XC_PathsWorker

UnrealTournament Editor Add-On Builder

powered by XC_Engine v24+

version September 2022

Description:

This is an UnrealTournament Editor custom builder tool which operates in Editor.

Audience: Advanced mappers understanding navigation and certain settings concerning a good state of a map. Mandatory requirement: XC_Engine v24 and above (if future versions are not going to get into ruins). XC_Engine add-on can be used successfully only in Editor if it does funky reactions in run-time.

Purpose:

Handling and creating very good or desired navigation paths in UT's Editor at a time creates headaches. This builder-plugin attempts to come up using some newer features that are implemented in the XC_Engine extension created for UT helping with pathing stuff.

Majority of issues are caused by over-crowding navigation nodes (any type) in a small room/spot causing Editor to create a lot of ReachSpecs for a single node and then, their number it's no longer matching internal Paths array going over boundaries - 23 ReachSpecs cannot go into a list aiming 16 ReachSpecs, and leaving unused paths un-referenced, out of navigation network. Builder is intended for reducing these debris junks ReachSpecs, and has options for adjusting paths charge, it will leave some free space for other needed paths which user might want to create - returning a simplified network and a faster processing task inside Engine.

We have the opportunity to edit such a path (REACHSPEC). We have the opportunity to create missing paths. We have the opportunity to see all the data related to a path and make corrections that automatic script in the Editor refuses to help us - Editor can not read our mind in the end, and it doesn't allow user to control paths as it does with geometry.

Architecture and Information

The plugin contains boolean variables (Right Click on Builder's Icon) that can take **True** or **False** values. The TRUE marked option followed by the Build button activates the execution of a function. Other variables can be TRUE but are connected with other main features, they do not do anything alone (bOneWay, bTryMonsterMode...) all are going to be explained later.

Installation:

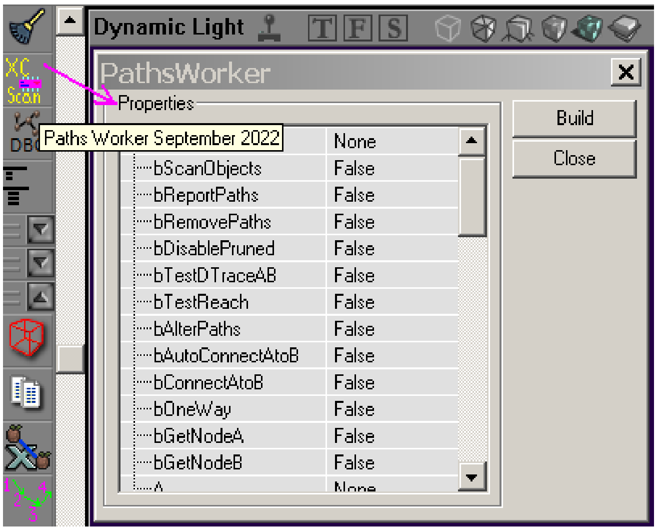
U files are going in **System** folder or whatever UT Path for U files - builder has a helper file (might be good for future needs). BMP file is Button aspect from Editor and this one will go in **editorres** folder from System folder. File **UnrealTournament.ini** will need to be completed for activating this builder as follows (see **colored** lines):

```
[Editor.EditorEngine]
...
EditPackages=ExtendedBuilders
EditPackages=RahnemBrushBuilders
EditPackages=TarquinBrushBuilders
EditPackages=TarquinExtrudeBuilder
EditPackages=DavesBrushBuilders
EditPackages=EditorTools
EditPackages=XC_Core
EditPackages=XC_Engine
```

```
EditPackages=XC_EditorAdds
EditPackages=XC_PathsWorker
...
```

As you might notice (or maybe not) XC_Engine is mandatory for this builder because it uses functions from this Engine Extension. Builder it's crashing your amazing environment without XC_Engine. Builder is tested with XC_Engine v24 (compiled here) and v25b/UT469b (sloower !). I understand that players are not always loving XC_Engine for playing game but it can be used only in Editor if in run-time does ugly stuff in game-play. It has several features that are helpful for mapping.

We open Builder's interface usually with the Right-Click on Builder's Hinted Icon. If everything was installed correctly we should see this Window in UT's Editor having all XC_Engine related packages set in "EditPackages" array from **UnrealTournament.ini** file (XC_Core, XC_Engine, XC_EditorAdds):



	None
bScanObjects	False
bReportPaths	False
bRemovePaths	False
bDisablePruned	False
bTestDTraceAB	False
bTestReach	False
bAlterPaths	False
bAutoConnectAtoB	False
bConnectAtoB	False
bOneWay	False
bGetNodeA	False
bGetNodeB	False
A	None

Explained vars:

Variable	***Connected data***	***Description***
TheMap	-	Auto-Completed by builder. Recommend manual completion only for failures.
bScanObjects	-	Listing all sort of objects visible with XC_Engine extension.
bReportPaths	-	It will log all Paths (ReachSpecs) hosted by map - even unused ones having a complete report of them. See **Extra** paragraph.
bTestReach	-	This option will check two selected Navigation Points for movement capabilities - it's a "PointReachable" test reported. Collision data for a navigable spot is captured/stored in a temporary reachSpec if we have as plan creating a path between these two tested PathNodes and we need to have a clue what collision would be suitable.
bAlterPaths	-	Here builder is brutalizing a bit the competition. The original navigation paths that have a maximum size 70×70 are modified at 117×112, here Titans can also move using the navigation network not only smaller creatures (Must have place !). Also the normal ways through elevators are adjusted for less intelligent creatures altering the special path definition - there is nothing special at a lift in UT, it's just another way - more data it's in code. After using this option for a big war, perhaps your lifts must be set **BT_PawnBump** or else creatures won't handle them.
bConnectAtoB	- bOneWay - connect one way A to B - A - node name set as Start; - B - node name set as End; - **TempReachSpec** is template for a future ReachSpec/Path	This is what we call Creating a Path manually. We need here Point A and Point B as START and END then other required data for this path, ReachFlags and Collision data. Distance is calculated automatically - if it's not defined. If the related variable **bOneWay** is **False**, the path created here will have the other return Path too. So we do have a path from A to B, and from B to A. Either way it will be one single Path from A to B (jumping down from a house). This ReachSpec/Path is created based on

		temporary template reachSpec variable called **TempReachSpec**. Reachspec created is a mirror of this temporary spec variable. Empty paths (no data here) are not accepted - logically they are USELESS all the way.
bModifySpec	- SpecIndex - ReachSpec delegated to be changed.	This option will modify a Path/reachSpec according to **TempReachSpec** data for Index of Path **SpecIndex**. If **SpecIndex** value is -1 nothing is changed. **ThePower:** If a Path it's an evil one causing a bad loop around a stair or a ledge, this path can be moved elsewhere in map where a Bot can move but Editor (or even this builder...) has "forgot" to add a path which looks definitely navigable. This will screw paths indexes referenced inside nodes involved but... the recovery is described later below.
bGetNodeA	-	One selected Node is copied and set as Variable A and START for a future reachspec creating or editing aka **A**.
bGetNodeB	-	Another selected Node is copied and set as Variable B and END for a future reachspec creating or editing aka **B**.
bRemoveSpec	- SpecIndex - ReachSpec delegated to be nulled - not deleted.	This option doesn't do actually a removal of a ReachSpec because XC_Engine doesn't do this - exactly. Said **SpecIndex** as number of the ReachSpec from array will go nullified and moved on top of ReachSpecs array. As result, all remaining reachspecs are renumbered. Apparently it looks like a break-down of entire network starting with "eliminated" ReachSpec. It is true but things are restored due to another option added for defragmenting and rebuilding Internal Lists with ReachSpecs based on their referenced Nodes. After altering a ReachSpec it's recommended defragmenting and listing them again because next ReachSpec candidate for removal it's no longer having the same Index number as it was after remapping ReachSpecs for matching their navigation nodes. "Removed" ReachSpec as debris data will go at end of array and all remaining ReachSpecs are recounted - their Index is changed with one point back. Example: Destroying ReachSpec 50 will have ReachSpec 51 remapped as 50, 52 remapped as 51 and so on. It's why defragmenting-remapping them needs to be used after each of such "removal". This task requires attention at details which usually it's not what mappers are getting used to have and understanding of these assets.
bCountReachSpecs	-	This is a complete Paths/ReachSpecs counter - including junks left in map and which are not having their Index referenced in Navigation Network.
bCreatePaths	- **bTryMonsterMode** - generates paths for all sort of creatures not only for Bots but Bot-Compatible as well; - **bAvoidStarts** - PlayerStart actors won't have mapped paths but they go in Navigation Chain; - **bKeepOldPaths** - old Navigation chain and reachSpecs are not removed and so neither old InventorySpots; - **ScanRange** - default UT has 1000 which can be exagerated in more maps and then it can be configurable; - **PLimit** - max number allowed in internal Paths Lists Array (0-15) from a node. Builder stops at 10 creating reserved space for extra connections; - **BunchSize** - The size of Spot with multiple paths or inventories considered overcrowded and delegated to be simplified;	Builder is creating here a navigation network using current PathNodes and combos from map - except those incomplete, hidden from Editor, warp-zones and bOneWayPath marked nodes. One-Way path will be user's job because here are expected unwanted routes which user doesn't need. It uses connected variables for creating usable paths without loading map with never used ReachSpecs out of navigation references. Paths in Monster Mode are enlarged but it will check Bot compatibility as well or else such paths won't be created. September 2021 update - Avoiding Starts, keeping old paths and not linking points bHiddenEd might unlock some creative options which original DevPath won't do.
TempReachSpec	- **bPruned** (0 or 1) - if path has to be a shortcut (invisible in original UT up to v469b so far); - **CollisionHeight** - maximum collision height for creature delegated to roam around; - **CollisionRadius** - maximum collision radius for creature delegated to roam around; - ***Distance*** - used if defined or else is automated;	This is the mirror of a future reachSpec/Path which will be manually created or modified. I'm not messing with Pruned Paths unless I turn them into normal needed paths - visible in Editor.

	- **End** - point name Known as B captured by user or written manually; - **Start** - point name known as A captured by user or written manually; - **reachFlags** - it's moving flags as an INT number according to physics required for taking this route. R_Walk = 1, etc. and making the sum of these for final value.	
bFixSpecsLocations	- bAndNullExtraSpecs If used, ReachSpecs that are not inside Navigation Network are no longer added back in their place, but are nullified. See **July 2022 Update Notes**.	This is important to do after editing several Paths/ReachSpecs. When a Path going in evil formula from PathNode0 to PathNode1 was moved between PathNode2 and PathNode3, this Index is still in old places and then it has to be remapped properly in newer nodes and deleted from old nodes. This option is checking ALL reachSpecs and it will put them in their place if are edited or screwed. ***August 2021 update*** **–** in extremely loaded maps (kinda desperate pathing or insanity), certain paths are not completed in internal Paths[0-15] array and also in upStreamPaths[0-15] array. I'm not going to study engine internals but I think there are generated a lot of paths for each reachable node available from a source node. When these are going over 16 Paths due to the OVER charge for no purpose - this sort of pathing makes life harder with BlockedPath which won't block anything being pruned easily - some of these ReachSpecs will remain in the wild. The code also will "forget" other paths, perhaps the bug is connected with previous issue. This builder doesn't do such bugs. In this fixing stage builder is capturing these lost ReachSpecs and will add them into Navigation Network for being used. Extra ReachSpecs are not added because... there is no more room for them in arrays. They will remain as debris data unused. The task result it's reported. Usually such maps might go in random flaws and then I'd rather prefer to simplify navigation network as much as possible but covering map properly - exactly this is what Polge was explaining but mappers are not reading. Of course, Polge forgot more explanations but... the story it's not for current document.
bDefragReachSpecs	-	This operation will drop out all ReachSpecs from Navigation Network and will add them back depending on their Start End and if are pruned or not. If we have manually removed a ReachSpec Index or we have lost paths, this is an alternate option for remapping them as needed. Extra junk ReachSpecs (no more places for them in arrays) are discarded at once with those "deleted" ones (they will have None and Zero data). All Lost ReachSpecs are result behind Editor's task in overcrowded maps. Either way, NO, Editor doesn't do debris data if map is pathed as Polge was recommending. Technically if map is covered correctly with multiple routes through multiple tunnels instead of multiple routes in the same tunnel these issues will cease to exist. Perhaps future UT updates will have another way of calculations without generating too many ReachSpecs for the same PathNode.
bCheckNodesZoning	-	It does a check around Navigationpoint type actors because they need to match exactly a zone where are Placed: In Water or not In Water. This out of sync happens when nodes are moved manually and properties are not updated for a Node in Editor. A node previously in air and not doing paths if it's pushed in water it will be like an air node because IT IS NOT updated. If certain nodes are not having paths, it is advisable to delete paths then do this check. Nodes are fixed if possible. Native function SetLocation will update properties for target zone where actor is being moved. Technically this is not a real problem of map for run-time environment where Engine while initializes the game it does the zone settings for actors. In Editor if zones are not matching, those paths will not be created by this Builder and you will need more manual working. In order to have paths for Nodes concerning water, they have to be placed/centered UNDER WATER surface and not above and keeping only those two leafs outside of water surface. A bad placement might result in bad paths or impossible paths and this is your fault.

Update September 2021

- Adjusted code for InventorySpot actors because on certain bridges they were added UNDER bridge and not over Inventory due to "Spawn" function from UT used for creating these markers recommended by Scout mapper, which had some quirks - it looks that are required multiple checks for UScript pathing... - still under tracking;
- Implemented a dual check for PointReachable, preventing Monster Routes very heavy or

impossible for Bot to be created;
- Connections to/from a Node having bPlayerOnly set to True will have Paths with reachFlag 64 aka R_PLAYERONLY.
- Implemented **bKeepOldPaths** which won't remove a previous navigation network and data before adding new paths.
- **bAvoidStarts** - will discard creating paths to and from PlayerStart actors - but map needs to have these.

Nodes bHiddenEd are excepted from being pathed, useful if nodes are connected doing damage in spot, letting user to manage what would be connected and where.

Small corrections in operating paths creating task.

Creating a Path is based on a double check. Bot Size is telling the story and not a big one which has more options when nodes are too high from the ground. If Bot is agree, builder will check the rest of options.

bAutoConnectAtoB	- requires two selected Navigation Points supposed to have a path if testing movement returns a good result.	Builder is helping here if for some calculus reason two navigation points might have a good path which was not added during paths definition process. Path parameters are tested automatically and if probing Scout returns positive results at movement tests in Editor path/paths is/are created without anything else needed. Technically it will attempt a Two-Way path if possible, if not path it's not mapped or it's mapped One-Way. This is actually a quick method for completing valid needs directly reachable not through walls. July 2022 - Added support for Teleporters and Combos which is done automatically without probing spot.
bTestDTraceAB	- requires two Navigation Points selected.	The two selected Navigation Points are tested if are candidates to a future connection with a direct sight line between them - center to center. Nodes closer without this trace response definitely won't have paths. If you need to avoid a path by placing node behind a thin column, you may want to ensure that node won't get linked and tracing result must be negative. Option was added after figuring that **PointReachable** might do messed up paths based on flawed calculations from Engine around some grates, stairs whatever. Preventing such a path requires this test for said two ugly nodes in order to no longer have a direct line each-other.
bShowNodeData	- requires a selected Navigation Node.	This option will report reachSpecs content from paths lists which a navigation node is having referenced. Here you have FULL data of reachspecs so you can figure if here are large paths or narrow paths, then you can do some fine tuning if it's needed.

Update October 2021

- Changed code toward Teleporters which are Source-Only;
- Added variables **bGetNodeC bGetNodeD C D** for being used in newer **bAutoMoveCDtoAB;**
- A few changes in creating paths.

bGetNodeC	-	A selected PathNode gets captured as variable C which will be source of an old current path delegated to be moved elsewhere between A B.
bGetNodeD	-	A selected PathNode gets captured as variable D which will be destination of an old current path delegated to be moved elsewhere between A B.
bAutoMoveCDtoAB	- C D A B - required nodes which will have path modified from C to D going between A B.	This option can actually modify a ReachSpec without to be necessary hunting index and making more moves. How does it work ? ReachSpec having as Start node C and as End node D. Goes edited with Start as Node A and End as Node B. In practice if a Path is moved we need to examine the other reversal ReachSpec too (from D to C) which will need to be recaptured as in first case as Start-C to End-D, path goes between A and B - also redefined for being way back. When ReachSpec is moved it will be de-referenced from source/destination nodes and relocated in newer places. Distance of path is calculated automatically but the rest of data will remain untouched. This ReachSpec if it needs to be adjusted later we have index copied in builder just in case that ReachSpec needs to be corrected (collision and/or reachflags). This is helping if certain path need to be re-oriented between other two points for preventing some bumping in a wall or a heavy move which fails often, or simply routing Pawn between items in other way.

Update November 2021

Added options for Warp-Zones, this requires definition of a WarpZoneMarker, a customized one rewritten might cause pawns to get a better behavior while are roaming through these Nodes.

bGenerateWarps	- **MarkerUsed** - this is mandatory to be declared, using bugger from stock or a	Around a WarpZoneInfo actor (or an external replacement) it's added a navigation actor specific to Warp-Zones in order to create future paths through these areas. Here is

	custom one.	recommended putting them properly on the navigable ground as long as aerial placement or placement on a wall from Warp-Zone doesn't have anything improved or having a valid logic.
bConnectWarps	-	After launching option for creating paths and having navigation chain connected using this builder, this option will create paths through WarpZoneMarker type actors if Warp-Zones are having connection strings declared correctly. Plain reachFlag for these reachSpecs is 32 like in stock strategy and similar to combos Le-Lc-Le and Teleporters. Required Steps: #1 generating warps markers #2 generating paths-net #3 connecting warps post-pathing

Update December 2021

Added a calculation concerning timers used for building paths - that was my own will. It will report the average of ReachSpecs per Second built.
A small update-code glitch, not a pathing code - I'm not a fan of ”known bugs”.
Added a separator variable where Pathing variables needs to be declared, in order to gain a small improvement at visual ergonomics.

To Do ?
- auto-connecting a new added and selected PathNode and linking it into NavigationChain ?
- declaring paths charge and a Min/Max range for this new one for being under extra-control ?

Update January 2022

Pathing a Combo delegated for jumping will have a small check - boys are not understanding that Combos are **TWO-WAY** paths causing sometimes an impossible return after jump. Builder attempts to prevent a connection from a JumpSpot to a higher LiftExit because Pawn doesn't jump to a LiftExit, this Actor is not for jumping and is not having any directive for A.I. concerning a jump back or... an **Impossible** jump back.

Update May 2022

Report during fixing ReachSpecs is reduced. It will show only errors, ReachSpecs are already listed using the other option described upper in the table.
Several codes have been re-written for reducing BotPack dependencies - at this moment builder won't trigger Editor to load anything from BotPack.
Code solution for pathing map has been changed for decreasing iterations at C++ Level - using ”Do-Until” instead of ”While”, ”For”.
Pathing report after creating ReachSpecs will include how big is Navigation Chain - previous report is covered and not really visible in Log Window.

Update July 2022

Expanded option **bFixSpecsLocations** with a sub-option **bAndNullExtraSpecs**.
Added option **bReIndexSpecs**.

bReIndexSpecs	-	This option will wrap ReachSpecs with reindexing them. If map has Valid ReachSpecs and Null ones, all valid are located first in array and those unused are going in last positions - See July 2022 Update Notes.

Update August 2022

bRemovePaths	-	All it does this option is getting rid of paths from map using Paths Undefine command but as first task it will null out data from ReachSpecs before calling Engine to remove them. Purpose was to no longer have these ReachSpecs objects linked at Actors from map, Reachspecs being linked to NavigationPoints from map with Start End.
bSeePrunedPaths	-	Editing PrunedPaths for being drawn. They are copied elsewhere into an external actor for mirroring these reachspecs - all of them not only those referenced in Navigation Chain due to latest UT patches.
bNoSeePrunedPaths	-	Editing PrunedPaths back into original format, not drawn in Editor with data copied from their mirror. Navigation Chain won't have any touch concerning these tasks.

Update September 2022

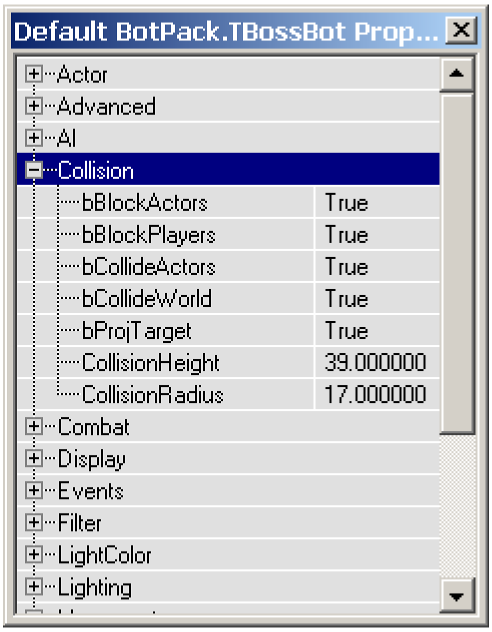
Version for September has not only a bit of revision at codes but it has some news which I needed - probably others will need these too...
Certain ”animated” Task has a performance deal, because I worked in slower and a bit faster machines, if repainting screen goes like a doggy slow, screen update period is decreased for preventing a lot of time wasted here. I'm talking about constructing paths process, we allow more time to Engine for computing paths instead of updating ViewPort.

bDisablePruned	-	Feature added in prior update was pointing me a lot of invisible misery happily operated by ”Goblin” types of Paths Constructors. Since the builder demonstrates already that Paths are normally usable without those things, we can get rid of them by nulling their content. They will

		remain in map (or not - if we want them flushed). Note: ALL PRUNEDPATHS are nulled, even those that are Left Out of network by buggers developed in newer UT patches.
bFindFreeSpec	-	After nulling some useless ReachSpecs with prior described features, if we need some nodes connected we can recycle several debris ReachSpecs instead of creating new ones. But we need to know an Index (number of a free/null ReachSpec) available for usage. The rest can be flushed out with next option...
bCleanNullSpecs	-	Purpose here is to completely remove debris data - null ReachSpecs if they are no longer needed. The process is dodgy, volatile nodes like InventorySpot and WarpZoneMarker types are gone, but they will be recovered after Cleaning task. Here is/are supported new types of these but... if they have new properties it will be needed adjusting them again. I don't know how many mappers were using custom types of these assets since this task is not very easy to do. This builder allows existence of custom types of InventorySpots and Warps - probably no map will be like that. Logic -> Whoever can handle these, definitely has skills for working clean. See **Flush Note** for Technical details.

Extra: A reachSpec known in UT Space as a Navigation Path is a sort of structure containing some data that is being processed during C++ navigation codes and it is drawn as a line between two navigation points, except Pruned Paths (shortcuts):

#1 Actor **START** - is usually Navigation Point where a path starts;
#2 Actor **END** - is usually Navigation Point where a path ends;
#3 UnSeen by original DescribeSpec function - **CollisionRadius** - Maximum Radius of creature accepted to take this path, Bot = 17.000000;
#4 UnSeen by original - // - **CollisionHeight** - Maximum Height of creature accepted to take this path, Bot = 39.000000;



bBlockActors True
bBlockPlayers True
bCollideActors True
bCollideWorld True
bProjTarget True
CollisionHeight 39.000000
CollisionRadius 17.000000

#5 **ReachFlags** - one or more movement capabilities in order to pass through this path (R_Walk, R_Jump, etc) - Pawn must be able to walk, jump, swim, etc.;
#6 **Distance** - How long is the path in UnrealUnits aka UU. The shortest routes are automated by Engine while a road to a target is computed;
#7 **bPruned** - values here are 0 or 1 which means if this is a pruned path (shortcut over other paths) or is a normal path. Such ShortCuts are not shown in plain Editor and neither in XC_Engine v24 but they do exist.

Red Path in original Editor means a Narrow Path **not a Heavy Path** - that's a dumb myth spread for years - a cheap and not payed stupid show;

Blue Path in original Editor means a large(/Enlarged) path where a Mercenary or a Krall can track Patrols or whatever. However, a Blue Path having a bit too "narrow" (small collision accepted) directive might be discarded for a Skaarj Berserker because this Boy it's a Big One. This is the moment when Builder does a complete report of paths and then you can figure if your path is compatible or not with Big Boys - if space is permitting, this

Path can be Edited/Enlarged.

Cave-At-s:

In U227 known as a recent UE1 update for Unreal, we do have options for deleting ReachSpecs/Paths - completely removal of these. XC_Engine is capable to "deactivate" paths but leaving them in Map - not really deleting them. As result, I did not implemented "deletion" of a Path because that's not exactly a Wiped Path, it's debris data which I dislike - but, if there is no way to get rid of bugger you can proceed doing it a null ReachSpec. Perhaps a wide range of maps can be pathed different using combos wisely placed instead of generating bugs.

Due to UScript and Editor Environment used, builder can fail in adding certain Paths. Here it's time for a small manual work and check, this is doable as long as the builder has this capability. In whatever big map you might see missing paths in water or whatever case. You can do them later anytime exactly as you want them.

WarpZones - because WarpZoneMarker has a critical bug in plain UT and here a Human Player can crash game during navigation/testing, I did not added anything here as a Scripted Pathing task. A piece of crap it's not my way of doing. You can add manually paths using combos or another forced links as you like. It also would be a problem in determining the Place for WarpZoneMarker in relation with WarpZoneInfo actor, some maps are having these placed in hilarious locations and we do need to find the valid stepable floor in UScript without to deploy to many processing cycles - this is way no go for me. Such a WarpZoneMarker can be added and connected manually using tools like MapGarbage and this builder. Yeah, WarpZoneMarker is defaulted with **bHiddenEd True** for making pathetic life more harder a la EPIC, but we can deal with it anyway because we can change properties without saving any stock packages and having them ready to use in current editing session.

Flush Note:

bCleanNullSpecs - option is coming like a sort of Trivia. XC_Engine the main Reactor of this builder has nothing for deleting a ReachSpec - at least not for now. However, builder can purge these ones using a method which might not be suitable for every single map. Let's see the methods and operating mode.

For each InventorySpot and WarpZoneMarker it's added a new NavigationPoint. Routes to/from these old points are reconfigured in new Navigation actor which also is informed if it was about an Inventory or a Warp. In next stage Reachspecs are saved in temporary actors added - including these modified ReachSpecs. After doing this "backup" and flushing navigation nodes from any data (stock maps are full of trash here), Engine is triggered to remove paths - exactly, Engine will flush them, not XC_Engine - LOL. After removal of paths (all is logged), builder will reconstruct lost points InventorySpots and WarpZoneMarkers linking them as in original but VISIBLE this time. Now ReachSpecs are restored from "backup" - only valid ones were saved and only valid ones are reconstructed. At end of task Navigation chain is being rebuild using InventorySpots as first nodes in chain and PlayerStarts as lasts - this is probably the order of navigation rules. Route is searched from Target to Seeker and linking nodes in reverse, with a minimum number of cycles - can be very helpful in certain maps.

Let's see a **Flush sample**. Taking stock **DM-Deck16][** map.

We will use options as follows:

#1 - using **bDisablePruned** - nulling all PrunedPaths;

#2 - using **bFixSpecsLocations** with sub-option **bAndNullExtraSpecs** - nulling Reachspecs that are not in nodes lists;

#3 - optional **bReIndexSpecs** - will wrap null ReachSpecs to last position in their array - here they will be updated in nodes with corresponding Index;

#4 - using **bCleanNullSpecs** - here is the flushing task.

This way map will no longer have 1997 ReachSpecs, only 919, those used ones at once with their bugged nodes and the same no access at several items but... more clean. First hit in flushing map will cause existence of InventorySpot0 - these will get wrapped. In other cases we will have other Index for these names but... a clean ground. Cleaning won't solve bad placed Nodes, it will only drop out debris data.

Custom Pathing Hints

It's important to understand reachFlags for preventing Pawn from being discarded or bugged with an impossible path.

We do have Movement Flags aka ReachFlags as follows:

1 - R_Walk -> Pawn requires walking capabilities (Tentacles, Devilfishies, are not compatible);

2 - R_Fly -> Engine recognizes this flag in run-time but original Editor won't do these unless you are using custom pathing (and XC_Engine) - yes, it's time to fly, boys;

4 - R_Swim -> This is for water and it is advisable to be ONLY flag 4 in Water as long as Tentacles can ONLY SWIM and nothing else than hanging. Believe me or not, Tentacles are swimming;

8 - R_Jump -> Usually this is needed at ledges and various small knee obstructions but it has to be combined with R_Walk resulting 8 and 1 = 9. If we want to go **in** water or go **out** of water I recommend there flags: R_Swim R_Jump R_Walk which means 1 and 4 and 8 = 13; The logic is: Pawn will swim to ledge, it jumps out of water and walks at next node. Reversal - it walks at ledge, jumps in water and it swims at next node;

16 - R_Door -> Another never mapped flag concerning doors and creatures capable or not to handle a door, mindless creatures can be discarded here in a logic way if we are adjusting such a path correctly;

32 - R_Special -> This is a common Flag used at Lifts for intelligent creatures or another ”smart” paths which a Bot will follow (Teleporters, WarpZones, Jumpy Combos in original Devs - look at operational STOCK maps);

64 - R_PlayerOnly -> Not often used but... we have it, perhaps the name is self-explanatory. If not, don't use it.

If we have PathNode10 and PathNode100 delegated to be connected, we might want to see which data can be suitable for path's collision. A test is recommended. If the test fails you need to go in game for figuring if this connection won't cause an impossible path. Either way add a creature temporary for figuring how big is the spot and try to stick around 70 Height and 50 Radius or such. If place looks small but navigable, you can go for 40 height and 20 radius - Bot will not deny this way if it has a purpose to walk through this path. A wrong added Path means reloading map without saving the screwed up work. After each good move it is advisable Saving MAP. You can use a temporary copy of map for figuring everything and dealing with custom paths.

Funky paths through holes where Bot jumps and fails to reach at target can be excepted from being created by using a small range then connecting manually missing paths. You can enhance this way Pawn's movement exactly as you like, except cases when geometry does damage. A ramp even if is very small as height, if it does an un-climbable surface it's a trouble maker. This will need to be covered with some blocker actor in order to simulate a stepable stair.

Paths over boxes won't need Combos, they are doable using PathNodes logically connected with reachFlag 9 - R_Walk and R_Jump.

Fields from builder (GUI) interface of this Window are not updated dynamically automated in Editor until it's getting a refresh by clicking on a field or crawling outside of Editor and coming back, then it will be painted.

For some reason if iterations count goes over engine boundaries, you can temporary start Editor using switch **-norunaway** until job it's done. Technically if Editor needs that option, map might be overloaded and this is not a good thing for UE1 assets.

If Map's geometry does funky paths in MonsterMode you can stick for Bot Paths only, these are more real to what Bot can do - Bot it's a small pawn compared to a titan, queen, etc. and novice Bot is clumsy at long jumps. After September 2021 probably builder won't mess that much here as long as Bot-Type test has the last word if Path will be created or not.

You can prevent a lot of paths heading to a bunch of Inventories by simply hiding them in Editor.

Whatever is not connected as expected, Builder supports moving a Path with option **bAutoMoveCDtoAB** and fields completed properly and logically. This is the stage when user has full control over movement and then Editor won't dictate dumb paths based on a Scout which is bigger than a Bot and creating lousy paths in ramps where boys are recommending to put nodes more crowded 300 500 UU. When you'll figure the geometrical problem of ramps, you'll understand what plain Editor does and what do YOU need to do for preventing lousy moves and not placing nodes more crowded because that's a WRONG solution in all the way told by various stories tellers without any prize taken for their Fake News over shared. A Map with Bot Support doesn't need ONLY BLUE paths, Red Paths are narrow Paths in Human format which are perfectly compatible with Bot too, don't get fooled by their mystical stories. If Bot does not move something is incomplete, a ReachSpec (/or more) is (/are) missing and nothing like a stupid color. Color stage needs to be in account in SP/Coop/MH maps aiming Monsters, AlarmPoints, PatrolPoints correctly mapped. The builder is reporting data concerning these Paths and you can have a clue sooner or later what's wrong and where and you can fix the problem as you like, you will also be capable to detonate mystical stories by making things normal with your own hands.

Another advantage is using builder combined with **MapGarbage** builder. This will allow user/mapper to create its own navigation network by producing manually all paths from map. In simple maps paths can be done in 15-30 minutes. MapGarbage can deploy InventorySpots (any type - even custom ones) over inventories which can be all selected by the same builder and using a defined height over Inventory. All these can be chained into ”NavigationPointlist” and then, current builder can be used for auto-connecting nodes at user desire. Combos are easy to be created exactly as user needs them - True one-way directions. Because of calculations done by auto-connector, those potential paths causing a clumsy Bot are not really created, you can prevent paths having lousy angles with ledges, etc, etc. Paths over Movers (bridges or secret floor-trap-doors) can be created with full manual control - it's what Editor doesn't do without hackish methods which are not really working in maps without brushes. If you have idea about navigation generally, you can do a perfect work, either way if you are drunk or on drugs, results will show what was in your brain.

Working stage

Builder during paths definition task takes a bit of time, it logs what it does and it will attempt a count for iterations. Due to log flood, builder is closing log Window and it will update viewport from time to time showing working status. At final point it will report time taken, number of reachSpecs added, iterations at UScript Level and opening log Window. Details might be relevant for figuring what was done at nodes.

Builder WILL NOT create paths in maps without PlayerStart actors. Since such a map is not really playable, pathing it doesn't make any sense.

Steps operated by builder in creating paths:

-> Removing previous Navigation Network and cleaning trash bytes from nodes - except when user wants to keep older Paths-Net untouched;
-> Iterating through all Inventories for creating InventorySpots - **excepting those bHiddenEd = True** - updated August 2021 and excepting already marked ones;
-> Creating NavigationPointlist - if it doesn't exists;
-> Log Window will be closed if it's opened;
-> Finding Groups of Inventories for establishing a "Master" which will have external connections out of bunch - area dictated by **BunchSize**;
-> Creating Paths through bunches of Inventories attempting a minimal load;
-> Iterating through Navigation Points and calculating "PointReachable" in defined range using a Giant Scout for MonsterMode and a Smaller Scout for Bot Mode, nodes will have paths until **PLimit** is reached and then Next Node is taken for dual probing (Bot and Monster type). Whatever Node found somehow closer to other and linked won't have multiple paths to that spot but only one;
-> pathed nodes are marked - viewport is receiving updates from time to time during pathing process;
-> When number of paths is filled at 10 builder will proceed at next node;
-> After ending task, markers are removed and all work it's reported;
-> Log Window it's gracefully opened.
Due to simplified operation mode, builder looks a few times faster in adding paths compared to original C++ methods from Editor, and it does more simple paths allowing user to connect nodes later and to polish navigation network as desired. Builder is hunting reachable nodes until limits are reached without going over boundaries. In a High ScanRange and a lot of charge with Navigation nodes more nodes are excepted because **PLimit** has been reached and no future connections are in account. In exchange, certain areas might be funky, it depends on map type, how is geometry, etc.

Modifying a Path

This chapter requires understanding about those lines drawn. By adjusting a Path which is a R_Walk into a R_Walk R_Jump (flag 9) we need data completed in **TempReachSpec** fields. Future modified reachSpec it's a mirror of this dummy reachspec. Made sure about having a START, an END, CollisionHeight, CollisionRadius and ReachFlags, distance is calculated automatically if is not added. Distance can be different if reachspec has been moved between other two navigation points. After these modifications map needs to be examined for specs locations. **bDefragReachSpecs** applied after all modifications will solve problem. Here is needed index of evil reachSpec or else nothing is touched. You can have it by listing "**bShowNodeData**" option from this builder which is enough for figuring what sort of paths are having source node and target node. Index is the number from Paths lists - those from 0-15 fields.

Feel free to develop your own methods more or less advanced. These are MY solutions which might not be the best ever due to UE1 limitations and blabbering at UScript Level.

July 2022 Update Notes

In overcrowded maps with nodes, when stock UT is used for building paths, a lot of ReachSpecs are only drawn in Editor but not placed inside Navigation Network. As result, they are not used in calculation for routing process causing a questioning navigation for A.I. The option for fixing them will harvest these lost ReachSpecs and adding them into Navigation Network. However, if they are too many they won't fit into specific lists from nodes. Such maps would need a revision concerning habits for adding paths just for the sake of pathing and not for a high-quality network. The sub option

bAndNullExtraSpecs won't add these junk ReachSpecs into Navigation Network but making them null ReachSpecs and causing Editor to draw only Paths that are used during routing process in run-time. This is the true image of navigation and not what Editor shows as lines (arrows whatever). Technically this is not the ideal case (null specs are still taking space in map) but is a option for figuring assets during working stage. However, if results are not bad, we can use the other option for re-indexing ReachSpecs in order to have those VALID ones as first in ReachSpecs Array leaving those unused re-indexed as last ones.

By example (virtual example with 8 Paths - 3 being null) if we have initially after several editing stages something like here,

0	1	2	3	4	5	6	7
Valid	Valid	Null	Valid	Null	Valid	Null	Valid

after applying the **bReIndexSpecs** option, it goes this way,

0	1	2	3	4	5	6	7
Valid	Valid	Valid	Valid	Valid	Null	Null	Null

5 6 7 remaining for future usage if someone else wants extra adds.

I did this option for a bit of wrapping, assuming if something iterates through ReachSpecs perhaps is better to find VALID data first and not a mix of ReachSpecs attempting to speed up finding desired ReachSpec. This option can also fix some error concerning a path ”removed” manually if user has a doubt about a mistake done during working process. Option will clean network from all paths, it will harvest VALID reachspecs with adding them into network and placing their new index over there, the rest of null ones are re-indexed at tail of array. In theory I would completely delete Null ReachSpecs but at this time I don't have available any native extension for doing this task - XC_Engine can only modify/create ReachSpecs known as paths lines.

User also might want to de-reference manually paths in certain cases for creating desired One-Way routes. Those paths removed can be nullified and later re-indexed at last positions for preventing run-time routing process to deal with them. Either way you can recycle them where is needed.

Update:

Problem of deleting null ReachSpecs looks solved somehow in **September 2022** after getting an idea... I was waiting a solution from someone else but... I decided to stop waiting two centuries...

A solution in C++ would be more suitable and I have some idea how to mess with them but... since only in 227 assets removal of a path is possible with altering index, let it be like that. To be honest even re-indexing them normally and defragmenting them would be possible under 0.1 seconds after removing a path but... it's not my problem if people in this century are working like in past century.

Credits timer:

EPIC - for delivering the stage called UT and UE1 where all these are happening;

Higor - for XC_Engine and some codes used as base and which I adapted for Editor;

Pikko and other mappers - for various test-maps where I could figure what I needed during pathing crowded maps.

Buggie - pointing some shots about special paths which could overload useless ReachSpecs (Combos, Teleporters).