

PathsLinker

Unreal Editor Add-On Builder

Release date target: July 20 2020

Description:

This is an Unreal Editor custom builder tool which operates in Editor, Tested and used in UnrealGold 227h. Stuff operated in simple maps might serve UnrealTournament too, indirectly it's addressing UT maps as well due to extra features from newer Unreal capable to solve problems in UT too with regard to Navigation Network which might be a pain if Editor takes control over it. It's your turn to take control with this tool.

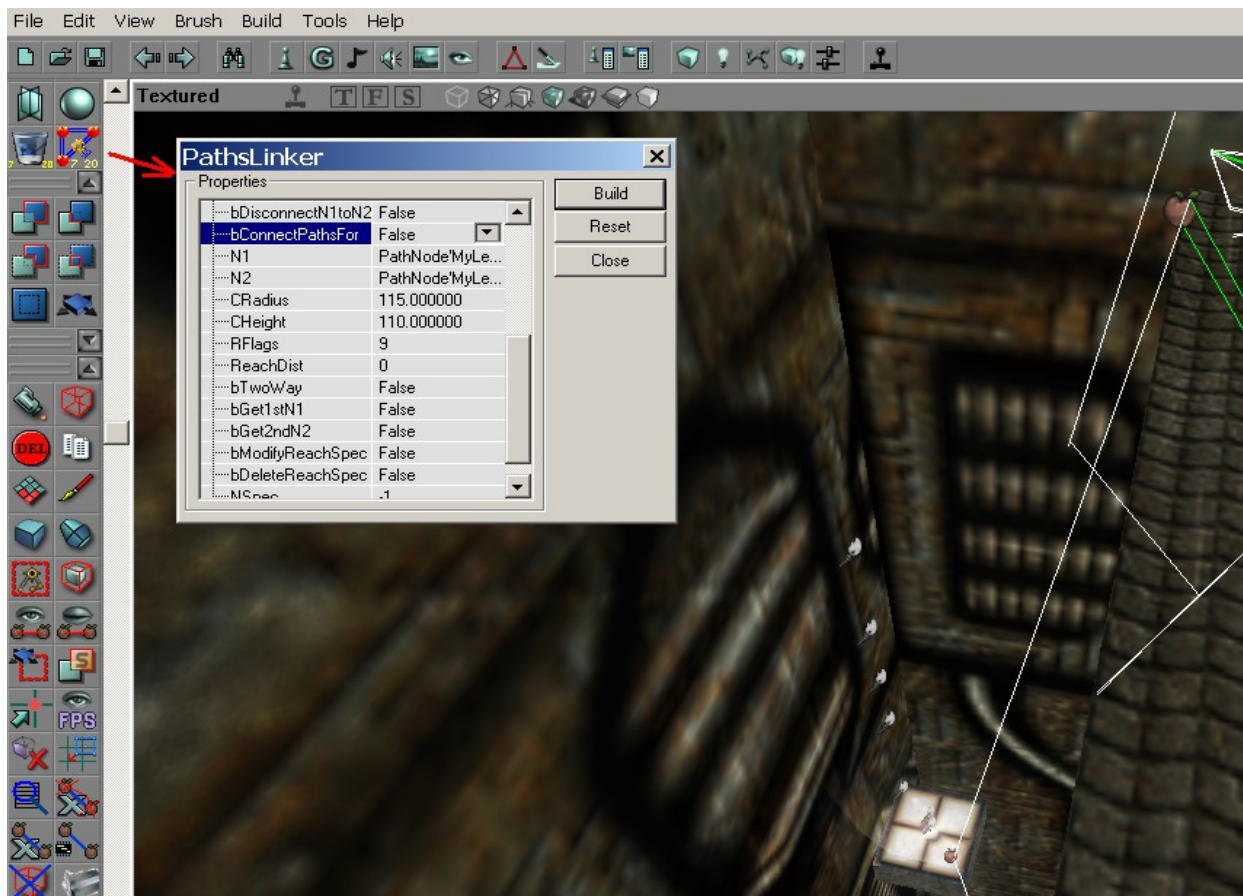
Purpose:

This was one of my Scripting tools - I did not removed those features because I might need them again some day, maybe...

Operation:

By clicking the Right Mouse button on that Added Icon from Editor (setup explained later) you can open this Builder.

We have to mark **True** options which we want to launch and then clicking on **BUILD** button shown. Once finished work or if some scroll visual problems from Editor are showing up, just close Builder and re-Open it (right click - in default OS's mouse setup) in case that you still need it. GUI-s from builders in Unreal Editors (any) aren't fully updated all time. You have to click here and there selecting/deselecting fields for getting On-Screen update, else values are updated even if User Interface doesn't show this instantly.



Explained bool values:

1. **bLogNewNav** - It logs a script for causing spawning a selected NavigationPoint - used in temporary maps for getting locations, I used this for generating NavAdder plugins by "scripting" instantly hundreds of

Navigation Points.

2. **bLogTweakPoint** - here is generated an XC code for tweaking whatever Actor - also I used it in scripting plugins.
3. **bGenerateDecoUC** - whatever static decoration class might cause generation of a custom decoration which I used to spawn in clients, script is logged too.
4. **bLogActors** - logging map actors, this is first prototype which now is also in MapGarbage builder by self-person.
5. **bComputeDistance** - if we have two actors selected we can get distance, such script generates distance for a reachSpec. Not accurate but... pretty usable.
6. **bCheckDuplicates** - this is primary prototype for checking duplicated actors.
7. **bFixDuplicates** - combined with previous boolean value might fix or not duplicates from a map, I don't know if Unreal needs this but in UT I needed such things.
8. **bLogBrushBuildLag** - some maps are having geometry issues, usually when such a map is being build, at a brush Editor takes a break, I'm not sure if is not computing a BSP cut there, if we record that brush lagging build, we can input number **NumBrush** and we can get around said lagger brush which we can examine if it's aligned or has various issues around it. Not always helpful but... I was trying everything when I worked at screwed maps.
9. **bTestReach** - having two Navigation points selected we can see if area is navigable in primary format, by walking. This might discover even BSP problems causing an UnReachable path. Updated version is doing this test and it will capture CRadius and CHeight from Scout used for probing spot, this being a small recommendation regarding to cylinder collision used for roaming through this spot. If you need a reachSpec here, collision values suggested can be used for future reachSpec. I used this basic check when I worked at patch plugins for adding navigation points in maps - XC_Engine here.
10. **bCountReachSpecs** - this is my primary counter for reachspecs embedded in NavigationPointlist.

These were "options". Now we have newer things explained:

11. **bDisconnectN1toN2** - This feature can disconnect a path going from a defined name for a Navigation Point **N1** which heads to another Navigation Point **N2**. **N1** and **N2** can be written manually in builder box using names or automatically captured only one by one using the two other booleans:

- 11.a **bGet1stN1** - selected NavigationPoint will be dropped as **N1** variable;
- 11.b **bGet2ndN2** - selected NavigationPoint will be dropped as **N2** variable.

After completion of N1 N2 fields - if we want to see them we must refresh GUI by inspecting them with some click, then pressing build with said bool marked True. Path from N1 to N2 will be DELETED and reachSpecs re-counted - keep this in mind.

12. **bConnectPathsFor** - it does a connection between two points as **N1** and **N2** - more or less manually set or captured as previously described. Here is generated a reachSpec or two as follows:

12.a **RFlags** - Means desired reachFlags for reachspec which will be created: 1 - Walk, 2 - Fly, 4 - Swim, 8 - Jump. A jumpy route needs walk and jump, which means $1 + 8 = 9$. **RFlags** value must be 9 in such a case. Lifts and Teleporters are using 32. If you want other flags I do not have any fault for what you do, it's your problem.

12.b **CRadius** - maximum CollisionRadius used by Pawn user of this path aka reachSpec.

12.c **CHeight** - maximum CollisionHeight used by Pawn user of this path aka reachSpec.

Note: We might want to not set these lower than 20 radius and 40 height for Bots and us, testers. I think zero means that everyone will get through this path - check it.

12.d **bTwoWay** - means that another reachSpec is created on the way back from **N2** to **N1** as well, not only from **N1** to **N2**. Unreal 227 shows two lines or one accordingly. Distance between points specified in reachSpec is computed automatically;

12.e **ReachDist** - This is *not a must be completed* field, distance is computed automatically, but you can use your own value if you have reasons for that. It won't affect WarpZoneMakers, Teleporters and Lift Combos which are using original values, 100 and 500.

13. **bModifyReachSpec** - if we did something wrong at a previous reachspec, we can modify said reachSpec if we know index (logged previously), which is set as **NSpec** variable. Modifications available are taken from above values reachFlags/Collision data set in boxes. Only a reachSpec is modified at time so the other one **bTwoWay** has no effect here.

14. **bDeleteReachSpec** - here we can remove a reachSpec from map or a wrong one which is proving later as being disturbing for pawn's movement.

Update 07 2020

Adjusting/Deleting Paths in this version will take in account plain Paths Networks having **PrunedPaths** data. Previous version was working perfectly but not in maps having paths executed with plain original DevPath. When a reachSpec is removed and the remaining ones are recounted we need to remap PrunedPaths too if they exist. XC_EditorAdds from UT addon XC_EngineV24 doesn't do these useless invisible junks but which are affecting pawn's navigation - it's why previously I did not take them in account.

NOTES:

1 - Because these stunts are a bit... entertaining, my advice is to have one type of operation per Editing session. Which means that we are only doing ReachSpecs in a session and we are removing ones in another session. I won't try to explain in a few words what's happening at C++ Level, but this is how I do things in order to not do damage or crash Editor with work UnSaved. Garbage Objects are purged at quitting Editor, stacking objects over future garbage ones might not be helpful - I believe otherwise here might be issues. Some Evil reachSpec in whatever map can be removed by only knowing index number. Usually I take the number from MapGarbage. After removing a reachSpec I will want to regain data for other move because each reachSpec deleted will recount remaining ones. Here Builder should recount them as well in references from All Navigation points actors from map. Starting with July 2020 release all reachspecs are remapped - PrunedPaths included, if exist.

2 - We can attach a new PathNode without to screw a reworked Path-Net. This requires adding new PathNode in NavigationPointlist - using MapGarbage. We disconnect all Array and then we reconnect it. PathNodes placed in good areas are linked in newer created Array. After that we can add reachspecs - paths lines to/from new PathNodes.

3 - We can change or add reachSpecs with any ReachFlags we want in some already done NavigationNetwork if it's bad or not connected, here we can create aerial paths, swim paths and even combined when zones are mixed and pawns should use those paths. This means that we can build a Path-Net as we want discarding those angled buggers or too longer jumps in seconds.

4. - If map has PrunedPaths (majority have these) - usually I'm using XC_EditorAdds from XC_Engine done in UT because this is not creating such useless invisible paths - perhaps some problems might be expected. Keep a backup of original map. After removing a path, it is advisable to check in a future session two navigation points connected if are referenced correctly by using MapGarbage and "bShowSpecs" option for a selected Node and read connections. If data is fake, then network has been damaged due to some lousy

thing from map or some other reason. Usually a simple path-net created with XC_EditorAdds from XC_Engine in UT is easier to adjust later in Unreal for getting a simple and a good Paths Network without heavy to follow paths.

Setup:

U File goes in **System** folder. BMP file is icon for button going in **editorres** folder. In file **Unreal.ini** we need to find **EditPackages** variables. There we can complete as last position or whatever:

EditPackages=PathsLinker

Here is the setup. Uninstalling means reversing install steps. Builder uses new functions from Unreal which UT'99 doesn't have, so don't try it in UT because it won't work. If map has common assets and can be opened in Unreal, here is editing environment.