Corvo PathFinding Class Documentation

This document shows how to interact with Corvo Pathfinding System to generate path and move units through them.

Class:

UnitPathfinder

This class is used to move units along generated path. After writing your own AI/Player script you can interact with this class to move your unit. You can also use this as example to write your own unit pathfinder seeker script.

Public parameters:

speed: How much fast unit will follow the found path.

rotateUnit: Choose if your unit will:
-*RotateAll*: Face the moving direction along all axis (For example if your entity is a vehicle that must also climb a mountain, then it will also rotate upward to follow the climbing angle).
-*RotateLeftRight*: Look on the destination direction only on X and Z axis (If your entity is a walking character that is climbing a mountain, it will not rotate perpendicular to the terrain, but will always stand up vertically, but rotated to the walking direction).
-*DontRotate*: Object will not rotate during the movement (Useful if you want to rotate the object by yourself).

Public Functions

goTo(Vector3 position)
Generate the path and start moving the object along it to reach the nearest walkable position to the given one.
If the position is far away from the object, or just outside the pathfinding area, while object is moving his path will regularly updated every time the unit is nearly going to the end of the found path, so it can go everywhere, even in a large open world space.

stop()
Interrupt the walking process.

Class:

CorvoPathFinding

This class is used to generate the 3D ambient where objects like Units or Vehicles can move, and the path to follow to go from the object position to a given destination, avoiding obstacles, climbing stairs, even on a big 3d environment or inside a complex building with multiple floors.

Public Parameters

NodeWidth: This parameter decide how accurate the scan will be. Keep In mind that a too high value means high probability to get a wrong final path, and a too small one means a more accurate path, but if you need to scan a bigger area, you have to increase the GridSize, and that means a heavy CPU use during the scanning process, so find the best NodeWidth for your units.

UnitRay: The width of the unit. Make it more accurate than you can to your unit size. If your unit is an humanoid, it can go through doors or holes only if they are less bigger than the UnitRay parameter.

ClimbAngle: Unit can walk upside stairs or hills only if their angle is less than ClimbAngle.

UnitStepHeight: if there are some little obstacles on the ground, unit can walk over them only if their height is less than UnitStepHeight parameter.

GridSizeX: This parameters decide how large the scanned zone should be. A too large area require more CPU time to generate the grid. Remember that you can generate the grid and scan the path multiple times during the movement of your unit to his destination, so if this destination is outside the grid scan zone is not such a big problem: You can scan a new path when your unit reach the border of the current one. If you are using UnitPathfinder script to do that, it already do it by his self.

GridSizeY: This parameter decide how depth the scanned zone should be long. A too large area require more CPU time to generate the grid.

GridLevels: This parameter will subdivide the scan in multiple layers. If your word contains multiple floors such as buildings, bridge etc. increase this value. More GridLevels scan means more CPU calculation but more precision on scanning multiple floors.

WalkMask: While generating the 3d grid, scan only the colliders with this mask parameters.

ProhibiteMask: While generating the 3d grid, if the scanned collider has one of those masks, make the area unwalkable. You can use it for example if unit shouldn't go inside rivers or lakes.

ExpansionCoefficient: If you really need to scan far away from unit for any reason, but you don't want to make the script too heavy for your CPU you can increase this value. An higher ExpansionCoefficient means a larger scan on terrain but with less precision on distance, so be carefull when using this value.

Diagonals: If this value is checked as "everywhere" your final path will be more smooth, but it will require more CPU calculation. This will not make a big difference, but if you don't really need to have an accurate result, consider to disable this.

PathFindingPos: You may prefer that the pathfinding position starts from a certain position instead from the unit position (for example a truck may start pathfinding from the front of the truck and not from the center). Assign this Transform to choose (if you want) where to start the pathfinding.

ProcessingSpeed: This value decide how much CPU will stressed during the path generation. An higher value means a faster result but a heavy CPU use. If you have an huge grid and you decide to set this parameter to VeryHigh expect some lag for half second. If you set this value to a lower one, and you have an huge grid, your game will be smooth again, but the time to calculate the grid will be higher.

TESTING FUNCTIONS
If you need to test your pathfinder in edit mode you can use those functions:

TestPathDestination: A testing transform that indicate the destination for test if you have set good parameters for your unit.

TestGridPath: Calculate the grid near the unit. You can some yellow lines: Those are the connections that your unit can walk from the given parameters. Adjust the parameters since you have the desired grid.

FindPath: If you have assigned a TestPathDestination you can use this function to have a preview of the path that the script will generate to reach it. This is useful to see if you have set up good parameters for your unit.

Clear: Clear the generated grid and path.

Public Functions.

findPath(Vector3 pos)
findPath(Transform pos)
Generate the grid and find the best path to a destination. If the destination is outside the scanning area the result path will the nearest one to the given destination. If you are working on a large open world and the final destination is outside the scanning area you just need to call this function again where your unit is going to be at the end of the current path, to generate a new path that will extends the older one. Our UnitPathfinder scripts already do that, so consider to use it for your Unit script.
Remember also that this function require time to execute, depending on the size of your grid and the choosen ProcessingCalculation value. The resulted path will available with the getPath function.

(bool) havePath()
Return TRUE if the unit has a path. Else return false.

(Vector3[]) getPath()
Return an array of each node of the found path. If script dosn't have a path, return null.

Vector3 getDestination()
Return the positon of the first node of the path. If you reached that node, you should call nextNode() to delete this node and get the next one with this function.

nextNode()
Delete the first node of the path. You can call this function every time you reach a node to delete it and go to the next one, getting the best node with getDestination().