RV SmartAI documentation

Join Ronis Vision's Discord support channel

note: always use online documentation if possible, as it will be more current than pdf version provided with package.
[Online documentation]

General overview

RVSmartAI works by usage of so called AiGraphs, those are graphs where you visually design your ai logic. Graphs are made of nodes.
After each node decision is made, and one utility is selected as 'winner', graph will go from this node's winner utility to other node via connection.

AiGraph components organisation/hierarchy:

- AiGraph
    - StageNode
        - AiUtility
            - AiScorer
            - AiScorer
            - AiTask
            - AiTask
            - AiTaskParams
                - AiScorerParams
                - AiScorerParams
            -
        - AiUtility
            - AiScorer
            - AiScorer
            - AiTask
            - AiTask
            - AiTaskParams
                - AiScorerParams
                - AiScorerParams

RVSmartAI components overview:
AiGraph - this is graph where you visually design your ai logic using nodes.
AiGraph consist of the following graph elements:
StageNode, AiUtility, AiTask, AiScorer.

StageNode - main type of node that you will be adding to your graphs. There are different types of StageNode: HighestUtilityWinsStageNode and FirstUtilityWinsStageNode. Each StageNode consist of AiUtilities.

AiUtility - has an array of AiScorers and AiTasks.

AiScorer - component that returns some value - score based on it’s configuration

AiTask - component that will execute some code if AiUtility it is assigned to wins

AiTaskParams - special type of task, that work on arrays of data;
it will score every data element from passed array using it’s AiScorerParams;
Then it selects ‘best’ data (one with the biggest score) and execute it’s code using this best data as parameter.
Example would be GoToBestPosNearMoveTarget task, this is AiTaskParams of generic type Vector3, because it work on positions. You can add to it many different AiScorerParams<Vector3> to score every position(Vector3) - for example using ProximityToMeAiScorerParams which scores every position based on distance to our ai agent, and this task will make agent move to position with the highest score - in our example that would be position closest to our ai agent.
AiTaskParams and AiScorerParams are generic types and you can assign scorers to tasks only with matching generic type.

How it works

RVSmartAI works by evaluating AiUtilities on each StageNode, then selects one of them as ‘winner’. How StageNode selects winner depends on StageNode type - AiUtility with highest score or first that scored anything. AiUtilities get scores by their AiScorers. After StageNode selects it’s winner AiUtility, it runs it’s AiTasks if there any, and goes to next node if there is any connected to it via output port.

Working with SmartAI

Quick start guide

This quick start guide will show you how to setup working AI step by step.

- Create new scene.
- Add to the scene AiAgent prefab located at *Assets\RVModules\RVSmartAI\Content\Prefabs\AiAgent.*
- This will be our AiAgent. It contains necessary minimum for Ai to be able to move using Unity navmesh pathfinding and have spatial awareness. For more detailed description of AiAgent prefab components see AiAgent components overview.
- As we want our agent to move we have to create some ground for it and bake navigation mesh. Create plane, place it at 0,0,0 and set scale to 10,1,10. This will be our ground.
- Set plane as navigation static and bake navigation by pressing Bake in Bake menu of Unity's navigation window. This will allow your agent to move on the plane.
- Now you've got all necessary elements for your AiAgent to be able to move and use AiGraphs! All that's left is to assign some actual AiGraph, which our agent will use.
- Drag Wander prefab located at *Assets\RVModules\RVSmartAI\Content\Graphs\Wander*
  into AiGraph field of Ai component attached to our AiAgent.
- Congratulations! You just configured you first SmartAi. Now to test it press play. Your AiAgent should wander around on plane(going to random places with short pauses).

Basics

To make your agents work you need Ai component added to every agent that you want to use SmartAi. Ai component needs to have AiGraph assigned. Note that you can only assign AiGraph prefabs for Ai. This behaviour is forced because it could quickly become confusing if many agents in scene would use instances of AiGraph. If you need some small changes for some specific agent you should use prefab variant of AiGraph instead.

Working with graphs

AiGraphs are SmartAi's core, and SmartAi graph window is place where you will spend most of time designing your Ai.
Note that there are two ways of editing AiGraphs: normal, where you work on your scene, and also in prefab mode. While system is designed so that no special actions are required from user editing in both modes, you should use prefab mode only when you work with prefabed elements - but you should never change AiGraph game objects hierarchy in any other way manually - it is maintained by SmartAi system, including making prefabs out of graph elements.

Opening graph editor

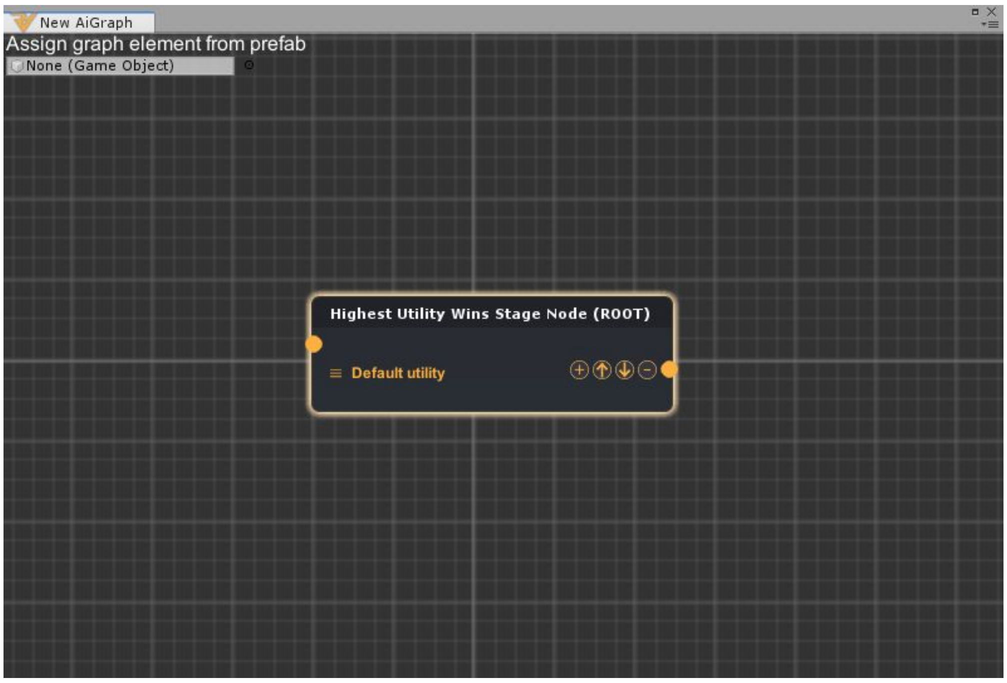
To open SmartAi graph editor, you need to first select some AiGraph and press *RVSmartAi -> Open SmartAi graph window* from top toolbar.

Creating new AiGraph

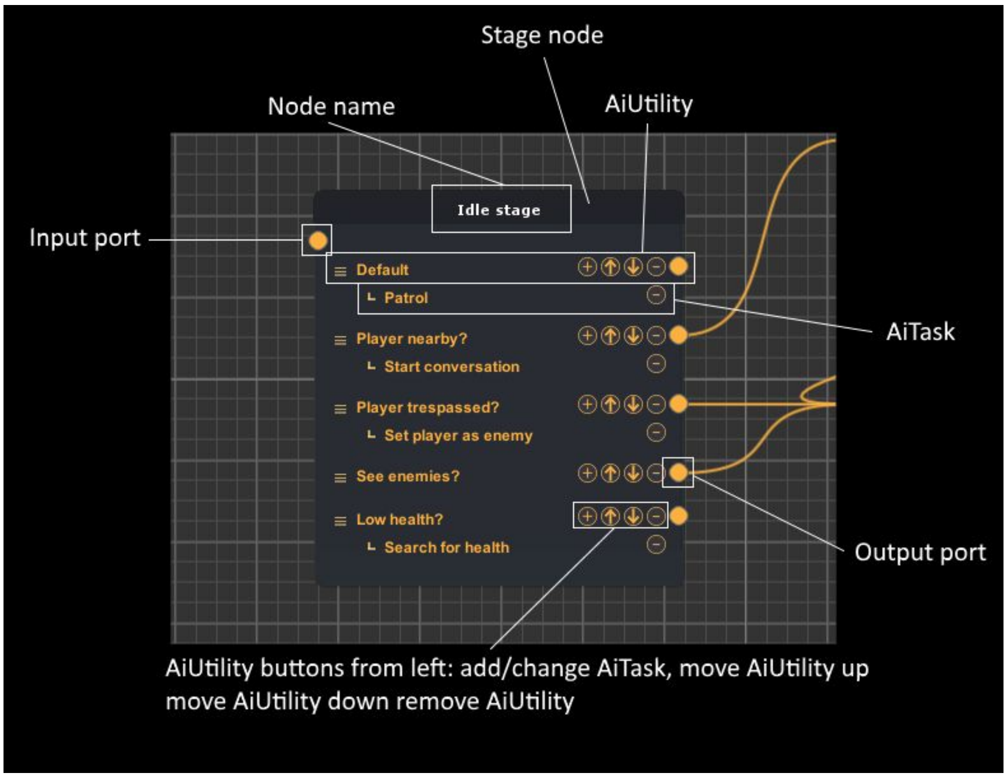
Select folder where you want it to be created and then press *RVSmartAi -> Create new AiGraph* from top toolbar. This will create prefab at your selected directory and you'll be able to type in your new AiGraph's name. After entering your AiGraph's name AiGraph editor will open automatically with your new, empty AiGraph.

Graph editor

AiGraph editor overview


New AiGraph
Assign graph element from prefab
None (Game Object)
Highest Utility Wins Stage Node (ROOT)
Default utility

Node overview



To create new node press rmb anywhere in editors background(grey grid) and select type of stage you want to create.
Every selected graph element's properties will be displayed in Unity's inspector after selecting it with lmb. Custom inspectors are written using proper API, so they fully support undo system and prefab properties overrides. You can expose any data from your graph elements that Unity can serialize using *SmartAiExposeField* attribute, which also has constructor overload with description as parameter - this description will be nicely displayed in inspector over your exposed field.
If utility or task font is greyed out it means it's disabled. It'll be skipped at runtime.
Blue font means this graph element is a prefab.

Working with prefabs

Creating prefabed graph elements
You can make prefab out of any graph element, including whole nodes.
To do so select your graph element of interest, then click rmb on node's title bar and select *Make selected graph element prefab.* This will automatically create prefab in directory where your edited AiGraph is located and selects it.
Despite having this option in some cases(nested prefabs) you will have to go into prefab mode and prefab graph element manually. To do so use *Select corresponding game object* from node's context menu, and prefab selected game object just like you would any other Unity game object. SmartAi will automatically detect that it's been prefabed.

Using prefabed graph elements
To add earlier prefabed graph element to your AiGraph, first select graph element to which you want to add your prefab - if it's node then you don't need to. Then just drag your prefab into object field displayed at top left corner of graph editor.
Note that SmartAi analyses your prefabs and will allow only for proper prefabs assignment, so you don't need to worry that something breaks if you try to assign to wrong type of graph elements or use wrong prefab.

Unpacking prefabed graph elements
For your convenience there is also option to unpack prefabed graph element without going into prefab mode - select your prefabed graph element, click rmb on this graph element's node and select *unpack selected graph element prefab.* Despite having this option in some cases(nested prefabs) you will have to go into prefab mode and unpack it manually.

Prefabs limitations
There are some limitations of what can be done with prefabed objects in Unity.

RVSmartAi takes all of those limitations into consideration, and will block not allowed operations, like removing prefabed object’s children. In such cases, for example if you don’t want some elements from prefabed graph element procedure is the same as with any other prefabs in Unity - you can simply disable them.

AiGraphs prefab variants
AiGraphs are standard Unity prefabs. You can make variant of your graph just like you would with any other Unity’s prefab. All above apply, and you don’t need to take any additional actions regarding AiGraphs prefab variants. You can manage prefab properties overrides directly from graph element’s inspector.

Runtime graph debugging and editing

To see what’s our AiAgent is currently doing while in play mode press *Debug graph* on Ai component. This will open and focus SmartAi graph window, where you will see decision process in real time.
In debug mode you will see numbers next to every AiUtility - those are last scores that those utilities scored.
Node at which Ai is currently will be highlighted with cyan tint.
Winning utility will be also highlighted with cyan frame, and will also highlight connection, making it easy to see where Ai will go next. It is good practice to set *graph steps per update* in Ai component to one for debugging. Just keep in mind that your agent ai behaviour will often not be deterministic in terms of different values of this setting. In other words *graph steps per update* can result in different outcome(behaviour) even for identical situation.

During runtime debugging you can do almost all operations as you would in edit time. That includes adding new elements, changing connections between nodes etc. Changes in debug mode, just like for any other Unity’s component aren’t saved(they are lost after exiting play mode), which might seem confusing at first since AiGraphs are prefabs, but don’t worry - in debug mode you only edit runtime copy of your graph, not actual prefab.
However you will often want to save - all this runtime tweaking wouldn’t be very useful if you couldn’t easily apply this perfect configuration you’ve done at runtime to your graph. This is very simply solved with prefabs - just make prefab out of element you tweaked at runtime(see Working with prefabs on how to), and after exiting play mode add this prefab where needed. Even if you can’t directly replace your graph element(for example if it’s variant or child of other prefab), you will still have all your runtime tweaked data in your prefab, and can apply it’s values to old one as overrides.

Fixing graphs

SmartAI is provided with tool for analysis and automatic fixing of AiGraphs.
To use it, open AiGraph in prefab mode then select from top toolbar
SmartAI -> Analyse selected AiGraph
or
SmartAI -> Analyse and fix selected AiGraph.

You should always make backup of you graph before using Analyse and fix selected AiGraph option, since even if it's not 'broken' it will reassign all references, so for example order of your graph elements will most likely change, so always analyse graph first, as this action doesn't change any data in your graph, and use fix option if there are some issues found.
If you are afraid of using automatic fix, or don't want your graph elements reordered you can of course manually fix found issues if you know what you are doing (you can always contact me by Unity forums or discord for such scenarios).

Automatic analyzer will log every issue with detailed information what's the issue and reference to problem-causing object, so you can click log and Unity will highlight this object.

Reflection-based scorers and tasks

You can find those type of graph elements at
RVModules\RVSmartAI\Content\Code\AI\ReflectionBased\

Those graph elements can be very helpful with fast prototyping and allows you to avoid writing graph element for every property you want to access via SmartAI graphs.
This convenience comes at some price, so there are some considerations you should take before using it in your project: to avoid using expensive MethodInfo.Invoke(), we're using expression-based solution that's using dynamic code generation.
That means it won't work on platform that don't support JIT compilation(that's basically means using IL2CPP scripting backend).
To workaround this SmartAI in such case will automatically fallback to using normal, reflection based MethodInfo.Invoke(), which is much more expensive to call.
You should be aware that despite all the effort to optimize them, those reflection-based graph elements will always be slower than manually writing normal graph element (they need to be 'cooked' at initialization), so if performance is critical (low performance mobile devices etc) you should avoid using them extensively, especially when using IL2CPP scripting backend.

Content overview

Examples

RVSmartAI has several example scenes to help get your started. Modify provided example graphs to see how changes to them affect AIs behaviours.
You can also make prefab variants from them since graphs are prefabs in RVSmartAI.
Example scenes are located at *Assets\RVModules\RVSmartAi\Content\Examples*.
Examples are non-interactive, meaning in play mode you will see behaviour automatically presented to you, no input needed.

Overview of example scenes

Flee example - example shows how to design simple ‘flee’ type graph where white characters will try to keep safe distance from blue character, which is just wandering around randomly.

Follower example - example shows how to design simple yet reliable ‘follower’ graph, very common behaviour in rpg games(player companion, summon etc). White characters will try to keep close to blue character, while avoiding each other(two different followers will never try to go into same place near blue character). They will also intelligently try not to block their leader - if distance to leader is too close, they will also move away from him to allow him to pass them. All of this logic combined makes them reliable followers and gives an impression of real intelligent behaviour.

Patrol example - example shows how to design simple patrol type behaviour, where agent will move through predefined waypoints in two different ways - in given order, or random order. This example also shows nicely taking advantage of prefab graph variant(random patrol graph is variant of patrol graph).

AiAgent components overview

Examples provide you with simple and easily extendable AiAgent with features that make it aware of its surroundings: it will know what other agents are around him, where he can and can’t go, and moving logic. Everything is designed modularly, as separate components, and can be easily replaced by your own scripts, as long as they implement necessary interfaces.

AiAgent - component with IScannable interface, so agent can be aware of other agents.

AiAgentGenericContext - this is simple example of IContext implementation, used for AI general awareness/’information storage’. It’s just component that graph’s elements can use to get necessary data like nearby other AiAgents, get list of waypoints etc.

PhysxEnvironmentScanner - component responsible for spatial awareness of AiAgents: it uses Physx queries to see what other objects of AI interest are around him. Scanned objects are stored in ‘nearbyObjects’ list of AiAgentGenericContext. For objects to be scannable by AI they need to implement IScannable interface.

PhysxMovementScanner - scans for walkable areas around provided position for AiAgent. Can be configured to get the best compromise between accuracy and performance of scanning.

UnityNavMeshMovement - moves your AiAgent by using Unity’s NavMeshAgent. You can very easily swap this to your own movement logic without changing any other components, by implementing IMovement interface.

Graph components overview

RVSmartAI provides you with lot of ready to use graph components to play with, all designed to be extensible and usable in real world project. Almost all of them are related to agents movement, since all other behaviours, especially combat related are too much project/genre specific and are provided by future additional packs based on RVSmartAI.
For help with usage of them check out provided example graphs.

Script templates
There are many script templates for creating your AiScorers and AiTasks that you can create from Asset -> Create -> RVSmartAI menu.