3D Radar Builder

DaiMangou.ProRadarBuilder.Editor.MiniMapModule Class Reference

Minimap Module

Public Member Functions

- Sprite **MaskSprite** ()
  generates are sprite specificially used for the mask layer of the Radar
- Mesh **ProceduralMapQuad** ()

Public Attributes

- **MapType mapType** = **MapType.Realtime**
  Choose between Realtime minimap or a stati minimap
- Sprite **MapTexture**
  Texture to be used for static minimaps
- bool **generated**
  Check if the map has been generated
- bool **calibrate**
  Determine if the static minimap is being calibrated
- GameObject **Map**
  the objet which will use the Map texture and Masked Material
- GameObject **Mask**
  the object which will use the mask material
- float **SavedSceneScale**
  Cashe of the SceneScale vlaue
- float **MapScale** = 1
  The value set during calibratin of ststic minimap
- float **SavedMapScale**
  Cashe of the MapScale vlaue
- float **Scalingfactor**
  Determines by what rate the minmap is scales at rintime
- Material **MapMaterial**
  Masked material
- Material **MaskMaterial**
  Mask Material
- LayerMask **layer**
  The layer on which the minimap will be rendered
- RenderTexture **renderTexture**
  the RenderTexture to be used with the realtime minimap
- Camera **RealtimeMinimapCamera**
  The camera reading the RenderTexture for the Minimap
- float **CameraHeight**
  the position of the RealtimeMinimapCamera in the Y axis
- int **OrderInLayer** = -1

the order in layer of the blip

DaiMangou.ProRadarBuilder.Editor.OptimizationModule Class Reference

Options for optimizing the radars functions

Public Attributes

- int **poolSize**
  pool size for objects you wish to track
- bool **SetPoolSizeManually** = false
  Determines if the blip will be using pooling
- **ObjectFindingMethod objectFindingMethod** = **ObjectFindingMethod.Recursive**
  Options for usng two diferent object finding methods
- bool **RemoveBlipsOnTagChange**
  if true , blips will be removed if the object they track has lost its original tag
- bool **RemoveBlipsOnDisable**
  if true , blips will be removed if the object they track has been disabled
- bool **RequireInstanceObjectCheck**
  if true and you are using Recursive optimization method then you can call _3DRadar.radar3D.doInstanceObjectCheck() trigger object search;
- bool **RecalculatePoolSizeBasedOnFirstFoundObjects**
  By setting this to true, you can ensure that evne if your pool value at atart is greater then the actual amount of objects that can be found , your pool value will be reset to the amount of objects found ao that recusrsive searching is avoided

---

DaiMangou.ProRadarBuilder.Editor.RadarBlips3D Class Reference

Public Attributes

- bool **DoRemoval** = false
  Tell the blip to do a removal of blips if the Recursive optimization method is used
- bool **Instanced**
  checks if all blips have ben instanced
- bool **IsActive**
  check if the blip is set turned on or off
- bool **ShowBLipSettings**
  INTERNAL USE ONLY
- bool **ShowSpriteBlipSettings**
  INTERNAL USE ONLY
- bool **ShowMeshBlipSettings**
  INTERNAL USE ONLY
- bool **ShowPrefabBlipSettings**
  INTERNAL USE ONLY

- bool **IsTrackRotation**
  Determines if the blip will be tracking the rotation of its target
- bool **BlipCanScleBasedOnDistance**
  Determines if th blips can scale by distance
- bool **ShowTrackingLineSettings**
  INTERNL USE ONLY
- bool **UseTrackingLine**
  Determines if we should use tracking lines or not.
- bool **UseBaseTracker**
  Determines if we should use basetrackers or not
- bool **ShowBaseTrackerSettings**
  INTERNAL USE ONLY
- bool **lockX**
  Determines if the X rotation of the tracked object should be locked to 0
- bool **lockY**
  Determines if the Y rotation of the tracked object should be locked to 0
- bool **lockZ**
  Determines if the Z rotation of the tracked object should be locked to 0
- bool **UseLOD**
  Determines if the mesh blip will use a the Radar Builder LOD system
- bool **ShowLODSettings**
  INTERNAL USE ONLY
- bool **ShowGeneralSettings**
  INTERNAL USE ONLY
- bool **ShowAdditionalOptions**
  INTERNAL USE ONLY
- bool **AlwaysShowBlipsInRadarSpace**
  determines if the blip should always remeing inside the radar
- bool **ShowLowMeshSetings**
  INTERNAL USE ONLY
- bool **ShowMediumMeshSettings**
  INTERNAL USE ONLY
- bool **ShowHighMeshSettings**
  INTERNAL USE ONLY
- bool **ShowOptimizationSettings**
  INTERNAL USE ONLY
- bool **SmoothScaleTransition**
  if you are using Always Show and Scale By Distance , this will ensure that you have a smooth ttansition from the moment your blip passes the Tracking Bounds to the moment is is scales to its minimaum scale
- Sprite **icon** = new Sprite()
  The blip icon if the blip is a sprite
- Sprite **BaseTracker** = new Sprite()
  The base tracker sprite
- Transform **prefab**

Prefab blip

- string **State** = ""
  INTERNAL USE ONLY
- string **Tag** = "Untagged"
  INTERNAL USE ONLY
- Material **SpriteMaterial**
  The material used for the sprite blip
- Material **TrackingLineMaterial**
  The material used for the tracking line ///
- Material **BaseTrackerMaterial**
  The material used for the base tracker
- Mesh **mesh**
  The mesh blip mesh when LOD is not active
- Mesh **Low**
  The low poly mesh when LOD is active
- Mesh **Medium**
  The medium poly count mesh when the LOD is active
- Mesh **High**
  The high poly count mesh when the LOD is active
- Material[] **MeshMaterials** = new Material[1]
  All mesh materials usd by the Mesh
- Color **colour** = new Color(1F, 0.6F, 0F, 0.8F)
  THe colour of the material
- Color **TrackingLineStartColour** = new Color(1F, 0.435F, 0F, 0.5F)
  The colour start of the tracking line
- Color **TrackingLineEndColour** = new Color(1F, 0.435F, 0F, 0.5F)
  The end colour of the tracking line
- Color **BaseTrackerColour** = new Color(1F, 0.435F, 0F, 0.5F)
  The colour ued by the base tracker material
- float **BlipSize** = 1
  The size of the blip
- const float **DynamicBlipSize** = 0.01f
  The default minimum scale of the blip
- float **BlipMinSize** = 0.5f
  The minimum size of the blip
- float **BlipMaxSize** = 1
  The maximum size of the blip
- float **TrackingLineDimention** = 0.02F
  The width of the tracking line
- float **LowDistance**
  The distance at which the LOW mesh will replace the current mesh of the mesh blip
- float **MediumDistance**
  The distance at which the MEDIUM mesh will replace the current mesh of the mesh blip
- float **HighDistance**

The distance at which the HIGH mesh will replace the current mesh of the mesh blip

- float **BaseTrackerSize** = 0.5f
  The scale of th base tracker
- int **NumberOfBLips**
  INTERNAL USE ONLY
- int **count**
  INTERNAL USE ONLY
- int **MatCount** = 1
  INTERNAL USE ONLY
- int **Layer** = 0
  INTERNAL USE ONLY
- List< GameObject > **TrackingLineObject** = new List<GameObject>()
  A list of All tracking lines
- List< GameObject > **gos** = new List<GameObject>()
  A list of the objects being tracked
- List< Transform > **RadarObjectToTrack** = new List<Transform>()
  A list of the actual blips you see in your radar
- List< GameObject > **BaseTrackers** = new List<GameObject>()
- **CreateBlipAs _CreateBlipAs**
  Determines what the blip should be created as , prefab or sprite
- int **ObjectCount** = -1
  records the amount of tracked objects in the radr far this blip type
- int **OrderInLayer** = 1
  the order in layer of the blip
- SortingLayer **sortingLayer**
  Sorting layer of the sprite blip
- **OptimizationModule optimization** = new **OptimizationModule**()
  Methods of optimizing radar performance

DaiMangou.ProRadarBuilder.Editor.RadarCenterObject3D Class Reference

Public Attributes

- bool **Instanced**
  checks if all blips have ben instanced
- bool **IsActive**
- bool **ShowBLipSettings**
  INTERNAL USE ONLY
- bool **ShowSpriteBlipSettings**
  INTERNAL USE ONLY
- bool **ShowMeshBlipSettings**
  INTERNAL USE ONLY

- bool **ShowPrefabBlipSettings**
  INTERNAL USE ONLY
- bool **IsTrackRotation**
  Determines if the blip will be tracking the rotation of its target
- bool **lockX**
  Determines if the X rotation of the tracked object should be locked to 0
- bool **lockY**
  Determines if the Y rotation of the tracked object should be locked to 0
- bool **lockZ**
  Determines if the Z rotation of the tracked object should be locked to 0
- bool **ShowGeneralSettings**
  INTERNAL USE ONLY
- bool **AlwaysShowCenterObject**
  Determines if the enter object or center blip should alwats be shown in th radar
- bool **CenterObjectCanScaleByDistance**
  Determines if the center object (center blip) can scale by distance
- bool **ShowAdditionalOptions**
  INTERNAL USE ONLY
- bool **ShowTrackingLineSettings**
  INTERNL USE ONLY
- bool **UseTrackingLine**
  Determines if we should use tracking lines or not.
- bool **UseBaseTracker**
  Determines if we should use basetrackers or not
- bool **ShowBaseTrackerSettings**
  INTERNAL USE ONLY
- bool **SmoothScaleTransition**
  if you are using Always Show and Scale By Distance , this will ensure that you have a smooth ttansition from the moment your blip passes the Tracking Bounds to the moment is is scales to its minimaum scale
- Sprite **icon** = new Sprite()
  The blip icon if the blip is a sprite
- Sprite **BaseTracker** = new Sprite()
  The base tracker sprite
- Transform **prefab**
  Prefab blip
- string **State** = ""
  INTERNAL USE ONLY
- string **Tag** = "Player"
  INTERNAL USE ONLY
- Material **SpriteMaterial**
  The material used for the sprite blip
- Material **TrackingLineMaterial**
  The material used for the tracking line ///
- Material **BaseTrackerMaterial**

The material used for the base tracker

- Mesh **mesh**
  The mesh blip mesh when LOD is not active
- Material[] **MeshMaterials** = new Material[1]
  All mesh materials usd by the Mesh
- Color **colour** = new Color(1F, 0.435F, 0F, 0.5F)
  THe colour of the material
- Color **TrackingLineStartColour** = new Color(1F, 0.435F, 0F, 0.5F)
  The colour start of the tracking line
- Color **TrackingLineEndColour** = new Color(1F, 0.435F, 0F, 0.5F)
  The end colour of the tracking line
- Color **BaseTrackerColour** = new Color(1F, 0.435F, 0F, 0.5F)
  The colour ued by the base tracker material
- float **BlipSize** = 1
  The size of the blip
- float **TrackingLineDimention** = 0.2f
  The width of the tracking line
- const float **DynamicBlipSize** = 0.01f
  The default minimum scale of the blip
- float **BlipMinSize** = 0.5f
  The minimum size of the blip
- float **BlipMaxSize** = 1
  The maximum size of the blip
- float **BaseTrackerSize** = 0.5f
  The scale of th base tracker
- int **Layer** = 0
  INTERNAL USE ONLY
- int **OrderInLayer** = 1
  the order in layer of the blip
- GameObject **CenterBlip**
  The blip at the center of the radar
- Transform **CenterObject**
  The object being tracked to and used to represent the CenterBlip
- int **MatCount** = 1
  INTERNAL USE ONLY
- **CreateBlipAs _CreateBlipAs**
  Determines what the blip should be created as , prefab or sprite
- GameObject **BaseTrackerObject**
  Object which will sit on the y plane of the radar at all time
- GameObject **TrackingLine**
  Line wich will indicate distance in height from the centerobject to the radar </summary

DaiMangou.ProRadarBuilder.Editor.RadarDesign3D Class Reference

Public Attributes

- float **RadarDiameter** = 1
  This is the Diameter of the radar, this value will directly change the scale of the Radars child object "Designs" once UseSceneScale is false
- float **SceneScale** = 100.0f
  This is the amound of the scene that the radar is able to 'see' in order to collect dats on things to track and display
- float **TrackingBounds** = 1
  The range in which all blips can be shown in the radar
- float **InnerCullingZone** = 0f
  The diameter of the zone at the center of the radar in which all blips will ce culled
- float **RadarRotationOffset** = 0f
  INTERNAL USE ONLY
- const float **ConstantRadarRenderDistance** = 4
  Do not replace this value
- float **xPadding**
  The padding on the x and Y axis of the radar system
- **RadarPositioning radarPositioning** = **RadarPositioning.Snap**
  Determins if the radar will ise Manual position or Snap Positioning
- **SnapPosition snapPosition** = SnapPosition.BottomLeft
  Determines where in scren space the radar system will be positioned
- **FrontIs frontIs** = **FrontIs.North**
  Determining what defines the forward facing position of the radar
- Rect **RadarRect**
  INTERNAL USE ONLY
- int **Count** = 0
  INTERNAL USE ONLY
- int **DesignsCount** = 0
  INTERNAL USE ONLY
- bool **UseLocalScale**
  Determines if we should use the scale of the Radar "Designs" child object instead of the RadarDiameter
- bool **Visualize** = true
  INTERNAL USE ONLY
- bool **LinkToTrackingBounds**
  Determines if the tracking bounds values will always be the same as
- bool **ShowScaleSettings**
  INTERNAL USE ONLY
- bool **ShowRenderCameraSettings**
  INTERNAL USE ONLY
- bool **ShowPositioningSettings**

INTERNAL USE ONLY

- bool **IgnoreDiameterScale** = false
  When true, the radar ; diameter (Sale of the Radars "Designs" child object) when scales to a vlue greater or less than one will not prompt the radar system to reposition itslf automatically to maintain a correct position in screen space
- bool **ManualCameraSetup**
  INTERNAL USE ONLY
- bool **UseMainCamera**
  determines if we will be using the gameobject in the scne with the tag "Main Camera"
- bool **_3DSystemsWithScreenSpaceFunction**
  Determines if the 3D Radar will also be using the screen space system
- bool **_3DSystemsWithMinimapFunction**
  Determines if the radar can also be a minimap
- bool **ShowMinimapSettings**
  INTERNAL USE ONLY
- GameObject **DesignsObject**
  INTERNAL USE ONLY
- Camera **camera**
  The camera which will be the camera your player views the world through at any time
- Camera **renderingCamera**
  The camera whuch will only render radar systems, (These camera are automatically created for you)
- string **CameraTag** = "MainCamera"
  INTERNAL USE ONLY
- List< **RotationTarget** > **RotationTargets** = new List<**RotationTarget**>()
  The list of Rotation targets
- Vector3 **Pan** = new Vector3()
  The pan of the blips in the radar

DaiMangou.ProRadarBuilder.Editor.RotationTarget Class Reference

Public Attributes

- bool **ShowDesignSetings**
  called only from editor , and is not necessary at runtime
- bool **UseY**
  When true , the z rotation will be the same as the Y rotation
- bool **FreezeX**
  Freeze rotation around particular axis
- float **RotationDamping**
  Damping used to control rotation of particular design layer
- string **tag**
  the string tag you define

- string **FindingName**
  the name of the object you wish to find
- string **InstancedObjectToTrackBlipName**
  The name of the instanced object you wish to target
- string **InstancedTargetBlipname**
  the name of the instanced blip you wish to track
- **Rotations rotations**
  Selection between Inverse rotation and Proportional rotation
- GameObject **TargetedObject**
  This may be a blip or any other object in scene
- GameObject **Target**
  the object whose rotation you wish to target
- **TargetObject ObjectToTrack** = **TargetObject.ThisObject**
  Selection of the way in which you wish to select and object
- **TargetBlip target** = **TargetBlip.ThisObject**
  The blip you wish to target
- **RetargetRotation RetargetedRotation** = **RetargetRotation.none**
  Determining what axis value we wish to pass to another axis value
- **valueState ValueState** = **valueState.positive**
  A selection between positive and negative
- float **AddedRotation** = 90
  this rotation value is usually used when using sprites
