Skill Tree

Skill Tree Documentation

⚠ Due to many requests, Skill Tree's API was changed significantly in v1.2.0. Updating to this version may result in data loss. Always make a backup of your project before updating.

Welcome to Skill Tree's documentation! Choose a link below to get started.

Quick Start

Installation

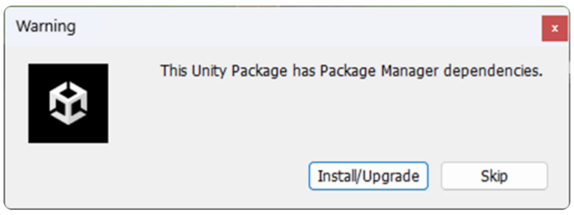
How to install Skill Tree in your Unity project.

Asset Store

You can find the latest version of Skill Tree in the asset store with this link.

Installation Steps:

1. Get Skill Tree from the asset store.
2. Open your Unity project, go into `Window > Package Manager`, switch to `My Assets` from the packages dropdown at the top-left, and then type `Skill Tree` in the search bar.
3. Download and import the package. You may be prompted to install package manager dependencies. If so, click `Install/Upgrade`.


Warning

This Unity Package has Package Manager dependencies.

Default Prefabs

How to use the default prefabs.

Skill Tree provides default UI for those that would rather not code their own. You can find them all in `Assets > StylishEsper > Skill Tree > Prefabs` folder.

Initializer

Drag and drop the `Initializer.prefab` (or `SkillTreeInitializer.prefab`) onto your scene. You can read about the Skill Tree Initializer component here.

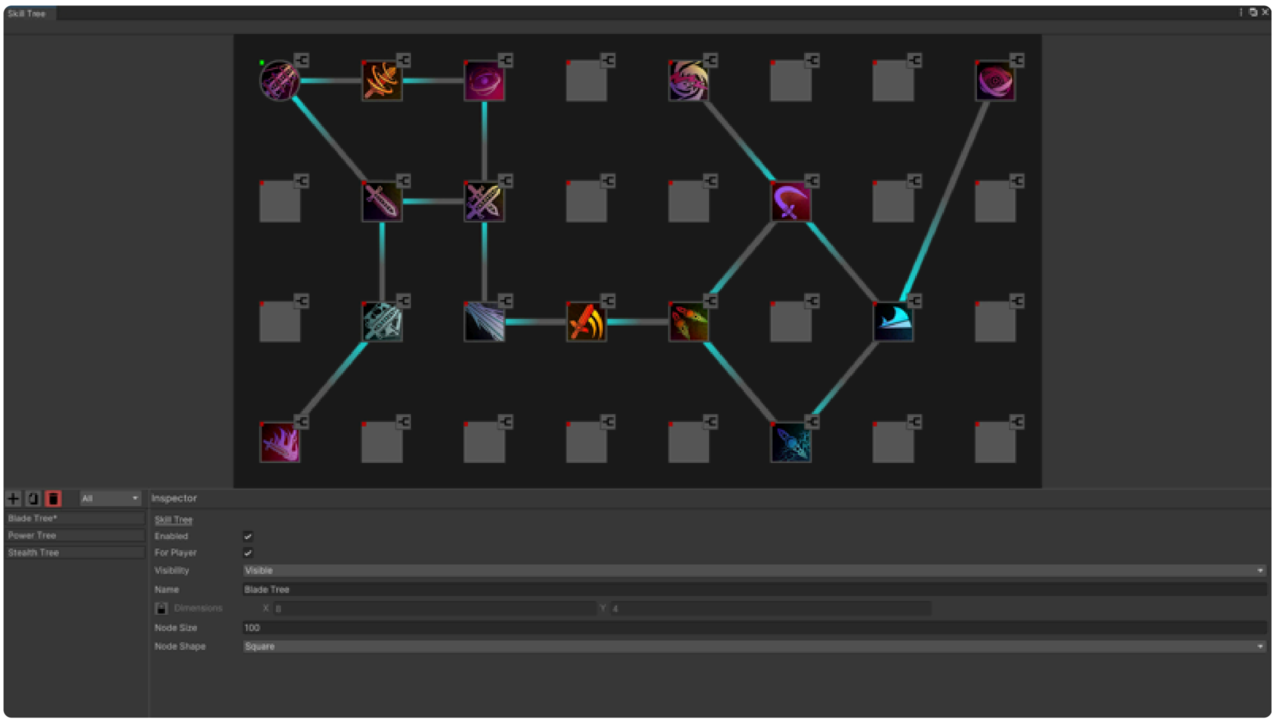
Default UI

Open the `DefaultUI` folder and drag and drop `SkillTreeWindow.prefab`, `SkillBar.prefab`, `SkillDetails.prefab`, `SkillMenu.prefab`, `DefaultInputHandler.prefab`, and `ConfirmationPrompt.prefab` onto your scene. Make sure to set an audio source for SkillMenu and SkillTreeWindow from the inspector. You can read more about each here.

Start Editing

Start creating your own skill trees.

Skill Trees



Enabled ✓
For Player ✓
Visibility: Visible
Name: Blade Tree
Dimensions X 8 Y 4
Node Size: 100
Node Shape: Square

Skill Tree editor window

Skill trees (or skill graphs) can be created easily in the Skill Tree editor window. You can access it from `Window > Skill Tree > Skill Tree Editor`.

Stats

You can create custom stats from `Window > Skill Tree > Stats`.

Making it Work

Components

Give the Skill Interpreter component to your player character to do something when a certain skill is used.

Give the Unit Stats Provider component to your player character to give them stats.

Summary of The Order

The order to get your skills working is quite simple.

1. Drag the Default Prefabs onto your scene.
2. Create your skills in the Editor Window.
3. Give the Skill Interpreter and Unit Stats Provider components to your player character and set them up.

You do have to code your own skills. Remember, Skill Tree is not a skill creator and doesn't advertise itself as one. If you're new to programming, try checking out the demo scene in the `Assets/StylishEsper/SkillTree/Example` folder.

Scripting

Remember to use the namespace `Esper.SkillTree`.

Initialization

Use `SkillTree.Initialize()` to initialize Skill Tree. This is required to get it working properly. You can use the Skill Tree Initializer component.

It may be required to call `SkillTree.ClearData()` when Skill Tree's functionality is no longer needed. The initializer component calls this automatically as necessary.

Handling Level Up

Whenever your player levels up, call `SkillTree.SetPlayerLevel()`, as this will let Skill Tree know what level the player currently is.

Setting Skill Points

You can set the player's current skill points with `SkillTree.playerSkillPoints`.

Completing Skill Use

Skill Tree prevents the usage of skills by a character if they are already in the middle of using one. Call `SkillTree.CompleteSkillUse` to let Skill Tree know that the character should now be free to use another skill.

If you have any issues, please see the Getting Help page.

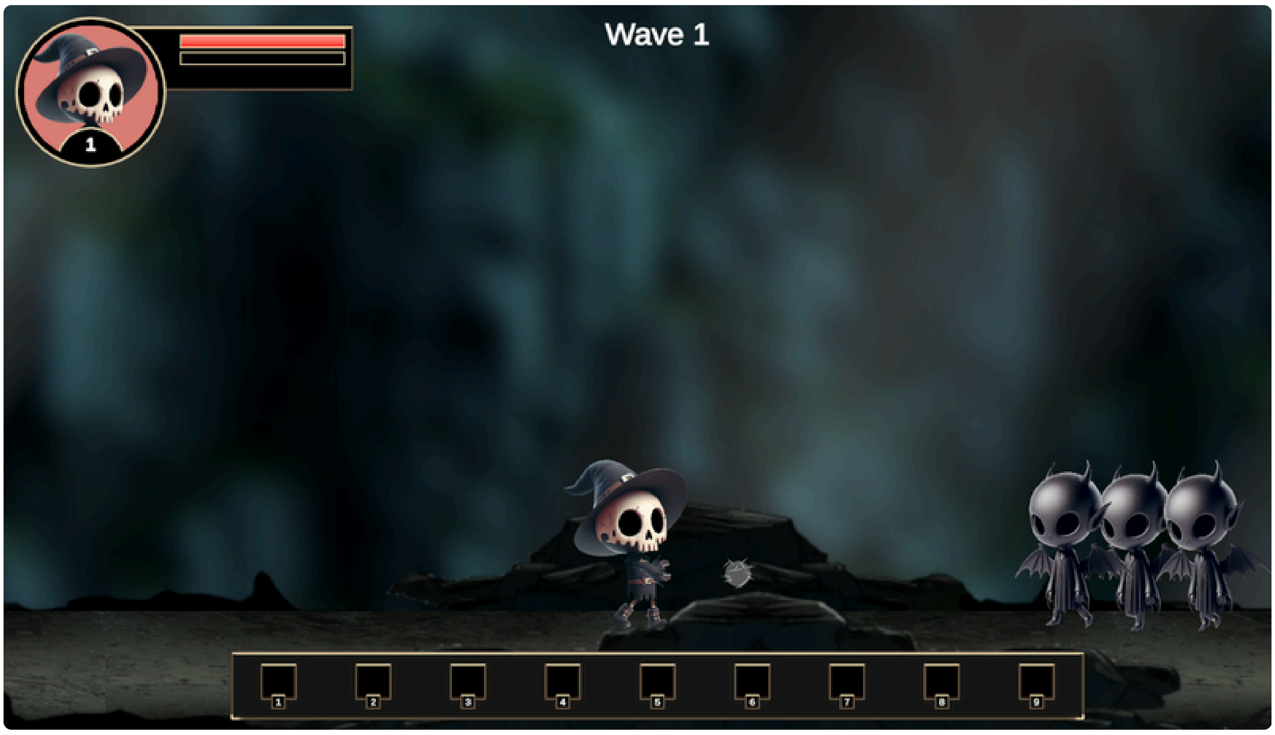
Demo Scene

Endless Death

A quick guide to running the scene.

Endless Death is a demo game created to showcase Skill Tree's potential. It uses all of Skill Tree's UI, and has 5 functional skills.

Running the Demo



Demo game

You can find the demo scene in the `Assets/StylishEsper/SkillTree/Example` folder. It should work as it is if its skill trees were not deleted from the Editor Window. In the case that they were, you can find the backups in the `Backup` folder. Simply copy the 2 folders inside the backup folder and paste them in the `Assets/StylishEsper/SkillTree/Resources/SkillTree` folder (replace the existing ones).

The scriptable objects store data of a single skill tree. They must be in the `Resources/SkillGraphs` folder to be accessible by Skill Tree. When you create a skill tree with the editor window, they are automatically placed in that folder.

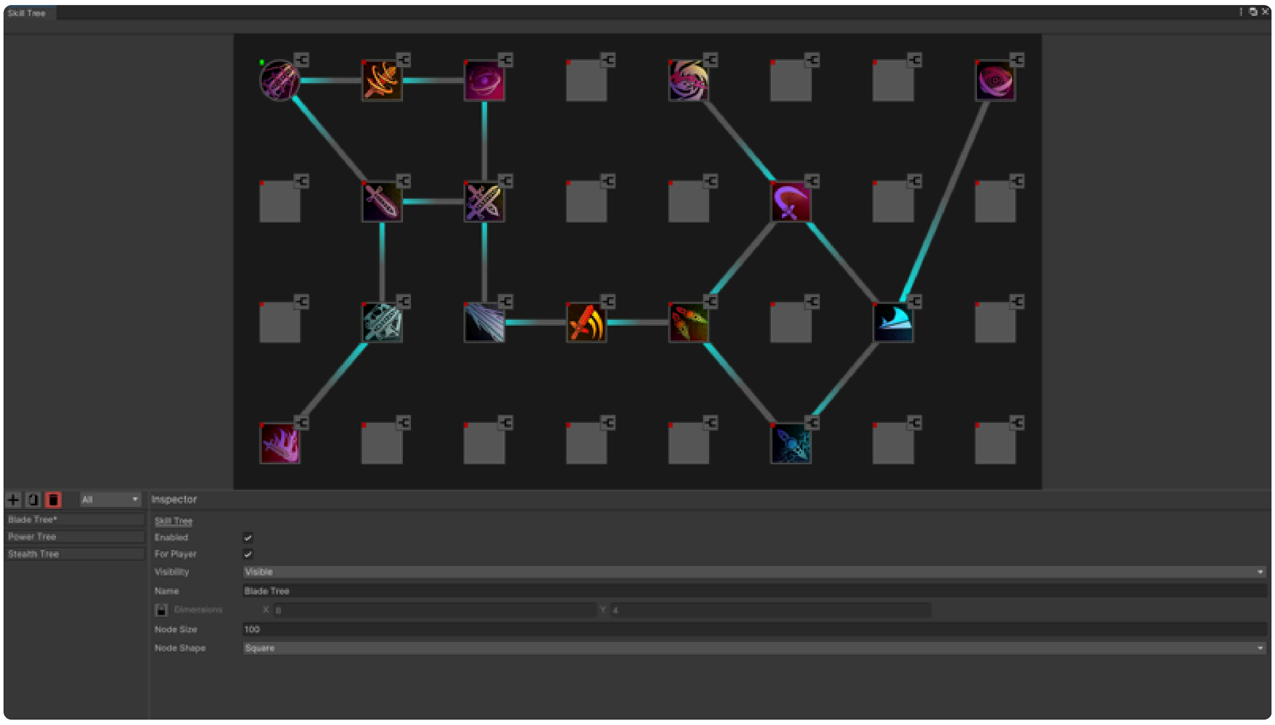
Systems

Skill Tree

The skill side of Skill Tree.

Editor Window

Going over the Skill Tree editor window.



Skill Tree

Enabled	✓
For Player	✓
Visibility	Visible
Name	Blade Tree
Dimensions	X 8 Y 4
Node Size	100
Node Shape	Square

Skill Tree editor window

Skill Trees can be created easily in the Skill Tree editor window. You can access it from `Window > Skill Tree > Skill Tree Editor`.

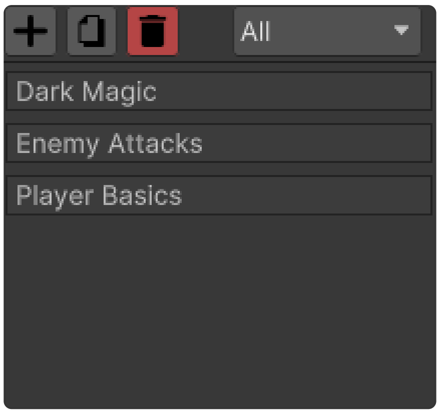
Shortcut Keys

Reset Split Views: CTRL + R

Cancel Connection Creation: ESC

Skill Trees (Skill Graphs)

Working with skill trees in the editor window.



Tree section

The bottom-left section of the window contains a list of all your skill trees. You can create, duplicate and delete trees.

Clicking on a tree will display its Skills (Skill Nodes) on the upper section and the tree properties will be displayed in the inspector section.

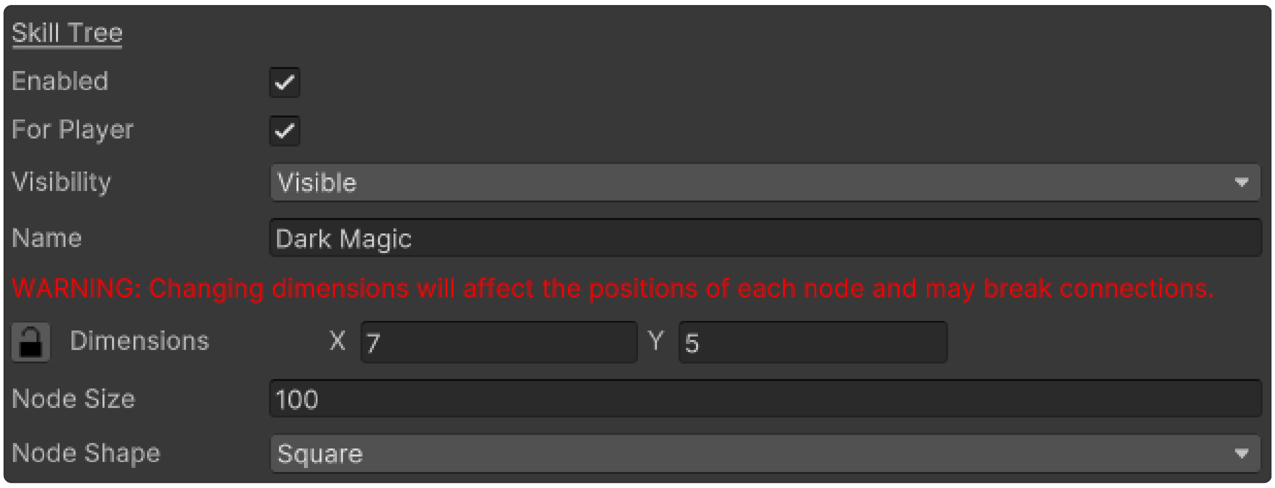
Tree Visibility

The dropdown allows you to change how the skill tree is viewed.

All: All nodes are visible.

Complete: Only completed nodes are visible. Incomplete nodes will not be displayed in-game.

Properties


Skill Tree

Enabled ✓

For Player ✓

Visibility: Visible

Name: Dark Magic

WARNING: Changing dimensions will affect the positions of each node and may break connections.

Dimensions X 7 Y 5

Node Size: 100

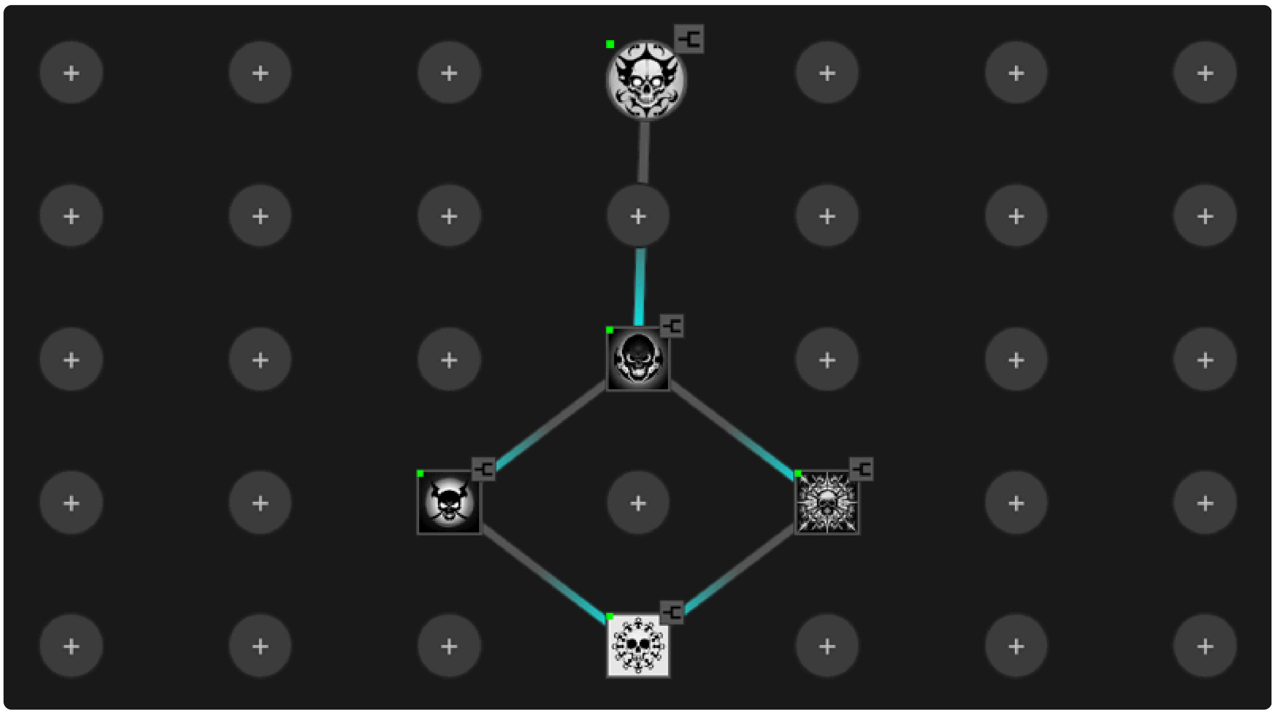
Node Shape: Square

Inspector properties

Name	Description
Enabled	The enabled status of the graph. Disabled graphs will not be available in-game.
For Player	If this graph is for the player.
Visibility	The graph's visibility in-game. If set to hidden, the skill graph will not be displayed in the default skill tree window.
Name	The display name of the graph.
Dimensions	The graph dimensions. This will set how many skills the graph can have. This value can be edited freely, but will affect the visual position of each skill, break connections, and/or hide skills.
Node Size	The sizes of each skill node.
Node Shape	The shape of each skill node.
Custom Object	An object of any type that you can use to connect your own data to the skill tree.

Skills (Skill Nodes)

Working with skills in the editor window.


Graph view section

Adding Skills

The plus buttons you see on the upper section of the editor window will create a skill (skill node) in that position.

Deleting Skills

To delete a skill, click on it, scroll down completely in the inspector section, and click the `Delete Node` button.

Completion Status

The green or red light in the top left indicate completion status.

Red: The skill details are incomplete. The skill will not be visible in-game.

Green: The skill details are complete. The skill will be visible in-game.

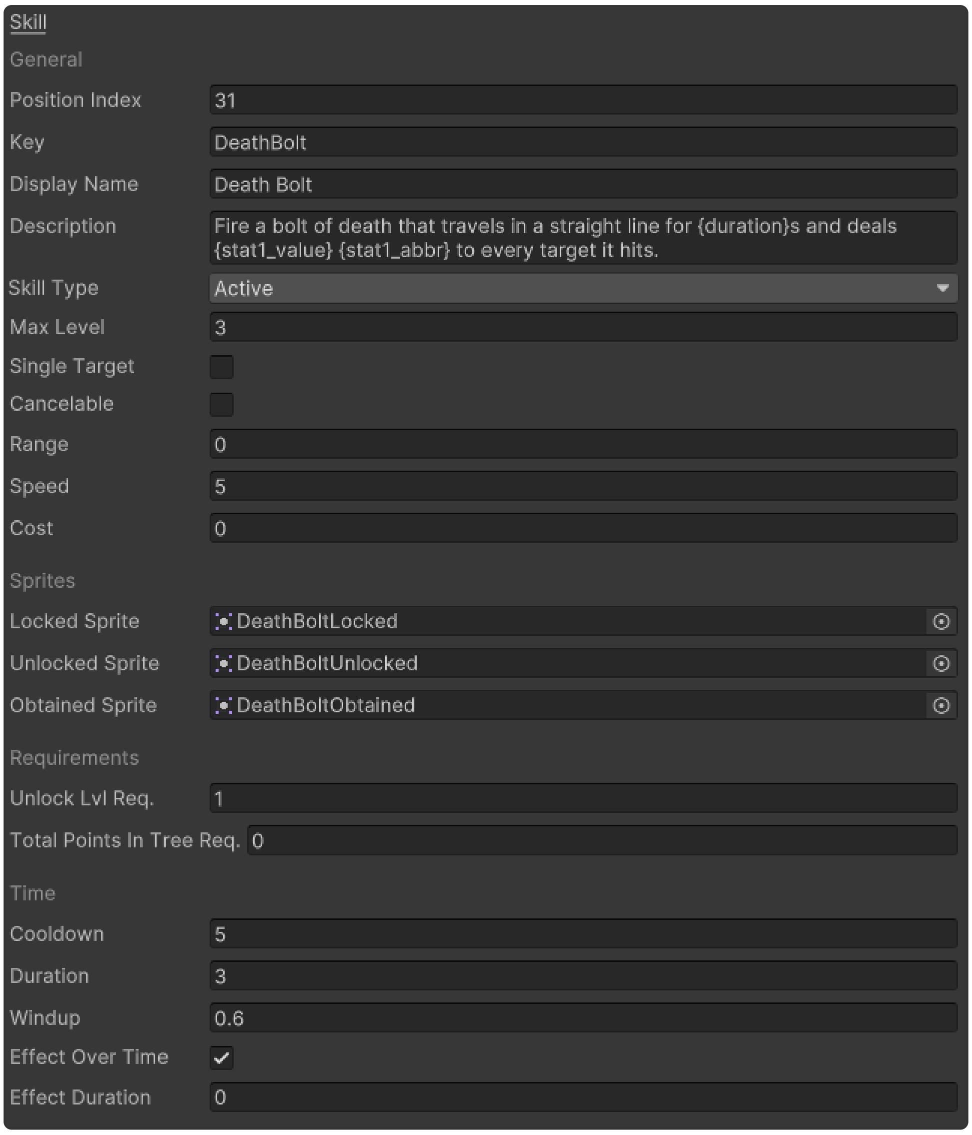
Connections

The tiny button on the top right will start a connection creation process from the skill. Clicking on another skill will create a connection between them.

Properties

Clicking on a skill will display its properties in the inspector section.

Skills contain data that can be used freely in any way you want. Most properties don't do anything automatically, while some do. Of course, the default UI would need to be used for automatic functionality. Remember, Skill Tree is not a skill creator.



Skill

General

Position Index	31
Key	DeathBolt
Display Name	Death Bolt
Description	Fire a bolt of death that travels in a straight line for {duration}s and deals {stat1_value} {stat1_abbr} to every target it hits.
Skill Type	Active
Max Level	3
Single Target	☐
Cancelable	☐
Range	0
Speed	5
Cost	0

Sprites

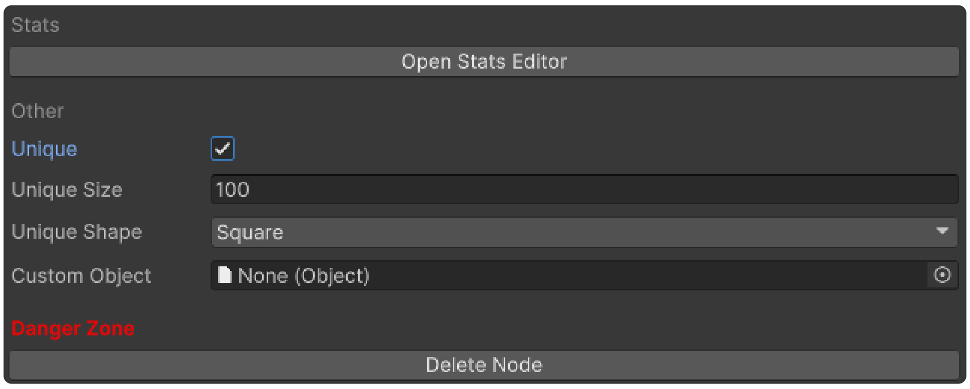
Locked Sprite	DeathBoltLocked
Unlocked Sprite	DeathBoltUnlocked
Obtained Sprite	DeathBoltObtained

Requirements

Unlock Lvl Req.	1
Total Points In Tree Req.	0

Time

Cooldown	5
Duration	3
Windup	0.6
Effect Over Time	✓
Effect Duration	0



Stats

Other

Unique	✓
Unique Size	100
Unique Shape	Square
Custom Object	None (Object)

Danger Zone

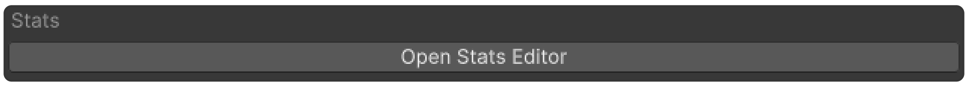
Name	Description
Position Index	The value that sets the position of the skill in the graph. The top-left most position is 0, and the bottom-right most position is the max value.

Key	The key of the skill. This is for retrieving the skill through code.
Display Name	The name of the skill as it will be displayed in-game.
Description	The description of the skill. This supports automatic interpolation with the usage of Text Tags.
Skill Type	The type of the skill. Skill types can be created from Skill Tree's settings.
Max Level	The max level of the skill.
Single Target	If this skill is single target.
Cancelable	If this skill can be canceled.
Range	The range of the skill.
Speed	The speed of the skill.
Cost	The (mana, energy, etc.) cost of the skill.
Locked Sprite	The sprite that will be displayed when the skill is in the locked state.
Unlocked Sprite	The sprite that will be displayed when the skill is in the unlocked state.
Obtained Sprite	The sprite that will be displayed when the skill is obtained.
Unlock Level Requirement	The player level required to unlock the skill.
Total Points in Tree Requirements	The total points in the skill tree required to unlock the skill.
Cooldown	The skill cooldown duration.
Duration	The duration of the skill.
Windup	The time it takes to windup the skill.
Effect Over Time	If this skill has an over time effect.

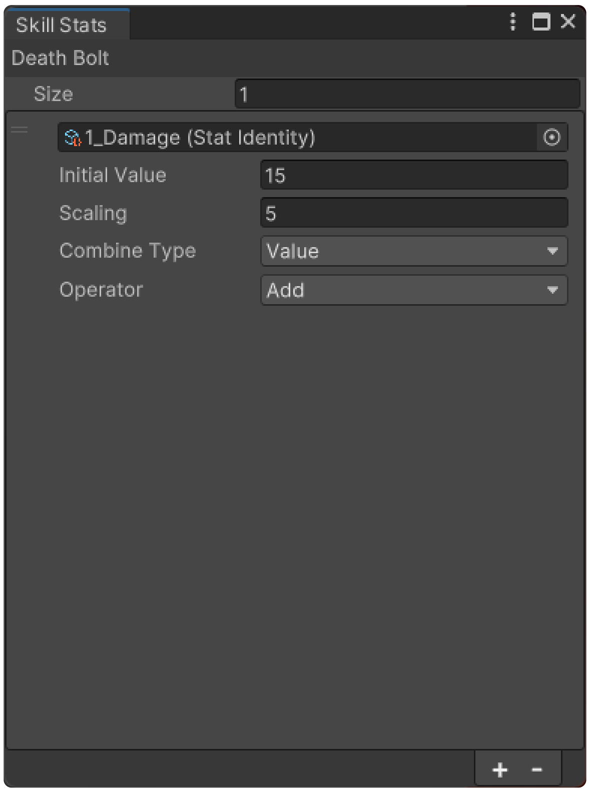
Effect Duration	The duration of the over time effect.
Unique	If this skill is unique from the others.
Unique Size	The unique size of this skill.
Unique Shape	The unique shape of this skill.
Custom Object	An object of any type that you can use to connect your own data to the skill node.

Stats

Use the 'Open Stats Editor' button to open the skill stats editor.



Skill Stats Editor



The skill stats editor simply contains a list of stats for a single skill. The list can be reordered. If this window is opened, clicking on a different skill in the Skill Tree editor window will automatically show the details of that skill.

You can add/remove stats with the plus/minus buttons at the bottom.

Properties

Name	Description
Stat Identity	The identity of the stat.
Initial Value	The initial (base) value of the stat.
Scaling	The value the stat increases by per level.
Combine Type	How this stat alters another when combined.

Operator	The combine operator used when combined with another stat.

Text Tags

Accurately display skill data.

If you'd like the skill description to automatically display the correct skill value (such as stats) as the skills are upgraded, use text tags. Only the skill description has tag interpolation.

You can interpolate any text through scripting with `SkillNode.GetInterpolatedText`.

Supported Tags

Tags are encapsulated in braces `{}` like so: `{tag}`. You can see the full list of possible tags and what property they display in the table below.

Tag	Skill Property
`{display_name}`	Display Name
`{cooldown}`	Cooldown
`{duration}`	Duration
`{windup}`	Windup
`{effect_duration}`	Effect Duration
`{range}`	Range
`{speed}`	Speed

Tags for Stats

Stat tags work slightly differently. Consider the example below:

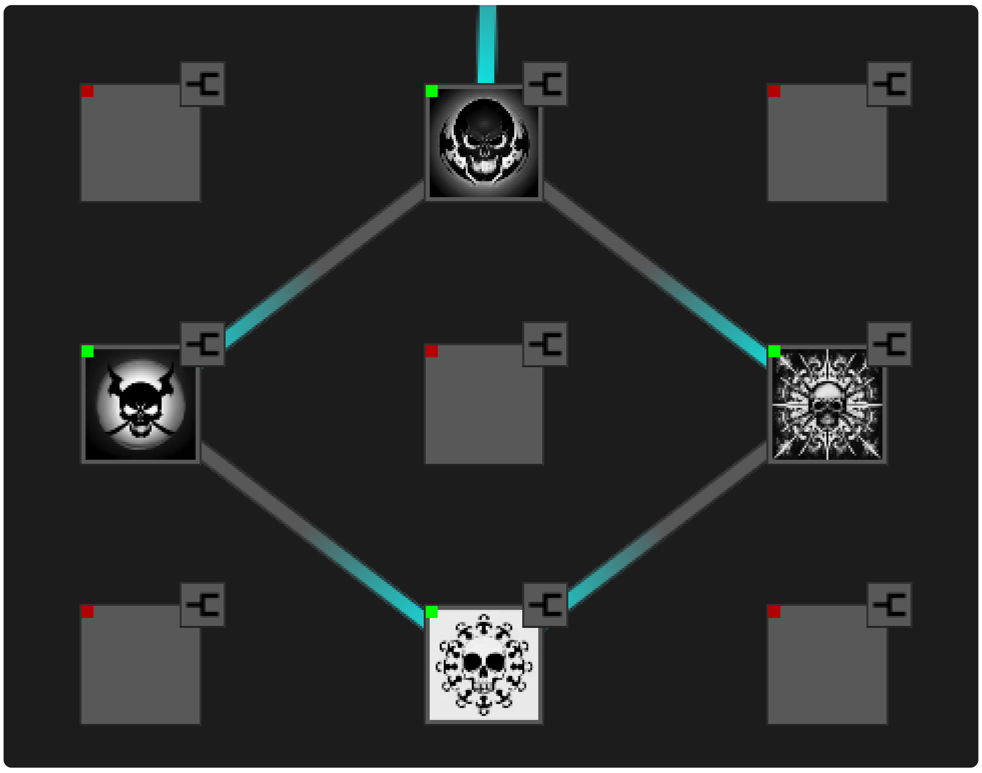
```
{stat1_value}
```

This will display the base value of the first stat. You can see the full list of possible stat tags below. Replace '[index]' with the number of the stat.

Tag	Description
`{stat[index]_value}`	The current value of the stat relative to the skill's level.
`{stat[index]_name}`	The full name of the stat.
`{stat[index]_abbr}`	The abbreviation of the stat name.

Connections

Working with connections in the editor window.



The blue and gray lines are skill connections. The starting point of the connection is the blue side, and the end is the gray side. If a connection is completely blue, this means the connection goes both ways.

Purpose

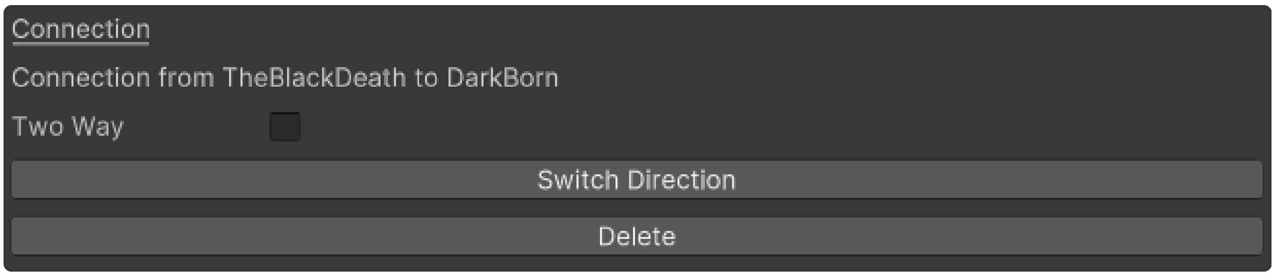
Connections are for skill progression purposes. It shows the player what path will be taken if a certain skill is obtained. Once a skill is obtained, Skill Tree will unlock other skills through connections. However, connections are not the only unlock requirement, so despite obtaining a skill, sometimes a connected skill may not be unlocked if the player doesn't meet the required level or not enough points have been spent in the skill tree.

Creating Connections

The tiny button on the top right of a skill node will start a connection creation process starting from itself. As of now, there's no visual indicator of this. Clicking on another skill will create a connection between them.

Properties

You can see connection details in the inspector by clicking on them.



Connection

Connection from TheBlackDeath to DarkBorn

Two Way

Name	Description
Two Way	If this connection goes both ways.

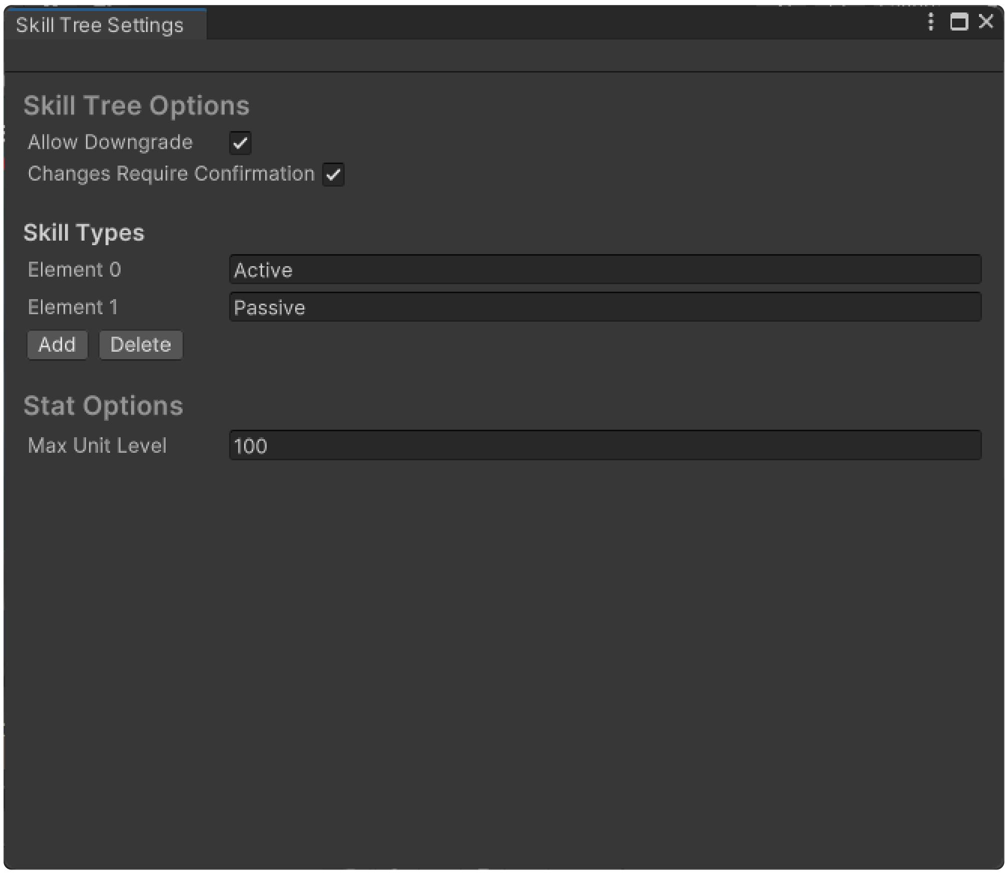
Explaining Directions

Connections have references to 2 skills (skills A and B). One-way connections unlock skill B when skill A is obtained. Switching the direction with the 'Switch Direction' button will simply reverse this (skill B will unlock skill A). In a two-way connection, if skill A is obtained, it will unlock skill B, and if skill B is obtained, it will unlock skill A.

Skill Tree Settings

Going over the skill tree settings.

You can access the settings from `Window > Skill Tree > Settings`.



Skill Tree Options

Allow Downgrade ✓

Changes Require Confirmation ✓

Skill Types

Element 0	Active
Element 1	Passive

Stat Options

Max Unit Level 100

Skill Tree Settings

Properties

Name	Description
Allow Downgrade	If this option is enabled, downgrading skills will be allowed. If disabled, downgrading will be prevented, but resetting the skill tree will still work.
Changes Require Confirmation	If this is enabled, the player will be required to confirm changes made when they upgrade or downgrade skills.

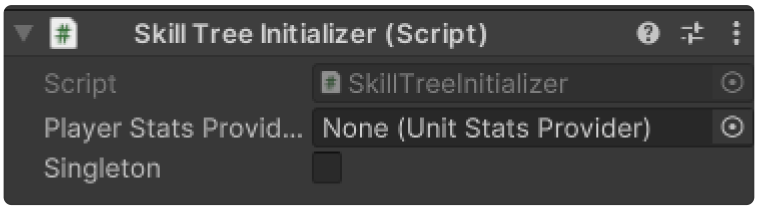
Skill Types	A list of all skill types.
Max Unit Level	The max level a single unit can reach.

Components

A list of components related to skills.

Skill Tree Initializer

A component that simply initializes Skill Tree.


Skill Tree Initializer (Script)

Script	SkillTreeInitializer
Player Stats Provid...	None (Unit Stats Provider)
Singleton	

Skill Tree initializer component

The Skill Tree Initializer component simply initializes Skill Tree on awake.

Properties

Property	Description
Player Stats Provider	The player character's Unit Stats Provider component. Not required.
Singleton	If this is checked, the component will be singleton and won't be destroyed on load.

Default Components

A list of all default UI.

You can find all of Skill Tree's default UI prefabs in `Assets > StylishEsper > Skill Tree > Prefabs > DefaultUI` folder. To use them, just drag and drop them onto your scene.

Confirmation Prompt

Prefab Name: ConfirmationPrompt.prefab

The confirm prompt only appears in-game if you have 'Changes Require Confirmation' field enabled in Skill Tree's settings. It simply asks the player to confirm any changes made to the skill tree. If the user pressed cancel, the changes will be reverted.

Properties

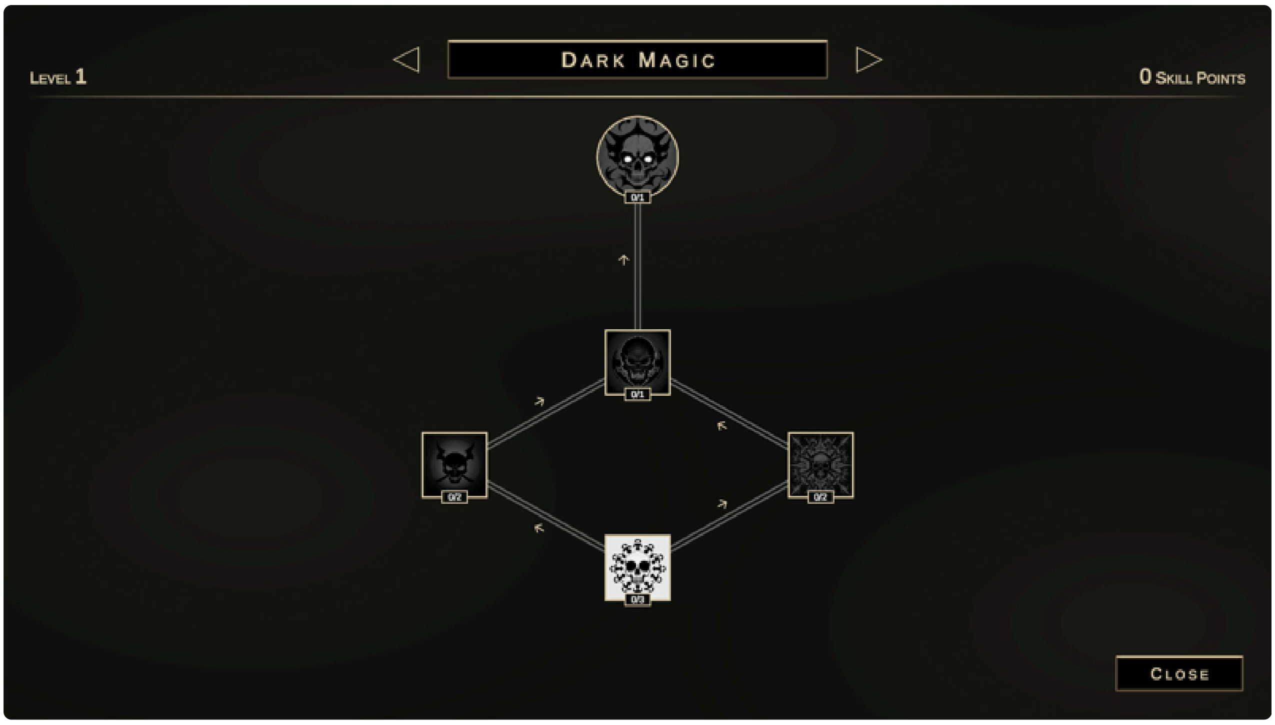
Name	Description
Content	The main content container.
Title Text Mesh	The title TMP object.
Description Text Mesh	The description TMP object.
Confirm Text Mesh	The confirm button TMP object.
Cancel Text Mesh	The cancel button TMP object.
Confirm Button	The button that will confirm changes.
Cancel Button	The button that will cancel changes.

Default Input Handler

Prefab Name: DefaultInputHandler.prefab

This prefab is for basic input to get Skill Tree's default UI working properly. Both the old and new input system are supported. It's recommended that you only use this prefab as an example to figure out how you can incorporate Skill Tree's required inputs using your preferred input method.

Skill Tree Window



Skill tree window

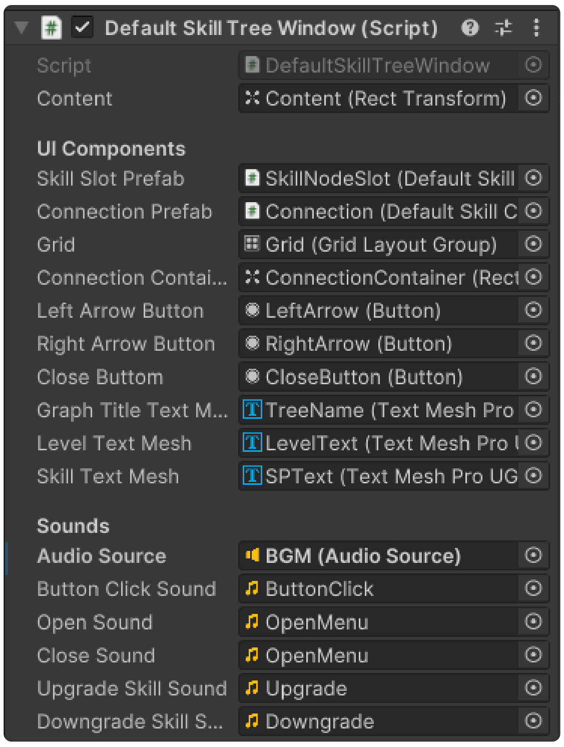
Prefab Name: SkillTreeWindow.prefab

The skill tree window can display every skill tree that you've created from the Skill Tree editor. The layout of the skills is similar to what you see in the editor window itself. You can switch between skill trees with the arrow buttons at the top.

Hovering over skills will display the details of the skill (requires SkillDetails prefab). Left clicking on a skill will upgrade it if possible. Right clicking on a skill will open a slot assignment menu (requires SkillMenu prefab). Using the middle mouse button will downgrade a skill.

Properties

Most properties are already set for this object. The only ones you will have to worry about are under 'Sounds'. Make sure to set an audio source from the inspector after dragging the prefab into your scene. You may edit other properties as you wish.



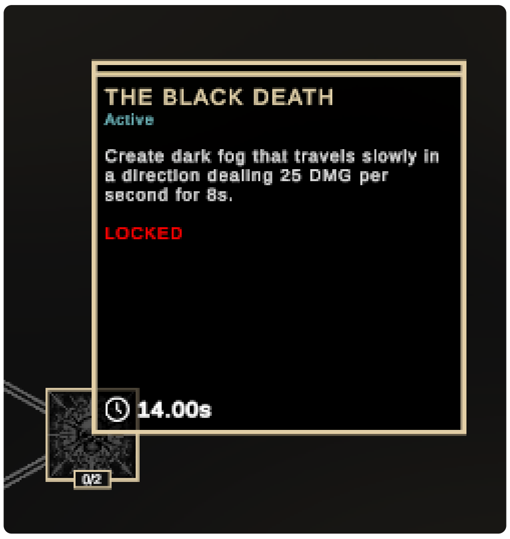
Default Skill Tree Window (Script)

Script	DefaultSkillTreeWindow
Content	Content (Rect Transform)
UI Components	
Skill Slot Prefab	SkillNodeSlot (Default Skill
Connection Prefab	Connection (Default Skill C
Grid	Grid (Grid Layout Group)
Connection Contai...	ConnectionContainer (Rect
Left Arrow Button	LeftArrow (Button)
Right Arrow Button	RightArrow (Button)
Close Buttom	CloseButton (Button)
Graph Title Text M...	TreeName (Text Mesh Pro
Level Text Mesh	LevelText (Text Mesh Pro U
Skill Text Mesh	SPText (Text Mesh Pro UG
Sounds	
Audio Source	**BGM (Audio Source)**
Button Click Sound	ButtonClick
Open Sound	OpenMenu
Close Sound	OpenMenu
Upgrade Skill Sound	Upgrade
Downgrade Skill S...	Downgrade

Name	Description
Content	The main content container.
Skill Slot Prefab	A prefab object of the `DefaultSkillNodeSlot` class.
Connection Prefab	A prefab object of the `DefaultSkillConnection` class.
Grid	A grid layout group that will help position the skills.
Connection Container	A `RectTransform` that will contain all connection instances.
Left Arrow Button	A button to switch to the previous skill tree.
Right Arrow Button	A button to switch to the next skill tree.
Close Button	A button that will close the window.
Graph Title Text Mesh	A `TextMeshProUGUI` object that will display the skill tree title.
Level Text Mesh	A `TextMeshProUGUI` object that will

	display the player's level.
Skill Text Mesh	A `TextMeshProUGUI` object that will display the player's skill points.
Audio Source	An audio source that will play each sound.
Button Click Sound	An audio clip that will play when any button is clicked.
Open Sound	An audio clip that will play when the window is opened.
Close Sound	An audio clip that will play when the window is closed.
Upgrade Skill Sound	An audio clip that will play when a skill is upgraded.
Downgrade Skill Sound	An audio clip that will play when a downgraded.

Skill Details


THE BLACK DEATH
Active
Create dark fog that travels slowly in a direction dealing 25 DMG per second for 8s.
LOCKED
14.00s
0/2

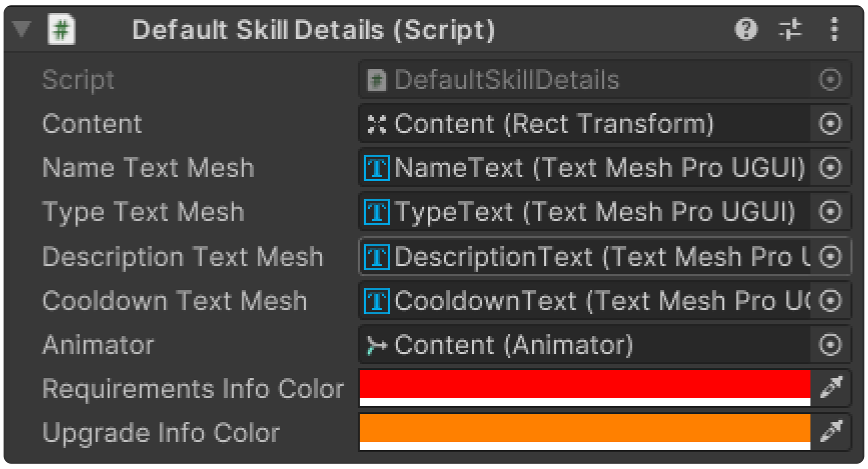
Skill details popup

Prefab Name: SkillDetails.prefab

This UI component simply display basic skill details, as set in the Skill Tree editor window.

Properties

All properties are set for this object. However, you may edit them as you wish.



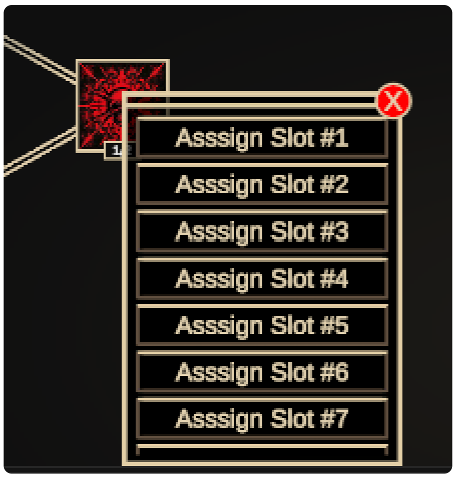
Default Skill Details (Script)

Script	DefaultSkillDetails
Content	Content (Rect Transform)
Name Text Mesh	NameText (Text Mesh Pro UGUI)
Type Text Mesh	TypeText (Text Mesh Pro UGUI)
Description Text Mesh	DescriptionText (Text Mesh Pro U
Cooldown Text Mesh	CooldownText (Text Mesh Pro U(
Animator	Content (Animator)
Requirements Info Color	🟥
Upgrade Info Color	🟧

Name	Description

Content	The main content container.
Name Text Mesh	A `TextMeshProUGUI` object that will display the skill's name.
Type Text Mesh	A `TextMeshProUGU` I object that will display the skill's type.
Description Text Mesh	A `TextMeshProUGUI` object that will display the skill's description, stats, and requirements.
Cooldown Text Mesh	A `TextMeshProUGUI` object that will display the skill's cooldown.
Animator	An animator to run fade animations. This should be on the main content container.
Requirements Info Color	The text color of the requirements information.
Upgrade Info Color	The text color of the upgrade information.

Skill Menu



Skill menu

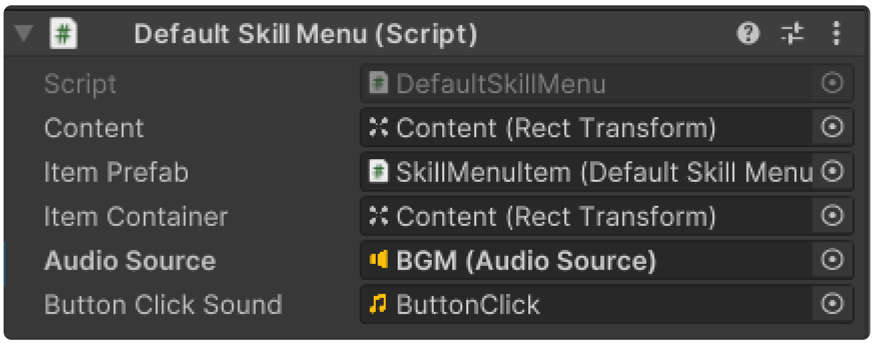
Prefab Name: SkillMenu.prefab

The default skill menu works with the default skill bar by providing options to assign skills to the skill bar's slots. You can even remove assigned skills from this menu.

Make sure to set an audio source for the DefaultSkillMenu component from the inspector after dragging it into your scene.

Properties

Most properties are already set for this object. Make sure to set an audio source from the inspector after dragging the prefab into your scene. You may edit other properties as you wish.

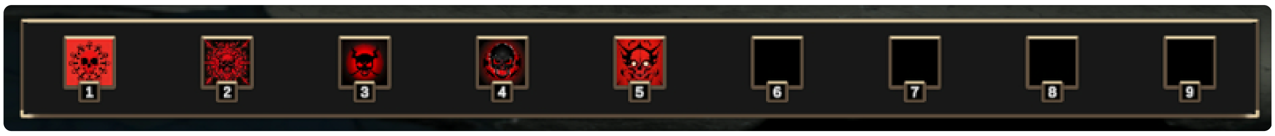


Default Skill Menu (Script)

Script	DefaultSkillMenu
Content	Content (Rect Transform)
Item Prefab	SkillMenuItem (Default Skill Menu
Item Container	Content (Rect Transform)
Audio Source	BGM (Audio Source)
Button Click Sound	ButtonClick

Name	Description
Content	The main content container.
Item Prefab	A prefab object of the `SkillMenuItem` class.
Item Container	A `RectTransform` that will contain all item instances.
Audio Source	An audio source that will play menu sounds.
Button Click Sound	A sound that will play when an item is clicked.

Skill Bar & Windup UI

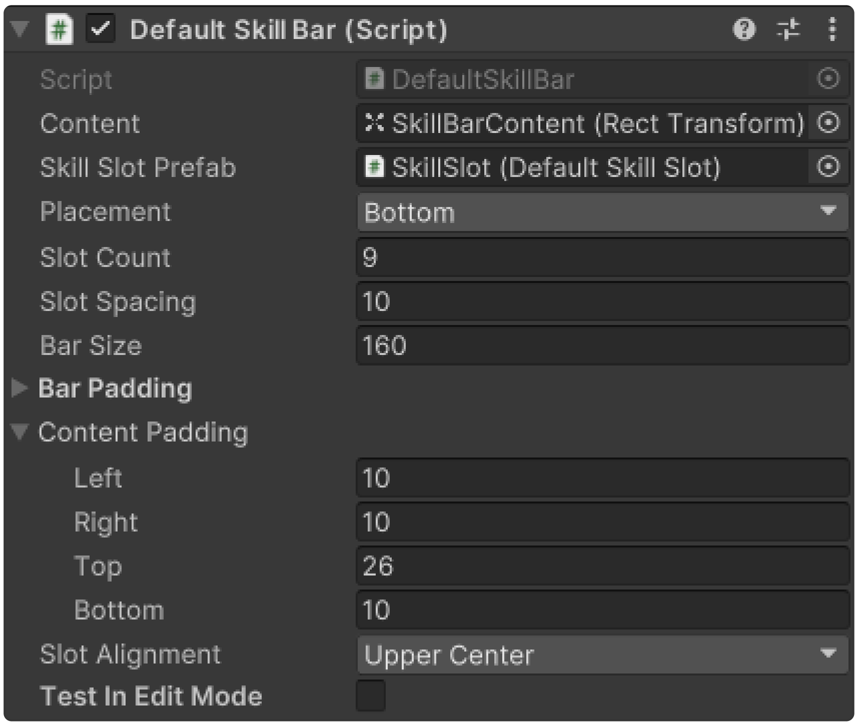


Skill bar

Prefab Name: SkillBar.prefab

The skill bar is generic action bar that you see in a lot of RPG games. It displays all of your assigned skills and hovering over the skills will show skill details (requires SkillDetails prefab).

Skill Bar Properties



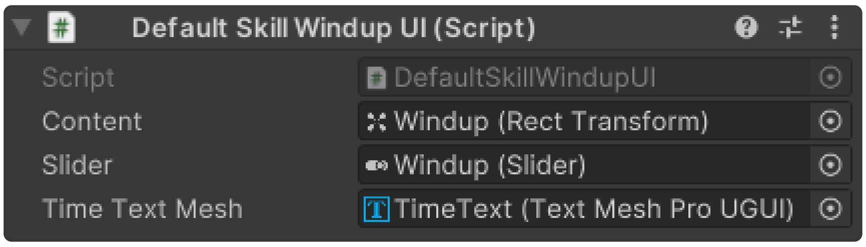
Default Skill Bar (Script)

Script	DefaultSkillBar
Content	SkillBarContent (Rect Transform)
Skill Slot Prefab	SkillSlot (Default Skill Slot)
Placement	Bottom
Slot Count	9
Slot Spacing	10
Bar Size	160
Bar Padding	
Content Padding	
Left	10
Right	10
Top	26
Bottom	10
Slot Alignment	Upper Center
Test In Edit Mode	☐

Name	Description
Content	The main content container.
Skill Slot Prefab	A prefab object of the `DefaultSkillSlot` class.

Placement	The placement location of the skill bar.
Slot Count	The number of slots to instantiate.
Slot Spacing	The spacing between each slot.
Bar Size	The size of the skill bar.
Bar Padding	The padding of the skill bar.
Content Padding	The padding amount of the skill bar's content.
Slot Alignment	The alignment of the slots.
Test In Edit Mode	If the skill bar should be updated in edit mode.

Windup UI Properties

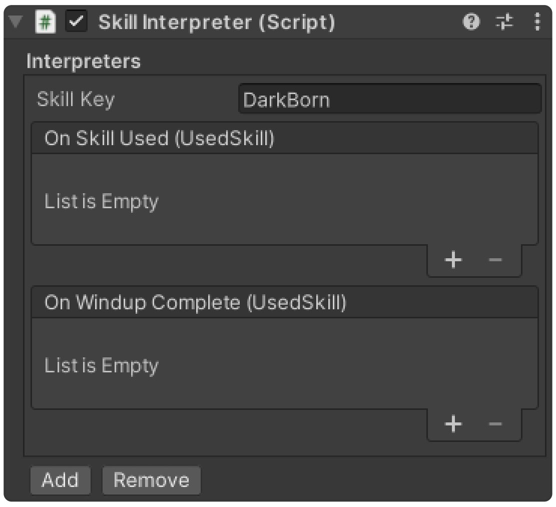

Default Skill Windup UI (Script)

Script: DefaultSkillWindupUI
Content: Windup (Rect Transform)
Slider: Windup (Slider)
Time Text Mesh: TimeText (Text Mesh Pro UGUI)

Name	Description
Content	The main content container.
Slider	A slider to visualize the windup duration.
Time Text Mesh	A `TextMeshProGUI` object that will display the current time remaining in seconds.

Skill Interpreter

A component that connects skills with your own code.



Skill Interpreter (Script)
Interpreters
Skill Key
DarkBorn
On Skill Used (UsedSkill)
List is Empty
On Windup Complete (UsedSkill)
List is Empty
Add
Remove

The Skill Interpreter component can be used to call your own functions for a specific character when a skill is used through Skill Tree.

Interpreters

You can add an interpreter with the `Add` button. An interpreter simply invokes functions whenever a skill with a specified key is used. There's an extra check to see if the user of the skill is the GameObject with this component.

It's recommended to attach a Unit Stats Provider component to the same GameObject with the Skill Interpreter component, as the UsedSkill class (the class that the interpreters use to know which skill was used by which user) requires a UnitStatsProvider reference.

Properties

Name	Property
Skill Key	The key of the skill.

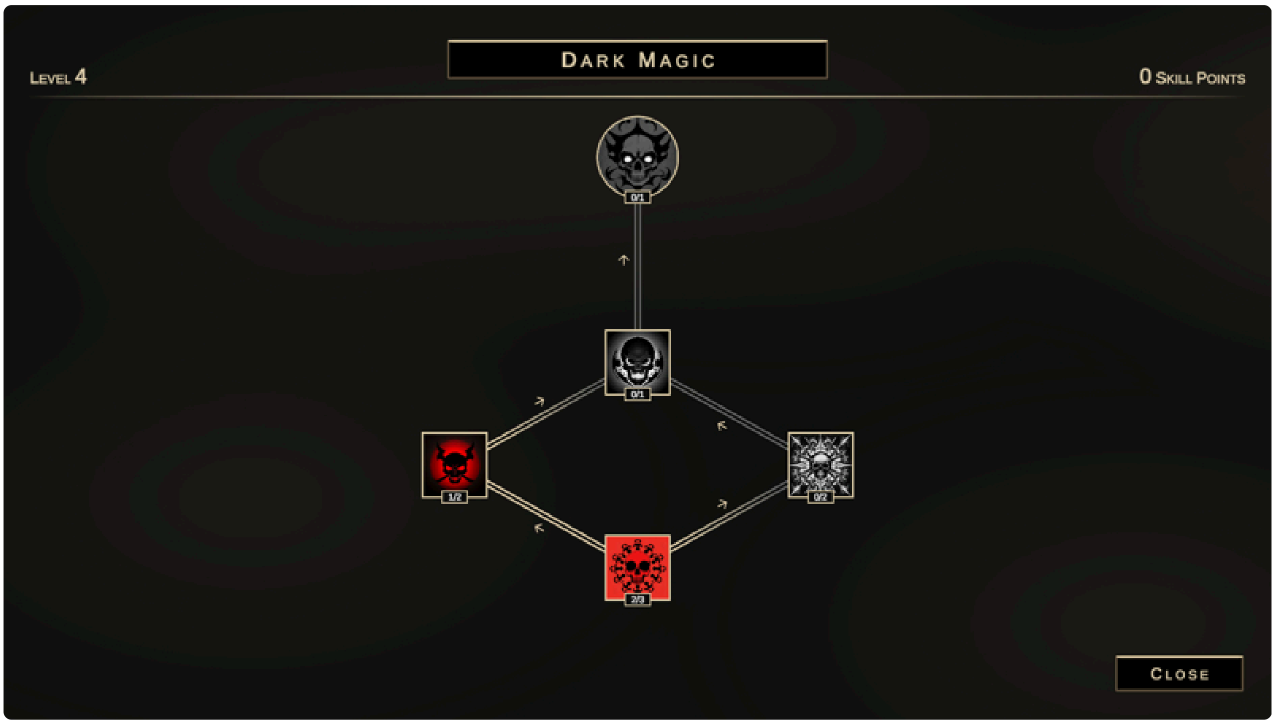
On Skill Used	A list of functions to invoke when the skill is used.
On Windup Complete	A list of functions to invoke when the skill's windup is complete.

You can read more about this in Using Skills.

Customizations

UI Customizations

Skill Tree uses basic Unity UI objects that can be found in the `Assets/StylishEsper/SkillTree/Prefabs/DefaultUI` folder. By editing these prefabs, you'd be directly customizing each part of the Skill Tree's in-game window.



Skill Tree's in-game window

When editing these prefabs, make sure to not delete any objects that a component may require. Simply disable those objects instead if you don't need them.

Custom Skill Menu

The DefaultSkillMenu shows the default functions that may not be what everyone is looking for. That's why it's possible to customize the menu items.

How to Customize

Step 1: Inherit

The best way to customize the skill menu is by creating your own class that inherits from DefaultSkillMenu.

```
using Esper.SkillTree.UI;

public class SkillMenu : DefaultSkillMenu
{

}
```

Step 2: Override

You can override the main method that displays the skill menu. You can use the AddMenuItem In this example, we are simply creating a single skill menu option that places a log in the console.

```csharp
using UnityEngine;
using Esper.SkillTree.UI;

public class SkillMenu : DefaultSkillMenu
{
    public override void DisplayMenuOptions(DefaultSkillNodeSlot slot)
    {
        // Clear previous options
        ClearMenuItems();

        // Create a menu item
        AddMenuItem("Do Something", () => { Debug.Log("Did Something"); }

        // Display the menu
        Open();

        // Force inside game viewport
        ForceInsideView();
    }
}
```

It's recommended to call ClearMenuItems to clear the previously displayed options. Open will actually display the skill menu and ForceInsideView to keep the skill menu inside the game viewport in the case it appears somewhat outside of it.

Step 3: Replace Default

Once you're done making your edits, replace the DefaultSkillMenu component with the new component you created (in this case, SkillMenu). Remember to properly set the required component properties in the inspector.

Custom Skill Details

The DefaultSkillDetails class is a UI component that displays skill details. It may be necessary to customize some of the information that is displayed here, which is why it's possible to customize this class.

How to Customize

Step 1: Inherit

The best way to customize the skill menu is by creating your own class that inherits from DefaultSkillDetails.

```
using Esper.SkillTree.UI;

public class SkillDetails : DefaultSkillDetails
{

}
```

UI text properties available to you will be `nameTextMesh`, `typeTextMesh`, `descriptionTextMesh`, `cooldownTextMesh`, and `costTextMesh`.

Step 2: Override

There are two methods that work similarly but for different UI objects. Both of them can be overriden. These methods are called automatically when a skill slot is hovered.

```csharp
using Esper.SkillTree.UI;

public class SkillDetails : DefaultSkillDetails
{
    public override void ShowDetails(DefaultSkillNodeSlot skillSlot)
    {
        // Use base method
        base.ShowDetails(skillSlot);
    }

    public override void ShowDetails(DefaultSkillSlot skillSlot)
    {
        // Use base method
        base.ShowDetails(skillSlot);
    }
}
```

Step 3: Customize

In this example we are customizing the details displayed when the DefaultSkillNodeSlot is hovered.

```csharp
using Esper.SkillTree.UI;

public class SkillDetails : DefaultSkillDetails
{
    public override void ShowDetails(DefaultSkillNodeSlot skillSlot)
    {
        // Use base method
        base.ShowDetails(skillSlot);

        // Change texts
        nameTextMesh.text = "Some Name";
        typeTextMesh.text = "Some Type";
        descriptionTextMesh.text = "Some description.";
    }
}
```

Step 4: Replace Default

Once you're done making your edits, replace the DefaultSkillDetails component with the new component you created (in this case, SkillDetails). Remember to properly set the required component properties in the inspector.

Custom Objects

If you'd like to use your own stat model instead of the one provided, you can add custom objects to each skill or each skill tree.

The custom object properties accept any Unity type that inherits from `Object`, which means it can accept any prefab or scriptable object.

Explicit Conversion

Since custom objects are type `Object` by default, you will have to manually convert them to the specific type you need.

In the example below, we are converting the custom object of a `SkillNode` to a `Stats` object (`Stats` is not provided in Skill Tree; this is just used here as an example). You can think of the `Stats` class as a scriptable object that stores your custom skill stat properties.

```
public float GetSkillDamage(SkillNode skill)
{
    Stats stats = skill.customObject as Stats;
    return stats.damage;
}
```

Scripting

Scripting with Skill Tree.

Most functionality is already handled by Skill Tree as long as you use the Default Prefabs, Skill Interpreter, and Unit Stats Provider components.

If you're using all of them, you'll only need to understand a few topics, as listed below.

Initialization

Initializing Skill Tree.

Use the `Esper.SkillTree` namespace when scripting with Skill Tree.

Skill Tree's manager is simply called `SkillTree`. This is the main class you'll be working with when using Skill Tree.

Initialization

To do anything will Skill Tree, it must be initialized first. Initializing Skill Tree will simply load all enabled skill graphs. Here's how it can be done:

```
SkillTree.Initialize()
```

Clearing Data

When Skill Tree is not required to be used anymore, you can clear its data. Skill Tree must be reinitialized if you'd like it to be used again.

```
SkillTree.ClearData();
```

Player Level & Skill Points

Editing player level and skill points.

Player Level

Setting the Level

Skills are unlocked with levels. To help Skill Tree keep track of the player's level, use `SkillTree.SetPlayerLevel`. This will also unlock locked skills that are now possible to become unlocked after the level change.

```
// Setting the player level to 1
SkillTree.SetPlayerLevel(1);
```

Getting the Level

To get the player's level, use `SkillTree.playerLevel`.

Skill Points

`SkillTree.playerSkillPoints` is used by Skill Tree to keep track of the player's skill points. This value will increase/decrease automatically as skills are upgraded/downgraded.

You can freely add or subtract from this value as needed.

> ⓘ If your game has multiple characters with differing levels and skill points, you will have to handle this yourself for now.

Skill Graphs

Skill graphs are essentially skill trees.

Getting a Skill Graph

If you've initialized Skill Tree you can access all skill graphs with `SkillTree.skillGraphs`.

Get By Name

```
SkillTree.GetSkillGraph("my skill graph name");
```

Get By ID

```
int mySkillGraphID = 0;
SkillTree.GetSkillGraph(mySkillGraphID);
```

Working with Skill Graphs

All of the data you can set for a skill graph from the Editor Window are public fields. '`mySkillGraph`' in the code examples below is an instance of a `SkillGraph`.

Nodes

You can get the list of all nodes using `mySkillGraph.nodes` field.

Get Node By Position Index

```
int myPositionIndex = 0;
SkillNode mySkillNode = mySkillGraph.GetNodeByPosition(myPositionIndex);
```

Get Node By Index

```
int myNodeIndex = 0;
SkillNode mySkillNode = mySkillGraph.GetNodeByPosition(myNodeIndex);
```

Get Node By Key

```
SkillNode mySkillNode = mySkillGraph.GetNodeByKey("MyNodeKey");
```

Resetting all Nodes

You can reset the level and state of all nodes with `ResetNodeStates`.

```
mySkillGraph.ResetNodeStates();
```

Skill Points

Skill graphs keep track of skill points spent on them.

Total Points

To get the total points required to fully upgrade all skills in a skill graph, use `CountTotalPoints`.

```
int total = mySkillGraph.CountTotalPoints();
```

Points Spent

To get the number of points that the player has spent on a skill graph, use `CountPointsSpent`.

```
int spent = mySkillGraph.CountPointsSpent();
```

Connections

You can access the list of all connections with the `mySkillGraph.connections` field.

Connections of a Skill Node

To get a list of connections for a specific skill node, use `GetConnectionsOfNode`.

```
var connectionsOfSkill = mySkillGraph.GetConnectionsOfNode(mySkillNode);
```

Get Skill Node From Connections

If you have a reference to a connection (`SkillConnection`), you can easily access the connected skill nodes.

Field Name	Description
skillNodeA	The skill node that the connection starts from.
skillNodeB	The skill node that the connection ends on.

Skill Nodes

A skill node is a skill's representation in a skill graph. '`mySkillNode`' in the code examples below is an instance of a `SkillNode`.

Getting a Skill Node

The recommended method of getting a skill node is through a skill graph. Alternatively, you can get a skill node with it's key.

```
SkillNode mySkillNode = SkillTree.GetSkill("MySkillKey");
```

Working with Skill Nodes

All of the data you can set for a skill node from the Editor Window are public fields.

Stats

You can use the `stats` field the access the full list of stats.

Getting a Stat

```
Stat stat = mySkillNode.GetStat("my stat name");
```

States

There are 4 possible skill states:

Name	Description
Locked	If the skill is locked.

Unlocked	If the skill is unlocked.
Obtained	If the skill is obtained.
Maxed	If the skill's level is maxed.

Skills start out locked. Skill states can be managed with the methods below.

You can check the current state of a skill by comparing the state field or you may use the Boolean properties listed below.

Name	Description
IsLocked	If the skill is locked.
IsUnlocked	If the skill is unlocked.
IsObtained	If the skill is obtained.
IsMaxed	If the skill's level is maxed.

Unlocking

Skills must be unlocked to be obtainable (or upgradable).

TryUnlock

This method attempts to unlock a skill if possible. Skills cannot be unlocked if the player doesn't meet the level requirement, no obtained skill leads to this skill through a connection, or if enough required skill points haven't been spent on skills in a specific skill tree.

```
mySkillNode.TryUnlock();
```

TryUnlockAllThroughConnections

This method tries to unlock all skills that the skill has a connection with. This is useful when you need to unlock skills connected to another. For example, if skill B, C, and D have a connection with skill A, this will unlock skills B, C, and D if skill A is obtained.

```
mySkillNode.TryUnlockAllThroughConnections();
```

Obtaining

Once a skill is upgraded at least once, it's state changes to `Obtained`.

TryUpgrade

This method will upgrade a skill if possible. If a skill is not obtained or maxed, nothing will happen. This accepts a skill points parameter (`int`), which will decrease the player's skill points by the amount provided (it will decrease the `SkillTree.playerSkillPoints` field).

```
mySkillNode.TryUpgrade();
```

Maxing

The skill state will change to `Maxed` if it cannot be upgraded anymore.

Downgrading

Skills can be downgraded with `TryDowngrade`. Skill states will change accordingly. Downgrading may fail if the skill is at the lowest level (0). Downgrading may also be prevented from Skill Tree's settings.

```
mySkillNode.TryDowngrade();
```

Optionally, you can use `Deplete` to completely downgrade a skill.

```
mySkillNode.Deplete();
```

Text Interpolation

Skills support text interpolation with the usage of text tags. You can learn more about this here. Essentially, you can use text tags in a string, run the string through the skill's `GetInterpolatedText` method, and it will return the same string except with the tags replaced by the skill's details.

```
// stat0_value refers to the current value of the first stat
string myString = "The skill deals {stat0_value} damage!"
string result = mySkillNode.GetInterpolatedText(myString);
```

Assuming `mySkillNode`'s first stat's current value is 100, `result` will equal:

```
"The skill deals 100 damage!"
```

Level Display String

You can get a string that represents the skill's current level out of the max level.

```
// Example: "0/1"
string level = mySkillNode.levelDisplayString;
```

Getting the Sprite

You can access the sprites with `mySkillNode.lockedSprite`, `mySkillNode.unlockedSprite`, and `mySkillNode.obtainedSprite`.

To get the relevant sprite based on the current state of the skill, use the `GetSprite` method.

```
Sprite skillSprite = mySkillNode.GetSprite();
```

Getting the Skill Graph

Skill nodes store a reference to the skill graph they are a part of.

```
SkillGraph mySkillGraph = mySkillNode.graphReference;
```

Getting the Skill Type

```
string mySkillType = mySkillNode.GetSkillType();
```

Using Skills

Understanding skill usage.

Using a Skill

When a skill should be used, `SkillTree.TryUseSkill` should be called. This will attempt to use a skill if possible. It goes through basic checks to see if a skill can be used. If not, nothing will happen. You wouldn't want a skill to be used while it's winding up or it's on cooldown, right?

This method requires a `SkillNode` and `UnitStatsProvider` reference. A skill node is required to know which skill was attempted. A stats provider is useful because it will make the skill user's stats easily accessible—more on that later.

```
// Where skill is a SkillNode and stats is a UnitStatsProvider object
UsedSkill usedSkill = SkillTree.TryUseSkill(skill, stats);
```

This is called automatically with the Default Prefabs. `TryUseSkill` returns a `UsedSkill` object.

Completing Skill Use

After a skill is used with `TryUseSkill`, you need to look to call `SkillTree.CompleteSkillUse`. This will tell Skill Tree when a skill's usage has been completed. This does not refer to the entire duration of the skill, but rather just the duration of the skill's animation because you wouldn't want another skill to be used while the previous skill's animation is currently running—unless you do, which is why you're free to call it whenever you'd like.

```
// Where skill is a SkillNode and stats is a UnitStatsProvider object
SkillTree.CompleteSkillUse(skill, stats);
```

Used Skill

A `UsedSkill` object is, as it sounds—a skill that was used. It handles a skill's windup and cooldown behind the scenes. You can use `UsedSkill.skill` to get the `SkillNode` and `UsedSkill.user` to get the `UnitStatsProvider`.

Custom Functions

With the [Skill Interpreter]() component, you can run your own functions when a certain skill is used. Each event in the interpreter accepts a `UsedSkill` object. Your custom function can use this to get the stats.

Example

For example, let's say you want to damage an enemy when the "Attack" skill is used. Here's how you could write a function for this:

```
public void OnAttackSkillUsed(UsedSkill usedSkill)
{
    // Combine user and skill damage
    var skillDmg = usedSkill.skill.GetStat("DMG");
    var userDmg = usedSkill.user.GetStat("DMG");
    var totalDmg = skillDmg.baseValue + userDmg.currentValue;

    // target is an enemy with HP
    // ApplyDamage is a method that will decrease the taget's HP by an am
    target.ApplyDamage(totalDmg);
}
```

In the Skill Interpreter component, add an interpreter and give it the key "Attack" (the key of the skill), and then add this function to the On Skill Used event.

This is just an example. The code will greatly depend on your game. Check out the demo scene in the `Assets > StylishEsper > SkillTree > Example` folder for better examples.

Saving & Loading Graphs

Learn how you can save and load skill graphs.

Skill graphs are serializable; thus, they can be saved by most save systems. If you're looking for a save system, try ESave—it has been tested with Skill Tree, and it's free!

Since every save system may be different, this example will only show you how a skill graph can be saved with ESave.

Saving a Graph

```
// Get the save file reference
var saveFile = GetComponent<SaveFileSetup>().saveFile;

// Get a graph named "Dark Magic"
var graph = SkillTree.GetSkillGraph("Dark Magic").CreateSavableGraph();

// Save the data
saveFile.AddOrUpdateData("graph", graph);
saveFile.Save();
```

Loading a Graph

```
// Get the save file reference
var saveFile = GetComponent<SaveFileSetup>().saveFile;

// Load the graph
var graph = saveFile.GetData<SavableSkillGraph>("graph");

// Set the data to Skill Tree
SkillTree.SetSkillGraph(graph);
```

Learn more about ESave here.

Default Skill Tree Window

`DefaultSkillTreeWindow` is singleton. You can access the instance from anywhere with `DefaultSkillTreeWindow.instance` as long as it exists in your scene.

Manually Open/Close

You can manually open or close the `DefaultSkillTreeWindow` by using the methods below.

Open

```
DefaultSkillTreeWindow.instance.Open();
```

Close

```
DefaultSkillTreeWindow.instance.Close();
```

Toggle Active State

This method will open the window if it's in the closed state. Otherwise, it will close it.

```
DefaultSkillTreeWindow.instance.ToggleActive();
```

Events

Skill Tree events.

You can use events to do something when some Skill Tree data is changed.

On Graph Changed

Called when any change has been made to a skill graph. It takes a `SkillGraph` parameter.

```
// Adding an event listener
SkillTree.onGraphChanged.AddListener((x) =>
{
    if (x.name == "some name")
    {
        // Do something...
    }
});
```

On Skill State Changed

This is invoked when any skill from any skill tree is changed. It takes a `SkillNode` parameter.

```
// Adding an event listener
SkillTree.onSkillStateChanged.AddListener((x) =>
{
    if (x.key == "some key")
    {
        // Do something...
    }
});
```

On Skill Upgraded

This is invoked when any skill from any skill tree is upgraded. It takes a `SkillNode` parameter.

```
// Adding an event listener
SkillTree.onSkillUpgraded.AddListener((x) =>
{
    if (x.key == "some key")
    {
        // Do something...
    }
});
```

On Skill Downgraded

This is invoked when any skill from any skill tree is downgraded. It takes a `SkillNode` parameter.

```
// Adding an event listener
SkillTree.onSkillDowngraded.AddListener((x) =>
{
    if (x.key == "some key")
    {
        // Do something...
    }
});
```

On Skill Depleted

This is invoked when any skill from any skill tree is depleted. It takes a `SkillNode` parameter.

```
// Adding an event listener
SkillTree.onSkillDepleted.AddListener((x) =>
{
    if (x.key == "some key")
    {
        // Do something...
    }
});
```

On Skill Used

This is invoked when any skill from any skill tree is used with `SkillTree.TryUseSkill`. It takes a `UsedSkill` parameter.

```
// Adding an event listener
SkillTree.onSkillUsed.AddListener((x) =>
{
    if (x.skill.key == "some key")
    {
        // Do something...
    }
});
```

On Skill Use Completed

This is invoked when `SkillTree.CompleteSkillUse` is called for a skill. It takes a `UsedSkill` parameter.

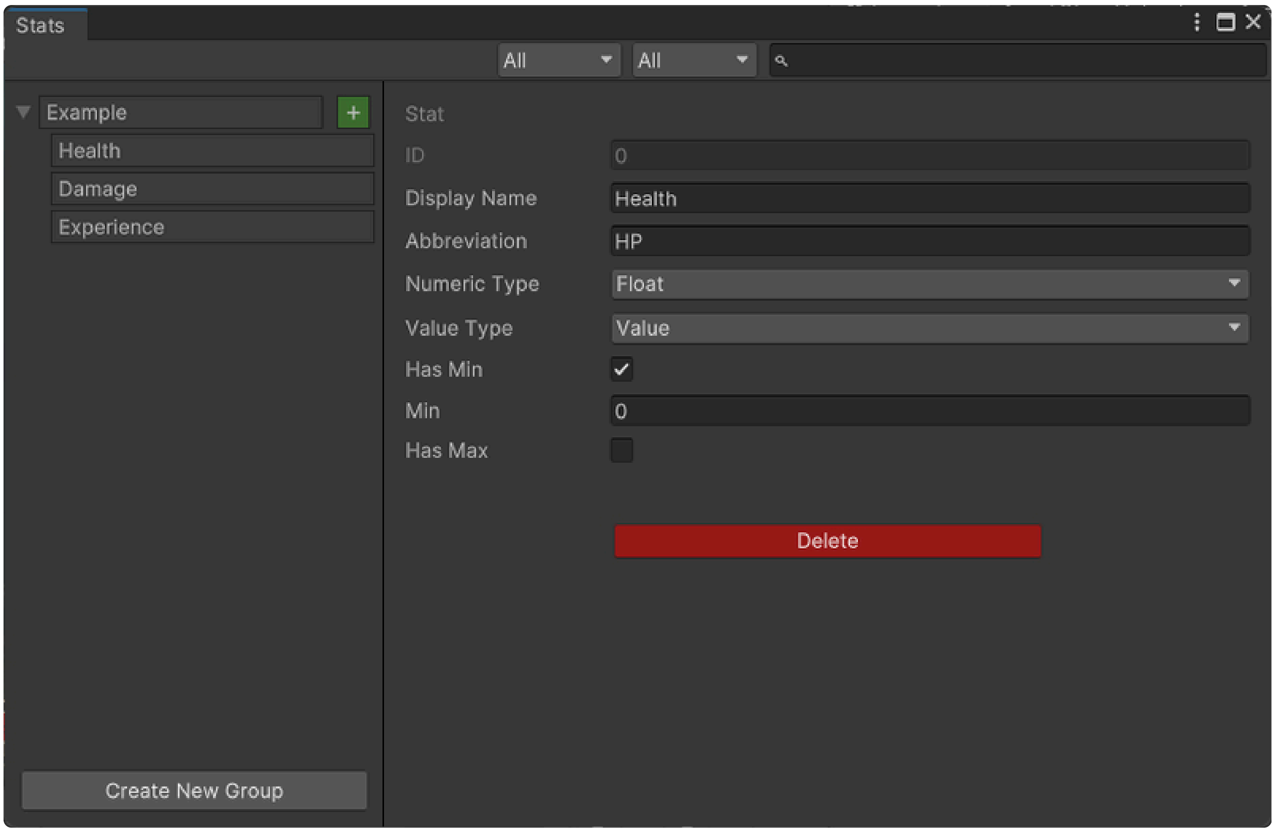
```
// Adding an event listener
SkillTree.onSkillUseCompleted.AddListener((x) =>
{
    if (x.skill.key == "some key")
    {
        // Do something...
    }
});
```

Stats

Skill Tree's built-in stats system.

Stats Editor

Going over the stat editor.



ID	0
Display Name	Health
Abbreviation	HP
Numeric Type	Float
Value Type	Value
Has Min	✓
Min	0
Has Max	

Stats Editor

Skill Tree has its own stats system that allows you to create custom stats for any purpose. You can manage stats from `Window > Skill Tree > Stats`.

The top bar contains search filters, the left-hand side contains stat groups. Stat groups contain a list of stat identities (or stat types). The right-hand side is the property inspector.

Stat Group

Stat groups are used to group together stats that are meant for a similar purpose. When you click on a stat group, it displays their properties on the right-hand side. The little green button creates a stat (StatIdentity) in that specific group.

Properties

Name	Description
ID	The ID of this stat group. This can be used to search for a stat group through code.
Display Name	The name of this stat group. This can be used to search for a stat group through code.
Numeric Type	The default numeric type for new stats created in this group.
Value Type	The default value type for new stats created in this group.
Has Min	The default 'has min' value for new stats created in this group.
Min	The default min value for new stats created in this group.
Has Max	The default 'has max' value for new stats created in this group.
Max	The default max value for new stats created in this group.

Stat Identities

A stat identity is a way to identify a stat and set general rules for it.

Properties

Name	Description
ID	The ID of this stat. This can be used to search for a stat through code.

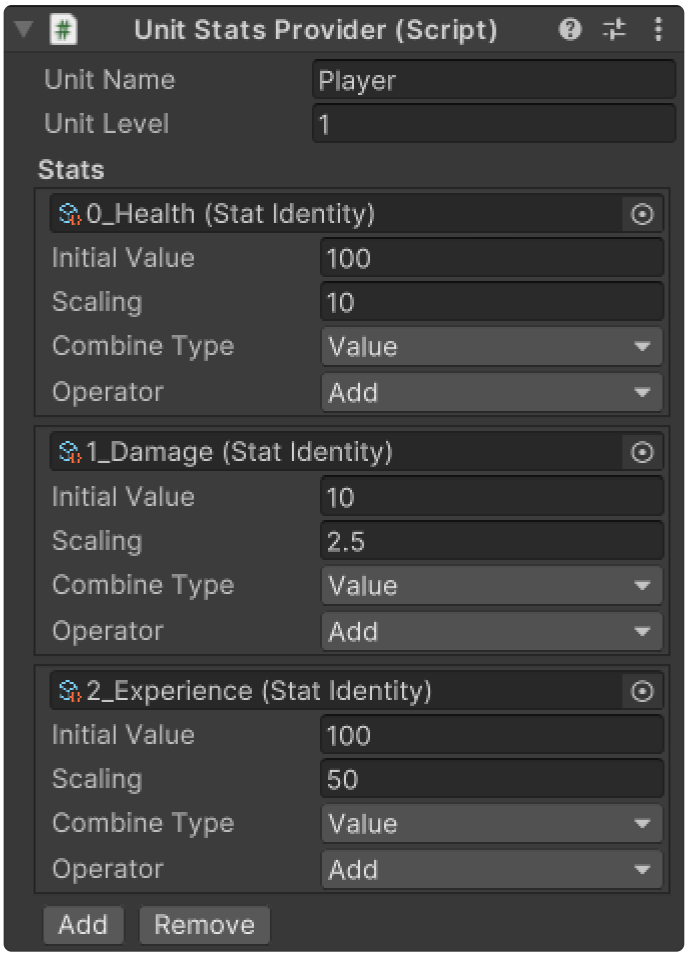
Display Name	The name of this stat. This can be used to search for a stat through code.
Abbreviation	An abbreviated version of the name (e.g. DMG for Damage). This can be used to search for a stat through code.
Type	The numeric type of the stat. **Value**: treat the stat values as is. **Percent**: treat the stat values as a percentage.
Has Min	If the stat has a minimum value. The stat will not go lower than its minimum value.
Min	The minimum value of the stat.
Has Max	If the stat has a max value. The stat will not go higher than its maximum value.
Max	The maximum value of the stat.

Components

A list of components related to stats.

Unit Stats Provider

A component that can provide stats to any GameObject.


Unit Stats Provider (Script)

Unit Name: Player
Unit Level: 1

Stats

0_Health (Stat Identity)
Initial Value: 100
Scaling: 10
Combine Type: Value
Operator: Add

1_Damage (Stat Identity)
Initial Value: 10
Scaling: 2.5
Combine Type: Value
Operator: Add

2_Experience (Stat Identity)
Initial Value: 100
Scaling: 50
Combine Type: Value
Operator: Add

The main component of the stats system is the Unit Stats Provider component. You can use this to provide stats to any `GameObject`. When you first add the component, the stats list is filled with default data for each stat you've created from Skill Tree's settings.

Properties

Name	Description
Unit Name	The name of the unit.
Unit Level	The starting level of the unit.

Single Stat Properties

Name	Description
Stat Identity	The identity of the stat.
Initial Value	The initial (base) value of the stat.
Scaling	The value the stat increases by per level.
Combine Type	How this stat alters another when combined.
Operator	The combine operator.

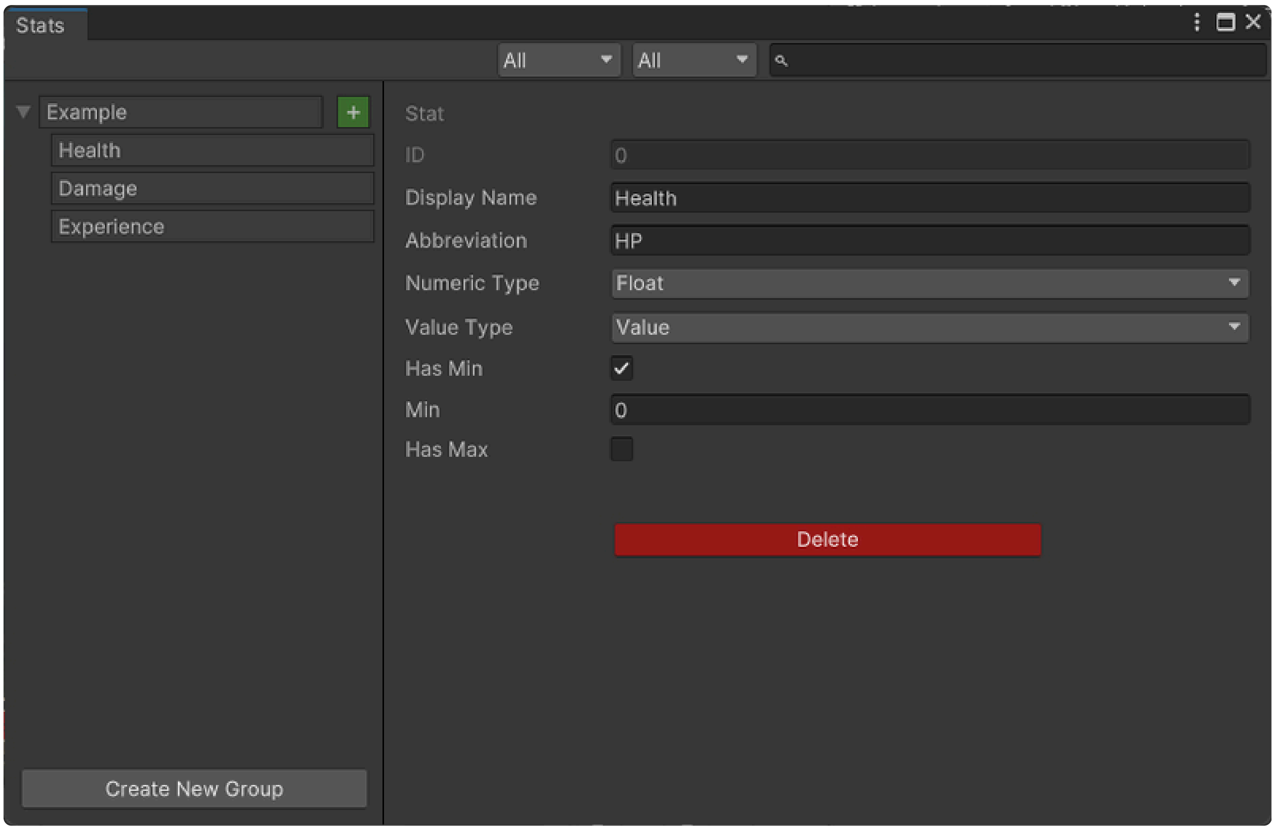
Scripting

See the scripting section.

Creating & Providing Stats

Learn how to create stats and provide them to GameObjects.

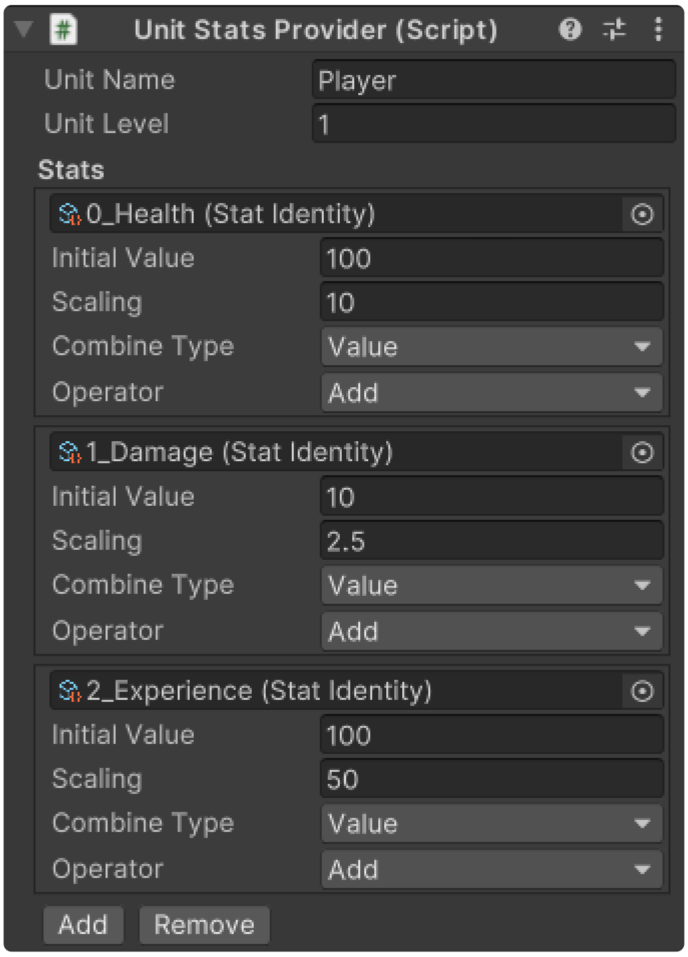
Step 1: Create Stat Identities



Stat	
ID	0
Display Name	Health
Abbreviation	HP
Numeric Type	Float
Value Type	Value
Has Min	✓
Min	0
Has Max	

Open Skill Tree's settings from `Window > Skill Tree > Stats`. Stat identities aren't stats themselves, but rather a way to identify stats and set basic rules for them. You can create a stat by clicking the 'Create New Group' and then the little green '+' button on the created group.

Step 2: Create Stats



Unit Stats Provider (Script)

Unit Name	Player
Unit Level	1

Stats

0_Health (Stat Identity)

Initial Value	100
Scaling	10
Combine Type	Value
Operator	Add

1_Damage (Stat Identity)

Initial Value	10
Scaling	2.5
Combine Type	Value
Operator	Add

2_Experience (Stat Identity)

Initial Value	100
Scaling	50
Combine Type	Value
Operator	Add

This is the simplest way to both create and provide stats to `GameObjects`. Simply add the Unit Stats Provider component to a `GameObject` and set the stats you'd like it to have from the inspector.

Scripting

Scripting with Skill Tree's built-in stats system.

Stat Groups & Identities

You can get stat groups and stat identities through code at runtime. However, they cannot be created at runtime.

Stat Groups

Stat groups are used to group together stats that are meant for a similar purpose.

Getting All Groups

```
var myGroups = SkillTree.GetAllStatGroups();
```

Getting a Single Group

By Name

```
var myGroup = SkillTree.GetStatGroup("My Group Name");
```

By ID

```
var myGroup = SkillTree.GetStatGroup(0);
```

Stat Identities

A stat identity is a way to identify a stat and set general rules for it.

Getting All Stat Identities

```
var myStatIDs = SkillTree.GetAllStatIdentities();
```

From a Stat Group

You can access all stat identities in the group with the `myStatGroup.statIdentities` field.

Getting a Single Stat Identity

By Name

```
var myStatID = SkillTree.GetStatIdentity("My Stat Name");
```

By ID

```
var myStatID = SkillTree.GetStatIdentity(0);
```

Stats

Learn how to work with stats.

Remember to use `Esper.SkillTree.Stats` namespace when scripting with stats.

Creating Stats

While you can create stats through the Skill Stats Editor and Unit Stats Provider, you can also create stats through code.

A reference to a `StatIdentity` is required.

```
// Get a stat identity
var statID = SkillTree.GetStatIdentity("HP");

// Create the stat
// Parameters: ID, initial value, current value, scaling, max stat level
var stat = new Stat(statID, 100, 100, 10, 100);
```

Getting the Stats

You can access stats by getting the `UnitStatsProvider` component of a `GameObject`.

Here's how you can get a stat called 'HP':

```
// Get the component
var statsProvider = GetComponent<UnitStatsProvider>();

// Get the HP stat
Stat hp = statsProvider.GetStat("HP");
```

You can get a stat by both its full name and abbreviation.

Skill Stats

Head over to #stats to learn how to stats from skills.

Working with Stats

There are a few key values of a `Stat` to take note of. All fields and properties listed below are `StatValue` types. `StatValue` is a struct created to support mathematical operations with stats without having to worry about the stat's numeric type.

Name	Description
initialValue	The initial (starting) value of the stat.
currentValue	The current value of the stat.
externalValue	The value added to the stat from external sources.
scaling	The value the stat will increase by per level.
BaseValue	The base value of the stat at the current level.
MaxValue	The max value of the stat at the current level including external value.
NextBaseValue	The base value of the stat if leveled up.
MaxBaseValue	The max base value of the stat if it reaches the max level.

For the HP example, the `currentValue` would be the unit's current HP, which would need to decrease as the unit gets damaged. The `MaxValue` would be the unit's HP without any damage taken.

Changing Values

You can freely change the initial, current, external, and scaling stat values. In the example below, `myStat` is an instance of a `Stat`.

```
myStat.currentValue += 100;
```

This will increase the currentValue by 100. Which is fine, unless you need it to be clamped within the min and max ranges. You can do this instead:

```
myStat += 100; //use -= to decrease
```

This will add the value to current value while also clamping it within the min and max values.

Combining

When using mathematical operators on the `Stat` class, it will alter the currentValue of the stat **if the right-hand side is a number.** If the right-hand side is a `Stat`, the stat on the right-hand side will be combined with the stat on the left-hand side as an external value. Multiply and divide operators are not supported for this purpose, as it's solved by understanding stat combining.

```
// += to combine, -= to remove combination
myStat += myOtherStat
```

Resetting

You can reset the current and external values of a stat with `ResetValues`.

```
myStat.ResetValues();
```

If you'd like to reset the level as well, use `Reset`.

```
myStat.Reset();
```

Understanding Stat Combining

When 2 stat are combined, their `CombineType` and `CombineOperator` fields are considered. These are enums that simplify stat combinations. When stat A is combined with stat B, stat B adds a value to stat A's `externalValue` field.

Combine Types

Name	Description
Value	When combined with another stat, the stat's value will be treated as is.
PercentMax	When combined with another stat, the stat's value will be treated as a percentage of the other stat's max value.
PercentCurrent	When combined with another stat, the stat's value will be treated as a percentage of the other stat's current value.

Combine Operators

Name	Description
Add	Add the value to the other stat.
Subtract	Subtract the value from the other stat.

Example

The best way to explain combining is to use a real-world example.

Let's say you have a player with the an HP stat and a helmet item that when equipped, will increase the players HP by 10%. You can give the helmet a stat that represents the 10% increase by setting the `combineType` to `PercentMax` and `combineOperator` to `Add`.

Representation as code:

```
// Get the HP stat identity
var hpID = SkillTree.GetStatIdentity("HP");

// Create the players HP stat (starts at 100, scaling by 10 per level, ma
Stat playerHP = new Stat(hpID, 100, 100, 10, 100);

// Create the helmet HP stat (simply starts at 10)
Stat helmetHP = new Stat(hpID, 10, 10);

// Set the combine operators for the helmet (so that it represents +10%)
helmetHP.combineType = Stat.CombineType.PercentMax;
helmetHP.combineOperator = Stat.CombineOperator.Add;

// Combine the stats (playerHP will increase by 10% of it's max value)
playerHP += helmetHP;
```

Of course, the HP stats would be a part of different classes (a class that represents the player and a class that represents the helmet item), but the idea is the same.

Stat Levels

Stats have `currentLevel` and `maxLevel` fields to keep track of their level. Stat levels will change automatically with Skill Nodes and Unit Stats Provider when their respective level up/down methods are used. However, it's possible to level up stats individually.

SetLevel

Sets the level of the stat.

```
// Set stat level to 1
myStat.SetLevel(1);
```

TryUpgrade

Increases the level of the stat by a certain amount. If the stat is already maxed, nothing will happen.

```
// Increase stat level by 1
myStat.TryUpgrade(1);
```

TryDowngrade

Decreases the level of the stat by a certain amount. If the stat level is 0, nothing will happen.

```
// Decrease stat level by 1
myStat.TryDowngrade(1);
```

Deplete

Completely downgrades the stat.

```
myStat.Deplete();
```

Unit Stats Provider

Scripting with the UnitStatsProvider class.

The `UnitStatsProvider` class is used to provide stats to `GameObjects`. It will be necessary to manage the stats through code at runtime. In the examples below, `unitStatsProvider` is an instance of a `UnitStatsProvider`.

Getting Stats

You can get a stat by its name or abbreviation.

```
Stat myStat = unitStatsProvider.GetStat("My Stat Name");
```

Alternatively, you can access all stats with `data.stats`.

```
var myStatList = unitStatsProvider.data.stats;
```

Updating Stats

Stat values can be set with the `SetStat` method.

```
// Parameters: stat name, new stat value
unitStatsProvider.SetStat("My Stat Name", 100);
```

You can increase a stat by a certain amount with the `IncreaseStat` method.

```
// Parameters: stat name, added stat value
unitStatsProvider.IncreaseStat("My Stat Name", 100);
```

A stat's value can be decreased with the `DecreaseStat` method.

```
// Parameters: stat name, removed stat value
unitStatsProvider.DecreaseStat("My Stat Name", 100);
```

To reset all stats, call the `ResetAllStats` method.

```
unitStatsProvider.ResetAllStats();
```

Leveling

When the level of a `UnitStatsProvider` is changed, all of its stats level's will be adjusted to match.

SetLevel

Set the level with the `SetLevel` method.

```
// Set the level to 1
unitStatsProvider.SetLevel(1);
```

LevelUp

Call `LevelUp` to increase the level by 1.

```
unitStatsProvider.LevelUp();
```

LevelDown

Use `LevelDown` to have the opposite effect.

```
unitStatsProvider.LevelDown();
```

Events

Working with stat events.

Each Unit Stats Provider has its own events that you can use to make something happen when a stat is changed. Each event accepts an `Stat` parameter.

Property	Description
onStatChanged	Called whenever a stat's current value is changed.
onStatReachedMin	Called whenever a stat's current value reaches its min value.
onStatReachedMax	Called whenever a stat's current value reaches its max value.
onStatLevelChanged	Called whenever a stat's level is changed.

Usage Example

Let's say you have a stat called HP. If you want something to happen when the HP reaches 0, you can use the `onStatReachedMin` event.

Script Example

In this example, we're calling a custom function called `Death` that handles player death when the HP stat reaches 0.

Adding the Listener

```
// Get the UnitStatsProvider component
var statsProvider = GetComponent<UnitStatsProvider>();

// Add a listener
statsProvider.onStatReachedMin.AddListener(Death);
```

Custom Function

```
public void Death(Stat stat)
{
    // Check if it's the HP stat
    if (stat.NameMatches("HP"))
    {
        // Your death code here...
    }
}
```

Saving & Loading Stats

Learn how you can save and load stats.

`UnitStats` and `Stat` are serializable classes; thus, they can be saved by most save systems. If you're looking for a save system, try ESave—it has been tested with Skill Tree, and it's free!

Since every save system may be different, this example will only show you how stats can be saved with ESave.

Saving Stats

```
// Get the save file reference
var saveFile = GetComponent<SaveFileSetup>().saveFile;

// Where unitStats is a UnitStats object (this can be obtained from a Uni
// Save the data
saveFile.AddOrUpdateData("stats", unitStats);
saveFile.Save();
```

Loading Stats

```
// Get the save file reference
var saveFile = GetComponent<SaveFileSetup>().saveFile;

// Load the stats
var unitStats = saveFile.GetData<UnitStats>("stats");

// Where statsProvider is a UnitStatsProvider object
// Set the data to the provider
statsProvider.data = unitStats;
statsProvider.Refresh();
```

Learn more about ESave here.

Support

Getting Help

In need of assistance?

A Little Heads Up...

I'm a solo developer trying to simplify game development for others (and myself) by creating useful Unity tools. I'm not a part of a large or even small team; it's just me. If I receive your message, I will respond as soon as I can (just give me a day or two).

Help Me Help You

Errors

If you're facing an error, please list them in your message and explain the steps you took that led to that error.

Bug Reports

If you're reporting a bug, please include answers to these questions in your report:

1. What were you trying to do?
2. What did you expect would happen?
3. What actually happened?

Feature Request

I'm always looking to improve my products. If you'd like a specific feature added, don't hesitate to let me know. In your message, please explain how the feature will help your use case.

Options

You can contact me using any of these methods.

1. Discord Server
2. Website Contact Form
3. Emailing developer@stylishesper.com
4. Messaging me on X (Twitter)

Changelogs

Latest Releases

v1.2.0

- Revamped the built-in stats model
- Added stat types editor window
- Simplified managing stats for skills

v1.1.3

- UI customization improvements

v1.1.2

- Added the ability to attach custom objects to skill graphs and skill nodes

v1.1.1

- Improved skill graph saving & loading

v1.1.0

- Fixed a bug causing skill icons to load too slowly
- Major backend changes to skill nodes and skill graphs (updated the editor window to reflect this)
- Added skill costs
- Dropped support for Unity 2021

v1.0.4

- Fixed a bug causing connections to not properly connect in different screen sizes
- Other minor improvements

v1.0.3

- Added the ability to revert changes
- You can now prevent skill downgrades

v1.0.2

- Added a reset button that removes all skill upgrades in the skill tree window
- Fixed a bug causing skills to not automatically deplete when there aren't enough skill points spent in the skill tree

v1.0.1

- Added SkillTreeInitializer
- Other minor improvements

v1.0.0

Initial release of Skill Tree.

Rate Me?

If it's not too much to ask.

I’m a solo developer passionate about making game development easier for everyone, including myself. If this asset has been helpful to you, I’d greatly appreciate it if you could leave a rating. Your support motivates me to improve this asset further and create even more useful tools. Thank you!

Skill Tree
UnityAssetStore