ADVANCED FPS RADAR

User Guide

v3.0

Advanced FPS Radar Overview

Advanced FPS radar is a complete out-of-the-box solution radar/minimap developed for team-based FPS games.

Advanced FPS Radar boasts a plethora of features like enemy spotting, player and objective highlighting, UAV scans, area scans, direction indicators, over 25 icons and so much more!

You can achieve similar look & functionality to the minimaps/radars in popular AAA games.

It has been developed for fast and easy integration into any custom projects, including multiplayer games.

All of the core functionality is inside 1 easily customisable script.

Any of the unused features do not take extra resources.

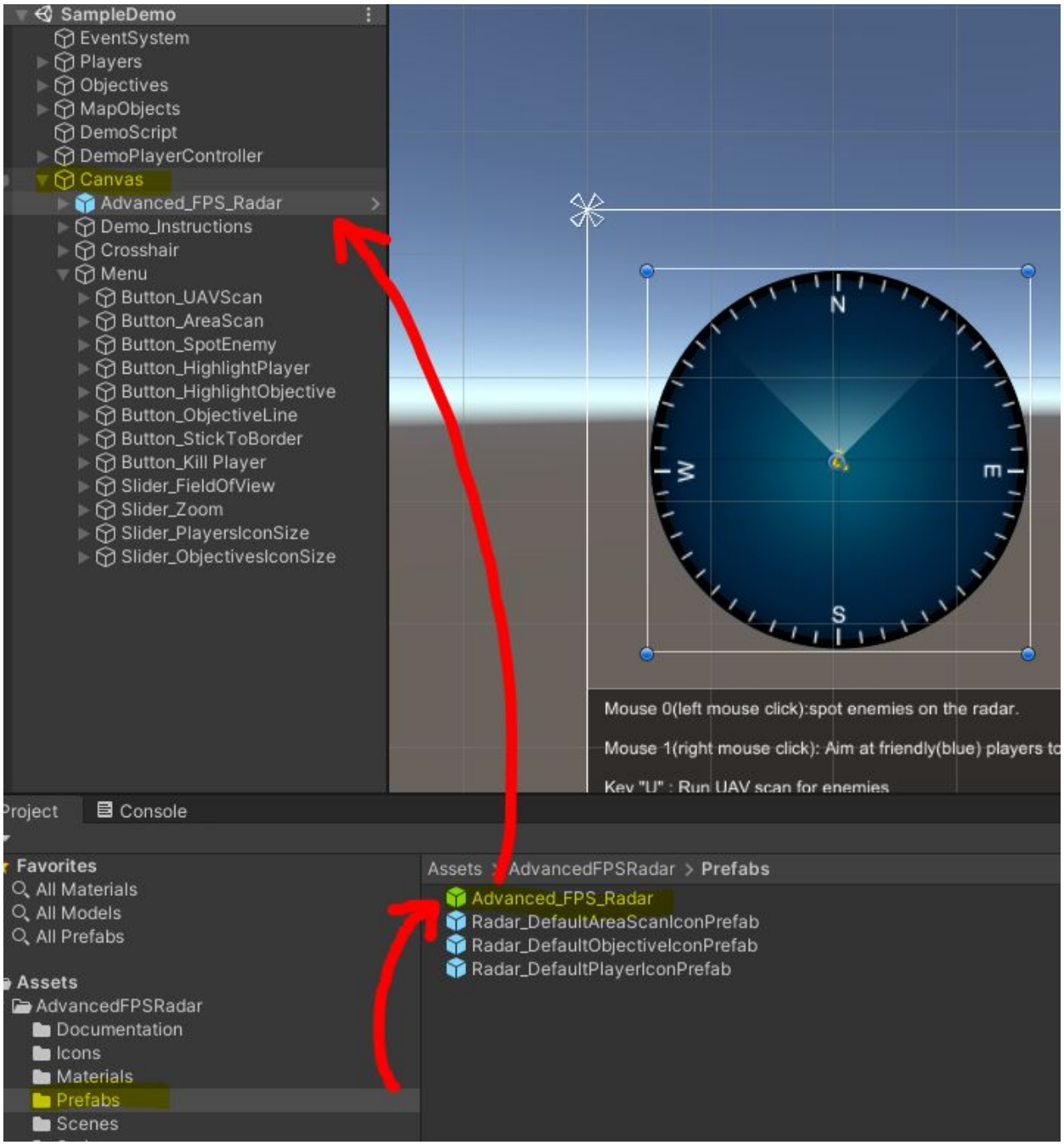
The code is well optimized for desktop and mobile platforms.

Follow the installation guide below.

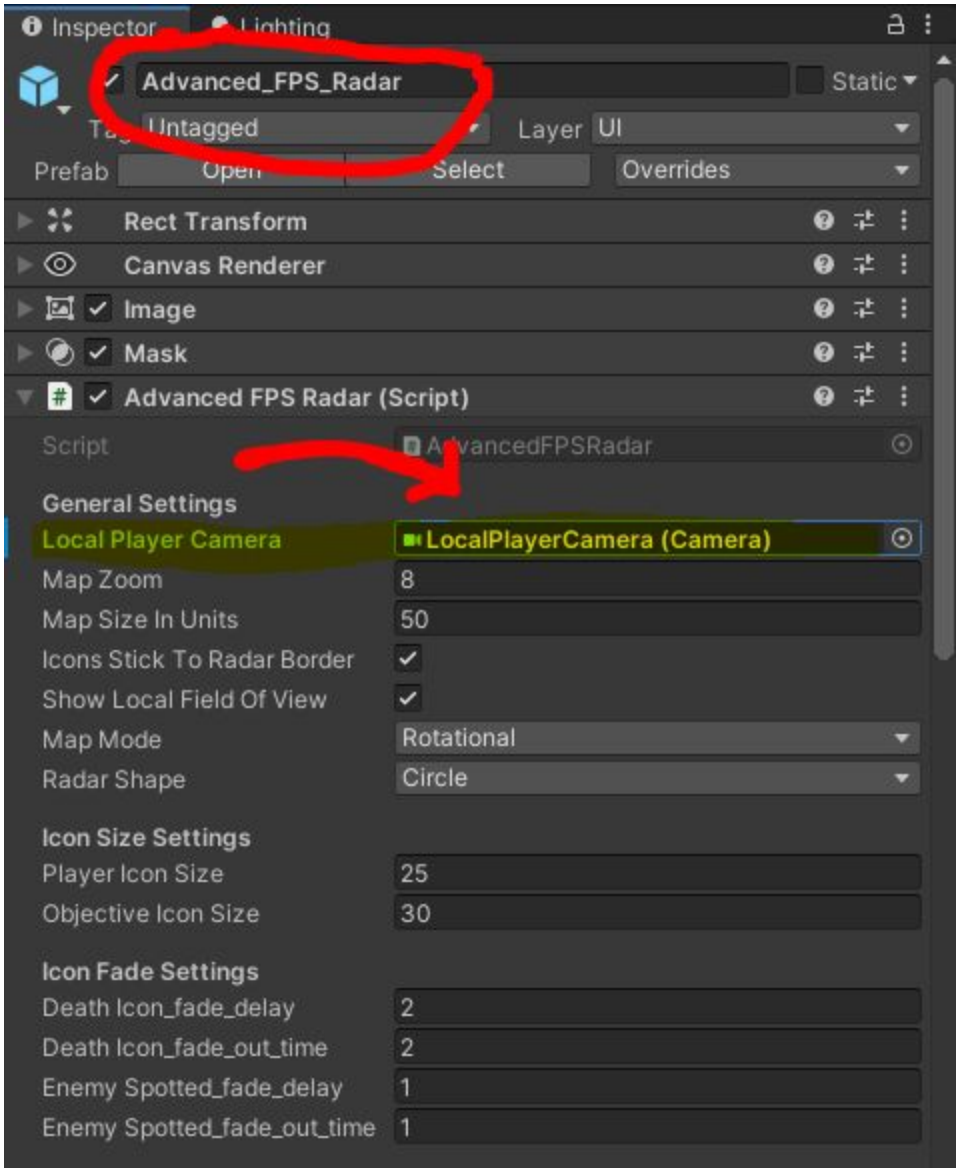
Installation Guide

1. **Import “AdvancedFPSRadar” package** into your project.

2. **Navigate to folder “AdvancedFPSRadar/prefabs” , drag and drop “Advanced_FPS_Radar” under the Canvas object inside your scene.** (*Note: Canvas must be set in Screenspace-overlay mode under Canvas Settings*).



3. On the“Advanced_FPS_Radar” prefab **assign a local player camera in the inspector window**. Also make sure all the references are assigned for transforms, pefabs and radar icons.(*Note: You can also assign local player camera thru script, using RegisterLocalPlayer(Camera _localPlayerCamera)*.



4. **Register players using RegisterRadarPlayerObject((GameObject _playerGameObject, bool _isEnemy, int _playerId)) - function.** Generally you will need to have a list of all the players and objectives inside a class or a struct,

which you can then loop through to get their unique Id-s and then register each one of them on the radar. You will need to manage yourself which ones are friend/enemy on the radar. Example of this is included in "*DemoScript.cs*".

***GameObject _playerGameObject** - is the actual GameObject of Player or Objective that the radar can track in your scene.

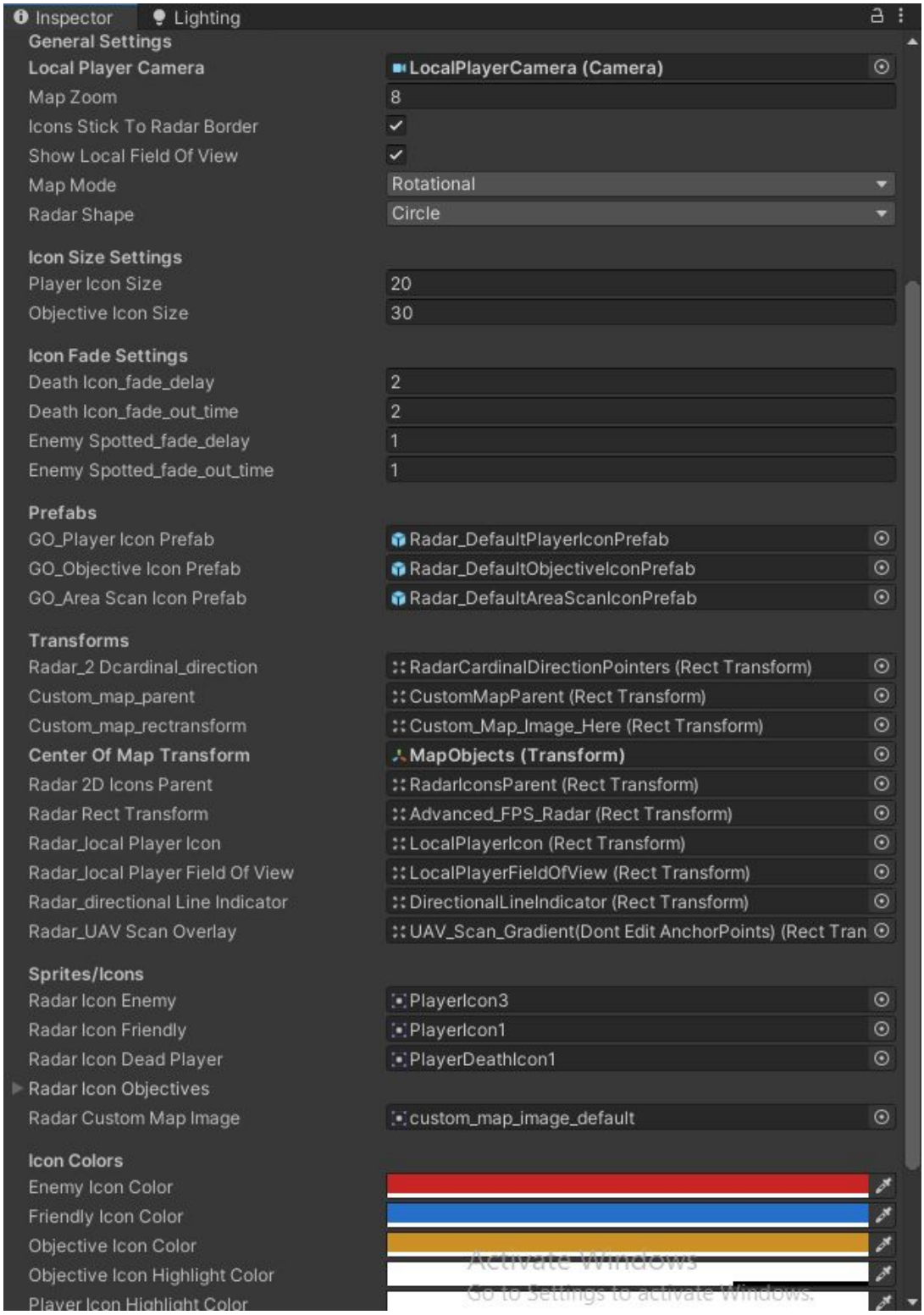
***bool _isEnemy** - you will need to set true or false based on the team. Enemy players will be rendered with a different icon and color on the radar, including the ability to spot them.

***int _playerId** - every player must have a unique Id assigned for the radar display, so you can later easily update specific players teams, colors, icons and other stuff by this player Id.

```
//Loop thrue all the friendly team players
for (int id = 0; id < friendlyGO.Length; id++)
{
    //Add friendly player to radar
    fpsRadar.RegisterRadarPlayerObject(friendlyGO[id], false, friendlyPlayerId[id]);
}

//Loop thrue all the enemy team players
for (int id = 0; id < enemyGO.Length; id++)
{
    //Add an enemy player to radar
    fpsRadar.RegisterRadarPlayerObject(enemyGO[id], true, enemyPlayerId[id]);
}
```

Inspector Window



- **LocalPlayerCamera -** assign a reference to your player camera. Without this, the radar will not work. You can either drag and drop the reference directly in your editor window or use a function RegisterLocalPlayer() to assign a player camera.
- **Map Zoom** - Zoom in and out of radar/minimap. Default value is at 8 units, but it really depends on the size of your game world. The bigger the world, the smaller the zoom. For example zoom level 8 is appropriate for a demo scene which is 50 meters/units wide.
- **Icons Stick To Radar Border** - if you check this box, all the player and objective icons will stick to the border of the radar if they move out of range from the radar/minimap(for both square and circular shapes).
- **Show Local Field Of View** - enables/disables the dynamic field of view marker for local players.
- **Map Mode(Rotiationa/Static)** - Rotational map mode rotates radar objects around the local player(the center icon), while localplayer icon stays static. Static map mode rotates only the local player icon(in the center) relative to the radar objects, while map and radar objects stay static.
- **Radar Shape(Circle/Square)** - The shape of the radar can be either full circle or perfect square. This is simply used to calculate the border area for “Icons Stick To Radar Border” - option. It does NOT visually change the shape of the radar, you will need to change the background Image of the radar/minimap yourself. There are different circular and square shaped textures included for both modes.
- **Player Icon Size** - Square size of the player icon on the radar(in pixels). Good value is 15 - 25 pixels(depending on the scale of your entire radar/minimap).
- **Objective Icon Size** - Square size of the objective icon on the radar(in pixels). Good value is 20 - 40 pixels(depending on the scale of your entire radar/minimap). It is recommended to use slightly bigger size than player icons.
- **Death Icon Fade Delay** - If _deathIconFade variable is activated for SetPlayerAliveOnRadar() function, this delay in seconds is the time before it starts to fade away the dead player on the radar.

- **Death Icon Fade Out Time** - If _deathIconFade variable is activated for SetPlayerAliveOnRadar() function, this fade option represents the time in seconds of the fading out effect of the dead player icon.
- **GO_PlayerIconPrefab** - this prefab holds the images of the players that we instantiate in the RegisterPlayerObject() function.
- **GO_ObjectiveIconPrefab** - this prefab holds the icons of the objectives that we instantiate in the RegisterRadarObjective() function.
- **GO_AreaScanIconPrefab** - this prefab holds the icons of the area scanner object that we instantiate in the RegisterRadarObjective() function.
- **Transforms** - assign references from the minimap/radar
- **Center Of Map Transform** - assign an absolute square center of your entire map. It needs to be the same center as your custom map image, as we use this transform position to calculate the positioning of the custom map image on the radar/minimap.
- **Radar Icon Enemy** - enemy icon on the minimap. Note that enemy icons should be regular circles without arrow pointers, so the player can't actually see the direction of the enemies, just positions. There are several variations of player icons in the “Icons” folder.
- **Radar Icon Friendly** - friendly icon on the minimap. Use arrow included icons to represent friendly team players.There are several variations of player icons in the “Icons” folder.
- **Enemy Icon Color** - Color of the enemy player icon on the radar. Red or orange is good here.
- **Friendly Icon Color** - Color of the friendly player icon on the radar. Green or blue is good here.
- **Objective Icon Color** - Color of the objectiver icon on the radar. Make sure it is different from player icon colors. White or yellow is good.
- **Objective Icon Highlight Color** - Color of the objectiver icon highlight(circle border around the main icon) on the radar.
- **Player Icon Highlight Color** - Color of the player icon highlight(circle border around the main icon) on the radar.

Advanced Radar Functions

public void RegisterLocalPlayer(Camera _localPlayerCamera)

Use this function to register your player camera for use in the radar/minimap calculations. This function should be used before registering any other players or objective. You can also assign the local player camera reference directly in the inspector of AdvancedFPSRadar script.

public void RegisterRadarPlayerObject(GameObject _playerGameObject, bool _isEnemy, int _playerId)

Use this function to register any friendly/enemy players to be tracked and displayed on the radar. Every player must have a unique Id assigned for other radar functionality to work(like updating player team, icon, color etc.)

public void UpdateRadarPlayerIconByTeam(int _playerId, bool _isEnemy)

Change the team of the player by player Id. bool _isEnemy is simply a switch for the team. (Note: only enemies can be spotted on the radar, by default, they are invisible on the radar, until spotted by player, Area scan or UAV scan).

public void UpdatePlayerIconSize(float _sizeInPixels)

Change the icon size of the player on the radar. Icons are square shaped so the size is with and height of the icon. The value depends on the overall size of your radar. Anything between 15 - 25 pixels is recommended.

public void UpdateRadarPlayerIconColor(int _playerId, Color _iconColor)

Change the color of the player icon by specific player id. Note: this is useful if you have “Fractions” or “Squads” in your game, where different friendly team players can have different colors.

public void HighLightPlayer(int _playerId, bool highLightActive)

Highlighting the player on the radar activates a circle around the player icon. You can change the sprite of the highlight on the “Radar_DefaultPlayerIconPrefab”, which is a child object called “HighlightBorder”. This can be useful to display that the player takes damage or for example is currently speaking over VOIP in a multiplayer match. You have to manage switching on/off this functionality per player Id by yourself as there are obviously many specific use cases for this.

public void SetPlayerAliveOnRadar(int _playerId, bool _isAlive, bool _deathIconFade)

Kill a player on the radar or set them alive again. Dead players have different icons(and possibly colors). You can choose to fade out the death icon or let it simply switch off after a certain time(by the settings you have in the inspector window for DeathIcons). It is recommended to enable _deathIconFade for smoother experience.

public void SpotEnemyOnRadar(int _spottedPlayerId)

By default, enemies on the radar are invisible. You can spot the enemy by its player Id, which makes the enemy player icon visible on your radar for a short period of time(according to your settings of “enemySpotted_fade_delay” and “enemySpotted_fade_out_time”).

public void RemoveRadarIconByPlayerId(int _playerId)

Removes the player icon from the radar by destroying the actual radar icon gameObject and removing the player from the list of radar objects.

public void RegisterRadarObjective(GameObject _objectiveGameObj, Sprite _objectiveIcon, int _objectiveId)

Use this function to register any objectives to be tracked and displayed on the radar. Every objective must have a unique Id assigned for other radar functionality to work(like updating icon, color, size etc.). Assuming you have more than 1 objective and every objective has a different icon, you should assign the sprite of the objective icon here yourself or use the list of icons you have assigned in the inspector under the “radarIconObjectives”.

public void UpdateRadarObjectiveIconColor(int _objectiveId, Color _objectiveIconColor)

Change the color of the objective icon by specific objective id. Note: this is useful if you have flags in your game, where the objective has different states, like defend, capture etc, so you can change the color of each flag by its current state.

public void HighLightObjective(int _objectiveId, bool highLightActive)

Highlighting the objective on the radar activates a circle around the player icon. You can change the sprite of the highlight on the “Radar_DefaultObjectiveIconPrefab”, which is a child object called “HighlightBorder”.

public void UpdateRadarObjectiveIcon(int _objectiveId, Sprite _objectiveIcon)

Change the objective icon sprite by objective Id. Note: this is useful if you have flags or other similar objectives in your game, where the objective has different states, like defend, capture etc, so you can change the icon of each flag by its current state.

public void UpdateObjectiveIconSize(float _sizeInPixels)

Change the size of all objective icons on the radar.

public void SetDirectionalIndicatorAtObjective(bool _active, int _objectiveId)

You can enable and disable a directional line indicator from your local player(center of radar) to the specific objective by its Id. If you simply want to disable the line indicator on the radar/minimap, just set the _active boolean to false.

public void StartUAVScan(float _scanLengthInSeconds)

UAV scan is a flat line moving from the top of your radar to the bottom, highlighting all the enemies on your radar/minimap for a breath period of time(according to your enemySpotting settings). _scanLengthInSeconds is the time it takes for the full UAV scan to finish. Recommended time here is anywhere between 3 - 10 seconds.

public void StartEnemyAreaScanAtPosition(GameObject _AreaScannerGameObj, float _scanLength, float _scanCircleRadius)

Enemy scan is a circular area inside your radar that has a pulsing effect and scans for enemy players within the radius of the pulsing circle. Area scan must be an actual gameObject inside your scene so we can track the position of it on the radar. _scanLength is the time in seconds that the area scan lasts. _scanCircleRadius is the maximum radius of the area/circle(relative to the size of 100 units by default, so adjust the radius value according to your needs as it does not represent the actual units of your game world).

Questions?

Email: rando.tkatsenko@gmail.com