ForceX 3D Radar EX

Version 1.0.5

Project / Scene Setup

1. Create five new layers:

- Radar_Layer_1
- Radar_Layer_2
- HUD_Layer_1
- HUD_Layer_2 (Optional : Oculus Rift)
- Radar_Contact

2. Create one new tags:

- Sub_Component

3. Create a new GameObject and name it " **_GameMgr** " and zero out its position and rotation values in the inspector and then assign both the "**FX_3DRadar_Mgr.js**" & "**FX_Faction_Mgr**" script. These scripts can be found in the "**Scripts**" folder. In the inspector go ahead and adjust the **Radar Range** value to 1000.

4. Find the prefab object "**_Radar2013**" located "**Assets → Prefabs → _Radar2013**" and assign it as a child of the "**_GameMgr**".

5. Create a new GameObject to represent your player and name it "**_Player**". Then assign your default scene camera as a child of the "**_Player**" GameObject and zero out its position and rotation values in the inspector. Change the cameras name to "**_PlayerCamera**"

6. Select your "**_GameMgr**" GameObject and in the inspector window assign the following:

A. Object assignment :

- Atlas Material = "**Radar_Atlas_Mat**" Found : **Assets→ Materials→ Radar_Atlas_Mat**
- Player **=** "**_Player**"
- Player Camera **=** "**_PlayerCamera**" Found : **_Player→ _PlayerCamera**
- Radar Camera **=** "**_RadarCamera**" Found : **_GameMgr→_Radar2013→_RadarCamera**
- Radar Object **=** "**_Radar2013**" Found : **_GameMgr→ _Radar2013**

B. Layer assignment :

- Radar Layer 1 = Radar_Layer_1
- Radar Layer 2 = Radar_Layer_2
- HUD Layer 1 = HUD_Layer_1
- HUD Layer 2 = HUD_Layer_2 (Optional : Oculus Rift)
- Radar Contact Layer = Radar_Contact

Now that the project and initial scene have been setup we will create and configure a new GameObject for the radar to target.

Creating A Target

1. Create a new GameObject and name it “**Target**” and assign it the “**FX_3DRadar_RID.js**” script. This script can be found in the “**Scripts**” folder.

That’s it you have created a target. You can setup your target any way you require.

Object Class : Defines the primary classification and will determine what “**Sub Class**” options are available.

Sub Class : Determines what texture will be assigned as the targets Radar ID.

Is Player : Determines if the Player is in control and will prevent the Players radar from displaying their own Radar ID.

Is NAV : Determines if the object will be treated as a NAV waypoint and not a typical radar contact.

Is Abandoned : Sets the objects IFF status to Abandoned.

Is Player Owned : Sets the objects IFF status to Owned & marks the object as being owned by the player.

Detectable : Determines if a target can be detected by the players Radar.

Perm Discovery : When “**Target Bounds**“ are set to “ **Show Always After Discovery** ” this determines if a target will permanently display its bounding indicators regardless of the players relation to the target.

UnDetectable Discovery Reset : When “**Target Bounds**“ are set to “ **Show Always After Discovery** ” and Perm Discovery is enabled, this determines if the target will reset it’s discovered flag when the target enters a undetectable state.

Blind Radar Override : When using Blind Radar this makes the target unable to be obstructed by any objects in the game world. Meaning that it’s will always be known to the player as long as the target is in the radars range.

Current IFF State : Displays the objects current IFF (Is Friend Foe) state. This is for feedback purposes only and cannot be changed manually through this value.

3D Radar Extended Settings

Project Scale

- Project scale is used in adjusting distance values if you are not using 1 unit = 1 meter (Unity Default)

Sound Effects

The radar system supports sound effects for the following events:

- **Select Target :** This sound will play any time a new target is selected.
- **Clear Target :** This sound will only play if a target is currently selected then cleared.
- **Warning :** This sound will play when a hostile target enters the players radars range.
- **Play Warning At Start :** This will play the warning at the start if any hostile contacts are in radar range.

Blind Radar

Blind Radar mode will allow targets to be blocked / obstructed from the player by other objects. This can be used to block a small fighter from the player if it fly's behind something like an asteroid.

- **Radar Refresh (sec) :** How often the radar will check to see if there is a LOS (Line of Sight) between the target and the player.
- **Auto Reacquire Time (sec) :** If a target is lost because of a loss of LOS, the target will be automatically reacquired if LOS is restored before this amount of time has passed.
- **Obstruction Layers :** Allows you to define layers for objects that can obstruct the players radar LOS.

Directional Indicator

When a target is selected an arrow will show up in the center of the players screen and point to the targets position. When the target is inside of the Directional indicator arrow arc then the arrow will disappear.

- **Enable Directional Indicator :** Determines if the Indicator arrow will be displayed.
- **Radius :** How far out from the center of the screen the arrow will rotate.

NAV Points

- **Display Nav In Radar :** Determines if the NAV icon will be displayed in the Players radar.
- **Arrival Distance :** Sets a distance from the current NAV that the player will be considered to have arrived at the NAV point.

Target List Settings

- **Display Target List :** Displays the players current targets as a scrolling list.

Radar Range

- **Radar Range :** Determines how far the players radar can detect potential contacts.
- **Radar Zoom Level :** Sets the players radar zoom or boost amount.

Target Lead Indicator

When a mobile target is selected this will draw a positional indicator where the player will need to shoot in order to hit the target. For this to function properly it is required that you assign the speed of your projectiles velocity to the radar manager script.

If the speed of the players projectile is known and will not change then this value can be applied in the inspector. Else if the value of the players projectile speed changes based on equipment the player may be swapping out during runtime then you will need to be able to set this value yourself through your own script. One way you can do this through your own script is:

"**GameObject.Find("_GameMgr").GetComponent(FX_3DRadar_Mgr). ProjectileVelocity = Your Value**"

Target Bounds

Target bounds will display a targets extents with indicators that display on the players HUD.

- **Bounds Show :** Determines what rules will be used to display a targets bounds indicators.
    - **Show Only In Radar Range :** This will only display the targets Bounds Indicators if the target is inside the players radar range.
    - **Show Always :** This will always display the targets Bounds Indicator. Unless the target is in an undetectable state.
    - **Show Always After Contact :** Same as **Show Always** but the target needs to be discovered first.

- **Bounds Calculation :** Determines which method will be used to calculate the object bounds.
    - **Simple** :
    - **Advanced** :

- **Bounds Size :**
    - **Dynamic Size :** The bounds size will grow or shrink depending on the size of the object on the screen.
    - **Static Size :** The bounds size will be the same no matter how large or small the object appears on the screen.

- **Limit Screen Size :** Sets the Maximum and Minimum size the bounds indicators can occupy on the screen. The calculations are performed in Viewport Space.
- **Padding :** Determines the relative size of the bounds in relation the objects actual bounds size.

- **Opacity :** Determines how opaque the bounds indicators will be.

Radar Display Setup

The following settings pertain to how the radar object will be displayed on the players screen.

- **Render As Perspective :** Enabling this will render the radar in a perspective view. Otherwise if disabled will render the radar in an orthographic view.

- **Position** : Determines the placement of the radar on the players screen.
  - **Use My Position :** Will render the radar on the screen where you decide to place it.
  - **Top Left :** Will render the radar in the top left corner of the screen.
  - **Top Right :** Will render the radar in the top right corner of the screen.
  - **Bottom Left :** Will render the radar in the bottom left corner of the screen.
  - **Bottom Right :** Will render the radar in the bottom right corner of the screen.
  - **Use Render Texture Pro :** Will render the radar out to a texture. **Unity Pro Only**

Use Render Texture Pro

Render Target : The GameObject / Mesh that will receive the Render Texture.

When using Render Texture Pro you will find that the radar ID icons will be much to small on the Render Texture. To compensate for this we must increase the size of the radar ID texture and the targets VDI (Vertical Directional Indicator) texture. This can be done with the following.

- **Radar ID Size Override :** Can increase the textures size by up to 8x.
- **Radar VDI Multiplier :** Used to increase the size of the VDI.

- **Tip :** Try these settings for a standard PC development.
  Radar ID Size = 128
  Radar VDI Multiplier = 4
  Render Texture Width & Height = 512 or 1024

Radar / HUD Display Settings

HUD Indicator : Screen Edge Padding : Sets how close to the edge of the screen the target selection box is allowed to get.

Radar Color Identifiers : These are used to define what color a target is in any given state.

HUD Override

- **HUD Offset :** Adjust the screen position for the target selection box and bounds indicators relative to the selected targets screen position. This is used to help compensate for differences between normal desktop development and development with the Oculus Rift

Keyboard Assignments

E : Closest Hostile
R : Next Hostile
F : Previous Hostile
T : Next target
Y : Previous target
U : Closest target
C : Clear target

M : Next SubComponent Target
N: Previous SubComponent Target
B: Clear SubComponent Target

L : Display Target Scrolling List
H: Display Only Hostile Contacts

Setting Up The Faction Manager For The 3D Radar

By now you should have followed the steps to get the radar system setup in your project. If not then go back and read the section on how to do this.

1. Apply the **FX_Faction_Mgr.js** script to your **"_GameMgr**" gameObject.
2. In the inspector click the Start Faction Manager button.
3. Set the number of factions by adjusting the **Factions slider's** position. Up to 32 factions can be created.
4. In the new text fields assign names for each of the faction groups. If not the default names will be used.
5. Click the **Faction Relations** button to switch to the relationship matrix window.
6. Assign a cutoff value for the Hostile and Friendly fields. Any number below the value assigned for Hostile will be considered hostile, while any number above the value assigned for Friendly will be considered to be friendly. Any number between these two values will be neutral.
7. Assign the players Faction by adjusting the slider's position.
8. Assign starting relationship values for the Player and each faction. **Optional**
9. Assign starting relationship values for each of the factions in the Faction Relationship Matrix.

Assigning Factions To Individual GameObjects

The 3D Radar stores all individual faction information inside the FX_3DRadar_RID so any gameObject with this script attached will have access to the Factions list stored in the Faction Manager. Assigning a Faction to any given gameObject is as simple as adjusting the Faction Slider's position.

Depending on the values assigned in the Faction Relation Matrix each of these gameObjects will now appear as Friendly, Neutral, or Hostile towards the player. Don't worry this doesn't just work for the player, any gameObject can compare their relationship with any other gameObject; For this we need to understand how the Faction Manager works.

Accessing The Faction Manager Through Script

How The Faction Manager Works

At its core the faction system is a Dictionary. Since a Dictionary stores information through the use of a Key, this means that in order to retrieve any information from the Dictionary we must have the Key. This leaves us with the question, what is the key? The answer to that is quite simple. The Faction Manager will assign each faction a number known as a FactionID. While this number by its self has no immediate use, but when added together with another faction's ID number, will give us the Key we need in order to look up the relationship values for these two factions in the Dictionary. Once we have the Key and access to the Key's paring value we can either simply use the returned value for our own purpose, such as checking if a faction is Hostile or Friendly, or we can update the value to something different.

Let's take a look at how this is done.

Check The Relationship Value Between Two Factions

1. Get the FactionID's for the two objects you need to compare.
- FactionID_1 = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFactionID;
- FactionID_2 = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFactionID;

2. Add both values from step 1 together. This will give you a unique key that is used to look up the relationship value for these two factions.
- var ThisRelation : int = FX_Faction_Mgr.FactionRelations[(FactionID_1 + FactionID_2)];
- The value returned is the numerical relationship for these two factions.

Change The Relationship Value Between Two Factions

1. Get the FactionID's for the two objects you need to update.
- FactionID_1 = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFactionID;
- FactionID_2 = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFactionID;

2. Add both values from step 1 together. This will give you a unique key that is used to access and change the relationship value between these two factions.
- FX_Faction_Mgr.FactionRelations[(FactionID_1 + FactionID_2)] = Your Value;

Check The Players Relationship Value With A Faction

1. Get the Faction number for the faction you want to compare.
- Faction_N = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFaction;

2. Get the value form the PlayerRelations array.
- Value = FX_Game_Mgr. PlayerRelations[Faction_N];

Change The Players Relationship Value With A Faction

3. Get the Faction number for the faction you want to compare.
- Faction_N = YourGameObject.GetComponent(FX_3DRadar_RID).ThisFaction;

4. Get the value form the PlayerRelations array.
- FX_Game_Mgr. PlayerRelations[Faction_N] = Your Value;

Changing The Players Faction

FX_Faction_Mgr.PlayerFaction = Your Value;

Changing The Others Faction

The Radar system stores the faction values for any given object other than the player in one place; this is in the FX_3DRadar_Rid.js script. The current faction value is stored in the ThisFaction : int, and the current factionID value is stored in ThisFactionID : int. These values can be updated manually by executing some code in a certain order. There is also a ThisFactionTemp value. This value is used to compare it's self against the currently assigned faction number and automatically update all faction information automatically if any change occurs to the faction value. However there may be instances where you need to update the faction number and the faction ID number manually. Below these methods are described.

Auto Update

- FX_3DRadar_RID.ThisFaction = Your Value;

Full Manual Update

- FX_3DRadar_RID.ThisFaction = Your Value;
- FX_3DRadar_RID.ThisFactionID = FX_Faction_Mgr.FactionID[ThisFaction];

Things To Avoid

One thing to note is that you cannot compare the values of two objects that are the same faction. If you are creating a function that checks the relationship values for a bunch of objects that may share the same factions as the object that is doing the checking then you will want to omit objects of the same faction.

If(FactionID_1 != FactionID_2) { Do Something }

Doing this simple check will avoid generating a Key value that does not exist in the Dictionary.