Vybor Airbase Example Mission - Development Overview

This time I wanted to write a guide that explains some of the implementation of the mission. Taking a look at the mission and its triggers alongside this guide will hopefully elucidate some things about my approach to using ANIMA. Note that it does not have to be used in this way!

The first thing I usually do is just fly around in the 3D editor for a bit and place some units of each side. In this case, some OPFOR on the roads leading to the airbase, and some BLUFOR hanging around the base in various defensive positions. This is not done with ANIMA and they are not given any waypoints. For this mission I also put a Game Logic named "airbase" right in the middle of the airfield, just as a general positional reference.

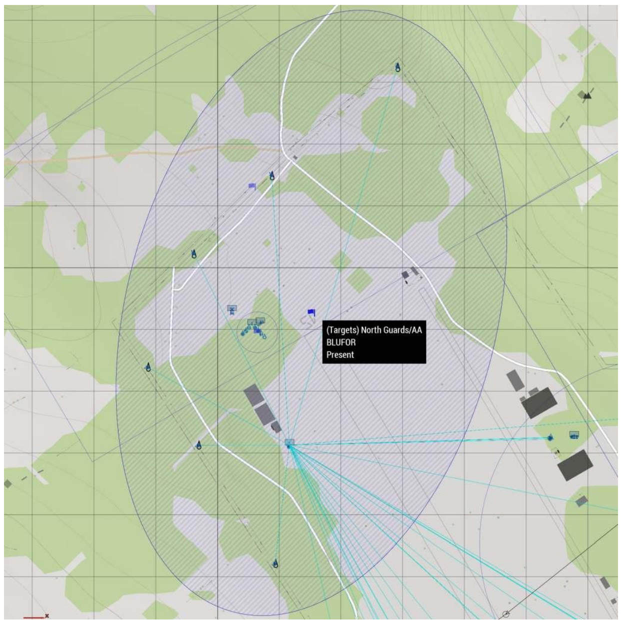
Next, I decide which groups of each side will be assigned to attack which targets, under which conditions. I use three different kinds of triggers to handle all the group selection, target selection, and attack conditions.

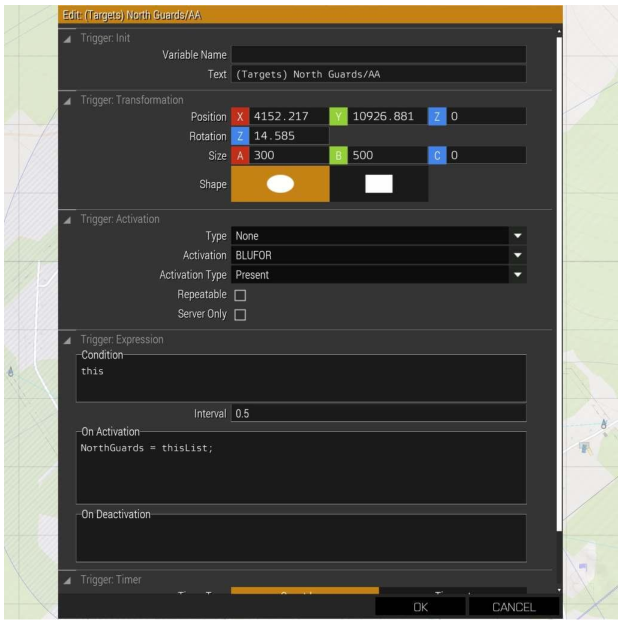
Targets triggers:

These triggers just mark units as targets for the other side. I set the trigger area to cover the target units, and activate on their presence, so the trigger goes off as soon as the mission starts. The trigger just assigns its "thisList" to a global variable. For this mission these variables are NorthGuards, EastGateGuards, and SouthGuards.

Type None | Activation: <target-side> | Activation Type: Present
On activation:

```
  <globalTargetArrayVariable> = thisList;
```




Edit: (Targets) North Guards/AA

Trigger: Init

Variable Name	
Text	(Targets) North Guards/AA

Trigger: Transformation

Position	X 4152.217	Y 10926.881	Z 0
Rotation	Z 14.585		
Size	A 300	B 500	C 0
Shape	●	■	

Trigger: Activation

Type	None
Activation	BLUFOR
Activation Type	Present
Repeatable	☐
Server Only	☐

Trigger: Expression

Condition
```
this
```
Interval 0.5

On Activation
```
NorthGuards = thisList;
```
On Deactivation

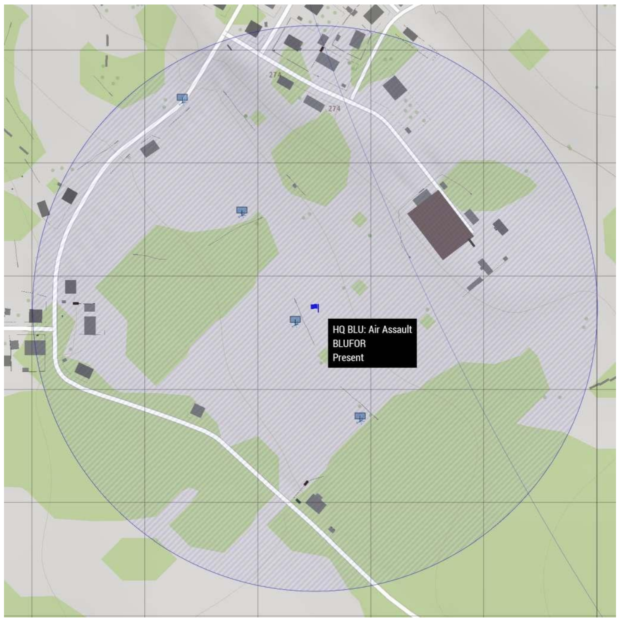
Trigger: Timer

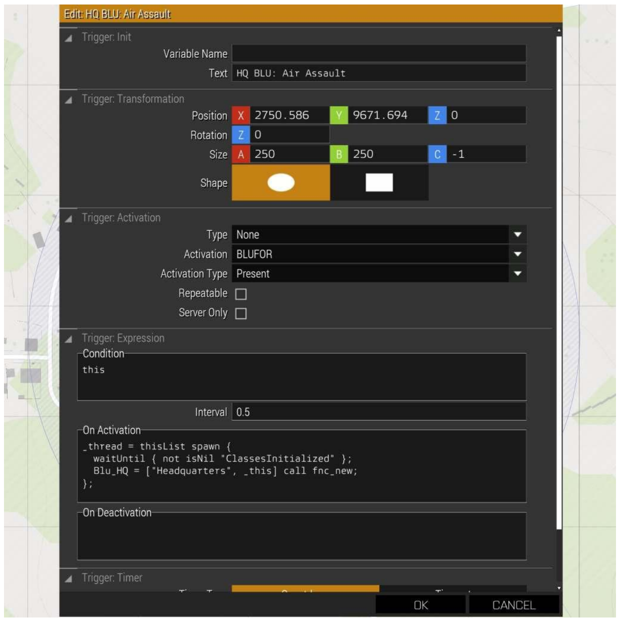
Headquarters triggers:

These triggers are set up a lot like targets triggers, but assign units to different Headquarters instances. The Headquarters is what allows us to issue "attack_targets" - we don't actually specify which groups go where; we just specify which Headquarters to use and how many groups to dispatch. (Technically groups can belong to multiple HQ's, but I don't do this here).

Type None | Activation: <hq-side> | Activation Type: Present
On activation:

```
  _thread = thisList spawn {
    waitUntil { not isNil "ClassesInitialized" };
    <HQ-variable> = ["Headquarters", _this] call fnc_new;
  };
```





Edit: HQ BLU: Air Assault

Trigger: Init

Variable Name	
Text	HQ BLU: Air Assault

Trigger: Transformation

Position	X 2750.586	Y 9671.694	Z 0
Rotation	Z 0		
Size	A 250	B 250	C -1
Shape	●	■	

Trigger: Activation

Type	None
Activation	BLUFOR
Activation Type	Present
Repeatable	☐
Server Only	☐

Trigger: Expression

Condition
```
this
```

Interval 0.5

On Activation
```
_thread = thisList spawn {
  waitUntil { not isNil "ClassesInitialized" };
  Blu_HQ = ["Headquarters", _this] call fnc_new;
};
```

On Deactivation

Trigger: Timer

You may note that in the example mission some of the HQ triggers contain additional code after assigning the HQ variable. Please see the User Manual if you'd like more information on what each argument means for attack_targets, land_transports, etc.

Here's some of the subsequent code from one of the triggers, with added comments to explain:

```
  // Do nothing until SouthGards target trigger has activated:
  waitUntil { not isNil "SouthGuards" };

  // Send two groups from OPFOR AT South HQ to attack SouthGuards,
  //  from the southeast (search starts 200m E and 200m S of targets)
  [Opf_AT_S_HQ, "attack_targets", SouthGuards, 2, true, true, [200, -200, 0]] call fnc_tell;
```

```
// Wait 5 minutes:
sleep 300;

// Send (same) two groups from OPFOR AT South HQ to attack both
//  EastGateGuards and SouthGuards, from the east.
[Opf_AT_S_HQ, "attack_targets",
 EastGateGuards+SouthGuards, 2, true, true, [150, 0, 0]] call fnc_tell;
```

Below is the subsequent code from another HQ trigger:

```
// Search for landing positions starting 800m east of "airbase"
//  and land Opf_heli_<1-4> there to unload.
[Opf_Heli_HQ, "land_transports",
 [[["_a", "_b"], {_a + _b}] call fnc_lambda,
  (position airbase), [800, 0, 0]] call fnc_map,
 [Opf_heli_1, Opf_heli_2, Opf_heli_3, Opf_heli_4],
 [], 130, 2, 6] call fnc_tell;
// Don't worry if you don't understand the lambda and map functions.
//  Those two lines together evaluate to add [800, 0, 0] to the
//  position of airbase and could also be written:
// [((position airbase) select 0) + 800,
//  (position airbase) select 1),
//  (position airbase) select 2]

// Send 4 groups from Opf_Heli_HQ to attack all BLUFOR targets. These
//  are the groups of infantry who are sitting inside Mi-24's. (Since
//  the Mi-24's are loaded transports, the HQ does not consider them
//  as dispatchable attack units!)
[Opf_Heli_HQ, "attack_targets", EastGateGuards+NorthGuards+SouthGuards, 4] call fnc_tell;

// Wait until all living units inside each Mi-24 who are *not* in the
//  same group as the flight crew have exited the helicopters:
waitUntil { 0 ==
  [[["_a", "_b"], {_a + _b}] call fnc_lambda,
   [[["_veh"], {
      { (alive _x) and
        (not ((group _x) == (group (driver _veh))))
      } count (crew _veh)
    }] call fnc_lambda,
    [Opf_heli_1, Opf_heli_2, Opf_heli_3, Opf_heli_4]] call fnc_map]
   call fnc_reduce};
// This mess of brackets/lambda/map/reduce just adds up the values of
// the count statement as applied to each Opf_heli in sequence.  When
```

```
  // the total is 0 we know the passengers got out (or died).

  // Now that they've unloaded, have the Mi-24's fly in a star back and
  //  forth over the airbase, with a radius of 3 kilometers:
  [Opf_Heli_HQ, "orbit_position",
     position airbase,
     [Opf_heli_1, Opf_heli_2, Opf_heli_3, Opf_heli_4],
     3000, 42, true, false, 13, true] call fnc_tell;
```

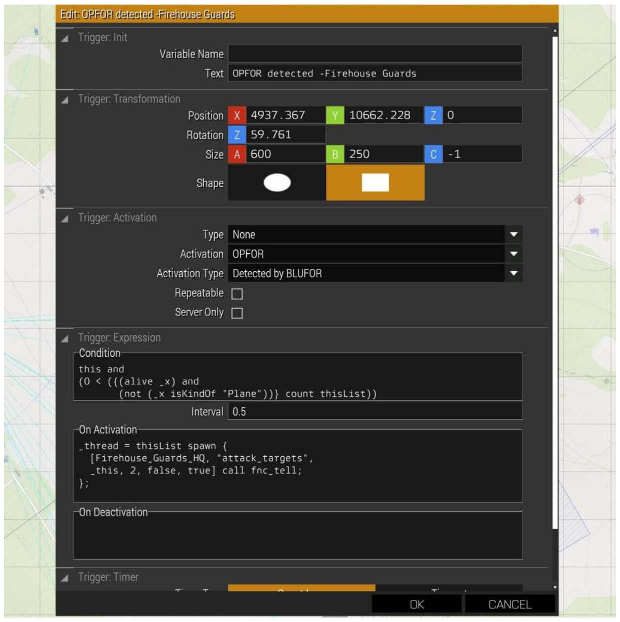
Detection triggers:

The third type of trigger I use in employing ANIMA is "Detected by", to dynamically give new attack orders once enemies come into contact.

Type: None | Activation: <detect-unit-side> | Activation Type: Detected by <other-side>
On activation:

```
  _thread = thisList spawn {
    [<some-hq>, “attack_targets”, _this, <n-groups>, false, true] call fnc_tell;
  };
```

OPFOR detected -Firehouse Guards
OPFOR
Detected by BLUFOR
Airfield


Edit: OPFOR detected -Firehouse Guards

Trigger: Init

Variable Name	
Text	OPFOR detected -Firehouse Guards

Trigger: Transformation

Position	X	4937.367	Y	10662.228	Z	0
Rotation	Z	59.761				
Size	A	600	B	250	C	-1
Shape	●		▭			

Trigger: Activation

Type	None
Activation	OPFOR
Activation Type	Detected by BLUFOR
Repeatable	☐
Server Only	☐

Trigger: Expression

Condition
```
this and
(0 < ({(alive _x) and
       (not (_x isKindOf "Plane"))} count thisList))
```
Interval 0.5

On Activation
```
_thread = thisList spawn {
  [Firehouse_Guards_HQ, "attack_targets",
  _this, 2, false, true] call fnc_tell;
};
```
On Deactivation

Trigger: Timer

A commented example:

```
_thread = thisList spawn {
  // Land Blu_heli_<1-4> at the airbase. Start the search
  //  at a 130m radius. Step count 2 and population size 6
  //  are very low, this mostly works because the airbase is
  //  already clear and flat. Normally higher values are used,
  //  which is unfortunately much slower.
  [Blu_HQ, "land_transports", position airbase,
   [Blu_heli_1, Blu_heli_2, Blu_heli_3, Blu_heli_4],
   [], 130, 2, 6] call fnc_tell;

  // Dispatch 4 groups in Blu_HQ to attack detected units:
  //  (These are the infantry squads in the Ghost Hawks)
  [Blu_HQ, "attack_targets", _this, 4] call fnc_tell;

  // Wait for the Ghost Hawk passengers to unload:
  waitUntil { 0 ==
    [[["_a", "_b"], {_a + _b}] call fnc_lambda,
     [[["_veh"], {
        { (alive _x) and
          (not ((group _x) == (group (driver _veh))))
        } count (crew _veh)
      }] call fnc_lambda,
      [Blu_heli_1, Blu_heli_2, Blu_heli_3, Blu_heli_4]] call fnc_map]
     call fnc_reduce};

  // Once empty, have Ghost Hawks orbit in a circle around the
  //  airbase, with a radius of 1km.
  [Blu_HQ, "orbit_position",
     position airbase,
     [Blu_heli_1, Blu_heli_2, Blu_heli_3, Blu_heli_4],
     1000, 42, true, false, 13, false] call fnc_tell;
};
```

Another detection trigger, for dispatching East AT Defense to detected OPFOR, contains a special condition (rather than the default "this"):

```
  private ["_ttl"];
  _ttl = East_AT_Defense getVariable "_ttl";
  if (not (isNil "_ttl")) then {
    if (120 < _ttl) then {
      East_AT_Defense setVariable ["_ttl", nil];
    } else {
      East_AT_Defense setVariable ["_ttl", _ttl + 0.5];
    };
  };
  this and (isNil "_ttl") and
    (0 < ({(alive _x) and ((_x isKindOf "Tank") or (_x isKindOf "Car"))} count thisList))
```

This trigger is set as repeatable with an interval of 0.5. It fires whenever a tank or car is detected, but because of the _ttl variable, which is saved and updated every 0.5s interval, the trigger only fully activates every 120 seconds. This is so new attack_targets orders are given to East_AT_Defense groups every two minutes.

Some other detection triggers have conditions such as:

```
this and
(0 < ({(alive _x) and
       (not (_x isKindOf "Air"))} count thisList))
```

That should ensure that the trigger doesn't fire for Planes or Helicopters.

Ending the mission:

This isn't really something I have much experience developing, but I figured I'd outline the three types of triggers I use to handle mission completion. None have a specific area or position.

Victory Condition triggers:

The mission only has one victory condition: for all OPFOR tanks and cars in the area to be destroyed. At that time (actually, between 5 and 30s later), the variable VCOND_Armor is set to true.

Condition:

```
this and (0 == ({((side _x) == opfor) and
                 (alive (driver _x)) and
                 ((_x isKindOf "Tank") or
                  (_x isKindOf "Car")) } count thisList))
```

On activation:

```
VCOND_Armor = true;
```

addAction triggers:

This mission doesn't immediately end when the victory condition is met, but adds an action to the player's interaction menu to end it. That's because the battle isn't really over at that point. The trigger for that is:

Condition:

```
VCOND_Armor
```

On activation:
```
  player addAction ["<t color='#FF0000'>End mission?</t>", {
    MissionComplete = true;
  }];
```

Mission Complete triggers:

When this mission is finally ended, which means MissionComplete is set to true, then the final trigger activates:

Condition:
```
  MissionComplete
```

On activation:
```
  "end1" call BIS_fnc_endMission;
```

Key considerations in using ANIMA:

Performance - ANIMA can be very slow. Whenever possible, use the smallest search populations and target step counts that you can.

Concurrency - Adding to the above, if multiple copies of the position finder are running at once, they get much slower. I frequently do this with attack_targets, but helicopter landing methods usually require a lot more computation and generally should not overlap each other (unless you really aren't in a hurry and don't mind them both taking random long amounts of time). This mission pretty much overlaps everything but I tried to keep things spread out in time as much as possible!

Search radius - Setting a small position search radius can help the algorithm evolve quickly if you start it in a suitable area, but it takes away some of the "automatic" aspects. If you already know where you want to go, after all, why use ANIMA? But a large radius can lead to aimless unsuccessful searching without a sufficiently high population size. So there are always trade offs.

Population size - This really depends on how variable the terrain is. Note that searching an empty field for landing spaces does NOT require a very high population count, because one spot is as good as any other. Searching a forest might require a much higher one, though, if you ever hope to get lucky enough that one of them learns where a clearing is.

Target step count - This can never be set lower than 2, and I find adjusting it to improve the search is rarely more efficient than just using a slightly larger population instead. But, YMMV. This is all trial and error! Note that the algorithm can be configured to, and will, run beyond this if it hasn't found clearly "better" positions.