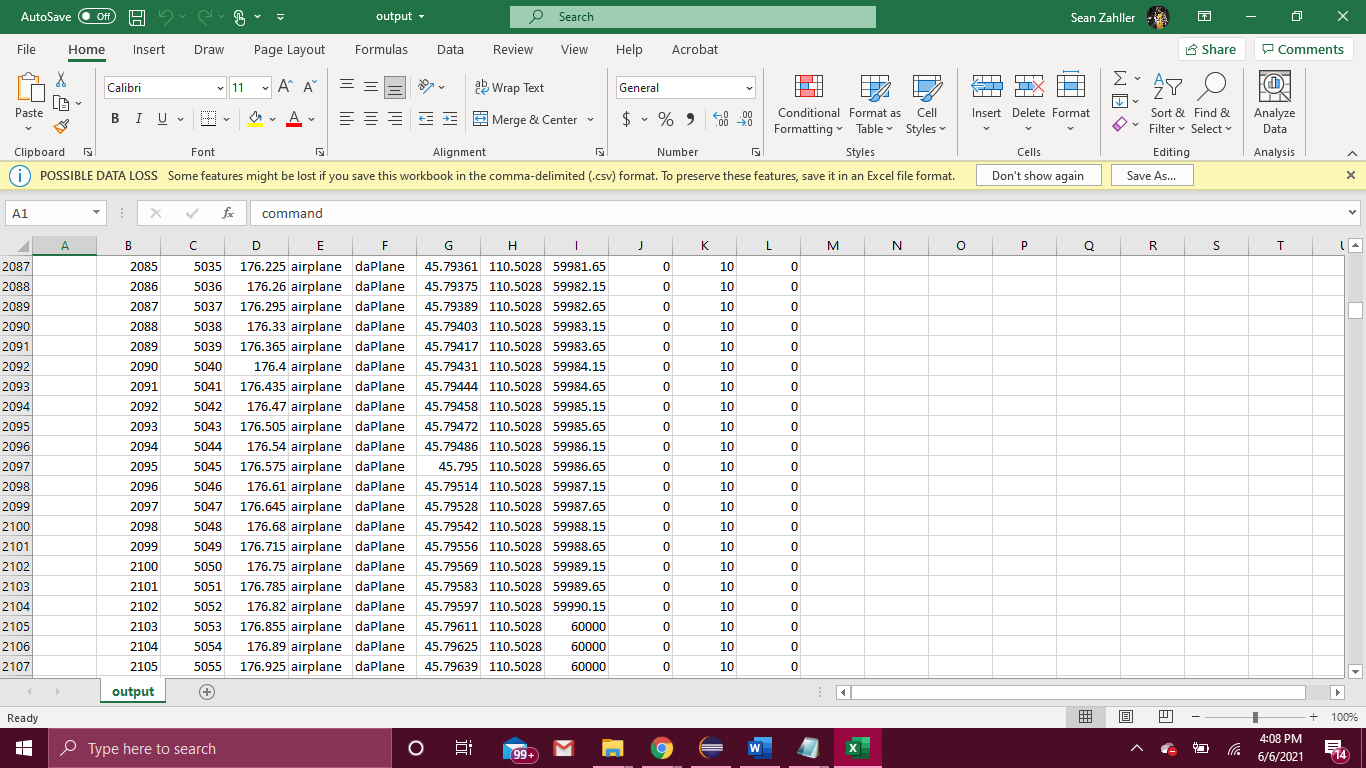
Graphical user interface

Description automatically generated

6.

Graphical user interface, application, table, Excel

Description automatically generated



In the first screenshot we can see that the plane begins to climb from altitude 200 at a constant rate from latitude 45.57778 while the longitude is at 110.5028

In the second screenshot we can see that the plane continues to climb to altitude 60000 at a constant rate from latitude 59987.15 while the longitude is at 110.5028.

This shows how the plane can climb elevation at a constant rate.

7.

The results and the expected results did not differ

8.

A way to extend this test would be to descend back down to ground level after ascending to a certain altitude.

**Test 6:** Airplane 180-Degree Turn, Slow-Speed

1.

This test is to test if an airplane can make a 180 degree turn(reverse directions) at a slow constant speed

2.

A bomb daBomb is created, from there we define an airplane plane, then we create daPlane from plane and give it the coordinates 49\*40'15#/117\*26'10#/200 with course 0 speed 2. From there we change the course to 180 degree turn.

3.

define munition bomb daBomb

define airplane plane with munition (daBomb)

create actor daPlane from plane at 49\*40'15#/117\*26'10#/200 with course 0 speed 2

set daPlane course 180

4.

The expected results are that a plane will be created and fly making a 180-degree turn and keep flying in that direction. The turn will will be a well rounded turn

5.

The actual results that a plane was created and made a 180-degree sharp turn and kept flying in that direction

Graphical user interface

Description automatically generated

6.

Graphical user interface, application, table, Excel

Description automatically generated

In the screenshot above we can see that plane began to head to 180 degrees in constant increments

7.

The results weren’t exactly as expected. The plane made an unrealistic very sharp 180-degree turn before the plane could move very far, I thought the plane would turn a little slower and would make a more rounded turn. This could be because the window zoom is very zoomed out

8.

A way to extend the test would be to see if you can make a 180 degree turn while descending at the same time, as if you were getting ready for an emergency landing in a flat area behind the plane.

Bomb Tests

**Test 8:** Bomb Drop, High Speed

1.

This test is to test if a bomb can be dropped from an airplane traveling at a high speed and at altitude 8000 down onto a ship

2.

A bomb daBomb is created, from there we define an ship target, then we create daTarget from target and give it the coordinates 50\*8'29#/117\*26'10#/0 with course 0 speed 0. From there we define an airplane plane, and then create daPlane from plane at 49\*40'15#/117\*26'10#/8000 with course 0 speed 10. We then load and deploy daBomb from the daPlane down to daTarget

3.

define munition bomb daBomb

define ship target with munition (daBomb)

create actor daTarget from target at 50\*8'29#/117\*26'10#/0 with course 0 speed 0

define airplane plane with munition (daBomb)

create actor daPlane from plane at 49\*40'15#/117\*26'10#/8000 with course 0 speed 10

set daPlane load munition daBomb

set daPlane deploy munition daPlane.daBomb.1

4.

The expected results are that a plane flying at a high speed will drop a bomb and hit the target ship from altitude 8000

5.

The actual results are that a plane flew at a high speed and dropped a bomb that hit the target ship from altitude 8000

A computer screen capture

Description automatically generated with medium confidence

6.

Graphical user interface, application, table, Excel

Description automatically generated

As you can see in the screenshot above the bombs last frame is at the exact same latitude 50.14, and the same longitude 117. This shows how the bomb landed on the ship.

7.

The results were as expected. It took me a few tries to get how far the bomb travels from altitude 8000, but I got it to hit the target after I knew the length of the bombs latitude travel from initial deployment

8.

A way to extend the test would be to have the ship moving in the same direction but much slower, this would simulate a moving target

**Test 9:** Bomb Drop, Low Speed, Hit

1.

This test is to test if a bomb can be dropped from an airplane traveling at a low speed and at altitude 8000 down onto a ship

2.

A bomb daBomb is created, from there we define an ship target, then we create daTarget from target and give it the coordinates 49\*44'59#/117\*26'10#/0 with course 0 speed 0. From there we define an airplane plane, and then create daPlane from plane at 49\*40'29#/117\*26'10#/8000 with course 0 speed 2. We then load and deploy daBomb from the daPlane down to daTarget

3.

define munition bomb daBomb

define ship target with munition (daBomb)

create actor daTarget from target at 49\*44'59#/117\*26'10#/0 with course 0 speed 0

define airplane plane with munition (daBomb)

create actor daPlane from plane at 49\*40'29#/117\*26'10#/8000 with course 0 speed 2

set daPlane load munition daBomb

set daPlane deploy munition daPlane.daBomb.1

4.

The expected results are that a plane flying at a low speed will drop a bomb and hit the target ship from altitude 8000

5.

The actual results are that a plane flew at a low speed and dropped a bomb that hit the target ship from altitude 8000

A screenshot of a computer

Description automatically generated

6.

Graphical user interface, application, table, Excel

Description automatically generated

As you can see in the screenshot above the bombs last frame is at the exact same latitude 49.74, and the same longitude 117. This shows how the bomb landed on the ship.

7.

The results were as expected. It took me a few tries to get how far the bomb travels from altitude 8000 now at a lower speed than before, but I got it to hit the target after I knew the length of the bombs latitude travel from initial deployment

8.

A way to extend the test would be to have the ship moving in the opposite direction and starting further north than the airplane so the cross paths, this would simulate a moving target that the plane passes over

**Test 10:** Bomb Drop, Low Speed, Miss

1. This test is to test if a bomb can be dropped from an airplane traveling at a low speed and at altitude 8000 down onto a ship but miss the bomb.

2. A bomb daBomb is created, from there we define an ship target, then we create daTarget from target and give it the coordinates 49\*44'59#/117\*26'10#/0 with course 0 speed 0. From there we define an airplane plane, and then create daPlane from plane at 49\*40'29#/117\*26'10#/8000 with course 0 speed 2. We then load and deploy daBomb from the daPlane down to daTarget and miss the daTarget.

3.

define munition bomb daBomb

define ship target with munition(daBomb)

create actor daTarget from target at 49\*44'59#/117\*26'10#/0 with course 0 speed 0

define airplane plane with munition (daBomb)

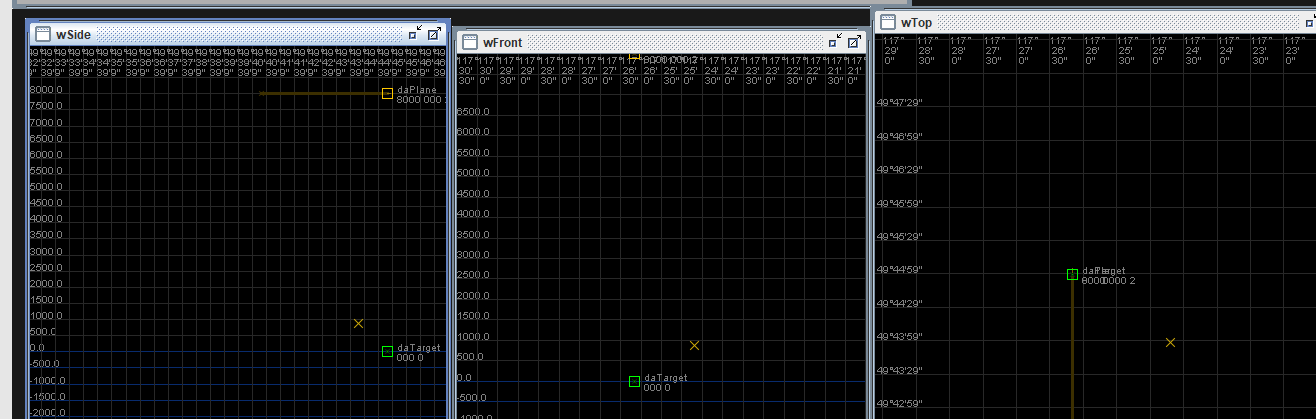
create actor daPlane from plane at 49\*40'29#/117\*26'10#/8000 with course 0 speed 2

set daPlane load munition daBomb

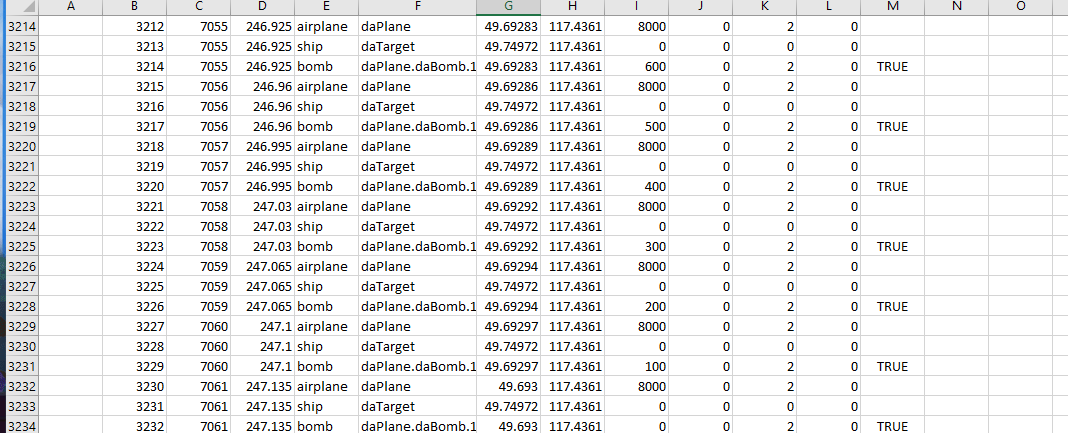
set daPlane deploy munition daPlane.daBomb.1

4. The expected results are that a plane flying at a low speeds will drop a bomb and miss the target ship from altitude 8000.

5. The actual results are that the plane flying at low speed missed the target ship at altitude 8000 by dropping it too early.



6.



As seen in the screenshot above the Latittude of daTarget and the deplane.dabomb are not the same latitude at the end of the bombs report.

7. The results were expected because the bomb was set to miss and the ship was no were close to being hit by the bomb.

8. To extend this practice is to make the target move at some speed making it harder for the plane to get a miss on the target.

**Depth-Charge Tests**

All depth charges are dropped by a ship.

**Test 11**: Depth Charge, Acoustic Fuze, Hit

1.

2.

3.

4.

5.

6.

7.

8.

**Test 12:** Depth Charge, Acoustic Fuze, Miss

1.

2.

3.

4.

5.

6.

7.

8.

**Test 13:** Depth Charge, Depth Fuze

1. This test is to test of the depth\_charge from a ship will activate at a depth fuze at -1000 depth.

2. A depth sensor was created first called (daSense) then the depth charge was created called (daCharge) with depth sensor. A ship was then defined(holder) with munition (daCharge). The actor (daholder) at 49\*44'59#/117\*26'10#/0 with course 0 speed 0 . We the set the actor with the munition (daCharge) and then deployed the depth\_charge from the (daholder).

3.

define sensor depth daSense with trigger depth 2000

define munition depth\_charge daCharge with fuze daSense

define ship holder with munition (daCharge)

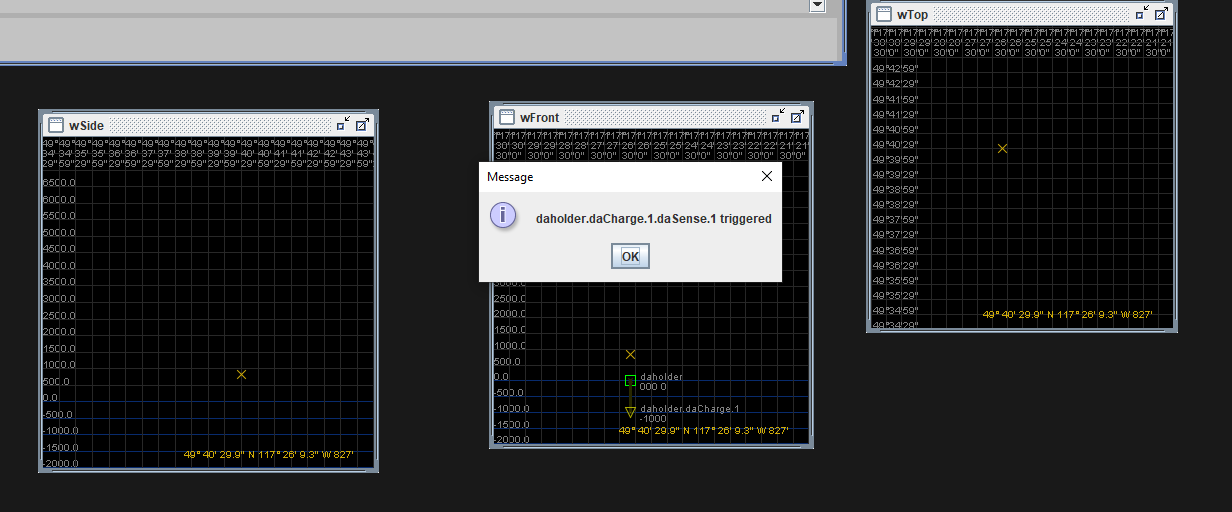
create actor daholder from holder at 49\*44'59#/117\*26'10#/0 with course 0 speed 0

set daholder load munition daCharge

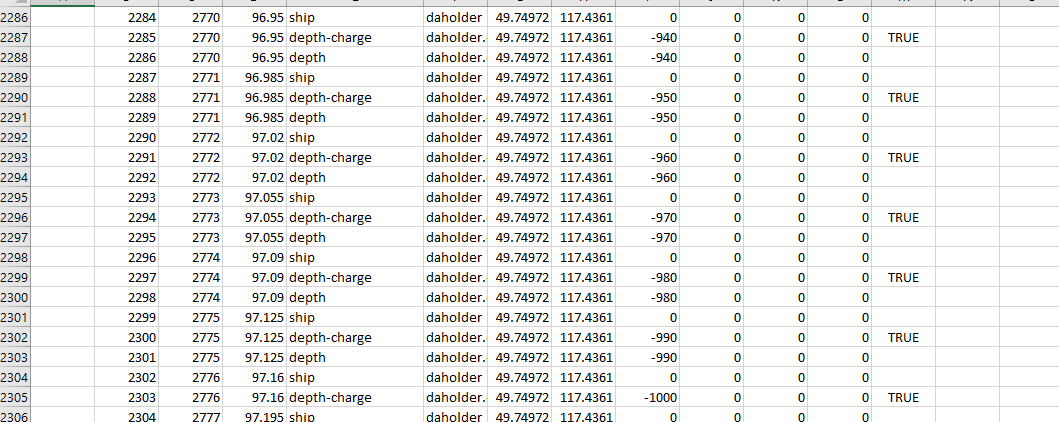
set daholder deploy munition daholder.daCharge.1

4. The expected results are that a ship would deploy a depth charge with a depth fuze to activate at -1000 depth.

5. The actual results are that the ship deployed the depth charge and the depth fuze activate the charge at exactly -1000 depth.



6.



In the screenshot above the depth-charge last action was a -1000 depth meaning that the depth fuzed worked properly.

7. The results were as expected. The first time I tried it did not work because I had set the depth to 1000 and not -1000 depth and never went off.

8. A way to extend this test is to have a submarine to hit at a certain depth with the depth\_charge at a certain depth.

**Test 14:** Depth Charge, Sonar Fuze

1. This test is to test the Passive Sonar Fuze from a ship and detect any movement.

2. At first the Passive sonar sensor was defined with 100 sensitivity called daSensor, the Depth\_charge was defined next with daSensor. We defined a ship called holder with munition daCharge. An actor was created at 49\*44'59#/117\*26'10#/0 with course 0 speed 0. Load the munition to the actor (daholder). We defined a bomb so it could go on daPlane and we create the plane at 49\*40'35#/117\*26'10#/50 with course 0 speed 3.

3.

define sensor sonar passive daSensor with sensitivity 100

define munition depth\_charge daCharge with fuze daSensor

define ship holder with munition (daCharge)

create actor daholder from holder at 49\*44'59#/117\*26'10#/0 with course 0 speed 0

set daholder load munition daCharge

define munition bomb daBomb

define airplane plane with munition (daBomb)

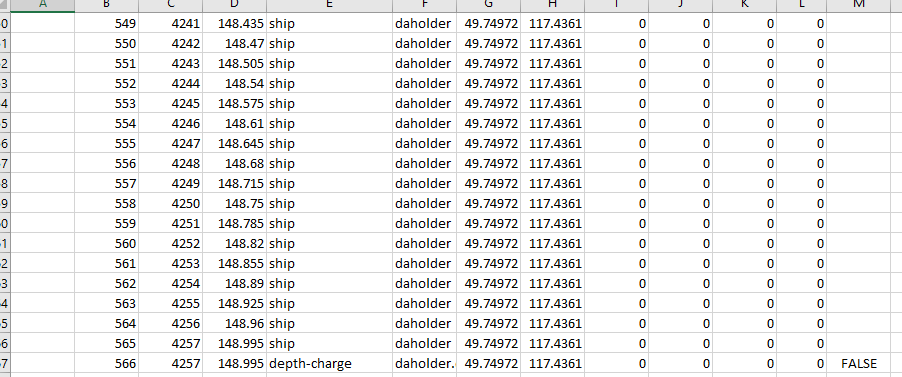
create actor daPlane from plane at 49\*40'35#/117\*26'10#/50 with course 0 speed 3

4. The expected results are that the ship would have a passive sensor detect the plane.

5.



The actual results show that the ship did detect the daPlane.

6. 

7. The results were not as expected. Took me a while figure out how to get the plane detected how fast the plane should be moving and a long time to figure out the sensitivity of the Passive sensor.

8. A way to extend the test is to have the depth\_charge detect a faster moving plane or submarine going the opposite direction as well have the depth\_charge move.

**Test 15:** Depth Charge, Time Fuze

1. This test is to test of the depth charge with a timed fuze.

2. At first the time fuze was created, from that we created the depth charge with the timed fuze(tick), we then needed a define ship called (holder) to hold the depth charge called (char). We then created the ship at coordinates 49 \*44'59#/117\*26'10#/0 with course 0 speed 0. From there we loaded the depth\_charge with the fuze to the ship and then deployed the depth\_charge.

3.

define sensor time tick with trigger time 20

define munition depth\_charge char with fuze tick

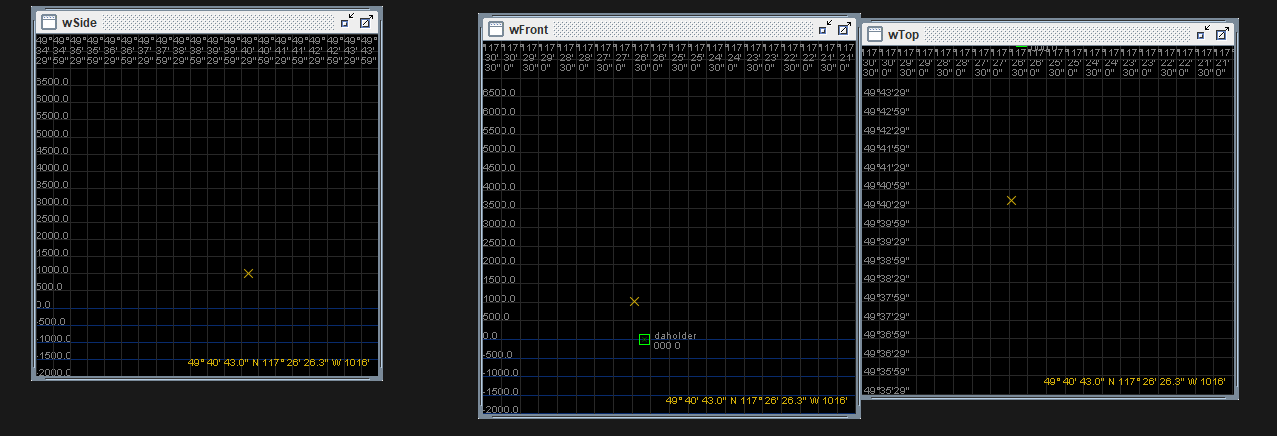
define ship holder with munition (char)

create actor daholder from holder at 49 \*44'59#/117\*26'10#/0 with course 0 speed 0

set daholder load munition char

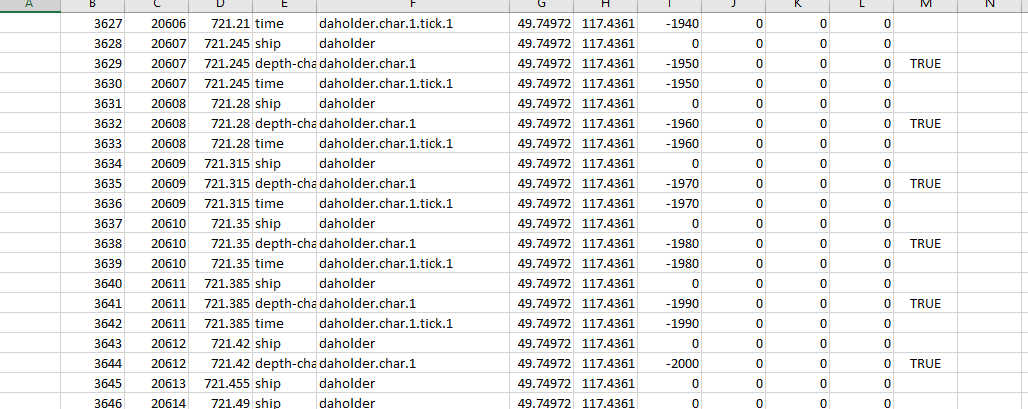
set daholder deploy munition daholder.char.1

4. The expected results were that the ship would drop the depth\_charge and the timer fuze would activate the depth\_charge after 5 seconds.

5. 

The actual results are that the ship dropped the depth\_charge and after 5 seconds the depth\_charge went off due to the fuze timer.

6.



As you can see the da.holder.char.1.tick.1 is a timer fuze and at -2000 the fuze went off.

7. The results as expected. Had trouble at first because the depth\_charge would take too long to go off due to giving too much time.

8. A way to extend this test is to have a longer depth so the depth\_charge could travel farther.

**Missile Tests**

**Test 16:** Missile, Radar Sensor, Depth Fuze

1.

2.

3.

4.

5.

6.

7.

8.

**Test 17:** Missile, Radar Sensor, Distance Fuze

1. This is a test to test a missile being fired from ship at a airplane using a radar sensor and a distance fuze denonating near the airplane.

2. We start with making a radar sensor called (daSense), second is the (daFuze) with trigger distance 5. We create the Missile with the two sensors. We define a ship and load the Missile to both the ship and plane. We load the Munition and deploy the Missile towards the airplane.

3.

define sensor radar daSense with field of view 30 power 50 sensitivity 10

define sensor distance daFuze with trigger distance 5.0

define munition missile daMissile with sensor daSense fuze daFuze arming distance 1

define ship holder with munition (daMissile)

define airplane planeholder with munition (daMissile)

create actor daShip from holder at 49\*44'59#/117\*26'10#/0 with course 0 speed 0

create actor daPlane from planeholder at 49\*44'59#/117\*21'10#/0 with course 0 speed 0

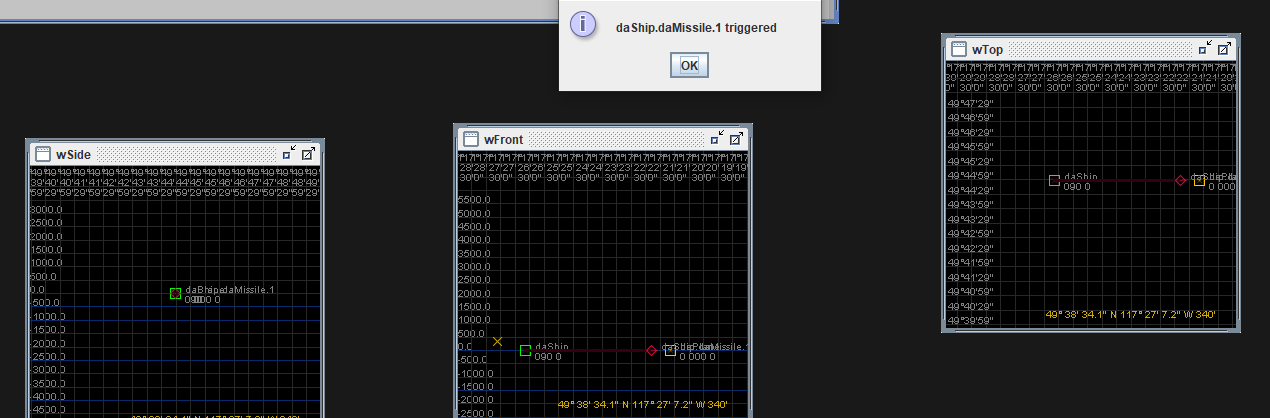
set daShip load munition daMissile

@wait 10

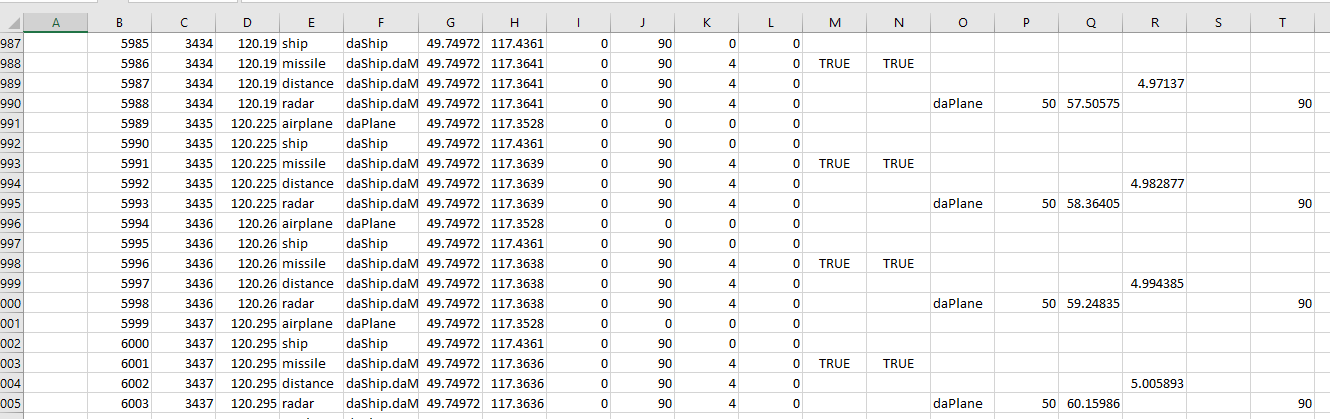
set daShip deploy munition daShip.daMissile.1

4. The expected results are that the ship would fire the Missile and due to the fuze the missile will detonate after the distance has been traveled making it miss the plane.

5. The actual results are that the ship fired a Missile and right before getting to the plane the Fuze set the Missile to detonate causing to miss the Plane.



6.



As you can see the Missile detonated at approximately 5.00583 and did indeed miss the plane because the Missiles longitude is at 117.3636 and the plane’s is at 117.3528.

7. The results were as expected a bit off but nothing is perfect. The Missile not exploding at the set distance of 5.0 as well as exploding near the plane and not at the plane were expected.

8. A way to extend this test is to have the airplane moving at slow speeds towards the ship and the ship having to still miss the plane while heading towards the Missile.

**Test 18:** Missile, Radar Sensor, Radar Fuze

1.

2.

3.

4.

5.

6.

7.

8.

**Test 19:** Missile, Radar Sensor, Thermal Fuze

1.

2.

3.

4.

5.

6.

7.

8.

**Test 20:** Missile, Radar Sensor, Time Fuze

1.

2.

3.

4.

5.

6.

7.

8.

**Test 21:** Missile, Thermal Sensor, Radar Fuze

1.

2.

3.

4.

5.

6.

7.

8.

**Test 22:** Missile, Thermal Sensor, Radar Fuze, Field-of-View Miss

1.

2.

3.

4.

5.

6.

7.

8.

**Test 23:** Missile, Radar Sensor, Radar Fuze, Aspect Angle

1.

2.

3.

4.

5.

6.

7.

8.

**Test 24:** Missile, Thermal Sensor, Thermal Fuze

1.

2.

3.

4.

5.

6.

7.

8.

**Torpedo Tests**

**Test 25:** Torpedo, Sonar Sensor, Acoustic Fuze, Fast Target

1.

2.

3.

4.

5.

6.

7.

8.

**Test 26:** Torpedo, Sonar Sensor, Acoustic Fuze, Slow-Target Miss

1.

2.

3.

4.

5.

6.

7.

8.

**Test 27:** Torpedo, Sonar Sensor, Sonar Fuze

1.

2.

3.

4.

5.

6.

7.

8.

**Test 28:** Torpedo, Acoustic Sensor, Acoustic Fuze, Fast Target

1.

2.

3.

4.

5.

6.

7.

8.

**Test 29:** Torpedo, Acoustic Sensor, Acoustic Fuze, Slow-Target Miss

1.

2.

3.

4.

5.

6.

7.

8.

**Test 30:** Torpedo, Acoustic Sensor, Thermal Fuze

1.

2.

3.

4.

5.

6.

7.

8.