PathsChecker

UnrealTournament Editor Add-On Builder

update October 2022

Description:

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

Audience: The plugin is addressed to those who do maps for all type of players - pathed maps, it aims to prevent the lost time with X tests of the map on the activity of artificial intelligence.

Purpose:

#1) Virtual players (Bots) aim to collect weapons, ammunition, armors of all kinds, etc. Directly from the Unreal Editor application we can perform a primary test to check the navigation to each actor that can be a destination of the navigation process by checking routes availability from each PlayerStart to each possible destination actor.

It is helping mapper to do logically connected Navigation Nodes in order to gain a better activity of Virtual Players (Stock Bots). Final tests concerning movement capabilities or quirks should be operated as usual - spectating one and/or two Bots as described in original tutorials. Until last stage we might be aware of potential problems right from Editor.

#2) Such tool should be embedded in Editor as a default option/feature but Epic did not have free time for it, all it was a rush for releasing an incomplete software.

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 

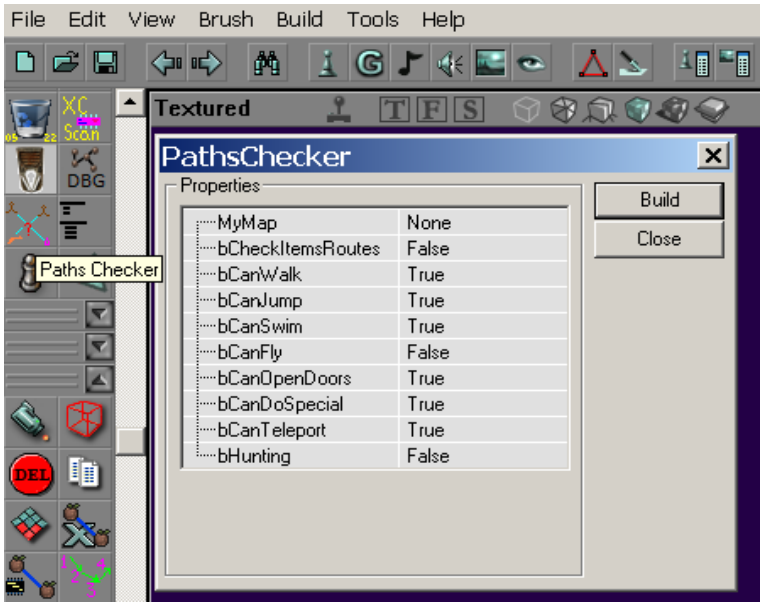
button activates the execution of a function. Other variables can be TRUE but are connected with the main feature (experiments at mapper's will), they do not do anything alone, all are going to be explained later.

Installation:

U file goes in **System** folder or whatever UT Path for U files. 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** line):

```
[Editor.EditorEngine]
...
EditPackages=ExtendedBuilders
EditPackages=RahnemBrushBuilders
EditPackages=TarquinBrushBuilders
EditPackages=TarquinExtrudeBuilder
EditPackages=DavesBrushBuilders
EditPackages=PathsChecker
...
```

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 fore mentioned package set in "EditPackages" array from **UnrealTournament.ini** file:



Explained vars:

Variable	***Connected data***	***Description***
TheMap	-	AutoCompleted by builder. Recommend manual completion only for failures.
bCheckItemsRoutes	- bOmniSeeking - scan entire network for gaps.	Main TASK and Purpose Builder starts testing paths from each PlayerStart actor or nearby spots heading to all Inventories, AmbushPoints, FlagBases (for CTFGame), AlternatePaths, ControlPoints (Domination), MonsterWaypoints (MonsterHunt checks), MonsterEnd (you won't see to many maps that are normal, no worries). Sub-Option takes time and iterations in maps loaded. In maps that are using One-Way paths logically executed you can expect a high number of failures. If map is supposed to be consistent and normal, failures are pointing issues.
Experimental Stuff		
bCanWalk	-	Tester ignited in Editor should logically be capable to walk or else... Recommended to be left alone...
bCanJump	-	Tester ignited will be capable to jump. Paths requiring jumping will deny Pawn if it doesn't jump.
bCanSwim	-	Tester ignited perhaps should swim if something is placed or has route though water - logic.
bCanFly	-	Tester ignited has flight capabilities but... Plain Bots won't do this. Testing results might be False.
bCanOpenDoors	-	Tester can be rejected in certain cases, depending or route calculus and paths definitions.
bCanDoSpecial	-	Tester ignited can see routes through Combos (LE-LC-LE).
bCanTeleport	-	Tester ignited can deal with this self-explanatory stuff.
bHunting	-	Tester will use a bit of brute force for navigation taking in account the closest node to the goal - During run-time Bots are not very tempted to use this way and then finding a route normally is conclusive by having this FALSE. **Cave At:** In run-time a goal placed in a small box-room covered by a door won't grant navigation for A.I. Pawn - only using this brute-force which usually is not really used in stock games, so here you can earn a false positive because Movers are not really colliding in Editor unless you'll activate their polygons to be visible and later map will need to be re-build (recompiled as others are saying)...
Altering these extra-options will deliver different results or... No results. You can toy with them if you know what you do.		
Update September 2022 - After probing map, all debris navigation data created by routing process will be completely deleted and not on the way of stock natives, here will be a full CleanUp.		
Update October 2022 - Added optional sub-option _bOmniSeeking. It will trigger builder to perform tests from every single Navigation Point to the rest of Navigation Points. Yes, here might be recommended parameter ***-norunaway*** in Editor's command-line.		

Builder's results info – Logic Cave At-s:

In theory if builder says "Greetings !..." this means that Engine was capable to calculate routes from all PlayerStart actors to each possible goal known so far - Assault stuff is not implemented, that game-type has errors and this is my response when things are left unfinished, but routes for items are still checked.

If Engine was finding routing logic, map still needs to be verified by Bot as a secondary important test during run-time. Several dudes are not knowing to deal with JumpSpots and their logic placement. This may deliver impossible paths locking Bot in spot. Engine has a logic route data but in reality A.I. movement might go wrong. Either way "trap" type located item might have a route in spot, but there might be no option for leaving that spot in order to keep roaming. This is why more extra quick checks are doable by more builders which community is having before wasting time to look at Bot in a test game session. If builder fails to find a logic route, there are big chances to not see Bot going to flawed item until during run-time there is another available goal and Bots will gain flawed items in a combat situation due to the desire for nearby stuff. Technically everything should be connected with logic data if we want Bots active and game balanced – not in "**Deck16][**" style. Debugging a broken route can be easy or hard depending on what user does. For stock UT if we have XC_EditorAdds installed in Editor, paths are drawn like arrows – later UT patches are having this as a default feature – HELPFUL if you ask me. User will have to look around for arrows linking routes in direction of item point by point. If such an arrow is missing (high mystical jumps, geometry issues) the failure is logic. A good route means that road to target should have an arrow pointing movement at each Navigationpoint which is involved in route oriented in direction which Bot will follow. The other "arrow" direction is a reversal direction good in return for any other goal. Two normal Navigation Nodes are having Two Paths (ReachSpecs), moving forward and back or else we have One-Way paths which are having other normal logic – jump down from house/box, but we cannot walk on walls to reach there, we will use inventories if available and navigation nodes with special directives, and these will need to have the same arrows node by node. If you are using only stock Editor 436/440 and nothing else, your life goes harder with debugging a route, and here you are on your own. Perhaps your experience is helpful and you don't need builders for checks – and then... Greetings !

To not forget that we can have maps with One-Way paths. In such maps it's logic to see flawed routes. Connections there are into a single direction and there is no returning route. Here we are interested only about certain items, goals, those from behind searching point won't be available.

Update September 2022

Not yet implemented reversal checks (for said traps locking Bot in spot) but we can have a full "CleanUp" for network in order to get rid of debris data resulted from routing calculus. Internal chains created that are not shown in plain Editor are destroyed. These are not useful at all, during run-time in network data is dynamically mapped when Pawn does a call to these routing natives from Engine and this is happening during pathing process when Editor is probing routes. As result, if you are checking a map using this builder, you will earn a clean Navigation Network without useless debris data.

Update October 2022

Sub-option usable optionally **_bOmniSeeking** is a little monster. We can have a clue how many Navigation nodes are not part of any route. Here is taken in account all Navigation Nodes if can be Target from every other Navigation Node. As result you can remove/adjust/move an offending node elsewhere or completing some forced links around bugged spot. If it comes to do such checks

in maps with hundreds of Navigation Nodes, the process can take time if your machine is slow. In certain stock maps, this check is doable under 1-2 seconds on a dusty rig. Definitely parameter **-norunaway** from Editor's command line must be added if it comes to do checks in maps that are having hundreds of navigation nodes.

As a hint, I'm starting Editor from a batch file delegated to delete that lousy junk called "Running.ini" with zero bytes length. Here I can use any switch available for Editor. This is said batch file (extension is BAT):

```
@echo off
echo Start Unreal Editor 440 + XC with higher priority
start "Unreal Editor 440 + XC" /WAIT /ABOVENORMAL /MAX UnrealEd.exe -norunaway
del Running.ini
exit
```

You don't need a high priority if this will cause issues, for me there is no issue, I did not have any at this point. When Editor is crashing, it will do it in the same way as it does since 1999.

Builder will print in **Editor.log** file all results. At this point you cannot see all data. For debugging you will need to close Editor. Walk through System folder and just rename Editor.log based on map-name (sample: Agony.txt for map DM-Agony) and TXT extension. Now you can open that TXT file and look for string **Route Not Found** or **Not Found**. If some Node can be seen multiple times as flawed you can check what's up with that - or a group of nodes bunching on some island that is not connected with the rest of map only as One-Way or not connected at all. A quick example is map **DM-Malevolence**. There are reported flaws, it can be seen *PathNode4* that is not found from anywhere - it's happening because this node is useless - placed outside of map, into the void.

```
...
PathNode4: Searching...
PathNode4: Route Not Found from LiftCenter1...
...
PathNode4: Searching...
PathNode4: Route Not Found from LiftExit2...
...
PathNode4: Searching...
PathNode4: Route Not Found from LiftExit3...
...
PathNode4: Searching...
PathNode4: Route Not Found from PathNode0...
...
...
Processing: Full Scan from PathNode4.
PathNode4: Check Failed !
...
```

In other hand **DM-Agony** doesn't include any flawed node that cannot be visited from any location of map where are placed other navigation nodes. Yes, map has lost paths out of network, but what map has available as navigation data will help Bot Pawn to move where it wants to go.

Credits:

EPIC - for delivering the stage called UT and UE1 where all these are happening and the lack of testing tools available in UT's Editor, in addition ZERO codes for inspecting Paths/ReachSpecs right in editing stage of a Level (known as map);

Buggie - for some light concerning a Tester Pawn and its needs in Editor's Environment (which I should manage to figure but I have only a tiny brain...).