



# TALENTUS

**SKILL/TALENT TREE ASSET FOR UNITY**

## THANK YOU FOR PURCHASING TALENTUS.

Talentus – skill/talent tree asset for Unity is a unity package allowing you to create and maintain skill/talent trees the easiest way. An intuitive editor is making sure the creation of a tree is a breeze, the wizard let you design a playable UI directly based upon the design and the engine makes sure everything is tight together at runtime.

As software, and thus this package, is made by human hands it is prone to issues. In case you have some feedback, features you would like to have implemented or even bugs please report those by sending me a mail to [support@cygnusprojects.com](mailto:support@cygnusprojects.com). I'll make sure to get back to you ASAP. On a side note: *I can only accept requests when you provide me your Unity invoice id.*

If you want to spread the word that Talentus is available on the asset store, please take some time to rate the asset.

Whishing you all the best using my product!

Wim Vancoillie

# CONTENT

Thank you for purchasing Talentus. ....	2
FEATURES .....	4
Design .....	4
Runtime .....	4
Not included .....	4
RELEASE NOTES .....	5
INSTALLATION – GENERAL SETTINGS .....	7
Unity 5.6.3p1 and up .....	7
Installation .....	7
Talent/skill tree design .....	9
TALENT/SKILL PROPERTIES .....	13
CONNECTION PROPERTIES .....	15
TIER MANAGEMENT .....	18
UI TEMPLATE .....	20
GET IT TO WORK AT RUNTIME .....	22
Variables .....	26
Methods .....	26
Events (1.0.2 or higher) .....	29
FAQ .....	30
What version am I currently running? .....	30
TROUBLESHOOTING .....	30
A talent seems to be missing in the editor, how to get it back ? .....	30
When I run my game the skill tree is visible but the root node(s) are disabled. ....	31
When I run my game the skill tree doesn't display the images nor the buttons. It doesn't look like the tree is disabled though.....	31

# FEATURES

## Design

- Extended custom editor to design a talent/skill tree
- Support for multiple categories within one tree
- Can be used for any design
- Multiple levels possible for one skill (and cost)
- Support for multiple branches within one tree
- Support for required and optional connections (new in 1.1.0)
- Ability to respec bought skills (new in 1.2.0)

## Runtime

- Wizard to translate the design in a working UI
- Engine capable to support all possible skill trees and conditions
- Engine capable to resolve required and optional connections (new in 1.1.0)

## Not included

Skill icons, we only provide an alphabet for testing. A lot of good skill icons can be bought on the asset store. Within this manual we will use the icons by ***Super Game Asset***.

# RELEASE NOTES

## Version 1.3.0 (August 2018)

- Fixed focus bug when switching to another skill while editing a field in the inspector.
- Added an explanation field to the talents. Can be used to store anything as text.
- Added a description field to all the cost levels.
- Added an overload on the CanBeUnbought method so all bought skills can be passed along iso fetched on every call, see the Tree Respec Example scene to find out the usage. This will lead to performance gains when you respec skills in big trees.
- Fixed a bug on the CanBeUnbought event capturing leading to performance gain.

## Version 1.2.0 (July 2018)

- Fix bug where the color of a circular connection wasn't set properly.
- Added the ability to have a respec on talent/skill base.

## Version 1.1.1 (June 2018)

- Option to add a connection properties component to a connection UI object in case of using custom prefabs.
- Fixed potential error popping up on closure of the UI wizard
- Fixed problem with UI colors on Unity Personal edition.

## Version 1.1.0 (June 2018)

- Added support for optional connections
- Rewritten the Talentus Tree Evaluator to support AND and OR connections
- Connections in the UI do have a reference to the connection, from and to talent objects
- Connections in the UI can now be colored according to their type (Required, Optional)
- Connections in the UI can be gameobject prefabs (through the UI wizard).
- Extra talent and connection script can be added using the UI wizard.
- Connections do get a proper naming when generated with the UI wizard.
- Added an event when the tree has been evaluated.

## Version 1.0.4 (May 2018)

- Improved performance of the TalentUI component

## Version 1.0.3 (March 2018)

- Added possibility to not draw the grid in the editor

### Version 1.0.2 (February 2018)

- Added events that are triggered when you buy, revert or apply a talent/skill (see the TalentEngineExtended script for an example)

### Version 1.0.1 (January 2018)

- Added support for disabled talents/skills icons
- TalentusEngine component virtualized so it can easily be extended. (added a TalentEngineExtended script in the example project)
- Added Runtime Save and Load functionality of a talent tree (see the TalentEngineExtended script for an example)
- Added a new example that has 3 skill trees in one UI (TripleTree), also in this example the root skills are auto bought on startup.

### Version 1.0 beta (November 2017)

- Initial version

# INSTALLATION – GENERAL SETTINGS

## Unity 5.6.3p1 and up

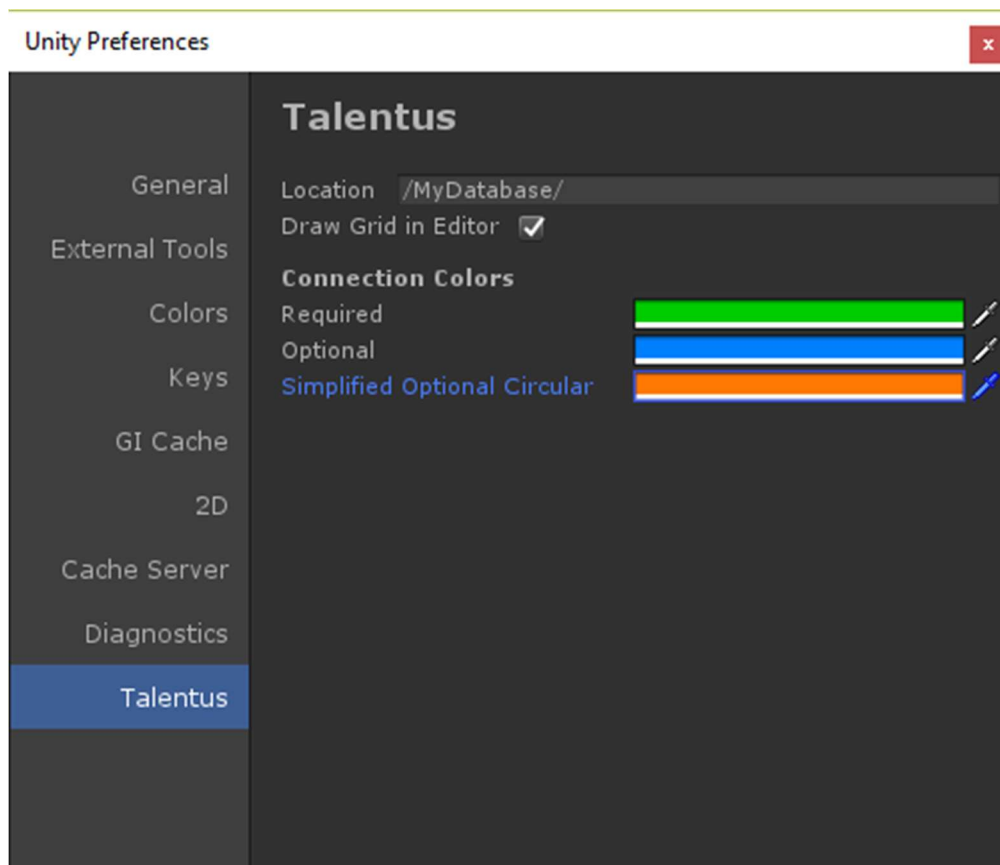
Talentus was designed using Unity 5.6.3p1 but we make sure the asset runs in every version newer than that.

We don't officially support unity version before the 5.6.3 lifetime cycle. Installing the asset in such a version is at your own risk.

## Installation

After installing the package from the asset store a new folder *Talent Tree* will have been created. You are free to move the complete folder towards a new location within you assets directory. Just make sure you keep the structure intact, all sub directories should stay as they are. One exception: you can delete the complete *Example* directory if you don't need the examples.

By default the store location for the skill/talents tree is set to `/Talentus/Example/Database/`, this can be changed in your project preferences (Edit\Preferences\Talentus). Make sure the pre and post slashes are provided (the system will add those when they are not provided).



**(new in 1.0.3)** When working with the editor you can enable or disable the grid by checking the checkbox next to 'Draw Grid in Editor'.

**(new in 1.1.0)** Connections can be colored depending on their type (Required, Optional). A color can be assigned to each type.

*Simplified Optional Circular connections* are some special kind of interactions between two talents/skills. See the Connection Properties to find out more.

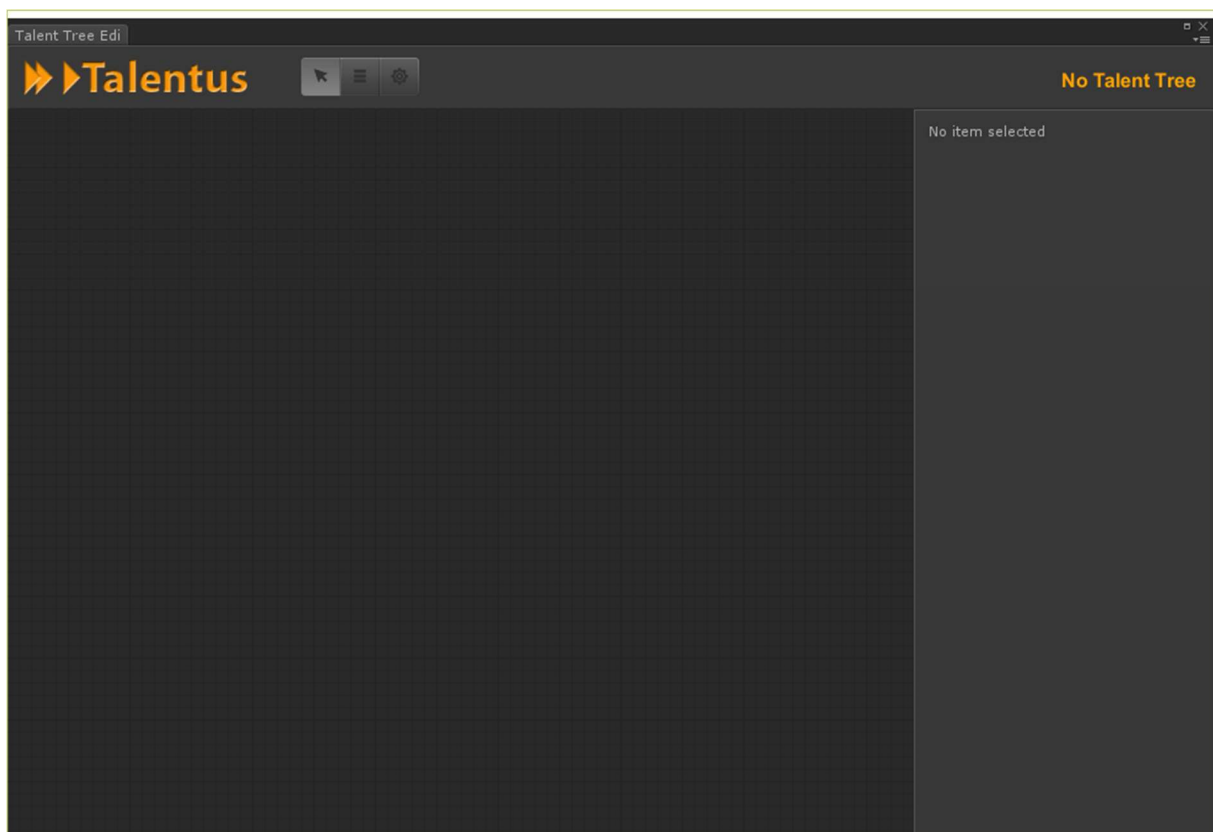


## TALENT/SKILL TREE DESIGN

Before we can start designing a tree we need to open the editor. This can be done by using the *Tool* menu and selecting the *Talent Tree Editor* under the *Cygnus Projects* item.



This will open the actual editor window:

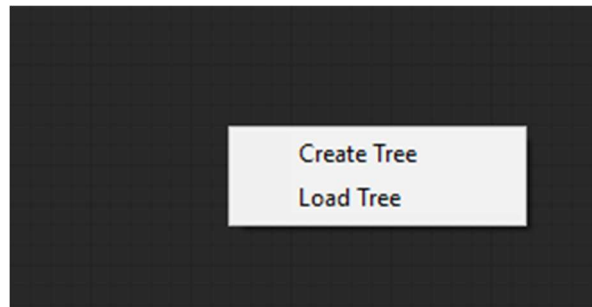


As you may notice there isn't actually a lot on the UI, this not to clutter your workspace. Besides the logo are some buttons: select, tiers (category) manager and global settings. On the right side you have the name of the current tree you are editing. You have a grid which is the actual workspace where you will design the tree, the area on the right is the custom editor (currently no item is selected, so empty).

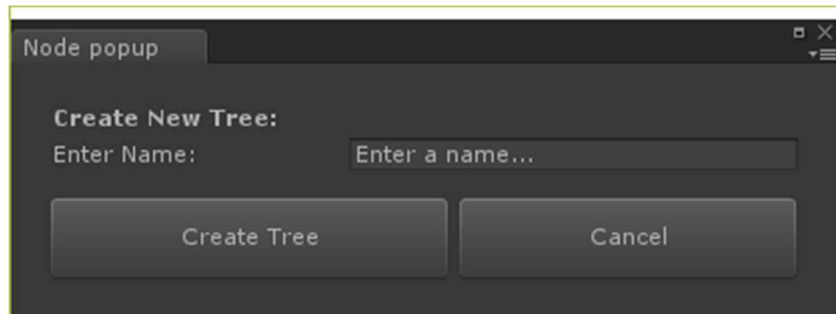
By right clicking in the grid you can display a popup menu with the following options:

- Create Tree
- Load Tree

It's clear what those options try to achieve. Note the location where the trees will be store is set within the preferences windows of your Unity installation (see Installation – General settings to know more).



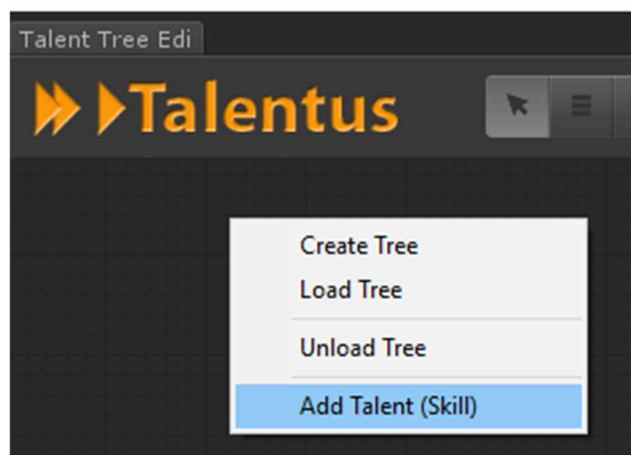
When creating a new tree a new creation dialog will be displayed asking for a name for the new tree:



Enter a name and select Create Tree to continue, second thoughts? Than click Cancel to abort.

When selecting the Load Tree option a OS dialog will point to your preferences folder to display all previous saved/created skill trees. Open one to continue (or select cancel to abort the operation).

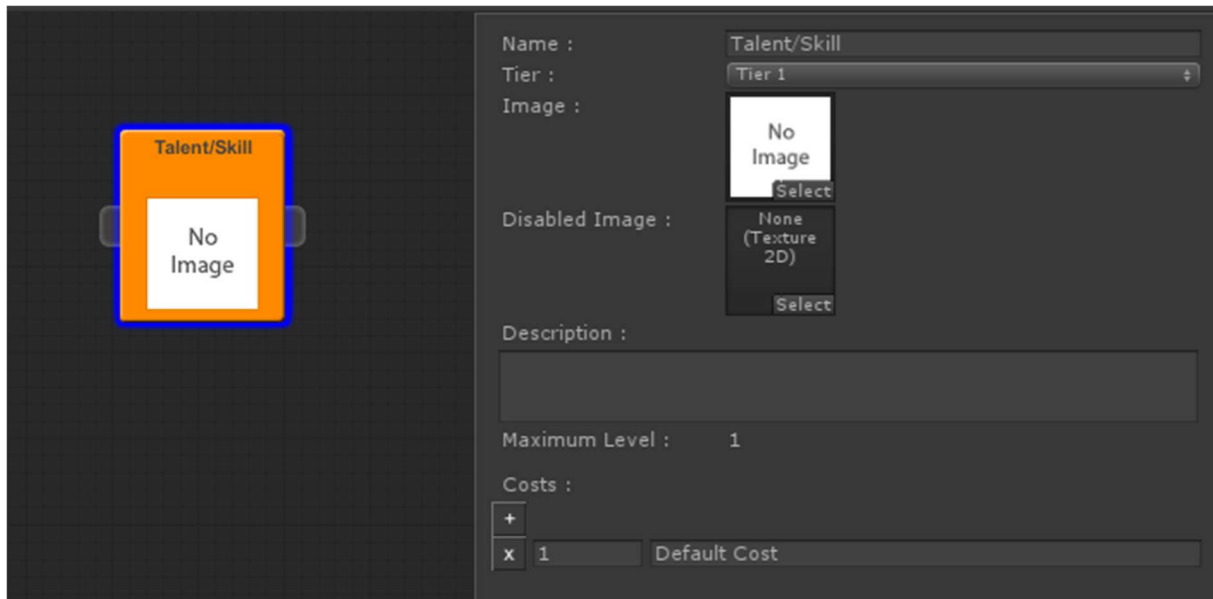
Let's add a single talent in this tree by right clicking in the grid area and selecting *Add Talent (Skill)* from the popup menu.



After doing so a talent/skill node is visible. This node can be dragged around and positioned wherever you like. You can move the canvas (grid) by holding the left mouse button on an empty space of the grid and drag along.

If you want to delete a node you can select it (left mouse click) and press the delete button.

Doing so will also show the talent/skill properties in the custom inspector of the editor. It's advisable to adjust at least the name of the talent/skill before proceeding.

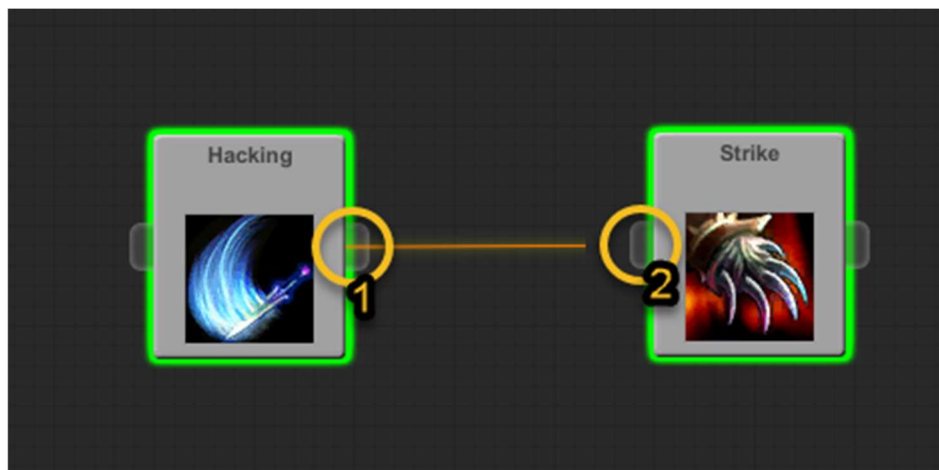


Adjust the properties to your liking, see the Talent/Skill property section of this manual to get a more in-depth description of all fields.

When done, repeat the process to add a second skill/talent in the workspace. Again adjust the properties to your liking.

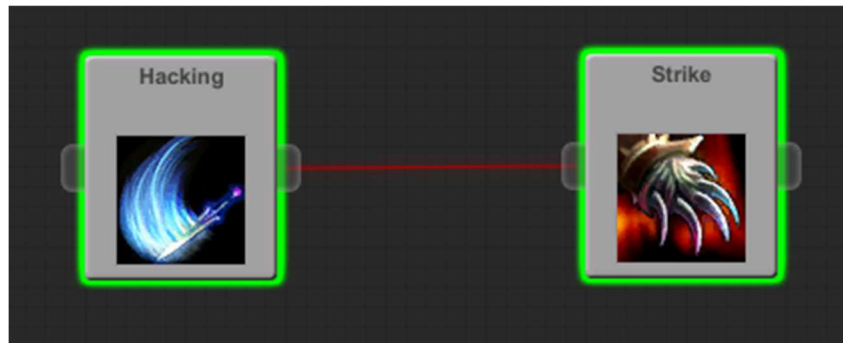
When done it's time to connect both talents together with a condition.

You may have noticed that a talent/skill has some rounded box on the left and the right side, those are called the connectors. The one on the left (2) is called the input connector, the one on the right (1) is the output connector. As we will make our second talent dependent on the first one we will left click in the output connector. Without the mouse button pressed you can move the mouse towards the in connector of the second talent and left click it to connect both together.



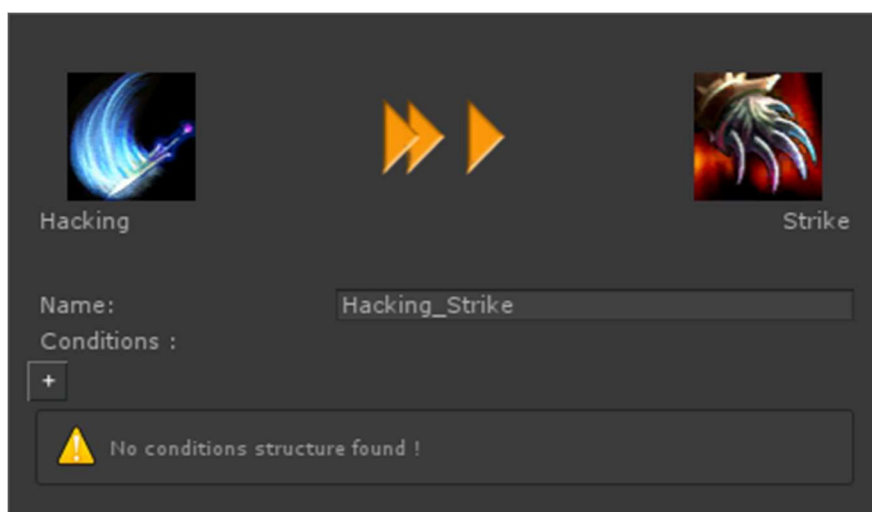
A connection will be created, represented by an orange line connecting both talents/skills. To delete a connection one can just press delete while the connection is selected/highlighted.

Click somewhere else in the canvas and you will notice the connection drawn is marked red.



As red means there must be something wrong, the system considering the connection invalid, we need to inspect its properties.

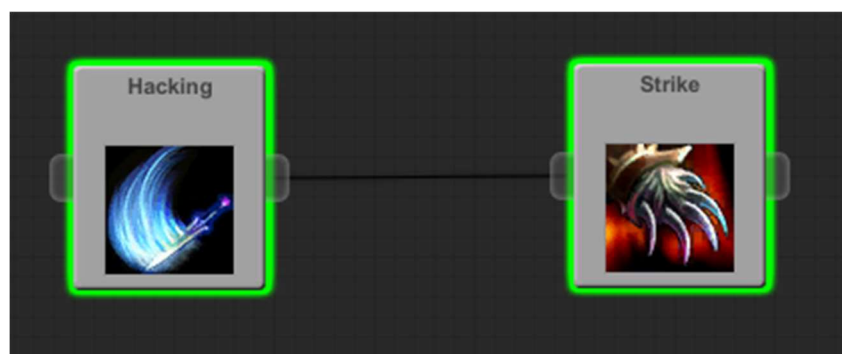
Click on the red line to select the connection and look in the custom properties inspector:



As you may notice an exclamation mark is catching our eye indicating no condition structure was found. Add one by pressing the '+' button above the warning.


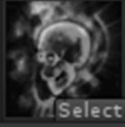
See the Connection Properties section below to have a good idea about the connection properties and how they are used.

When ready click somewhere within the canvas to deselect the connection and the line should be turning black to indicating a valid connection was setup.



Continue to add skills/talents to your liking, create connections between them until you are satisfied with you talent/skill tree.

## TALENT/SKILL PROPERTIES

Name :	Element Erosion
Tier :	Magic
Image :	 Select
Disabled Image :	 Select
Description :	Bend everything to you will.
Explanation :	You must own the EarthQuake skill.
Maximum Level :	3
Costs :	
+	
x 3	Default Cost
Description :	3 skill points needed for the base version of the skill.
x 4	Level 2 cost
Description :	4 extra skills will get your skill to level 2.
x 5	Level 3 cost
Description :	No one is save with this skill on its max level.

Every talent/skill needs a name, a default one is provided when the skill is created in the editor. I do advice to rename the talent properly, this can help when in need of some troubleshooting.

The Tier the talent does belong to, can be selected in the Tier property. How to setup Tiers is explained later in this manual (see Tier Management section).

You can assign an image and disabled image (**new in 1.0.1**) to your talent. Any 2D Texture can be used for the images. Note, none of those are required though, if they are provided they are used by the UI template.

As with everything in Talentus you can specify a description, this can also be used afterwards in the user interface, if needed.

You can add an optional Explanation to each skill **(new in 1.3.0)**. This field can be used to store some extra data (fi the requirements to obtain the skill).

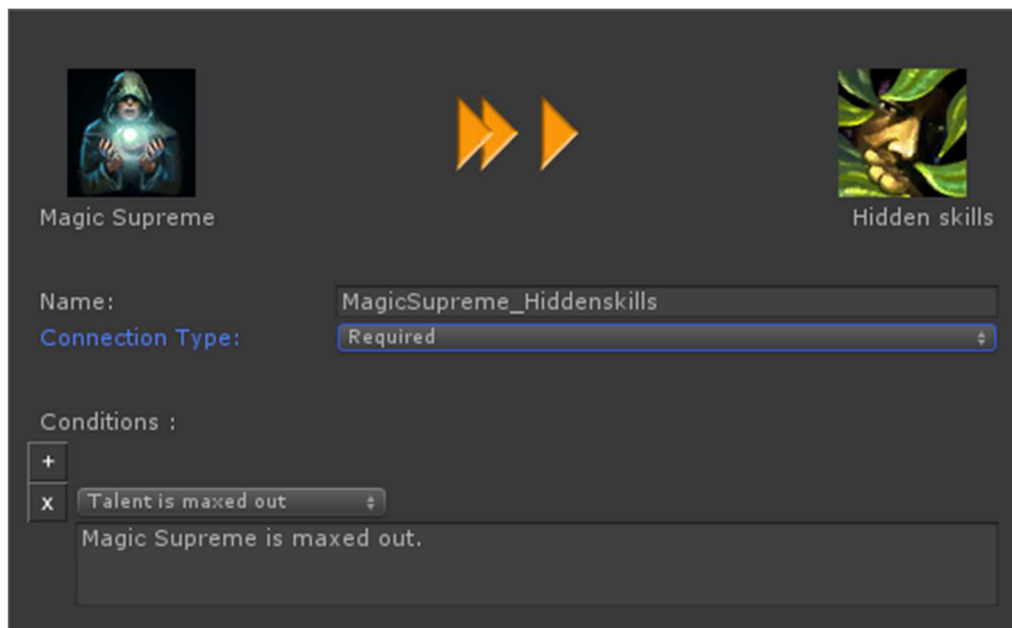
Every talent/skill can have multiple levels, each level does need to have a cost though (cost means skill points to unlock). By default a talent/skill needs one level to actually work, skills that doesn't comply with this rule will be ignored by the engine.

You can add a level by pressing the '+' button under the Costs label, a level/cost can be removed by pressing the 'x' button besides the cost. You can add a small description to every level/cost of the talent/skill.

Each level can also be described using the Description field **(new in 1.3.0)**. This field can however contain everything you want.

**(new in 1.3.0)** To change the skill explanation and/or the cost/level description at runtime you can have a look in the TalentusAutoEnableRoots example script. Same logic can be used to get those values at runtime.

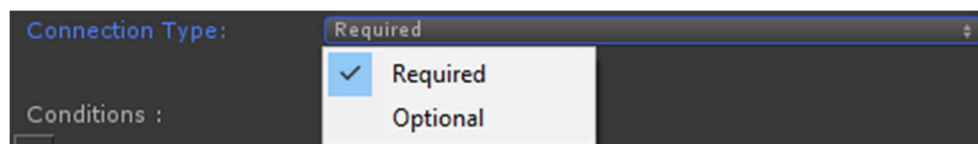
## CONNECTION PROPERTIES



By connecting the in- and outputs of a talent/skill node you have created a connection. To alter an existing connection simply left click on it and the properties will be displayed in the custom editor pane.

The name of the connection is automatically defined by the from and to talent of the connection. You can of course alter the name if needed. The connection name itself isn't used in any UI, only internally by the engine.

**(new in 1.1.0)** A connections can be required (default) or optional.



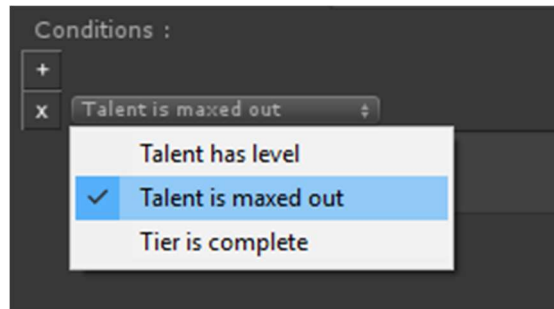
Note that when a talent is dependent on both required and optional connections at least one of the optional connections must be valid. The evaluator is using an algorithm of this format:  $r^1 \text{ AND } r^2 \text{ AND } R^n \text{ AND } (o^1 \text{ OR } o^2 \text{ OR } o^n)$  where  $r$  is a required and  $o$  and optional connection.

Note that the connection can be colored depending on his type by setting the colors in the Preferences window. You can also color the generated connections in the UI but more on that in the 'Get it to work at runtime' section.

A connection always needs one or more conditions. Those conditions does specify when the to skill/talent will be enabled. You can add a new condition by using the '+' button, or remove an existing condition by pressing the 'x' beside it.

All conditions must be met before the to talent/skill will be available for selection (and operation for the techies).

Currently we support 3 kind of conditions as listed below:

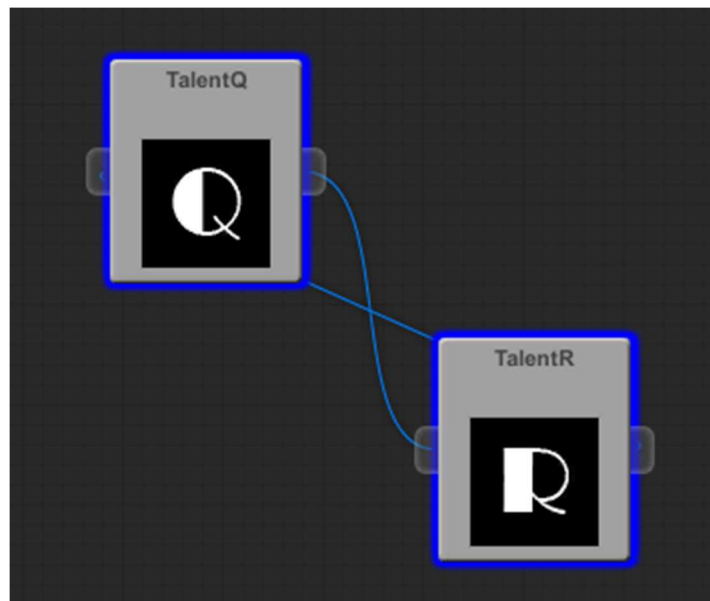


**Talent has level** allows you to set a specific level requirement for the from talent/skill. For instance Magic Supreme must be level 3.

**Talent is maxed out** means the from talent must have maximum level. For instance is Magic Supreme has 5 levels than all need to have been bought with skill points.

**Tier is complete** means all talents/skills belonging to the selected Tier must have been maxed out. In our we selected the Magic tier, so we must have bought all levels of all Magic talents before we can select the Hidden Skills talent.

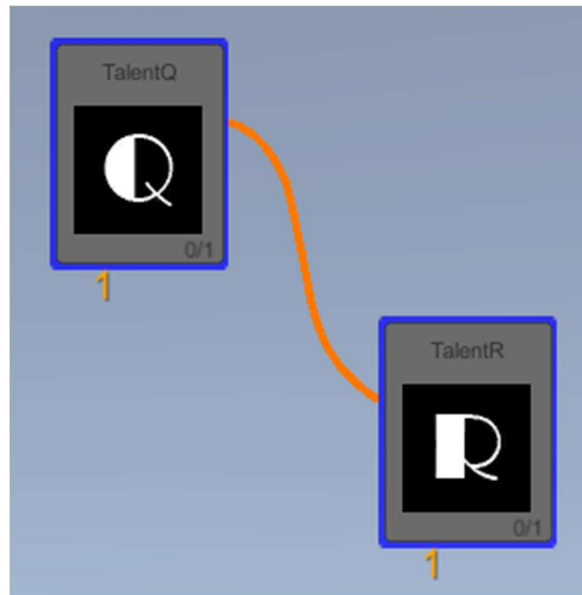
**(new in 1.1.0)** Circular Optional connections are created when 2 talents are connected with a connection between the in and out connector and again with another connection between the out and the in connectors.



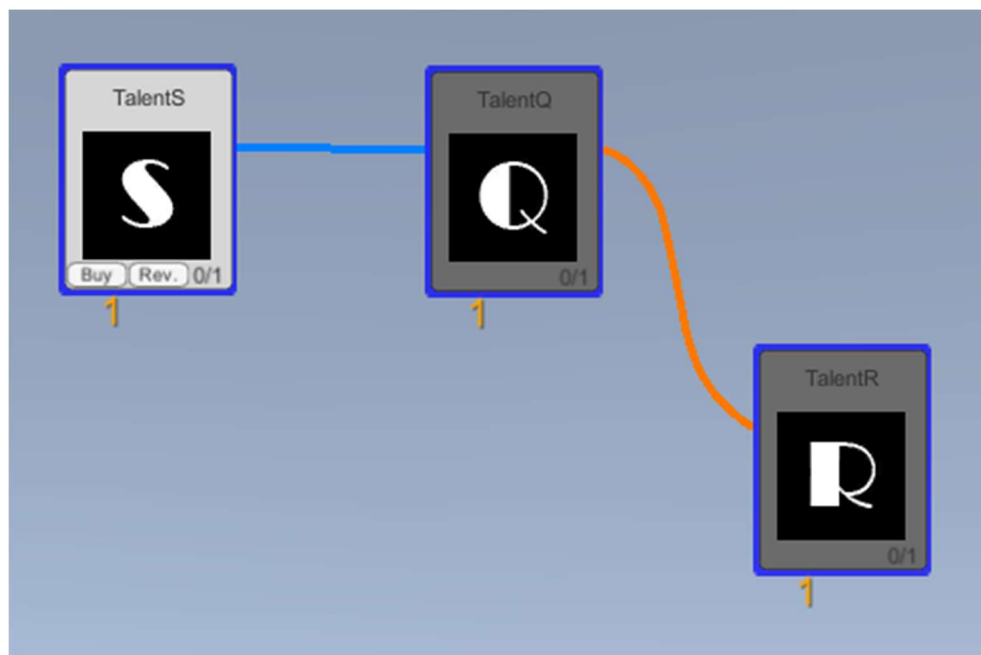
In such a case the TalentQ can be dependent on TalentR and the other way around. Note that Talentus is able to simplify this within the generated UI by only drawing one of both the connection so the UI doesn't get cluttered.

When asking the UI wizard to simplify those connection you will get this result:

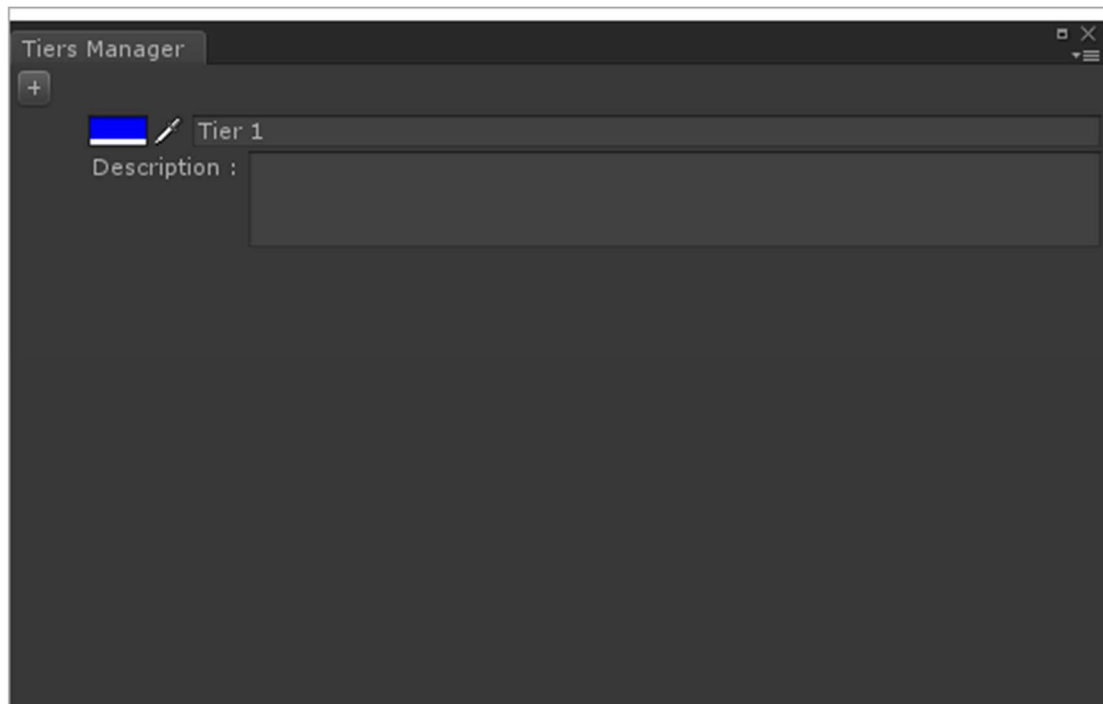




Note that because of the circular starting points those talents/skills can not be enabled so you need to make sure you have an entry point towards the circular connections to get it working. This connection should also be an optional connection otherwise it will not work.



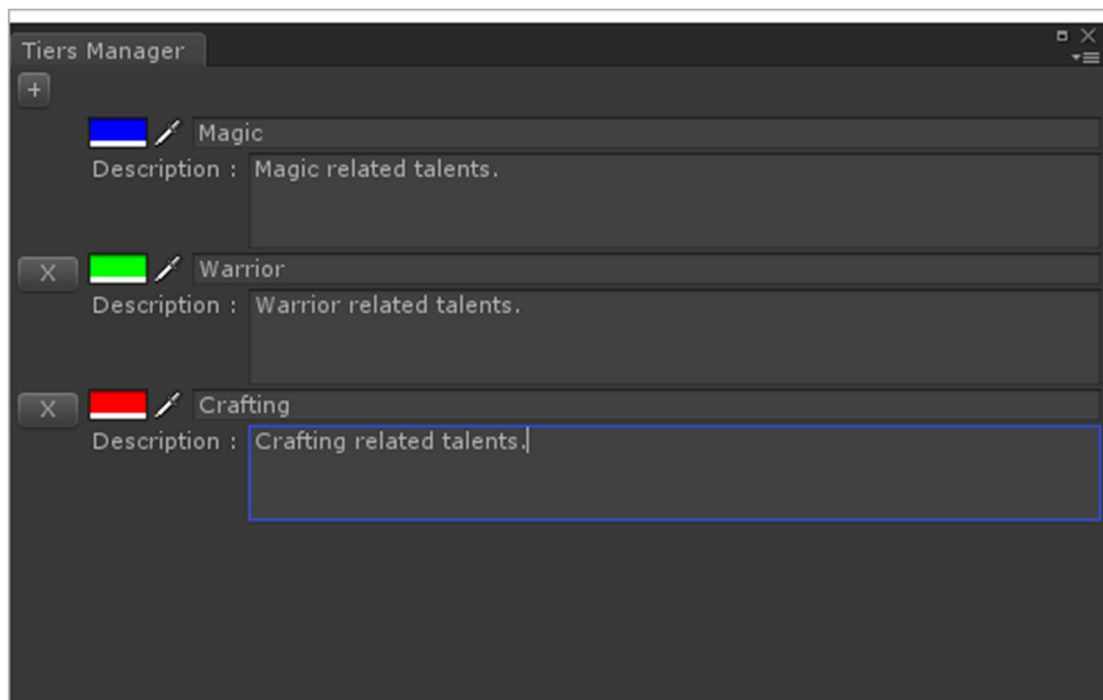
# TIER MANAGEMENT



By clicking the Tier manager icon in the editors toolbar the above window is opened.



The Tier manager allows you to categorize your talents/skills within one and the same talent/skill tree. For instance you can have Magic skills, Warrior skills and Master skills. Those master skills can only be unlocked when all the Magic or Warrior skills are already unlocked.

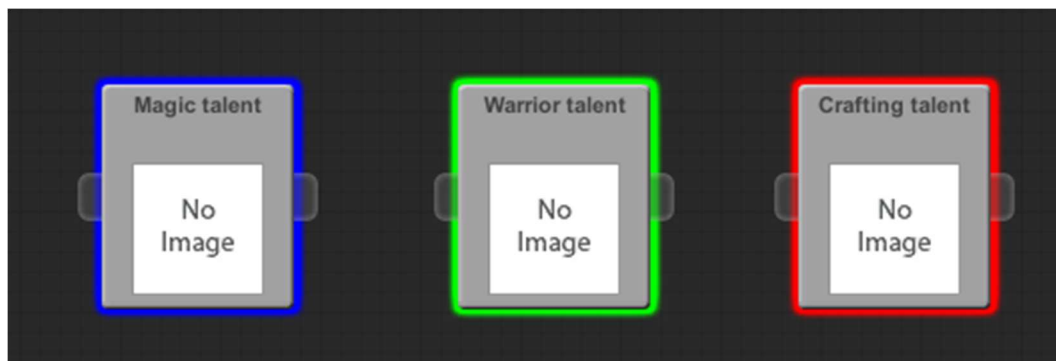


By clicking on the '+' button a new Tier can be added. The color picker will be defaulted to a non-existing color by default (only for the first 8 tiers). You can always change the assigned color by using the common color pickers.

You can remove a Tier by clicking the 'x' button besides the tier you want to remove. When you remove a tier all talents/skills belonging to it will be assigned the previous Tier in the manager list. For instance by removing the Crafting tier, the talents belonging to it will be assigned to the Warrior tier.

You can close this manager just by clicking the close button of the window.

Within the talent/skill properties you defined Tiers will be available within the Tier dropdown list, the specified Tier color will be drawn around the talents to indicate the Tier they belong to.



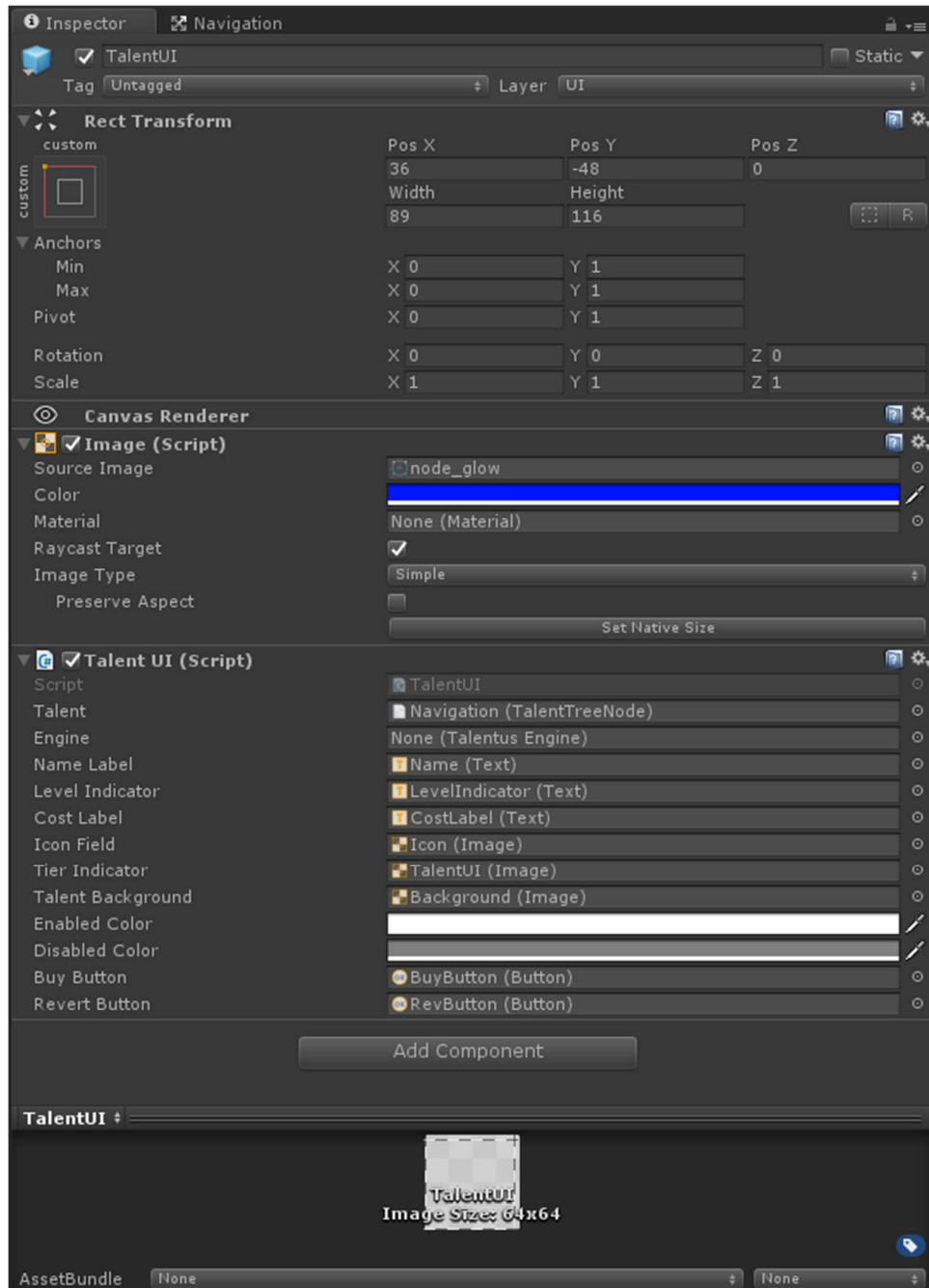
This coloring can also be used within the UI if you desire.

# UI TEMPLATE

Before we can use our talent/skill tree in the game at runtime we need at least a visual representation of a talent as prefab.

I provided an example of a prefab within the Example/UI folder called TalentUI. This example will represent the in editor look of a talent as much as possible. You are free of course to setup your prefab however you want.

Let me go over the example so you get a clear grasp what it stands for and how it will be used.



As you can see this is a simple Unity UI pane with a Talent UI script assigned to it. To investigate this further drag it into a scene under a canvas so it's easier to explore.

The prefab consists out of background and some sub components like labels, texts and 2 buttons.

The 2 buttons are here so you can buy and revert the talent in the example scene. You are free to hookup the talent/skill buying process however you like of course. You noticed all labels, buttons etc are referenced within the prefab except for the Engine. The engine field is assigned by the wizard with the actual talentus engine component.

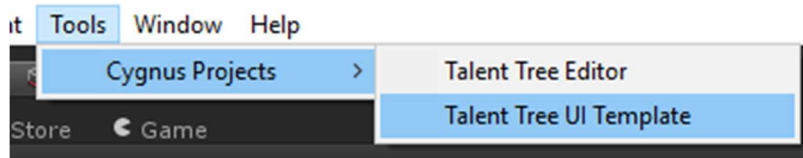
When you open up the source of the TalentUI.cs file you will notice nothing special is happening in there as everything is managed by the engine. That engine can be consulted though at any time to know the actual status of the talent/skill. This can be seen by the Talent.IsValid checks before enabling the buy button, also actual buying the skill is done by the engine by the BuyTalent call. More info on the API of the engine can be found in the Talentus Engine section.

I'm encourage you to create your own UI template so it matches your project perfectly. Every talent/skill in the tree will be represented by this prefab at runtime so it's an important part of your game visuals.

## GET IT TO WORK AT RUNTIME

For starters, open the Empty Example scene. In this scene a demo canvas without skills was prepared. The canvas exists out of an apply and revert button. A talentus engine game object is also already located in the scene.

Start the UI wizard by selecting Talent Tree UI Template from the Cygnus Projects menu.



A screen will appear where you need to fill in some references to other objects (scene and project).

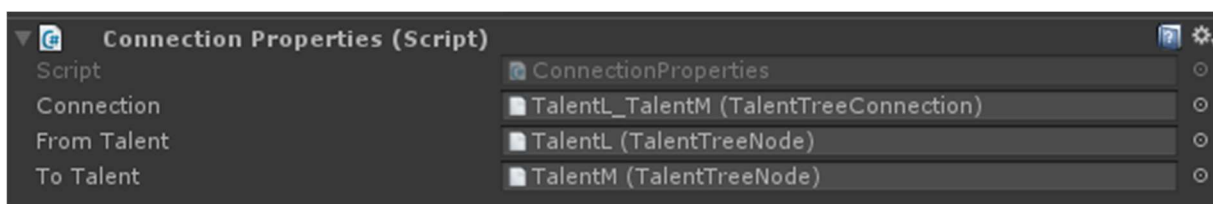
**(new in 1.1.0)** The screen has 3 different zones that can be extended or collapsed those are 'General Settings', 'Talent Settings' and 'Connection Settings'. By clicking on the label or expand/collapse indicator the settings will be made visible or invisible.

By default only the 'General Settings' will be visible.

Within the 'Talent Settings' only one field is available, this to assign a default MonoBehaviour script to each talent UI object created in the UI. This can be used to prevent you for having to assign a script manually to each Talent object. An example of a script is available in the examples folder called 'TalentExtraScript'.

Within the 'Connection Settings' there a lot more fields that can assigned, the combination of the field will make out how the Connection UI elements will appear in the UI.

**(new in 1.1.1)** When using custom prefabs there are no references created towards the talents/skills and connection (as this is very project specific), however by ticking the checkbox near 'Attach connection settings when using prefabs' each generated connection object will have a specific Connection properties component attached.



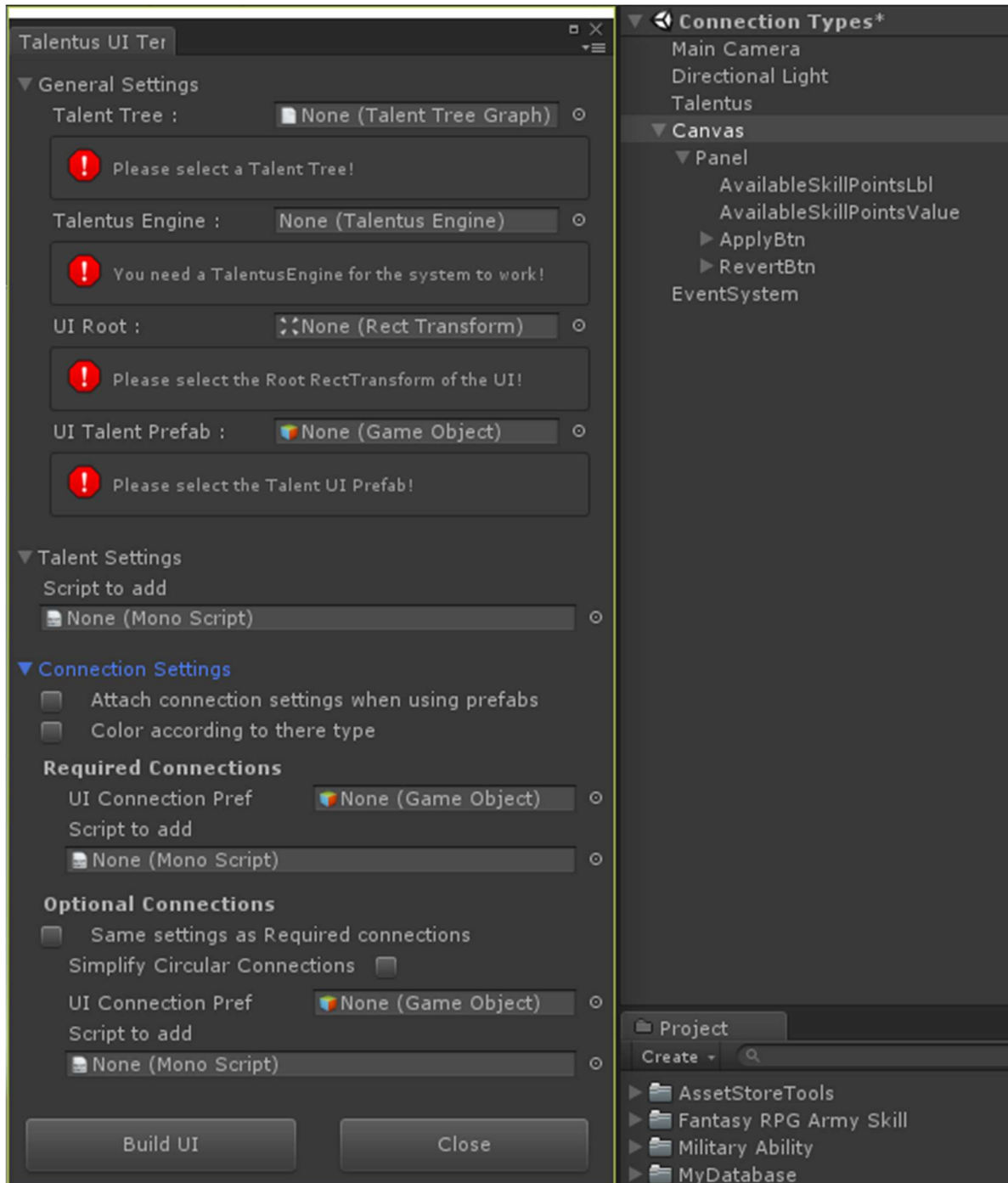
When checking the 'Color according to there type' the connection lines will be colored according to their type specified in the Preference settings. Not selecting this will default to the gray color as before.

Assigning a Connection prefab to the UI Connection Pref field will disable the rendering of the connection lines and thus overrule the coloring settings. You can assign UI prefab to this field, it will be located at the position when the connection should have it's starting point.

As within the talent settings you can assign a script to be added to the generated connection UI objects. You can still assign even when no prefab has been specified.

If you want you can specify a specific prefab and/or script for the optional connections, however if you check the 'Same settings as Required connections' those will not be used and inherit the behavior for the required connections.

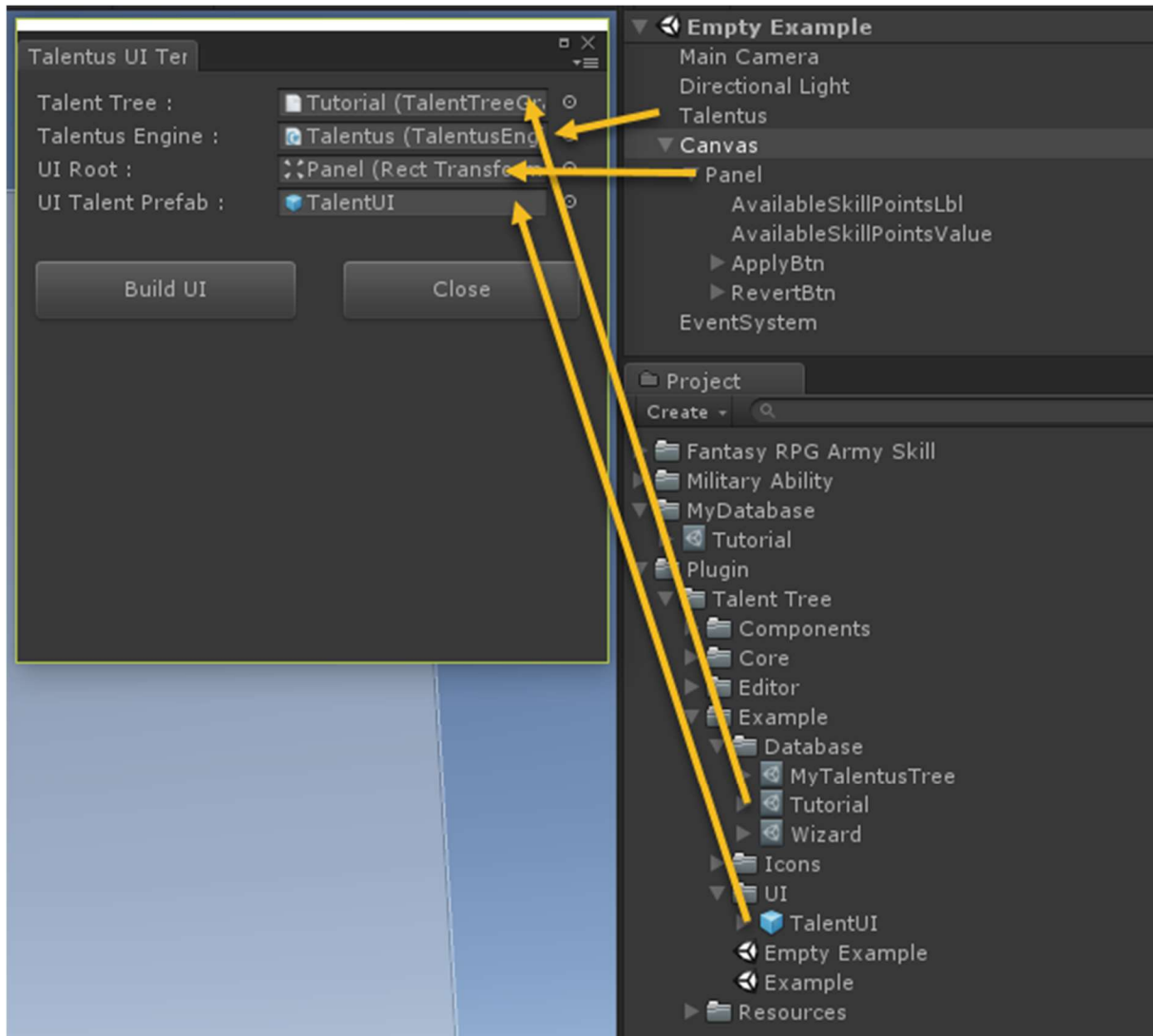
Within the optional connection settings you can select if the circular optional connections as explained in the Connection Properties section should be simplified or not. Note that these connection will get their special color assigned in case the 'Color according to their type' checkbox has been ticked.



You can always close this screen by pressing the 'Close' button.

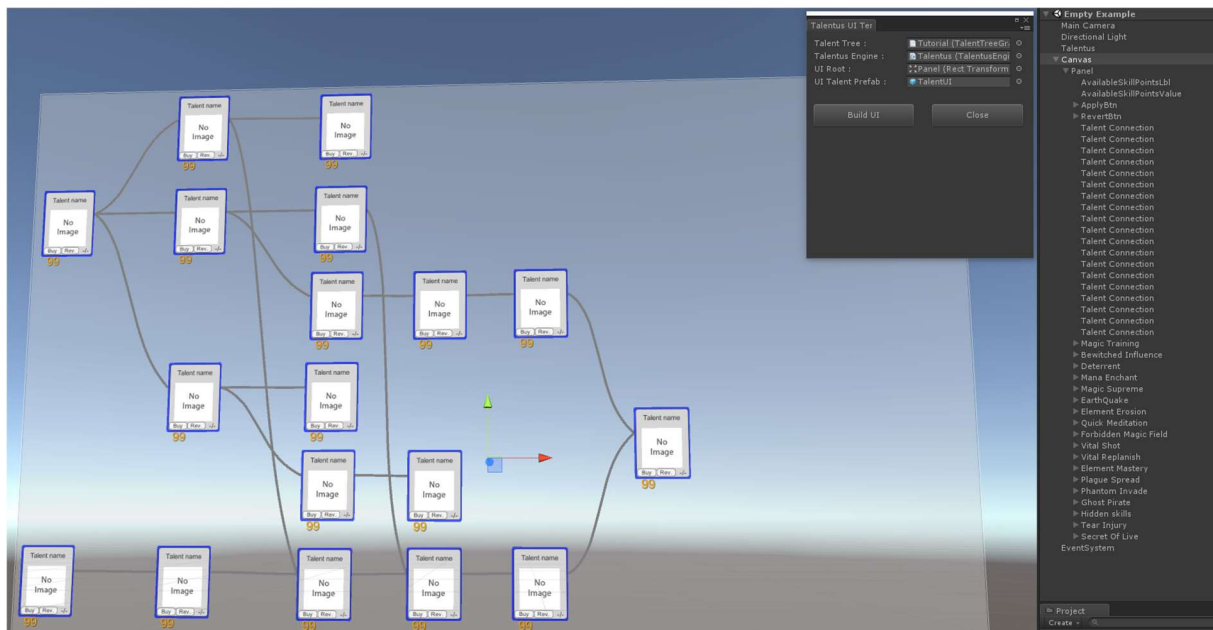
Let's assign the fields:

The Talentus Engine and the UI root should be already located in your scene, the Talent Tree and the UI Talent Prefab should be available within your project view.



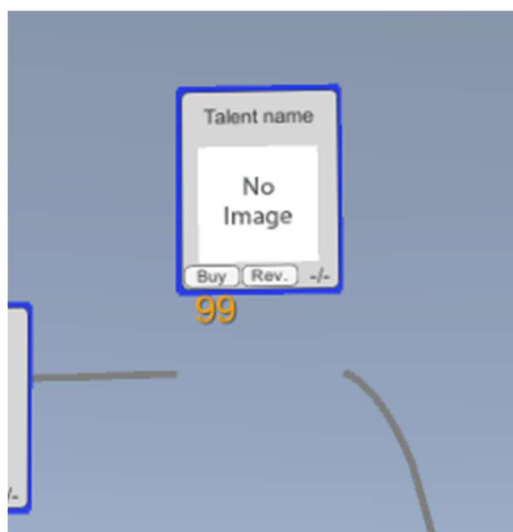
When done assigning press the 'Build UI' button, it shouldn't take long before your skill tree is represented in the scene as it was designed in the editor.





It's safe to close the wizard at this time.

Side note: when you rearrange the talents/skills the connections will not reposition as this is only the case when the UI is generated.



If you want to reposition the skill and maintain the connections visually you have to do this in the editor and regenerate the UI with the wizard.

However personally it's best to reposition and draw your UI using your art style, so no extra effort was made to have that visual link at any time. Note however that the talents/skills are linked at any time with the designed connections, rather at database level than visually. So everything will work as expected even with disconnected visual links.

# TALENTUS ENGINE

What is that TalentusEngine we seems to keep talking about and reference to? The Talentus engine is the component used to drive the Talent/Skill tree at design time. The engine has functionalities that allow to buy, revert and apply skills using skill points. It also provides the functionalities to save and load a state of a skill tree at runtime.

To use the engine the best approach would be to derive a class from the TalentusEngine and extend where needed. ***I provided a TalentusEngineExtended script that functions as an example for your custom integration.***

**(New in 1.2.0)** To demonstrate the respect of a talent/skill a new example called Respec Example was created. See the TalentRespecScript and TalentusEngineWithRespec to see how everything is tight together.

Let's delve in-depth into the API of the engine.

## Variables

The engine has only to public variables being a TalentTreeGraph called TalentTree that is used to assign your skill tree. This variable is used everywhere in the engine so without a tree the engine will do squad.

The AvailableSkillPoints is the number of skill points the user currently has to buy skills from the tree. This value should be set by your own custom code (note to save/load when needed).

## Methods

### public virtual void Start ()

The engine has a common Start method that can be overridden by your own class. Note you should still call the base class start to make things work properly.

### public virtual void Evaluate ()

Use this method to check if talents do have the correct state. It's recommended to call this method every time you did alter a talents/skill property from outside the engine.

### public virtual void Apply ()

When a player is buying a skill it is marked as bought but that can always be reverted until the Apply() method was called. This allows the player to check his options when assigning skill points in the tree. In this case he can always undo his selection. Apply will make the changes permanent. This means that for saving purposes the tree should have been applied before.

### **public virtual void Revert ()**

With the revert method the selections an end user did not apply are all undone. This will bring the skill tree back in the state before the end user was playing around with the available skill points.

### **public virtual void BuyTalent(TalentTreeNodeBase talent)**

BuyTalent is used to mark a specific talent/skill as bought. Note you need to call the trees Apply before the change is permanent. You'll need to pass in the talent/skill you want to buy. Note when a talent has multiple levels/costs the system will handle this for you.

After every call to this method the tree will Evaluate itself.

### **public bool CanBeUnbought(TalentTreeNodeBase talent) (new in 1.2.0)**

Perform a check on the provided talent to see if it would be ok to rollback the buy and apply operation without breaking the tree validation.

Returns true in case it would be save to respect the talent, otherwise the method will return false.

### **public TalentTreeNodeLevel UnBuy(TalentTreeNodeBase talent) (new in 1.2.0)**

Unbuy the specified talent, make sure the check if this doesn't break the tree by calling CanBeUnbought().

Returns a class containing the level and cost of the unbought talent.

### **public virtual void RevertTalent(TalentTreeNodeBase talent)**

RevertTalent does undo the state of marking a skill as bought before the Apply. You'll need to pass in the talent/skill you want to buy. Note when a talent has multiple levels/costs the system will handle this for you.

After every call to this method the tree will Evaluate itself.

### **public string SaveToString()**

SaveToString does return a string that has a predefined format which contains the current state of the talent tree (note the status are only fixed after a call to Apply(), non-applied changes will not be saved into this string)

Use this string to save the skill tree together with you save file from the game (functionality you should be implementing as Talentus isn't a game save/load asset).

I provided an example for saving/loading to a simple text file in the TalentusTreeExtended.cs file.

### **public void LoadFromString(string statuses)**

LoadFromString() does set the state of the talent tree according to the settings provided by the statuses string. Note that this string must be in a predefined format generated by the SaveToString() method. Any attempt to alter the parameter value can break the skill tree.

Use this LoadFromString method to load the skill tree together with your load game functionality (functionality you should be implementing as Talentus isn't a game save/load asset).

I provided an example for saving/loading to a simple text file in the TalentusTreeExtended.cs file.

## Events (1.0.2 or higher)

### (TalentTree) TalentSelected(object sender, TalentTreeGraph.TalentSelectedEventArgs e)

The talent (argument of parameter e) was select, meaning it was bought but not confirmed (applied) yet. The cost will be reduced from the available skill points.

### (TalentTree) TalentReverted(object sender, TalentTreeGraph.TalentRevertedEventArgs e)

The talent (argument of parameter e) has be reverted, meaning the selected operation was cancelled and the cost will be returned to the available skill points.

### (TalentTree) TalentBought(object sender, TalentTreeGraph.TalentBoughtEventArgs e)

The talent (argument of parameter e) has been applied and can't be reverted, no action is happening with the available skill points at this time.

### (TalentTree) TalentTree\_TreeEvaluated(object sender, TalentTreeGraph.TreeEvaluatedEventArgs e)

**(new in 1.1.0)** The tree has been evaluated, this event can be used to redraw the state of the skills in the tree. See TalentUI for an example.

### (TalentTree) TalentTree\_TalentUnBought(object sender, TalentTreeGraph.TalentUnBoughtEventArgs e)

**(new in 1.2.0)** A talent was unbought, this means the talent was rolled back because of a respect request. See TalentEngineWithRespec for an example.

## FAQ

### What version am I currently running?

Press the gear icon in the talentus editor.

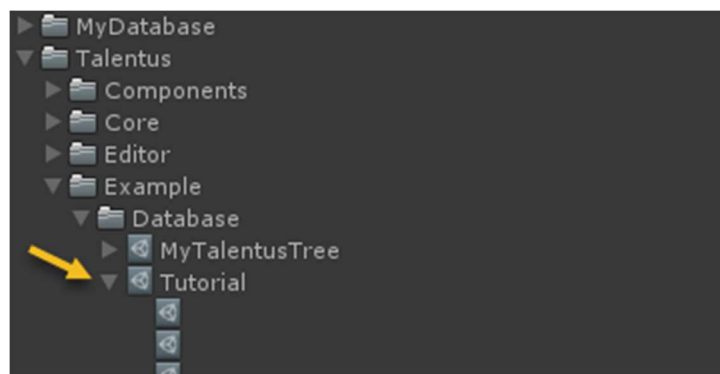


A dialog box will be displayed indicating the version you are currently running.

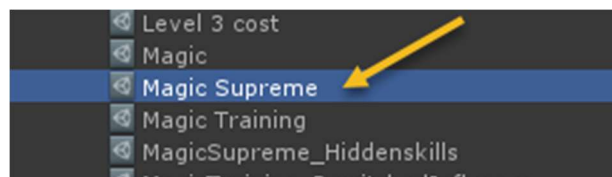
## TROUBLESHOOTING

### A talent seems to be missing in the editor, how to get it back ?

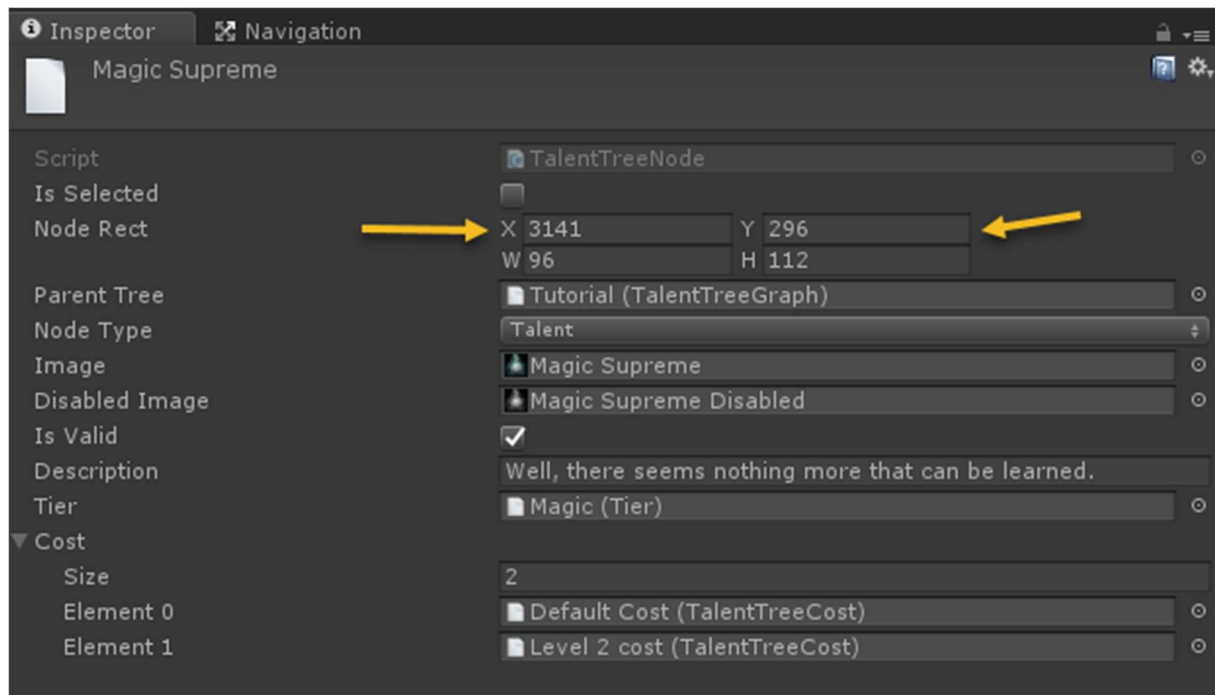
Find your talent tree asset in your project window and expand it.



In the expanded list find the talent/skill that's seems to be missing and click on it. If you named your talents you'll definitely find your talent/skill right away.



When you look at the unity inspector you'll see some strange values for the Node Rect X and/or Y values.



Changing those both to 100 will place the talent/skill in the top left position within the editor.

### **When I run my game the skill tree is visible but the root node(s) are disabled.**

The reason for this behavior is a mismatch between the visible talent tree and the one assigned in the talentus engine component in your scene.

Make sure to assign the correct talent tree to the engine, the same that was used for generating the UI.

### **When I run my game the skill tree doesn't display the images nor the buttons. It doesn't look like the tree is disabled though.**

1. Move your initializations and event subscriptions to the OnEnable instead of the Start method.
2. Try to regenerate the tree using the UI wizard as this can occur when the underlying scriptable objects don't match the signature of those items used when generating the UI. If you change the code of the talentbasenode class.