HPJ Pathfinding Pro

Table of Contents

Introduction

Check out the Quick-Start Guide to help you setup!

HPJ Pathfinding Package is a pathfinding solution that is based on the Jump Point Search algorithm.
HPJ Allows for multiple maps, w/ a large amount of unit & obstacles at speeds much faster than A*.
HPJ includes basic Navigation Agents, and Navigation Obstacles that work w/ an easy setup.
HPJ plans to add more functionality, QOL features, map types, and pathfinding types in the future.

Script Reference

Scripts

1.Presentation

1.Manager

NavigationManager.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

NavigationManager

```
public class NavigationManager
```

The Manager used to manage the Map sets, Obstacles as Gameobjects, Navigation Agents as Gameobjects

Classes

AStarBridgeData

```
private struct AStarBridgeData
```

Constructors

AStarBridgeData

```
public  AStarBridgeData(
     int MapInstanceID,
     int ThisID,
     int EndCost)
```

Variables

ID

```
public int ID
```

FromMapInstanceID

```
public int FromMapInstanceID
```

FCost

```
public int FCost
```

GCost

```
public int GCost
```

HCost

```
public int HCost
```

Variables

_mapSets

```
[SerializeField]
protected List<MapSet> _mapSets
```

_activeSets

```
[NonSerialized]
protected List<MapSet> _activeSets
```

_bridges

```
[SerializeField]
protected List<MapBridge> _bridges
```

A List of all the bridges

_seams

```
[SerializeField]
protected List<MapSeam> _seams
```

A List of all the bridges

ManagerSettings

```
public ManagerSetting ManagerSettings
```

The Navigation Manager Settings

MapSeamSettings

```
public SeamSettings MapSeamSettings
```

Seam settings for the Navigation Manager

Threads

```
public Thread[] Threads
```

Map Simulatin Threads

ExtraSimulatinThread

```
public Thread ExtraSimulatinThread
```

Extra Simulation Thrad

_parallelOptions

```
protected ParallelOptions _parallelOptions
```

The options for parrellism in the map simulation threads

Painter

```
public PainterSetting Painter
```

Painting Settings

AutoPainter

```
public AutoPainterSetting AutoPainter
```

Auto-Painter Settings

Instance

```
public static NavigationManager Instance
```

GizmosSettings

```
public GizmosSetting GizmosSettings
```

Navigation Manager Gizmos Settings

HelperStopwatch

```
protected System.Diagnostics.Stopwatch HelperStopwatch
```

Stopwatch to keep track of the helper update

RandomValidPosition

```
protected System.Random RandomValidPosition
```

Properties

MapSets

```
public List<MapSet> MapSets
     { get;
     }
```

A List of the maps

ApplicationPlaying

```
protected bool ApplicationPlaying
     { get;
     set; }
```

Bridges

```
public List<MapBridge> Bridges
     { get;
     }
```

Seams

```
public List<MapSeam> Seams
     { get;
     }
```

Running

```
public bool Running
     { get;
     protected set; }
```

Whether the map simulation threads are running

HelperRunning

```
public bool HelperRunning
     { get;
     protected set; }
```

Whether the map simulation helper thread is running

Logs

```
public List<string> Logs
     { get;
     protected set; }
```

Log list for callback from the map simulation Threads

Warnings

```
public List<string> Warnings
     { get;
     protected set; }
```

Errors

```
public List<string> Errors
     { get;
     protected set; }
```

Baking

```
public bool Baking
     { get;
     set; }
```

Whether the Auto-Painter is baking in the tiles right now or not

Initialized

```
public bool Initialized
     { get;
     set; }
```

Obstacles

```
public List<NavigationObstacle> Obstacles
     { get;
     protected set; }
```

List of all the obstacles that are currently active

Agents

```
public List<NavigationAgent> Agents
     { get;
     protected set; }
```

List of all the agents that are currently active

AttemptedBridge

```
public MapBridge AttemptedBridge
     { get;
     set; }
```

Current Bridge that is being attempted

SimulationAgents

```
public Dictionary<ushort,SimAgent> SimulationAgents
     { get;
     protected set; }
```

Dictionary of all the simulation Agent

SimulationAgentList

```
public List<SimAgent> SimulationAgentList
     { get;
     protected set; }
```

List of all the simulation agents

HelperTargetPeriod

```
public long HelperTargetPeriod
     { get;
     set; }
```

The Helper threads target period in ms

HelperTargetDeltaTime

```
public float HelperTargetDeltaTime
     { get;
     set; }
```

The Helper threads target period in sec

HelperDeltaTime

```
public float HelperDeltaTime
     { get;
     protected set; }
```

The Helper threads Delta Time of the last frame

Methods

SetDestination

```
public void SetDestination(
     NavigationJob Job)
```

Used to send a Navigation Job Request to the simulation

AddNavigationOrder

```
internal void AddNavigationOrder(
     NavigationOrder newOrder)
```

Adds a navigation order for the Navigation Agents

SetMapBridges

```
public void SetMapBridges()
```

Initilaizes all the map bridges

GetDesiredBridgeOrSeam

```
public bool GetDesiredBridgeOrSeam(
     IntVector3 start,
     MapSet startingMap,
     MapSet endingMap,
     out MapBridge ClosestBridge,
     out MapSeam ClosestSeam,
     out int BridgeScore,
     out int SeamScore)
```

Gets a desired bridge or seam for a position

GetDesiredBridge

```
public MapBridge GetDesiredBridge(
     IntVector3 start,
     MapSet startingMap,
     MapSet endingMap)
```

Gets a desired bridge for a position

GetDesiredSeam

```
public MapSeam GetDesiredSeam(
     IntVector3 start,
     MapSet startingMap,
     MapSet endingMap)
```

Gets a desired seam for a position

AStarBridgeNode

```
public MapBridge AStarBridgeNode(
     MapSet StartMap,
     MapSet EndMap,
     IntVector3 Start)
```

Used A* Node Pathfinding to find a set of bridge to a ending map

GetNextMap

```
private int GetNextMap(
     Dictionary<int, AStarBridgeData> sortedList,
     MapSet EndMap,
     MapSet StartMap)
```

GetLowestFCostMapInstanceID

```
private int GetLowestFCostMapInstanceID(
     List<int> openList,
     MapSet endMap)
```

GetMapDistance

```
private int GetMapDistance(
     MapSet Map1,
     MapSet Map2)
```

BridgeEventuallyLeadsToMap

```
public bool BridgeEventuallyLeadsToMap(
     MapBridge Bridge,
     MapSet CurrentMap,
     MapSet EndingMap,
     List<MapSet> ClosedList)
```

AStarSeamNode

```
public MapSeam AStarSeamNode(
     MapSet StartMap,
     MapSet EndMap,
     IntVector3 Start)
```

Used A* Node Pathfinding to find a set of seams to a ending map

SeamEventuallyLeadsToMap

```
public bool SeamEventuallyLeadsToMap(
     MapSeam Seam,
     MapSet CurrentMap,
     MapSet EndingMap,
     List<MapSet> ClosedList)
```

ValidBridge

```
public bool ValidBridge(
     MapBridge Bridge)
```

Checks whether a specific bridge is valid

ValidateBridge

```
private void ValidateBridge(
     MapBridge bridge)
```

AddDebugLog

```
public void AddDebugLog(
     string Log,
     SimulationDebugLog LogType)
```

Logs all the callbacks from the simulation

AutoPaint

```
public void AutoPaint()
```

Autopaints a givent map by the auto painter settings

BakeTiles

```
protected virtual IEnumerator BakeTiles()
```

Bakes all the tiles from the Auto-Painter

CreateSeam

```
public void CreateSeam()
```

Creates a seam from the seam settings. Editor Only

ClearMap

```
public void ClearMap()
```

Clears all maps of all the painted tiles

ClearMap

```
public void ClearMap(
     int MapIndex)
```

Clears a map of all the painted tiles

Awake

```
protected void Awake()
```

Update

```
private void Update()
```

FixedUpdate

```
public void FixedUpdate()
```

OnDestroy

```
private void OnDestroy()
```

OnDrawGizmosSelected

```
private void OnDrawGizmosSelected()
```

SaveMaps

```
public void SaveMaps()
```

Saves all the maps for the current scene

LoadMaps

```
public void LoadMaps()
```

Load all the maps for the current scene

LoadMaps

```
protected void LoadMaps(
     List<MapSet> NewList)
```

Load all the maps for the current scene and puts them in the list

UpdateThread0

```
public void UpdateThread0()
```

Update0

```
public void Update0(
     MapSet Map)
```

UpdateThread1

```
public void UpdateThread1()
```

Update1

```
public void Update1(
     MapSet Map)
```

UpdateThread2

```
public void UpdateThread2()
```

Update2

```
public void Update2(
     MapSet Map)
```

UpdateThread3

```
public void UpdateThread3()
```

Update3

```
public void Update3(
     MapSet Map)
```

UpdateThread4

```
public void UpdateThread4()
```

Update4

```
public void Update4(
     MapSet Map)
```

UpdateThread5

```
public void UpdateThread5()
```

Update5

```
public void Update5(
     MapSet Map)
```

UpdateThread6

```
public void UpdateThread6()
```

Update6

```
public void Update6(
     MapSet Map)
```

UpdateThread7

```
public void UpdateThread7()
```

Update7

```
public void Update7(
     MapSet Map)
```

UpdateThread8

```
public void UpdateThread8()
```

Update8

```
public void Update8(
     MapSet Map)
```

UpdateThread9

```
public void UpdateThread9()
```

Update9

```
public void Update9(
     MapSet Map)
```

SimulationHelperUpdate

```
public void SimulationHelperUpdate()
```

This helper is used to run the simulation agents and the agent avoidence

HelperUpdate

```
protected virtual void HelperUpdate()
```

The main update section of the simulation helper update. Removed this section for clarity and ease of use

GetTileWorldIndexCenter

```
private Vector3 GetTileWorldIndexCenter(
     int x,
     int y,
     MapSet map)
```

Gets the tile world index center for a specific map

CircleOnMap

```
internal bool CircleOnMap(
     MapSet set,
     Vector3 newPosition,
     float radius)
```

Checks to see if any point of a circle is on a specific map

RectOnMap

```
internal bool RectOnMap(
     MapSet set,
     PositionSquareRect newPositionRect,
     float HeightCheck)
```

Checks to see if any point of a rectangle is on a specific map

GetMapAtTileWorldPosition

```
public MapSet GetMapAtTileWorldPosition(
     IntVector3 intVector3)
```

Get the best map at a world tile index. Can be NULL

GetMapAtWorldPosition

```
public MapSet GetMapAtWorldPosition(
     Vector3 position)
```

Get the best map at a world position. Can be NULL

AddObstacle

```
public void AddObstacle(
     NavigationObstacle navigationObstacle)
```

Adds the Obstacle to the Manager

RemoveObstacle

```
public void RemoveObstacle(
     NavigationObstacle navigationObstacle)
```

Removes the Obstacle to the Manager

AddAgent

```
public void AddAgent(
     NavigationAgent agent)
```

Adds the Agent to the Manager

AddSimAgent

```
public void AddSimAgent(
     SimNavigationAgent agent)
```

Adds the simulation Agent to the Manager

RemoveAgent

```
public void RemoveAgent(
     NavigationAgent agent)
```

Removes the Agent to the Manager

RemoveSimAgent

```
public void RemoveSimAgent(
     SimNavigationAgent agent)
```

Removes the simulation Agent to the Manager

ValidPoint

```
public bool ValidPoint(
     Vector3 hitPosition,
     MapSet set)
```

Checks whether a point is valid on a map

ValidPoint

```
public bool ValidPoint(
     Vector3 hitPosition,
     out MapSet HitMap)
```

Checks whether a point is valid on a any map

ValidPoint

```
public bool ValidPoint(
     IntVector3 hitWorldTilePosition,
     MapSet set)
```

Checks whether a point is valid on a map

GetTileWorldPosition

```
public IntVector3 GetTileWorldPosition(
     Vector3 hitPosition)
```

Get the world tile index from a position

GetTileWorldPosition

```
public Vector3 GetTileWorldPosition(
     IntVector3 tilePosition)
```

Get the position world tile index from a world tile index

GetTileCenterWorldPosition

```
public Vector3 GetTileCenterWorldPosition(
     Vector3 hitRoughPosition,
     MapSet Map)
```

Gets the tile index center position from a position

GetTileCenterWorldPosition

```
public Vector3 GetTileCenterWorldPosition(
     IntVector2 mapTileIndex,
     MapSet Map)
```

Gets the tile index center position from a map tile index

GetTileCenterWorldPosition

```
public Vector3 GetTileCenterWorldPosition(
     IntVector3 worldTileIndex,
     MapSet Map)
```

Gets the tile index center position from a world tile index

GetTileCenterWorldPosition

```
public Vector3 GetTileCenterWorldPosition(
     IntVector3 worldTileIndex)
```

Gets the world tile index center position from a world tile index

GetTileIndexForMap

```
public IntVector2 GetTileIndexForMap(
     IntVector3 TileWorldIndex,
     MapSet Map)
```

Gets the tile index from a world tile index

GetTileIndexForMap

```
public IntVector2 GetTileIndexForMap(
     Vector3 WorldPosition,
     MapSet Map)
```

Gets the tile index from a position

GetTileTypeForMap

```
public TileTypes GetTileTypeForMap(
     Vector3 WorldPosition,
     MapSet Map)
```

Gets the tile type from a position

GetTileTypeForMap

```
public TileTypes GetTileTypeForMap(
     IntVector2 MapIndex,
     MapSet Map)
```

Gets the tile type from a tile index

GetTileTypeForMap

```
public TileTypes GetTileTypeForMap(
     int x,
     int y,
     MapSet Map)
```

Gets the tile type from a tile index

GetTileBoundsForMap

```
public TileBounds GetTileBoundsForMap(
     int x,
     int y,
     MapSet Map)
```

Get the tile bounds of a map

GetTileBoundsForMap

```
public TileBounds GetTileBoundsForMap(
     IntVector2 TileIndex,
     MapSet Map)
```

Get the tile bounds of a map

GetRandomValidPosition

```
public Vector3 GetRandomValidPosition(
     List<TileTypes> ValidTypes,
     bool PlaceAtCenter = false)
```

Get a random valid position in the scene

GetRandomValidPosition

```
public Vector3 GetRandomValidPosition(
     List<TileTypes> ValidTypes,
     MapSet Map,
     bool PlaceAtCenter = false)
```

Get a random valid position oon this map

GetRandomValidPositionAtPosition

```
public Vector3 GetRandomValidPositionAtPosition(
     List<TileTypes> ValidTypes,
     IntVector2 Position,
     int Range = 30,
     bool PlaceAtCenter = false)
```

Get a random position in range of a position

GetRandomValidPosition

```
public Vector3 GetRandomValidPosition(
     MapSet Map,
     List<TileTypes> ValidTypes,
     bool PlaceAtCenter)
```

get a random valid position for a map

GetRandomValidPositionAtPosition

```
public Vector3 GetRandomValidPositionAtPosition(
     MapSet Map,
     List<TileTypes> ValidTypes,
     IntVector2 Position,
     int Range = 30,
     bool PlaceAtCenter = false)
```

Get a random valid position at a position within a range

GetRandomPosition

```
public Vector3 GetRandomPosition()
```

GetRandomPosition

```
public Vector3 GetRandomPosition(
     MapSet Map)
```

Get a random position on a map

GetCurrentMap

```
public MapSet GetCurrentMap(
     Vector3 WorldPosition)
```

Gets the best map from a position

TileBounds

```
[Serializable]
public struct TileBounds
```

Constructors

TileBounds

```
public  TileBounds(
     Vector3 TilePosition,
     float TileSize)
```

Variables

Center

```
public Vector3 Center
```

BottomLeft

```
public Vector3 BottomLeft
```

TopLeft

```
public Vector3 TopLeft
```

BottomRight

```
public Vector3 BottomRight
```

TopRight

```
public Vector3 TopRight
```

Methods

InBounds

```
public bool InBounds(
     Vector3 Point)
```

IntersectsBounds

```
public bool IntersectsBounds(
     Vector3 PointA,
     Vector3 PointB)
```

LineLineCollision

```
public static bool LineLineCollision(
     Vector3 Position1Start,
     Vector3 Position1End,
     Vector3 Position2Start,
     Vector3 Position2End)
```

HPJ.Presentation.NavigationManagerSettings

```
namespace HPJ.Presentation.NavigationManagerSettings
```

Classes

ManagerSetting

```
[Serializable]
public class ManagerSetting
```

The Manager settings of the Navigation Manager

Variables

MapRedundancy

```
[SerializeField]
[Range(0, 8)]
[Tooltip("The amount of redundant maps generated - More
Maps = Faster Pathfinding at cost of memory")]
public int MapRedundancy
```

The amount of redundant maps generated - More Maps = Faster Pathfinding at cost of memory

PathfindingParrellism

```
[SerializeField]
[Range(2, 16)]
[Tooltip("Setting used for parrellism for each maps
pathfinding")]
public int PathfindingParrellism
```

Setting used for parrellism for each maps pathfinding

MapAutoCreationGridSize

```
[Tooltip("The size of the grid of maps you want to
create")]
public Vector2Int MapAutoCreationGridSize
```

The size of the grid of maps you want to create

MapAutoGridTileWidthAndHeight

```
[Tooltip("The number of tiles in the x and z of the
generated grid maps")]
public Vector2Int MapAutoGridTileWidthAndHeight
```

The number of tiles in the x and z of the generated grid maps

MapAutoGridTileSize

```
[Tooltip("The physical size of the autogenerated tiles on the generated maps")]
public int MapAutoGridTileSize
```

The physical size of the autogenerated tiles on the generated maps

MapAutoGridOffset

```
[Tooltip("The physical offset of the bottom left map")]
public IntVector3 MapAutoGridOffset
```

The physical offset of the bottom left map

MapAutoGridAutoSeam

```
[Tooltip("Whether to create seams for each connected side of the autogrid")]
public bool MapAutoGridAutoSeam
```

Whether to create seams for each connected side of the autogrid

MapSaveDataFolder

```
[Tooltip("The filepath for the save data")]
public string MapSaveDataFolder
```

The filepath for the save data for this scene

SimulationSettings

```
public ExtraSimulationSettings SimulationSettings
```

Extra Simulation Settings

PainterSetting

```
[Serializable]
public class PainterSetting
```

The Painter settings, used to paint tiles on the Maps

Variables

CurrentPainterRadius

```
[Range(0, 500)]
public int CurrentPainterRadius
```

The Paint Brush Size

CurrentPainterType

```
public TileTypes CurrentPainterType
```

The Tile Types we want to paint

PainterType

```
public PainterTypes PainterType
```

The Paint Brush Type

CleanMapIndex

```
[Tooltip("Which Map do you want to clean (Change to
Default Tile)")]
public int CleanMapIndex
```

Which Map do you want to clean (Change to Default Tile)

SquarePaintBrush

```
[NonSerialized]
public MeshRenderer SquarePaintBrush
```

CirclePaintBrush

```
[NonSerialized]
public MeshRenderer CirclePaintBrush
```

SquareFilter

```
[NonSerialized]
public MeshFilter SquareFilter
```

CircleFilter

```
[NonSerialized]
public MeshFilter CircleFilter
```

AutoPainterSetting

```
[Serializable]
public class AutoPainterSetting
```

The Auto-Painter settings, used to paint tiles on the Maps automatically

Variables

AutoPainterType

```
[Tooltip("What Tile Type the Auto painter turns the Tile Into")]
public TileTypes AutoPainterType
```

What Tile Type the Auto painter turns the Tile Into

AutoPainterCheckForLayers

```
[Tooltip("What Layers the Raycast must HIT to turn that tile")]
public LayerMask AutoPainterCheckForLayers
```

What Layers the Raycast must HIT to turn that tile

AutoPainterCheckAgainstLayers

```
[Tooltip("What Layers the Raycast must AVOID HITTING to turn that tile")]
public LayerMask AutoPainterCheckAgainstLayers
```

What Layers the Raycast must AVOID HITTING to turn that tile

AutoPainterElavationIncrease

```
[Tooltip("How much the Raycast will increase its starting position from the height of the Map (Offset Y)")]
public float AutoPainterElavationIncrease
```

"How much the Raycast will increase its starting position from the height of the Map (Offset Y)"

AutoPainterElavationIDecrease

```
[Tooltip("How much the Raycast will go below the height of the Map (Offset Y)")]
public float AutoPainterElavationIDecrease
```

How much the Raycast will go below the height of the Map (Offset Y)

AutoPaintMapIndex

```
[Tooltip("Which Map do you want to Auto Paint")]
public int AutoPaintMapIndex
```

Which Map do you want to Auto Paint

AboveMapTypeChangeHeight

```
[Tooltip("How high above the map, where the tiles change type")]
public float AboveMapTypeChangeHeight
```

How high above the map, where the tiles change type

AutoPainterAboveHeightType

```
[Tooltip("What Tile Type the Auto painter turns the Tile Into on height check above")]
public TileTypes AutoPainterAboveHeightType
```

What Tile Type the Auto painter turns the Tile Into on height check above

BelowMapTypeChangeHeight

```
[Tooltip("How far below the map, where the tiles change type")]
public float BelowMapTypeChangeHeight
```

How far below the map, where the tiles change type

AutoPainterBelowHeightType

```
[Tooltip("What Tile Type the Auto painter turns the Tile Into on height check below")]
public TileTypes AutoPainterBelowHeightType
```

What Tile Type the Auto painter turns the Tile Into on height check below

SeamSettings

```
[Serializable]
public class SeamSettings
```

The all seam settings

Variables

DisplaySeams

```
public bool DisplaySeams
```

Whether to display seams

MapAInsertSettings

```
public MapSeamInsertSettings MapAInsertSettings
```

Map 1 or A Seam Insert Settings

MapBInsertSettings

```
public MapSeamInsertSettings MapBInsertSettings
```

Map 2 or B Seam Insert Settings

MapSeamInsertSettings

```
[Serializable]
public class MapSeamInsertSettings
```

The map seam settings

Variables

MapIndex

```
public int MapIndex
```

Map Index

ConnectionSide

```
public SeamConnectionSide ConnectionSide
```

Map Index

ExtraSimulationSettings

```
[Serializable]
public class ExtraSimulationSettings
```

Extra Simulation Settings

Variables

UseExtraSimulation

```
public bool UseExtraSimulation
```

Whether to use another thread for extra simulation. Used by Simulation Agents and Agent Avoidance

UseAgentAvoidance

```
public bool UseAgentAvoidance
```

Whether to use agent avoidence for the simulation agents

DesiredHelperTickrate

```
public int DesiredHelperTickrate
```

The Desired tickrate of the helper update

GizmosSetting

```
[Serializable]
public class GizmosSetting
```

The Gizmos settings of the Navigation Manager

Variables

DisplayGizmos

```
[Header("GUI")]
public bool DisplayGizmos
```

Whether to show the Gizmos

DisplayBridges

```
public bool DisplayBridges
```

Whether to diplay the bridges

DisplayBridgeDirections

```
public bool DisplayBridgeDirections
```

Shows the direction the bridge allows. Nothing for both ways, otherwise direction is from Red to Blue

InsertBridgePointIndex

```
public int InsertBridgePointIndex
```

The Bridge we want in inset a midpoint on

DisplayTiles

```
public bool DisplayTiles
```

Whether to display tile grid

DisplayMapHandles

```
public bool DisplayMapHandles
```

Whether to display transform handle for the map

DisplayMapLabels

```
public bool DisplayMapLabels
```

Whether to display map nale label for the map

DisplayMapBase

```
public bool DisplayMapBase
```

Whether to display the base of the Maps

DisplayMapEdge

```
public bool DisplayMapEdge
```

Whether to display the Map Edge

DisplayMapAlpha

```
[Range(0, 1)]
public float DisplayMapAlpha
```

The Alpha of the Map Base and Edge

DisplayMapArrows

```
public bool DisplayMapArrows
```

Whether to display the Map arrows on the Maps

DisplayArrowSize

```
public float DisplayArrowSize
```

Adjusts the Map Arrow Size

DisplayPaintBrush

```
public bool DisplayPaintBrush
```

Whether to display the paint brush or not

DebugPaintBrushPosition

```
public bool DebugPaintBrushPosition
```

Whether to debug the paint brush position

DisplayPaintedTiles

```
public bool DisplayPaintedTiles
```

Whether to display the painted tiles. Painter and Auto-Painter

OnlyPaintTheseTypes

```
public List<TileTypes> OnlyPaintTheseTypes
```

The Tiles we want to see

DisplayPaintedTilesMapIndex

```
[Tooltip("Which Map do you want the tiles rendered, (-1 for All)")]
public int DisplayPaintedTilesMapIndex
```

The Map index we want to show the painted tiles on. -1 for all

DisplayMapGizmos_ForMapIndecies

```
public List<int> DisplayMapGizmos_ForMapIndecies
```

limits the amount of gizmos on screen by the desired map indecies. a empty list means every map

DisplayAgentOwnedTiles

```
public bool DisplayAgentOwnedTiles
```

Whetehr to display the tiles owned by agents

2.Agents

AgentGroundCast.cs

Namespaces

HPJ.Presentation.Agents

```
namespace HPJ.Presentation.Agents
```

Classes

AgentGroundCast

```
public class AgentGroundCast
```

A example ground cast that connects a agent to the ground

Variables

_groundLayers

```
[SerializeField]
private LayerMask _groundLayers
```

_raycastSpot

```
[SerializeField]
private Transform _raycastSpot
```

_rayCastDistance

```
[SerializeField]
private float _rayCastDistance
```

_rayCastElevatedDistance

```
[SerializeField]
private float _rayCastElevatedDistance
```

_rayCastEveryXFrames

```
[SerializeField]
[Range(1, 10)]
private int _rayCastEveryXFrames
```

_effectTransform

```
[SerializeField]
private Transform _effectTransform
```

_floatAmount

```
[SerializeField]
private float _floatAmount
```

_raycastDirection

```
[SerializeField]
private Vector3 _raycastDirection
```

AttemptTeleportUpOnFail

```
[SerializeField]
private bool AttemptTeleportUpOnFail
```

_rayHit

```
private RaycastHit _rayHit
```

_physicsCheckCounter

```
private int _physicsCheckCounter
```

Properties

RayCastDistance

```
public float RayCastDistance
     { get;
     set; }
```

RayCastElevatedDistance

```
public float RayCastElevatedDistance
     { get;
     set; }
```

RayCastEveryXFrames

```
public int RayCastEveryXFrames
     { get;
     set; }
```

FloatAmount

```
public float FloatAmount
     { get;
     set; }
```

RaycastDirection

```
public Vector3 RaycastDirection
     { get;
     set; }
```

Methods

PhysicsCheck

```
public void PhysicsCheck()
```

TeleportUp

```
public void TeleportUp()
```

C# AgentLine.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

AgentLine

```
[RequireComponent(typeof(LineRenderer))]
[RequireComponent(typeof(NavigationAgent))]
public class AgentLine
```

Displays a line of the Navigation Agents Remaining Path

Variables

_updateInterval

```
[SerializeField]
private float _updateInterval
```

How Often the line is updated

_updateTime

```
private float _updateTime
```

_line

```
private LineRenderer _line
```

_agent

```
private NavigationAgent _agent
```

_linePoints

```
private List<Vector3> _linePoints
```

Methods

Awake

```
private void Awake()
```

Update

```
void Update()
```

UpdateLine

```
public void UpdateLine()
```

Updates the Line Renderer to the navigation agent remaining path

C# NavigationAgent.cs

Namespaces

HPJ.Presentation.Agents

```
namespace HPJ.Presentation.Agents
```

Classes

NavigationAgent

```
public class NavigationAgent
```

Navigation Agents are used to move and pathfind from its transform.position to a destination

Classes

BreadthFirstData

```
private class BreadthFirstData
```

Constructors

BreadthFirstData

```
public  BreadthFirstData(
     IntVector2 index,
     int score)
```

Variables

Index

```
public IntVector2 Index
```

StepScore

```
public int StepScore
```

Variables

_settings

```
[SerializeField]
protected AgentSettings _settings
```

_state

```
protected AgentStates _state
```

_previousState

```
protected AgentStates _previousState
```

_rotateTransform

```
[SerializeField]
protected Transform _rotateTransform
```

_currentMovementPathIndex

```
protected int _currentMovementPathIndex
```

_currentDirection

```
protected Vector3 _currentDirection
```

OnPathfindingSucceed

```
public UnityEvent<NavigationAgent> OnPathfindingSucceed
```

The Event that is called when the Pathfinding is completed correctly

OnPathfindingFailed

```
public UnityEvent<NavigationAgent> OnPathfindingFailed
```

The Event that is called when the Pathfinding is completed incorrectly

OnMovingComplete

```
public UnityEvent<NavigationAgent> OnMovingComplete
```

The Event that is called when the agent is done moving to it’s destination

GizmosSettings

```
public AgentGizmosSettings GizmosSettings
```

The Gizmos settings for the Agent

Properties

Settings

```
public AgentSettings Settings
     { get;
     }
```

The Agent Settings

State

```
public AgentStates State
     { get;
     internal set; }
```

The Current State of the Agent

PreviousState

```
public AgentStates PreviousState
     { get;
     }
```

The Current State of the Agent

CurrentOrder

```
public NavigationOrder CurrentOrder
     { get;
     protected set; }
```

The Current Navigation Order of the Agent

CurrentMap

```
public MapSet CurrentMap
     { get;
     internal set; }
```

The Current Map the Agent is On. Can be NULL

StartingMap

```
public MapSet StartingMap
     { get;
     internal set; }
```

The Starting Map the Agent is started On. Can be NULL

Path

```
public List<Vector3> Path
     { get;
     internal set; }
```

The Current Path of the Agent

WantedDestination

```
public Vector3 WantedDestination
     { get;
     set; }
```

The Destination this agent wants to move to

GroundCast

```
public AgentGroundCast GroundCast
     { get;
     internal set; }
```

The Ground Cast Component on the gameobject

MovementSpeed

```
public float MovementSpeed
     { get;
     set; }
```

The Movement Speed of the Agent

AutoValidateCurrentPath

```
public bool AutoValidateCurrentPath
     { get;
     set; }
```

If the Agent should validate its current path "Local Avoidance"

AutoValidateTime

```
public float AutoValidateTime
     { get;
     set; }
```

The auto validation interval

LocalAvoidanceRange

```
public int LocalAvoidanceRange
     { get;
     set; }
```

How Far the local avoidance will check for

ValidateCurrentTile

```
public bool ValidateCurrentTile
     { get;
     set; }
```

If the agent should validate the tile the agent is currently on

KeepAttemptingOnFail

```
public bool KeepAttemptingOnFail
     { get;
     set; }
```

If the agent should try and recalculate its path if it fails during the movement of the agent. like during "Local Avoidance"

KeepAttemptingTime

```
public float KeepAttemptingTime
     { get;
     set; }
```

How often the agent should keep trying to recalculate its path

NavigationType

```
public NavigationTypes NavigationType
     { get;
     }
```

The navigation type this agent uses

TraversableTiles

```
public List<TileTypes> TraversableTiles
     { get;
     set; }
```

The Traversable tile types this agent uses

PrefferedTiles

```
public List<TileTypes> PrefferedTiles
     { get;
     set; }
```

The Tiles this agent preffers during pathfinding. Only used for pathfinding types that uses it. Like A*. Not JPS

TraversableTilesKey

```
public string TraversableTilesKey
     { get;
     set; }
```

The Key for the traversable tile types used to get Byte Maps

PrefferedTilesKey

```
public string PrefferedTilesKey
     { get;
     set; }
```

CurrentPathingIndex

```
public int CurrentPathingIndex
     { get;
     internal set; }
```

Current Path Index the agent is on

_lerpPercentage

```
public float _lerpPercentage
     { get;
     internal set; }
```

_lerpTotal

```
public float _lerpTotal
     { get;
     internal set; }
```

_autoValidateTimer

```
public float _autoValidateTimer
     { get;
     internal set; }
```

_keepAttemptingTimer

```
public float _keepAttemptingTimer
     { get;
     internal set; }
```

CurrentTileType

```
public TileTypes CurrentTileType
     { get;
     internal set; }
```

CurrentTileIndex

```
public IntVector2 CurrentTileIndex
     { get;
     internal set; }
```

CurrentDirection

```
public Vector3 CurrentDirection
     { get;
     internal set; }
```

The Current Direction the agent is going on

FailedLastPath

```
public bool FailedLastPath
     { get;
     protected set; }
```

If the agent failed its last pathfinding

Methods

Awake

```
protected virtual void Awake()
```

AddToManager

```
protected virtual void AddToManager()
```

Adds the Agent to the Navigation Manager

SetCurrentMap

```
public void SetCurrentMap(
     MapSet map)
```

Sets the current map of the agent, if its not on the map, it will move to the nearest point on this map

GetRandomDestination

```
internal virtual IntVector3 GetRandomDestination(
     int randomRange)
```

Finds a random valid point within a range of the agents position

GetRandomDestinationAwayFromPoint

```
internal virtual IntVector3
GetRandomDestinationAwayFromPoint(
     Vector3 hitPoint,
     int randomTileDistanceVariance)
```

Finds a random valid point away the wanted location

GetRandomDestinationNearPoint

```
public virtual IntVector3 GetRandomDestinationNearPoint(
     Vector3 hitPoint,
     int randomTileDistanceVariance)
```

Finds a random valid point near the wanted location

OnDestroy

```
protected virtual void OnDestroy()
```

OnDrawGizmosSelected

```
protected virtual void OnDrawGizmosSelected()
```

AutoUpdateAgent

```
public virtual void AutoUpdateAgent()
```

Update is called once per frame

UpdateAgentState

```
public virtual void UpdateAgentState()
```

Updates the Agent

Stop

```
public virtual void Stop()
```

Stops the agent, continue makes this go into the idle state

Pause

```
public virtual void Pause()
```

Stops the Agent, Can be continued

Continue

```
public virtual void Continue()
```

Makes the agent continue its path or stop being stopped (Paused to Moving or Stopped to Idle)

Idle

```
public virtual void Idle()
```

Idles the Agent

ValidateStandingTile

```
public virtual bool ValidateStandingTile()
```

Tells you if the current tile the unit is on is valid

CurrentPathIsValid

```
public virtual bool CurrentPathIsValid(
     int Range = -1)
```

Checks the tiles in the path to see if they are still valid

SetDestination

```
public virtual void SetDestination(
     Vector3 Destination,
     bool RecenterOnTile = false,
     bool SetWantedDestination = true)
```

Used to have the Agent calculate its path and move towards this desintation

SetDestination

```
internal virtual void SetDestination(
     IntVector3 WorldTileIndexDestination,
     bool RecenterOnTile = false,
     bool SetWantedDestination = false)
```

Used to have the Agent calculate its path and move towards this desintation

ReadjustToNearestPoint

```
public virtual void ReadjustToNearestPoint()
```

Readjusts the adjust so it sets it destination to the nearest tile to it’s destination that is valid

SetDestinationToNearestPoint

```
public virtual void SetDestinationToNearestPoint(
     Vector3 Destination,
     bool RecenterOnTile = false)
```

Sets the destination to the nearest valid point near the desired destination

SetDestinationToNearestPoint

```
protected virtual void SetDestinationToNearestPoint(
     IntVector3 Destination,
     bool RecenterOnTile = false)
```

Sets the destination to the nearest valid point near the desired destination

RecievePathingInfoCallback

```
protected virtual void RecievePathingInfoCallback()
```

The Callback for when the agent recieves its path

SetInitialPositionToPath

```
protected virtual void SetInitialPositionToPath()
```

Adds the initial position to the list. Replaces first index if starting position in closer to end

GetCurrentPosition

```
internal virtual Vector3 GetCurrentPosition()
```

Helps get the current position for this agent. Sim agents override this to use the simagent position

ValidateTileIndex

```
public virtual bool ValidateTileIndex(
     IntVector2 TileIndex,
     MapSet Map)
```

This tells you if the current tile index is valid for this unit

GetTraversableTilesInRange

```
public List<IntVector2> GetTraversableTilesInRange(
     int Range,
     DistanceCheck checkType = DistanceCheck.Distance)
```

Gives a list of tile indicies that are traverable in range

GetTraversableTilesInRangeBySteps

```
public List<IntVector2>
GetTraversableTilesInRangeBySteps(
     int Steps,
     MovementType MovementType =
     MovementType.EightDirection)
```

Gives a list of tile indicies that are traverable in the number of steps required to get there. Suggest not running this every frame

NavigationOrder

```
[Serializable]
public class NavigationOrder
```

The Navigation Order for the agent to bridge itself to the simulation

Constructors

NavigationOrder

```
public  NavigationOrder(
     NavigationAgent agent)
```

Variables

Agent

```
public NavigationAgent Agent
```

Agent for this Order

Start

```
public IntVector3 Start
```

Navigation Start

Destination

```
public IntVector3 Destination
```

Navigation Current End

FinalDestination

```
public IntVector3 FinalDestination
```

Navigation Desired End

PreviousStart

```
public IntVector3 PreviousStart
```

Navigation Previous Movement Start

PreviousDestination

```
public IntVector3 PreviousDestination
```

Navigation Movement End

CheckPointDestination

```
public IntVector3 CheckPointDestination
```

Navigation Current Destination

DestinationState

```
public DestinationStates DestinationState
```

The Current State the Navigation is at

NavJob

```
public NavigationJob NavJob
```

The Navigation Job request used to pathfind for the agent

PathfindingStep

```
public PathfindingCalculationStep PathfindingStep
```

Current pathfinding step of the agent

StartingMap

```
public MapSet StartingMap
```

Starting map of the order

EndingMap

```
public MapSet EndingMap
```

Ending map of the order

FoundBridge

```
public MapBridge FoundBridge
```

The bridge that was found and needed to be move onto to get to the destination

FoundSeam

```
public MapSeam FoundSeam
```

The seam that was found and needed to be move onto to get to the destination

Recenter

```
public bool Recenter
```

Whether to move on the center of the tile or not

Methods

Clear

```
public void Clear()
```

Clears the order to default

SetDestination

```
public virtual void SetDestination(
     Vector3 Position,
     bool RecenterOnTile = false)
```

Sets the destination for the agent

SetDestination

```
internal virtual void SetDestination(
     IntVector3 Position,
     bool RecenterOnTile = false)
```

Sets the destination for the agent

NavJobCallback

```
private void NavJobCallback()
```

The callback when the pathfinding is completed

SetPath

```
public void SetPath(
     List<Vector3> path)
```

Sorts the path for the agent

SetPrevious

```
internal void SetPrevious()
```

Sets the previous pathing info

AgentGizmosSettings

```
[Serializable]
public class AgentGizmosSettings
```

The gizmos settings for the agent

Variables

ShowGizmos

```
public bool ShowGizmos
```

Whether to show the Gizmos

ShowPath

```
public bool ShowPath
```

Whether to show the path

ShowStartAndEnd

```
public bool ShowStartAndEnd
```

Whether to show the start and end positions

ShowTraverableRange

```
public bool ShowTraverableRange
```

Whether to show a traversable range around the agent

TraversableRange

```
public int TraversableRange
```

The traversable range we want to show

ShowValidTilesInRange

```
public bool ShowValidTilesInRange
```

whether to show all the surrounding tiles within range of the aganet

ValidTileRange

```
public int ValidTileRange
```

The Valid tile range we want to show

ShowClosestValidTileToWantedPosition

```
public bool ShowClosestValidTileToWantedPosition
```

Whether to show the closest tile to the wanted position

AgentSettings

```
[Serializable]
public class AgentSettings
```

The Agent settings for the Agent

Variables

AutomaticUpdate

```
public bool AutomaticUpdate
```

Whether to automatically update the agent with the given systems

GroundCheck

```
public bool GroundCheck
```

whether to check the ground beneath the agent

RotateTowardsMovingDirection

```
public bool RotateTowardsMovingDirection
```

whether the agent rotates towards pathing direction

RotateSpeed

```
public float RotateSpeed
```

Rotation speed

_movementSpeed

```
[SerializeField]
private float _movementSpeed
```

Movement speed of the agent

_autoValidateCurrentPath

```
[SerializeField]
private bool _autoValidateCurrentPath
```

whether to auto validate the current path

_autoValidateTime

```
[SerializeField]
private float _autoValidateTime
```

Auto validate interval

_localAvoidanceRange

```
[SerializeField]
private int _localAvoidanceRange
```

Auto validate range

_validateCurrentTile

```
[SerializeField]
private bool _validateCurrentTile
```

Whether to validate the current tile the agent is on

_keepAttemptingOnFail

```
[SerializeField]
private bool _keepAttemptingOnFail
```

Whether to keep attempting to find a path when the current path is invalidated

_onFailAttemptToNearestPoint

```
[SerializeField]
private bool _onFailAttemptToNearestPoint
```

Whether to try to move to the nearest point of your destination

_keepTryingToReadjustToWantedPosition

```
[SerializeField]
private bool _keepTryingToReadjustToWantedPosition
```

Whether to try and keep to readjust to wanted position

_keepAttemptingTime

```
[SerializeField]
private float _keepAttemptingTime
```

Keep checking on fail interval

_navigationType

```
[SerializeField]
private NavigationTypes _navigationType
```

The Agent navigation type

_traversableTiles

```
[SerializeField]
private List<TileTypes> _traversableTiles
```

The Traversable tiles for the agent

_prefferedTiles

```
[SerializeField]
private List<TileTypes> _prefferedTiles
```

the preffered pathing tiles for the agent

Properties

MovementSpeed

```
public float MovementSpeed
     { get;
     set; }
```

AutoValidateCurrentPath

```
public bool AutoValidateCurrentPath
     { get;
     set; }
```

AutoValidateTime

```
public float AutoValidateTime
     { get;
     set; }
```

LocalAvoidanceRange

```
public int LocalAvoidanceRange
     { get;
     set; }
```

ValidateCurrentTile

```
public bool ValidateCurrentTile
     { get;
     set; }
```

KeepAttemptingOnFail

```
public bool KeepAttemptingOnFail
     { get;
     set; }
```

OnFailAttemptToNearestPoint

```
public bool OnFailAttemptToNearestPoint
     { get;
     set; }
```

KeepTryingToReadjustToWantedPosition

```
public bool KeepTryingToReadjustToWantedPosition
     { get;
     set; }
```

KeepAttemptingTime

```
public float KeepAttemptingTime
     { get;
     set; }
```

NavigationType

```
public NavigationTypes NavigationType
     { get;
     }
```

TraversableTiles

```
public List<TileTypes> TraversableTiles
     { get;
     set; }
```

PrefferedTiles

```
public List<TileTypes> PrefferedTiles
     { get;
     set; }
```

TraversableTilesKey

```
public string TraversableTilesKey
     { get;
     set; }
```

PrefferedTilesKey

```
public string PrefferedTilesKey
     { get;
     set; }
```

More Agents

SimAgent.cs

Namespaces

HPJ.Presentation.Agents

```
namespace HPJ.Presentation.Agents
```

Classes

SimAgent

```
[System.Serializable]
public class SimAgent
```

Constructors

SimAgent

```
public  SimAgent(
     SimNavigationAgent Agent,
     bool UseAgentAvoidence)
```

Creates and Initializes the Agent

Variables

SimIDCounter

```
public static ushort SimIDCounter
```

The ID for each simulation Agent

NullTile

```
public static IntVector2 NullTile
```

Properties

SimID

```
public ushort SimID
     { get;
     protected set; }
```

NavAgent

```
public SimNavigationAgent NavAgent
     { get;
     protected set; }
```

The corresponding Navigation Agent for this Simulation Agent

Position

```
public Vector3 Position
     { get;
     protected set; }
```

Position of the simulation agent

AgentAvoidenceActive

```
public bool AgentAvoidenceActive
     { get;
     internal set; }
```

Whether agent avoidance is active

CallOnMovingComplete

```
public bool CallOnMovingComplete
     { get;
     internal set; }
```

Whether the base agent should call the on moving complete call

CallOnRecievePath

```
public bool CallOnRecievePath
     { get;
     internal set; }
```

Whether the base agent should call the on recieved path call

UpdateToUnity

```
public bool UpdateToUnity
     { get;
     internal set; }
```

Updates to the Core Agents position

OwnedTile

```
public IntVector2 OwnedTile
     { get;
     internal set; }
```

Tile owned by the Agent

Methods

Update

```
public void Update()
```

Updates the agent simuliarily the unity version of Navigaiton Agent gets updated

UpdatePositionFromAgent

```
internal void UpdatePositionFromAgent()
```

Updates the position to the position of the Agent

UpdatePosition

```
protected void UpdatePosition(
     Vector3 NewPosition)
```

Internal way to update position and tile ownership for the agent

RecheckOwnership

```
public void RecheckOwnership()
```

Rechecks the ownership of the tile

OnDestroy

```
internal void OnDestroy()
```

Make sure to remove this agent when it’s destroyed

C# SimNavigationAgent.cs

Namespaces

HPJ.Presentation.Agents

```
namespace HPJ.Presentation.Agents
```

Classes

SimNavigationAgent

```
public class SimNavigationAgent
```

Properties

SimulationAgent

```
public SimAgent SimulationAgent
     { get;
     protected set; }
```

Simulation Counterpart of this agent

UsingAgentAvoidence

```
public bool UsingAgentAvoidence
     { get;
     protected set; }
```

Whether sim agents are using Agent Avoidence

DesiredPosition

```
public Vector3 DesiredPosition
     { get;
     protected set; }
```

The Position you want to update the agent to

Methods

AddToManager

```
protected override void AddToManager()
```

Adds the Agent to the Navigation Manager as a simulation agent

UpdateAgentState

```
public override void UpdateAgentState()
```

OnDrawGizmosSelected

```
protected override void OnDrawGizmosSelected()
```

RecievePathingInfoCallback

```
protected override void RecievePathingInfoCallback()
```

The Callback for when the agent recieves its path

UpdateAgentPosition

```
public void UpdateAgentPosition(
     Vector3 NewPosition,
     bool AutocancelPathing = false)
```

Used to adjust the agent position in the simulation. Note make sure that the state is not moving or the agent will teleport back to it's simulated position

ValidateStandingTile

```
public override bool ValidateStandingTile()
```

Tells you if the current tile the unit is on is valid

CurrentPathIsValid

```
public override bool CurrentPathIsValid(
     int Range = -1)
```

Checks the tiles in the path to see if they are still valid

ValidateTileIndex

```
public override bool ValidateTileIndex(
     IntVector2 TileIndex,
     MapSet Map)
```

This tells you if the current tile index is valid for this unit

SetDestinationToNearestPoint

```
public override void SetDestinationToNearestPoint(
     Vector3 Destination,
     bool RecenterOnTile = false)
```

Sets the destination to the nearest valid point near the desired destination

SetDestinationToNearestPoint

```
protected override void SetDestinationToNearestPoint(
     IntVector3 Destination,
     bool RecenterOnTile = false)
```

Sets the destination to the nearest valid point near the desired destination

GetCurrentPosition

```
internal override Vector3 GetCurrentPosition()
```

Helps get the current position for this agent. Sim agents override this to use the simagent position

Awake

```
protected override void Awake()
```

OnDestroy

```
protected override void OnDestroy()
```

3.Obstacles

C# NavigationObstacle.cs

Namespaces

HPJ.Presentation.Obstacle

```
namespace HPJ.Presentation.Obstacle
```

Classes

NavigationObstacle

```
public abstract class NavigationObstacle
```

The Base class for all navigation obstacles

Variables

_automaticUpdate

```
[SerializeField]
protected bool _automaticUpdate
```

Does the obstacle update automatically through the built in systems

_checkMapHeight

```
[SerializeField,Tooltip("The Height from the transform y
position to the ground that this counts as an obstacle")]
protected float _checkMapHeight
```

The Height difference from the map to count as being on top of the map

_obstacleTileType

```
[SerializeField]
protected TileTypes _obstacleTileType
```

The tile type the obstacle changes the tiles beneath it too

Properties

AutomaticUpdate

```
public bool AutomaticUpdate
     { get;
     set; }
```

CheckMapHeight

```
public float CheckMapHeight
     { get;
     set; }
```

ObstacleTileType

```
public TileTypes ObstacleTileType
     { get;
     }
```

MapTileRelationships

```
public Dictionary<MapSet,MapObstacleRelationshipData>
MapTileRelationships
     { get;
     protected set; }
```

The tiles and map relationship data with this obstacle

Methods

Start

```
public virtual void Start()
```

OnDestroy

```
public virtual void OnDestroy()
```

UpdateObstacle

```
public virtual void UpdateObstacle()
```

The Update cycle of the obstacle

UpdatePosition

```
public abstract void UpdatePosition(
     Vector3 NewPosition,
     bool ForceCheck = false)
```

Checks to see if the new position changes and if so it changes the mesh to block out the new corresponding tiles

PositionChanged

```
protected abstract bool PositionChanged(
     Vector3 NewPosition)
```

What happens the the obstacle mesh position changes

Rotate

```
public abstract void Rotate(
     Quaternion Rotation)
```

Rotates the obstacle based on the desired Rotation. Basic NavigationObstacle will only swap the object size x and y values. Inherit to ovverride this behaviour

RevalidateTiles

```
public abstract void RevalidateTiles()
```

Revalidates the tiles of the obstacle in case another obstacle travels over top of this one

PositionSquareRect

```
[Serializable]
public struct PositionSquareRect
```

Constructors

PositionSquareRect

```
public  PositionSquareRect(
     Vector3 position,
     Vector2 size)
```

PositionSquareRect

```
public  PositionSquareRect(
     Vector3 position,
     Vector3 LowerBounds,
     Vector3 HigherBounds)
```

Variables

Position

```
public Vector3 Position
```

Size

```
public Vector2 Size
```

Corners

```
public Vector3[] Corners
```

Methods

UpdatePosition

```
public void UpdatePosition(
     Vector3 position)
```

UpdateSize

```
public void UpdateSize(
     Vector2 newSize)
```

InBounds

```
public bool InBounds(
     Vector3 Point)
```

MapObstacleRelationshipData

```
[Serializable]
public class MapObstacleRelationshipData
```

Constructors

MapObstacleRelationshipData

```
public  MapObstacleRelationshipData(
     MapSet map,
     NavigationObstacle obstacle,
     TileTypes ObstacleType)
```

Variables

Map

```
public MapSet Map
```

Obstacle

```
public NavigationObstacle Obstacle
```

RelatedTilesData

```
public Dictionary<IntVector3,ObstacleData>
RelatedTilesData
```

ValidatedTiles

```
public HashSet<IntVector3> ValidatedTiles
```

InvalidatedTiles

```
public HashSet<IntVector3> InvalidatedTiles
```

AddedTiles

```
public HashSet<IntVector3> AddedTiles
```

ObstacleTileType

```
public TileTypes ObstacleTileType
```

Methods

UnvalidateTiles

```
public void UnvalidateTiles()
```

AddOrValidateTile

```
public void AddOrValidateTile(
     IntVector3 tileIndex)
```

SortRelatedTileData

```
public void SortRelatedTileData()
```

GenerateMapChangeJob

```
public ChangeMapJob GenerateMapChangeJob()
```

Revalidate

```
internal void Revalidate()
```

C# NavigationObstacleMesh.cs

Namespaces

HPJ.Presentation.Obstacle

```
namespace HPJ.Presentation.Obstacle
```

Classes

NavigationObstacleMesh

```
public class NavigationObstacleMesh
```

A Navigation Obstacle that adjusts the Map based on the given mesh, position and rotation

Enumerations

ObstacleCollisionCheckType

```
public enum ObstacleCollisionCheckType{
     None,
     LineCollisions,
     AreaCollisions,
     Both,}
```

The Type of Collision the Obstacle Mesh attempts

Classes

VertexPoints

```
[Serializable]
public class VertexPoints
```

A Vertex Point based on the obstacle mesh

Constructors

VertexPoints

```
public  VertexPoints(
     Vector3 Position,
     int vertexIndex)
```

Variables

VertexIndex

```
public int VertexIndex
```

VertexPosition

```
public Vector3 VertexPosition
```

SharedPoints

```
public List<int> SharedPoints
```

Triangles

```
public List<Vector3Int> Triangles
```

ConnectedVertex

```
[NonSerialized]
public List<VertexPoints> ConnectedVertex
```

Methods

ContainsTriangeIndex

```
public bool ContainsTriangeIndex(
     Vector3Int currentTriangle)
```

Checks to see if this triange shares a mesh index with this vertex point

ContainsTriange

```
public bool ContainsTriange(
     Vector3Int currentTriangle)
```

Checks to see if this triange shares has this specific triange

VertexConnection

```
[Serializable]
public class VertexConnection
```

This is a connection between two vertex points

Constructors

VertexConnection

```
public  VertexConnection(
     VertexPoints A,
     VertexPoints B)
```

Variables

PointA

```
public VertexPoints PointA
```

PointB

```
public VertexPoints PointB
```

Variables

ObstacleRenderer

```
[SerializeField]
protected MeshRenderer ObstacleRenderer
```

Mesh Renderer we want to use for the obstacle

ObstacleFilter

```
[SerializeField]
protected MeshFilter ObstacleFilter
```

Mesh we want to use for the obstacle

CollisionCheckMode

```
public ObstacleCollisionCheckType CollisionCheckMode
```

Type of Collision we want to use for this obstacle

Points

```
public Dictionary<Vector3,VertexPoints> Points
```

A Dictionary of the Vertex points based on their physical position relative to the default mesh

Vertices

```
public List<VertexPoints> Vertices
```

A List of the Vertex points for this mesh

Connections

```
public Dictionary<Vector2Int,VertexConnection>
Connections
```

A Dictionary of the connection based on index of the smallest vertex point index as Vector2Int.x and the largest vertex point index as Vector2Int.y

VertexConnections

```
public List<VertexConnection> VertexConnections
```

A List of the Vertex connections for this mesh

Triangles

```
public List<Vector3Int> Triangles
```

A List of the traingles for the mesh

_previousPosition

```
protected Vector3 _previousPosition
```

_previousRotation

```
protected Quaternion _previousRotation
```

OrientationRect

```
protected PositionSquareRect OrientationRect
```

Properties

VertexCount

```
public int VertexCount
     { get;
     protected set; }
```

The number of Vertex on this Mesh

Methods

Awake

```
private void Awake()
```

Start

```
public override void Start()
```

OnDrawGizmosSelected

```
public virtual void OnDrawGizmosSelected()
```

UpdatePosition

```
public override void UpdatePosition(
     Vector3 NewPosition,
     bool ForceCheck = false)
```

Checks to see if the new position changes and if so it changes the mesh to block out the new corresponding tiles

RevalidateTiles

```
public override void RevalidateTiles()
```

Revalidate tiles

PositionChanged

```
protected override bool PositionChanged(
     Vector3 NewPosition)
```

What happens the the obstacle mesh position changes

Rotate

```
public override void Rotate(
     Quaternion Rotation)
```

Rotates the obstacle based on the desired Rotation. Basic NavigationObstacle will only swap the object size x and y values. Inherit to ovverride this behaviour

OrientationChanged

```
protected virtual void OrientationChanged(
     Quaternion Rotation,
     Vector3 NewPosition)
```

Changes the mesh to block out the new corresponding tiles

ObstacleCollidesWithBounds

```
public bool ObstacleCollidesWithBounds(
     TileBounds Bounds,
     ObstacleCollisionCheckType CheckMode =
     ObstacleCollisionCheckType.Both)
```

Collision Check for the obstacle and the maps Tiles

ObstacleLinesCollide

```
protected bool ObstacleLinesCollide(
     TileBounds Bounds)
```

Line Line Collision between all the vertex points and the map tiles

ObstacleAreaCollide

```
protected bool ObstacleAreaCollide(
     TileBounds Bounds)
```

Area Collision between all the vertex triangles and the map tiles

TriangeArea

```
public float TriangeArea(
     Vector3 point1,
     Vector3 point2,
     Vector3 point3)
```

Calculates the Area of the triange with the given points

IsInsideTriange

```
public bool IsInsideTriange(
     Vector3 TriangePoint1,
     Vector3 TriangePoint2,
     Vector3 TriangePoint3,
     Vector3 CheckingPoint)
```

Checks to see if a given point is withen a triangle

IsInsideTriange

```
public bool IsInsideTriange(
     Vector3 TriangePoint1,
     Vector3 TriangePoint2,
     Vector3 TriangePoint3,
     TileBounds TileBounds)
```

Checks to see if a given tile is withen a triangle

GetVertexPosition

```
public Vector3 GetVertexPosition(
     Vector3 Position)
```

Converts a position relative to the local transforms roation and scale

ResizeBoundsEstimate

```
public void ResizeBoundsEstimate()
```

Calculates the bounds of the mesh

C# NavigationObstacleSimpleSquare.cs

Namespaces

HPJ.Presentation.Obstacle

```
namespace HPJ.Presentation.Obstacle
```

Classes

NavigationObstacleSimpleSquare

```
public class NavigationObstacleSimpleSquare
```

Variables

_obstacleSize

```
[SerializeField]
private Vector2 _obstacleSize
```

The size of the obstacle in the x and z world directions

_previousPositionRect

```
private PositionSquareRect _previousPositionRect
```

_newPositionRect

```
private PositionSquareRect _newPositionRect
```

_previousRotation

```
protected Quaternion _previousRotation
```

Properties

ObstacleSize

```
public Vector2 ObstacleSize
     { get;
     set; }
```

Methods

Start

```
public override void Start()
```

OnDrawGizmosSelected

```
public virtual void OnDrawGizmosSelected()
```

RevalidateTiles

```
public override void RevalidateTiles()
```

Revalidate Tiles

UpdatePosition

```
public override void UpdatePosition(
     Vector3 NewPosition,
     bool ForceCheck = false)
```

Checks to see if the new position changes and if so it changes the mesh to block out the new corresponding tiles

PositionChanged

```
protected override bool PositionChanged(
     Vector3 NewPosition)
```

What happens the the obstacle mesh position changes

Rotate

```
public override void Rotate(
     Quaternion Rotation)
```

Rotates the obstacle based on the desired Rotation. Basic NavigationObstacle will only swap the object size x and y values. Inherit to ovverride this behaviour

C# NavigationObstacleSphere.cs

Namespaces

HPJ.Presentation.Obstacle

```
namespace HPJ.Presentation.Obstacle
```

Classes

NavigationObstacleSphere

```
public class NavigationObstacleSphere
```

Variables

_radius

```
[SerializeField]
protected float _radius
```

_previousPosition

```
private Vector3 _previousPosition
```

Properties

Radius

```
public float Radius
     { get;
     set; }
```

Methods

Start

```
public override void Start()
```

OnDrawGizmosSelected

```
public virtual void OnDrawGizmosSelected()
```

UpdatePosition

```
public override void UpdatePosition(
     Vector3 NewPosition,
     bool ForceCheck = false)
```

RevalidateTiles

```
public override void RevalidateTiles()
```

PositionChanged

```
protected override bool PositionChanged(
     Vector3 NewPosition)
```

Rotate

```
public override void Rotate(
     Quaternion Rotation)
```

Rotates the obstacle based on the desired Rotation. Basic NavigationObstacle will only swap the object size x and y values. Inherit to ovverride this behaviour

4.Editor

NavigationManagerEditor.cs

Namespaces

HPJ.Presentation.UnityEditor

```
namespace HPJ.Presentation.UnityEditor
```

Classes

NavigationManagerEditor

```
[CustomEditor(typeof(NavigationManager))]
public class NavigationManagerEditor
```

Variables

NearestHandle

```
private int NearestHandle
```

_previousMousePosition

```
private Vector3 _previousMousePosition
```

_previousTileAmount

```
private int _previousTileAmount
```

_previousOffset

```
private IntVector3 _previousOffset
```

_shifted

```
private bool _shifted
```

_controlDown

```
private bool _controlDown
```

_paintButtonDown

```
private bool _paintButtonDown
```

_removePaintButtonDown

```
private bool _removePaintButtonDown
```

IsMapFoldout

```
bool IsMapFoldout
```

IsPainterFoldout

```
bool IsPainterFoldout
```

IsBridgeFoldout

```
bool IsBridgeFoldout
```

MiscButtonsFoldout

```
bool MiscButtonsFoldout
```

ControlsFildOut

```
bool ControlsFildOut
```

Methods

OnEnable

```
private void OnEnable()
```

OnSceneGUI

```
protected virtual void OnSceneGUI()
```

OnInspectorGUI

```
public override void OnInspectorGUI()
```

5.Sample Scenes

AgentAreaDisplay.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

AgentAreaDisplay

```
[RequireComponent(typeof(NavigationAgent))]
public class AgentAreaDisplay
```

Variables

_tilePrefab

```
[SerializeField]
private AreaDisplayTile _tilePrefab
```

TileSpawn

```
private Transform TileSpawn
```

_showTilesOutOfReach

```
[SerializeField]
private bool _showTilesOutOfReach
```

_range

```
[SerializeField]
private int _range
```

_updatePeriod

```
[SerializeField]
private float _updatePeriod
```

_updateTime

```
private float _updateTime
```

_activeTiles

```
private List<AreaDisplayTile> _activeTiles
```

_pool

```
private List<AreaDisplayTile> _pool
```

_agent

```
private NavigationAgent _agent
```

Methods

Awake

```
private void Awake()
```

Update

```
void Update()
```

UpdateTiles

```
public void UpdateTiles()
```

GetTile

```
protected AreaDisplayTile GetTile()
```

ReturnTile

```
protected void ReturnTile(
     AreaDisplayTile Tile)
```

ReturnAllTiles

```
protected void ReturnAllTiles()
```

C# AreaDisplayTile.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

AreaDisplayTile

```
public class AreaDisplayTile
```

Variables

_canReachMaterial

```
[SerializeField]
private Material _canReachMaterial
```

_cannotReachMaterial

```
[SerializeField]
private Material _cannotReachMaterial
```

_meshRenderer

```
[SerializeField]
private MeshRenderer _meshRenderer
```

Properties

CurrentTileIndex

```
public IntVector2 CurrentTileIndex
     { get;
     protected set; }
```

Methods

Initialize

```
public void Initialize(
     int TileSize)
```

Set

```
public void Set(
     MapSet Map,
     IntVector2 TileIndex,
     bool CanReach)
```

C# LocalAvoidanceSceneScript.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

LocalAvoidanceSceneScript

```
public class LocalAvoidanceSceneScript
```

Am Example script to display the local avoidance feature

Variables

StartTransform

```
public Transform StartTransform
```

EndTransform

```
public Transform EndTransform
```

AgentPrefab

```
public NavigationAgent AgentPrefab
```

_firstAgent

```
private NavigationAgent _firstAgent
```

_direction

```
private bool _direction
```

Methods

Start

```
private void Start()
```

RepeatDestination

```
private void RepeatDestination(
     NavigationAgent Agent)
```

C# MapSpriteRenderer.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

MapSpriteRenderer

```
[RequireComponent(typeof(SpriteRenderer))]
public class MapSpriteRenderer
```

The Map Sprite Renderer is a set of sprite renderers that render the tile state of a desired map.

Variables

UpdatePeriod

```
[SerializeField]
protected float UpdatePeriod
```

UpdateTimer

```
protected float UpdateTimer
```

DesiredMapIndex

```
[SerializeField]
protected int DesiredMapIndex
```

ColorData

```
[SerializeField]
protected List<TileColorData> ColorData
```

SortedData

```
protected Dictionary<TileTypes,TileColorData> SortedData
```

SpriteRen

```
protected SpriteRenderer SpriteRen
```

GridSpriteRen

```
[SerializeField]
protected SpriteRenderer GridSpriteRen
```

FirstUpdate

```
protected bool FirstUpdate
```

tex

```
protected Texture2D tex
```

colors

```
protected Color[] colors
```

Methods

Awake

```
private void Awake()
```

Update

```
private void Update()
```

UpdateSprite

```
public void UpdateSprite()
```

Updates the sprite to match the current state of the desired map

TileColorData

```
[System.Serializable]
public struct TileColorData
```

The Color data for a given tile type

Constructors

TileColorData

```
public  TileColorData(
     Color color,
     TileTypes Type)
```

Variables

TileColor

```
public Color TileColor
```

TileType

```
public TileTypes TileType
```

C# MoveTowardsPoint.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

MoveTowardsPoint

```
public class MoveTowardsPoint
```

Variables

target

```
public Transform target
```

speed

```
public float speed
```

acceleration

```
public float acceleration
```

Methods

Start

```
private void Start()
```

Update

```
private void Update()
```

C# RandomPathsExampleScene.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

RandomPathsExampleScene

```
public class RandomPathsExampleScene
```

The example scene to display random pathing

Variables

_getRandomPointEvery

```
[SerializeField]
private float _getRandomPointEvery
```

_timer

```
private float _timer
```

_randomTileDistanceVariance

```
[SerializeField]
private int _randomTileDistanceVariance
```

_randomRange

```
[SerializeField]
private int _randomRange
```

_testAgent

```
[SerializeField]
private NavigationAgent _testAgent
```

_testLine

```
[SerializeField]
private LineRenderer _testLine
```

_testJob

```
private NavigationJob _testJob
```

_linePoints

```
private List<Vector3> _linePoints
```

RandomMode

```
public int RandomMode
```

Properties

PaththingCompleted

```
public bool PaththingCompleted
     { get;
     set; }
```

PreviousRandomDestination

```
public IntVector3 PreviousRandomDestination
     { get;
     protected set; }
```

Methods

Awake

```
private void Awake()
```

OnCompleteCallback

```
public void OnCompleteCallback()
```

Update

```
void Update()
```

UpdateLine

```
private void UpdateLine()
```

SampleSceneScript.cs

Namespaces

HPJ.Presentation.Sample

```
namespace HPJ.Presentation.Sample
```

Classes

SampleSceneScript

```
public class SampleSceneScript
```

The all round example scene script

Variables

_agentSpawnCount

```
[SerializeField]
private int _agentSpawnCount
```

TestAgents

```
[SerializeField]
private NavigationAgent TestAgents
```

_rollOutSlow

```
[SerializeField]
private bool _rollOutSlow
```

_closeRangeRandomPosition

```
[SerializeField]
private bool _closeRangeRandomPosition
```

_closeRangeRadius

```
[SerializeField]
private int _closeRangeRadius
```

_keepAgentsOnSameMap

```
[SerializeField]
private bool _keepAgentsOnSameMap
```

Methods

Start

```
private void Start()
```

RollOutSlow

```
private IEnumerator RollOutSlow()
```

RollOut

```
private IEnumerator RollOut()
```

SetRandomDestination

```
private void SetRandomDestination(
     NavigationAgent Agent)
```

SwarmSample.cs

Namespaces

HPJ.Presentation

```
namespace HPJ.Presentation
```

Classes

SwarmSample

```
public class SwarmSample
```

This sample script is to spawn and control multiple units to see how swarming runs

Variables

_testSwarmAmount

```
[SerializeField]
private int _testSwarmAmount
```

_testAgentPrefab

```
[SerializeField]
private SimNavigationAgent _testAgentPrefab
```

_targetLayer

```
[SerializeField]
private LayerMask _targetLayer
```

Properties

Agents

```
public List<SimNavigationAgent> Agents
     { get;
     protected set; }
```

Methods

Awake

```
private void Awake()
```

Start

```
private void Start()
```

RollOut

```
private IEnumerator RollOut()
```

Update

```
void Update()
```

C# TargetAgentMoveScript.cs

Namespaces

HPJ.Presentation.Sample

```
namespace HPJ.Presentation.Sample
```

Classes

TargetAgentMoveScript

```
public class TargetAgentMoveScript
```

Variables

_testAgent

```
[SerializeField]
private NavigationAgent _testAgent
```

_targetLayer

```
[SerializeField]
private LayerMask _targetLayer
```

_testJob

```
private NavigationJob _testJob
```

_linePoints

```
private List<Vector3> _linePoints
```

Properties

PaththingCompleted

```
public bool PaththingCompleted
     { get;
     set; }
```

PreviousDestination

```
public IntVector3 PreviousDestination
     { get;
     protected set; }
```

Methods

Awake

```
private void Awake()
```

Start

```
private void Start()
```

OnCompleteCallback

```
public void OnCompleteCallback()
```

Update

```
void Update()
```

2.Simulation

1.Map

AgentMap.cs

Namespaces

HPJ.Simulation.Map

```
namespace HPJ.Simulation.Map
```

Classes

AgentMap

```
public class AgentMap
```

Constructors

AgentMap

```
public  AgentMap(
     Map baseMap)
```

Initializes the agent map based on the base map

Properties

Base

```
public Map Base
     { get;
     protected set; }
```

Base map related to this map

Tiles

```
public ushort[,] Tiles
     { get;
     protected set; }
```

Tile that descripe the agent ownership

Methods

DisownTile

```
public void DisownTile(
     IntVector2 ownedTile,
     ushort ID)
```

Disowns this tile so other agents can claim it or to know its availible

ClaimTile

```
public bool ClaimTile(
     IntVector2 newTilePosition,
     ushort ID)
```

Attempts to claim a tile for a specified ID

IsBlocked

```
internal bool IsBlocked(
     int x,
     int y,
     ushort CompareID)
```

ValidTileForAgent

```
public bool ValidTileForAgent(
     ushort simNavigationAgentID,
     int x,
     int y)
```

C# BytePointMap.cs

Namespaces

HPJ.Simulation.Map

```
namespace HPJ.Simulation.Map
```

Classes

BytePointMap

```
[System.Serializable]
public class BytePointMap
```

A Flattenned version of a map that changes every tile type into a 1 or 0. Is it traverable or is it not

Constructors

BytePointMap

```
public  BytePointMap(
     byte[,] TileTypeMap,
     TileTypes[] TraversableTiles,
     int TileMapWidth,
     int TileMapLength,
     bool CornerCuttingEnabled)
```

Variables

_tiles

```
private byte[,] _tiles
```

_width

```
private int _width
```

_length

```
private int _length
```

_cornerCutting

```
private bool _cornerCutting
```

_traversableKey

```
private string _traversableKey
```

_traversable

```
private TileTypes[] _traversable
```

Properties

Tiles

```
public byte[,] Tiles
     { get;
     }
```

The Tiles for the Byte Map

Width

```
public int Width
     { get;
     }
```

The Width of the Byte Map

Length

```
public int Length
     { get;
     }
```

The Length of the Byte Map

CornerCutting

```
public bool CornerCutting
     { get;
     set; }
```

Whether corner cuttins is enabled on this map

TraversableKey

```
public string TraversableKey
     { get;
     }
```

The Key used to flatten the map

Traversable

```
public TileTypes[] Traversable
     { get;
     }
```

The Tiles that are traversable on this map

Methods

ChangeTile

```
public void ChangeTile(
     int x,
     int y,
     TileTypes newType)
```

Changes the tile of the byte map

ChangeTile

```
public void ChangeTile(
     IntVector2 TileIndex,
     TileTypes newType)
```

Changes the tile of the byte map

GetTileSpeed

```
public byte GetTileSpeed(
     int x,
     int y)
```

Gets the speed of the tile

C# ChangeMapJob.cs

Namespaces

HPJ.Simulation.Map

```
namespace HPJ.Simulation.Map
```

Classes

ChangeMapJob

```
[System.Serializable]
public class ChangeMapJob
```

Constructors

ChangeMapJob

```
public  ChangeMapJob(
     List<(IntVector2, TileTypes)> TilesChanged)
```

ChangeMapJob

```
public  ChangeMapJob()
```

Variables

TileTypes)> TilesToBeChanged

```
public List<(IntVector2, TileTypes)> TilesToBeChanged
```

Methods

AddTileChange

```
public void AddTileChange(
     IntVector2 TileIndex,
     TileTypes TypeChange)
```

C# Map.cs

Namespaces

HPJ.Simulation.Map

```
namespace HPJ.Simulation.Map
```

Classes

Map

```
public class Map
```

The Map that represents a area with square tiles

Constructors

Map

```
public  Map(
     MapSettings mapSettings,
     string MapName,
     int DegreeOfParallelism,
     bool CreateAgentMap)
```

Variables

_name

```
protected string _name
```

_tiles

```
protected byte[,] _tiles
```

_settings

```
public MapSettings _settings
```

_pointMaps

```
protected Dictionary<string,BytePointMap> _pointMaps
```

_bridges

```
public List<MapBridge> _bridges
```

_seams

```
public List<MapSeam> _seams
```

_parallelOptions

```
protected ParallelOptions _parallelOptions
```

The Parrallel options used by the map for the multithreading of each Unit that wants pathfinding

DebugLogCallback

```
public Action<string,SimulationDebugLog> DebugLogCallback
```

The Debug callback to Unity

_requestedJobs

```
protected List<NavigationJob> _requestedJobs
```

A List of the Requested Navigation Jobs

_requestedMapChanges

```
protected List<ChangeMapJob> _requestedMapChanges
```

A List of the Requested Map Changes

_agentClaimRequests

```
protected List<TileOwnershipClaim> _agentClaimRequests
```

A List of the Requested Tile Ownership claims

_agentDisownmentClaimRequests

```
protected List<DisownTileOwnership>
_agentDisownmentClaimRequests
```

A List of the Requested Tile Disownment claims

_needToBeUpdatedPaths

```
protected List<NavigationPath> _needToBeUpdatedPaths
```

A List of all the saved paths that need to be updated because they were invalidated or are new

_savedPaths

```
protected Dictionary<PathingInfo,NavigationPath>
_savedPaths
```

A Dictionary of all the saved paths on this Map

Properties

Name

```
public string Name
     { get;
     }
```

The Name of the Map

Tiles

```
public byte[,] Tiles
     { get;
     }
```

A 2D Array of the tiles of the Map in bytes

Settings

```
public MapSettings Settings
     { get;
     }
```

The Map Settings

Initialized

```
public bool Initialized
     { get;
     protected set; }
```

If the Map has been Initialized

Running

```
public bool Running
     { get;
     protected set; }
```

If the Map logic is currently running

AgentAvoidence

```
public bool AgentAvoidence
     { get;
     protected set; }
```

If the Map uses agent avoidence

CompletingNavigationJobs

```
public bool CompletingNavigationJobs
     { get;
     protected set; }
```

If the map is currently completing navigation job requests

CompletingMapChanges

```
public bool CompletingMapChanges
     { get;
     protected set; }
```

If the map is currently completing navigation map change requests

CompletingClaims

```
public bool CompletingClaims
     { get;
     protected set; }
```

If the map is currently completing tile ownership claims

CompletingDisownment

```
public bool CompletingDisownment
     { get;
     protected set; }
```

If the map is currently completing tile disownment claims

PointMaps

```
public Dictionary<string,BytePointMap> PointMaps
     { get;
     }
```

The Byte point maps used by the JPS pathfinding for each valid tile types

Bridges

```
public List<MapBridge> Bridges
     { get;
     }
```

A List of all the bridges connected to this Map

Seams

```
public List<MapSeam> Seams
     { get;
     }
```

A List of all the seams connected to this Map

AgentOwnershipMap

```
public AgentMap AgentOwnershipMap
     { get;
     protected set; }
```

The map copy that keeps record of tile ownership by agent

Methods

Initialize

```
public void Initialize(
     Action<string, SimulationDebugLog> debugLogCallback)
```

Initialized the Map

OnDestroy

```
public void OnDestroy()
```

Log

```
public void Log(
     string Log)
```

Debug.Log

LogWarning

```
public void LogWarning(
     string Log)
```

Debug.Warning

LogError

```
public void LogError(
     string Log)
```

Debug.Error

GetBytePointMap

```
internal BytePointMap GetBytePointMap(
     NavigationPath path)
```

Gets or Generates a Byte point map based on a set of traversable tiles types

GetTileSpeedData

```
internal byte GetTileSpeedData(
     TileTypes TileType)
```

Gets the movement speed for a specific tile type on this Map

Update

```
public void Update()
```

The Update function of the map each update tick

JobCompatible

```
internal bool JobCompatible(
     NavigationJob job)
```

A Check to see if the navigation job can be run on this map

PointInMap

```
public bool PointInMap(
     IntVector3 WorldPosition)
```

Checks to see if this point is valid on this map

PointInMap

```
public bool PointInMap(
     IntVector2 TilePosition)
```

Checks to see if this point is valid on this map

AddMapChange

```
public void AddMapChange(
     ChangeMapJob MapChange)
```

Adds a Map change Request for this map

AddClaim

```
public void AddClaim(
     TileOwnershipClaim Claim)
```

Adds a agent tile request claim for this map

AddDisownmentClaim

```
public void AddDisownmentClaim(
     DisownTileOwnership Claim)
```

Adds a agent tile disownment claim for this map

ChangeMap

```
protected void ChangeMap(
     ChangeMapJob MapChange)
```

Changes the map tiles based on the map change requests

Claim

```
protected void Claim(
     TileOwnershipClaim Claim)
```

Agent claims a tile

Disown

```
protected void Disown(
     DisownTileOwnership Claim)
```

Agent disowns a claims to a tile

SetTile

```
protected void SetTile(
     IntVector2 TilePosition,
     TileTypes TypeChange)
```

Sets the tile to the new type

GetTileType

```
public TileTypes GetTileType(
     IntVector3 WorldPosition)
```

Gets the current tile type of this tile index

GetTileType

```
public TileTypes GetTileType(
     IntVector2 TilePosition)
```

Gets the current tile type of this tile index

GetTileType

```
public TileTypes GetTileType(
     int x,
     int y)
```

Gets the current tile type of this tile index

GetTileIndexPosition

```
public IntVector2 GetTileIndexPosition(
     IntVector3 WorldTileIndexPosition)
```

Gets the current tile index at this world position

AddNavigationJob

```
public void AddNavigationJob(
     NavigationJob Job)
```

Adds a navigation Request to the Map

CompletePathfind

```
protected void CompletePathfind(
     NavigationPath Path)
```

Calculates the Requested Path on the Map

FinishPathfinding

```
protected void FinishPathfinding(
     NavigationJob Job)
```

Comples the Navigation Job Request

MapSettings

```
[Serializable]
public class MapSettings
```

The Map settings used to shape the Map

Constructors

MapSettings

```
public  MapSettings()
```

Variables

TileSize

```
public int TileSize
```

The Size of the tile in centimeters not meters

MapWidth

```
public int MapWidth
```

The Number of tiles in the x direction

MapHeight

```
public int MapHeight
```

The Number of tiles in the z direction

Offset

```
public IntVector3 Offset
```

The Physical offset of the map in centimeters

CornerCutting

```
public bool CornerCutting
```

If corner cutting is enabled in the pathfinding on this Map

TileData

```
public List<TileData> TileData
```

The Data for each tile on this Map

MapInfo

```
public MapData MapInfo
```

DefaultTraversaleTiles

```
public TileTypes[] DefaultTraversaleTiles
```

The Default traversable tiles for this Map

Methods

AdjustMap

```
public void AdjustMap(
     int TilesChanged,
     MapExpansionSide ExpansionSide,
     int NewOffset,
     int Translation = 0)
```

Adjusts the Map Size in a specific direction

UpdateMapSize

```
public void UpdateMapSize()
```

Updates the Map Size

MapBridge

```
[Serializable]
public class MapBridge
```

A Set of points that the navigation can use to transfer a Agent from one map to another

Constructors

MapBridge

```
public  MapBridge(
     MapSet A,
     MapSet B,
     List<IntVector3> BridgePoints)
```

Variables

MapA

```
public MapSet MapA
```

Map on Side A of the Bridge

MapB

```
public MapSet MapB
```

Map on Side B of the Bridge

WalkableDirections

```
public BridgeDirections WalkableDirections
```

Which ways the bridge can be used

MidPoints

```
public List<IntVector3> MidPoints
```

All the points from Map A to Map B

ValidBridge

```
public bool ValidBridge
```

If this map is a valid and can be used

Methods

DirectConnects

```
public bool DirectConnects(
     MapSet startingMap,
     MapSet endingMap)
```

Checks to see this bridge connects two maps

CanWalkInDirection

```
public bool CanWalkInDirection(
     MapSet startingMap,
     MapSet endingMap)
```

Checks to see if you can walk on the bridge in this direction. Starting Map to Ending Map

GetPoint

```
public IntVector3 GetPoint(
     MapSet startingMap)
```

Gets the starting point for a Map if that map connects to this Bridge

GetOpposingPoint

```
public IntVector3 GetOpposingPoint(
     MapSet startingMap)
```

Gets the ending point for a Map if that map connects to this Bridge

Contains

```
public bool Contains(
     MapSet Map)
```

Checks to see if the given maps is on one side of the Bridge

Other

```
public MapSet Other(
     MapSet Map)
```

Gets the other Map if you know one side the bridge connects too

RefreshWithNew

```
public void RefreshWithNew(
     MapSet Map)
```

Verify

```
public void Verify()
```

MapSeam

```
[Serializable]
public class MapSeam
```

A Set of points that the navigation can use to transfer a Agent from one map to another

Constructors

MapSeam

```
public  MapSeam(
     MapSet A,
     MapSet B,
     SeamConnectionSide mapA_ConectionSide,
     SeamConnectionSide mapB_ConectionSide)
```

Variables

MapA

```
public MapSet MapA
```

Map on Side A of the Bridge

MapA_ConectionSide

```
public SeamConnectionSide MapA_ConectionSide
```

The connection side for Map A

MapA_StartingTileIndex

```
public IntVector2 MapA_StartingTileIndex
```

The Map A Starting Tile Index

MapA_EndingTileIndex

```
public IntVector2 MapA_EndingTileIndex
```

The Map A Ending Tile Index

MapB

```
public MapSet MapB
```

Map on Side B of the Bridge

MapB_ConectionSide

```
public SeamConnectionSide MapB_ConectionSide
```

The connection side for Map A

MapB_StartingTileIndex

```
public IntVector2 MapB_StartingTileIndex
```

The Map B Starting Tile Index

MapB_EndingTileIndex

```
public IntVector2 MapB_EndingTileIndex
```

The Map B Ending Tile Index

ValidSeam

```
public bool ValidSeam
```

If this map is a valid and can be used

Methods

DirectConnects

```
public bool DirectConnects(
     MapSet startingMap,
     MapSet endingMap)
```

Checks to see this bridge connects two maps

Contains

```
public bool Contains(
     MapSet Map)
```

Checks to see if the given maps is on one side of the Bridge

Other

```
public MapSet Other(
     MapSet Map)
```

Gets the other Map if you know one side the bridge connects too

RefreshWithNew

```
public void RefreshWithNew(
     MapSet Map)
```

Verify

```
public void Verify()
```

GetPoint

```
public IntVector3 GetPoint(
     MapSet startingMap,
     IntVector3 start)
```

Gets the closest valid point for the starting map to the seam

GetOppositePoint

```
public IntVector3 GetOppositePoint(
     MapSet startingMap,
     IntVector3 start)
```

MapData

```
[System.Serializable]
public class MapData
```

Tile Size Data for the Map, used to save default tiles to the Map

Constructors

MapData

```
public  MapData()
```

Variables

SavedTiles

```
public byte[,] SavedTiles
```

Width

```
public int Width
```

Height

```
public int Height
```

Methods

MakeSureSizeIsRight

```
public bool MakeSureSizeIsRight()
```

UpdateToMapSize

```
public void UpdateToMapSize(
     int mapWidth,
     int mapHeight)
```

Updates the size of the Data

TransalateValues

```
public void TransalateValues(
     int HorizontalTranslation,
     int VerticalTranslation)
```

Moves the values of the tiles

CleanMap

```
public void CleanMap()
```

Clears the map of all tile types and turns them into standard tiles

TileData

```
[System.Serializable]
public class TileData
```

The Data that tells the speed multiplier for a given tile type

Constructors

TileData

```
public  TileData()
```

TileData

```
public  TileData(
     TileTypes tileType,
     byte speedMult)
```

Variables

Type

```
public TileTypes Type
```

Speed

```
public byte Speed
```

ObstacleData

```
[Serializable]
public struct ObstacleData
```

Used to keep track of a obstacles tiles and its state in the obstacle

Constructors

ObstacleData

```
public  ObstacleData(
     IntVector3 Index)
```

Variables

TileIndexKey

```
public IntVector3 TileIndexKey
```

ValidationState

```
public ObstacleValidationStates ValidationState
```

Methods

UnValidate

```
public void UnValidate()
```

Means this tile is no longer valid and will need to be checked if it still belongs on this obstacle

Validate

```
public void Validate()
```

Validates the tile, it belongs on this obstacle

C# MapSet.cs

Namespaces

HPJ.Simulation.Map

```
namespace HPJ.Simulation.Map
```

Classes

MapSet

```
[Serializable]
public class MapSet
```

The set of maps (base and redundant) that represent a specific area

Constructors

MapSet

```
public  MapSet()
```

Variables

SetSettings

```
public MapSetSettings SetSettings
```

The Settings for this Map Set

InstanceID

```
public int InstanceID
```

The Instance ID to distinguish each map set apart

DebugLogCallback

```
public Action<string,SimulationDebugLog> DebugLogCallback
```

The Callback to links back up to Unity. Cannot use Debug.Log() in the simulation side of the navigation

_redundantMapCounter

```
private int _redundantMapCounter
```

Properties

Initialized

```
public bool Initialized
     { get;
     protected set; }
```

Shows if this Map Set is initialized or not

BaseMap

```
public Map BaseMap
     { get;
     protected set; }
```

The Base map that comes with each Map Set

RedundantMaps

```
public Map[] RedundantMaps
     { get;
     protected set; }
```

The Redundant Maps that help are used to help reduce queue times for each Navigation Agent

MapBridges

```
public List<MapBridge> MapBridges
     { get;
     set; }
```

The List of Bridges that connect to this Map Set

MapSeams

```
public List<MapSeam> MapSeams
     { get;
     set; }
```

The List of Seams that connect to this Map Set

KeyBorderPositions

```
public List<IntVector3> KeyBorderPositions
     { get;
     set; }
```

The List of Key points around the border of the Map Set

BottomLeftPosition

```
public IntVector3 BottomLeftPosition
     { get;
     set; }
```

Bottom Left Corner of the Map

LeftPosition

```
public IntVector3 LeftPosition
     { get;
     set; }
```

Left Center Position of the Map

BottomPosition

```
public IntVector3 BottomPosition
     { get;
     set; }
```

Bottom Center Position of the Map

BottomRightPosition

```
public IntVector3 BottomRightPosition
     { get;
     set; }
```

Bottom Right Corner of the Map

TopRightPosition

```
public IntVector3 TopRightPosition
     { get;
     set; }
```

Top Right Corner of the Map

RightPosition

```
public IntVector3 RightPosition
     { get;
     set; }
```

Right Center Position of the Map

TopPosition

```
public IntVector3 TopPosition
     { get;
     set; }
```

Top Center Position of the Map

TopLeftPosition

```
public IntVector3 TopLeftPosition
     { get;
     set; }
```

Top Left Corner of the Map

MiddlePosition

```
public IntVector3 MiddlePosition
     { get;
     set; }
```

Middle Position of the Map Position of the Map

OutgoingClaims

```
public Dictionary<IntVector2,TileOwnershipClaim>
OutgoingClaims
     { get;
     protected set; }
```

A dictionary to keep track of the outgoing claims

Methods

Initialize

```
public void Initialize(
     int Parrelism,
     int Redundancy,
     bool CreateAgentMap,
     Action<string, SimulationDebugLog> debugLogCallback)
```

Initilizes the Map Set

OnDestroy

```
public void OnDestroy()
```

Log

```
public void Log(
     string Log)
```

Logs the debugs occured in the simulation

LogWarning

```
public void LogWarning(
     string Log)
```

Logs the warning occured in the simulation

LogError

```
public void LogError(
     string Log)
```

Logs the errors occured in the simulation

SetDestination

```
public void SetDestination(
     NavigationJob Job)
```

The Call a navigation job sends to the map to be calculated

JobCompatible

```
public bool JobCompatible(
     NavigationJob job)
```

Checks to see if this job is compatible with this map set. For example a job that has its start in another map is not compatible with this map set

OutOfRange

```
public bool OutOfRange(
     IntVector2 newTilePosition)
```

Whether this index is out of range of the map

ChangeMap

```
public void ChangeMap(
     ChangeMapJob MapJob)
```

The Call that changes the tiles on each map in the Map Set

AttemptClaim

```
public bool AttemptClaim(
     IntVector2 Tile,
     ushort AgentID)
```

Requests to claim a tile

DisownClaim

```
public bool DisownClaim(
     IntVector2 Tile,
     ushort AgentID)
```

Disowns a claim to a tile

ClearClaims

```
public void ClearClaims()
```

Clears the list of claims

Claim

```
protected void Claim(
     TileOwnershipClaim Claim)
```

Adds a agent tile request claim for this map set

CleanMapTiles

```
public void CleanMapTiles()
```

This Changed the DATA the Default data and makes all the tiles to the default tile type (TileTypes.Standard)

GetClosestPoint

```
public IntVector3 GetClosestPoint(
     IntVector3 reachPoint)
```

The Closest point on the map to the reach point

GetClosestValidPoint

```
public IntVector3 GetClosestValidPoint(
     IntVector3 ClosestPoint,
     TileTypes[] ValidTypes)
```

The Closest valid point to the reach point

GetClosestValidPoint

```
public IntVector3 GetClosestValidPoint(
     IntVector3 ClosestPoint,
     List<TileTypes> ValidTypes)
```

The Closest valid point to the reach point

GetClosestValidPoint

```
public IntVector3 GetClosestValidPoint(
     IntVector2 CurrentTileIndex,
     TileTypes[] ValidTypes)
```

The Closest valid point to the current tile index

GetClosestValidPoint

```
public IntVector3 GetClosestValidPoint(
     IntVector2 CurrentTileIndex,
     List<TileTypes> ValidTypes)
```

The Closest valid point to the current tile index

GetClosestValidIndex

```
public IntVector2 GetClosestValidIndex(
     IntVector2 CurrentTileIndex,
     TileTypes[] ValidTypes)
```

The Closest valid tile index to the current tile index

GetClosestValidIndex

```
public IntVector2 GetClosestValidIndex(
     IntVector2 CurrentTileIndex,
     List<TileTypes> ValidTypes)
```

The Closest valid tile index to the current tile index

GetClosestValidIndex

```
public IntVector2 GetClosestValidIndex(
     IntVector2 CurrentTileIndex,
     List<TileTypes> ValidTypes,
     ushort SimAgentID)
```

The Closest valid tile index to the current tile index

GetWorldTileIndex

```
public IntVector3 GetWorldTileIndex(
     IntVector2 newTileIndex,
     bool GetTileCenter = true)
```

Gets the world tile index relative to the tile index of the map set

GetWorldTileIndex

```
public IntVector3 GetWorldTileIndex(
     int x,
     int y,
     bool GetTileCenter = true)
```

Gets the world tile index relative to the tile index of the map set

GetMapTileIndex

```
public IntVector2 GetMapTileIndex(
     IntVector3 WorldTileIndexPosition)
```

This gets you the Tile Index of the Default Map

PointInMap

```
public bool PointInMap(
     IntVector3 hitTileIndexPosition)
```

Checks to see if this point is on the Map

PointInMap

```
public bool PointInMap(
     IntVector2 TileIndex)
```

Checks to see if this point is on the Map

ChangeDefaultTileType

```
public void ChangeDefaultTileType(
     IntVector3 WorldTileIndexPosition,
     TileTypes NewType,
     int Radius,
     PainterTypes painterType)
```

This Changed the DATA the Default data, the maps that are generated on awake are not effected by this

ChangeDefaultTileType

```
public void ChangeDefaultTileType(
     int x,
     int y,
     TileTypes NewType)
```

This Changed the DATA the Default data, the maps that are generated on awake are not effected by this

GetDefaultTileType

```
public TileTypes GetDefaultTileType(
     IntVector2 TileIndex)
```

This gets you the Tile Type at a Index from the Default Map

GetDefaultTileType

```
public TileTypes GetDefaultTileType(
     int x,
     int y)
```

This gets you the Tile Type at a Index from the Default Map

GetTileType

```
public TileTypes GetTileType(
     IntVector2 TileIndex)
```

This gets you the Tile Type at a Index from the Map

GetTileType

```
public TileTypes GetTileType(
     IntVector3 worldTileIndex)
```

This gets you the Tile Type at a World Position Index from the Map

GetTileType

```
public TileTypes GetTileType(
     int x,
     int y)
```

This gets you the Tile Type at a Index from the Default Map

AddBridge

```
public void AddBridge(
     MapBridge bridge)
```

Adds the bridge if this bridge is connected to this Map

AddSeam

```
public void AddSeam(
     MapSeam seam)
```

Adds the seam if this seam is connected to this Map

ClearMaps

```
public void ClearMaps()
```

Clears the Maps and the Bridges connected to this Map

GetClosestBridge

```
public MapBridge GetClosestBridge(
     MapSet nextMap,
     IntVector3 start)
```

Finds the closest bridge that connects the destination Map

GetClosestSeam

```
public MapSeam GetClosestSeam(
     MapSet nextMap,
     IntVector3 start)
```

Finds the closest seam that connects the destination Map

GetClosestMapTileIndex

```
public IntVector2 GetClosestMapTileIndex(
     IntVector2 tileIndex)
```

Gets the closest Map Tile Index based on a tile index relative to this Map

GetClosestMapTileIndex

```
public IntVector2 GetClosestMapTileIndex(
     IntVector3 worldIndex)
```

Gets the closest Map Tile Index based on a tile index relative to this Map

MapSetSettings

```
[Serializable]
public class MapSetSettings
```

The settings of the Map Set

Constructors

MapSetSettings

```
public  MapSetSettings()
```

Variables

Name

```
public string Name
```

The Name of the Map

MapSettings

```
public MapSettings MapSettings
```

The Settings for each Map on the Map Set

C# TileOwnershipClaim.cs

Namespaces

HPJ.Simulation

```
namespace HPJ.Simulation
```

Classes

TileOwnershipClaim

```
public struct TileOwnershipClaim
```

Constructors

TileOwnershipClaim

```
public  TileOwnershipClaim(
     ushort ID,
     IntVector2 Tile)
```

Variables

OwnerID

```
public ushort OwnerID
```

RequestTile

```
public IntVector2 RequestTile
```

DisownTileOwnership

```
public struct DisownTileOwnership
```

Constructors

DisownTileOwnership

```
public  DisownTileOwnership(
     ushort ID,
     IntVector2 Tile)
```

Variables

OwnerID

```
public ushort OwnerID
```

RequestTile

```
public IntVector2 RequestTile
```

2.Pathfinding

AStar.cs

Namespaces

HPJ.Simulation.Pathing

```
namespace HPJ.Simulation.Pathing
```

Classes

AStar

```
public class AStar
```

Classes

AStarTileData

```
public struct AStarTileData
```

Constructors

AStarTileData

```
public  AStarTileData(
     IntVector2 RouteFromTile,
     int EndCost)
```

Variables

RouteFromTileIndex

```
public IntVector2 RouteFromTileIndex
```

FCost

```
public int FCost
```

GCost

```
public int GCost
```

HCost

```
public int HCost
```

Variables

Instance

```
public static AStar Instance
```

Methods

Initialize

```
public override void Initialize()
```

CalculatePath

```
public override void CalculatePath(
     NavigationPath Path,
     Map.Map OccuringMap,
     out string Status,
     out bool Succeeded)
```

GetLowestFCostTileIndex

```
protected int GetLowestFCostTileIndex(
     List<IntVector2> OpenList,
     Map.Map OccuringMap,
     IntVector2 Start,
     IntVector2 End,
     IntVector2 CurrentTileIndex,
     TileTypes[] PrefferedTiles)
```

CalculateDistanceCost

```
protected int CalculateDistanceCost(
     int x1,
     int y1,
     int x2,
     int y2)
```

SortPath

```
protected void SortPath(
     NavigationPath Path,
     Dictionary<IntVector2, AStarTileData> SortedData,
     IntVector2 EndTileIndex)
```

C# JumpPointSearch.cs

Namespaces

HPJ.Simulation.Pathing

```
namespace HPJ.Simulation.Pathing
```

Classes

JumpPointSearch

```
public class JumpPointSearch
```

A Jump point search script used by Navigation Jobs to calculate a path based on the jump point search (JPS) Algorithm

Classes

JPS_VariablePackage

```
public class JPS_VariablePackage
```

Constructors

JPS_VariablePackage

```
public  JPS_VariablePackage(
     bool AgentAvoidence)
```

Variables

OpenList

```
public List<JPSTile> OpenList
```

SortedList

```
public Dictionary<IntVector2,JPSTile> SortedList
```

UpRightData

```
public CurrentJunpData UpRightData
```

UpLeftData

```
public CurrentJunpData UpLeftData
```

DownRightData

```
public CurrentJunpData DownRightData
```

DownLeftData

```
public CurrentJunpData DownLeftData
```

Pool

```
public static List<JPS_VariablePackage> Pool
```

TilePool

```
public List<JPSTile> TilePool
```

Methods

LoadJumpData

```
public void LoadJumpData(
     IntVector2 Start,
     IntVector2 End)
```

Clear

```
public void Clear()
```

GetPackage

```
public static JPS_VariablePackage GetPackage(
     bool Avoidence)
```

ReturnPackage

```
public static void ReturnPackage(
     JPS_VariablePackage Package)
```

GetTile

```
public JPSTile GetTile(
     IntVector2 position,
     int EndCost)
```

ReturnTile

```
public void ReturnTile(
     JPSTile Package)
```

SetAgentID

```
internal void SetAgentID(
     ushort agentID)
```

JPSTile

```
[Serializable]
public class JPSTile
```

Constructors

JPSTile

```
public  JPSTile(
     IntVector2 position,
     int EndCost)
```

Variables

Position

```
public IntVector2 Position
```

SplitPosition

```
public IntVector2 SplitPosition
```

JumpFromTile

```
public JPSTile JumpFromTile
```

HCost

```
public int HCost
```

GCost

```
public int GCost
```

SplitGCost

```
public int SplitGCost
```

HitAdjacentJump

```
public bool HitAdjacentJump
```

Methods

Clear

```
public void Clear()
```

Reset

```
public void Reset(
     IntVector2 position,
     int EndCost)
```

Clone

```
public JPSTile Clone()
```

Clone

```
public JPSTile Clone(
     JPS_VariablePackage Package)
```

CurrentJunpData

```
[Serializable]
public class CurrentJunpData
```

Constructors

CurrentJunpData

```
public  CurrentJunpData(
     IntVector2 Direciton,
     IntVector2 Start,
     IntVector2 End,
     bool Avoidence)
```

CurrentJunpData

```
public  CurrentJunpData(
     IntVector2 Direciton,
     bool Avoidence)
```

Variables

JumpDirection

```
public IntVector2 JumpDirection
```

StartIndex

```
public IntVector2 StartIndex
```

EndIndex

```
public IntVector2 EndIndex
```

AgentAvoidence

```
public bool AgentAvoidence
```

Properties

AgentID

```
public ushort AgentID
     { get;
     internal set; }
```

Methods

SetData

```
public void SetData(
     IntVector2 Start,
     IntVector2 End)
```

Variables

Instance

```
public static JumpPointSearch Instance
```

Methods

Initialize

```
public override void Initialize()
```

CalculatePath

```
public override void CalculatePath(
     NavigationPath Path,
     Map.Map OccuringMap,
     out string Status,
     out bool Succeeded)
```

DiagonalJump

```
private JPSTile DiagonalJump(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

HorizontalJump

```
private JPSTile HorizontalJump(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

VerticalJump

```
private JPSTile VerticalJump(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

HorizontalJump_FromDiagonal

```
private void HorizontalJump_FromDiagonal(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

VerticalJump_FromDiagonal

```
private void VerticalJump_FromDiagonal(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

IsDiagonalForced

```
protected bool IsDiagonalForced(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

IsHorizontalForced

```
protected bool IsHorizontalForced(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

IsVerticalForced

```
protected bool IsVerticalForced(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

IsHorizontalForced_Diagonal

```
protected bool IsHorizontalForced_Diagonal(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

IsVerticalForced_Diagonal

```
protected bool IsVerticalForced_Diagonal(
     BytePointMap BitMap,
     AgentMap TileOwnership,
     JPSTile PlaceholderTile,
     CurrentJunpData JumpData)
```

IsBlocked

```
protected bool IsBlocked(
     int x,
     int y,
     BytePointMap ByteMap,
     AgentMap TileOwnership,
     ushort ID)
```

GetLowestFCostTileIndex

```
protected int GetLowestFCostTileIndex(
     List<JPSTile> OpenList,
     int TileEndX,
     int TileEndY)
```

SortPath

```
protected void SortPath(
     Dictionary<IntVector2, JPSTile> SortedData,
     NavigationPath Path,
     IntVector2 EndTileIndex)
```

CalculateDistanceCost

```
internal int CalculateDistanceCost(
     int x1,
     int y1,
     int x2,
     int y2)
```

C# PathingCalculation.cs

Namespaces

HPJ.Simulation.Pathing

```
namespace HPJ.Simulation.Pathing
```

Classes

PathingCalculation

```
public abstract class PathingCalculation
```

The Base class for any pathfinding algorithms

Constructors

PathingCalculation

```
public  PathingCalculation()
```

Variables

PathingTypes

```
public static
Dictionary<NavigationTypes,PathingCalculation>
PathingTypes
```

The Dictionary of all the pathing algorithms

Properties

NavigationType

```
public NavigationTypes NavigationType
     { get;
     protected set; }
```

The Navigation Type this pathing is. Will need to add yours to the enum

Methods

Initialize

```
public virtual void Initialize()
```

Initializes the Algorithm, You will need to call this or add this for your algorithm

CalculatePath

```
public abstract void CalculatePath(
     NavigationPath Path,
     Map.Map OccuringMap,
     out string Status,
     out bool Succeeded)
```

Calculates the Path for a Navigation Job Request on a specific Map

NavigationJob

```
[Serializable]
public class NavigationJob
```

A Navigation Job Request

Constructors

NavigationJob

```
public  NavigationJob(
     Action onCompleteCallback)
```

Variables

Info

```
public PathingInfo Info
```

Infomation on this Job

CurrentStep

```
public PathfindingCalculationStep CurrentStep
```

Current Step on the Pathfinding Calculation

TilesAway

```
public int TilesAway
```

The Number of tiles away from the start to the end

WorldPointA

```
public IntVector3 WorldPointA
```

Point A of the Path

WorldPointB

```
public IntVector3 WorldPointB
```

Point B of the Path

Compatable

```
public bool Compatable
```

If this Job was compatable or not

RepathSavedPath

```
public bool RepathSavedPath
```

Next time this path is pathed, repath it

TraverableTileTypes

```
public List<TileTypes> TraverableTileTypes
```

List of Traversable Tiles

PrefferedTileTypes

```
public List<TileTypes> PrefferedTileTypes
```

List of Prefferred Tiles

OnCompleteCallback

```
public Action OnCompleteCallback
```

The Callback to the presentation side

Properties

CurrentPath

```
public NavigationPath CurrentPath
     { get;
     protected set; }
```

The Resulting Path for this Job Request

AgentID

```
public ushort AgentID
     { get;
     set; }
```

Methods

Clear

```
public void Clear()
```

Clears the Job and reverts it back to default

SetDestinationInfo

```
public void SetDestinationInfo(
     IntVector3 Start,
     IntVector3 End,
     List<TileTypes> TraverableTiles,
     List<TileTypes> PrefferedTiles,
     NavigationTypes Navigation,
     MapSet ForMap,
     bool RepathAnyway = false)
```

Initializes the Job with the neccessary pathing information

SetNewCurrentPath

```
internal void SetNewCurrentPath(
     NavigationPath path)
```

Sets the resulting path to the request

PathfindingComplete

```
internal void PathfindingComplete()
```

The callback from the simulation when the job request is finished

NavigationPath

```
[System.Serializable]
public class NavigationPath
```

The Calculated Path on a specific Map

Constructors

NavigationPath

```
public  NavigationPath()
```

NavigationPath

```
public  NavigationPath(
     PathingInfo PathInfo,
     Map.Map hostMap,
     ushort ID = 0)
```

Variables

HostMap

```
public Map.Map HostMap
```

The map the path was calculated on

Info

```
public PathingInfo Info
```

The Information on this Path

Path

```
public List<IntVector2> Path
```

The Resulting Path

ValidPath

```
public bool ValidPath
```

If this Path resulting in a valid path

AgentID

```
public ushort AgentID
```

Asking For Path from specific agent

Properties

PreviousStatus

```
public string PreviousStatus
     { get;
     set; }
```

Previous status on the path

Methods

NeedToReverse

```
public bool NeedToReverse(
     IntVector3 startPoint)
```

Checks to see if your starting point is the first index of the path or the last

PathingInfo

```
[System.Serializable]
public struct PathingInfo
```

The information on a given path

Constructors

PathingInfo

```
public  PathingInfo(
     IntVector2 A,
     IntVector2 B,
     NavigationTypes Type,
     string Traverable,
     string Preffered)
```

Variables

PointA

```
public IntVector2 PointA
```

Point A of a Path

PointB

```
public IntVector2 PointB
```

Point B of a Path

NavigationType

```
public NavigationTypes NavigationType
```

Navigation type used for this Path

TraversableTilesKey

```
public string TraversableTilesKey
```

Traversable Tiles Key

PrefferedTilesKey

```
public string PrefferedTilesKey
```

Preffered Tile Kep

3.Enums

NavigationTypes.cs

Namespaces

HPJ.Simulation.Enums

```
namespace HPJ.Simulation.Enums
```

Enumerations

NavigationTypes

```
public enum NavigationTypes{
     None = 0,
     Free = 1,
     JumpPointSearch = 2,
     AStar = 3,
     Unknown = 4,}
```

The Navigation Types a agent can use for pathfinding

TileTypes

```
public enum TileTypes
: byte{
     Standard = 0,
     Blocked = 1,
     Free = 2,
     Void = 3,
     Obstacle = 4,
     Water = 5,
     OutOfBounds = 6,}
```

The Navigation Tile Types that are availble on a Map, add more as needed

BridgeDirections

```
public enum BridgeDirections
: byte{
     Both = 0,
     A_To_B = 1,
     B_To_A = 2,}
```

Bridge Direction Types

RightOfWay

```
public enum RightOfWay
: byte{
     NorthWest = 0,
     SouthEast = 1,}
```

Who has right of way based on your moving direction

AvoidenceType

```
public enum AvoidenceType
: byte{
     StopAndGo = 0,
     TryToPathAround = 1,}
```

How agents try to avoid each other

SeamConnectionSide

```
public enum SeamConnectionSide
: byte{
     North = 0,
     West = 1,
     South = 2,
     East = 3,}
```

The seam connection side

DistanceCheck

```
public enum DistanceCheck{
     Manhatten = 0,
     Distance = 1,
     SqrDistance,}
```

MovementType

```
public enum MovementType{
     FourDirection = 0,
     EightDirection = 1,}
```

PathfindingCalculationStep

```
public enum PathfindingCalculationStep{
     Incomplete = 0,
     Pathfinding = 1,
     Complete = 2,
     Finalized = 3,}
```

SimulationDebugLog

```
public enum SimulationDebugLog{
     Log = 0,
     Warning = 1,
     Error = 2,}
```

MapExpansionSide

```
public enum MapExpansionSide{
     Top = 0,
     Bottom = 1,
     Left = 2,
     Right = 3,}
```

PainterTypes

```
public enum PainterTypes{
     Square = 0,
     Circle = 1,}
```

ObstacleValidationStates

```
public enum ObstacleValidationStates
: byte{
     Validated = 0,
     Invalidated = 1,
     Added = 2,}
```

AgentStates

```
public enum AgentStates
: byte{
     Stopped = 0,
     Paused = 1,
     Pathfinding = 2,
     Moving = 3,
     FailedPathing = 4,
     SucceededPathing = 5,
     Idle = 6,}
```

DestinationStates

```
public enum DestinationStates
: byte{
     None = 0,
     SameMap = 1,
     Bridge = 2,
     NextMap_FromBridge = 3,
     ClosestPoint = 4,
     Null = 5,
     SamePosition = 6,
     Seam = 7,
     NextMap_FromSeam = 8,}
```

4.Utilities

Extensions.cs

Namespaces

HPJ.Simulation.Utilities

```
namespace HPJ.Simulation.Utilities
```

Classes

Extensions

```
public static class Extensions
```

Methods

Distance

```
public static float Distance(
     this IntVector2 vector)
```

SqrDistance

```
public static int SqrDistance(
     this IntVector2 vector)
```

TilesToString

```
public static string TilesToString(
     this TileTypes[] Tiles)
```

TilesToString

```
public static string TilesToString(
     this List<TileTypes> Tiles)
```

StringToTilesArray

```
public static TileTypes[] StringToTilesArray(
     this string TilesStringArray)
```

IntVector2.cs

Namespaces

HPJ.Simulation

```
namespace HPJ.Simulation
```

Classes

IntVector2

```
[Serializable]
public struct IntVector2
```

Constructors

IntVector2

```
public  IntVector2(
     int x,
     int y)
```

Operators

operator +

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector2 operator +(
     IntVector2 a,
     IntVector2 b)
```

operator -

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector2 operator -(
     IntVector2 a,
     IntVector2 b)
```

operator -

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector2 operator -(
     IntVector2 a)
```

*operator **

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector2 operator *(
     IntVector2 a,
     int d)
```

*operator **

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector2 operator *(
     int d,
     IntVector2 a)
```

operator /

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector2 operator /(
     IntVector2 a,
     int d)
```

operator ==

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool operator ==(
     IntVector2 lhs,
     IntVector2 rhs)
```

operator !=

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool operator !=(
     IntVector2 lhs,
     IntVector2 rhs)
```

Variables

x

```
public int x
```

y

```
public int y
```

Properties

Zero

```
public static IntVector2 Zero
     { get;
     }
```

Returns the vector (0,0).

One

```
public static IntVector2 One
     { get;
     }
```

Returns the vector (1,1).

UnitX

```
public static IntVector2 UnitX
     { get;
     }
```

Returns the vector (1,0).

UnitY

```
public static IntVector2 UnitY
     { get;
     }
```

Returns the vector (0,1).

Methods

Equals

```
public override bool Equals(
     object obj)
```

ToString

```
public override string ToString()
```

Returns a String representing this Vector2 instance.

Returns: The string representation.

LengthSquared

```
public int LengthSquared()
```

Returns the length of the vector squared. This operation is cheaper than Length().

Returns: The vector’s length squared.

DistanceSquared

```
public static int DistanceSquared(
     IntVector2 value1,
     IntVector2 value2)
```

Returns the Euclidean distance squared between the two given points.

value1: The first point.
value2: The second point.

Returns: The distance squared.

Distance

```
public static float Distance(
     IntVector2 value1,
     IntVector2 value2)
```

Returns the Euclidean distance between the two given points.

value1: The first point.
value2: The second point.

Returns: The distance squared.

Clamp

```
public static IntVector2 Clamp(
     IntVector2 value1,
     IntVector2 min,
     IntVector2 max)
```

Restricts a vector between a min and max value.

value1: The source vector.
min: The minimum value.
max: The maximum value.

Negate

```
public static IntVector2 Negate(
     IntVector2 value)
```

// Adds two vectors together. //

left: The first source vector.
right: The second source vector.

Returns: The summed vector.

GetHashCode

```
public override int GetHashCode()
```

SqrMagnitude

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static int SqrMagnitude(
     IntVector2 vector)
```

IntVector3

```
[Serializable]
public struct IntVector3
```

Constructors

IntVector3

```
public  IntVector3(
     int x,
     int y,
     int z)
```

Operators

operator +

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector3 operator +(
     IntVector3 a,
     IntVector3 b)
```

operator -

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector3 operator -(
     IntVector3 a,
     IntVector3 b)
```

operator -

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector3 operator -(
     IntVector3 a)
```

*operator **

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector3 operator *(
     IntVector3 a,
     int d)
```

*operator **

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector3 operator *(
     int d,
     IntVector3 a)
```

operator /

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector3 operator /(
     IntVector3 a,
     int d)
```

operator ==

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool operator ==(
     IntVector3 lhs,
     IntVector3 rhs)
```

operator !=

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool operator !=(
     IntVector3 lhs,
     IntVector3 rhs)
```

Variables

x

```
public int x
```

y

```
public int y
```

z

```
public int z
```

Properties

Zero

```
public static IntVector3 Zero
     { get;
     }
```

Returns the vector (0,0,0).

One

```
public static IntVector3 One
     { get;
     }
```

Returns the vector (1,1,1).

UnitX

```
public static IntVector3 UnitX
     { get;
     }
```

Returns the vector (1,0,0).

UnitY

```
public static IntVector3 UnitY
     { get;
     }
```

Returns the vector (0,1,0).

UnitZ

```
public static IntVector3 UnitZ
     { get;
     }
```

Returns the vector (0,0,1).

Methods

xVector

```
public IntVector3 xVector()
```

yVector

```
public IntVector3 yVector()
```

zVector

```
public IntVector3 zVector()
```

xyVector

```
public IntVector3 xyVector()
```

yzVector

```
public IntVector3 yzVector()
```

xzVector

```
public IntVector3 xzVector()
```

Vector2

```
public IntVector2 Vector2()
```

Equals

```
public override bool Equals(
     object obj)
```

ToString

```
public override string ToString()
```

Returns a String representing this Vector2 instance.

Returns: The string representation.

LengthSquared

```
public int LengthSquared()
```

Returns the length of the vector squared. This operation is cheaper than Length().

Returns: The vector’s length squared.

DistanceSquared

```
public static int DistanceSquared(
     IntVector3 value1,
     IntVector3 value2)
```

Returns the Euclidean distance squared between the two given points.

value1: The first point.
value2: The second point.

Returns: The distance squared.

Clamp

```
public static IntVector3 Clamp(
     IntVector3 value1,
     IntVector3 min,
     IntVector3 max)
```

Restricts a vector between a min and max value.

value1: The source vector.
min: The minimum value.
max: The maximum value.

Negate

```
public static IntVector3 Negate(
     IntVector3 value)
```

// Adds two vectors together. //

left: The first source vector.
right: The second source vector.

Returns: The summed vector.

GetHashCode

```
public override int GetHashCode()
```

SqrMagnitude

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static int SqrMagnitude(
     IntVector3 vector)
```

Displacement

```
public static int Displacement(
     IntVector3 Position1,
     IntVector3 Position2)
```

IntVector4

```
[Serializable]
public struct IntVector4
```

Constructors

IntVector4

```
public  IntVector4(
     int x,
     int y,
     int z,
     int w)
```

Operators

operator +

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector4 operator +(
     IntVector4 a,
     IntVector4 b)
```

operator -

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector4 operator -(
     IntVector4 a,
     IntVector4 b)
```

operator -

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector4 operator -(
     IntVector4 a)
```

*operator **

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector4 operator *(
     IntVector4 a,
     int d)
```

*operator **

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector4 operator *(
     int d,
     IntVector4 a)
```

operator /

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IntVector4 operator /(
     IntVector4 a,
     int d)
```

operator ==

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool operator ==(
     IntVector4 lhs,
     IntVector4 rhs)
```

operator !=

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool operator !=(
     IntVector4 lhs,
     IntVector4 rhs)
```

Variables

x

```
public int x
```

y

```
public int y
```

z

```
public int z
```

w

```
public int w
```

Properties

Zero

```
public static IntVector4 Zero
     { get;
     }
```

Returns the vector (0,0,0,0).

One

```
public static IntVector4 One
     { get;
     }
```

Returns the vector (1,1,1,1).

UnitX

```
public static IntVector4 UnitX
     { get;
     }
```

Returns the vector (1,0,0,0).

UnitY

```
public static IntVector4 UnitY
     { get;
     }
```

Returns the vector (0,1,0,0).

UnitZ

```
public static IntVector4 UnitZ
     { get;
     }
```

Returns the vector (0,0,1,0).

Unitw

```
public static IntVector4 Unitw
     { get;
     }
```

Returns the vector (0,0,0,1).

Methods

Equals

```
public override bool Equals(
     object obj)
```

ToString

```
public override string ToString()
```

Returns a String representing this Vector2 instance.

Returns: The string representation.

LengthSquared

```
public int LengthSquared()
```

Returns the length of the vector squared. This operation is cheaper than Length().

Returns: The vector’s length squared.

DistanceSquared

```
public static int DistanceSquared(
     IntVector4 value1,
     IntVector4 value2)
```

Returns the Euclidean distance squared between the two given points.

value1: The first point.
value2: The second point.

Returns: The distance squared.

Clamp

```
public static IntVector4 Clamp(
     IntVector4 value1,
     IntVector4 min,
     IntVector4 max)
```

Restricts a vector between a min and max value.

value1: The source vector.
min: The minimum value.
max: The maximum value.

Negate

```
public static IntVector4 Negate(
     IntVector4 value)
```

// Adds two vectors together. //

left: The first source vector.
right: The second source vector.

Returns: The summed vector.

GetHashCode

```
public override int GetHashCode()
```

SqrMagnitude

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static int SqrMagnitude(
     IntVector4 vector)
```

C# Pool.cs

Namespaces

HPJ.Simulation.Utilities

```
namespace HPJ.Simulation.Utilities
```

Classes

Pool

```
[System.Serializable]
public class Pool
```

Constructors

Pool

```
public  Pool()
```

Variables

Active

```
public List<T> Active
```

Inactive

```
public List<T> Inactive
```

Methods

GetItem

```
public T GetItem()
```

ReturnItem

```
public void ReturnItem(
     T Item)
```

AddItem

```
public void AddItem(
     T Item,
     bool AddToActiveList = true)
```

ReturnAll

```
public void ReturnAll()
```

3.Extra or Temp

CameraController.cs

Namespaces

HPJ.Examples

```
namespace HPJ.Examples
```

Classes

CameraController

```
public class CameraController
```

A Example Camera Controller for the example scenes

Variables

lookSpeed

```
public float lookSpeed
```

moveSpeed

```
public float moveSpeed
```

sprintSpeedMultiplier

```
public float sprintSpeedMultiplier
```

elevationSpeed

```
public float elevationSpeed
```

LockCurser

```
public bool LockCurser
```

UseLockCommands

```
public bool UseLockCommands
```

rotationX

```
private float rotationX
```

rotationY

```
private float rotationY
```

Methods

Awake

```
private void Awake()
```

Start

```
void Start()
```

Update

```
void Update()
```

C# ObstaclePingPong.cs

Namespaces

HPJ.Examples

```
namespace HPJ.Examples
```

Classes

ObstaclePingPong

```
public class ObstaclePingPong
```

An Example Script to ping pong an object between 2 points

Variables

startPoint

```
public Transform startPoint
```

endPoint

```
public Transform endPoint
```

speed

```
public float speed
```

startingPercent

```
[Range(0, 1)]
public float startingPercent
```

ReverseDirection

```
public bool ReverseDirection
```

pingPongTime

```
private float pingPongTime
```

Methods

Start

```
private void Start()
```

OnValidate

```
private void OnValidate()
```

Update

```
void Update()
```

C# Rotate.cs

Namespaces

HPJ.Examples

```
namespace HPJ.Examples
```

Classes

Rotate

```
public class Rotate
```

An example script that rotates a object over time

Variables

RotationSpeed

```
public float RotationSpeed
```

RotationSpace

```
public Space RotationSpace
```

Rotation

```
public Vector3 Rotation
```

RotationVector

```
private Vector3 RotationVector
```

Methods

Awake

```
private void Awake()
```

Update

```
void Update()
```

SceneLoader.cs

Namespaces

HPJ.Examples

```
namespace HPJ.Examples
```

Classes

SceneLoader

```
public class SceneLoader
```

Methods

Update

```
private void Update()
```

LoadNextScene

```
public void LoadNextScene()
```

LoadPreviousScene

```
public void LoadPreviousScene()
```