Briefing Mission (Random) Tutorial

The ‘Fixed’ Briefing Tutorial was centered on Agia Marina .This tutorial will extend the ‘find random town’ logic developed in the PoW Random Tutorial to further the increase the randomization and replay value of a player made mission.

Preparing to Create the Mission

Copy the MissionBriefingRandom.Stratis subdirectory to:

> <MyDocuments>\Arma 3 - Other Profiles\<MyProfile>\missions\

Copy the Zen_FrameworkFunctions directory from the Shell.Stratis directory to:

> <MyDocuments>\Arma 3 - Other Profiles\<MyProfile>\missions\
> MissionBriefingRandom.Stratis\

Review the Map

No changes are required to the map. The area that will contain the mortar and patrolling squads will be randomly generated by Framework code.

If you do look in the map you’ll see the playable BLUFOR rifle unit south of Agia Marina. This squad will be moved to within a few hundred yards of the mortar using a call to a Framework function. Since the squad will be moved the mission designer can place the playable group anywhere on map.

Reviewing the Initialization Script

Open the *init.sqf* and *misc.sqf* files. This tutorial is not the ‘enter the code’ variety. It will just discuss the various sqf statements.

External Functions

The mission starts off with a call to *f_getrandomcityAreaMarker*, which has been moved from the *init.sqf* to the *misc.sqf* file.

Since this is the type of function that a mission designer might use across multiple missions it is easier to reuse from an external file.

And so the game engine can find the file at runtime it is ‘included’ at the top of *init.sqf*.

The mission continues by placing the eliminate mortar objective in a random position within the city marker. And then a squad is assigned to protect the mortars.

Sidebar: Documentation

No time like the present to start documenting our work. This tutorial will use the same style of documentation as the Framework.

So for *f_getrandomcityAreaMarker* we have this:

```
f_getrandomcityAreaMarker

Returns a reference to the area marker of a randomly chosen town.
The area marker is the size and shape defined in the map's config
file.

Usage: Call
Params: None
Return: Marker
```

See the file *documentation.txt* for all of this mission's function documentation.

Random Markers

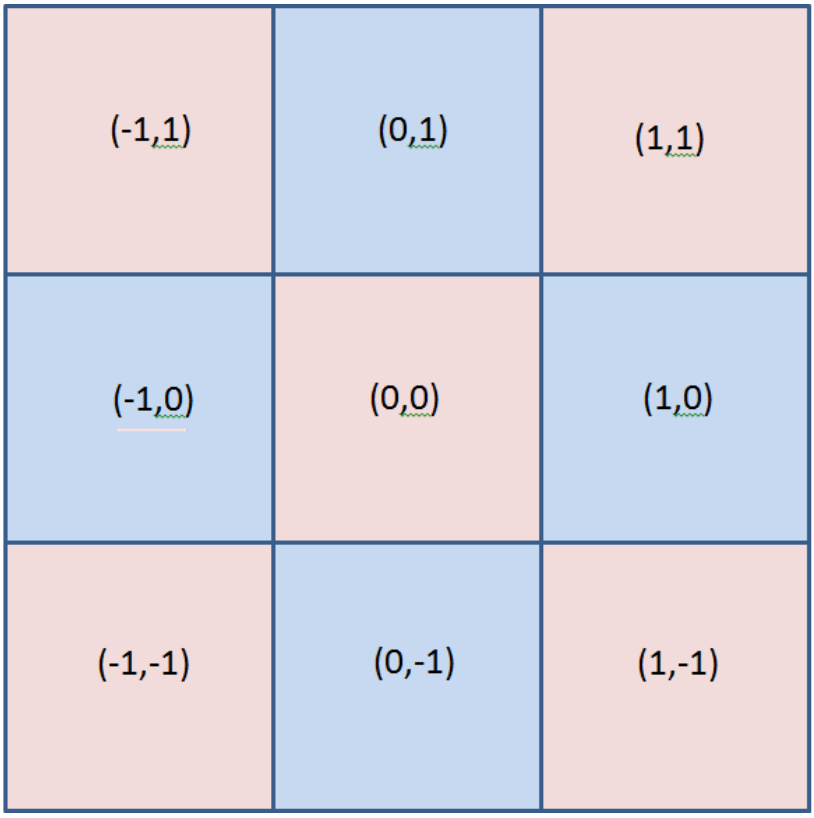
This mission code will generate a grid of markers over and around the randomly chosen city. It will do this in three steps, each step represented by a function in the *misc.sqf* file:

Create array of positions
Convert positions to array of area markers
Filter out 'watery' markers

Creating an Array of Positions

The first function is *f_getnearbyPositionsNine*.. It will create a 3 by 3 array of positions (X,Y coordinates) using two arguments: a center (represented as a X,Y coordinate) and the difference from this center point.

The nested for loops in the function work from the idea of a 'normalized' array in which the center is 0,0 and all other points are relative by -1 or 1. The diagram below illustrates the abstract grid:



Where the X value is ‘1’ add the difference to the X value of the center point. If the value is ‘-1’ subtract the difference value from the X value. If zero, just use X coordinate as supplied in the argument.

Apply the same idea to the Y coordinate of the center point.

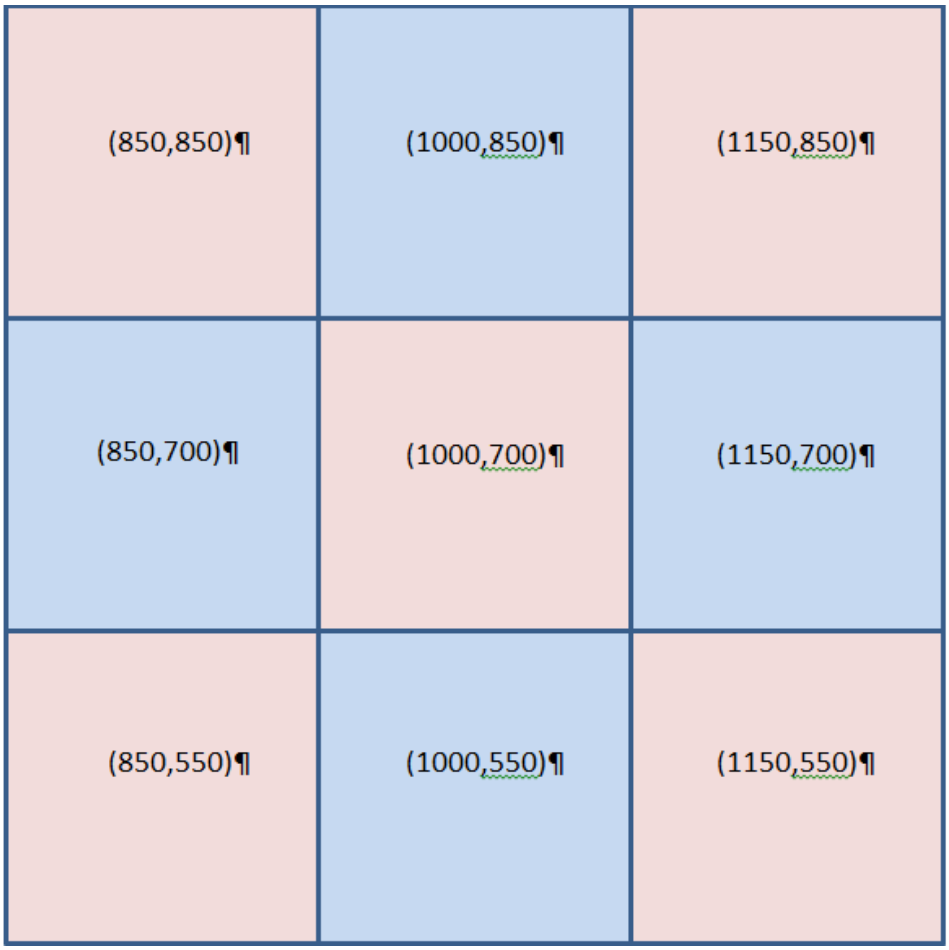
Sidebar: Documentation

```
f_getnearbyPositionsNine

Creates an array of nine positions with center (1) and offset of
(2).

Usage : Call
Params: 1. [int,int]
        2. Integer
Return: Array of [int,int]
```

Suppose the center position is 1000,700 and difference is 150. The picture below shows the positions that will be generated and appended to an array.



Creating Markers from Positions

Create an array of markers with a call of to *f_createMarkersfromArray*. The size of the area marker is the same as the 'difference' value used in *f_getnearbyPositionsNine* but doesn't have to be.

Sidebar: Documentation

```
f_createMarkersfromArray

Converts the array of positions (1) into an array of square area
markers with side length (2).

Usage : Call
Params: 1. Array of [int,int]
        2. Integer
Return: Array of objects of type marker
```

Filter the Array

Putting objectives and squads into deep water is probably not what you want. So invoke *f_filtermarkersbyTerrain* to remove from the array of markers all those that are mostly water.

The function uses a 'hard-coded' value that filters all markers that are more than 60% water.

Finally, choose randomly one marker in which to place a patrolling squad.

Sidebar: Documentation

```
f_filterpositionsbyTerrain

Filters (1). If the position is water, remove it from the array.

Usage : Call
Params: 1. Array of objects of type marker
Return: Array of objects of type marker
```

Placing Playable Squad

Move and track the playable groiup.

Play the Mission

Launch the mission from inside the editor by selecting 'Preview'.

Post-Mortem

If you played the mission here's what you should have seen:

- A two part briefing.
- A task for the "Destroy the mortar" objective.
- A dot that shows approximately the location of the mortar.
- A single squad patrolling near the city.
- Two guarded mortars.
- Ammo cache near the mortars with explosives.
- After destroying the mortars and the ammo cache, the task should have shown completed.
- Each time you start the mission a random city will 'host' the objective.

Technical Corner

The ideas of choosing randomly one city from an array or choosing one marker from an array are very similar. When the idea is expanded it can be thought of a design pattern for mission creators.

The pattern is in three parts:

> Generate a set of objects of the same type,
> Filter this set, i.e., choose a subset based on some attribute of the object,
> Choose one element of the set randomly.

In the example of choosing the city, the framework filters the list using the ‘Village’, ‘City’ or ‘Capital’ attributes.

When choosing a random area around a central location both the generation and filtering are provided by the designer’s code.

In this mission, the filtering criterion was ‘water zone’ but the Framework provides functions that allow filtering by many other criteria, including degree of urbanization or type of terrain.