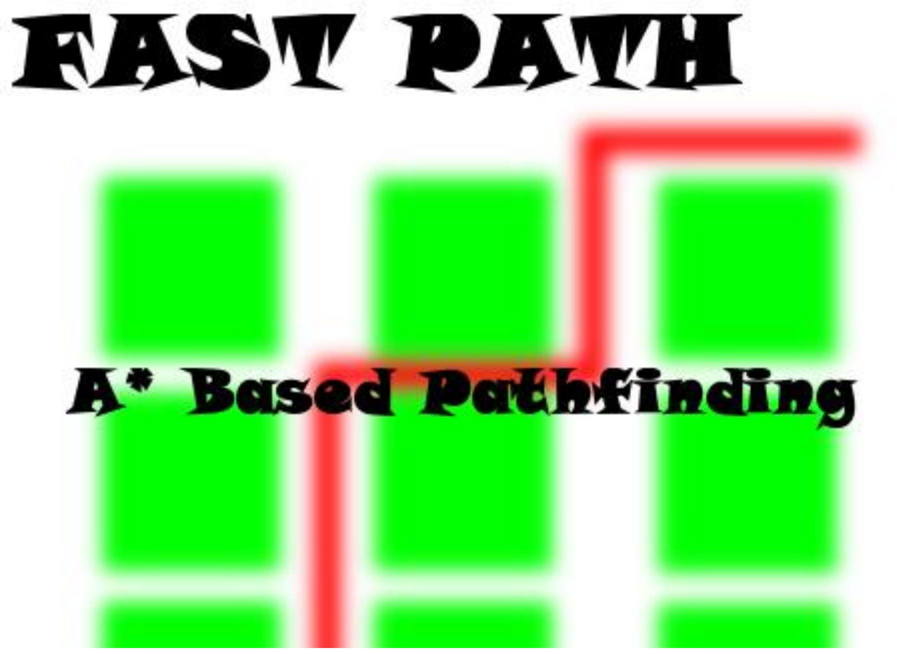

Thanks for choosing Fast Path!

Fast Path is a fast grid based A* Based pathfinding solution for 2D and 3D games. It is extremely easy to use yet very flexible.

To start using it we must first generate a ***Map***. A map is a Class which holds ***Nodes***. A node is simply a point in world which is either walkable or non-walkable and have a cost of walking above it.

To generate a map, you first need a configuration. The map will be generated on the bases on Configuration. We'll use the class ***Generator.Config*** as configuration for generating the map. Generator.Config is visible in the inspector. So it can be easily configured from the inspector window.

The members of Generator.Config are as follows:

- **Tile Size:** The gap between two nodes. The lesser this is the more detailed the map will be. But the more performance heavy it will be. If you have a simply map you can use a higher TileSize. If you have a complex one use lower TileSize.

- **Start**: The Starting Point of your map. This should be no less than the point you are finding the path in otherwise it can result in precious memory being wasted.
- **End:** The Ending Point of your map. This should also be no more than the point you are finding path in.

  <u>**Note**</u>**:** In above settings if you are using a XY/2D Grid Y axis will remain Y but in a 3D XZ Grid, Y will be treated as Z axis.

- **Walkable Tags:** The tags of GameObjects you want to be walkable. The cost determines the cost of moving above the GameObject. The more that it the more the GameObject will be avoided.
- **Wakable Layers:** The name of the layers you want to be walkable. The cost works the same as WalkableTags but, if *MatchAny* is not selected then the cost of a GameObject = Cost of Layer * Cost of GameObject's tag.
- **Non Walkable Tags:** The tags of GameObjects you don't want to be walkable.
- **Non Walkable Layers:** The layers you don't want to be walkable.
- **Single Cast:** If this is selected then only the first object the ray hits (highest Y (XY Grid) / lowest Z (XZ Grid)) will only be considered during generation. Can increase performance of Generation and Updating Map.
- **Math Any:** If this is selected then if both layer and tag of the object are Walkable/NonWalkable, then only it will be considered during generation. Otherwise it will be ignored.
- **Check Larger Area:** If this is true then a larger area (equal to the TileSize / 2) will be checked around the position of node to determine it walkable or non-walkable.
- **Ignore Depth:** If this is selected then no matter what the Object's depth is, the node's depth will be taken as DefaultDepth.
- **Min Depth:** If the depth of a GameObject is lesser than this it would be ignored during the generation.

- **Max Depth:** If the depth of a GameObject is greater than this it would be ignored during the generation.
- **Default Depth:** The depth of all points in 2D Grid or Depth of all points if IgnoreDepth is selected.
- **Use3DPhysics:** Selected this is you are using 3D Colliders in you game or you game is 3D. If using 2D colliders do not select this.
- **XY Grid:** If you are using 3D Colliders but want a want to find path on XY dimension only. Like 2.5D games then select this. If this is true then the depth is Z axis else it is Y axis. This is always true if Use3DPhysics is not selected.

After you have configured all the above settings you can build the Map by using ***FastPath.Generate(Config);*** method. And now you are ready to find paths! To find one just make use this statement: ***Path path = new Path(StartPosition, EndPosition);***. The instance of the ***Path*** class Length property and indexers so it can be used with the syntax of an array. But before using the path make sure that ***path.IsReady;*** return true. This is because paths are found on different thread so they are not immediately ready. You can also subscribe to then even which triggers when the path is found. The event is ***path.OnPathBuilt;*** You can also use ***path.ForceBuild();*** function if the path is taking too long to build. You can also use ***path.DrawPath();*** method if you want to path to be drawn in editor for debugging. If you want to build the path immediately directly then use static method ***Path.BuildImmediate(StartPosition, EndPosition);*** method.

There are many overloads to ***new Path();*** and ***Path.BuildImmediate()*** methods; Both of them have same parameters. They are:

[StartPosition, EndPosition]

[StartPosition, EndPosition, DisallowedIndexes]

[Map, Start, End, EstimateAggression, MoveDiognal, DisallowedIndexes]

[Map, Start, End, EstimateAggression, MoveDiognal, MaxDepthDifference, DepthCostMultiplier, DisallowedIndexes]

The meaning of each parameter is given below:

- **Start Position:** The position you want to find the path from.

- **End Position:** The position you want to find the path to.
- **Map:** The map you want to find paths in.
- **Move Diognal:** If you want to let the path take 8 directions rather than just 4
- **Estimate Aggression:** Usually the more this is the faster the path would be found but the more inaccurate it will be. Reducing it will result in better paths but paths but finding them will be slower. Reducing this below 1 can result in some problems.
- **Max Depth Difference:** The maximum Depth difference between two points (Nodes) than the path can take. If the difference is greater than this the path would not go from the route.
- **Depth Cost Multiplier:** The cost multiplier of moving from points with different depths. The more this is the more the path will avoid un-flat areas. 0 this if you want to disable this feature.
- **Disallowed Indexes:** The indexes of nodes on the map you don't want take in the path.

Every above setting can be found in the ***FastPath*** class with a prefix of Default with its name. The defaults are used when you call a method which do not require than parameter. The first map you generate is automatically used as the default map.

There are many more things you can do with the ***FastPath*** class.

<u>Properties:</u>

- **PathfinderBusy:** Is there any path being found currently.
- **PathfinderPriority:** The priority of the thread on which paths are found.
- **QueLength:** The number of paths that are yet to be found.

<u>Methods:</u>

- **IsMapBusy(Map);** Returns true if the specified map is being used to find paths currently.
- **Generate(Generator.Config);** Generates a map based on the Config specified and returns it.

- **IndexesBetween(Map, GameObject);** Returns all the indexes of the points the GameObject's colliders cover on the given Map.
- **IndexesBetween(Map, StartPosition, EndPosition);** Returns the indexes between the rectangle StartPosition to EndPosition.
- **IndexesBetween(Map, StartIndex, EndIndex);** Returns all the indexes between StartIndex and EndIndex
- **DrawMapInEditor(Map);** Draws the specified map in Editor. Good for visualizing the generated map.
- **DrawPathInEditor(Path);** calls path.DrawPath(); on the specified path.
- **PositionToIndexFloor(Map, Vector3);** Returns the (floor) index of specified position
- **PositionToIndexCeil(Map, Vector3);** Return the (ceil) index of specified position
- **BrindInBounds(Map, Position);** Brings the specified position in bounds of the map and returns it, if it is out of bounds.
- **BringInBounds(Map, Index);** Brings the specified index in bounds if it is out of bounds of the map and returns it.
- **BringInBounds(Map, Bounds);** Bring the specified bounds in bounds if they are out of bounds of the provided map and returns it.
- **UpdateMap(Map);** Updates the whole map.
- **UpdateMap(Map, indexes);** Updates the provided indexes of the map
- **UpdateMap(Map, Bounds);** Updates the area in the bounds.
- **UpdateMap(Map, Vector);** Converts the position to index and Updates the index.
- **UpdateMap(Map, Index);** Updates the node at the specified index in the specified map.
- **UpdateMap(Map, StartPosition, EndPosition);** Update the rectangle StartPosition, EndPosition
- **UpdateMap(Map, StartIndex, EndIndex);** Update the indexes between StartIndex and EndIndex.