Game Creator Documentation

From Zero to Hero

Table of contents

1.	Ga	me Creator	5
	1.1	Welcome to Game Creator	5
	1.2	Getting Started	8
	1.3	Characters	14
	1.4	Cameras	47
	1.5	Visual Scripting	51
	1.6	Variables	655
	1.7	Advanced	661
	1.8	Releases	698
2.	In	ventory	735
2	2.1	Inventory	735
2	2.2	Setup	736
2	2.3	Items	737
2	2.4	Bags	746
2	2.5	Currencies	750
2	2.6	Merchants	751
2	2.7	Tinkering	753
2	2.8	Visual Scripting	754
2	2.9	User Interface	866
2	2.10	Releases	871
3.	Di	alogue	879
3	3.1	Dialogue	879
3	3.2	Setup	880
3	3.3	Dialogues	881
3	3.4	Actors	889
3	3.5	Skins	892
3	3.6	User Interface	894
3	3.7	Visual Scripting	896
3	3.8	Releases	912
4.	ats	916	
4	1.1	Stats	916
4	1.2	Setup	917
2 2 2		Classes	918
	1.4	Formulas	924
	1.5	Tables	928

4.6	Stat Modifiers	929
4.7	Status Effects	930
4.8	User Interface	932
4.9	Visual Scripting	937
4.10	Releases	965
5. Qu	ıests	969
5.1	Quests	969
5.2	Setup	970
5.3	Quests	971
5.4	Journal	979
5.5	Visual Scripting	980
5.6	User Interface	1024
5.7	Releases	1027
6. Be	ehavior	1030
6.1	Behavior	1030
6.2	Setup	1033
6.3	State Machines	1034
6.4	Behavior Trees	1038
6.5	GOAP	1043
6.6	Utility AI	1047
6.7	Visual Scripting	1051
6.8	Releases	1065
7. Pe	erception	1066
7.1	Perception	1066
7.2	Setup	1067
7.3	Awareness	1068
7.4	Sensors	1071
7.5	Luminance	1079
7.6	Camouflage	1081
7.7	Obstruction	1082
7.8	Din (Ambient Noise)	1083
7.9	Evidence	1084
7.10	Visual Scripting	1086
7.11	User Interface	1137
7.12	Releases	1142
8. Sh	nooter	1143
8.1	Shooter	1143

9. Melee	1144	
9.1 Melee	1144	
9.2 Setup	1145	
9.3 Weapons	1146	
9.4 Shields	1148	
9.5 Skills	1151	
9.6 Reactions	1163	
9.7 Combos	1165	
9.8 Input	1167	
9.9 Targets	1169	
9.10 Visual Scripting	1171	
9.11 Releases	1212	
10. Traversal		
10.1 Traversal	1215	
11. Extensions	1216	
11.1 Welcome to Extensions	1216	
11.2 Transitions	1217	
11.3 Localization	1220	
11.4 Addressables	1224	
11.5 Footsteps	1227	

1. Game Creator

1.1 Welcome to Game Creator

Every game begins with an idea - a world to build, a compelling game mechanic, a feature that players are bound to fall in love with - but it takes a lot of work to bring that idea into fruition. Game Creator is a collection of tools to help make the journey from idea to playable game a lot smoother.

Game Creator 2



Game Creator is sometimes informally abbreviated as GC.

1.1.1 Who is it for?

Game Creator is the perfect tool for both beginners and experienced users.

- Newcomers will find an easy-to-use tool with a very smooth learning curve, thanks to the small amount of concepts one has to learn in order to get started.
- Experienced users will find that these small set of tools have a lot of depth and can be synergically used to create any mechanic with ease, while favoring quick iteration.

Game Creator also has a very straight-forward API for programmers, from which they can extend the tools with new features and seamlessly integrate them with the rest of the ecosystem of tools. Level and art designers can quickly test their environments, creating a playable character and a camera type that fits their game with just a couple of clicks. And game designers will be delighted with a plethora of tools that they can use and exploit to create intrincate game mechanics.

1.1.2 How to get started

The easiest way to start learning how to use **Game Creator** is to jump to the **Getting Started** section. It overviews everything you need to know to get up to speed and assumes you have no technical knowledge. It also contains links to other learning resources from where to learn more.

1.1.3 What is it?

The **Game Creator** package comes with a slew of tools that help you very easily make the game of your dreams. These tools have been carefully crafted to be as flexible and intuitive as possible. Each tool takes care of dealing with the heavy-math under the hood and present it to you in a very human-friendly form, so you can focus on what really matters: Making games.

- Characters: Characters are entities living in your scene. These come loaded with common features, such as inverse kinematics, obstacle avoidance navigation, user input, jumps, footstep sound effects and animation systems.
- Cameras: Cameras allow to control how your game is framed. From an orbiting third-person perspective with zoom and geometry clipping avoidance to more traditional fixed camera angles, top-down perspectives or first-person views.
- **Visual Scripting:** Visual Scripting in Game Creator is very unique: Instead of using a typical node graph, it borrows the concept of task lists. This makes it really easy to read, organize and keep all interactions under control without the project quickly becoming a spaghetti mess.
- · Variables: Variables allow to keep track of the game's progress and storing it when the user saves the game.

Wre tools

Game Creator comes with more tools than the aforementioned above. However, we recommend beginners focus on understanding these first. Experienced users and programmers can jump to the Advanced section to know more about the rest.

1.1.4 Modules

Game Creator is built to be extremely flexible and extensible. Modules are add-on packages that extend the features provided even further. For example, the Inventory module allows to easily define items with different properties, which can later be equipped, consumed, crafted, dropped, sold, bought or stored in chests.

- · Inventory: Manage and equip items, craft new ones and trade them with other merchants.
- Dialogue: Create conversations with other characters with branching narratives.
- Stats: Make complex RPG interactions with intertwined stats, attributes and status effects.
- · Quests: Keep your game's progress and lore under control with a mission manager.
- · Behavior: Easily manage character's AI using Behavior Trees and other mechanisms.
- · Perception: Allow entities to use sight, smell or hearing to understand the world.
- Shooter: Create long-ranged shooting mechanics.
- · Melee: Define close quarter combat mechanics with parries and combos.
- Traversal: Give characters the ability to climb and other traversing skills.

Nidular synergy

Modules do not just extend **Game Creator**'s capabilities, but can also communicate with other **Modules**. This allows to intertwine their features and develop even more complex game mechanics.

Example of use case

A very common case is using the **Dialogue** module along with the **Stats**. The first one allows to easily manage conversations between characters, where the player is prompted with choices and characters react to these. The **Stats** module, on the other hand, allows to define RPG traits to objects.

By combining these two modules you can create more interesting mechanics, such as displaying an option during a conversation with a character, where trying to intimidate it will only yield in success if the player has a certain stat (for example strength) above a certain value.

1.1.5 Documentation

If you're reading this from a PDF file, make sure you're reading the latest version of the documentation. Click Download PDF to get the latest version.

However, we recommend you read this documenation from the website itself, which contains GIFs, higher quality images and better navigation options. PDF should only be used as an offline alternative.

The documentation is structured as follows:

documentation structure

- 1. The top navigation shows a list of all the available <code>Modules</code> with their own documentation.
- 2. The central page is dedicated to the content of the current page.
- 3. The left side-bar shows the current page you are reading.
- 4. The right side-bar shows the table of contents of the current page.

Game Creator 1.x Support

Game Creator 2.0 is not compatible with Game Creator 1.x because its code base has been re-architectured. However, most concepts are identical or very similar.

Each module has one or multiple pages dedicated to the description of what each sub-system does, with clear examples, tips and tricks. Moreover, for those who want to go one step further, all sub-systems have an *Advanced* chapter with more technical details on how it works and how it can be extended through the exposed scripting API.

1.1.6 Errata

If you find a mistake or omission in the documentation, please send us an email at docs@gamecreator.io with a link to the relevant entry and an explanation what you think is wrong. We'll take a look and make any necessary updates.

1.2 Getting Started

1.2.1 Getting Started

Welcome to the Getting Started section. Here you will find all necessary resources to get you started with Game Creator.

- Installation: Learn how to install Game Creator from the Unity Asset Store.
- $\boldsymbol{\cdot}$ $\boldsymbol{\mathsf{First}}$ $\boldsymbol{\mathsf{Steps:}}$ Get to know the basic first steps towards using Game Creator.

Once you are comfortable with the core concepts, we recommend checking the **Examples** that come with Game Creator and the free **Courses** available on the website. If you prefer to learn in non-written format, you can also check our Youtube channel, where we upload new video tutorials.

- Examples: Discover examples to learn from and production-ready templates.
- · Video Tutorials: A collection of courses you can take at your own pace.

We also recommend checking out the **Game Creator Hub**: It's a community-driven platform where anyone can download further free Instructions, Conditions and Events.

• Game Creator Hub: Explore how the Hub can help you connect with other developers and expand the tools at your disposal.

1.2.2 Installation

This guide explains how to set up your Game Creator project from scratch. It includes information about prerequisites, installing the package, creating an initial workspace and verify your setup.

Creating a new project

Start by downloading the Unity Hub software and install the latest Unity version. Create a new blank project and choose the rendering pipeline that suits you best.

Kendering Pipeline

We recommend using the **Built-in Rendering Pipeline** (BRP) if it's the first time you're using Unity or you just want to try out Game Creator. If you want to use **URP** or **HDRP**, convert the materials automatically clicking on *Edit Rendering Materials Convert all built-in materials to URP/HDRP*.

Get the Game Creator core package from the Unity Asset Store following the link below:

Get Game Creator

Once you have purchased it, click on the "Import" button on the website and the Unity Editor's **Package Manager** window should appear with the **Game Creator** package selected. Click on *Download* and *Import* afterwards.

Package Manager

Let the process complete and if everything went fine, your console shouldn't have any errors. If you do, please feel free to reach out to our support email.

Verify installation

If you have successfully installed Game Creator you should see a new "Game Creator" menu at the top-toolbar with a set of options. You'll also have access to a new "Game Creator" section right clicking on both the *Hierarchy* panel and the *Project* panel.

Setting up for Git

We highly recommend using GitHub or GitLab for backing up your projects. If you use Git as your main repository source be sure to add the following snippet at your ligitignore file:

Game Creator /Assets/Plugins/GameCreator/Documentation.pdf /Assets/Plugins/GameCreator/Packages

This will avoid adding the offline documentation file to your git repository as well as the examples & code from the Game Creator asset. The reason why the code can be ignored is that it can be easily downloaded from the Asset Store. If you prefer to save a local copy of the current version of your Game Creator package, skip the last two lines and only include the following on your .gitignore file:

Game Creator /Assets/Plugins/GameCreator/Documentation.pdf

1.2.3 First Steps

In this section you'll learn to setup a very simple example that uses some of the core features of Game Creator. It shouldn't take you more than 5 minutes to have it up and running.

Preparing the scene

Let's start creating the geometry that will hold the scene. Right click on the *Hierarchy Panel* and select 3D Object Plane. This is going to be the floor.

If the scene doesn't have a light, create one right clicking again on the *Hierarchy Panel* and select Light Directional Light and place it somewhere that shines downwards towards the plane.

Finally, if the scene doesn't have a camera object, create one clicking on the *Hierarchy Panel* and select Create Camera. Select it and, in the upper-part of the *Inspector* window, change its tag from Untagged to MainCamera. You should also change the camera's position and rotation so it points towards the center of the plane, in order to visualize what happens in it.

Geometry Setup

Creating the Player

To create a player character, open the *Hierarchy Panel* context menu and select Game Creator Characters Player. This should have created a character object in the scene in T-pose. If you click play, you should be able to control the default player using the WASD keys or a controller, if you have one plugged in.

Player Setup

Creating a camera

Game Creator uses Camera Shots to tell the main camera how to behave and which target/s to follow. The easiest way to follow the player character is to use the Third-Person camera shot, which automatically orbits around the player using the mouse's movement and allows to zoom in/out.

To create a **Camera Shot** open again the **Hierarchy Panel**'s context menu and select Game Creator Cameras Camera Shot.

Automatic camera detection

Creating a new Camera Shot will automatically add the Main Camera component on the scene's main camera, if any at all. If the main camera doesn't have any Camera Shot assigned, it will assign this newly created shot.

The default Camera Shot is the Fixed one. However, we want to use the Third-Person Orbit shot. To change the type of camera shot, click on its name and select **Third Person** from the dropdown menu.

New options should appear now. We need to specify the target at which the camera will look at and orbit around. In both cases, this is the Player, so choose the "Player" option from the Look Target and Orbit Target fields.

Enter Play-Mode and you should be able to move the player like before, but the camera should also track it and orbit around it using the mouse or controller's right stick.

Complete Setup



Check out Game Creator's free courses for more step-by-step tutorials

1.2.4 Toolbar

Since version 2.3.15, **Game Creator** comes with a dockable **Toolbar** that can be used to create common components in the scene view.

Game Creator Toolbar

Asplay Toolbar

If the Toolbar is not displayed by default, focus on the scene view and right click on the top tab and select Overlay Menu. This will pop a vertical menu that allows to show/hide different toolbars. Click on Game Creator to enable its visibility.

Show Toolbar

The toolbar can be docked as any other toolbar. Simply drag the handles and drop them on any corner or edge.

The orientation can also be changed to fit the position. To do so, right click the handles and select one of the following options:

- · Panel: Displays an horizontal stripe with the name and icons for each button
- Horizontal: Shows an horizontal stripe with just the icons
- · Vertical: Similar to Horizontal, but displays each button vertically stacked

Tooltips

We recommend using either Horizontal or Vertical layouts. Hovering over any of the icons will display a small tooltip with a description of what that button does.

1.2.5 Examples

Game Creator comes packed with a collection of examples that have been carefully hand-crafted to speed up your development process even further with common mechanics. You can think of them as *templates* of game mechanics you can use for your projects.

To install an example, head to the top toolbar and click **Game Creator** Install.... A window will appear with a collection of available examples to install. Select one that you want to add and click *Install*.

Install Window



An example may or may not have a list of dependencies. The **Install** window will display a green icon if the example dependency is installed or a red icon if it is not. Installing a module with dependencies will install and update all dependencies.

Once you do that, the example will appear under Assets/Plugins/GameCreator/Installs/ or you can simply click the Select button to automatically select the example's folder.



When installing an example, it is located at the Plugins/GameCreator/Installs/ directory. The name of the example's folder is the [name of the module] followed by a dot, the [name of the example] followed by an @ (at) symbol and the version number. For example, Game Creator's Example 1 with version 1.2.3 will be located at: Plugins/GameCreator/Installs/GameCreator.Example1@1.2.3/.

Uninstalling an Example

If you want to uninstall an example, simply delete root folder of the example. For instance, if you installed a Game Creator example called "Example 1", you can right click the folder at Assets/Plugins/GameCreator/Installs/GameCreator.Example1@1.0.0/ and choose Delete. This will permanently delete the example from your project. However, you can still reinstall it again from the Install window.

1.3 Characters

1.3.1 Characters

One of Game Creator's main systems is the Character. It represents any interactive playable or non-playable entity and comes packed with a collection of flexible and independent features that can be used to enhance and speed up the development process.

Main Features

A Character is defined by a Character component that can be attached to any game object. It is organized into multiple collapsable sections, each of which controls a very specific feature of this system.

Some of the most noticeable features are:

- · Player Input: An input system that allows to change how the Player is controlled at any given moment. Including directional, point & click, tank-controls, and more.
- · Rotation Modes: Controls how and when the character rotates. For example facing the camera's direction, its movement direction or strafing around a world position.
- · World Navigation: Manages how the character moves around a scene. It can use a Character Controller, a Navigation Mesh Agent, or plug-in a custom controller.
- · Gestures & States: An animation system built on top of Unity's Mecanim which simplifies how to play animations on characters.
- · Inverse Kinematics: An extendable IK system with feet-to-ground alignement or realistic body orientation when looking at points of interest.
- · Footstep Sounds: A very easy to use foot-step system that mixes different sounds based on the multiple layers of the ground's materials and textures
- · Dynamic Ragdoll: Without having to configure anything, the Ragdoll system allows a character to seamlessly transition to (and from) a ragdoll state.

Player Character

The Player character uses the same Character component as any other non-playable character but with the difference that it has the Is Player checkbox enabled. A Character with this option enabled processes the user's input based on its Player section.



che Player per Scene

There can be only one Player character per scene. You can use the Change Player instruction to change who the Playable character is, but at any given time, just one Character might have the Is Player checkbox ticked.



Note that when creating a Player game object from the Hierarchy menu or the Game Creator Toolbar, it ticks the Is Player checkbox by default.

1.3.2 Component

The Character system is built using a single component called Character component and handles everything a character can do; From playing animations to footstep sounds, modifying animations though inverse kinematics and much more.

Character Component

General Settings

This block includes the big mannequin icon and two fields:

- Is Player: Determines whether this character is a Player character or not. A Player character processes input events and makes the character respond accordingly.
- Update Time: Indicates whether the character should work with the internal game's clock the real-life clock.

Character Component



Game Time vs Unscaled Time

By default all characters should use the game's clock. Setting the game's time scale to zero will freeze the game, which is useful for pausing it. However if your game has a mechanic where a character ignores the time scale, you can use the unscaled real-life clock.

The mannequin icon isn't just an aesthetic ico, but a debugging tool. When the game is running, the icon will change into a green colored one and will turneach of its limbs red every time the character performs a blocking action that prevents that limb from doing something else. For example, performing a jump makes the legs be busy for a little less than a second, as well as landing.

The mannequin icon will change into a red skull when the character is considered dead.

Kernel Settings

This block is the most important one. A Character behavior is divided into 5 main categories (known as Units) and each one can be changed individually without affecting the rest.



This settigs block is called the Kernel of the character and each individual row is called a Control Unit or Unit for short.

Character Component

To change each type of Unit click on the right-most icon of each and choose the implementation you want. Clicking on the name of the **Unit** will expand/collapse its available options.



Custom Character Controllers

Game Creator comes with a collection of Units so you can customize how you want your characters to work. However, these lists are not fixed and can be extended via code. As Game Creator grows, so will the amount of options available. If you are a programmer you can create **Unit** that integrates a third-party character system. To know more about extending the Character component see the Character Controller section.

PLAYER

The **Player** unit controls how the character is controlled by the user. It only affects the character if its Is Player checkbox is enabled. **Game Creator** comes with a bunch of different **Player** units the user can choose from:

- Directional: The character moves relative to the main camera's direction and reacting to the keyboard's WASD keys or any Gamepad's Left Stick. This is the most common control scheme for most games.
- Point & Click: The character moves towards the point in space click with the mouse cursor. If the Driver is set to Navigation Agent, the character will try to reach the clicked position avoiding any obstacles along its path.
- Tank: Pressing the advance key will make the character move forward in their local space, regardless of the main camera orientation. This option requires the Tank option as its Rotation unit.

MOTION

The **Motion** unit defines a character's properties and what it can or can't do. It comes with a list of options that can be modified both in the editor and at runtime.



Game Creator comes with just a single Motion unit called Motion Controller. Unless the character is implementing a custom character controller, the Motion unit shouldn't be changed to anything else.

Character Component

These options are:

- · Speed: The maximum velocity at which the character can move. In Unity units per second.
- Rotation: The maximum angular speed at which the character can rotate. In degrees per second.
- Mass: The weight of the character. In kilograms.
- ${\boldsymbol{\cdot}}$ ${\boldsymbol{\mathsf{Height:}}}$ How tall the character is. In Unity units.
- Radius: The amount of space the character occupies around itself. In Unity units.
- $\boldsymbol{\cdot}$ $\boldsymbol{\text{Gravity:}}$ The pull force applied to the character that keeps it grounded.
- Terminal Velocity: The maximum speed reached by a character when falling.
- **Use Acceleration:** Determines if the character accelerates/decelerates when moving. If set to false, the character will start moving at full speed.
- · Acceleration: How fast the character increases its velocity until it reaches its maximum speed.
- · Deceleration: How fast the character decreases its velocity until it stops.
- · Can Jump: Determines if the character can execute a jump.
- Air Jumps: The number of double jumps the character can perform in mid-air. Most games allow zero or up to one air-jump.
- $\boldsymbol{\cdot}$ $\boldsymbol{\mathsf{Jump}}$ $\boldsymbol{\mathsf{Force}}\boldsymbol{\cdot}$ The vertical force used when executing a jump.
- Jump Cooldown: The minimum amount of time that needs to pass between each successive jump. Useful to prevent the user from spamming jumps.

The **Motion** unit also has the **Interaction** section at the bottom, which allows to configure how the character can interact with elements from the scene.

DRIVER

The **Driver** unit is responsible for translating the *math* of the processed motion data into actual movement. Depending on the controller type the character will move slightly different.

- Character Controller: The default unit. It uses Unity's default Character Controller which provides a versatile controller which should work fine for most cases.
- Navmesh Agent: It uses Unity's Navmesh Agent as the character controller. It allows to avoid obstacles when moving a character to a point in space but has the con that prevents the character from being able to jump.
- Rigidbody: It uses Unity's Rigidbody component so the character is affected by external forces using Unityis Physics Engine.

≪onometry Settings

Since version 2.9.36 the **Driver** unit comes with an *Axonometry* field that allows to post-process the character movement and constrain by some rules. These rules can be:

- **Side-Scroll XY**: The character can only move within the X axis and gravity affects the Y axis. Locked on zero in Z axis.
- **Side-Scroll YZ:** The character can only move within the Z axis and gravity affects the Y axis. Locked on zero in X axis.
- Isometric 8 Directions: The character can only move around the XZ plane in multiples of 45 degrees.
- Isometric Cardinal: The character can move north, south, east and west.
- Isometric Ordinal: The character can move in diagonals from world-space perspective.

ROTATION

The **Rotation** handles how the character rotates and its facing direction at any time. There are multiple **Units** available by default although the most common one is the **Pivot**.

- Pivot: The character rotates towards the direction it last moved to.
- **Pivot Delayed:** Very similar to **Pivot** but the character waits a few seconds before it starts rotating towards the direction it's moving. This option looks best for slow-paced movements, like walking slowly, sneaking or crawling.
- Look at Target: The character always faces towards an object in the scene and wil strafe when moving sideways relative to the object. This option is most used when locking onto enemies.
- Object Direction: The character faces the direction of another object. This is mostly used third and first person shooting games where the character must look straight towards where the camera aims so the weapon's direction is aligned with the camera's point of view.
- Towards Direction: The character faces a 3D world-space direction. Mostly used in games on-rails or infinite runners.
- Tank: The character pivots around itself when pressing the specified buttons.

Sitching at Runtime

It's important to highlight the fact that these options can be changed at runtime. For example, the player can use the **Pivot** unit when wandering the world but switch to a **Look at Target** unit when encountering an enemy. The character will seamlessly transition between them.

Ronometry Settings

Since version 2.9.36 the Rotation unit also comes with an Axonometry field that allows to post-process the character rotation and constrain it by some rules. These rules can be:

- Side-Scroll XY: The character can only look at right of left.
- Side-Scroll YZ: The character can only look forward or backwards.
- Isometric 8 Directions: The character can only rotate in multiples of 45 degrees.
- Isometric Cardinal: The character can rotate towards north, south, east and west.
- Isometric Ordinal: The character can rotate in diagonals from world-space perspective.

It's worth noting that both Driver and Facing Axonometry values should match for best results.

ANIMATION

The **Animation** unit controls how the character model moves as a reaction of any internal or external stimulus and also manages the representation of the character's 2D or 3D model.

Character Component

Just like the **Motion** unit, there is one single **Animation** unit option available called **Kinematic** which controls any generic character model's animations. There are different configuration blocks within the **Kinematic** animation unit:

- Position: Determines the local position of the mannequin inside the Character component. Rotation and Scale also change the transform of the mannequin in local space.
- Smooth Time: Determines how long it takes to transition between most character's animations, in seconds. Higher values make transitions look smoother but also take longer and feel less responsive. Lower values closer to zero make the character feel more responsive but also snappier.
- · Mannequin: A reference to the intermediate game object between the root Character and the 3D model.
- Animator: The Animator component of the character's 3D or 2D model.

Rantime Animator Controller

The character's model **Animator** component should use **Game Creator**'s Locomotion runtime animator controller or a custom controller that follows the same parameter names. To use a custom runtime animator controller it is necessary to implement a custom IAnimim unit (see Character Controller for more information).

- Start State: Optional field that allows to set an initial character State. The starting state is set to layer number -1.
- · Reaction: An optional field that determines the default hit reaction for Shooter and Melee modules.

Still pose animations

Combining the **breathing** and **twitching** systems allows using single-frame still poses feel like fully-fledged animations, thanks to the additive *breathing* and *twitching* animations. In fact, **Game Creator**'s default idle poses have a duration of a single frame. It's the twitching and breathing animations that make the pose look like it's real.

Extra Settings

The Character component has 3 extra sections at the bottom of the component which allow to control more specific parts of the character.

INVERSE KINEMATICS

Inverse Kinematics (IK for short) allow characters to change their bone rotations in order to transform the overall structure and reach with the tip a targeted position and rotation. A common use of Inverse Kinematics is making sure the character correctly align their feet to the steepness of the terrain.

Character IK

Game Creator allows to dynamically add or remove new IK systems onto each character individually and are processed from top to bottom. To add a new IK system simply click onto the "Add IK Rig Layer" button and select the option you want from the list.

custom IK Rigs

You can also create your own custom IK systems. Check out the Custom IK section for more information.

The Character component comes with some common IK systems used on most games:

- Look at Target: This IK system allows characters to slightly rotate their head, neck, chest and spine chain in order to look at a specific point of interest. This is specially useful when paired with the Hotspots component. Requires the character model to be *Humanoid*.
- Align Feet with Ground: This IK system allows a character to automatically detect when the character is touching the ground and smoothly align their feet with the inclination of the ground. It can also lower the position of the hip so both feet touch the ground, in case the ground is very steep and one foot is higher than the other.

FOOTSTEPS

The Footstep system allows the character to signal when it has performed a step. This is useful when you want a character to leave a trail of footprints, play some particle effects simulating the dust of each step or playing a sound effect.

Character Footsteps

Rumanoid and Generic characters

The Footstep system doesn't require the character model to be humanoid. It uses an array of objects that identify the character's feet bones. By default it assumes the character is a human and has two feet, but this can be easily customized clicking on the "Add Foot" button.

• The **Sound Asset** field references a *Footstep Sounds* asset that determines which textures play which sound effects. For more information about how to configure this asset see Footstep Sounds section.

Mysically accurate sounds

The Footstep Sounds does not play the raw step sound effect but automatically distorts it in order for the player to hear different slightly different sounds each time. It also changes the pitch of the sound if there are multiple layers of textures, muffling those that are less prominent.

Custom Feet Phases

A **Character**'s footsteps are determined by the feet phases (when the character touches ground with their feet). These values are driven by animation curves. If you want to use custom animations, you can download and use for free a custom tool for assigning feet phases clicking here.

RAGDOLL

The Character component comes with a built-in Ragdoll physics system that allows to quickly turn any character into an inanimate object that reacts to physics with a set of constraints on each of its limbs.

Character Ragdoll



The **Ragdoll** system uses the **Skeleton** configuration asset to determine which parts of the model correspond to which bone. It can't work without one.

• Transition Duration: When a character recovers from a ragdoll state, it plays an animation based on the direction its body faces. This field determines the time it takes to blend between the ragdoll position to the animation clip being played when recovering.

eve plenty of transition time

It is recommended to use large transition values, above 0.5 seconds. The character's limbs can be in very awkward positions that doesn't match the initial pose of the recovery animation clip; so having small transitions will make the character appear to snap into an animation, instead of smoothly blending into it.

- Recover Face Down: The recovery animation played when the root of the character's ragdoll faces downwards.
- · Recover Face Up: The recovery animation played when the root of the character's ragdoll faces upwards.

For more information check its dedicated Ragdoll section.

1.3.3 Animation

Animation

Game Creator has a built-in custom animation system built on top of Unity's Mecanim that makes it easier and faster to manage character animations.

It introduces the concept of **Gestures** and **States**, which are two mechanisms that allow to play different types of animations without having to previously register them inside an Animator Controller graph.

Mecanim vs Gestures & States

It is preferable that users use the **Gestures** and **States** system to manage and play all their animations. However if a user prefers to use a more traditional approach, there's a base Mecanim layer that allows to use Unity's runtime controller workflow. Check the Animator section to know more about this.

Animation Flow

Gestures are animations that are played once and are removed from the animation graph when finished. For example, an animation of a character throwing a punch can be played as a *Gesture*; This will make a character play the *punch* animation and smoothly restore its previous animation after the animation finishes.

States are animations that are played on a repeating loop. For example, a character sitting on a chair is an *Animation State* while a character moving crouched is a *Locomotion State*.

- · Animation States play a single animation clip over and over again, until told to stop.
- Locomotion States are more complex states that react to certain parameters such as character speed. Can have multiple clips transitioning and blending with each other.

Click on $\ensuremath{\mathsf{Gestures}}$ and $\ensuremath{\mathsf{States}}$ to know more about how to use them in your game.

Animator

Character components reference a child game object called the *Model* which contains an Animator component. This component must referece a Runtime Animator Controller graph, that determines which animations are played when and how these transition between them.

CUSTOM MODEL

Game Creator makes it very easy to change the 2D or 3D model from a character. All that needs to be done is to open the **Animation** section of the Character component and drag and drop the Character prefab onto the indicated drop zone.

Change Character Model

changing model at runtime

To change the character model at runtime use the Change Model instruction.

LOCOMOTION RUNTIME ANIMATOR

Game Creator comes with a default **Runtime Animator Controller** called the *Locomotion* controller. It comes packed with a collection of animations and features that fit most projects.

Changing the Locomotion controller

It is not recommended modifying the Locomotion controller. In most cases using a custom State is easier and provides enough flexibility to create new simple or complex locomotion animations.

However if you need to use a custom **Runtime Animator Controller** you must also creata new class that implements the <code>IAnimim</code> interface to feed the Character's data onto your custom controller. See Character Controller section for more information.

Gestures

The **Gesture** system allows characters to play a single animation that stops after it finishes. This is specially useful for animations such as a character throwing a punch, vaulting an obstacle or waving a hand.

These animations are always played on top of any other animations.

Character Gesture Waving

PARAMETERS

The easiest way to play a **Gesture** animation is using the **Play Gesture** instruction, which has a few configuration parameters.

Character Play Gesture instruction



It may seem a bit overwhelming the amount of parameters available for a single animation. Note that the most important ones are the **Character** and **Animation Clip** fields. The rest can be left with their default values and should work on most cases.

Character

The **Character** field determines the object that the animation clip will be played. The game object referenced must contain a Character component in order to work. Otherwise the instruction will be skipped.

Animation Clip

The Animation Clip references an animation asset. Without this field the instruction will not work.

Avatar Mask

The **Avatar Mask** is an optional field that determines which parts of a character will play the animation and which won't. If this field is left empty the whole body will play the animation. For more information about masking animations, see the Unity documentation about **Avatar Masks**.

Blend Mode

The **Blend Mode** field determines whether the animation clip overrides or adds up its movement on top of any other animations being played.

- **Blend:** The default parameter. Blend overrides any animations and plays the animation clip on top of them. This is the most common option for most animations.
- Additive: This blend mode allows to play an animation by adding up the motion on top of any other clips being played.

Delay

The **Delay** field allows to start playing the animation after a certain amount of seconds have passed. If the value is set to zero the animation will start to play immediatelly.

Speed

The **Speed** field is a coefficient that determines the speed at which the animation is played. A value of 1 plays the animation at its original speed. Higher values will play the animation faster while lower ones will play the animation slower. For example a value of 2 will play the animation twice as fast.

Root Motion

Determines whether this animation should take control over the character and use its root motion to also move and rotate it. Notice that using *root motion* takes control of the character while the animation plays and the user's input will be ignored.

Transitions

The **Transition In** field determines the amount of seconds the animation will take to blend between the current animation and the new Gesture animation clip.

Animation Gesture Transition

Similarly, the **Transition Out** field determines how much time, in seconds, it takes to blend out the current gesture animation to the animation being played underneath.

Wait to Complete

The **Wait to Complete** checkbox allows the instruction to be put on hold and only continue once the animation finishes. This is specially useful when chaining multiple gestures one after another.



For more information about how to use instructions to interact with other systems, see the Visual Scripting section.

States

The **States** system allows to dynamically blend in/out arbitrary animations or entire animator controllers at runtime. All that needs to be done is to specify which animation or controller a character should play, and which layer should it be assigned to.

Character Animation State Asset



It is important to note that the **States** system is built on top of Unity's Mecanim and it complements it; It does not prevent or restrict from using any of its features. It simply adds a new and more flexible workflow on top of it.

TYPES OF STATES

There are primarily two types of States, but both work the same way: An instruction feeds a State to a Character and this one plays the animation/s based on the behavior of the State.

Animation States

Animation States are single animation clips that are played over and over again, until told to stop and blend out.

For example a character playing a single looped animation of sitting on a chair is an *Animation State*. These are the most common and basic forms of **States**, where an **Animation Clip** must be provided and the Character plays it in a loop.

It is also possible to create an *Animation State* asset that allows to play a looped animation as well as providing a fields for gestures that are played when entering and exiting the State. To do so, right click on the Project Panel and select *Create Game Creator Characters Animation State* and drop the Animation Clip file onto the corresponding field.

Character Animation State Asset

The **State Clip** field determines which animation is played in a loop, while **State Mask** discerns which body parts are affected by the animation. Note that this last field only works with Humanoid characters. See **Avatar Mask** for for information about masking animations.

The **Entry** and **Exit** sections contain optional fields that allow to play a **Gesture** right before entering or exiting the current State. For example, you may want a character to play the *unsheathe* sword animation every time it enters a sword combat stance, and play the *sheathe* animation when exiting the combat stance state.

Refresh Instructions

Since version 2.5.20 there's an **Instruction** list at the bottom of any **State** asset called **On Refresh**. These instructions are called in order, from the lowest *Layer* to the upper-most one, any time a Character adds or removes a **State**.

Character Animation State Asset



Since version 2.9.34 each State contains a Properties section that allows to modify common values from a Character, such as its linear and angular speed, jump options, gravity, etc...

Character Animation State Asset

changing Movement Speed

This allows to change the move speed from within the **State** itself. For example, let's say we have the following States:

- Running **State** at layer 1 that sets the player's speed to 10
- Walking State at layer 2 that sets the player's speed to 5

The "Running" State properties will be called first, and afterwards the "Walking" State, because it's on a greater layer number. The second State will override the player's movement speed and set it to 5.

Removing the Walking State will run again the Properties values. However, this time, only the **Running** State properties are called, and thus the player's speed will be set to 10.

Locomotion States

These are more complex States that react to certain parameters such as the speed of a character, its direction and fall velocity. Locomotion States have multiple clips transitioning and blending with each other.

For example a character that idles in a prone position and crawls when the character moves is a *Locomotion State*.

To create a Locomotion State, right click anywhere on the *Project Panel* and select *Create Game Creator Characters* Locomotion Basic State or Create Game Creator Characters Locomotion Complete State.

Character Locomotion State Asset

The **Locomotion State** asset may seem a bit daunting at first, but it's fairly straight forward. There are two types of **Locomotion States** and those are:

- · Basic States: Have an idle and an 8-axis directional animation clip fields for moving
- Complete States: Have an idle and a 16-axis directional animation clip fields for moving: 8 for moving at half speed and another 8 for moving at full speed.

The first fields, **Airborne Mode**, controls the amount of animation clips available and can take one of the following values:

- · Single: Displays a single animation clip for that particular phase.
- Circular 8 Point: Displays animation clip fields for the 8 cardinal directions: Forward, Backwards, Right, Left and each of the diagonals.
- Circular 16 Points: Displays animation clip fields for the 8 cardinal directions, and another 8 for half-way points between the first and the origin.

Points vs 16 Points

This decision comes down to the type of controller and animations available. If your game is meant to have analogic controls, the user might slightly push the movement joystick forward, making the character move slow. In this case, it is recommended using the **Complete Locomotion State**, as it allows to have both running and walking animations in a single State.

LAYERS

The **States** system is built around the concept of *Layers*, which is similar to the concept found in image editing tools, such as *Photoshop*. The idea is that any **State** is assigned a layer number. With higher numbers taking higher priority when playing an animation.

Example

Let's say we have a character with three Layers, each one with a single State, numbered 1, 2 and 10 respectively.

Character States Layer

In this case, the animation played would be be the one found at the layer number 10. However, if this layer was to be removed, the animation at layer 2 would be the next one with highest priority and thus, its State would be played.

It is recommended to add a transition time when adding or removing a **State** from a *Layer* in order to smoothly blend between the new animation and the one underneath.

Animation Gesture Transition

When adding a new **State** onto a *Layer* that already has a **State**, this last one will be smoothly faded out taking into account the new **State**'s transition time, until it is replaced by the new one. After that happens, it will be automatically disposed.

destures and States

Note that although **States** can have different priorities, a **Gesture** animation will always have higher priority than any **State** and will play on top of it.

WEIGHTS

Setting a new State is not an all-or-nothing operation and the new animation can be blended by a percetage with any other animations playing underneath the stack.

For example, if a character is currently playing a running upstraight animation, a running crouched animation can be blended at 50% to to make the character look like it's running halfway between standing and crouched.

Weight at runtime

The weight can be modified at runtime using the Change State Weight instruction.

ENTERING A STATE

The easiest way to make a character enter an Animation or Locomotion State is using the Enter State Instruction.

Instruction Enter State

The **Character** field references the targeted character game object that enters the state. The **State Type** field determines whether the State is an *Animation Clip*, a *State* asset or a Runtime Animation Controller.

kuntime Animation Controller as a State

Game Creator allows to use a Runtime Animation Controller as a State. However, this is an advanced feature and should only be used if one understands how Gestures & States work under the hood.

The Layer field allows to determine which layer this State occupies in the Character's layer stack. Blend Mode by default is set to *Blend*, which overrides the underlying animation with the animations provided by the State. If set to *Additive* it adds up the new State's animation as a delta movement on top of any other animation being played.

The **Delay** field allows to delay in a few seconds the time to start playing the State. **Speed** is a coefficient value that determines how fast the State plays. For example, a value of 1 makes the State play its animation at its default speed. A value of 0.5 plays the animation at half speed and a value of 2 plays it twice as fast.

The **Weight** field determines the opacity of the State. A value of 1 plays the animation as it is. Lower values allow any previous animations to bleed through and mix the effect between the new State and any other animation being played in lower layers.

The Transition field is the time in seconds that the new State takes to fade in.

EXITING A STATE

The instruction Stop State can be used to smoothly stop playing a State on a character.

Instruction Enter State

The **Character** field determines the targeted game object that stops playing a State found at the layer identified by the **Layer** number field.

Similarly, the **Delay** and **Transition** fields allow to delay the fading of the State by a certain amount of seconds.

1.3.4 Inverse Kinematics

Inverse Kinematics

Inverse Kinematics (IK for short) is the process of calculating the rotation of bones from a chain of bones, in order for the leading one to reach a desired position. **Game Creator** makes use of both limbic and full-body IK.

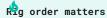
Character Feet IK

A common case scenario is adjusting the bending of the knees so the character naturally plants its feet on the ground.

MANAGE IK RIGS

The Character component has a section at the bottom that allows to manage which rigs affect the character and change their properties.

Character Feet IK



The IK Rigs are excuted from top to bottom. So if two IK systems affect the same bone chains, the last rig will override any previous ones.

To add a new Rig, click on the Add IK Rig button and choose one from the dropdown list.

RIGS

Game Creator comes with a few IK rigs that work out of the box:

- Feet Align: Allows to align a Character's feet to uneven terrain.
- $\boldsymbol{\cdot}$ \boldsymbol{Look} at $\boldsymbol{Target}\colon$ Allows a Character to use the Look At system from Hotspots.

Feet Align

This **IK Rig** allows a character to plant their feet and adjust the rotation on uneven terrain. This rig also allows the hips to be lowered by a certain amount if the height difference between both feet is very large.

Character Feet IK



The Feet Align rig only works with Humanoid characters.

Character Feet Align

The Feet Align rig has the follow options:

- Foot Offset: An optional vertical offset applied to each foot. This is useful in cases where the foot penetrates the ground or floats above it, due to differences between the bone's tip position and skin mesh bounds
- Foot Mask: Allows to choose which Layers should the character consider when aligning with ground. For example, water typically has a collider component, but the character should not align its feet on its surface.

Look at Target

The **Look at Target** rig allows a character to rotate their head, neck, chest and body in order to look at a Hotspot.

Character Feet IK



The \boldsymbol{Look} at \boldsymbol{Target} rig only works with Humanoid characters.

Character Look at Target

The **Look at Target** rig has the follow options:

- Track Speed: The angular speed at which each bone rotates to track the target. In degrees per second.
- Max Angle: The maximum peripheral angle, in degrees.
- · Head Weight: The contribution of the head to the total rotation.
- Neck Weight: The contribution of the neck to the total rotation.
- Chest Weight: The contribution of the chest to the total rotation. Note that the Chest is an optional bone and some models may not have it.
- Spine Weight: The contribution of the spine bone to the total rotation.

Default values

The default parameters have been carefully picked to work for the majority of human-like characters.

1.3.5 Footstep Sounds

Game Creator's characters can mix and play multiple sound effects depending on the type of ground it's stepping on.

Character Footstep Sounds



This system works for humanoid and non-humanoid characters alike. Though humanoids don't require any kind of setup and work out of the box.

Detecting Steps

The Footstep System (also known as **Phases** system) uses **Animation** Curves to detect when a **Character** has one of its limbs in contact with the ground and when it does not.

This system plays a role in other systems, such as correctly aligning the feet when standing on uneven terrain, or detecting when the character takes a step, and plays a tiny dust particle and sound effect.

The Phases system supports up to 4 different phases, although humanoids only require 2 (one for reach leg).

By default, a Humanoid character has the following curve names assigned to each leg:

- · Phase-0 to the Left Leg
- · Phase-1 to the Right Leg

A non-humanoid character can also define the Phase-2 and Phase-3 if necessary.

Using custom animations

Game Creator animations contain the phase curves already set up for you. However, if you plan on using your own animations, you'll need to set them up by editing the Animation Clip and adding the Phase-0 and Phase-1 curves.

Character Footstep Phases Animation Curves

The phase curves are evaluated at runtime depending on the animation(s) being played at that time. If the value of a phase is **zero** means the limb is currently not touching the ground and is high up in the air. On the other hand, if the curve has a value of **one**, it means the limb is currently planted on the ground.

Playing Footstep Sounds

The **Footstep Sounds** system comes with a built-in tool for playing different sounds and sound variations depending on the surface the character is stepping onto. To create a material sound library, right click on the *Project Panel* and select Create - Game Creator - Common - Material Sounds.

Character Footstep Reaction

The **Material Sounds** asset allows to define which textures produce which sound effects. Each texture can have multiple sound effects, which will be picked up randomly every time the character takes a step.

Seudo-Random Sound Picking

Note that although it's completely random, two sound effects will never be played in succession in order to avoid repetition.

The **Material Sounds** asset also allows to instantiate a game object from a pool of prefabs at the impact position. The instantiated object is aligned with the incision angle. This is very useful when spawning particle effects of dust.

The human hearing quickly recognizes sound patterns. To avoid hearing the same sound effects over and over again, the Footstep Sound System intelligently shifts the pitch and speed of each audio clip every time it's played. By doing so, a single clip can be played hundreds of times with various nuances that tricks the human hearing into perceiving each clip as a different sound effect.

dadient Footstep Sounds

Floors are not always composed of discreet materials. For example, there might be a sound effect for when the player steps on shallow water and another one when steps on sand. However, if the character runs along the shore, where there's a blend between the water and sand textures, the resulting sound effect is a proportional mix between the two audio clips and their pitch is shifted to fit how real-life audio blending occur.

Drop the Material Sounds asset onto the Character's Sound Asset to link them.

Reacting to Footsteps

The **Footstep** system also allows Characters to react every time a step is taken. Using the **On Step** Trigger, which is executed every time a defined Character takes a step. This is useful for things like leaving footsteps behind.

Character Footstep Reaction

1.3.6 Ragdoll

Ragdoll

A *Ragdoll* system lets characters react to physics and external forces without any direct input from itself. This is commonly used for enemies that have been defeated or when the player falls unconscious due to a strong attack or a big fall.

Character Ragdoll

A **Character** requires a **Skeleton** definition asset in order to correctly identify the size of each of its bones and how they form the joint connection chain.

dickly generate a Skeleton

Defining all **Skeleton** volumes and how these relate to their parent bones is tedious and time consuming process. Luckily **Game Creator** makes it very easy to automatically generate a humanoid Ragdoll asset. With the Skeleton asset selected, drag and drop any *Humanoid* 3D model onto the bottom drop-zone and it will generate the structure for you. You can then tweak the values to perfectly match your model.

STARTING AND STOPPING

To initiate a ragdoll state, simply use the *Instruction* **Start Ragdoll** and select the targeted character. Notice that the player's input will still be in effect though. This is why Game Creator's default character comes with 2 Triggers that make it even easier to handle Ragdolls: When a character is considered to be *dead* it will automatically trigger the *Start Ragdoll* instruction on the character. When a character is revived, it will also automatically handle playing the correct animation and get the character up from the floor.

This means that, in order to start and stop the ragdoll effects, all that needs to be done is to use the *Instruction* **Kill Character** to disable any interactions from a character and it will automatically enter ragdoll-mode. On the other hand, using the **Revive Character** *Instruction* will give back control to the character and get it up from the floor using the correct animation.



The character will automatically handle transitioning from its ragdoll pose to the default idle animation and pick up the most suitable gesture, depending on whether its currently facing down or up.

CONFIGURE RAGDOLL ANIMATIONS

To setup the *getting up* animations, select the Character and drag and drop the desired animations onto the **Recover Face Down** and **Recover Face Up** clip fields.

Ragdoll Animations Setup

The **Transition Duration** field allows to specify the duration between the time the character is not controllable due to being in ragdoll-mode and recovered. Ideally this value will be a few milliseconds shorter than both recover animations.

The most important part of a ragdoll is knowing the length and size of each of its physical bones and how they interact with the rest of the body. This is done using the Skeleton asset file. To know more about configuring a Skeleton asset and associate it with a Character, see the Skeleton section.

Skeleton

A **Skeleton** asset is a scriptable object asset that contains all the necessary information to identify the bounding volume of a character's bones and how these form a chain of joints that conforms the whole body.



The **Skeleton** asset is used on multiple systems, such as the Ragdoll system, or the Melee and Shooter hit detection systems.

Link Skeleton to Character

CREATE A SKELETON

To Create a Skeleton asset, right click on the *Project Panel* and select **Create Game Creator Characters Skeleton**.

To assign a **Skeleton** asset to a **Character** simply select the desired **Character** and expand the *Animation* tab. Drag and drop the **Skeleton** asset onto its corresponding field.

Link Skeleton to Character

CONFIGURE SKELETON

The **Skeleton** asset is divided into different sections:

The first is a big button that allows to enter *Skeleton Configuration* mode. In this mode, the scene is replaced by an empty one with a character in the middle, which can be changed by dragging and dropping a prefab model onto the field below and clicking on the *Change Character* button.

The second section determines the Physical Material and collision detection mode of the rigidbody system stemmed from the volumes.

At the bottom there's a list of all volumes set up. This list can be either manually configured or use the heuristic creator for humanoid characters.



To more easily configure the volumetric bounds of a humanoid character, see the next section.

Configure Skeleton

To create a volumetric bone, click on Add Volume and select the type of bone to create:

- Box: A cubic volume. Mostly used for chest and flat surfaces.
- Sphere: A spherical volume. Used for hands and head mostly.
- Capsule: The most widely used volume bone. Used for most limbs.

Configure Skeleton

A Volumetric Bone is composed of a Bone Type, a volume definition and an optional Joint.

The bone type can be specified by setting the humanoid bone from a dropdown list or from a path. For example, to reference the front right foot of a model of a Dog, the bone could be Root/Spine/Collar/Right_Leg/Right_Foot.

The volume definition depends on the type of volume created. For example, a Sphere volume bone contains a radius and a position offset field.

The Joint field allows to determine how a bone is related to other bones via a joint system..

Nore on Joints

For more information about character joints, visit this Unity documentation link.

SETTING UP A HUMANOID SKELETON

Game Creator comes with a tool that makes it much easier to automatically *guess* and extract the bounding volumes of a humanoid model. To use it, simply change the character model using the *Change Character* button and click on the *Create Humanoid* button. It will auto-magically approximate a Skeleton for you that you can then tweak it to your game needs.

Configure Skeleton

1.3.7 Markers

A Marker is a component that is used by Characters as destination points. It allows to define a target position and rotation so the Character is at the correct location before doing something else, like opening a door.

Marker Gizmo in Scene

A Marker has a yellow shaped arrow that indicates the direction the Character will face after moving towards it.

Marker Gizmo in Scene

Optionally, a Marker can specify a **Stop Distance** threshold from which a Character is considered to have reached its destination.

By default it's zero, but if the destination is a very crowded, there might not be enough space for a character to be at the exact marker's position. Having some error threshold allows Characters to more or less reach their destination without getting stuck or pushing other characters around.

The **Type** field allows to determine how the Marker works. By default its set to *Directional* which forces the character to end at the same position and rotation as the arror-shaped gizmo in the scene.

Another available mode is *Inwards* which tells the character to move to the closest point around a circle and rotate towards its center. This is specially useful when you want the character to pick up an item and you don't care from which angle it is picked up.

1.3.8 Interaction

Game Creator comes with a built-in interaction system that lets characters (both Players and NPCs) dynamically focus on a scene element and decide whether to interact with it or not.



It's important to note that a Character that is Busy cannot interact with Interactive elements.

Character setup

How a Character interacts with scene objects is specified in the Motion unit.

Character Interaction

The **Radius** option determines the minimum distance an object has to be in order for the character to focus on it.

The Mode option allows to determine how to prioritize how objects are focused:

- Near Character: Picks the closes object to the character's interaction center, which can be offset by a certain amount. This option is best for console and games that require a controller.
- Screen Center: Interactive objects closer to the center of the screen have higher priority. This is the best option for first person games.
- Screen Cursor: Interactive objects closer to the cursor take precendence. This option is best for point and click adventures.



The character will automatically focus and unfocus any interactive object. To interact with the currently focused object, use the **Interact** instruction.

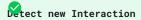
Interactive Objects

Any game object with the **On Interact** event on a **Trigger** component will be automatically marked as an interactive one.

This event will be fired every time a character attempts to interact with this trigger.

Trigger On Interact

If a character attemps to interact, but there is no *Interactive* object available, it will simply ignore the call.



Apart from the On Interact event, one can also detect when a Trigger becomes focused or loses focus (also known as blur). This can be tracked using the On Focus and On Blur events.

Hotspots can also display a text or activate a prefab when the game object is focused by a character. To do so, you can add the **Text on Focus** spot on a **Hotspot** component and it will display the chosen text every time the selected character focuses on this interactive element.

Hotspot Interactive has Focus

1.3.9 Busy

The Busy feature allows to query whether a specific limb of the character is being used or not. This allows other systems to determine whether an action can be performed or not.

ising the right hand

For example, a character that is shooting with its right hand can set its right arm as *busy*. By doing so you can prevent the character from opening a door with the right hand until the right arm is available again.

Busy at Runtime

When entering Play-Mode the mannequin icon at the top of the Character component will change its color from grey to green.

Character Busy at Runtime

Its color changes in real-time and indicates:

- Green: The Limb is available to use.
- Red: The Limb is currently busy.

You can use Instructions and Conditions to set and retrieve the current Busy status of a Character.

Scripting

ding Knowledge

The following concepts are meant for experienced programmers.

The follow properties can be queried and inform of the availability state of the limb or group of limbs:

IsArmLeftBusy : boolean
IsArmRightBusy : boolean
IsLegLeftBusy : boolean
IsLegRightBusy : boolean
AreArmsBusy : boolean
AreLegsBusy : boolean
IsBusy : boolean

Additionally, limbs can be marked as busy or make them available using the MakeLimbXXX() method, where XXX is the limb of the body. For example, to set the Left Leg as busy, call the MakeLegLeftBusy() method.

available methods

For more information about all the available methods on the Busy system, check the script under Plugins/GameCreator/Packages/Core/Runtime/Characters/Busy.

1.3.10 Handles

Handles are an optional asset that can be used to determine the bone where a prop is attached to, and it's precise position and rotation.

Handles for Props

The **Handles** asset has a list that checks its **Conditions** from top to bottom. These conditions can determine which location of the handle should be the most optimal.

Handles for Humans and Beasts

For example, you could have a condition that checks if Self is a humanoid character or not. If it is, the prop could be attached to the right hand (like a sword). Otherwise the prop would be attached on the beast's mouth.

Handles also help re-use the same position and rotation for multiple weapons, which comes in handy if a game has lots of props to equip, such as swords, shields, daggers, etc...

1.3.11 Scripting

Scripting

This section covers topics that require some degree of programming knowledge and assumes certain level of coding expertise.

- $\boldsymbol{\cdot}$ $\boldsymbol{\textbf{Character:}}$ How to customize and extend the character system.
- Inverse Kinematics: How to construct new inverse kinematic character rigs.

Character

Game Creator Characters have been build to be easy to use and highly customizable. This section go over what a Character does every frame cycle. This will put you in perspective in order to create a custom Character that works with Game Creator or you want to integrate a Character system from another package into Game Creator.

KFRNFI

The **Character** component is composed of 5 different **Units** which conform the **Kernel**. These units can be changed at runtime without affecting the rest:

- Player: Defines whether the Character is a playable one and how the user can interact with it. If you want to create a custom Character input system, you'll need to implements the IUnitPlayer interface.
- Motion: Acts as an interface between the scene and the Character. All movement commands are relayed through this system and also takes into account the *Player*'s information. It decides which locomotion system should be used. If you want to create a different motion system for your characters, create a class that implements the IUnitMotion interface.
- **Driver:** Manages how the Character moves around the scene based on the *Motion*'s input. If you want to integrate another Character system from another Asset Store package, create a new class that implements from TUnitDriver.
- Facing: Is responsible for rotating the character towards a desired direction. For example, the default behavior is to have the character look towards where it's moving. If you want to customize where the character faces, create a custom class that implements the IUnitFacing interface.
- Animim: This system takes the *Driver*'s input and tells the Animator component which animation should be played via Mecanim parameters. If you want to use a custom Animator for your Character, crete a class that implements IUnitAnimim interface.

Every new cycle tick the Character updates all these systems in a very specific order.

Character Cycle

It starts by calling the **Player**'s system <code>Update()</code> method. This takes the user's input and calls one of the <code>Motion</code>'s public movement methods:

- MoveToDirection()
- MoveToPosition()

After the **Player**'s system has been processed, the **Character** calls the **Motion** system's Update() method. This is where external forces are calculated, such as gravity, sliding through slopes, dashing, jumping, ...

Communication between systems

The **Motion** system takes into account the **Player**'s system before running the update. A system can access any of the other's systems data before processing its <code>Update()</code> cycle.

After the final **Motion** movement is calculated, the **Character** executes the **Driver**'s <code>Update()</code> method. This is where the *Transform* component is updated based on the movement type provided by the **Motion** parameter.

After the **Driver** system is completed, the **Facing** system starts. Based on the information provided by the **Driver** and **Motion** systems it calculates the direction in which the Character should be facing at.

Finally, the **Character** system calls the **Animim**'s <code>Update()</code> method, which feeds the **Animator** component with the necessary parameter values based on the information of the rest of the systems.

Mdular design

It is important to highlight the fact that each system is independent of the other. You can create a custom animation system by implementing a <code>IUnitAnimim</code> interface and still use the default <code>Player</code>, <code>Motion</code> and <code>Driver</code> systems.

Plaver

The **Player** unit handles how the user interacts with the Player character. If the Character does not have the Is Player field checked, this unit is skipped entirely.

The Player also contains the IsControllable flag that defines whether a character processes the input received or not. This is very useful when a character is in the middle of a cutscene and you don't want the user to have control over the player.

Motion

The **Motion** unit is the brain of the character. It contains all of its quirks, such as its height, its move speed, terminal velocity and so.

The Motion unit also is in charge of receiving any locomotion commands:

- MoveToDirection defines a direction towards where the character must go. This method has to be called every frame or the character will stop.
- StopToDirection stops the character's movement. Useful when the character moves due to its deceleration value.

A character can also be instructed to move to a certain position:

- MoveToLocation instructs a character to move to a specific location. The Location class accepts a position and/or a rotation.
- MoveToTransform instructs the character to move to a specific transform's position. If the transform changes its position, the character will follow it until it reaches the target.
- MoveToMarker is similar to the previous method, but also takes into account the marker's rotation and forces the character to end facing the same direction as the navigation marker.

A character can also follow another target without an end condition:

- · StartFollowingTarget starts following a target and stays within a minRadius and maxRadius distance.
- StopFollowingTarget instructs a character to stop following a target.

The **Motion** unit is also responsible for dealing with character's jumps. The Jump() method will instruct a character to perform a jump (or air jump), if it's possible.

Drive

The **Driver** unit controls *how* a character moves around the scene: Whether it's using Unity's Character Controller, the Navigation Mesh Agent for obstacle avoidance or a physics-based rigidbody entity.

This unit recieves the locomotion information of *Motion* and *Facing*, and transforms it into a physical translation and rotation.

Facing

The **Facing** unit controls where the body of the character (not the head) points at. By default all characters do not rotate their body unless they are moving; in which case the body rotates towards where the character is moving.

However, there are certain situations where the character might want to temporary face at a certain direction. For example, when the character aims with the gun at a certain object, or when talking to a character. **Game**Creator comes with a layer system that provides a neat solution for these cases.

kecommendation

If you plan on creating your own facing system, we recommend creating a class that inherits from TUnitFacing instead of the interface IUnitFacing. This base class comes with the layer system built out of the box, so you don't have to recode it.

The Facing system interfaces provides access to 3 methods:

- int SetLayerDirection(int key, Vector3 direction, bool autoDestroyOnReach)
- int SetLayerTarget(int key, Transform target)
- void DeleteLayer(int key)

The first two methods, SetPlayerDirection and SetLayerTarget allow to make the character look at a certain direction or keep track of a particular scene object. Making the character change its default direction is done using a layer system.

When any of these methods is called for the first time, it creates a new entry in the layer system and returns its identifier: an integer known as key. To subsequently update a particular layer, simply pass as the key argument the resulting key from the previous iteration.

For example, if you want to make a character look at a certain character (defined by the variable lookAtTransform), you'll simply need to call:

```
private int key = -1;
public Character character;
public Transform lookAtTransform;

public void StartFacing()
{
    IUnitFacing face = this.character.Facing.Current;
    this.key = face.SetLayerTarget(this.key, this.lookAtTransform, false);
}

public void StopFacing()
{
    IUnitFacing face = this.character.Facing.Current;
    face.DeleteLayer(this.key);
}
```

Exceptions

It is important to note that the layer system won't throw any exceptions. If you try to attempt to delete a layer but the key doesn't exist, it will simply do nothing.

When calling the <code>StartFacing()</code> method, the character will smoothly rotate towards the target defined until the <code>StopFacing()</code> method is called.

However, in some cases, you may not want to manually remove the facing layer, but instead stop facing a particular direction when the character reaches its target direction. For these cases, simply set the SetLayerDirection method's last parameter to true. This will tell Game Creator to automatically remove the layer when the character reaches its target direction.

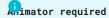
For example:

```
public Character character;
public Vector3 direction;

public void LookAt()
{
    IUnitFacing face = this.character.Facing.Current;
    face.SetLayerDirection(-1, this.direction, true);
}
```

Animim

The **Animim** unit handles everything related to the visual representation of a character: From its appearance to its animations.



This unit requires an Animator component reference in order to deal with animations

The default character system comes with a set of procedural animations played on top that add subtle but consistent movement across different animations, such as breathing and exertion. The breathing rate and exertion amount can be modified using the HeartRate, Exertion and Twitching proprerties.

CHANGE MODEL

To change a character model, call the ChangeModel(...) method. Its signature contains 2 parameters:

- \cdot A prefab object reference, which should be the FBX model
- A configuration struct of type ChangeOptions

This last optional parameter allows to define the new model's footstep sounds, its skeleton's bounding volumes as well as a new animator controller and an offset. For example, to change the player's model without any optional parameters:

GameObject instance = character.ChangeModel(prefab, default);

Custom IK

Characters in **Game Creator** have a layered *Inverse Kinematic* system that can be stack one after another in order to modify the animation of a character. The most common form of inverse kinematics is the Feet IK, which makes sure a character's feet are correctly placed and aligned with the floor below it.

ACCESSING A RIG

Accessing a rig is done using the IK property of the Character's component. To deactivate the rig that aligns the feet on the ground, for example, can be done using:

```
character.IK.GetRig<RigFeetPlant>().IsActive = false;
```

Note that character.IK.GetRig<RigFeetPlant>() returns an instance of that particular rig (null if it can't be found).

CREATING A CUSTOM RIG

Game Creator offsers two types of IK system wrappers:

- Riggings powered by DOTS
- Riggings powered by the AnimatorIK method

To create a new IK system you must crete a class that inherits from either TRigAnimationRigging (for DOTS) or TRigAnimatorIK (for AnimatorIK). We recommend using the new DOTS-based approach when possible, as it's more performant.

In either case, you should override the DoStartup(...) and DoUpdate(...) methods, which are called once at the beginning and every frame respectively.

```
public class MyCustomRig : TRigAnimationRigging
{
    protected override bool DoStartup(Character character)
    { }
    protected override bool DoEnable(Character character)
    { }
    protected override bool DoDisable(Character character)
    { }
    protected override bool DoUpdate(Character character)
    { }
}
```

1.4 Cameras

1.4.1 Cameras

Cameras are devices that capture and display the world to the user. **Game Creator** uses two components to determine how the action is framed:

- Camera Controllers: A component attached to the camera. For itself it does nothing but mimic the behavior that its active camera shot feeds. By default, the Main Camera component is the primary camera controller.
- Camera Shot: A component that has multiple configurations, depending on which, its associated camera controller will respond in one way or another.

For example, if the camera controller Main Camera has the *Third Person* Shot associated with it, the main camera will mimic the behavior of that shot, which is to follow and look at a target, while the user can orbit around it.

A camera controller can transition to another camera shot. This transition can either happen over time, or instantly.

1.4.2 Camera Controller

A Camera Controller is a component attached to a camera object that has a associated at most one Camera Shot reference. This associated camera shot can be changed at runtime and will dictate the behavior of the camera controller.



Most games will only have one single camera. The camera in these cases will have the Main Camera component attached, which is a camera controller that can be accessed globally by any script.

Creating the Main Camera

To creata a main camera, right click on the *Hierarchy Panel* and select Game Creator Cameras Main Camera from the dropdown menu.

Main Camera

The ${\bf Main}\ {\bf Camera}\ {\bf component}\ {\bf has}\ {\bf three}\ {\bf distinct}\ {\bf sections:}$

- Game Time: Defines the time mode used to update the camera. By default it uses the Game Time option, which can pause time when the time scale is set to zero.
- Shot: Determines the Camera Shot associated with this camera controller. If none is set, the camera won't have any behavior.
- · Avoid Clipping: Allows the camera to avoid clipping through the geometry of the scene.

Smooth Camera Movement

The Shot's smoothing options determine how much the camera lags from the Shot's behavior. It's recommended to add some lag to avoid any jittering. However, introducing too much lag will make controls feel a a bit unresponsive.

Transition to a new Shot

To transition a Camera Controller from one Camera Shot to another one, it's recommended to use the **Change Shot** instruction.

Change Camera Shot instruction

Simply drop in the Camera Shot you want the Camera Controller mimic and how long should it take to transition. Game Creator will handle the rest.

1.4.3 Camera Shots

Camera Shots are components that provide the Camera Controller (or Main Camera) information about how they should move and behave.

Camera Shots Analogy

Think of Shots as a collecion of camera angles scattered around the scene, each trying to frame the action as best as possible. Then you, the Director, decide which camera is visualized on the screen, for how long and when to swap to another shot.

Creating a Camera Shot

To create a **Camera Shot** right click on the *Hierarchy* panel and select Game Creator Cameras Shot Camera from the dropdown menu. This will place a new game object on the scene with the **Camera Shot** attached to it.



If your scene doesn't have a Main Camera attached to the scene camera, creating a new **Camera Shot** will create one for you and link it to the newly created shot automatically for you.

Camera Shot

A Camera Shot component contains its shot type and a collection of parameters that can be modified to fine-tune its behavior. In the example above, the *Third Person* camera shot has 3 sections that allow to modify the target tracked, whether the user should be able to zoom in/out and how the orbit should be done. Clicking on each of these sections reveals or hides its content.



Since version 2.3.15 all Camera Shots have a toggle field called Is Main Shot.

Ticking this will allow to define it as the primary one, which can be used as a shortcut when selecting the **Main** Camera Shot field drop a camera selection dropdown.

Camera Shot Types

To change a camera shot type, simply click on its type name. A dropdown menu will appear from which the new type can be selected.

Camera Shot



Since version 2.7.28 all Camera Shots come with a Viewport section that allows to customize multiple properties of the camera when switching to the Shot, including the Field of View and the Projection mode.

Character Footstep Phases Animation Curves

FIXED POSITION

This camera shot doesn't move from its place. However, it can be instructed to keep track of a target's position by pivoting around itself. Think of this camera's behavior as a security camera.

FOLLOW TARGET

This camera is very similar to the *Fixed Position* but also allows to follow the target from a certain distance. Useful for top-down view games like Diablo.

FOLLOW TRACK

This camera shot allows to track a target as well as move along a pre-defined rail-like path. This path's position is defined by the position of the targeted object along another path. This camera shot is useful for games that have very linear corridors but want to smoothly turn the camera around corners.

ANIMATION

This camera shot moves along a pre-defined path over a certain amount of time. When it reaches the end of the animation, it stops there and does nothing else. This shot is very useful for cinematic sequences where multiple animation shots can be chained together to dynamically follow the action.

FIRST PERSON

This shot is perfect for first person games. The target object (usually a humanoid) determines the position of the shot and follows it while allowing to spin the head around.

Comes with a vast collection of features such as:

- **Head Bobbing:** The amount of up and down and side movement due to the character's change of weight when walking or running.
- Head Leaning: A subtle rotation on the local X and Z axis that is applied when the character moves in order to display the impulse required to go towards that particular direction.
- **Noise:** Another subtle yet realistic random movement applied to both the rotation and translation of the shot to simulate restless idle motion and breathing.

All these parameters can be changed at runtime to accommodate to different situations, such as increasing the noise after sprinting and such.

THIRD PERSON

This shot is used on third person games where the camera follows a target but the user is free to orbit around it.

LOCK ON

This shot allows to follow a target's position while the rotation follows another one, always framing both targets on screen. This shot is perfect for locking on enemies when making an action game or hinting the player something they should not be missing.

ANCHOR PEEK

This shot anchors itself to the chosen game object and allows to pan and tilt the camera vertically and horizontally, up to a certain amount. The *restitute* field brings back the shot to the center if no further input is detected. This is specially useful when using a gamepad controller and you want the character to peek around corners.

1.5 Visual Scripting

1.5.1 Visual Scripting

Game Creator comes with a unique high-level and intuitive visual scripting toolset that makes it very easy to code interactions. It only consists of 3 components:

- · Actions: A list of instructions that are executed one after another.
- · Triggers: A component that listens to events in the scene
- Conditions: Branch off to instructions, depending on certain conditions.

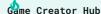
Usual Scripting nomenclature

The Actions component consists of a list of Instructions. The Conditions component is made of Branches, which contain a list of Conditions and Instructions. Lastly, the Trigger component listens for a specific Event in the scene.

Apart from these three visual scripting components, **Game Creator** also includes **Hotspots**, which is a special type of component that doesn't directly affect gameplay, but highlights interactive objects in different ways: For example, making a character's head turn towards a point when near, showing a text above an interactive element, and so on.

High Level Scripting

A high-level scripting language is a methodology in which programming interactions is closer to what humans are used to use. For example, in **Game Creator** you can tell a character to follow a target object; freeing the user from having to think what it means to *follow* an object.



Game Creator and each module comes packed with a unique set visual scripting tools. The Game Creator Hub is a web platform where community members upload free Instructions, Conditions and Events for everyone to download and use in their projects. Be sure to check it out!

Why not Playmaker

Why not both? Playmaker and Unity's Visual Scripting solution are graph-based, which tend to be closer to a programming language. If you're used to using these, you'll find these complement **Game Creator** very well.

On one hand, **Game Creator** makes it very fast and easy to structure common interactions without the need to code the low-level stuff. However, if you need more fine-grain control over some parts and you don't know how to code your own Instructions, you can use these graph-based solutions that perfectly complement the process of making games.

1.5.2 Actions

Actions

Actions are components that have a list of individual **Instructions** which are executed from top to bottom. It's important to note that an **Instruction** won't be executed until the previous one has finished.

Actions



Actions can be thought as task lists that must be completed from top to bottom.

CREATING ACTIONS

There are two ways to create an Actions object. One is to create an object that contains an Actions component, by right clicking on the *Hierarchy* panel and selecting *Game Creator Visual Scripting Actions*. This creates a scene object with the component attached to it.

However, an Actions component can also be added to any game object. Simply click on any game object's Add Component button and type Actions.

beleting Actions

To delete an Actions component, simply click on the component's little cog button and select "Remove Component" from the dropdown menu.

ADDING INSTRUCTIONS

To add an **Instruction** to an **Actions** component, click on the "Add Instruction" button to pop a dropdown list with a searchable field. Navigate through the different categories or search for a specific instruction and click it to add it at the bottom of the list.

Add a new Instruction

It is also possible to add **Instructions** at any point of the list. To do so, right click on any existing **Instruction** and choose "Insert Above" or "Insert Below" from the contextual menu that appears.

Accessible Fuzzy Search

Game Creator uses an advanced indexed search algorithm that allows to both syntactically and semmantically understand what the user is trying to search, even if the search contains mispelled words. For example, searching for "move" will display the "Move Character" instruction, but also the "Change Position" one.

BUILT-IN DOCUMENTATION

All **Instructions** have built-in documentation that explain what it does as well as a small description of each of its parameters. To access its documentation, either search for that particular instruction on the documentation, or right click it on the **Instruction** and select *Help*. A new floating window will appear with all the necessary information.

Instruction Documentation

DEBUGGING TOOLS

Actions come with built-in tools that allow to easily visualize and what's happening at runtime. Right click on any Instruction to pop a context menu with the *Disable* and add a *Breakpoint* options.

Disable Instruction

This option disables a particular instruction, as if it was not there.

Disable Instruction

The **Instruction** is greyed out and a special icon appears on its right side. Click the icon to enable the instruction again.

Add a Breakpoint

A breakpoint pauses the Unity Editor upon reaching a particular Instruction, right before executing it. This is very useful if you want to check the state of certain data before the execution progresses any further.

Breakpoint Instruction

When an **Instruction** has a breakpoint, it displays a red icon on its right side. Clicking it will remove the breakpoint from the Instruction.



It is important to note that *breakpoints* only work on the Editor and have no effect when building the project as a standalone application.

Instructions

INSTRUCTIONS

Sub Categories

- Animations
- Application
- Audio
- Cameras
- Characters
- Debug
- Game Objects
- Input
- Lights
- Math
- Physics 2D
- Physics 3D
- Renderer
- Scenes
- Storage
- Testing
- Time
- Transforms
- Ui
- Variables
- · Visual Scripting

ANIMATIONS

Animations

Instructions

- Change Animator Float
- Change Animator Integer
- Change Animator Layer
- Change Blend Shape
- Play Animation Clip
- Set Animation
- Set Animator Boolean
- Set Animator Trigger

Change Animator Float

Animations » Change Animator Float

Description

Changes the value of a 'Float' Animator parameter

Parameters

Name	Description			
Parameter Name	The Animator parameter name to be modified			
Value	The value of the parameter that is set			
Duration	How long it takes to perform the transition			
Easing	The change rate of the parameter over time			
Wait to Complete	Whether to wait until the transition is finished			
Animator	The Animator component attached to the game object			

Keywords

Parameter Number

Change Animator Integer

Animations » Change Animator Integer

Description

Changes the value of a 'Integer' Animator parameter

Parameters

Name	Description			
Parameter Name	The Animator parameter name to be modified			
Value	The value of the parameter that is set			
Duration	How long it takes to perform the transition			
Easing	The change rate of the parameter over time			
Wait to Complete	Whether to wait until the transition is finished			
Animator	The Animator component attached to the game object			

Keywords

Parameter Number

Change Animator Layer

Animations » Change Animator Layer

${\tt Description}$

Changes the weight of an Animator Layer

Parameters

Name	Description			
Layer Index	The Animator's Layer index that's being modified			
Weight	The target Animator layer weight			
Duration	How long it takes to perform the transition			
Easing	The change rate of the parameter over time			
Wait to Complete	Whether to wait until the transition is finished			
Animator	The Animator component attached to the game object			

Keywords

Weight

Change Blend Shape

Animations » Change Blend Shape

Description

Changes the value of a Blend Shape parameter

Parameters

Name	Description					
Skinned Mesh	The Skinned Mesh Renderer component attached to the game object					
Blend Shape	Name of the Blend Shape to change					
Value	The target value of the blend shape					
Duration	How long it takes to perform the transition					
Easing	The change rate of the parameter over time					
Wait to Complete	Whether to wait until the transition is finished					

Keywords

Morph Target

Play Animation Clip

Animations » Play Animation Clip

Description

Plays an Animation Clip on the chosen Animator

Parameters

Name	Description			
Animation Clip	The Animation Clip that is played			
Animator	The Animator component attached to the game object			

Keywords

Animate Reproduce Sequence Cinematic

Set Animation

Animations » Set Animation

Description

Sets the value of an Animation Clip

Parameters

Name	Description			
То	The location where to save the Animation Clip			
Animation Clip	The Animation Clip reference to store			

Keywords

Animation Clip Animator

Set Animator Boolean

Animations » Set Animator Boolean

Description

Sets the value of a 'Bool' Animator parameter

Parameters

Name	Description
Parameter Name	The Animator parameter name to be modified
Value	The value of the parameter that is set
Animator	The Animator component attached to the game object

Keywords

Parameter Bool

Set Animator Trigger

Animations » Set Animator Trigger

Description

Sets the value of a 'Trigger' Animator parameter

Parameters

Name	Description
Parameter Name	The Animator parameter name modified
Animator	The Animator component attached to the game object

Keywords

Parameter Once Flag Notify

APPLICATION

Application

Sub Categories

• Cursor

Instructions

- Open Web Page
- Quit Application

Open Web Page

Application » Open Web Page

Description

Opens the specified URL with the default web browser

Parameters

Na	me	Descrip	tion	1											
UR	L	The rou	te l	ink	to	open.	Must	include	the	protocol	prepended	(http o	r h	ttps)	

Keywords

Site Internet

Quit Application

Application » Quit Application

Description

Closes the application and exits the program. This instruction is ignored in the Unity Editor or WebGL platforms

Keywords

Exit Close Shutdown Turn

Cursor

Cursor Instructions

- Cursor Texture
- Cursor Visibility
- · Lock Cursor

Cursor Texture

Application » Cursor » Cursor Texture

Description

Changes the image of the hardware cursor

Parameters

Name	Description
Texture	The new appearance of the cursor. The texture must be set to Cursor type
Tip	The offset from the top left of the texture used as the target point
Mode	Determines if the cursor is rendered using software or hardware rendering

Keywords

Mouse Crosshair Click

Cursor Visibility

Application » Cursor » Cursor Visibility

Description

Determines if the hardware cursor is visible or not

Parameters

Name Description

Is Visible If true the cursor is visible, unless it is set as Locked

Keywords

Mouse FPS Crosshair

Lock Cursor

Application » Cursor » Lock Cursor

Description

Determines if the hardware pointer is locked to the center of the view or not

Parameters

Name Description

Lock Mode The behavior of the cursor. The default value is None

Keywords

Mouse State FPS Center Confine

AUDIO

Audio

Instructions

- Audio Mixer Parameter
- · Audio Source Pitch
- · Audio Source Volume
- Change Ambient Volume
- · Change Master Volume
- · Change Music Volume
- · Change Snapshot
- Change Sound Effects Volume
- · Change Speech Volume
- · Change Ui Volume
- Fade All Ambient
- Fade All Music
- Play Ambient
- Play Music
- Play Sound Effect
- · Play Speech
- Play Ui Sound
- Stop Ambient
- Stop Music
- Stop Sound Effect
- · Stop Speech On Game Object

Audio Mixer Parameter

Audio » Audio Mixer Parameter

Description

Changes the value of an Audio Mixer exposed parameter

Parameters

Name	Description
Audio Mixer	The Audio Mixer asset with the exposed parameter
Parameter Name	A string representing the name of the exposed parameter
Parameter Value	The value which the exposed parameter is set

Keywords

Float Exposed Effect Change

Audio Source Pitch

Audio » Audio Source Pitch

Description

Changes the pitch of an Audio Source component

Parameters

Name	Description
Audio Source	The Audio Source component
Pitch	The new targeted pitch to change
Transition	How long it takes to reach the new value

Keywords

Clip Music

Audio Source Volume

Audio » Audio Source Volume

Description

Changes the volume of an Audio Source component

Parameters

Name	Description
Audio Source	The Audio Source component
Volume	The new targeted volume to change
Transition	How long it takes to reach the new value

Keywords

Clip Music

Change Ambient volume

Audio » Change Ambient volume

Description

Change the Volume of Ambient music

Parameters

Name Description

Volume A value between 0 and 1 that indicates the volume percentage

Keywords

Audio Ambience Background Volume Level

Change Master volume

Audio » Change Master volume

Description

Change the Master volume. The Master volume controls how loud all other channels are

Parameters

Name	Description
Volume	A value between 0 and 1 that indicates the volume percentage

Keywords

Audio Sounds Volume Level

Change Music volume

Audio » Change Music volume

Description

Change the Volume of Music

Parameters

Name Description

Volume A value between 0 and 1 that indicates the volume percentage

Keywords

Audio Music Background Volume Level

Change Snapshot

Audio » Change Snapshot

Description

Smoothly transitions to a new snapshot over a period of time $% \left\{ 1,2,\ldots ,n\right\}$

Parameters

Name	Description
Snapshot	The Audio Mixer Snapshot that is activated
Transition	How long it takes to transition to the new Snapshot

Keywords

Effect Transition Effect Change

Change Sound Effects volume

Audio » Change Sound Effects volume

Description

Change the Volume of Sound Effects

Parameters

Name Description

Volume A value between 0 and 1 that indicates the volume percentage

Keywords

Audio Sounds Volume Level

Change Speech volume

Audio » Change Speech volume

Description

Change the Volume of character Speech

Parameters

Name	Description
Volume	A value between 0 and 1 that indicates the volume percentage

Keywords

Audio Character Voice Voices Volume Level

Change UI volume

Audio » Change UI volume

Description

Change the Volume of UI elements

Parameters

Name Description

Volume A value between 0 and 1 that indicates the volume percentage

Keywords

Audio User Interface Button Volume Level

Fade all Ambient

Audio » Fade all Ambient

Description

Stops all Ambient currently playing

Parameters

Name	Description
Wait To Complete	Check if you want to wait until the sound has faded out
Transition Out	Time it takes for the sound to fade out

Keywords

Audio Ambience Background Fade Mute

Fade all Music

Audio » Fade all Music

Description

Stops all Music currently playing

Parameters

Name	Description
Wait To Complete	Check if you want to wait until the sound has faded out
Transition Out	Time it takes for the sound to fade out

Keywords

Audio Music Background Fade Mute

Play Ambient

Audio » Play Ambient

Description

Plays a looped Audio Clip. Useful for background effects or persistent sounds.

Parameters

Name	Description
Audio Clip	The Audio Clip to be played
Transition In	Time it takes for the sound to fade in
Spatial Blending	Whether the sound is placed in a 3D space or not
Target	A Game Object reference that the sound follows as the source

Keywords

Audio Ambience Background

Play Music

Audio » Play Music

Description

Plays a looped Audio Clip. Useful for background music or persistent sounds.

Parameters

Name	Description
Audio Clip	The Audio Clip to be played
Transition In	Time it takes for the sound to fade in
Spatial Blending	Whether the sound is placed in a 3D space or not
Target	A Game Object reference that the sound follows as the source

Keywords

Audio Ambience Background

Play Sound Effect

Audio » Play Sound Effect

Description

Plays an Audio Clip sound effect just once

Parameters

Name	Description
Audio Clip	The Audio Clip to be played
Wait To Complete	Check if you want to wait until the sound finishes
Pitch	A random pitch value ranging between two values
Transition In	Time it takes for the sound to fade in
Spatial Blending	Whether the sound is placed in a 3D space or not
Target	A Game Object reference that the sound follows as its source

Keywords

Audio Sounds SFX FX

Play Speech

Audio » Play Speech

Description

Plays an Audio Clip speech over just once

Parameters

Name	Description
Audio Clip	The Audio Clip to be played
Wait To Complete	Check if you want to wait until the sound finishes
Spatial Blending	Whether the sound is placed in a 3D space or not
Target	A Game Object reference that the sound follows as its source

Keywords

Audio Voice Voices Sounds Character

Play UI sound

Audio » Play UI sound

Description

Plays a non-diegetic user interface Audio Clip

Parameters

Name	Description
Audio Clip	The Audio Clip to be played
Wait To Complete	Check if you want to wait until the sound finishes
Pitch	A random pitch value ranging between two values
Spatial Blending	Whether the sound is placed in a 3D space or not
Target	A Game Object reference that the sound follows as its source

Keywords

Audio Sounds User Interface Beep Button

Stop Ambient

Audio » Stop Ambient

${\tt Description}$

Stops a currently playing Ambient audio

Parameters

Name	Description
Audio Clip	The Audio Clip to be played
Wait To Complete	Check if you want to wait until the sound has faded out
Transition Out	Time it takes for the sound to fade out

Keywords

Audio Ambience Background Fade Mute

Stop Music

Audio » Stop Music

${\tt Description}$

Stops a currently playing Music audio

Parameters

Name	Description
Audio Clip	The Audio Clip to be played
Wait To Complete	Check if you want to wait until the sound has faded out
Transition Out	Time it takes for the sound to fade out

Keywords

Audio Music Background Fade Mute

Stop Sound Effect

Audio » Stop Sound Effect

Description

Stops a currently playing Sound Effect

Keywords

Audio Sounds Silence Fade Mute SFX FX

Stop Speech on Game Object

Audio » Stop Speech on Game Object

Description

Stops any Speech clips being played by a specific Game Object

Parameters

Name	Description
Target	A game object that is set as the source of the speech
 ords	

Keywords

Audio Voice Voices Sounds Character Silence Mute Fade

CAMERAS

Cameras

Sub Categories

- Properties
- Shakes
- Shots

Instructions

- · Change To Shot
- Revert To Previous Shot
- Set Main Shot

Change to Shot

Cameras » Change to Shot

Description

Changes the active Shot for a particular camera

Parameters

Name	Description
Camera	The target camera component
Shot	The camera Shot that becomes active
Duration	How long it takes to transition to the new Shot, in seconds
Wait To Complete	If the instruction waits till the transition is complete

Keywords

Cameras Render Switch Move

Revert to Previous Shot

Cameras » Revert to previous Shot

Description

Reverts the active Shot of a particular camera to the previous one

Parameters

Name	Description
Camera	The target camera component
Duration	How long it takes to transition to the new Shot, in seconds

Keywords

Cameras Render Switch Move

Set Main Shot

Cameras » Set Main Shot

Description

Assigns a Camera Shot as the new Main Shot

Parameters

Name	Description

Shot The new main Camera Shot

Properties

Properties Instructions

- · Change Culling Mask
- · Change Field Of View
- · Change Orthographic Size
- Change Projection

Change Culling Mask

Cameras » Properties » Change Culling Mask

Description

Changes the camera culling mask

Parameters

Name	Description
Camera	The camera component whose property changes
Culling Mask	The mask the camera uses to discern which objects to render

Keywords

Cameras Render

Change Field of View

Cameras » Properties » Change Field of View

Description

Changes the camera field of view

Parameters

Name	Description
Camera	The camera component whose property changes
FoV	The field of view of the camera, measured in degrees
Duration	The time in seconds, it takes for the camera to complete the change
Easing	The easing function used to transition

Keywords

Cameras Perspective FOV 3D

Change Orthographic Size

Cameras » Properties » Change Orthographic Size

Description

Changes the camera's orthographic size

Parameters

Name	Description
Camera	The camera component whose property changes
Size	The new size of the orthographic view
Duration	The time in seconds, it takes for the camera to complete the change
Easing	The easing function used to transition

Keywords

Cameras Orthographic Size 2D

Change Projection

Cameras » Properties » Change Projection

Description

Changes the camera projection to either Perspective or Orthographic

Parameters

Name	Description
Camera	The camera component whose property changes
Projection	Whether to change to Orthographic or Perspective mode

Keywords

Cameras Orthographic Perspective 3D 2D

Shakes

Shakes Instructions

- Shake Camera Burst
- Shake Camera Sustain
- Stop Camera Sustain Shake
- Stop Shake Camera Bursts

Shake Camera Burst

Cameras » Shakes » Shake Camera Burst

Description

Shakes the camera for an amount of time

Parameters

Name	Description
Camera	The camera that receives the burst shake effect
Delay	Amount of time in seconds before the shake effect starts
Duration	Amount of time the shake effect stays active
Shake Position	If the shake affects the position of the camera
Shake Rotation	If the shake affects the rotation of the camera
Magnitude	The maximum amount the camera displaces from its position
Roughness	Frequency or how violently the camera shakes
Transform	[Optional] Defines the origin of the shake
Radius	[Optional] Distance from the origin that the shake starts to fall-off

Keywords

Cameras Animation Animate Shake Impact Play

Shake Camera Sustain

Cameras » Shakes » Shake Camera Sustain

Description

Starts shaking the camera until the effect is manually turned off

Parameters

Name	Description
Camera	The camera that receives the sustain shake effect
Delay	Amount of time in seconds before the shake effect starts
Transition	Amount of seconds the shake effect takes to blend in
Shake Position	Whether the shake affects the position of the camera
Shake Rotation	Whether the shake affects the rotation of the camera
Magnitude	The maximum amount the camera displaces from its position
Roughness	Frequency or how violently the camera shakes
Transform	[Optional] Defines the origin of the shake
Radius	[Optional] Distance from the origin that the shake starts to fall-off

Keywords

Cameras Animation Animate Shake Wave Play

Stop Camera Sustain Shake

Cameras » Shakes » Stop Camera Sustain Shake

Description

Stops a Sustain Shake camera effect in a particular layer layer

Parameters

Name	Description
Camera	The camera target that stops a Sustain Shake effect
Layer	The camera layer from which the Sustain Shake effect is removed
Delay	Amount of time before the Sustain Shake effect starts blending out
Transition	Amount of time it takes to blend out the Sustain Shake effect

Keywords

Cameras Animation Animate Shake Wave Play

Stop Camera Shake Bursts

Cameras » Shakes » Stop Shake Camera Bursts

Description

Stops any ongoing camera Burst Shake effects

Parameters

Name	Description
Camera	The camera target that stops all its active Burst Shake effects
Delay	Amount of time before all Burst Shake effects start blending out
Transition	Amount of time it takes to blend out all Burst Shake effects

Keywords

Cameras Animation Animate Shake Impact Play

Shots

Shots Sub Categories

- Anchor
- Animation
- First Person
- Follow
- · Head Bobbing
- · Head Leaning
- · Lock On
- Look
- Orbit
- Zoom

Anchor

Anchor Instructions

- · Change Distance
- · Change Offset
- · Change Target

Change Distance

Cameras » Shots » Anchor » Change Distance

Description

Changes the anchored position the Shot sits relative to the target

Parameters

Name	Description	
Distance	The new distance relative to the target in local coordinates	
Shot	The camera Shot targeted	

Keywords

Cameras View Cameras Shot

Change Offset

Cameras » Shots » Anchor » Change Offset

Description

Changes the offset position of the targeted object

Parameters

Name	Description
Offset	The new offset in target local coordinates
Shot	The camera Shot targeted

Keywords

Change Target

Cameras » Shots » Anchor » Change Target

Description

Changes the targeted game object

Parameters

Name	Description
Target	The new target
Shot	The camera Shot targeted

Keywords

Animation Animation Instructions

• Change Duration

Change Duration

Cameras » Shots » Animation » Change Duration

Description

Changes the duration it takes for the Animation shot to complete

Parameters

Name Description	
Duration The new duration in seconds	
Shot The camera Shot targeted	

Keywords

First person First Person Instructions

- · Change Max Pitch
- Change Offset
- · Change Sensitivity
- Change Smooth Time
- Change Target

Change Max Pitch

Cameras » Shots » First Person » Change Max Pitch

Description

Changes the maximum rotation (up and down) allowed

Parameters

Name	Description
Max Pitch	The amount the Shot is allowed to look up and down, in degrees
Shot	The camera Shot targeted

Keywords

Change Offset

Cameras » Shots » First Person » Change Offset

Description

Changes the offset position of the targeted object

Parameters

Name	Description
Offset	The new offset in self local coordinates
Shot	The camera Shot targeted

Keywords

Change Sensitivity

Cameras » Shots » First Person » Change Sensitivity

Description

Changes how sensitive the Shot reacts to input

Parameters

Name	Description
Sensitivity	Input sensitivity for X and the Y axis
Shot	The camera Shot targeted

Keywords

Change Smooth Time

Cameras » Shots » First Person » Change Smooth Time

Description

Changes the maximum rotation (up and down) allowed

Parameters

Name	Description
Smooth Time	How smooth the camera operates when rotating
Shot	The camera Shot targeted

Keywords

Change Target

Cameras » Shots » First Person » Change Target

Description

Changes the targeted game object to view from

Parameters

Name	Description
Target	The new target
Shot	The camera Shot targeted

Keywords

Follow

Follow Instructions

- · Change Distance
- · Change Target

Change Distance

Cameras » Shots » Follow » Change Distance

Description

Changes the offset distance between the Shot and the targeted object

Parameters

Name	Description
Distance	The new offset distance in world coordinates
Shot	The camera Shot targeted

Keywords

Change Target

Cameras » Shots » Follow » Change Target

Description

Changes the targeted game object to Follow

Parameters

Name	Description
Follow	The new target to follow
Shot	The camera Shot targeted

Keywords

Head bobbing Head Bobbing Instructions

• Enable Head Bobbing

Enable Head Bobbing

Cameras » Shots » Head Bobbing » Enable Head Bobbing

Description

Toggles the active state of a Camera Shot's Head Bobbing system

Parameters

Name	Description
Active	The next state
Shot	The camera Shot targeted

Keywords

Cameras Disable Activate Deactivate Bool Toggle Off On Cameras Shot

Head leaning Head Leaning Instructions

• Enable Head Leaning

Enable Head Leaning

Cameras » Shots » Head Leaning » Enable Head Leaning

Description

Toggles the active state of a Camera Shot's Head Leaning system

Parameters

Name	Description
Active	The next state
Shot	The camera Shot targeted

Keywords

Cameras Disable Activate Deactivate Bool Toggle Off On Cameras Shot

Lock on Lock On Instructions

- · Change Anchor
- · Change Distance
- · Change Offset

Change Anchor

Cameras » Shots » Lock On » Change Anchor

Description

Changes the targeted game object to Lock On

Parameters

Name	Description
Anchor	The new target to Anchor onto
Shot	The camera Shot targeted

Keywords

Change Distance

Cameras » Shots » Lock On » Change Distance

Description

Changes the distance from the anchor point

Parameters

Name	Description
Distance	The new distance in self local coordinates
Shot	The camera Shot targeted

Keywords

Change Offset

Cameras » Shots » Lock On » Change Offset

Description

Changes the offset position of the targeted object

Parameters

Name	Description
Offset	The new offset in self local coordinates
Shot	The camera Shot targeted

Keywords

Look

Look Instructions

- Change Offset
- · Change Target
- Enable Look

Change Offset

Cameras » Shots » Look » Change Offset

Description

Changes the offset position of the targeted object

Parameters

Name	Description
Offset	The new offset in self local coordinates
Shot	The camera Shot targeted

Keywords

Change Target

Cameras » Shots » Look » Change Target

Description

Changes the targeted game object to look

Parameters

Name	Description
Target	The new target
Shot	The camera Shot targeted

Keywords

Enable Look

Cameras » Shots » Look » Enable Look

Description

Toggles the active state of a Camera Shot's Look system

Parameters

Name	Description
Active	The next state
Shot	The camera Shot targeted

Keywords

Cameras Disable Activate Deactivate Bool Toggle Off On Cameras Shot

Orbit

Orbit Instructions

- Change Alignment
- · Change Max Pitch
- Change Max Radius
- Change Offset
- Change Sensitivity
- Change Smooth Time
- Change Target

Change Alignment

Cameras » Shots » Orbit » Change Alignment

Description

Changes whether and how the Shot aligns behind the targeted object

Parameters

Name	Description	
Align with Target	If the Shot should move behind the target after some idle time	
Delay	If the Shot should move behind the target after some idle time	
Smooth Time	The speed at which	
Shot	The camera Shot targeted	

Keywords

Change Max Pitch

Cameras » Shots » Orbit » Change Max Pitch

Description

Changes the maximum rotation (up and down) allowed

Parameters

Name	Description
Max Pitch	The amount the Shot is allowed to look up and down, in degrees
Shot	The camera Shot targeted

Keywords

Change Max Radius

Cameras » Shots » Orbit » Change Max Radius

Description

Changes the maximum rotation (up and down) allowed

Parameters

Name	Description
Max Radius	The amount the Shot is allowed to look up and down, in degrees
Shot	The camera Shot targeted

Keywords

Change Offset

Cameras » Shots » Orbit » Change Offset

Description

Changes the offset position of the targeted object to orbit

Parameters

Name	Description
Offset	The new offset in self local coordinates
Shot	The camera Shot targeted

Keywords

Change Sensitivity

Cameras » Shots » Orbit » Change Sensitivity

Description

Changes how sensitive the Shot reacts to input

Parameters

Name	De	scription
Sensitivi	ity In	out sensitivity for X and the Y axis
Shot	The	e camera Shot targeted

Keywords

Change Smooth Time

Cameras » Shots » Orbit » Change Smooth Time

Description

Changes how smooth the orbit responds to input

Parameters

Name	Description
Smooth Time	How smooth is the orbital translation
Shot	The camera Shot targeted

Keywords

Change Target

Cameras » Shots » Orbit » Change Target

Description

Changes the targeted game object to orbit around

Parameters

Name	Description
Target	The new target
Shot	The camera Shot targeted

Keywords

Zoom

Zoom Instructions

- Change Level Zoom
- · Change Min Distance
- Change Smooth Time

Change Level Zoom

Cameras » Shots » Zoom » Change Level Zoom

Description

Changes the targeted zoom level percentage

Parameters

Name	Description
Level	The zoom level value between zero and one
Shot	The camera Shot targeted

Keywords

Change Min Distance

Cameras » Shots » Zoom » Change Min Distance

Description

Changes the targeted zoom level percentage

Parameters

Name	Description
Min Distance	The minimum zoom distance between the target and the Shot
Shot	The camera Shot targeted

Keywords

Cameras Shot

Change Smooth Time

Cameras » Shots » Zoom » Change Smooth Time

Description

Changes how smooth the zoom responds to input

Parameters

Name	Description
Smooth Time	How smooth is the zoom transition
Shot	The camera Shot targeted

Keywords

Cameras Shot

CHARACTERS

Characters

Sub Categories

- Animation
- Busy
- Combat
- Footsteps
- Ik
- Interaction
- Navigation
- Player
- Properties
- Ragdoll
- Visuals

Animation

Animation Instructions

- Change Smooth Time
- · Change State Weight
- Enter State
- · Play Gesture
- Stop Gesture
- Stop State

Change Smooth Time

Characters » Animation » Change Smooth Time

Description

Changes the average blend time between locomotion animations

Parameters

Name	Description
Smooth Time	The target Smooth Time value. Values usually range between 0 and 0.5 $$
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished
Character	The game object with the Character target

Example 1

The Smooth Time controls how fast a Character animation blends into another when reacting to external factors. A value of 0 makes the Character react instantly whereas a value of 0.5 takes half a second to completely blend in. A value between 0.2 and 0.4 usually provide the best results, though it depends on the look and feel the creator wants to achieve.

Keywords

Fade Realistic Old School Reaction

Change State Weight

Characters » Animation » Change State Weight

Description

Changes the weight of the State over time at the specified layer

Parameters

Name	e	Description
Cha	racter	The character that plays the animation state
Laye	er	Slot number in which the animation state is allocated
Weig	ght	The targeted opacity of the animation
Trai	nsition	The duration of the transition, in seconds

Keywords

Characters Animation Blend State Opacity

Enter State

Characters » Animation » Enter State

Description

Makes a Character start an animation State

Parameters

Name	Description
Character	The character that plays the animation state
State	The animation data necessary to play a state
Layer	Slot number in which the animation state is allocated
Blend Mode	Additively adds the new animation on top of the rest or overrides any lower layer animations
Delay	Amount of seconds to wait before the animation starts to play
Speed	Speed coefficient at which the animation plays
Weight	The opacity of the animation that plays. Between 0 and 1 $$
Transition	The amount of seconds the animation takes to blend in

Keywords

Characters Animation Animate State Play

Play Gesture

Characters » Animation » Play Gesture

Description

Plays an Animation Clip on a Character once

Parameters

Name	Description
Character	The character that plays the animation
Animation Clip	The Animation Clip that is played
Avatar Mask	(Optional) Allows to play the animation on specific body parts of the Character
Blend Mode	Additively adds the new animation on top of the rest or overrides any lower layer animations
Delay	Amount of seconds to wait before the animation starts to play
Speed	Speed coefficient at which the animation plays. 1 means normal speed
Transition In	The amount of seconds the animation takes to blend in
Transition Out	The amount of seconds the animation takes to blend out
Wait To Complete	If true this Instruction waits until the animation is complete

Keywords

Characters Animation Animate Gesture Play

Stop Gestures

Characters » Animation » Stop Gesture

Description

Stops any animation Gestures playing on the Character

Parameters

Name	Description
Character	The character that plays animation Gestures
Delay	Amount of seconds to wait before the animation starts to blend out
Transition	The amount of seconds the animation takes to blend out

Keywords

Characters Animation Animate Gesture Play

Stop State

Characters » Animation » Stop State

Description

Stops an animation State from a Character

Parameters

Name	Description
Character	The character that stops its animation State
Layer	Slot number from which the state is removed
Delay	Amount of seconds to wait before the animation stops playing
Transition	The amount of seconds the animation takes to blend out

Keywords

Characters Animation Animate State Exit Stop

Busy

Busy Instructions

- Set Available
- Set Busy

Set Available

Characters » Busy » Set Available

Description

Sets the Available state of a Character's Limbs

Parameters

Name	Description
Character	The Character game object
Limbs	The Limbs that are changed to Available

Keywords

Characters Busy Occupied Using

Set Busy

Characters » Busy » Set Busy

Description

Sets the Busy state of a Character's Limbs

Parameters

Name	Description
Character	The Character game object
Limbs	The Limbs that are changed to Busy

Keywords

Characters Busy Occupied Using

Combat

Combat Sub Categories

- Invincibility
- Poise
- Targeting

Invincibility
Invincibility Instructions

• Set Invincible

Set Invincible

Characters » Combat » Invincibility » Set Invincible

Description

Changes the Invincibility state of a Character

Parameters

Name	Description
Character	The Character that attempts to change its invincibility
Duration	The duration of the invincibility
Wait Until Complete	Whether to wait until the invincibility wears off

Keywords

Character Combat Invincibility

Poise Instructions

• Set Poise

Set Poise

Characters » Combat » Poise » Set Poise

Description

Changes the current Poise value of a Character

Parameters

Name	Description
Character	The Character that attempts to change its Poise value
Poise	The new Poise value

Keywords

Character Combat

Targeting

Targeting Instructions

- Add Target Candidate
- · Clear Target
- Cycle Closest Target
- Cycle Direction Target
- Cycle Next Target
- Cycle Previous Target
- Remove Target Candidate
- Set Target

Add Target Candidate

Characters » Combat » Targeting » Add Target Candidate

Description

Adds a new candidate target for the specified character

Parameters

Name	Description	
Character	The Character that attempts to change its candidate target	
Target	The new target candidate game object by the character	

Keywords

Character Combat Focus Pick

Clear Target

Characters » Combat » Targeting » Clear Target

Description

Clears the targeted game object by the specified Character

Parameters

Name Description

Character The Character that attempts to change its target

Keywords

Character Combat Focus Pick

Cycle Closest Target

Characters » Combat » Targeting » Cycle Closest Target

Description

Cycles to the closest candidate target to the character from the Targets list

Parameters

Name Description

Character The Character that attempts to change its candidate target

Keywords

Cycle Direction Target

Characters » Combat » Targeting » Cycle Direction Target

Description

Cycles to the visually closest target candidate from the Targets list and camera

Parameters

Name	Description	
Character	The Character that attempts to change its candidate target	
Camera	The point of view from which the direction is calculated	
Direction	The local space direction (only $[X,Y]$ components are used)	

Keywords

Cycle Next Target

Characters » Combat » Targeting » Cycle Next Target

Description

Cycles to the next candidate target from the Targets list

Parameters

Name Description

Character The Character that attempts to change its candidate target

Keywords

Cycle Previous Target

Characters » Combat » Targeting » Cycle Previous Target

Description

Cycles to the previous candidate target from the Targets list

Parameters

Name Description

Character The Character that attempts to change its candidate target

Keywords

Remove Target Candidate

Characters » Combat » Targeting » Remove Target Candidate

Description

Removes a new candidate target for the specified character

Parameters

Name	Description
Character	The Character that attempts to change its target candidate
Target	The target candidate to remove by the character

Keywords

Character Combat Focus Pick

Set Target

Characters » Combat » Targeting » Set Target

Description

Changes the targeted game object by the specified Character

Parameters

Name	Description
Character	The Character that attempts to change its target
Target	The new targeted game object by the character

Keywords

Character Combat Focus Pick

Footsteps

Footsteps Instructions

- Change Footstep Sounds
- Play Footstep

Change Footstep Sounds

Characters » Footsteps » Change Footstep Sounds

Description

Changes the sound table that links textures with footstep sounds

Parameters

Name	Description
Character	The character that plays animation Gestures
Footsteps	The sound table asset that contains information about how and when footstep sounds play

Keywords

Character Foot Step Stomp Foliage Audio Run Walk Move

Play Footstep

Characters » Footsteps » Play Footstep

Description

Plays a Footstep sound from a Material Sound asset

Parameters

Name	Description
Character	The character target
Material Sound	The material sound asset

Keywords

Step Foot Impact Land Sound

П

Ik Instructions

- Active Feet Ik
- Active Lean Ik
- Active Look Ik
- · Clear Looking Around
- Start Looking At
- Stop Looking At

Active Feet IK

Characters » IK » Active Feet IK

Description

Changes the active state of the Character Feet IK

Parameters

Na	me	Description
Ch	aracter	The character target
Ac	tive	Whether the IK system is active or not

Keywords

Active Lean IK

Characters » IK » Active Lean IK

Description

Changes the active state of the Character Lean IK

Parameters

Name	Description
Character	The character target
Active	Whether the IK system is active or not

Keywords

Active Look IK

Characters » IK » Active Look IK

Description

Changes the active state of the Character Look IK

Parameters

Name	Description
Character	The character target
Active	Whether the IK system is active or not

Keywords

Clear Looking Around

Characters » IK » Clear Looking Around

Description

Stops looking at any target that isn't in a Hotspot (priority zero)

Parameters

Name Description

Character The character target

Keywords

Start Looking At

Characters » IK » Start Looking At

Description

Starts looking at a target using the Look At IK system

Parameters

Name	Description
Character	The character target
Target	The targeted Transform to look at
Layer	The priority of this IK over other Look At attempts

Keywords

Stop Looking At

Characters » IK » Stop Looking At

Description

Stops looking at a target using the Look At IK system

Parameters

Name	Description
Character	The character target
Target	The targeted Transform to look at
Layer	The priority of this IK over other Look At attempts

Keywords

Inverse Kinematics IK

Interaction

Interaction Instructions

• Interact

Interact

Characters » Interaction » Interact

Description

Changes how the Player Character reacts to input commands

Parameters

Name Description

Character The Character that attempts to interact

Keywords

Character Button Pick Do Use Pull Press Push Talk

Navigation

Navigation Instructions

- Cancel Dash
- Dash
- Jump
- Move Direction
- Move To
- Set Character Driver
- Set Character Rotation
- Start Following
- Stop Following
- Stop Move
- Teleport

Cancel Dash

Characters » Navigation » Cancel Dash

Description

Cancels any existing Dash on the chosen Character

Parameters

Name Description

Character The game object with the Character target

Keywords

Leap Blink Roll Flash Character Player

Dash

Characters » Navigation » Dash

Description

Moves the Character in the chosen direction for a brief period of time

Parameters

Name	Description
Direction	Vector oriented towards the desired direction
Velocity	Velocity the Character moves throughout the whole movement
Duration	Defines the duration it takes to move forward at a constant velocity
Wait to Finish	If true this Instruction waits until the dash is completed
Mode	Whether to use Cardinal Animations (4 clips for each direction) or a single one
Animation Speed	Determines the speed coefficient applied to the animation played
Transition In	The time it takes to blend into the animation
Transition Out	The time it takes to blend out of the animation
Character	The game object with the Character target

Example 1

The Transition Out parameter is also used to determine the movement blend between the dash and the character's intended movement. Higher values will make characters take longer to regain control after dashing

Keywords

Leap Blink Roll Flash Character Player

Jump

Characters » Navigation » Jump

Description

Instructs the Character to jump

Parameters

Name Description

Character The game object with the Character target

Keywords

Hop Leap Reach Character Player

Move Direction

Characters » Navigation » Move Direction

Description

Attempts to move the Character towards the specified direction $% \left(1\right) =\left(1\right) \left(1$

Parameters

Name	Description
Direction	The the direction to move towards
Priority	Indicates the priority of this command against others
Character	The game object with the Character target

Keywords

Constant Walk Run To Vector Character Player

Move To

Characters » Navigation » Move To

Description

Instructs the Character to move to a new location

Parameters

Name	Description
Location	The position and rotation of the destination
Stop Distance	Distance to the destination that the Character considers it has reached the target
Cancel on Fail	Stops executing the rest of Instructions if the path has been obstructed
On Fail	A list of Instructions executed when it can't reach the destination
Character	The game object with the Character target

Example 1

The Stop Distance field is useful if you want [Character A] to approach another [Character B]. With a Stop Distance of 0, [Character A] tries to occupy the same space as the other one, bumping into it. Having a Stop Distance value of 2 allows [Character A] to stop 2 units away from [Character B]'s position

Keywords

Walk Run Position Location Destination Character Player

Set Character Driver

Characters » Navigation » Set Character Driver

Description

Changes the driver behavior of the Character

Parameters

Name	Description
Character	The Character that changes its Driver behavior
Driver	The Driver behavior that decides how the Character moves

Keywords

Character Drive Controller Navmesh Agent Rigidbody

Set Character Rotation

Characters » Navigation » Set Character Rotation

Description

Changes the rotation behavior of the Character

Parameters

Name	Description
Character	The Character that changes its Rotation behavior
Rotation	The Rotation behavior that decides where the Character faces

Keywords

Character Face Look Direction Pivot Lock

Start Following

Characters » Navigation » Start Following

Description

Instructs a Character to follow another game object

Parameters

Name	Description
Target	The target game object to follow
Min Distance	Distance from the Target the Character aims to move when approaching the Target
Max Distance	Maximum distance to the Target the Character leaves before attempting to move closer
Character	The game object with the Character target

Keywords

Lead Pursue Chase Walk Run Position Location Destination Character Player

Stop Following

Characters » Navigation » Stop Following

Description

Instructs a Character to stop following a game object

Parameters

Name Description

Character The game object with the Character target

Keywords

Cancel Lead Pursue Chase Character Player

Stop Move

Characters » Navigation » Stop Move

Description

Attempts to stop the character from moving

Parameters

Name	Description
Priority	Indicates the priority of this command against others
Character	The game object with the Character target

Keywords

Constant Walk Run To Vector Character Player

Teleport

Characters » Navigation » Teleport

Description

Instantaneously moves a Character from its current position to a new one

Parameters

Name	Description
Location	The position and/or rotation where the Character is teleported
Character	The game object with the Character target

Keywords

Change Position Location Respawn Spawn Character Player

Player

Player Instructions

- · Change Player
- Set Player Input

Change Player

Characters » Player » Change Player

Description

Changes the Character identified as the Player

Parameters

Name Description

Character The Character becomes the new Player character

Keywords

Character Is Control

Set Player Input

Characters » Player » Set Player Input

Description

Changes how the Player Character reacts to input commands

Parameters

Name	Description
Character	The Character that changes its Player Input behavior
Input	The new input method that the Character starts to listen

Keywords

Character Button Control Keyboard Mouse Gamepad Joystick

Properties

Properties Instructions

- Axonometry
- Can Collide
- Can Jump
- · Change Angular Speed
- Change Gravity
- · Change Height
- Change Id
- · Change Jump Force
- · Change Mass
- · Change Movement Speed
- · Change Radius
- · Change Terminal Velocity
- · Change Time Mode
- Is Controllable
- Kill Character
- Mannequin Position
- Mannequin Rotation
- · Mannequin Scale
- · Reset Vertical Velocity
- · Revive Character

Change Axonometry

Characters » Properties » Axonometry

Description

Changes the Character's Axonometry value

Parameters

Name	Description
Axonometry	The new Axonometry value
Character	The game object with the Character target

Keywords

Isometric Side Scroll

Can Collide

Characters » Properties » Can Collide

Description

Changes whether the Character can collide with other objects or not

Parameters

Name	Description
Character	The character target
Can Collide	Whether the character collides with other physic objects

Can Jump

Characters » Properties » Can Jump

Description

Changes whether the Character is allowed to jump or not

Parameters

Name	Description
Character	The character target
Can Jump	Whether the character is allowed to jump or not

Keywords

Hop Elevate

Change Angular Speed

Characters » Properties » Change Angular Speed

Description

Changes the Character's angular speed over time

Parameters

Name	Description
Angular Speed	The target Angular Speed value for the Character, measured in degrees per second
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished
Character	The game object with the Character target

Keywords

Rotation Euler Direction Face Look

Change Gravity

Characters » Properties » Change Gravity

Description

Changes the Character's gravity over time

Parameters

Mode Whether the upwards, downwards or both Gravity values are changed Gravity The target Gravity value for the Character Duration How long it takes to perform the transition Easing The change rate of the parameter over time Wait to Complete Whether to wait until the transition is finished Character The game object with the Character target	Name	Description
Duration How long it takes to perform the transition Easing The change rate of the parameter over time Wait to Complete Whether to wait until the transition is finished	Mode	Whether the upwards, downwards or both Gravity values are changed
Easing The change rate of the parameter over time Wait to Complete Whether to wait until the transition is finished	Gravity	The target Gravity value for the Character
Wait to Complete Whether to wait until the transition is finished	Duration	How long it takes to perform the transition
	Easing	The change rate of the parameter over time
Character The game object with the Character target	Wait to Complete	Whether to wait until the transition is finished
	Character	The game object with the Character target

Keywords

Space

Change Height

Characters » Properties » Change Height

Description

Changes the Character's height over time

Parameters

Name	Description
Height	The target Height value for the Character
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished
Character	The game object with the Character target

Keywords

Length

Change ID

Characters » Properties » Change ID

Description

Changes the Character's ID

Parameters

Name	Description
ID	The new ID of the Character
Character	The game object with the Character target

Keywords

Unique Guid

Change Jump Force

Characters » Properties » Change Jump Force

Description

Changes the Character's jump force over time

Parameters

Name	Description
Jump Force	The target Jump Force value for the Character
Duration	How long it will take to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished
Character	The game object with the Character target

Keywords

Hop Build Wind Fly

Change Mass

Characters » Properties » Change Mass

Description

Changes the Character's mass over time

Parameters

Name	Description
Mass	The target Mass value for the Character
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished
Character	The game object with the Character target

Keywords

Weight

Change Movement Speed

Characters » Properties » Change Movement Speed

Description

Changes the Character's maximum speed over time

Parameters

Name	Description
Speed	The target movement Speed value for the Character
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished
Character	The game object with the Character target

Keywords

Linear Walk Run Jog Sprint Velocity Throttle

Change Radius

Characters » Properties » Change Radius

Description

Changes the Character's radius over time

Parameters

Name	Description
Radius	The target Radius value for the Character
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished
Character	The game object with the Character target

Keywords

Diameter Space Fat Thin

Change Terminal Velocity

Characters » Properties » Change Terminal Velocity

Description

Changes the Character's maximum fall-speed over time. Useful for gliding

Parameters

Name	Description
Terminal Velocity	The target Terminal Velocity value for the Character
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished
Character	The game object with the Character target

Keywords

Fall Glide Parachute Height

Change Time Mode

Characters » Properties » Change Time Mode

Description

Changes the Character's Time Mode

Parameters

Name	Description
Time Mode	The target Time Mode for the Character
Character	The game object with the Character target

Keywords

Scale Game

Is Controllable

Characters » Properties » Is Controllable

Description

Changes whether the Character (Player) responds using input commands

Parameters

Name	Description
Character	The character target
Is Controllable	Whether the character responds to input commands

Kill Character

Characters » Properties » Kill Character

Description

Changes the state of the Character to dead

Parameters

Name Description

Character The character target

Keywords

Dead Die Murder

Mannequin Position

Characters » Properties » Mannequin Position

Description

Changes the local position of the Mannequin object within the Character

Parameters

Nam	e	Description
Cha	racter	The character target
Pos	ition	The Local Position of the Mannequin

Keywords

Location Model Local Change Set Root

Mannequin Rotation

Characters » Properties » Mannequin Rotation

Description

Changes the local rotation of the Mannequin object within the Character

Parameters

Name	Description
Character	The character target
Rotation	The Local Rotation of the Mannequin

Keywords

Location Model Local

Mannequin Scale

Characters » Properties » Mannequin Scale

Description

Changes the local scale of the Mannequin object within the Character

Parameters

Name	Description
Character	The character target
Scale	The Local Scale of the Mannequin

Keywords

Location Model Local

Reset Vertical Velocity

Characters » Properties » Reset Vertical Velocity

Description

Changes the Character's vertical velocity to zero

Parameters

Name Description

Character The game object with the Character target

Keywords

Fall Speed

Revive Character

Characters » Properties » Revive Character

Description

Changes the state of the Character to alive

Parameters

Name Description

Character The character target

Keywords

Respawn Alive Resurrect

Ragdoll

Ragdoll Instructions

- Recover Ragdoll
- Start Ragdoll

Recover from Ragdoll

Characters » Ragdoll » Recover Ragdoll

Description

Recovers a Character from the Ragdoll state and stands up

Parameters

Name Description

Keywords

Characters Ragdoll Recover Stand

Start Ragdoll

Characters » Ragdoll » Start Ragdoll

Description

Makes a Character enter a ragdoll state

Parameters

Name Description

Keywords

Characters Ragdoll Dead Kill Die

Visuals

Visuals Instructions

- Attach Prop
- · Change Model
- Drop Prop
- Put On Skin Mesh
- Remove Prop
- · Take Off Skin Mesh

Attach Prop

Characters » Visuals » Attach Prop

Description

Attaches a prefab or instance Prop onto a Character's bone

Parameters

Name	Description
Character	The character target
Туре	Whether to attach the prop as a prefab or instance
Prop	The prefab or instance object that is attached to the character
Bone	Which bone the prop is attached to
Position	Local offset from which the prop is distanced from the bone
Rotation	Local offset from which the prop is rotated from the bone

Keywords

Characters Add Grab Draw Pull Take Object

Change Model

Characters » Visuals » Change Model

Description

Changes the Character current model

Parameters

Name	Description
Character	The character target
Model	The prefab object that replaces the current Character model
Skeleton	Optional parameter that replaces the configuration of volumes
Footstep Sounds	Optional parameter that replaces the current Footstep sounds
Offset	A local offset from the center of the Character

Keywords

Characters Model

Drop Prop

Characters » Visuals » Drop Prop

Description

Drops a prefab or instance Prop (if any) from a Character

Parameters

Name	Description
Character	The character target
Туре	Whether to drop the prop form a prefab or as its instance
Prop	The prefab or instance object prop that is dropped from the character

Keywords

Characters Detach Let Sheathe Put Holster Object

Put on Skin Mesh

Characters » Visuals » Put on Skin Mesh

Description

Creates a new instance of a skin mesh renderer and puts it on a Character

Parameters

Name	Description	
Prefab	Game Object reference with a Skin Mesh Renderer that is instantiated	
On Character	Target Character that uses its armature to wear the skin mesh	

Keywords

Renderer New Game Object Armature

Remove Prop

Characters » Visuals » Remove Prop

Description

Removes a prefab or instance Prop (if any) from a Character

Parameters

Name	Description
Character	The character target
Туре	Whether to remove the prop form a prefab or as its instance
Prop	The prefab or instance object prop that is removed from the character

Keywords

Characters Detach Let Sheathe Put Holster Object

Take off Skin Mesh

Characters » Visuals » Take off Skin Mesh

Description

Removes an instance of a Skin Mesh from a Character

Parameters

Name	Description
Prefab	Game Object reference with a Skin Mesh Renderer that is removed
From Character	Target Character that uses its armature to wear the skin mesh

Keywords

Renderer Game Object Armature

DEBUG

Debug

Sub Categories

- Console
- Gizmos

Instructions

- Beep
- Comment
- Frame Step
- Log Number
- Log Text
- Pause Editor

Веер

Debug » Beep

Description

Plays the Operative System default 'beep' sound. This is intended for debugging purposes and doesn't do anything on a runtime application

Keywords

Debug

Comment

Debug » Comment

Description

Displays an explanation or annotation in the instructions list. It is intended to make instructions easier for humans to understand

Parameters

Name	Description
Text	The text of the comment

Keywords

Debug Note Annotation Explanation

Frame Step

Debug » Frame Step

${\tt Description}$

Performs a single frame step. It requires the Editor to be paused $% \left(1\right) =\left(1\right) \left(1\right) \left$

Keywords

Debug

Log Number

Debug » Log Number

Description

Prints a text from a numeric source to the Unity Console

Parameters

Name	Description
Number	The number to log

Keywords

Debug Log Print Show Display Test Float Double Decimal Integer Message

Log Text

Debug » Log Text

Description

Prints a message to the Unity Console

Parameters

Name Description

Message The text message to \log

Keywords

Debug Log Print Show Display Name Test Message String

Pause Editor

Debug » Pause Editor

Description

Pauses the Editor. This has no effect on standalone applications

Keywords

Debug Break Pause Stop

Console

Console Instructions

- · Console Close
- · Console Command
- Console Open
- · Console Text
- Console Toggle

Console Close

Debug » Console » Console Close

Description

Closes the Runtime Console

Keywords

Terminal Log Debug

Console Command

Debug » Console » Console Command

Description

Submits a Command onto the Runtime Console

Parameters

Name Description

Command The command message to submit

Keywords

Debug Log Terminal Submit Send Execute Run

Console Open

Debug » Console » Console Open

Description

Opens the Runtime Console

Keywords

Terminal Log Debug

Console Text

Debug » Console » Console Text

Description

Prints a message to the Runtime Console

Parameters

Name Description

Message The text message to log

Keywords

Debug Log Print Show Display Name Test Message String Terminal

Console Toggle

Debug » Console » Console Toggle

Description

Toggles the Runtime Console

Keywords

Terminal Log Debug

Gizmos

Gizmos Instructions

• Gizmo Line

Gizmo Line

Debug » Gizmos » Gizmo Line

Description

Displays a line between two points for a certain time

Keywords

Debug Gizmo Draw

GAME OBJECTS

Game Objects

Sub Categories

- Components
- Pooling

Instructions

- Change Layer
- · Change Name
- · Change Tag
- Destroy
- Instantiate
- Set Active
- Set Game Object
- Toggle Active

Change Layer

Game Objects » Change Layer

${\tt Description}$

Changes the layer value of a game object

Parameters

Name	Description
Layer	The layer where the game object belongs to
Children Too	Whether to also change the layer of the game object's children or not
Game Object	Target game object

Keywords

MonoBehaviour Behaviour Script

Change Name

Game Objects » Change Name

${\tt Description}$

Changes the name of a game object

Parameters

Name	Description
Name	The new name assigned to the game object
Game Object	Target game object

Keywords

MonoBehaviour Behaviour Script

Change Tag

Game Objects » Change Tag

${\tt Description}$

Changes the Tag of a game object

Parameters

Name	Description
Tag	The tag value which the game object belongs to
Game Object	Target game object

Keywords

MonoBehaviour Behaviour Script

Destroy

Game Objects » Destroy

Description

Destroys a game object scene instance

Parameters

Name Description

Game Object Target game object

Keywords

Remove Delete Flush MonoBehaviour Behaviour Script

Instantiate

Game Objects » Instantiate

${\tt Description}$

Creates a new instance of a referenced game object

Parameters

Name	Description
Game Object	Game Object reference that is instantiated
Position	The position of the new game object instance
Rotation	The rotation of the new game object instance
Save	Optional value where the newly instantiated game object is stored

Keywords

Create New Game Object

Set Active

Game Objects » Set Active

Description

Changes the state of a game object to active or inactive

Parameters

Name Description

Game Object Target game object

Keywords

Activate Deactivate Enable Disable MonoBehaviour Behaviour Script

Set Game Object

Game Objects » Set Game Object

Description

Sets a game object value equal to another one

Parameters

Name	Description
Set	Where the value is set
From	The value that is set

Keywords

Change Instance Variable Asset

Toggle Active

Game Objects » Toggle Active

Game Object

Description

Toggles the state of a game object to active or to inactive

Target game object

Parameters

Name Description

Keywords

Activate Deactivate Enable Disable Switch Swap MonoBehaviour Behaviour Script

Components

Components Instructions

- Add Component
- · Disable Collider
- Disable Component
- · Disable Renderer
- Enable Collider
- Enable Component
- Enable Renderer
- · Remove Component

Add Component

Game Objects » Components » Add Component

Description

Adds a component class to the game object

Parameters

Name Description

Game Object Target game object

Keywords

Add Append MonoBehaviour Behaviour Script

Disable Collider

Game Objects » Components » Disable Collider

Description

Disables a Collider component from the game object

Parameters

Name Description

Game Object Target game object

Keywords

Inactive Turn Off Box Sphere Capsule Mesh

Disable Component

Game Objects » Components » Disable Component

Description

Disables a component class from the game object

Parameters

Name Description

Game Object Target game object

Keywords

Deactivate Turn Off MonoBehaviour Behaviour Script

Disable Renderer

Game Objects » Components » Disable Renderer

Description

Disables a Renderer component from the game object

Parameters

Name Description

Game Object Target game object

Keywords

Inactive Turn Off Mesh

Enable Collider

Game Objects » Components » Enable Collider

Description

Enables a Collider component from the game object

Parameters

Name Description

Game Object Target game object

Keywords

Active Turn On Box Sphere Capsule Mesh

Enable Component

Game Objects » Components » Enable Component

Description

Enables a component class from the game object

Parameters

Name Description

Game Object Target game object

Keywords

Active Turn On MonoBehaviour Behaviour Script

Enable Renderer

Game Objects » Components » Enable Renderer

Description

Enables a Renderer component from the game object

Parameters

Name Description

Game Object Target game object

Keywords

Inactive Turn Off Mesh

Remove Component

Game Objects » Components » Remove Component

Description

Removes an existing component from the game object $% \left(1\right) =\left(1\right) \left(1\right) \left($

Parameters

Name Description

Game Object Target game object

Keywords

Delete Destroy MonoBehaviour Behaviour Script

Pooling

Pooling Instructions

- Pool Destroy
- Pool Prewarm

Pool Destroy

Game Objects » Pooling » Pool Destroy

Description

Destroys an existing game object pool

Parameters

Name Description

Game Object The Game Object reference is used as the template for the pool

Example 1

Use this Instruction to dispose those pools that have been pre-warmed. Pools created at runtime are automatically disposed when their scene is unloaded.

Keywords

Dispose Destroy Delete Game Object

Pool Prewarm

Game Objects » Pooling » Pool Prewarm

Description

Creates or makes sure an existing game object pool has enough instances

Parameters

Name	Description
Game Object	The Game Object reference is used as the template for the pool
Pool Size	The size of the pool of game objects

Example 1

Pre-warming a Pool moves it to the DontDestroyOnLoad scene. This means its contents will never be destroyed even after loading new scenes. To delete a pre-warmed pool use the Pool Destroy instruction.

Keywords

Create New Initialize Game Object

INPUT

Input

Instructions

- Disable Input Action
- · Disable Input Map
- · Display Touchstick Left
- · Display Touchstick Right
- Enable Input Action
- Enable Input Map

Disable Input Action

Input » Disable Input Action

Description

Disables an Input Action asset which stops it from reading user input

Parameters

Name Description

Input Asset The Input Asset reference

Keywords

Deactivate Inactive

Disable Input Map

Input » Disable Input Map

Description

Disables an Input Action asset with a Map value which stops reading user input

Parameters

Name Description

Input Asset The Input Asset reference

Keywords

Deactivate Inactive

Display Touchstick Left

Input » Display Touchstick Left

Description

Shows or hides the default Touchstick on the left side

Parameters

Name	Description	
Show	Shows the touchstick if ticked. Hides the touchstick otherwise	

Keywords

Joystick

Display Touchstick Right

Input » Display Touchstick Right

Description

Shows or hides the default Touchstick on the right side

Parameters

Name	Description
Show	Shows the touchstick if ticked. Hides the touchstick otherwise

Keywords

Joystick

Enable Input Action

Input » Enable Input Action

Description

Enables an Input Action asset which allows it to start reading user input

Parameters

Name Description

Input Asset The Input Asset reference

Keywords

Activate Active Start

Enable Input Map

Input » Enable Input Map

Description

Enables an Input Action asset with a Map value which allows reading user input

Parameters

Name Description

Input Asset The Input Asset reference

Keywords

Activate Active Start

LIGHTS

Lights

Instructions

- Light Color
- Light Intensity

Light Color

Lights » Light Color

Description

Smoothly changes the color of a Light component

Parameters

Name	Description
Color	The color the Light component starts emitting
Light	The game object with a Light component
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished

Keywords

Colour Hue Mood RGB Light Light Spot Sun Point Strength Burn Dark

Light Intensity

Lights » Light Intensity

Description

Smoothly changes the intensity of a Light component

Parameters

Name	Description
Intensity	The intensity change that the Light component undergoes
Light	The game object with a Light component
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished

Keywords

Light Spot Sun Point Strength Burn Dark

MATH

Math

Sub Categories

- Arithmetic
- Boolean
- Geometry
- Shading
- Text

Arithmetic

Arithmetic Instructions

- · Absolute Number
- Add Numbers
- · Clamp Number
- Cosine
- Divide Numbers
- · Increment Number
- Modulus Numbers
- Multiply Numbers
- Set Number
- · Sign Of Number
- Sine
- Subtract Numbers
- Tangent

Absolute Number

Math » Arithmetic » Absolute Number

Description

Sets a value without its sign

Parameters

Name	Description
Set	Where the value is stored
Number	The input value

Keywords

Change Float Integer Variable Sign Positive Modulus Magnitude

Add Numbers

Math » Arithmetic » Add Numbers

Description

Add two values together

Parameters

Name	Description	
Set	Where the resulting value is set	
Value 1	The first operand of the arithmetic operation	
Value 2	The second operand of the arithmetic operation	

Keywords

Sum Plus Float Integer Variable

Clamp Number

Math » Arithmetic » Clamp Number

Description

Clamps a value between a range defined by two others (inclusive)

Parameters

Name	Description
Set	Where the resulting value is set
Value	The value that is clamped between two others
Minimum	The smallest possible value
Maximum	The largest possible value

Keywords

Min Max Negative Minus Float Integer Variable

Cosine

Math » Arithmetic » Cosine

Description

Sets a value equal the Cosine of a number

Parameters

Name	Description
Set	Where the value is stored
Cosine	The angle input in radians

Keywords

Change Float Integer Variable

Divide Numbers

Math » Arithmetic » Divide Numbers

Description

Performs a division between the first and the second values

Parameters

Name	Description
Set	Where the resulting value is set
Value 1	The first operand of the arithmetic operation
Value 2	The second operand of the arithmetic operation

Keywords

Fraction Float Integer Variable

Increment Number

Math » Arithmetic » Increment Number

Description

Sets a value equal the sum of itself, plus another number

Parameters

Name	Description
Set	The value being incremented
Value	The value to add

Keywords

Change Float Integer Variable

Modulus Numbers

Math » Arithmetic » Modulus Numbers

Description

Calculates the modulus between the first and the second value

Parameters

Name	Description
Set	Where the resulting value is set
Value 1	The first operand of the arithmetic operation
Value 2	The second operand of the arithmetic operation

Keywords

Fraction Float Integer Variable Module

Multiply Numbers

Math » Arithmetic » Multiply Numbers

Description

Multiplies two values together

Parameters

Name	Description	
Set	Where the resulting value is set	
Value 1	The first operand of the arithmetic operation	
Value 2	The second operand of the arithmetic operation	

Keywords

Product Float Integer Variable

Set Number

Math » Arithmetic » Set Number

Description

Sets a value equal to another value

Parameters

Name	Description
Set	Where the value is set
From	The value that is set

Keywords

Change Float Integer Variable

Sign of Number

Math » Arithmetic » Sign of Number

Description

Sets a value equal to $\mbox{-1}$ if the input number is negative. 1 otherwise

Parameters

Name	Description
Set	Where the value is stored
Number	The input value

Keywords

Change Float Integer Variable Positive Negative

Sine

Math » Arithmetic » Sine

Description

Sets a value equal the Sine of a number

Parameters

Name	Description
Set	Where the value is stored
Sine	The angle input in radians

Keywords

Change Float Integer Variable

Subtract Numbers

Math » Arithmetic » Subtract Numbers

Description

Subtracts the second value from the first one

Parameters

Name	Description	
Set	Where the resulting value is set	
Value 1	The first operand of the arithmetic operation	
Value 2	The second operand of the arithmetic operation	

Keywords

Rest Negative Minus Float Integer Variable

Tangent

Math » Arithmetic » Tangent

Description

Sets a value equal the Tangent of a number

Parameters

Name	Description
Set	Where the value is stored
Tangent	The angle input in radians

Keywords

Change Float Integer Variable

Boolean

Boolean Instructions

- · And Bool
- · Nand Bool
- Nor Bool
- Or Bool
- Set Bool
- · Toggle Bool

AND Bool

Math » Boolean » AND Bool

Description

Executes an AND operation between to values and saves the result

Parameters

Name	Description
Set	Where the resulting value is set
Value 1	The first operand of the boolean operation
Value 2	The second operand of the boolean operation

Keywords

Subtract Minus Variable Boolean

NAND Bool

Math » Boolean » NAND Bool

Description

Executes a NAND operation between to values and saves the result

Parameters

Name	Description	
Set	Where the resulting value is set	
Value 1	The first operand of the boolean operation	
Value 2	The second operand of the boolean operation	

Keywords

Not Negative Subtract Minus Variable Boolean

NOR Bool

Math » Boolean » NOR Bool

Description

Executes a NOR operation between to values and saves the result

Parameters

Name	Description
Set	Where the resulting value is set
Value 1	The first operand of the boolean operation
Value 2	The second operand of the boolean operation

Keywords

Not Negative Sum Plus Variable Boolean

OR Bool

Math » Boolean » OR Bool

Description

Executes an OR operation between to values and saves the result

Parameters

Name	Description
Set	Where the resulting value is set
Value 1	The first operand of the boolean operation
Value 2	The second operand of the boolean operation

Keywords

Sum Plus Variable Boolean

Set Bool

Math » Boolean » Set Bool

Description

Sets a boolean value equal to another value

Parameters

Name	Description
Set	Where the value is set
From	The value that is set

Keywords

Change Boolean Variable

Toggle Bool

Math » Boolean » Toggle Bool

Description

Toggles the value of a Boolean value

Parameters

Name	Description
Set	The boolean value that stores the result
From	The boolean value that is toggled

Keywords

Change Boolean Variable Not Flip Switch

Geometry

Geometry Instructions

- Add Directions
- Add Points
- Clamp
- · Cross Product
- Distance
- Dot Product
- Normalize
- Project On Plane
- Reflect On Plane
- · Remap Coordinates
- Scale Product
- Set Direction
- Set Point
- Set Vector X
- Set Vector Y
- Set Vector Z
- Subtract Directions
- Subtract Points
- Transform To Local Direction
- Transform To Local Point
- Transform To World Direction
- · Transform To World Point
- Uniform Scale

Add Directions

Math » Geometry » Add Directions

Description

Adds two values that represent a direction in space and saves the result

Parameters

Name	Description
Set	Where the resulting value is set
Direction 1	The first operand of the geometric operation that represents a direction
Direction 2	The second operand of the geometric operation that represents a direction

Keywords

Sum Plus Position Location Variable

Add Points

Math » Geometry » Add Points

Description

Adds two values that represent a point in space and saves the result

Parameters

Name	Description
Set	Where the resulting value is set
Point 1	The first operand of the geometric operation that represents a point in space
Point 2	The second operand of the geometric operation that represents a point in space

Keywords

Sum Plus Position Location Variable

Clamp

Math » Geometry » Clamp

Description

Clamps all components of a Vector3 between two values

Parameters

Name	Description
Set	Dynamic variable where the resulting value is set
Value	The Vector3 value clamped between Minimum and Maximum
Minimum	The minimum value
Maximum	The maximum value

Keywords

Limit Vector3 Vector2 Constraint Variable

Cross Product

Math » Geometry » Cross Product

Description

Calculates the cross product of two direction values and saves the result

Parameters

Na	me	Description
Se	t	Where the resulting value is set
Di	rection 1	The first operand of the geometric operation that represents a direction
Di	rection 2	The second operand of the geometric operation that represents a direction

Keywords

Multiply Orthogonal Perpendicular Normal Position Location Variable

Distance

Math » Geometry » Distance

Description

Calculates the distance between two points in space and saves the result

Parameters

Name Description		Description
	Set	Where the resulting value is set
	Point 1	The first operand of the geometric operation that represents a point in space
	Point 2	The second operand of the geometric operation that represents a point in space

Keywords

Magnitude Position Location Variable

Dot Product

Math » Geometry » Dot Product

Description

Calculates the dot product between two directions and saves the result

Parameters

Name	Description	
Set	Where the resulting value is set	
Direction 1	The first operand of the geometric operation that represents a direction	
Direction 2	The second operand of the geometric operation that represents a direction	

Keywords

Direction Parallel Perpendicular

Normalize

Math » Geometry » Normalize

Description

Makes the magnitude of a direction vector equal to ${\bf 1}$

Parameters

Name	Description
Set	Dynamic variable where the resulting value is set
From	The direction vector that is normalized

Keywords

Change Vector3 Vector2 Unit Magnitude Variable

Project on Plane

Math » Geometry » Project on Plane

Description

Projects a direction on a plane defined by a normal vector and saves the result

Parameters

Name	Description
Set	Where the resulting value is set
Direction	The direction vector that is projected on a plane
Plane Normal	The plane represented by the direction of its normal vector

Keywords

Direction Surface Sway

Reflect on Plane

Math » Geometry » Reflect on Plane

Description

Reflects a direction on a plane defined by a normal vector and saves the result

Parameters

Name	Description	
Set	Where the resulting value is set	
Direction	The direction vector that is reflected on a plane	
Plane Normal	The plane represented by the direction of its normal vector	

Keywords

Direction Bounce Ricochet Snell

Remap Coordinates

Math » Geometry » Remap Coordinates

Description

Changes each of the components of a Vector3 value

Parameters

Name	Description
Value	The Vector3 value affected by the operation
Χ	Where the X coordinate component is remapped
Υ	Where the Y coordinate component is remapped
Z	Where the Z coordinate component is remapped

Keywords

Change Vector3 Vector2 Component Towards Look Variable Axis

Scale Product

Math » Geometry » Scale Product

Description

Multiplies two vectors component-wise

Parameters

Name	Description	
Set	Where the resulting value is set	
Direction 1	The first operand of the geometric operation that represents a direction	
Direction 2	The second operand of the geometric operation that represents a direction	

Keywords

Multiply Uniform Component Axis Position Location Variable

Set Direction

Math » Geometry » Set Direction

Description

Changes the value of a Vector3 that represents a direction in space

Parameters

Name	Description
Set	Dynamic variable where the resulting value is set
From	The value that is set

Keywords

Change Vector3 Vector2 Towards Look Variable

Set Point

Math » Geometry » Set Point

Description

Changes the value of a Vector3 that represents a position in space

Parameters

Name	Description
Set	Dynamic variable where the resulting value is set
From	The value that is set

Keywords

Change Vector3 Vector2 Position Location Variable

Set Vector X

Math » Geometry » Set Vector X

Description

Changes the X component of a vector

Parameters

Name	Description
Set	Where the resulting value is set
Х	The value that is changed for

Keywords

Change Component Axis

Set Vector Y

Math » Geometry » Set Vector Y

Description

Changes the Y component of a vector

Parameters

Name	Description
Set	Where the resulting value is set
Υ	The value that is changed for

Keywords

Change Component Axis

Set Vector Z

Math » Geometry » Set Vector Z

Description

Changes the Z component of a vector

Parameters

Name	Description
Set	Where the resulting value is set
Z	The value that is changed for

Keywords

Change Component Axis

Subtract Directions

Math » Geometry » Subtract Directions

Description

Subtracts two values that represent a direction in space and saves the result

Parameters

Name	Description
Set	Where the resulting value is set
Direction 1	The first operand of the geometric operation that represents a direction
Direction 2	The second operand of the geometric operation that represents a direction

Keywords

Minus Rest Position Location Variable

Subtract Points

Math » Geometry » Subtract Points

Description

Subtracts two values that represent a point in space and saves the result

Parameters

Name	Description
Set	Where the resulting value is set
Point 1	The first operand of the geometric operation that represents a point in space
Point 2	The second operand of the geometric operation that represents a point in space

Keywords

Rest Minus Position Location Variable

Transform to Local Direction

Math » Geometry » Transform to Local Direction

Description

Transform the Direction from World to Local space

Parameters

Name	Description
Set	Where the resulting value is set
Transform	The reference object to transform the coordinates
Direction	The direction that changes its space mode

Keywords

Direction Local World Space Variable Inverse

Transform to Local Point

Math » Geometry » Transform to Local Point

Description

Transform the Point from World to Local space

Parameters

Name	Description
Set	Where the resulting value is set
Transform	The reference object to transform the coordinates
Point	The point that changes its space mode

Keywords

Location Position Local World Space Variable Inverse

Transform to World Direction

Math » Geometry » Transform to World Direction

Description

Transform the Direction from Local to World space

Parameters

Name	Description
Set	Where the resulting value is set
Transform	The reference object to transform the coordinates
Direction	The direction that changes its space mode

Keywords

Direction Local World Space Variable

Transform to World Point

Math » Geometry » Transform to World Point

Description

Transform the Point from Local to World space

Parameters

Name	Description
Set	Where the resulting value is set
Transform	The reference object to transform the coordinates
Point	The point that changes its space mode

Keywords

Location Position Local World Space Variable

Uniform Scale

Math » Geometry » Uniform Scale

Description

Multiplies each component of a vector with a decimal

Parameters

Name	Description
Set	Where the resulting value is set
Vector	The first operand of the geometric operation that represents a direction
Value	The second operand of the geometric operation that represents a decimal number

Keywords

Direction Homogeneous Multiply Product

Shading

Shading Instructions

- Lerp Color
- Lerp Lightness
- Lerp Saturation
- Set Color

Lerp Color

Math » Shading » Lerp Color

Description

Linearly interpolates between to colors over time

Parameters

Name	Description
Color 1	The starting Color value
Color 2	The targeted Color value
Duration	How long it takes to perform the transition
Easing	The change rate of the transition over time
Wait to Complete	Whether to wait until the transition is finished or not
Set	Where the resulting Color value is set

Keywords

Change Value Transition Shade Tint Hue Colour Color Paint Tone

Lerp Lightness

Math » Shading » Lerp Lightness

Description

Linearly interpolates between to the desired lightness value over time

Parameters

Name	Description
Lightness	The targeted Lightness value (between 0 and 1)
Duration	How long it takes to perform the transition
Easing	The change rate of the transition over time
Wait to Complete	Whether to wait until the transition is finished or not
Set	Where the resulting Color value is set

Keywords

Change Value Transition Shade Tint Hue Colour Color Paint Tone

Lerp Saturation

Math » Shading » Lerp Saturation

Description

Linearly interpolates between to the desired saturation value over time

Parameters

Name	Description
Saturation	The targeted Saturation value (between 0 and 1)
Duration	How long it takes to perform the transition
Easing	The change rate of the transition over time
Wait to Complete	Whether to wait until the transition is finished or not
Set	Where the resulting Color value is set

Keywords

Change Value Transition Shade Tint Hue Colour Color Paint Tone

Set Color

Math » Shading » Set Color

Description

Sets the value of a Color

Parameters

Name	Description
Color	The Color value to set
Set	Where the resulting Color value is set

Keywords

Change Value Shade Tint Hue Colour Color Paint Tone

Text

Text Instructions

- Join
- Replace
- Set Text
- Substring

Join

Math » Text » Join

Description

Joins two string values and stores them

Parameters

Name	Description
Text 1	The source of the first text
Text 2	The source of the second text
Set	Where the resulting value is set

Keywords

Concat Concatenate Together Mix String Text Character

Replace

Math » Text » Replace

Description

Replaces all occurrences of a string with another string

Parameters

Name	Description
Text	The source of the text
Old Text	The text replaced
New Text	The text that replaces each occurrence
Set	Where the resulting value is set

Keywords

Substitute Change String Text Character

Set Text

Math » Text » Set Text

Description

Changes the value of a string

Parameters

Name	Description
Text	The source of the text
Set	Where the resulting value is set

Keywords

String Text Character

Substring

Math » Text » Substring

Description

Extracts a substring based on an index and length

Parameters

Name	Description
Text	The source of the text
Index	Starting index of the substring
Length	Amount of characters extracted
Set	Where the resulting value is set

Keywords

String Text Character

PHYSICS 2D

Physics 2D

Instructions

- Add Explosion Force 2D
- Add Force 2D
- · Change Mass 2D
- Change Velocity 2D
- Gravity Scale 2D
- Is Kinematic 2D

Add Explosion Force 2D

Physics 2D » Add Explosion Force 2D

Description

Applies a force to a Rigidbody2D that simulates explosion effects

Parameters

Name	Description
Rigidbody	The game object with a Rigidbody2D component that receives the force
Origin	The position where the explosion originates
Radius	How far the blast reaches
Force	The force of the explosion, which its at its maximum at the origin
Force Mode	How the force is applied

Keywords

Apply Velocity Impulse Propel Push Pull Boom Physics Rigidbody

Add Force 2D

Physics 2D » Add Force 2D

Description

Adds a force to a game object with a Rigidbody2D

Parameters

Name	Description
Rigidbody	The game object that will receive the force. A Rigidbody2D attached is required
Direction	The direction in which the force will be applied
Force	The amount of force applied
Force Mode	The type of force applied

Keywords

Apply Velocity Impulse Propel Push Pull Physics Rigidbody

Change Mass 2D

Physics 2D » Change Mass 2D

${\tt Description}$

Changes the mass of a Rigidbody2D

Parameters

ı	Name	Description
F	Rigidbody	The game object with a Rigidbody2D attached that will change its mass
N	1ass	The new mass the game object will be set to have

Keywords

Weight Physics Rigidbody

Change Velocity 2D

Physics 2D » Change Velocity 2D

Description

Changes the current velocity of a Rigidbody2D

Parameters

Name	Description
Rigidbody	The game object with a Rigidbody2D attached that will change its velocity
Velocity	The velocity the game object will change to

Keywords

Speed Movement Physics Rigidbody

Gravity Scale 2D

Physics 2D » Gravity Scale 2D

Description

Controls whether how gravity affects the Rigidbody2D

Parameters

Name	Description
Rigidbody	The game object with a Rigidbody2D attached that changes its gravity scale
Gravity Scale	The degree to which this object is affected by gravity

Keywords

Physics Rigidbody

Is Kinematic 2D

Physics 2D » Is Kinematic 2D

${\tt Description}$

Controls whether physics affects the Rigidbody2D $\,$

Parameters

Name	Description
Rigidbody	The game object with a Rigidbody2D attached that changes its kinematic usage
Is Kinematic	If enabled, forces, collisions or joints do not affect the rigidbody anymore

Keywords

Physics Rigidbody

PHYSICS 3D

Physics 3D

Instructions

- Add Explosion Force 3D
- Add Force 3D
- · Change Mass 3D
- Change Velocity 3D
- Is Kinematic 3D
- Overlap Box 2D
- Overlap Box 3D
- Overlap Circle 3D
- Overlap Sphere 3D
- Trace Line 3D
- Use Gravity 3D

Add Explosion Force 3D

Physics 3D » Add Explosion Force 3D

Description

Applies a force to a Rigidbody that simulates explosion effects

Parameters

Name	Description
Rigidbody	The game object with a Rigidbody component that receives the force
Origin	The position where the explosion originates
Radius	How far the blast reaches
Force	The force of the explosion, which its at its maximum at the origin
Force Mode	How the force is applied

Keywords

Apply Velocity Impulse Propel Push Pull Boom Physics Rigidbody

Add Force 3D

Physics 3D » Add Force 3D

Description

Adds a force to a game object with a Rigidbody

Parameters

Name	Description
Rigidbody	The game object with a Rigidbody component that receives the force
Direction	The direction in which the force is applied
Force	The amount of force applied
Force Mode	The type of force applied
Space Mode	Whether the force is applied in local or world space

Keywords

Apply Velocity Impulse Propel Push Pull Physics Rigidbody

Change Mass 3D

Physics 3D » Change Mass 3D

${\tt Description}$

Changes the mass of a Rigidbody

Parameters

Name	Description	
Rigidbody	The game object with a Rigidbody attached that changes its mass	
Mass	The new mass the game object	

Keywords

Weight Physics Rigidbody

Change Velocity 3D

Physics 3D » Change Velocity 3D

Description

Changes the current velocity of a Rigidbody

Parameters

Name	Description	
Rigidbody	The game object with a Rigidbody attached that changes its velocity	
Velocity	The velocity the game object changes to	

Keywords

Speed Movement Physics Rigidbody

Is Kinematic 3D

Physics 3D » Is Kinematic 3D

${\tt Description}$

Controls whether physics affects the Rigidbody

Parameters

Name	Description	
Rigidbody	The game object with a Rigidbody attached that changes its kinematic usage	
Is Kinematic	If enabled, forces, collisions or joints do not affect the rigidbody anymore	

Keywords

Physics Rigidbody

Overlap Box 2D

Physics 3D » Overlap Box 2D

Description

Captures all colliders caught inside a box

Parameters

Name	Description
Center	The center of the box
Size	The size of the box in each axis
Angle	The rotation of the box in world space
Store In	The list where the colliders (if any) are stored
Layer Mask	A mask that determines which colliders are ignored and which aren't

Keywords

Cube Cast Collect Physics Rigidbody

Overlap Box 3D

Physics 3D » Overlap Box 3D

Description

Captures all colliders caught inside a box

Parameters

Name	Description
Center	The center of the box
Half Extents	Half of the size of the box in each axis
Rotation	The rotation of the box in world space
Store In	The list where the colliders (if any) are stored
Layer Mask	A mask that determines which colliders are ignored and which aren't

Keywords

Cube Cast Collect Physics Rigidbody

Overlap Circle 2D

Physics 3D » Overlap Circle 3D

Description

Captures all colliders caught inside a Circle defined by a point and radius

Parameters

Name	Description
Center	The center of the circle
Radius	The radius of the circle
Store In	The list where the colliders (if any) are stored
Layer Mask	A mask that determines which colliders are ignored and which aren't

Keywords

Cast Collect Physics Rigidbody

Overlap Sphere 3D

Physics 3D » Overlap Sphere 3D

Description

Captures all colliders caught inside a sphere defined by a point and radius

Parameters

Name	Description
Center	The center of the sphere
Radius	The radius of the sphere
Store In	The list where the colliders (if any) are stored
Layer Mask	A mask that determines which colliders are ignored and which aren't

Keywords

Circle Cast Collect Physics Rigidbody

Trace Line 3D

Physics 3D » Trace Line 3D

Description

Captures all colliders caught inside a line between A and B $\,$

Parameters

Name	Description
Point A	The position of the first point
Point B	The position of the second point
Store In	The list where the colliders (if any) are stored
Layer Mask	A mask that determines which colliders are ignored and which aren't

Keywords

Line Trace Raycast Cast Collect Physics Rigidbody

Use Gravity 3D

Physics 3D » Use Gravity 3D

${\tt Description}$

Controls whether gravity affects the Rigidbody

Parameters

Name	Description	
Rigidbody	The game object with a Rigidbody attached that changes its gravity usage	
Use Gravity	If set to false the rigidbody behaves as in outer space	

Keywords

Physics Rigidbody

RENDERER

Renderer

Instructions

- · Change Material Color
- Change Material Float
- · Change Material Texture
- · Change Material
- Change Sprite

Change Material Color

Renderer » Change Material Color

Description

Changes over time the Color property of an instantiated material of a Renderer component

Parameters

Name	Description
Property	Name of the property to change
Color	Color target that the instantiated Material turns into
Duration	How long it takes to perform the transition
Easing	The change rate of the transition over time
Wait to Complete	Whether to wait until the transition is finished or not
Renderer	The game object with a Renderer component attached

Keywords

Set Shader Hue Change

Change Material Float

Renderer » Change Material Float

Description

Changes over time the Float property of an instantiated material of a Renderer component

Parameters

Name	Description
Property	Name of the property to change
Float	Decimal target that the instantiated Material's property turns into
Duration	How long it takes to perform the transition
Easing	The change rate of the transition over time
Wait to Complete	Whether to wait until the transition is finished or not
Renderer	The game object with a Renderer component attached

Keywords

Set Shader Hue Change

Change Material Texture

Renderer » Change Material Texture

Description

Changes the main texture of an instantiated material of a Renderer component

Parameters

Name	Description
Texture	Texture that replaces the Renderer's instantiated material
Renderer	The game object with a Renderer component attached

Keywords

Set Shader Change

Change Material

Renderer » Change Material

Description

Changes instantiated material of a Renderer component

Parameters

Name	Description
Material	Material that is set as the primary type of the Renderer
Renderer	The game object with a Renderer component attached

Keywords

Set Shader Texture Change

Change Sprite

Renderer » Change Sprite

Description

Sets the Sprite value

Parameters

Name	Description
То	Where to store the new Sprite value
Sprite	The Sprite value to be stored

Keywords

Texture Renderer

SCENES

Scenes

Instructions

- · Load Scene
- Unload Scene

Load Scene

Scenes » Load Scene

Description

Loads a new Scene

Parameters

Name	Description
Scene	The scene to be loaded
Mode	Single mode replaces all other scenes. Additive mode loads the scene on top of the others
Async	Loads the scene in the background or freeze the game until its done
Scene Entries	Define the starting location of the player and other characters after loading the scene

Keywords

Change

Unload Scene

Scenes » Unload Scene

Description

Unloads an active scene

Parameters

Name	Description
Scene	The scene to be unloaded

Keywords

Change Remove

STORAGE

Storage

Instructions

- Delete Game
- Load Game
- · Load Latest Game
- Reset Game
- Save Game

Delete Game

Storage » Delete Game

Description

Deletes a previously saved game state

Parameters

Name Description

Save Slot Slot number that is erased. Default is 1

Keywords

Load Save Delete Profile Slot Game Session

Load Game

Storage » Load Game

Description

Loads a previously saved state of a game

Parameters

Name	Description
Save Slot	ID number to load the game from. It can range between 1 and 9999

Keywords

Load Save Profile Slot Game Session

Load Latest Game

Storage » Load Latest Game

Description

Loads the latest previously saved state of a game $\,$

Keywords

Load Save Last Profile Game Session

Reset Game

Storage » Reset Game

${\tt Description}$

Resets the current game to its default values

Parameters

Name	Description	
Scene	The scene to move after resetting the data	

Keywords

Load Save Profile Slot Game Session

Save Game

Storage » Save Game

Description

Saves the current state of the game

Parameters

Name	Description
Save Slot	ID number to save the game. It can range between 1 and 9999

Keywords

Load Save Profile Slot Game Session

TESTING

Testing

Instructions

• Instruction Tester

Tester

Testing » Instruction Tester

Description

Appends a character to a static Chain field. For internal testing use only

Parameters

Name Description

Character A character that will be appended to InstructionTester.Chain

Example 1

Note that this Instruction is not accessible through the Inspector to avoid confusing new users. To run the test suit environment, create a new InstructionList object and append as many InstructionTester instances as your test requires.

```
InstructionList instructions = new InstructionList(
    new InstructionTester('a'),
    new InstructionTester('b'),
    new InstructionTester('c')
);

InstructionTester.Clear();
instructions.Run(null);

Debug.Log(InstructionTester.Chain);
// Prints: 'abc'
```

This instruction is for internal testing only.

TIME

Time

Instructions

- Time Scale
- Wait Frames
- · Wait Seconds

Time Scale

Time » Time Scale

Description

Changes the Time Scale of the game

Parameters

Name	Description
Time Scale	The scale at which time passes. This can be used for slow motion effects
Blend Time	How long it takes to transition from the current time scale to the new one
Layer	Any time scale values using the same Layer is overwritten by this one.

Example 1

Setting a Time Scale of 0 will freeze the game. Useful for pausing the game

Example 2

The resulting Time Scale will be equal to the lowest time scale value between all Layers. For example, if the Time Scale with Layer = 0 has a value of 0.5 (which makes characters move in slow motion), and another Time Scale with Layer = 1 with a value of 0, the resulting Time Scale will be 0

Keywords

Slow Motion Bullet Time Matrix

Wait Frames

Time » Wait Frames

Description

Waits a certain amount of frames

Parameters

Name	Description
Frames	The amount of frames to wait

Example 1

This instruction is particularly useful in cases where you want to control the order of execution of two Actions. For example, imagine there are two Triggers executing at the same time, but you want to execute the instructions associated with one after the execution of the other one. You can use the 'Wait Frames' instruction to defer its execution 1 frame so the other one has had time to complete its own execution

Keywords

Wait Time Frames Yield

Wait Seconds

Time » Wait Seconds

Description

Waits a certain amount of seconds

Parameters

Name	Description
Seconds	The amount of seconds to wait
Mode	Whether to use the time scale or not

Keywords

Wait Time Seconds Minutes Cooldown Timeout Yield

TRANSFORMS

Transforms

Instructions

- Change Position
- · Change Rotation
- · Change Scale
- · Clear Parent
- · Look At
- Set Parent

Change Position

Transforms » Change Position

Description

Changes the position of a game object over time $% \left(1\right) =\left(1\right) \left(1\right$

Parameters

Name	Description
Position	The desired position of the game object
Space	If the transformation occurs in local or world space
Duration	How long it takes to perform the transition
Easing	The change rate of the translation over time
Wait to Complete	Whether to wait until the translation is finished or not
Transform	The Transform of the game object

Keywords

Location Translate Move Displace Set

Change Rotation

Transforms » Change Rotation

Description

Changes the rotation of a game object over time

Parameters

Name	Description
Rotation	The desired rotation of the game object
Space	If the transformation occurs in local or world space
Duration	How long it takes to perform the transition
Easing	The change rate of the rotation over time
Wait to Complete	Whether to wait until the rotation is finished or not
Transform	The Transform of the game object

Keywords

Rotate Angle Euler Tilt Pitch Yaw Roll

Change Scale

Transforms » Change Scale

Description

Changes the local scale of a game object over time

Parameters

Name	Description
Scale	The desired scale of the game object
Duration	How long it takes to perform the transition
Easing	The change rate of the scaling over time
Wait to Complete	Whether to wait until the scaling is finished or not
Transform	The Transform of the game object

Keywords

Size Resize Grow Reduce Small Big

Clear Parent

Transforms » Clear Parent

Description

Clears the parent of a game object

Parameters

Name Description

Transform The Transform of the game object

Keywords

Child Children Hierarchy Orphan

Look At

Transforms » Look At

Description

Rotates the transform towards the chosen target

Parameters

Name	Description
Target	The desired targeted object to look at
Transform	The Transform of the game object

Keywords

Rotate Rotation See

Set Parent

Transforms » Set Parent

${\tt Description}$

Changes the parent of a game object

Parameters

Name	Description
Parent	The game object that becomes the parent
Transform	The Transform of the game object

Keywords

Child Children Hierarchy Hang Inherit

UI

U

Instructions

- · Canvas Group Alpha
- Canvas Group Block Raycasts
- · Canvas Group Interactable
- Change Dropdown
- Change Font Size
- Change Graphic Color
- · Change Height
- Change Image
- · Change Input Field
- Change Slider
- · Change Text
- · Change Toggle
- · Change Width
- Focus On
- Submit
- Unfocus

Canvas Group Alpha

UI » Canvas Group Alpha

Description

Changes the opacity of the Canvas Group and affects all of its children

Name	Description
Canvas Group	The Canvas Group component that changes its value
Alpha	The new opacity value transformation of the Canvas Group
Duration	How long it takes to perform the transition
Easing	The change rate of the parameter over time
Wait to Complete	Whether to wait until the transition is finished

Canvas Group Block Raycasts

UI » Canvas Group Block Raycasts

Description

Changes whether the Canvas Group blocks raycasts or not

Name	Description
Canvas Group	The Canvas Group component that changes its value
Block Raycasts	If true, the canvas group and its children block raycasts

Canvas Group Interactable

UI » Canvas Group Interactable

Description

Changes the interactable value of a Canvas Group component

Name	Description
Canvas Group	The Canvas Group component that changes its value
Interactable	The on/off state value

Change Dropdown

UI » Change Dropdown

Description

Changes the value of a Dropdown or Text Mesh Pro Dropdown component

Name	Description
Text	The Text or Text Mesh Pro component that changes its value
Index	The new index value of the Dropdown

Change Font Size

UI » Change Font Size

Description

Changes the size of the Text or Text Mesh Pro component content

Parameters

Name	Description
Text	The Text or Text Mesh Pro component that changes its font size
Size	The new text size, in pixels

Keywords

Text

Change Graphic Color

UI » Change Graphic Color

Description

Changes the color of a Graphic component

Name	Description
Graphic	The Graphic component that changes its tint color
Color	The new Color

Change Height

UI » Change Height

${\tt Description}$

Changes the Height of a Rect Transform

Name	Description
Rect Transform	The Rect Transform component to change
Height	The new height value. Also known as sizeDelta.y

Change Image

UI » Change Image

${\tt Description}$

Changes the Sprite of an Image component

Name	Description
Override Sprite	If the Sprite replaced is the original or the overriden
Image	The Image component that changes its sprite value
Sprite	The new Sprite reference

Change Input Field

UI » Change Input Field

Description

Changes the value of an Input Field or Text Mesh Pro Input Field

Name	Description
Input Field	The Input Field or TMP Input Field component that changes its value
Value	The new value set

Change Slider

UI » Change Slider

${\tt Description}$

Changes the value of a Slider component

Name	Description
Slider	The Slider component that changes its value
Value	The new value set

Change Text

UI » Change Text

${\tt Description}$

Changes the value of a Text or Text Mesh Pro component

Name	Description
Text	The Text or Text Mesh Pro component that changes its value
Value	The new value set

Change Toggle

UI » Change Toggle

${\tt Description}$

Changes the value of a Toggle component

Name	Description
Toggle	The Toggle component that changes its value
Value	The new value set

Change Width

UI » Change Width

Description

Changes the Width of a Rect Transform

Name	Description
Rect Transform	The Rect Transform component to change
Width	The new width value. Also known as sizeDelta.x

Focus On

UI » Focus On

${\tt Description}$

Focuses on a specific UI component

Parameters

Name	Description
Focus On	The UI component that takes focus

Keywords

Select

Submit

UI » Submit

${\tt Description}$

Performs a submit action on a UI element

Keywords

Enter Press Confirm

Unfocus

UI » Unfocus

${\tt Description}$

Removes the focus from any UI component

Keywords

Deselect Lose

VARIABLES

Variables

Instructions

- · Change Id
- · Clear List
- · Collect Characters
- · Collect Markers
- Filter List
- Iterator Next
- Iterator Previous
- Iterator Random
- Loop List
- Move List
- Remove From List
- Reverse List
- Shuffle List
- Sort List Alphabetically
- Sort List By Distance
- Swap List

Change ID

Variables » Change ID

Description

Changes the Local Name or List Variable's ID. It only works on non-Savable variables

Parameters

Name	Description
ID	The new ID of the Local Variable

Keywords

Unique Guid

Clear List

Variables » Clear List

Description

Removes all elements of a given Local or Global List Variables

Parameters

Name	Description
List Variable	Local List or Global List which elements are removed

Keywords

Clean Remove Delete Destroy Size Array List Variables

Collect Characters

Variables » Collect Characters

Description

Collects all Characters that within a certain radius of a position $% \left(1\right) =\left(1\right) +\left(1\right)$

Parameters

Name	Description
Origin	The position where the rest of the game objects are collected
Max Radius	How far from the Origin the game objects are collected
Min Radius	How far from the Origin game objects start to be collected
Store In	List where the collected game objects are saved
Filter	Checks a set of Conditions with each collected game object

Example 1

Note that in most cases it is not desirable to set the Min Radius to 0. Doing so will also collect game objects at a distance of 0 from the Origin. For example, if we want to collect all enemies around the Player and we set a Min Radius of 0, the Player will also be collected because it's a Character at a distance 0 from himself

Keywords

Gather Get Set Array List Variables

Collect Markers

Variables » Collect Markers

Description

Collects all Markers that within a certain radius of a position

Parameters

Name	Description
Origin	The position where the rest of the game objects are collected
Max Radius	How far from the Origin the game objects are collected
Min Radius	How far from the Origin game objects start to be collected
Store In	List where the collected game objects are saved
Filter	Checks a set of Conditions with each collected game object

Example 1

Note that in most cases it is not desirable to set the Min Radius to 0. Doing so will also collect game objects at a distance of 0 from the Origin. For example, if we want to collect all enemies around the Player and we set a Min Radius of 0, the Player will also be collected because it's a Character at a distance 0 from himself

Keywords

Gather Get Set Array List Variables

Filter List

Variables » Filter List

Description

Checks Conditions against each element of a list and removes it if the Condition is not true

Parameters

Name	Description
List Variable	Local List or Global List which elements are filtered
Filter	Checks a set of Conditions with each collected game object and removes the element if the Condition is not true

Example 1

The Filter field runs the Conditions list for each element in a Local List Variables or Global List Variables. It sets as the 'Target' value the currently examined game object. For example, filtering by the tag name 'Enemy' can be done using the 'Tag' Condition and comparing the field 'Target' with the string 'Enemy'. All game objects that are not tagged as 'Enemy' are removed

Keywords

Remove Pick Select Array List Variables

Iterator Next

Variables » Iterator Next

Description

Increases in one unit the value used as an iterator for a List Variable

Parameters

Name	Description
Index	The numeric value used as an index
List Variables	The List Variable targeted
Mode	Whether the index loops back to the first index or is clamped

Keywords

Iterate Index For Loop Access

Iterator Previous

Variables » Iterator Previous

Description

Decreases in one unit the value used as an iterator for a List Variable

Parameters

Name	Description
Index	The numeric value used as an index
List Variables	The List Variable targeted
Mode	Whether the index loops back to the last index or is clamped at zero

Keywords

Iterate Index For Loop Access

Iterator Random

Variables » Iterator Random

Description

Sets a random value between zero and the list count

Parameters

Name	Description
Index	The numeric value used as an index
List Variables	The List Variable targeted

Keywords

Iterate Index For Loop Access

Loop List

Variables » Loop List

Description

Loops a Game Object List Variables and executes an Actions component for each value

Parameters

Name	Description
List Variable	Local List or Global List which elements are iterated
Actions	The Actions component executed for each element in the list. The Target argument of any Instruction contains the object inspected

Keywords

Iterate Cycle Every All Stack

Move List

Variables » Move List

Description

Move a position from a list to another position

Parameters

Name	Description
List Variable	Local List or Global List which elements are moved

Keywords

Order Change Array List Variables

Remove from List

Variables » Remove from List

Description

Deletes an elements from a given Local or Global List Variables

Parameters

List Variable Local List or Global List which elements are removed	Name	Description
	List Variable	Local List or Global List which elements are removed

Keywords

Delete Destroy Size Array List Variables

Reverse List

Variables » Reverse List

Description

Reorders the elements of a list so the first ones become the last ones

Parameters

Name	Description
List Variable	Local List or Global List which elements are reversed

Keywords

Invert Order Sort Array List Variables

Shuffle List

Variables » Shuffle List

Description

Randomly shuffles the position of each element on a List Variable

Parameters

Name	Description	
List Variable	Local List or Global List which elements are shuffled	

Keywords

Randomize Sort Array List Variables

Sort List Alphabetically

Variables » Sort List Alphabetically

Description

Sorts the List Variable elements based on their alphabet distance

Parameters

Name	Description
List Variable	Local List or Global List which elements are sorted
Order	Sort alphabetically ascending or descending
Ignore Case	Whether the string comparison should ignore upper/lower case

Keywords

Order Organize Array List Variables

Sort List by Distance

Variables » Sort List by Distance

Description

Sorts the List Variable elements based on their distance to a given position

Parameters

Name	Description
List Variable	Local List or Global List which elements are sorted
Position	The reference position that is used to measure the sorting distance
Order	From Closest to Farthest puts the closest elements to the Position first

Keywords

Order Organize Array List Variables

Swap List

Variables » Swap List

Description

Swaps two positions of a list

Parameters

Name Description

List Variable Local List or Global List which elements are swapped

Keywords

Order Change Array List Variables

VISUAL SCRIPTING

Visual Scripting

Instructions

- Activate Hotspots
- Broadcast Message
- Check Conditions
- Emit Signal
- Invoke Method
- Restart Instructions
- Run Actions
- Run Conditions
- Run Trigger
- Stop Actions
- Stop Conditions
- Stop Trigger

Activate Hotspots

Visual Scripting » Activate Hotspots

Description

Determines whether Hotspots can be activated or are inactive by type

Parameters

Name	Description
Туре	The type of Hotspots to activate or deactivate
Active	Determines if Hotspots can run or are inactive

Keywords

Execute Enable Disable Show Hide Deactivate

Broadcast Message

Visual Scripting » Broadcast Message

Description

Invokes any method on any component found on the target game object

Parameters

Name	Description
Game Object	The target game object that receives the broadcast message
Message	The name of the method or methods that are called
Send Upwards	If true the message travels from the game object towards the root

Example 1

By default all broadcast messages travel from the target game object and towards all its children. Setting the Send Upwards field to true makes the message travel from the game object towards the root parent

Keywords

Execute Call Invoke Function

Check Conditions

Visual Scripting » Check Conditions

Description

If any of the Conditions list is false it early exits and skips the execution of the rest of the Instructions below

Parameters

Name	Description
Conditions	List of Conditions that can evaluate to true or false
Mode	Whether to check the Conditions as an AND or an OR set

Keywords

Execute Call Check Evaluate

Emit Signal

Visual Scripting » Emit Signal

Description

Emits a specific signal, which is captured by other listeners

Parameters

Name Description
Signal The signal name emitted

Keywords

Event Raise Command Fire Trigger Dispatch Execute

Invoke Method

Visual Scripting » Invoke Method

Description

Invokes a method from any script attached to a game object

Parameters

Name	Description
Method	The method/function that is called on a game object reference

Keywords

Execute Call Invoke Function

Restart Instructions

Visual Scripting » Restart Instructions

Description

Stops executing the current list of Instructions and starts again from the top

Keywords

Reset Call Again

Run Actions

Visual Scripting » Run Actions

${\tt Description}$

Executes an Actions component object

Parameters

Name	Description
Actions	The Actions object that is executed
Wait Until Complete	If true this instruction waits until the Actions object finishes running

Keywords

Execute Call Instruction Action

Run Conditions

Visual Scripting » Run Conditions

Description

Executes a Conditions component object

Parameters

Name	Description
Conditions	The Conditions object that is executed
Wait Until Complete	If true this instruction waits until the Conditions object finishes running

Keywords

Execute Call Check Evaluate

Run Trigger

Visual Scripting » Run Trigger

${\tt Description}$

Executes a Trigger component object

Parameters

Name	Description
Trigger	The Trigger object that is executed
Wait Until Complete	If true this instruction waits until the Trigger object finishes running

Keywords

Execute Call

Stop Actions

Visual Scripting » Stop Actions

Description

Stops an Actions component object that is being executed

Parameters

Name	Description
Actions	The Actions object that is stopped

Keywords

Cancel Pause

Stop Conditions

Visual Scripting » Stop Conditions

Description

Stops a Conditions component object that is being executed

Parameters

Name	Description	
Conditions	The Conditions object that is stopped	

Keywords

Cancel Pause

Stop Trigger

Visual Scripting » Stop Trigger

Description

Stops a Trigger component object that is being executed

Parameters

Name	Description
Trigger	The Trigger object that is stopped

Keywords

Cancel Pause

Custom Instructions

Game Creator allows to very easily create custom Instructions and use them along with the rest.



This section assumes you have some programming knowledge. If you don't know how to code you might be interested in checking out the Game Creator Hub page. Programmers altrusitically create custom **Instructions** for others to download and use in their project.

CREATING AN INSTRUCTION

The easiest way to create an **Instruction** C# script is to right click on your *Project* panel and select *Create Game Creator Developer C# Instruction*. This will create a template script with the boilerplate structure of an Instruction:

```
using System;
using GameCreator.Runtime.Common;
using GameCreator.Runtime.VisualScripting;

[Serializable]
public class MyInstruction : Instruction
{
    protected override Task Run(Args args)
    {
        // Your code here...
        return DefaultResult;
    }
}
```

Anatomy of an Instruction

An **Instruction** is a class that inherits from the Instruction super class. The abstract Run(...) method is the entry point of an **Instruction**'s execution, which is automatically called when it's this instruction's time to be executed.

The Run(...) method has a single parameter of type Args, which is a helper class that contains a reference to the game object that initiated the call (args.Self) and the targeted game object (args.Target), if any.

Yielding in Time

Most instruction will be executed in a single frame. However, some instructions might require to put the execution on hold for a certain amount of time, before resuming the execution. The most simple example is with the "Wait for Seconds" instruction, which pauses the execution for a few seconds before resuming.

The Instruction super class contains a collection of methods that helps with time management.



Instructions use the async/await methodology to manage the flow of an instruction over the course of time. Using the await symbol requires the Run() method to have the async symbol on its method definition:

```
protected override async Task Run(Args args)
{ }
```

NextFrame

The NextFrame() methods pauses the execution of the Instruction for a single frame, then resumes.

```
protected override async Task Run(Args args)
{
    await this.NextFrame();
}
```

Time

The Time(float time) method pauses the execution of an Instruction for a certain amount of time. The time parameter is in seconds.

```
protected override async Task Run(Args args)
{
   await this.Time(5f);
}
```

While

The While(Func<bool> function) method pauses the execution of an Instruction for as long as the result of the method passed as a parameter returns true. This method is executed every frame and the execution will resume as soon as it returns false.

```
protected override async Task Run(Args args)
{
    await this.While(() => this.IsPlayerMoving());
}
```

Until

The Until(Func<bool> function) method pauses the execution of an Instruction for as long as the result of the method passed as a parameter returns true. This method is executed every frame and the execution will resume as soon as it returns true.

```
protected override async Task Run(Args args)
{
    await this.Until(() => this.PlayerHasReachedDestination());
}
```

Decoration & Documentation

It is highly recommended to document and decorate the **Instruction** so it's easier to find and use. It is done using class-type attributes that inform **Game Creator** of the quirks of this particular instruction.

For example, to set the title of an instruction to "Hello World", use the [Title(string name)] attribute right above the class definition:

```
using System;
using System.Threading.Tasks;
using GameCreator.Runtime.Common;
using GameCreator.Runtime.VisualScripting;

[Title("Hello World")]
[Serializable]
public class MyInstruction : Instruction
{
    protected override Task Run(Args args)
    {
        // ...
    }
}
```

Title

The title of the Instruction. If this attribute is not provided, the title will be a beautified version of the class name.

```
[Title("Title of Instruction")]
```

Description

A description of what the Instruction does. This is both used in the floating window documentation, as well as the description text when uploading an Instruction to the Game Creator Hub.

```
[Description("Lorem Ipsum dolor etiam porta sem magna mollis")]
```

Image

The [Image(...)] attribute changes the default icon of the Instruction for one of the default ones. It consists of 2 parameters:

- Icon [Type]: a Type class of an IIcon derived class. Game Creator comes packed with a lot of icons although you can also create your own.
- · Color [Color]: The color of the icon. Uses Unity's Color class.

For example, one of the icons included is the "Solid Cube" icon. To display a red solid cube as the icon of the instruction, use the following attribute:

```
[Image(typeof(IconCubeSolid), Color.red)]
```

Category

A sequence of sub-categories organized using the slash (/) character. This attribute helps keep the Instructions organized when the Instructions list dropdown is displayed.

```
[Category("Category/Sub Category/Name")]
```

The example above will display the Instruction under the sub directory Category Sub Category Name.

Version

A semmantic version to keep track of the development of this Instruction. It's important to note that when updating an Instruction to the Game Creator Hub, the version number must always be higher than the one on the server.

The semmantic version follows the standard *Major Version*, *Minor Version*, *Patch Version*. To know more about how semmantic versioning works, read the following page: https://semver.org.

```
[Version(1, 5, 3)]
```

Parameters

When an Instruction has exposed fields in the Inspector, it's a good idea to document what these do. You can add as many [Parameter(name, description)] attributes as exposed fields has the Instruction.

For example, if the Instruction has these two fields:

```
public bool waitForTime = true;
public float duration = 5f;
```

You can document those fields adding:

```
[Parameter("Wait For Time", "Whether to wait or not")]
[Parameter("Duration", "The amount of seconds to wait")]
```

Keywords

Keywords are strings that help the fuzzy finder more easily search for an instruction. For example, the "Change Position" instruction doesn't reference the word "move" or "translate" anywhere in its documentation. However, these words are very likely to reference this instruction when the user types them in the search box.

```
[Keywords("Move", "Translate")]
```

Example

The Example attribute allows to display a text as an example of use of this Instruction. There can be more than one [Example(...)] attribute per instruction. This is particularly useful when uploading instructions on the Game Creator Hub.



It is recommended to use Markdown notation when writing examples

[Example("Sed posuere consectetur est at lobortis)]



You can use the @ character in front of a string to break the example text in multiple lines. To create a new paragraph, simply add two new lines. For example:

```
[Example(@"
    This is the first paragraph.
    This is also in the first paragraph, right after the previous sentence
    This line is part of a new paragraph.
)]
```

Dependency

This attribute is optional and only used in the Game Creator Hub. If this Instruction uses some particular feature of a specific module, it will first check if the user downloading this instruction has that module installed. If it does not, it will display an error message and forbid downloading it. This is useful to avoid throwing programming errors.

The [Dependency(...)] attribute consists of 4 parameters:

- · Module ID: For example, the ID of the Inventory module is gamecreator.inventory.
- Major Version: The minimum major version of the dependency module.
- Minor Version: The minimum minor version of the dependency module.
- Patch Version: The minimum patch version of the dependency module.

[Dependency("gamecreator.inventory", 1, 5, 2)]

1.5.3 Triggers

Triggers

Triggers are components attached to game objects that listen to events that happen on the scene and react by executing a sequence of instructions.

Triggers



In the image above, the **Trigger** is listening for the *Space* keyboard key to be pressed down. As soon as that happens, it calls the instructions list from below, which prints the message "Space key pressed!"

CREATING A TRIGGER

Right click on the *Hierarchy* panel and select *Game Creator Visual Scripting Trigger*. A game object named 'Trigger' will appear in the scene with a component of the same name.

Alternatively you can also add the **Trigger** component to any game object clicking on the Inspector's Add Component button and searching for Trigger.

Veleting Triggers

To delete a Trigger component, simply click on the component's little cog button and select "Remove Component" from the dropdown menu.

CHANGING THE EVENT

Triggers listen to very specific events, chosen by the user. To change the type of **Event** a Trigger listens, click on the event name and a dropdown menu will appear. Navigate it using the mouse or searching for a specific event in the seach box field.

Change Trigger Event

INSTRUCTIONS

The **Instructions** list that appear below work exactly the same was the **Actions** component. For more information about this component, visit the **Actions** page.

Events

EVENTS

Sub Categories

- Audio
- Cameras
- Characters
- Input
- Interactive
- Lifecycle
- Logic
- Physics
- Storage
- Ui
- Variables

AUDIO

Audio

Events

- On Change Ambient Volume
- On Change Master Volume
- On Change Music Volume
- On Change Sound Effects Volume
- On Change Speech Volume
- On Change Ui Volume

On Change Ambient Volume

Audio » On Change Ambient Volume

Description

Executed when the Ambient Volume is changed

Keywords

On Change Master Volume

Audio » On Change Master Volume

Description

Executed when the Master Volume is changed

Keywords

On Change Music Volume

Audio » On Change Music Volume

Description

Executed when the Music Volume is changed

Keywords

On Change Sound Effects Volume

Audio » On Change Sound Effects Volume

Description

Executed when the Sound Effects Volume is changed

Keywords

On Change Speech Volume

Audio » On Change Speech Volume

Description

Executed when the Speech Volume is changed

Keywords

On Change UI Volume

Audio » On Change UI Volume

Description

Executed when the UI Volume is changed

Keywords

CAMERAS

Cameras

Events

- On Camera Change
- On Change From Shot
- On Change To Shot

On Camera Change

Cameras » On Camera Change

Description

Executed when the Camera changes to another Camera Shot

Keywords

Shot Switch Cut

On Change from Shot

Cameras » On Change from Shot

Description

Executed when the Camera Shot is deactivated

Keywords

Shot Switch Cut

On Change to Shot

Cameras » On Change to Shot

Description

Executed when the Camera Shot is activated

Keywords

Shot Switch Cut

CHARACTERS

Characters

Sub Categories

- Combat
- Navigation
- Ragdoll

Events

- On Become Npc
- On Become Player
- On Change Model
- On Die
- On Revive

On Become NPC

Characters » On Become NPC

Description

Executed when a character that is a Player becomes an $\ensuremath{\mathsf{NPC}}$

On Become Player

Characters » On Become Player

Description

Executed when a character becomes the Player

On Change Model

Characters » On Change Model

Description

Executed when a character changes its model

On Die

Characters » On Die

Description

Executed when the character dies

On Revive

Characters » On Revive

Description

Executed when a dead character revives

Keywords

Resurrect Respawn

Combat

Combat Events

- On Defense Change
- On Dodge
- On Invincibility Change
- On Poise Break
- On Poise Change
- On Target Change

On Defense Change

Characters » Combat » On Defense Change

Description

Executed when the Character's defense changes

Keywords

Defend Block Combat

On Dodge

Characters » Combat » On Dodge

Description

Executed every time the character evades an attack

On Invincibility Change

Characters » Combat » On Invincibility Change

Description

Executed when the character's Invincibility changes

On Poise Break

Characters » Combat » On Poise Break

Description

Executed when a character's Poise is broken

Keywords

Resistance Combat

On Poise Change

Characters » Combat » On Poise Change

Description

Executed every time the character's combat Poise changes

Keywords

Resistance Combat

On Target Change

Characters » Combat » On Target Change

Description

Executed every time the character's combat Target changes

Keywords

Focus Combat Aim

Navigation

Navigation Events

- On Dash
- On Jump
- On Land
- On Navlink Enter
- On Navlink Exit
- On Step

On Dash

Characters » Navigation » On Dash

Description

Executed every time the character performs a dash

On Jump

Characters » Navigation » On Jump

Description

Executed every time the character performs a jump

On Land

Characters » Navigation » On Land

Description

Executed every time the character lands on the ground

On NavLink Enter

Characters » Navigation » On NavLink Enter

Description

Executed when a character enters a navigation mesh Off Mesh Link

On NavLink Exit

Characters » Navigation » On NavLink Exit

Description

Executed when a character exists a navigation mesh Off Mesh Link

On Step

Characters » Navigation » On Step

Description

Executed every time the character takes a step

Keywords

Footstep Foot Feet Ground

Ragdoll

Ragdoll Events

- On Recover Ragdoll
- On Start Ragdoll

On Recover Ragdoll

Characters » Ragdoll » On Recover Ragdoll

Description

Executed when the character recovers from the ragdoll mode

On Start Ragdoll

Characters » Ragdoll » On Start Ragdoll

Description

Executed when the character enters the ragdoll mode

INPUT

Input

Events

- On Cursor Click
- On Input Button
- On Input Flick
- On Touch

On Cursor Click

Input » On Cursor Click

Description

Detects when the cursor clicks this game object

Parameters

Name	Description
Button	The mouse button to detect
Min Distance	If set to None, the mouse input acts globally. If set to Game Object, the event only fires if the target object is within a certain radius

Keywords

Down Mouse Button Hover Left Middle Right

On Input Button

Input » On Input Button

Description

Detects when a button is interacted with

Parameters

Name	Description
Button	The button that triggers the event
Min Distance	If set to None, the input acts globally. If set to Game Object, the event only fires if the target object is within the specified radius

Keywords

Down Up Press Release Keyboard Mouse Button Gamepad Controller Joystick

On Input Flick

Input » On Input Flick

Description

Detects when Input (Vector 2) is flicked

Parameters

Name	Description
Value	The Input value read
Compare	The comparison applied to the input value
Min Distance	If set to None, the input acts globally. If set to Game Object, the event only fires if the target object is within the specified radius

Keywords

Left Right Down Up Press Move Direction Keyboard Mouse Button Gamepad Controller Joystick

On Touch

Input » On Touch

Description

Detects when a finger touches this game object on a touchscreen

Parameters

Name	Description
Min Distance	If set to None, the touch input acts globally. If set to Game Object, the event only fires
	if the target object is within a certain radius

Keywords

Down Finger Press Click Finger Press Click

INTERACTIVE

Interactive

Events

- On Blur
- On Focus
- On Interact

On Blur

Interactive » On Blur

${\tt Description}$

Executed when the Character loses focus on this Interactive object

On Focus

Interactive » On Focus

${\tt Description}$

Executed when the Character focuses on this Interactive object

On Interact

Interactive » On Interact

Description

Executed when a Character interacts with this Trigger

Parameters

Name	Description
Use Raycast	Checks if there is something between the character and the Trigger

Example 1

The 'Use Raycast' option checks if there is no other collider between the Character and the Trigger

LIFECYCLE

Lifecycle

Events

- On App Focus
- On App Pause
- On App Quit
- On Become Invisible
- On Become Visible
- On Disable
- On Enable
- On Fixed Update
- On Interval
- On Invoke
- On Late Update
- On Start
- On Update

On App Focus

Lifecycle » On App Focus

${\tt Description}$

Executed when the standalone application is brought to focus $% \left(1\right) =\left(1\right) \left(1\right)$

Keywords

Foreground

On App Pause

Lifecycle » On App Pause

${\tt Description}$

Executed when the standalone application loses its focus

Keywords

Background Suspend

On App Quit

Lifecycle » On App Quit

${\tt Description}$

Executed right before exiting the standalone application

Keywords

Exit Close

On Become Invisible

Lifecycle » On Become Invisible

Description

Executed when the game object it is attached to is no longer visible by any camera ${\sf E}$

Keywords

Hide Disappear

On Become Visible

Lifecycle » On Become Visible

Description

Executed when the game object it is attached to becomes visible to any camera

Keywords

Show Render Appear

On Disable

Lifecycle » On Disable

Description

Executed when the game object it is attached to becomes disabled or inactive

Keywords

Inactive Active Enable

On Enable

Lifecycle » On Enable

Description

Executed when the game object it is attached to becomes enabled and active

Keywords

Active Disable Inactive

On Fixed Update

Lifecycle » On Fixed Update

Description

Executed every fixed frame as long as the game object is enabled (physics loop

Keywords

Loop Tick Continuous Physics Rigidbody

On Interval

Lifecycle » On Interval

Description

Executes after an amount of seconds have passed between each call

Parameters

Name	Description
Time Mode	The time scale in which the interval is calculated
Interval	Amount of seconds between each iteration

Keywords

Loop Tick Continuous FPS

On Invoke

Lifecycle » On Invoke

${\tt Description}$

Executed only when calling its Invoke() method

Keywords

Script Manual

On Late Update

Lifecycle » On Late Update

Description

Executed every frame after all On Update events are fired, as long as the game object is enabled

Keywords

Loop Tick Continuous

On Start

Lifecycle » On Start

Description

Executed on the frame when the game object is enabled for the first time $% \left(\frac{\partial f}{\partial x}\right) =\frac{\partial f}{\partial x}$

Keywords

Initialize

On Update

Lifecycle » On Update

${\tt Description}$

Executed every frame as long as the game object is enabled $% \left(1\right) =\left(1\right) \left(1\right) \left$

Keywords

Loop Tick Continuous

LOGIC

Logic

Events

- On Hotspot Activate
- On Hotspot Deactivate
- On Receive Signal

On Hotspot Activate

Logic » On Hotspot Activate

Description

Executed when its associated Hotspot is activated

Keywords

Spot

On Hotspot Deactivate

Logic » On Hotspot Deactivate

Description

Executed when its associated Hotspot is deactivated

Keywords

Spot

On Receive Signal

Logic » On Receive Signal

Description

Executed when receiving a specific signal name from the dispatcher

Keywords

Event Command Fire Trigger Dispatch Execute

PHYSICS

Physics

Events

- On Collide Exit
- On Collide
- On Trigger Enter Tag
- On Trigger Enter
- On Trigger Exit Tag
- On Trigger Exit
- On Trigger Stay

On Collide Exit

Physics » On Collide Exit

Description

Executed when the Trigger that collided with a game object, stops colliding

Keywords

Crash Touch Bump Collision Stop

On Collide

Physics » On Collide

Description

Executed when the Trigger collides with a game object

Keywords

Crash Touch Bump Collision

On Trigger Enter Tag

Physics » On Trigger Enter Tag

Description

Executed when a game object with a Tag enters the Trigger collider

Parameters

Name	Description
Tag	A string that represents a group of game objects

Keywords

On Trigger Enter

Physics » On Trigger Enter

Description

Executed when a game object enters the Trigger collider

Keywords

On Trigger Exit Tag

Physics » On Trigger Exit Tag

Description

Executed when a game object with a Tag exists the Trigger collider

Parameters

Name	Description
Tag	A string that represents a group of game objects

Keywords

On Trigger Exit

Physics » On Trigger Exit

Description

Executed when a game object leaves the Trigger collider

Keywords

On Trigger Stay

Physics » On Trigger Stay

Description

Executed while a game object stays inside the Trigger collider

Keywords

STORAGE

Storage

Events

- · On Delete
- On Load
- On Save

On Delete

Storage » On Delete

Description

Executed when a previously saved game deleted

Keywords

Load Save Delete Profile Slot Game Session

On Load

Storage » On Load

Description

Executed when a previously saved game is loaded

Keywords

Load Save Profile Slot Game Session

On Save

Storage » On Save

Description

Executed when the game is saved

Keywords

Load Save Profile Slot Game Session

UI

Ui

Events

- On Deselect
- On Hover Enter
- On Hover Exit
- On Select

On Deselect

UI » On Deselect

Description

Executed when the UI element is deselected

Keywords

Mouse Choose Focus Pick Pointer

On Hover Enter

UI » On Hover Enter

${\tt Description}$

Executed when the pointer hovers the UI element

Keywords

Mouse Over Pointer

On Hover Exit

UI » On Hover Exit

${\tt Description}$

Executed when the pointer exits the hovered UI element

Keywords

Mouse Over Pointer

On Select

UI » On Select

Description

Executed when the UI element is selected

Keywords

Mouse Choose Focus Pick Pointer

VARIABLES

Variables

Events

- On Global List Variable Change
- On Global Name Variable Change
- On Local List Variable Change
- On Local Name Variable Change

On Global List Variable Change

Variables » On Global List Variable Change

Description

Executed when the Global List Variable is modified

On Global Name Variable Change

Variables » On Global Name Variable Change

Description

Executed when the Global Name Variable is modified

On Local List Variable Change

Variables » On Local List Variable Change

Description

Executed when the Local List Variable is modified

On Local Name Variable Change

Variables » On Local Name Variable Change

Description

Executed when the Local Name Variable is modified

Custom Events

Game Creator allows to create custom **Events** that listen to events and react accordingly. Note that it's up to the programmer to determine the most performant way to detect an event.



This section assumes you have some programming knowledge. If you don't know how to code you might be interested in checking out the Game Creator Hub page. Programmers altrusitically create custom **Events** for others to download and use in their project.

CREATING AN EVENT

The easiest way to create an **Event** C# script is to right click on your *Project* panel and select _Create Game Creator Developer C# Event. This will create a template script with the boilerplate structure:

```
using System;
using GameCreator.Runtime.VisualScripting;

[Serializable]
public class MyEvent : Event
{
    protected override void OnStart(Trigger trigger)
    {
        base.OnStart(trigger);
        _ = trigger.Execute(this.Self);
    }
}
```

Anatomy of an Event

An **Event** is a class that inherits from the Event super class. It contains a large collection of virtual methods to inherit from, which are very similar to MonoBeheaviour methods.

<u>|</u> Example

For example, to detect when the **Trigger** component is initialized, you can override the <code>OnAwake()</code> or the <code>OnStart()</code> methods. For a full list of all available methods to override, check the <code>Event.cs</code> script file.

All methods come with a trigger parameter, which references the Trigger component that owns this Event.

Fire an Event

Once you have setup the necessary code to detect an event, it's time to tell the **Trigger** to exeecute the specified reaction. This is done using the Execute(target) method from the Trigger component:

```
trigger.Execute(this.Self);
```



Note that the $\mathsf{Execute}(\dots)$ method returns an async task so the code can wait until the reaction completes before resuming the execution. Most of the times however, you will prefer to fire and forget about the reaction. In those cases you can use the discard ($_$) modifier:

```
_ = trigger.Execute(this.Self);
```

On the other hand, if you want to wait until the instruction sequence has completed, you can await for the resolution of these:

```
await trigger.Execute(this.Self);
```

The Execute(target) method allows to pass a game object parameter, which is the *Target* game object of the instructions list. For example, if the **Event** you are programming is trying to detect the collision between 2 colliders, the target should reference the other collider game object.

Decoration & Documentation

It is highly recommended to document and decorate the **Event** so it's easier to find and use. It is done using class-type attributes that inform **Game Creator** of the quirks of this particular event.

For example, to set the title of an Event to "Hello World", use the [Title(string name)] attribute right above the class definition:

```
using System;
using GameCreator.Runtime.VisualScripting;

[Title("Hello World")]
[Serializable]
public class MyEvent : Event
{
    protected override void OnStart(Trigger trigger)
    {
        base.OnStart(trigger);
        _ = trigger.Execute(this.Self);
    }
}
```

Title

The title of the Event. If this attribute is not provided, the title will be a beautified version of the class name.

```
[Title("Title of Event")]
```

Description

A description of what the Event does. This is used as the description text when uploading an Event to the Game Creator Hub.

```
[Description("Lorem Ipsum dolor etiam porta sem magna mollis")]
```

Image

The [Image(...)] attribute changes the default icon of the Event for one of the default ones. It consists of 2 parameters:

- Icon [Type]: a Type class of an IIcon derived class. Game Creator comes packed with a lot of icons although you can also create your own.
- · Color [Color]: The color of the icon. Uses Unity's Color class.

For example, one of the icons included is the "Solid Cube" icon. To display a red solid cube as the icon of the event, use the following attribute:

```
[Image(typeof(IconCubeSolid), Color.red)]
```

Category

A sequence of sub-categories organized using the slash (/) character. This attribute helps keep the Events organized when the dropdown list is displayed.

```
[Category("Category/Sub Category/Name")]
```

The example above will display the Event under the sub directory Category Sub Category Name

Version

A semmantic version to keep track of the development of this Event. It's important to note that when updating an Event to the Game Creator Hub, the version number must always be higher than the one on the server.

The semmantic version follows the standard *Major Version*, *Minor Version*, *Patch Version*. To know more about how semmantic versioning works, read the following page: https://semver.org.

```
[Version(1, 5, 3)]
```

Parameters

When an Event has exposed fields in the Inspector, it's a good idea to document what these do. You can add as many [Parameter(name, description)] attributes as exposed fields has the Event.

For example, if the Event has these two fields:

```
public bool checkDistance = true;
public float distance = 5f;
```

You can document those fields adding:

```
[Parameter("Check Distance", "Whether to check the distance or not")]
[Parameter("Distance", "The maximum distance between targets")]
```

Keywords

Keywords are strings that help the fuzzy finder more easily search for an Event. For example, the "On Become Visible" event doesn't reference the word "hide" anywhere in its documentation. However, these words are very likely to reference this event when the user types them in the search box.

```
[Keywords("Hide")]
```

Example

The Example attribute allows to display a text as an example of use of this Event. There can be more than one [Example(...)] attribute per event. This is particularly useful when uploading events on the Game Creator Hub.

Markdown

It is recommended to use Markdown notation when writing examples

[Example("Sed posuere consectetur est at lobortis)]

Maltiple Lines

You can use the @ character in front of a string to break the example text in multiple lines. To create a new paragraph, simply add two new lines. For example:

```
[Example(@"
   This is the first paragraph.
   This is also in the first paragraph, right after the previous sentence
   This line is part of a new paragraph.
)]
```

Dependency

This attribute is optional and only used in the Game Creator Hub. If this Event uses some particular feature of a specific module, it will first check if the user downloading this event has that module installed. If it does not, it will display an error message and forbid downloading it. This is useful to avoid throwing programming errors.

The [Dependency(...)] attribute consists of 4 parameters:

- Module ID: For example, the ID of the Inventory module is gamecreator.inventory.
- $\boldsymbol{\cdot}$ $\boldsymbol{\mathsf{Major}}$ $\boldsymbol{\mathsf{Version}}\boldsymbol{\cdot}$ The minimum major version of the dependency module.
- $\boldsymbol{\cdot}$ $\boldsymbol{\mathsf{Minor}}$ $\boldsymbol{\mathsf{Version}}\boldsymbol{\cdot}$ The minimum minor version of the dependency module.
- ullet Patch Version: The minimum patch version of the dependency module.

[Dependency("gamecreator.inventory", 1, 5, 2)]

1.5.4 Conditions

Conditions

Conditions are components attached to game objects that, when executed, start checking the conditions in each **Branch**, from top to bottom. If all the **Conditions** of a branch return success, then the **Instructions** associated to that branch are executed, and stops checking any further.

If any of the Conditions of a Branch returns false, it skips to the next branch.

Conditions



In the image above, the **Conditions** component has just one **Branch**. This branch checks whether the player is moving or not. If it happens to move moving while this Conditions component is executed, it will print the "Player is moving" message on the console.

CREATING CONDITIONS

Right click on the *Hierarchy* panel and select *Game Creator Visual Scripting Conditions*. A game object named 'Conditions' will appear in the scene with a component of the same name.

Alternatively you can also add the **Conditions** component to any game object clicking on the Inspector's Add Component button and searching for Conditions.

beleting Conditions

To delete a Conditions component, simply click on the component's little cog button and select "Remove Component" from the dropdown menu.

ADDING BRANCHES

To add a new **Branch** simply click on the *Add Branch* button. This will create a new branch at the bottom of the **Conditions** component. You can then click and drag the symbol on the right and reorder the branch list.

Banch Order

Remember that top branches have higher priority than lower ones when executed.

All **Branches** have a *Description* field, which can be used to more easily identify what that branch does. It has no gameplay effect.

CONDITIONS AND INSTRUCTIONS

A **Branch** is composed of a list of **Conditions** and a list of **Instructions**. Adding them is as easy as clicking on the *Add Condition* and *Add Instruction* respectively and choose the desired element.

Kegate Condition

It is important to note that a specific **Condition** can be negated. For example, if the condition "Is Player Moving" returns success when the player is moving, but false when it's not, you can check for the opposite effect clicking on the small green toggle. It will now return true of the player is not moving, and true otherwise.

Toggle Condition

Empty Conditions List

An empty conditions list will always return success.

Conditions

CONDITIONS

Sub Categories

- Audio
- Cameras
- Characters
- Game Objects
- Input
- Math
- Physics
- Platforms
- Scenes
- Storage
- Text
- Transforms
- Variables
- · Visual Scripting

AUDIO

Audio

Conditions

- Is Ambient Playing
- Is Music Playing
- Is Sound Effect Playing
- Is Speech Playing
- Is Speech Target Playing
- Is Ui Playing

Is Ambient Playing

Audio » Is Ambient Playing

Description

Returns true if the given Ambient sound is playing

Parameters

Name Description

Audio Clip The audio clip to check

Keywords

Is Music Playing

Audio » Is Music Playing

Description

Returns true if the given music is playing

Parameters

Name Description

Audio Clip The audio clip to check

Keywords

Is Sound Effect Playing

Audio » Is Sound Effect Playing

Description

Returns true if the given sound effect is playing

Parameters

Name Description

Audio Clip The audio clip to check

Keywords

Is Speech Playing

Audio » Is Speech Playing

Description

Returns true if the given Speech sound is playing

Parameters

Name Description

Audio Clip The audio clip to check

Keywords

Is Speech Target Playing

Audio » Is Speech Target Playing

Description

Returns true if the given target game object is playing any audio clip

Parameters

Name		Description				
Т	arget	The	game	object	target	

Keywords

SFX Speech Audio Running

Is UI Playing

Audio » Is UI Playing

Description

Returns true if the given UI sound is playing

Parameters

Name Description

Audio Clip The audio clip to check

Keywords

CAMERAS

Cameras

Conditions

• Is Shot Active

Is Shot Active

Cameras » Is Shot Active

Description

Returns true if the Camera Shot is assigned to the Main Camera

Parameters

Name	Description
Shot	The camera shot

Keywords

Camera Enabled Assigned Running

CHARACTERS

Characters

Sub Categories

- Animation
- Busy
- Combat
- Interaction
- Navigation
- Properties
- Visuals

Animation

Animation Conditions

• Has State In Layer

Has State in Layer

Characters » Animation » Has State in Layer

Description

Returns true if the Character has a State running at the specified layer index

Parameters

Name	Description		
Layer	The layer in which the Character may have a State running		
Character	The Character instance referenced in the condition		

Keywords

Characters Animation Animate State Play Character Player

Busy

Busy Conditions

- Are Arms Available
- Are Legs Available
- Is Available
- Is Busy
- Is Humanoid
- Is Left Arm Available
- Is Left Leg Available
- Is Right Arm Available
- Is Right Leg Available

Are Arms Available

Characters » Busy » Are Arms Available

Description

Returns true if the Character's arms are available to start a new action

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Occupied Available Free Doing Hand Finger Character Player

Are Legs Available

Characters » Busy » Are Legs Available

Description

Returns true if the Character's legs are available to start a new action

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Occupied Available Free Doing Foot Feet Character Player

Is Available

Characters » Busy » Is Available

Description

Returns true if the Character is not doing any action and is free to start one

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Occupied Available Free Doing Character Player

Is Busy

Characters » Busy » Is Busy

Description

Returns true if the Character doing an action that prevents from starting another one

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Occupied Available Free Doing Character Player

Is Humanoid

Characters » Busy » Is Humanoid

Description

Returns true if the Character has a humanoid model

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Human Biped Character Player

Is Left Arm Available

Characters » Busy » Is Left Arm Available

Description

Returns true if the Character's left arm is available to start a new action

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Occupied Available Free Doing Hand Finger Character Player

Is Left Leg Available

Characters » Busy » Is Left Leg Available

Description

Returns true if the Character's left leg is available to start a new action

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Occupied Available Free Doing Foot Feet Character Player

Is Right Arm Available

Characters » Busy » Is Right Arm Available

Description

Returns true if the Character's right arm is available to start a new action

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Occupied Available Free Doing Hand Finger Character Player

Is Right Leg Available

Characters » Busy » Is Right Leg Available

Description

Returns true if the Character's right leg is available to start a new action

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Occupied Available Free Doing Foot Feet Character Player

Combat

Combat Conditions

• Is Invincible

Is Invincible

Characters » Combat » Is Invincible

Description

Returns true if the Character is Invincible

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Invincibility Combat Character Player

Interaction

Interaction Conditions

• Can Interact

Can Interact

Characters » Interaction » Can Interact

Description

Returns true if the Character has any interactive element available

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Character Button Pick Do Use Pull Press Push Talk Character Player

Navigation

Navigation Conditions

- Is Airborne
- · Is Dashing
- Is Grounded
- Is Idle
- Is Moving
- · Raycast Floor

Is Airborne

Characters » Navigation » Is Airborne

Description

Returns true if the Character not touching the ground

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Fly Fall Flail Jump Float Suspend Character Player

Is Dashing

Characters » Navigation » Is Dashing

Description

Returns true if the Character is dashing

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Leap Blink Roll Flash Character Player

Is Grounded

Characters » Navigation » Is Grounded

Description

Returns true if the Character touching the floor

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Floor Stand Land Character Player

Is Idle

Characters » Navigation » Is Idle

Description

Returns true if the Character is not moving

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Stay Quiet Still Character Player

Is Moving

Characters » Navigation » Is Moving

Description

Returns true if the Character is currently in an active moving phase

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Translate Towards Destination Target Follow Walk Run Character Player

Raycast Floor

Characters » Navigation » Raycast Floor

Description

Returns true if there is an obstacle the specified units below the character

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Floor Stand Land Ground Obstacle Character Player

Properties

Properties Conditions

- · Can Jump
- · Compare Foot Phase
- Compare Gravity
- · Compare Height
- · Compare Mass
- · Compare Radius
- · Compare Speed
- Is Controllable
- Is Dead
- Is Player
- Jump Force
- · Terminal Velocity

Compare Mass

Characters » Properties » Can Jump

Description

Returns true if the character has the Can Jump property set to true

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Active Enabled Leap Hop Character Player

Compare Foot Phase

Characters » Properties » Compare Foot Phase

Description

Returns true if the chosen foot phase is currently grounded

Parameters

Name Description

Character The Character instance referenced in the condition

Example 1

Phases are the name given to the feet system that detects when a limb is grounded

Example 2

Characters can have up to 4 phases

Example 3

By default, humanoid characters assign the 'Phase 0' value to the left foot, and 'Phase 1' to the right foot. This can be customized in the Footsteps section

Keywords

Feet Foot Grounded Character Player

Compare Gravity

Characters » Properties » Compare Gravity

Description

Returns true if the comparison between a number and the Character's gravity is satisfied

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Force Vertical Character Player

Compare Height

Characters » Properties » Compare Height

Description

Returns true if the comparison between a number and the Character's height is satisfied

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Length Long Character Player

Compare Mass

Characters » Properties » Compare Mass

Description

Returns true if the comparison between a number and the Character's mass is satisfied

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Weight Character Player

Compare Radius

Characters » Properties » Compare Radius

Description

Returns true if the comparison between a number and the Character's radius is satisfied

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Diameter Width Fat Skin Space Character Player

Compare Speed

Characters » Properties » Compare Speed

Description

Returns true if the comparison between a number and the Character's speed is satisfied

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Velocity Travel Movement Walk Run Step Character Player

Is Controllable

Characters » Properties » Is Controllable

Description

Returns true if the Player unit of the Character is controllable

Parameters

Name Description

 ${\it Character} \qquad \qquad {\it The Character instance referenced in the condition}$

Keywords

Control Character Player Character Player

Is Dead

Characters » Properties » Is Dead

Description

Returns true if the character has been killed

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Kill Kaput Character Player

Is Player

Characters » Properties » Is Player

Description

Returns true if the Character is marked as a Player

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Control Character Character Player

Compare Jump Force

Characters » Properties » Jump Force

Description

Returns true if the comparison between a number and the Character's jump force is satisfied

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Hop Leap Character Player

Compare Terminal Velocity

Characters » Properties » Terminal Velocity

Description

Returns true if the comparison between a number and the Character's terminal velocity is satisfied

Parameters

Name Description

Character The Character instance referenced in the condition

Keywords

Max Fall Vertical Down Character Player

Visuals

Visuals Conditions

• Has Prop Attached

Has Prop Attached

Characters » Visuals » Has Prop Attached

Description

Returns true if the Character has a Prop attached to the specified bone

Parameters

Name	Description
Bone	The bone that has the prop attached to
Character	The Character instance referenced in the condition

Keywords

Characters Holds Grab Draw Pull Take Object Character Player

GAME OBJECTS

Game Objects

Conditions

- Compare Game Objects
- Compare Layer
- Compare Tag
- Does Component Exist
- Does Game Object Exist
- Is Component Enabled
- Is Game Object Active

Compare Game Objects

Game Objects » Compare Game Objects

Description

Returns true if the game object is the same as another one

Parameters

Name	Description					
Game Object	The game object instance used in the comparison					
Compare To	The game object instance that is compared against					

Keywords

Same Equal Exact Instance

Compare Layer

Game Objects » Compare Layer

Description

Returns true if the game object belongs to any of the layer mask values

Parameters

Name	Description
Game Object	The game object instance used in the condition
Layer Mask	A bitmask of Layer values

Keywords

Mask Physics Belong Has

Compare Tag

Game Objects » Compare Tag

Description

Returns true if the game object is tagged with a concrete name

Parameters

Name	Description
Game Object	The game object instance used in the condition
Tag	The Tag name checked against the game object

Keywords

Belong Has Is

Does Component Exist

Game Objects » Does Component Exist

Description

Returns true if the game object has the component attached

Parameters

Name	Description			
Game Object	The game object instance used in the condition			
Component	The component type that is searched			

Keywords

Null Scene Lives

Does Game Object Exist

Game Objects » Does Game Object Exist

Description

Returns true if the game object reference is not $\ensuremath{\mathsf{null}}$

Parameters

Name		Desc	cripti	ion						
Game Obj	ect	The	game	object	instance	used	in	the	conditio	n

Keywords

Null Scene Lives

Is Component Enabled

Game Objects » Is Component Enabled

Description

Returns true if the game object has the component enabled

Parameters

Name	Description
Game Object	The game object instance used in the condition
Component	The component type checked

Keywords

Null Active

Is Game Object Active

Game Objects » Is Game Object Active

Description

Returns true if the game object reference exists and is active

Parameters

Name Description

Game Object The game object instance used in the condition

Keywords

Null Scene Enabled

INPUT

Input

Conditions

- Is Input Held Down
- Is Input Pressed
- Is Input Released
- Is Key Held Down
- Is Key Pressed
- Is Key Released
- Is Mouse Held Down
- Is Mouse Pressed
- Is Mouse Released

Is Input Held Down

Input » Is Input Held Down

Description

Returns true while the Input Action asset with a button behavior is being pressed

Parameters

Name	Description
Input	A reference to the Input Action asset with map and action name

Keywords

Unity Button While Hold Press Input Action System Map

Is Input Pressed

Input » Is Input Pressed

Description

Returns true if the Input Action asset with a button behavior is pressed during this frame

Parameters

Name	Description
Input	A reference to the Input Action asset with map and action name

Keywords

Unity Button Down Input Action System Map

Is Input Released

Input » Is Input Released

Description

Returns true if the Input Action asset with a button behavior is released during this frame

Parameters

Name	Description
Input	A reference to the Input Action asset with map and action name

Keywords

Unity Button Up Input Action System Map

Is Key Held Down

Input » Is Key Held Down

Description

Returns true if the keyboard key is being held down this frame

Parameters

Name	Description
Key	The Keyboard key that is checked

Keywords

Button Active Down Press

Is Key Pressed

Input » Is Key Pressed

Description

Returns true if the keyboard key is pressed during this frame $% \left(1\right) =\left(1\right) \left(1\right$

Parameters

Name	Description
Key	The Keyboard key that is checked

Keywords

Button Down

Is Key Released

Input » Is Key Released

Description

Returns true if the keyboard key is released during this frame $% \left(1\right) =\left(1\right) \left(1$

Parameters

Name	Description
Key	The Keyboard key that is checked

Keywords

Button Up

Is Mouse Held Down

Input » Is Mouse Held Down

Description

Returns true if the mouse button is being held down

Parameters

Name Description

Button The Mouse button that is checked

Keywords

Key Up Click Cursor

Is Mouse Pressed

Input » Is Mouse Pressed

Description

Returns true if the mouse button is pressed during this frame

Parameters

Name	Description	
Button	The Mouse button that is checked	

Keywords

Key Down Cursor

Is Mouse Released

Input » Is Mouse Released

Description

Returns true if the mouse button is released during this frame

Parameters

Name Description

Button The Mouse button that is checked

Keywords

Key Up Click Cursor

MATH

Math

Sub Categories

- Arithmetic
- Boolean
- Geometry

Arithmetic

Arithmetic Conditions

- Compare Decimal
- Compare Integer

Compare Decimal

Math » Arithmetic » Compare Decimal

Description

Returns true if a comparison between two decimal values is satisfied

Parameters

Name	Description
Value	The decimal value that is being compared
Comparison	The comparison operation performed between both values
Compare To	The decimal value that is compared against

Keywords

Number Float Comma Equals Different Bigger Greater Larger Smaller

Compare Integer

Math » Arithmetic » Compare Integer

Description

Returns true if a comparison between two integer values is satisfied

Parameters

Name	Description
Value	The integer value that is being compared
Comparison	The comparison operation performed between both values
Compare To	The integer value that is compared against

Keywords

Number Whole Equals Different Bigger Greater Larger Smaller

Boolean

Boolean Conditions

- Always False
- Always True
- Compare Boolean

Always False

Math » Boolean » Always False

Description

Always returns false

Keywords

Boolean No Contradiction

Always True

Math » Boolean » Always True

Description

Always returns true

Keywords

Boolean Yes Tautology

Compare Bool

Math » Boolean » Compare Boolean

Description

Returns true if a comparison between two boolean values is satisfied

Parameters

Name	Description
Value	The boolean value that is being compared
Comparison	The comparison operation performed between both values
Compare To	The boolean value that is compared against

Keywords

Boolean

Geometry

Geometry Conditions

- Compare Direction
- Compare Distance Flat
- Compare Distance Vertical
- Compare Distance
- · Compare Point

Compare Direction

Math » Geometry » Compare Direction

Description

Returns true if a comparison between two direction values is satisfied

Parameters

Name	Description
Value	The direction value that is being compared
Comparison	The comparison operation performed between both values
Compare To	The direction value that is compared against

Keywords

Compare Distance Flat

Math » Geometry » Compare Distance Flat

Description

Returns true if a comparison of the flat XZ distance between two points is satisfied

Parameters

Name	Description
Point A	The first operand that represents a point in space
Point B	The second operand that represents a point in space
Comparison	The comparison operation performed between both values
Distance	The distance value compared against

Keywords

Compare Distance Vertical

Math » Geometry » Compare Distance Vertical

Description

Returns true if a comparison of the vertical distance between two points is satisfied

Parameters

Name	Description
Point A	The first operand that represents a point in space
Point B	The second operand that represents a point in space
Comparison	The comparison operation performed between both values
Distance	The distance value compared against

Keywords

Compare Distance

Math » Geometry » Compare Distance

Description

Returns true if a comparison of the distance between two points is satisfied

Parameters

Name	Description
Point A	The first operand that represents a point in space
Point B	The second operand that represents a point in space
Comparison	The comparison operation performed between both values
Distance	The distance value compared against

Keywords

Compare Point

Math » Geometry » Compare Point

Description

Returns true if a comparison between two points in space is satisfied

Parameters

Name	Description
Value	The point in space that is being compared
Comparison	The comparison operation performed between both values
Compare To	The point in space that is compared against

Keywords

PHYSICS

Physics

Conditions

- Check Box 2D
- Check Box 3D
- · Check Capsule
- Check Character 3D Fits
- · Check Circle
- · Check Sphere
- Is Kinematic
- Is Sleeping
- Raycast 2D
- Raycast 3D

Check Box 2D

Physics » Check Box 2D

Description

Returns true if casting a 2D box at a position collides with something

Parameters

Name	Description
Position	The scene position where the box's center is cast. Z axis is ignored
Size	Size of each side's extension along its local axis
Angle	Clock-wise rotation measured in degrees
Layer Mask	A bitmask that skips any objects that don't belong to the list

Example 1

Note that this Instruction uses Unity's 2D physics engine. It won't collide with any 3D objects

Keywords

Check Collide Touch Suit Square Cube 2D

Check Box 3D

Physics » Check Box 3D

Description

Returns true if casting a 3D box at a position collides with something

Parameters

Name	Description
Position	The scene position where the box's center is cast
Rotation	The rotation of the cube cast in world space
Half Extents	Half size of the cube that extents along its local axis
Layer Mask	A bitmask that skips any objects that don't belong to the list

Example 1

Note that this Instruction uses Unity's 3D physics engine. It won't collide with any 2D objects

Keywords

Check Collide Touch Suit Square Cube 3D

Check Capsule

Physics » Check Capsule

Description

Returns true if casting a capsule at a position collides with something

Parameters

Name	Description
Position	The scene position where the capsule's center is cast
Height	The height of the capsule in Unity units
Radius	The radius of the capsule in Unity units
Layer Mask	A bitmask that skips any objects that don't belong to the list

Example 1

Note that this Instruction uses Unity's 3D physics engine. It won't collide with any 2D objects

Keywords

Check Collide Touch Suit Character Fit 3D

Check Character 3D Fits

Physics » Check Character 3D Fits

Description

Returns true if the character fits with the new radius and height values

Parameters

Name	Description
Character	The character to check
Height	The height of the character in Unity units
Radius	The radius of the character in Unity units
Layer Mask	A bitmask that skips any objects that don't belong to the list

Example 1

Note that this Instruction uses Unity's 3D physics engine. It won't collide with any 2D objects

Keywords

Check Collide Capsule Touch Suit Character Fit 3D

Check Circle

Physics » Check Circle

Description

Returns true if casting a circle at a position doesn't collide with anything

Parameters

Name	Description
Position	The scene position where the circle's center is cast. Z axis is ignored
Radius	The radius of the circle in Unity units
Layer Mask	A bitmask that skips any objects that don't belong to the list

Example 1

Note that this Instruction uses Unity's 2D physics engine. It won't collide with any 3D objects

Keywords

Check Collide Touch Suit Sphere Circumference Round 2D

Check Sphere

Physics » Check Sphere

Description

Returns true if casting a sphere at a position collides with something

Parameters

Name	Description
Position	The scene position where the sphere's center is cast
Radius	The radius of the sphere in Unity units
Layer Mask	A bitmask that skips any objects that don't belong to the list

Example 1

Note that this Instruction uses Unity's 3D physics engine. It won't collide with any 2D objects

Keywords

Check Collide Touch Suit Circle Circumference Round 3D

Is Kinematic

Physics » Is Kinematic

Description

Returns true if the game object's Rigidbody or Rigidbody2D is marked as Kinematic

Parameters

Name	Description
Game Object	The game object instance with a Rigidbody or Rigidbody2D

Keywords

Affect Physics Force Rigidbody

Is Sleeping

Physics » Is Sleeping

Description

Returns true if the game object's Rigidbody or Rigidbody2D is sleeping

Parameters

Name	Description
Game Object	The game object instance with a Rigidbody or Rigidbody2D

Keywords

Affect Physics Force Rigidbody Awake

Raycast 2D

Physics » Raycast 2D

Description

Returns true if there any object between two positions in 2D space $\ensuremath{\mathsf{R}}$

Parameters

Name	Description
Source	The scene position where the raycast originates
Target	The targeted position where the raycast ends
Layer Mask	A bitmask that skips any objects that don't belong to the list

Example 1

Note that this Instruction uses Unity's 2D physics engine. It won't collide with any 3D objects

Keywords

Check Collide Linecast See 2D

Raycast 3D

Physics » Raycast 3D

Description

Returns true if there's an object between two positions

Parameters

Name	Description
Source	The scene position where the raycast originates
Target	The targeted position where the raycast ends
Layer Mask	A bitmask that skips any objects that don't belong to the list

Example 1

Note that this Instruction uses Unity's 3D physics engine. It won't collide with any 2D objects

Keywords

Check Collide Linecast See 3D

PLATFORMS

Platforms

Conditions

- · Check Platform
- Is Batch Mode
- Is Console
- Is Editor
- Is Mobile

Check Platform

Platforms » Check Platform

Description

Check if the running platform matches the selected one $% \left(1\right) =\left(1\right) \left(1\right$

Is Batch mode

Platforms » Is Batch mode

Description

Keywords

Server

Is Console

Platforms » Is Console

Description

Returns true if the running platform is a console $% \left\{ 1,2,\ldots ,n\right\}$

Keywords

PS4 PS5 Switch XBox Deck

Is Editor

Platforms » Is Editor

${\tt Description}$

Returns true if the running platform is the Unity Editor $\,$

Keywords

Unity

Is Mobile

Platforms » Is Mobile

Description

Returns true if the running platform is a smartphone or tablet

Keywords

Smartphone Tablet iOS Android

SCENES

Coone

Conditions

· Is Scene Loaded

Is Scene Loaded

Scenes » Is Scene Loaded

Description

Returns true if the scene has been loaded

Parameters

Name	Description
Scene	The Unity Scene reference used in the condition

STORAGE

Storage

Conditions

- Has Save At Slot
- Has Save

Has Save at Slot

Storage » Has Save at Slot

Description

Returns true if there is a saved game at the specified slot

Keywords

Game Load Continue Resume Can Is

Has Save

Storage » Has Save

Description

Returns true if there is at least one saved game

Keywords

Game Load Continue Resume Can Is

TEXT

Tex

Conditions

- Text Contains
- Text Equals

Text Contains

Text » Text Contains

Description

Returns true if the second text string occurs in the first one

Parameters

Name	Description
Text	The text string
Substring	The text string contained in Text

Keywords

String Char Sub

Text Equals

Text » Text Equals

${\tt Description}$

Returns true if two text Strings are equal

Parameters

Name	Description
Text 1	The first text string to compare
Text 2	The second text string to compare

Keywords

String Char

TRANSFORMS

Transforms

Conditions

- Child Count
- Is Child Of
- Is Sibling Of

Child Count

Transforms » Child Count

Description

Compares the amount of direct children of a game object

Parameters

Name	Description
Target	The children amount of this game object instance
Comparison	The comparison operation between the child count and a value
Compare To	The second value compared

Keywords

Transform Hierarchy Descendant Ancestor Parent Father Amount

Is Child Of

Transforms » Is Child Of

Description

Returns true if the game object is the parent of the other one

Parameters

Name	Description	
Child	The game object instance further down in the hierarchy of the parent	
Parent	The game object instance that is higher in the hierarchy	

Keywords

Transform Hierarchy Descendant Ancestor Parent Father Mother

Is Sibling Of

Transforms » Is Sibling Of

Description

Returns true if the game object shares the same parent as the other one

Parameters

Name	Description
Sibling A	The game object instance compared
Sibling B	Another game object instance compared

Keywords

Transform Hierarchy Ancestor Brother Sister

VARIABLES

Variables

Conditions

• List Is Empty

List is Empty

Variables » List is Empty

Description

Checks whether a List Variable is empty or not

Parameters

Name Description

List Variables The Local or Global List Variable to check

Keywords

Size Length Any Local Global Variable

VISUAL SCRIPTING

Visual Scripting

Conditions

- · Conditions As And
- Run Conditions As Or

Conditions as AND

Visual Scripting » Conditions as AND

Description

Returns true only if all the Conditions from the list are $\ensuremath{\mathsf{True}}$

Keywords

& All Sequence

Conditions as OR

Visual Scripting » Run Conditions as OR

Description

Returns true if at least one of the Conditions from the list is True

Keywords

| One Selector

Custom Conditions

Game Creator allows to very easily create custom Conditions.

Pogramming Knowledge Required

This section assumes you have some programming knowledge. If you don't know how to code you might be interested in checking out the Game Creator Hub page. Programmers altrusitically create custom Conditions for others to download and use in their project.

CREATING A CONDITION

The easiest way to create an **Condition** C# script is to right click on your *Project* panel and select *Create**Game Creator Developer C# Condition. This will create a template script with the boilerplate structure:

```
using System;
using GameCreator.Runtime.Common;
using GameCreator.Runtime.VisualScripting;

[Serializable]
public class MyCondition : Condition
{
    protected override bool Run(Args args)
    {
        return true;
    }
}
```

Anatomy of an Instruction

A **Condition** is a class that inherits from the Condition super class. The abstract Run(...) method is the entry point of a **Condition**'s execution, which is automatically called. This method must always return true if it's successful, or false otherwise.

The Run(...) method has a single parameter of type Args, which is a helper class that contains a reference to the game object that initiated the call (args.Self) and the targeted game object (args.Target), if any.

Decoration & Documentation

It is highly recommended to document and decorate the **Condition** so it's easier to find and use. It is done using class-type attributes that inform **Game Creator** of the quirks of this particular condition.

For example, to set the title of a condition to "Hello World", use the [Title(string name)] attribute right above the class definition:

```
using System;
using GameCreator.Runtime.Common;
using GameCreator.Runtime.VisualScripting;

[Title("Hello World")]
[Serializable]
public class MyCondition : Condition
{
    protected override bool Run(Args args)
    {
        return true;
    }
}
```

Title

The title of the Condition. If this attribute is not provided, the title will be a beautified version of the class name.

```
[Title("Title of Condition")]
```

Description

A description of what the Condition does. This is both used in the floating window documentation, as well as the description text when uploading a Condition to the Game Creator Hub.

```
[Description("Lorem Ipsum dolor etiam porta sem magna mollis")]
```

Image

The [Image(...)] attribute changes the default icon of the Condition for one of the default ones. It consists of 2 parameters:

- Icon [Type]: a Type class of an IIcon derived class. Game Creator comes packed with a lot of icons although you can also create your own.
- Color [Color]: The color of the icon. Uses Unity's Color class.

For example, one of the icons included is the "Solid Cube" icon. To display a red solid cube as the icon of the condition, use the following attribute:

```
[Image(typeof(IconCubeSolid), Color.red)]
```

Category

A sequence of sub-categories organized using the slash (7) character. This attribute helps keep the Conditions organized when the dropdown list is displayed.

```
[Category("Category/Sub Category/Name")]
```

The example above will display the Condition under the sub directory Category Sub Category Name.

Version

A semmantic version to keep track of the development of this Condition. It's important to note that when updating a Condition to the Game Creator Hub, the version number must always be higher than the one on the server.

The semmantic version follows the standard *Major Version*, *Minor Version*, *Patch Version*. To know more about how semmantic versioning works, read the following page: https://semver.org.

```
[Version(1, 5, 3)]
```

Parameters

When a Condition has exposed fields in the Inspector, it's a good idea to document what these do. You can add as many [Parameter(name, description)] attributes as exposed fields has.

For example, if the Condition has these two fields:

```
public bool condition1 = true;
public bool condition2 = false;
```

You can document those fields adding:

```
[Parameter("Condition 1", "First condition value to check")]
[Parameter("Condition 2", "Second condition value to check")]
```

Keywords

Keywords are strings that help the fuzzy finder more easily search for a condition. For example, the "Is Character Moving" condition doesn't reference the word "idle" or "walk" anywhere in its documentation. However, these words are very likely to reference this condition when the user types them in the search box.

```
[Keywords("Idle", "Walk", "Run")]
```

Example

The Example attribute allows to display a text as an example of use of this Condition. There can be more than one [Example(...)] attribute per condition. This is particularly useful when uploading conditions on the Game Creator Hub.

Markdown

It is recommended to use Markdown notation when writing examples

[Example("Sed posuere consectetur est at lobortis)]

€1tiple Lines

You can use the @ character in front of a string to break the example text in multiple lines. To create a new paragraph, simply add two new lines. For example:

```
[Example(@"
   This is the first paragraph.
   This is also in the first paragraph, right after the previous sentence
   This line is part of a new paragraph.
)]
```

Dependency

This attribute is optional and only used in the Game Creator Hub. If this Condition uses some particular feature of a specific module, it will first check if the user downloading this condition has that module installed. If it does not, it will display an error message and forbid downloading it. This is useful to avoid throwing programming errors.

The [Dependency(...)] attribute consists of 4 parameters:

- ullet Module ID: For example, the ID of the Inventory module is gamecreator.inventory.
- ${\boldsymbol{\cdot}}$ ${\boldsymbol{Major}}$ ${\boldsymbol{Version}} \colon$ The minimum major version of the dependency module.
- Minor Version: The minimum minor version of the dependency module.
- Patch Version: The minimum patch version of the dependency module.

[Dependency("gamecreator.inventory", 1, 5, 2)]

1.5.5 Hotspots

Hotspots

Hotspots are components attached to game objects that don't have any direct impact on gameplay. Instead, they help the user understand what's interactive and what is not. For example, highlighting a specific object when the player character is nearby, making the head turn towards an important object and so on.

Hotspots



Triggers are usually placed along side with **Hotspot** components. One deals with the interaction itself, while the other hints the player about the **Trigger** being an interactive object.

HOW IT WORKS

A **Hotspot** consists of a *Target* field and a *Mode*, which determine the object being followed and when it reacts. There are 4 possible modes:

- Radius: Displays an extra field with a numeric value. The Hotspot will react when the target is inside its radius.
- On Interaction Focus: The Hotspot will react whenever the Target's Interaction system focuses on the Hotspot.
- On Interaction Reach: The Hotspot will react whenever the Target's Interaction system has the Hotspot is within reach but isn't focused on it.
- · Always Active: The Hotspot will always react regardless of the distance to the Target.

Interaction Focus & Reach

These two modes require the Target to be a Character component. To know more about how the Interaction system works, see the Interaction section.

Hotspot Gizmo

Selecting a game object with a **Hotspot** component with a *Radius* mode will display in the scene a visual representation of the distance at which the target is considered close enough to activate it.

Debugging

On playmode, the red gizmo appears in a much lighter color. If the targeted object activates the Hotspot, the Hotspot's gizmo will change to green, to indicate the Hotspot is active.

No Phyics Engine

The **Hotspot** distance check doesn't use Unity's Phyics engine because it would force both the Hotspot and the targeted object to have a *Collider* component attached to them. Instead it simply checks the distance between the center of the hotspot and the targeted game object.

CREATING HOTSPOTS

There are two ways to create a Hotspot object. One is to create an object that contains a Hotspot component, by right clicking on the *Hierarchy* panel and selecting *Game Creator Visual Scripting Hotspot*. This creates a scene object with the component attached to it.

However, an Actions component can also be added to any game object. Simply click on any game object's Add Component button and type Actions.

Veleting Actions

To delete an Actions component, simply click on the component's little cog button and select "Remove Component" from the dropdown menu.

ADDING SPOTS

Spots are individual elements that highlight something specific and are evaluated from top to bottom.

Add new Spot

To add a new **Spot** click on the *Add Spot* button and choose the desired one from the dropdown list. Note that **Spots** are evaluated from top to bottom. There can be two spots of the same type, but if they both overlap, the last one will override the effect.

Spots

SPOTS

Sub Categories

- Audio
- Characters
- · Game Objects
- Materials
- Ui

AUDIO

Audio

Spots

• Play Sound

Play Sound

Audio » Play Sound

Description

Plays a User Interface sound effect when the Hotspot is activated or deactivated

Keywords

Audio Sounds

CHARACTERS

Characters

Spots

· Look At

Look At

Characters » Look At

Description

Makes the Character look at the center of the Hotspot when it's activated nd smoothly look away when it's deactivated

GAME OBJECTS

Game Objects

Spots

- Activate Object
- Instantiate Prefab

Activate Object

Game Objects » Activate Object

Description

Activates a game object scene instance when the Hotspot is enabled and deactivates it when the Hotspot is disabled

Instantiate Prefab

Game Objects » Instantiate Prefab

Description

Creates or Activates a prefab game object when the Hotspot is enabled and deactivates it when the Hotspot is disabled

MATERIALS

Materials

Spots

· Change Material

Change Material

Materials » Change Material

Description

Changes the Material depending on whether the Hotspot is active or not

Keywords

Material Color Shader

UI

Ui

Spots

- Change Text
- Cursor
- Show Floating Text

Change Text

UI » Change Text

Description

Changes the chosen Text value

Cursor

UI » Cursor

${\tt Description}$

Changes the cursor image when hovering the Hotspot

Show Floating Text

UI » Show Floating Text

Description

Displays a text in a world-space canvas when the Hotspot is enabled and hides it when is disabled. If no Prefab is provided, a default UI is displayed

1.6 Variables

1.6.1 Variables

Variables are data containers that allow to dynamically change their value and let the game keep track of the player's progress.



A very simple use case of **Variables** is keeping track of the player's score. Let's say we have a named variable called *score* and has an initial value of 0. Every time the player picks up a star, the *score* variable is incremented and its value is displayed.

Types of Variables

Game Creator has two types of variables:

NAME VARIABLES

Are identified by their unique name. For example, the name *score* can reference a numeric variable that keeps track of the player's score value.

Name Variables

LIST VARIABLES

Are identified by their θ -based index. Think of them as a collection of values, placed one after another. For example, to access the first value, use the index θ . To access the second position, use the index 1, etc...

Note all values of a List Variable are of a particular type.

List Variables



As a rule of thumb, it is recommended the use of **Name Variables**. **List Variables** are useful when you have an unknown number of objects to choose from. For example, when locking on an enemy from a group that surrounds the player.

Scope of Variables

Variables can either be local or global.

LOCAL VARIABLES

Local Variables are bound to a particular scene and can't be used outside of it.

GLOBAL VARIABLES

On the other hand, Global Variables can be queried and modified from any scene.



Both Global Variables and Local Variables can be List or Name based.

Value Types

All **Variables** have an initial value assigned to them that can be modifed at runtime. By default, **Game Creator** comes with a limited number of types to choose from, but other modules might increment the amount available.

- Number: Stores numeric values. Both decimal and integers.
- · String: Stores text-based characters.
- Boolean: Can only store two values: true or false.
- Vector 3: Stores an (x,y,z) vector value
- Color: Stores an RGBA color value. Can also contain HDR information.
- · Texture: Stores a reference to a Texture asset.
- Sprite: Stores a reference to a Sprite asset.
- · Game Object: Stores a reference to a game object.



It is important to note that not all data types can be saved between play-sessions. **Textures**, **Sprites** and **Game Objects** and not primitive types and thus, they can't be serialized at runtime.

Nested Access

Nested Access is a concept that allows jumping between different variables using one single command.

For example, let's say the Player object has a **Local Named Variable** called target of type Game Object. This game object is dynamic but let's say the targeted object will always have another **Local Named Variable** called health that contains how many hit points the enemy has.

The health variable can be accessed using the key target/health (with a slash). This means: Get the variable value health that the variable target points to.

1.6.2 Global Name Variables

Global Name Variables are variables identified by a unique string of characters that live outside the scene and can be accessed and modified from anywhere.

Creating a Global Name Variable

To create a **Global Name Variable**, right click on the *Project Panel* and select *Create Game Creator Variables*Name Variables. A new asset will appear in the project panel, which can be used to define each of the variables contained within.

Global Name Variables



Note that two Global Variables can't have the same unique ID. Otherwise they'll override each other's values. To generate a new unique ID, expand the *ID* field and click the "Regenerate" button.

Adding new entries

To add a new variable entry, type the name of the variable on the creation field and press enter (or click on the little [+] button).

The name of a variable can be modified, as well as its value type. The *Value* field also contains the starting value of this particular variable entry.

Save & Load

Values can be saved between play sessions to later be restored when loading a game. Disabling the save option will make all variables keep the initial value as their starting value, even after loading a previously saved game.

1.6.3 Global List Variables

Global List Variables are variables identified by their numberic index value and can be accessed from anywhere.

Creating a Global List Variable

To create a **Global List Variable**, right click on the *Project Panel* and select *Create Game Creator Variables*List Variables. A new asset will appear in the project panel, which can be used to define the collection of variables.

Global List Variables



Note that two Global Variables can't have the same unique ID. Otherwise they'll override each other's values. To generate a new unique ID, expand the *ID* field and click the "Regenerate" button.

Save & Load

Vales can be saved between play sessions to later be restored when loading a game. Disabling the save option will make all variables keep the initial value as their starting value, even after loading a previously saved game.

1.6.4 Local Name Variables

Local Name Variables are variables identified by a unique string of characters that live inside a scene and can only reference objects that are contained inside this scene.

Creating a Local Name Variable

To create a **Local Name Variable**, right click on the Hierarchy Panel_ and select *Game Creator Variables*Name Variables. A new game object will appear with the **Local Name Variables** component. Alternatively you can also add this component to any existing game object.

Global Name Variables



Note that two Local Variables can't have the same unique ID. Otherwise they'll override each other's values. To generate a new unique ID, expand the *ID* field and click the "Regenerate" button.

Adding new entries

To add a new variable entry, type the name of the variable on the creation field and press enter (or click on the little [+] button).

The name of a variable can be modified, as well as its value type. The *Value* field als contains the starting value of this particular variable entry.

Save & Load

Vales can be saved between play sessions to later be restored when loading a game. Disabling the save option will make all variables keep the initial value as their starting value, even after loading a previously saved game.

1.6.5 Local List Variables

Local List Variables are variables identified by their numberic index value and can only be accessed from the scene they are part of.

Creating a Local List Variable

To create a **Local List Variable**, right click on the Hierarchy Panel_ and select *Create Game Creator Variables List Variables*. A new game object with the component will appear in the scene and hierarchy. Alternatively, you can also add the *Local List Variables* component to any existing game object.

Local List Variables



Note that two Local Variables can't have the same unique ID. Otherwise they'll override each other's values. To generate a new unique ID, expand the ID field and click the "Regenerate" button.

Save & Load

Vales can be saved between play sessions to later be restored when loading a game. Disabling the save option will make all variables keep the initial value as their starting value, even after loading a previously saved game.

1.7 Advanced

1.7.1 Advanced

Game Creator includes a collection of tools used throughout the entire ecosystem. This section briefly goes over all of them and provides a link to each tool's page, where they are explained in-depth, with use cases and examples.



This section of the Documentation assumes you are familiar with Unity and Game Creator. Some sections may require you to also have some coding knowledge.

Audio

Game Creator has a 4 channel audio system that makes it very easy to change volume settings and play both diegetic and non-diegetic sound effects.

Learn about Audio

Signals

Communication between game objects is handled using the visual scripting tools, such as **Triggers** and **Actions**. However, there may be cases where the developer needs to respond to more tailored events that don't exist in Game Creator.

Signals

The **Raise Signal** instruction broadcasts a message with a specific identifier and any **Trigger**(s) listening to that specific id will be executed. To receive a signal message, use the **On Receive Signal** and specify the identifier.



To avoid misspelling mistakes you can mark a **Signal** name as *favorite*, which can be used selecting them from the dropdown button on the right side. To unfavorite a name, simply click again on the *star* button.

Data Structures

Advanced Data Structures (also known as ADS) are generic data structures that help better perform certain tasks.

- Unique ID: Uniquely identifies an object with a serializable Guid.
- Singleton: It ensures there's zero or one instance of a class at any given moment and its value is globally accessible.
- Dictionary: A serializable dictionary.
- Hash Set: A serializable Hash Set.
- · Link List: A serializable Linked List.
- Matrix 2D: A serializable 2D matrix.
- Tree: Generic structure that allows to have acyclic parent-child dependencies between multiple class instances.
- Ring Buffer: This structure is similar to a generic list, but sequentially accessing its elements yields in an infinite circular loop, where the last element connects with the first one.
- State Machine: A data structure that allows to dynamically manipulate a state machine and define logic on each of its nodes independently.
- Spatial Hash: An advanced data structure that allows to detect collisions of any radial size inside an infinite spatial domain with an 0 complexity of log(n).

Variables API

Local Variables and **Global Variables** can be modified at runtime using the exposed API. Note that **Local** variables are accessed via their component and **Global** variables require to be accessed through a singleton manager that contain their runtime values.

Learn how to use the Variables API

Properties

Properties are a core feature that allows to dynamically access a value. They are usually displayed as a drop-down menu and allow to retrieve them depending on the option selected.

For example, a PropertyGetPosition allows to get a Vector3 that represents a position, from different sources; A constant value, the Player's position, the main camera's position, from a Local Variable, etc...

Learn more about $\ensuremath{\textbf{Properties}}$

Saving & Loading

Game Creator comes with a fully extensible save and load system that allows to easily keep track of the game progress and restore its state at any time. All that needs to be done is to implement an interface called IGameSave and subscribe/unsubscribe inside the OnEnable() and OnDisable() methods respectively.

- · Saving and Loading
- · Saving custom data
- · Saving on custom databases

There is a special component called **Remember** that allows to cherry-pick the bits of data you want to save when saving a game.

Tweening

Game Creator comes packged with a powerful **Tweening** (or automatic frame interpolation, from in-between-ing) system. It allows to *fire & forget* a command that creates a tween between a starting value and end value. The transition can be linear or an easing function can be specified.

Learn more about **Tweening**

Examples and Templates

Game Creator and all modules come with a collection of examples and templates ready to be used on your games and applications. Other developers can leverage this feature in order to create reusable examples that can be installed/uninstalled across multiple projects or share them if you are a module developer using the **Example Manager** window.

Learn more about Creating custom Examples

Domain Reload

Game Creator supports skipping domain reloading, which reduces the time it takes for Unity to enter and exit play-mode.

Make sure Enter Play Mode Options is ticked and the Reload Domain option is disabled.

Skip Domain Reload

1.7.2 Audio

Game Creator comes with an audio manager that automatically manages and optimizes the creation and decomission of audio sources. There are 4 different types of audio channels, each with its own volume slider and properties.

Ambient

Ambient sounds are what one could also call background music or ambience. It's a looped tracked played in the background, and can be diegetic or non-diegetic. For example, a battle music track, the chirping of birds in a forest, or the sound of a waterfall.



Use the **Play Ambient** Instruction to play an audio clip as an Ambient sound. It will keep playing until a **Stop Ambient** Instruction is executed.

Sound Effects

Sound Effects (also known as SFX) are one-time clips played at a very specific time. The majority of sounds on a game will be sound effects, for example: Punching a character, footstep sounds, or a slash of a sword. Most sound effects are diegetic and thus, by default expect a spatial position.



To avoid the jarring effect where the same sound effect is played over and over again in a small time window, sound effects can automatically randomly alter the *speed* and *pitch* of sounds. This allows to, for example, play a machine gun sound effect, where each shot is slightly different than the previous one.



Use the **Play Sound Effect** Instruction to play an audio clip as a Sound Effect. It will automatically decommision the audio source once the clip finishes playing.

UI

UI sound effects are non-diegetic clips played when the player interacts with the user interface. For example, hovering over a button, clicking it or crafting an item after the user waits a timeout.



Use the **Play UI** Instruction to play an audio clip as a UI sound effect.

Speech

Speech clips are very similar to **Sound Effects** with the difference that they are bound to a Character, so that a specific character can only play one speech clip at a time.

Pay Sound Effect Instruction

Use the **Play Speech** Instruction to play an audio clip as a Speech sound effect. If another clip is was being played on the same target, it will stop the previous speech and play the new one. This is useful when the user skips conversations.

1.7.3 Data Structures

Index

DATA STRUCTURES

Advanced Data Structures (also known as ADS) are generic data structures that help better perform certain tasks.

- Unique ID: Uniquely identifies an object with a serializable Guid.
- Singleton: It ensures there's zero or one instance of a class at any given moment and its value is globally accessible.
- Dictionary: A serializable dictionary.
- Hash Set: A serializable Hash Set.
- Link List: A serializable Linked List.
- Matrix 2D: A serializable 2D matrix.
- Tree: Generic structure that allows to have acyclic parent-child dependencies between multiple class instances
- Ring Buffer: This structure is similar to a generic list, but sequentially accessing its elements yields in an infinite circular loop, where the last element connects with the first one.
- State Machine: A data structure that allows to dynamically manipulate a state machine and define logic on each of its nodes independently.
- Spatial Hash: An advanced data structure that allows to detect collisions of any radial size inside an infinite spatial domain with an O complexity of log(n).

Unique ID

To generate unique identifiers, it is usually used the System.Guid class, because it provides a fast and reliable mechanism to generate long enough IDs that the collision chance is almost zero.

However, this class is not serializable. That's why **Game Creator** comes with the UniqueID class, which serves two purposes:

- · Serializable: This means that any changes made to this ID will be kept between editor sessions.
- Custom UI: When showing this ID in a Unity Window, it automatically displays a nice and handy box with buttons that allow to easily modify this ID or even regenerate it, in case that's necessary.

INITIALIZATION

To initialize a class instance of UniqueID is as easy as calling the constructor class. For example, let's say we want to add a unique ID to a MonoBehaviour class:

```
public class MyComponent : MonoBehaviour
{
    public UniqueID myID = new UniqueID();
}
```

This will automagically assign a unique ID to the myID field. If we drag and drop this component onto a scene game object, we'll see this field with its associated ID.

ACCESSING ID

Accessing the ID value can be performed getting the IdString struct, which contains a string based ID and its hash value. This last one is recommended when comparing to ids:

To get the hash value:

```
int hash = this.myID.Get.Hash;
```

To get the string value:

string id = this.myID.Get.String;



Accessing the string value of the UniqueID should only be done if you plan on serializing this value somewhere. For comparing two IDs, it is best if you simply compare their hash value, as the probablity that two strings have the same hash value its very, very very low. On the other hand, comparing two int values is extremely fast and performant.

Singleton

The **Singleton** pattern ensures there's, at most, one instance of a class at any given time. Because of that, it can be globally accessed from its class name. To make a singleton class, inherit from the Singleton<T> type:

```
public MyClass : Singleton<MyClass>
{ }
```

To access this class, use MyClass.Instance which returns an instance of the MyClass. If none was present, it creates one and then it returns it, so you don't have to worry about keeping track whether it has been created or not

AnoBehaviour

This Singleton pattern is specifically designed to work with Unity and thus, it requires the MyClass to inherit from MonoBehaviour. However, this is defined automatically when inheriting from the Singleton<T> class.

If you need to perform some setup when creating a new class instance, override the the OnCreate() method. Likewise, you can also override the OnDestroy() method to execute some logic when the instance is destroyed.

```
public MyClass : Singleton<MyClass>
{
    protected override void OnCreate() {
        base.OnCreate();
        // This is executed only once when created }

    protected override void OnDestroy() {
        base.OnDestroy();
        // This is executed only once when destroyed }
}
```

Singleton instances can survive or be destroyed every time their scene is unloaded. By default all singleton classes survivde scene reloading. But if you want to destroy them when changing between scenes, override the SurviveSceneLoads and set it to false:

```
public MyClass : Singleton<MyClass>
{
    protected override bool SurviveSceneLoads => false;
}
```

Dictionary

The serializable dictionary allows to have the whole fully fledged functionality of System.Collections.Dictionary but also allows to automatically serialize its values.

To create a serializable dictionary, simply inherit from TSerializableDictionary<TKey, TValue>. For example, to create a dictionary that uses string as their key and GameObject as their value:

```
public MyDictionary : TSerializableDictionary<string, GameObject>
{ }
```

You can now create a dictionary that automatically serializes its values and use it as any normal dictionary:

```
public MyComponent : MonoBehaviour
{
   public MyDictionary dictionary = new MyDictionary();

   private void Awake()
   {
        // Add element to dictionary:
        this.dictionary.Add("Hello World", this.gameObject);

        // Print element added
        Debug.Log(this.dictionary["Hello World"].name);
   }
}
```

Hash Set

The serializable hash set allows to have the functionality of System.Collections.HashSet but also allows to automatically serialize its values.

To create a serializable hash set, simply inherit from TSerializableHashSet<T>. For example, to create a hash set that uses string types:

```
public MyHashSet : TSerializableHash<string>
{ }
```

You can now create a hash set that automatically serializes its values and use it as:

```
public MyComponent : MonoBehaviour
{
   public MyHashSet hashSet = new MyHashSet();

   private void Awake()
   {
        // Add element:
        this.hashSet.Add("Hello World");

        // Print if it can find the elements
        Debug.Log(this.hashSet.Contains("Hello World"));
        Debug.Log(this.hashSet.Contains("Foo"));
   }
}
```

Link List

The serializable linked list allows to have the functionality of System.Collections.LinkedList but also allows to automatically serialize its values.

To create a serializable linked list, simply inherit from TSerializableLinkList<T>. For example, to create a hash set that uses GameObject types:

```
public MyLinkedList : TSerializableLinkList<GameObject>
{ }
```

You can now create a list that automatically serializes its values and use it as:

```
public MyComponent : MonoBehaviour
{
   public MyLinkedList list = new MyLinkedList();

   public GameObject objectA;
   public GameObject objectB;
   public GameObject objectC;

   private void Awake()
   {
        // Add element:
        this.list.Add(this.objectA);
        this.list.AddFirst(this.objectB);
        this.list.AddFirst(this.objectC);

        // Print the first element:
        Debug.Log(this.list.First().name);
    }
}
```

Matrix 2D

The serializable 2D matrix allows to have an array of arrays (where all rows and columns have the same size) and the structure can be serialized in order to persist in the Inspector or saving the game.

To create a serializable matrix, simply inherit from TSerializableMatrix2D<T>. For example, to create a matrix that uses GameObject:

```
public MyMatrix : TSerializableMatrix2D<GameObject>
{ }
```

You can now create a matrix that automatically serializes its values:

```
public MyComponent : MonoBehaviour
{
   public MyMatrix matrix = new MyMatrix(10, 5);

   private void Awake()
   {
      // Add element:
      this.matrix[2, 3] = this.gameObject;

      // Print element added
      Debug.Log(this.matrix[2, 3].name);
   }
}
```

Tree

The Tree class allows to create acyclic dependency graphs that start from a root node and end with leaf nodes. A single node can have an unlimited number of branches.

To create a Tree, inherit from the Tree<T> class, where T is the value type of the node. For example, to create a tree of game objects:

A Tree<T> class is both the tree and the node class. So any child of a tree returns a tree object too. A tree can return its parent:

```
MyTree parent = this.tree.Parent
```

And it's children, which is a dictionary indexed by its Ids:

```
KeyValuePair<string, GameObject> = this.tree.Children;
```

Ring Buffer

The **Ring Buffer** is a very interesting data structure that works very similar to an array, except that its capacity is capped and iterating over its elements will automatically jump from its tail to its head when reaching the end of the list. Think of it as an array with a limited capacity where the tail joins the head, thus shaping it a ring.

To create a ring buffer, create a class that inherits from the Ring<T> class or directly use the Ring<T> type. For example, to create a ring buffer with 5 elements:

```
Ring<string> myRing = new Ring<string>(
    "string 1",
    "string 2",
    "string 3",
    "string 4",
    "string 5",
);
```

The ring buffer starts with its index pointing to the first element. Calling Next(), Current() and Previous() will change the pointer and return the new value. For example:

The previous code snippet will iterate the previous ring 20 times (100 / 5) and print the name of each entry.

An interesting method of the ring buffer is the Update(callback). This method accepts a method as its parameter and executes it for every element of the ring. For example:

```
myRing.Update(Debug.Log);
```

The previous method will print each of the entries of the ring buffer, as the <code>Debug.Log()</code> method is applied to each one of them.

State Machine

A **State Machine** is a commonly used pattern that allows to isolated the complexity of multiple tasks in different nodes, in a way that each node is not aware of what others do.

About State Machines

For a full description of what a finite state machine is check this Wikipedia article.

CREATING STATES

Let's start seeing how to create states before creating a state machine. A **State** is a single node unit from the state machine. To create one, create a class that inherits from the StateMachine.State abstract class:

```
public class MyState1 : StateMachine.State
{ }
```

A State has 3 virtual methods that can be overriden in order to execute its custom logic:

```
// Executed when the machine changes to this state
void WhenEnter(StateMachine machine)
{ }

// Executed when the machine exists from this state
protected virtual void WhenExit(StateMachine machine)
{ }

// Executed every frame while this state is active
protected virtual void WhenUpdate(StateMachine machine)
{ }
```

A state has an IsActive property that can be queried to check if this state is currently the active one.

If you need to hook events to a State in order to make it work with other scripts, you can also subscribe to its event system.

```
// Executed when the machine changes to this state
event Action<StateMachine, State> EventOnEnter;

// Executed when the machine exists from this state
event Action<StateMachine, State> EventOnExit;

// Executed every frame while this state is active, before the WhenUpdate(...)
event Action<StateMachine, State> EventOnBeforeUpdate;
```

For example, let's first create an instance of MyState1:

```
MyState1 state1 = new MyState();
```

Now let's hook an external method that prints a message when the state is entered:

```
state1.EventOnEnter += this.OnEnterState;
```

The $\mbox{{\tt OnEnterState}}(\ldots)$ method must have the following signature:

```
public void OnEnterState(StateMachine machine, State state)
{
    Debug.Log("Hello World!");
}
```

CREATING A STATE MACHINE

To create a state machine, create a class that inherits from ${\tt StateMachine}$:

```
public class MyStateMachine : StateMachine
{
    public MyStateMachine(State state) : base(state)
    { }
}
```

rst State

Note that a State Machine requires at least one state to be passed to the constructor. This is the first starting state that the machine will begin with.

The developer is responsible for calling its Update() method. We recommend calling it in a MonoBehaviour's Update().

To instruct the machine to change from one state to another, use the Change(State) method:

```
MyState1 state1 = new MyState1();
MyState2 state2 = new MyState2();

// Initialize with state1
MyStateMachine machine = new MyStateMachine(state1);

// Change to state2
machine.Change(state2);
```

A State Machine also has 2 events that allow methods to be subscribed, which are launched as soon as there is a change in the currently active state:

```
event Action<State> EventStateEnter;
event Action<State> EventStateExit;
```

Spatial Hash

The **Spatial Hash** algorithm is a performant non-physics based query system that returns a list of objects contained in a position and a certain radius.



This algorithm scales with the amount of objects tracked. Its performance shines the most when there are multiple queries launched in a single frame. For more information about how this algorithm works check this Twitter post:

https://twitter.com/catsoftstudios/status/1201520331724333058

CREATING A DOMAIN

The first thing needed is to create a world domain from where to track all objects and organize the space partitioning. We recommend setting up a static class that will handle registering all the changes that happen in the scene. For example:

```
public static class MySpatialHash {
    public static SpatialHash Value { get; private set; } = new SpatialHash();
    [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.SubsystemRegistration)]
    private static void OnSubsystemsInit()
    {
            Value = new SpatialHash();
        }
}
```

The previous code snippet initializes the Value field with the default SpatialHash constructor. the OnSubsystemInit() is a method that gets called at the very beginning of starting the game, before any scene is loaded, thanks to its attribute.

TRACKING CHANGES

Each object instance is responsible for updating the domain value when it changes. To do so, the object must implement the ISpatialHash interface, as well as call the Insert(), Remove() and Update() methods to start, stop and update the spatial hash's domain. For example:

Bost Performance

This code is meant for demonstration purposes and might not be optimal on every case. If you want to squeeze every drop of performance, you may want to cache the last tracked position and only call the Update(this) method when its position has changed.

REQUESTING COLLECTIONS

To request all the objects around a point and within a specific radius, use the Query(Vector3 point, float radius) method, which returns a list of game objects contained in the specified region.

```
// Define a point and radius in the 3D space:
Vector3 point = new Vector3(0,0,0);
float radius = 10f;

// request for all tracked game object within:
List-ISpatialHash> list = MySpatialHash.Value.Query(point, radius);
```

The list contains all components that implement the <code>ISpatialHash</code> interface tracked in this domain that are within the spherical region defined.

1.7.4 Variables API

Local Variables

Local Name Variables and **Local List Variables** are components attached to game objects and their value is bound to the scene they are. To access their runtime values you reference the component and call one of their public methods.

LOCAL NAME VARIABLES

Local Name Variables are components attached to game objects and can be referenced like any other script. To access any of its values you can use the following methods:

Getting values

```
bool Exists(string name)
```

Returns true if the variable exists. False otherwise

```
object Get(string name)
```

Returns the value of the variable. Requires to be casted to the correct value

Setting values

```
void Set(string name, object value)
```

Sets the value of a variable

Listening to events

You can also register when a Local Name Variable changes using the following methods:

```
void Register(Action<string> callback)
```

Executes the callback every time a variable changes its value

```
void Unregister(Action<string> callback)
```

Stops executing the callback when the variable changes

LOCAL LIST VARIABLES

A Local List Variables component has the following methods for getting and manipulating its values:

Getting values

```
object Get(IListGetPick pick)
```

Returns the value indexed by the pick parameter

```
int Count
```

Property that returns the number of elements of the list

Setting values

```
void Set(IListSetPick pick, object value)
```

Sets a value indexed by the pick parameter

```
void Insert(IListGetPick pick, object content)
```

Inserts a value at the indexed position $% \left(1\right) =\left(1\right) \left(1\right)$

void Push(object value)

Adds a new value at the end of the list

void Remove(IListGetPick pick)

Removes the value indexed by the pick parameter

void Clear()

Removes all values from the list

void Move(IListGetPick pickA, IListGetPick pickB)

Moves the value indexed at a position to a new index

Listening to events

You can also register when a Local List Variable changes any of its items using the following methods:

void Register(Action<ListVariableRuntime.Change, int> callback)

Executes the callback method whenever there's a change

void Unregister(Action<ListVariableRuntime.Change, int> callback)

Stops executing the callback when the list changes

Global Variables

Global Name Variables and **Global List Variables** are scriptable objects and their runtime value is stored in a separate singleton manager called GlobalNameVariablesManager and GlobalListVariablesManager.

GLOBAL NAME VARIABLES

The GlobalNameVariablesManager has the following methods available:

Getting values

bool Exists(GlobalNameVariables asset, string name)

Returns true if the variable exists. False otherwise

 ${\tt object} \ {\tt Get}({\tt GlobalNameVariables} \ {\tt asset}, \ {\tt string} \ {\tt name})$

Returns the value of the variable. Requires to be casted to the correct value

Setting values

void Set(GlobalNameVariables asset, string name, object value)

Sets the value of a variable

Listening to events

You can also register when a Global Name Variable changes using the following methods:

void Register(GlobalNameVariables asset, Action<string> callback)

Executes the callback every time the variable changes its value

void Unregister(GlobalNameVariables asset, Action<string> callback)

Stops executing the callback when the variable changes

GLOBAL LIST VARIABLES

The GlobalListVariablesManager has the following methods:

Gettings values

int Count(GlobalListVariables asset)

Returns the number of elements of the list

object Get(GlobalListVariables asset, IListGetPick pick)

Setting values

Returns the value indexed by the pick parameter

void Set(GlobalListVariables asset, IListSetPick pick, object value)

Sets a value indexed by the pick parameter

void Insert(GlobalListVariables asset, IListGetPick pick, TValue content)

Inserts a value at the indexed position

void Push(GlobalListVariables asset, TValue value)

Adds a new value at the end of the list

void Remove(GlobalListVariables asset, IListGetPick pick)

Removes the value indexed by the pick parameter

void Clear(GlobalListVariables asset)

Removes all values from the list

void Move(GlobalListVariables asset, IListGetPick pickA, IListGetPick pickB)

Moves the value indexed at a position to a new index

Listening to events

You can also register when a Global List Variable changes any of its items using the following methods:

void Register(GlobalListVariables asset, Action<ListVariableRuntime.Change, int> callback)

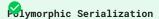
Executes the callback method whenever there's a change

void Unregister(GlobalListVariables asset, Action<ListVariableRuntime.Change, int> callback)

Stops executing the callback when the list changes

1.7.5 Properties

Game Creator properties are a special type of class that allows to dynamically specify the source of a field value using a dropdown menu. The menu's options are dynamic and can be added without the need of overwriting Game Creator core code, allowing to write maintainable and decoupled code.



Properties take advantage of Unity's polymorphic serialzation, which means that the dropdown menu options are decoupled from the core code. Anyone can plug in their own menu options without overwriting any scripts.

There are different types of **Properties**, each with its own set of options. All of them have in common that, when retrieving them, an instance of Args parameter is passed, which contains two fields:

- · Target: A reference to the Game Object responsible for calling the property
- · Self: A reference to the Game Object containing the property reference.



There are some cases where the Target and Self fields will reference the same game object.

Property Get types allow to retrieve a value and Property Set types allow to set a value. **Game Creator** comes with a collection of both types, but each module increases the amount available. You can even create your own property types to extend the existing ones.

Property Types

There are a multiple default property types available, which start with $PropertyGet \sim and end with the type.$ For example PropertyGetNumber is the property type for numeric values, and PropertyGetBool is the analog for boolean (true/false) values.

There are also property setters, which allow to set the value of a property. They start with $PropertySet \sim$ and also end with the type name. For example the PropertySetGameObject allows to set the value of a game object reference.

Using Properties



Using properties requires the Editor scripts to be written using Unity's UI Toolkit. IMGUI is not supported.

To use a property it's very simple. You just need to declare them as you would with a primitive type, but instead of getting the value directly, call the Get(args) method to retrieve its value.

For example, let's say that in a component, you want to get a string value. Instead of declaring a value like this:

```
public string myValue = "This is my string";
```

You could use a property so the source of that string value isn't hard-coded, but set from the Inspector. Like this:

public PropertyGetString myValue = new PropertyGetString();

This will display a dropdown menu on the Inspector with the current option selected. By default it's a constant string, but the value can be chosen to come from the name of a game object, a local or global variable, etc.

To get the value you simply call the Get(args) method:

```
string value = this.myValue.Get(args);
```



The Args (arguments) class is a two-field struct that contains the game object considered as the *source* of the call as well as the *targeted* game object. This class is necessary in order to use properties that reference the "Self" or "Target" values. If you are not sure what the self and target objects are, simply pass in the current MonoBehaviour's game object:

```
Args args = new Args(this.gameObject);
```

Creating Properties

Just like **Instructions** and other visual scripting nodes, one can create custom properties to interact with other assets or custom code. To create a new GET Property simply inherit the <code>PropertyTypeGet[TYPE]</code>.

Example of a custom number property

For example let's say we want to create a custom number GET property that always returns 42 (for some reason). In that case we create a new script that inherits from PropertyTypeGetNumber called GetNumber42:

```
public class GetNumber42 : PropertyTypeGetNumber
{
   public override Vector3 Get(Args args)
   {
      return 42;
   }
}
```

Adding fields

You can also expose fields just like you would do in a custom Inspector script. For example, if you want to display an integer field which will be returned by the property you can do so:

```
public class GetNumberInteger : PropertyTypeGetNumber
{
    [SerializedField] private int myNumber = 42;

    public override Vector3 Get(Args args)
    {
        return this.myNumber;
    }
}
```

You can add as many fields as you want. Even other properties.

To return the value the Get(args) method must be implemented. The Args parameter contains the Self and Target values calling the property, which can be used to dynamically get the final value if necessary.

Optionally the PropertyTypeGetNumber child class can also override the String property to display a nicer title in the Unity Inspector. For example:

```
public class GetNumber42 : PropertyTypeGetNumber
{
   public override Vector3 Get(Args args)
   {
      return 42;
```

```
public override string String => "42";
}
```

Property classes can also be decorated with the Title, Category, Description and other attributes, just like it is done on **Instructions** and other visual scripting nodes.

1.7.6 Save & Load

Saving and Loading

Game Creator comes with a flexible mechanism to keep track of changes made at runtime and store these by calling a simple Save() method. Likewise, restoring any previously saved game can be done executing a Load() method from the class responsible for managing this functionality.

CHOOSING WHERE AND HOW TO SAVE DATA

Game Creator makes it very easy to choose how data is saved. By default it uses the *Player Prefs* system from Unity, which stores plain data in a very specific location which is determined by the platform running the game.

Since version **2.5.20** Game Creator allows to choose where to save data, by opening the **Settings** window (top toolbar Game Creator Settings) and opening the *General* tab.

Choosing the Save system

WHAT CAN BE SAVED AND LOADED

The Save/Load system can save any primitive serializable field: integers, booleans, strings, positions, rotations or any managed instance type marked with the [System.Serializable] attribute.

However, it does not serialize objects inheriting or fields referencing objects that inherit from Unity.Object. For example: Game Objects, Transforms, MonoBehaviours, ...

SAVE SLOTS

Most games allow to store multiple saves and allow the user to choose which one to restore when loading a previous saved play. With **Game Creator**, each one of these save spaces are called **slots** and they are represented by an integer number ranging from 1 up to 999.



Notice that you can have up to 998 slots. The number 0 is reserved for $\it shared settings.$

SAVING

To save a game, it's as easy as calling the <code>Save(slot: integer)</code> method through the <code>SaveLoadManager</code> singleton class. This class is responsible for tracking all objects in the scene and silently collects their state in a background process. Saving a game can be done using the following line, passing a constant save slot number 1 as a parameter:

```
SaveLoadManager.Instance.Save(1);
```

By default, the saving system uses Unity's *PlayerPrefs* system, which blocks the main thread until al data is written. However, **Game Creator** provides tools that allow to customize how data is saved. You could even have an online database where you dump the player's save files.

Because we can't assume the saving will be done synchronously, the <code>Save(slot: int)</code> method returns a <code>Task</code> that can be awaited. This is very useful if you plan on synchronizing the game save with an external database, such as <code>Steam</code>, <code>Firebase</code> or any other online data warehouse service.

To handle these cases, all that needs to be done is use an async method and await the result. Like so:

```
public async Task MySaveFunction()
{
    Debug.Log("Start saving game");
    await SaveLoadManager.Instance.Save(1);
    Debug.Log("Game has been saved");
}
```

However, if you are using the default *PlayerPrefs* save system or your custom one does block the main thread when saving, you can either await the task or use a discard operator:

```
public void MySaveFunction()
{
    Debug.Log("Start saving game");
    _ = SaveLoadManager.Instance.Save(1);
    Debug.Log("Game has been saved");
}
```

LOADING

Loading a previously saved game is very similar to saving one.

It is important to highlight that loading a game forces to unload the current scene and loads the saved one afterwards. Even if they are the same.

```
Save Load {\tt mManager.Instance.Load} ({\tt 1}) \,;
```

The Load(slot: int) method returns a Task object, just like the Save(slot: int). You can choose to either await the load or, in most cases, use the discard operator:

```
public void MyLoadFunction()
{
    _ = SaveLoadManager.Instance.Load(1);
}
```

DELETING

A user may want to delete all the information associated to a save slot. This can be done using the following line:

```
SaveLoadManager.Instance.Delete(1);
```

The Delete(slot: int) method also returns a Task object. However, in this case, it may be more interesting knowing when a delete operation has finished.

EVENTS

The saving and loading system contains 6 events that programmers can hook onto to detect when a saving and a loading process has started.

- public event Action<int> EventBeforeSave;
- public event Action<int> EventAfterSave;
- public event Action<int> EventBeforeLoad;
- public event Action<int> EventAfterLoad;
- public event Action<int> EventBeforeDelete;
- public event Action<int> EventAfterDelete;

For example, doing something when a save operation is about to start can be achieved subscribing to the EventBeforeSave event:

You can subscribe to as many methods as you need in each event. However, make sure to remove the subscription when the class that is doing subscribing is destroyed. For example, following the excerpt from above, it would also be optimal to do:

```
void OnDestroy()
{
    SaveLoadManager.Instance.EventBeforeStart -= this.OnBeforeSave;
}
```

CUSTOMIZE

As mentioned before, **Game Creator** doesn't assume a specific save or load procedure. In fact, it provides with tools to customize how data is collected and stored in order for the developer to customize it and tailor it to its needs.

In the following sections we'll see how to:

- Create a custom class that can be saved
- Create a custom database communication service

Custom Data

The SaveLoadManager class keeps track of all savable objects in the scene and collects their state in a background process so when the Save() method is invoked, it contains all the information required to successfully perfom the oppration.

In order to let the SaveLoadManager know what objects it needs to keep track of the developers need to implement the IGameSave interface on each object that contains data to save.

As soon as the object is available, it must call the Subscribe(reference: IGameSave, priority: int) method. Likewise, when the object is destroyed it should call Unsubscribe(reference: IGameSave).

THE IGAMESAVE INTERFACE

The IGameSave interface requires to fill the following methods and properties:

- string SaveID: Gives an id that uniquely identifies this data
- · bool IsShared: Tells whether this data is shared across all save games
- · Type SaveType: Returns the type of the object to be serialized and stored
- · object SaveData: Returns the instance of the object that's going to be saved
- · LoadMode LoadMode: Define whether loading happens following a Greedy or a Lazy format
- void OnLoad(object value): Callback for when the game is loaded

In order to understand better how this works, it's better to demonstrate this with an example.

Let's say that in our game we have one single chest in a scene that the player can only open once.

```
public class MyChest: MonoBehaviour
{
    public bool hasBeenOpened = false;

    public void OnOpen()
    {
         Debug.Log("Do something, like giving a potion to player");
         this.hasBeenOpened = true;
    }
}
```

In order to keep track of whether the chest has been opened or not, we implement the IGameSave interface on the component that defines the behavior of the chest:

```
public class MyChest: MonoBehaviour, IGameSave
    public bool hasBeenOpened = false:
    public void OnOpen()
        if (this.hasBeenOpened) return;
        Debug.Log("Do something, like giving a potion to player");
        this.hasBeenOpened = true:
   // The id for this save game is 'my-chest'
public string SaveID => "my-chest";
    // This save should not be shared across multiple slots
    public bool IsShared => false;
    // The object type we're going to be saving
    public Type SaveType => typeof(bool);
    // The value we're going to store
    public object SaveData => this.hasBeenOpened;
    // The loading mode should be set as lazy
    public LoadMode LoadMode => LoadMode.Lazy;
    // When loading the game, restore the state
    public void OnLoad(object value)
        this.hasBeenOpened = (bool)value;
```

Most fields should be self explanatory. It is importnat to highlight though, that it's up to the developer to implement how the state is restored. The <code>OnLoad(object value)</code> is called when a game is loaded, and the <code>value</code> parameter is the value from a previously saved game. It's the developer's responsability to cast the object value to a valid type and assign the values to whichever fields are necessary.

The Load Mode is a tricky concept. It's an enum that allows to choose between two options:

- Lazy: This should be the default option for 90% of the cases. When this option is selected, the save and load system will restore the state of an object when this object is created. Not before.
- **Greedy**: This requires a persistent object that survives cross-scene transitions (set as <code>DontDestroyOnLoad()</code> method). Most commonly used with singleton patterns, this mode forces the load as soon as the event is triggered.

SUBSCRIPTION

Now, all that's left to do is tell the SaveLoadManager to keep track of this component as soon as it's initialized, and unsubscribe from it when the component is destroyed. Following the previous example, we implement the <code>OnEnable()</code> and <code>OnDisable()</code> Unity methods to subscribe and unsubscribe respectively:

```
public class MyChest: MonoBehaviour, IGameSave
{
    public bool hasBeenOpened = false;

    void OnEnable()
    {
        _ = SaveLoadManager.Subscribe(this);
    }

    void OnDisable()
    {
        _ = SaveLoadManager.Unsubscribe(this);
    }

    // IGameSave implementation below
    // ...
}
```

This gives all the necessary information to the save and load system about the life-cycle of this object so it can keep track of its state progress. If your object is never destroyed and survives scene transitions, you can skip the unsubscription.

To wrap things up, here's the full script of the example:

```
public class MyChest: MonoBehaviour, IGameSave
    public bool hasBeenOpened = false;
    public void OnOpen()
        if (this.hasBeenOpened) return;
        Debug.Log("Do something, like giving a potion to player");
        this.hasBeenOpened = true;
    void OnEnable()
        _ = SaveLoadManager.Subscribe(this);
    void OnDisable()
        _ = SaveLoadManager.Unsubscribe(this);
    public string SaveID => "my-chest";
    public bool IsShared => false;
    public Type SaveType => typeof(bool);
public object SaveData => this.hasBeenOpened;
    public LoadMode LoadMode => LoadMode.Lazv:
    public void OnLoad(object value)
        this.hasBeenOpened = (bool)value;
```

The hasBeenOpened property will always return false if the OnOpen() method has never been executed, but will return true if it has at some point. If the user saves and loads back the game, its value will be kept.

Custom Save Location

By default, **Game Creator** saves games using the *PlayerPrefs* built-in system. However, although this solution is cross-platform and will work for most users, some might prefer to sync their saves with an online database or use a different system than Unity's *PlayerPrefs*.

Here we will explore how easy it is to extend the save location.

IDATASTORAGE INTERFACE

To create a custom save location, one must create a class that implements the IDataStorage interface, which contains all the necesary methods to store game information.

To make things easier, we're going to create a very simple system that communicates with an online database and stores the game saves there using http requests.

Note

Notice that there aren't any error handling mechanism for sake of simplicity. A production-ready product should also check and inform of the necessary errors that may ocurr.

Let's create our storage location class called MyOnlineDatabase.cs:

```
[Serializable]
public class MyOnlineDatabase: IDataStorage
    private const string URL_DB_SET = "https://database.mywebsite.com/set";
    private const string URL_DB_GET = "https://database.mywebsite.com/get"
private const string URL_DB_DEL = "https://database.mywebsite.com/del"
    string IDataStorage.Title => "My Online Database";
    string IDataStorage.Description => "Store data in online database";
    async Task IDataStorage.DeleteAll()
         // Create a web request to delete the content
        UnityWebRequest request = UnityWebRequest.Post(URL_DB_DEL, "");
UnityWebRequestAsyncOperation handle = request.SendWebRequest();
         while (!handle.isDone) await Task.Yield();
    async Task IDataStorage.DeleteKey(string key)
         // Create a web request to delete a key
        UnityWebRequest request = UnityWebRequest.Post(URL_DB_DEL, key);
UnityWebRequestAsyncOperation handle = request.SendWebRequest();
         while (!handle.isDone) await Task.Yield();
    async Task<bool> IDataStorage.HasKey(string key)
         // Checks whether a key exists in the database (code 200)
         UnityWebRequest request = UnityWebRequest.Post(URL_DB_GET, key);
         UnityWebRequestAsyncOperation handle = request.SendWebRequest();
         while (!handle.isDone) await Task.Yield();
         return handle.webRequest.responseCode == 200
    asvnc Task<object> GetBlob(string key, Type type, object value)
        // Create a request to get the value identified by a key
UnityWebRequest request = UnityWebRequest.Post(URL_DB_GET, key);
         UnityWebRequestAsyncOperation handle = request.SendWebRequest();
         while (!handle.isDone) await Task.Yield();
        return JsonUtility.FromJson(
             handle.webRequest.downloadHandler.text,
             type
    async Task<string> IDataStorage.GetString(string key, string value)
    async Task<float> IDataStorage.GetFloat(string key, float value)
    async Task<int> IDataStorage.GetInt(string key, int value)
    async Task SetBlob(string key, object value)
```

```
{
    // Requests the creation or update of a value onto the database
    UnityWebRequest request = UnityWebRequest.Post(URL_DB_SET, new Data(){
        id = key,
        data = JsonUtility.ToJson(value)
    });
    UnityWebRequestAsyncOperation handle = request.SendWebRequest();
    while (!handle.isDone) await Task.Yield();
}

async Task IDataStorage.SetString(string key, string value)
{ /* ... */ }

async Task IDataStorage.SetFloat(string key, float value)
{ /* ... */ }

async Task IDataStorage.SetInt(string key, int value)
{ /* ... */ }
```

The first properties Title and Description allow to give a name to this system, which later can be selected from a dropdown menu in the Preferences window.

The following methods define how data is manipulated: retrieving data, setting data and deleting data. There are 3 URL we're using to exemplify how we can create an http request to send the information to our server, which can delete, create or retrieve the information depending on the endpoint used.

Some methods have been skipped because their implementation was very similar to other ones.

It is important to note though that all methods have the async prefix and either return a Task object or a Task associated with an object. This is because there's a certain amount of time elapsed between the http request and the answer from the server. Being able to await requests let's you tailor how to safely chain commands and make sure each request is successfully fulfilled.

Remember

The **Remember** component allows to cherry-pick the data that is stored when saving the game. By default, it stores the position, rotation and scale.

Remember

To add a new element to be saved, click on the Add Memory button and select the type of data to save.

CREATING A MEMORY

Game Creator comes with a set of default memories, but you can create custom ones that extend the data stored. To create a new **Memory** create a new class that inherits from the Memory class. For this example, we'll create a memory that saves name of the game object attached to this memory.

```
[Serializable]
public class MemoryName : Memory
{
    public override string Title => "Name of Game Object";

    public override Token GetToken(GameObject target)
    {
        return new TokenName(target);
    }

    public override void OnRemember(GameObject target, Token token)
    {
        if (token is TokenName tokenName)
        {
            target.name = tokenName.text;
        }
    }
}
```

The Title property determines the name of this memory. This has no effect on the data stored but it displays this value on the Inspector.

The GetToken(...) method returns the Token instance of this memory and is called when the game data is scheduled to be saved. A Token is a data container that contains the data to be stored. In this case, we'll need to create a new class called TokenName that inherits from Token and has a serializable field to save the name of the object.

```
[Serializable]
public class TokenName : Token
{
    public string text;

    public TokenName(GameObject target) : base()
    {
        this.text = target.name;
    }
}
```

The OnRemember(...) method is called when loading a previously saved game and is used to restore its state. In this case, it changes the name of the game object to the one it tries to remember.

Decorations

The custom Memory class instance can be decorated using any of the attributes found in the Instruction, Condition and Event classes.

1.7.7 Tween

Tweening is the process to define a starting position and an end position, and let it transition from one to the other over the course of a specified duration.

For exmaple, opening a door can be easily achieved defining it's starting position as its current position and its end point as the same as its starting one, plus 2 units up in the Y axis. Once you specify the duration, the door will slide upwards when the tweening is activated.

The Tweening library has been created with Game Creator in mind, but can also be leveraged to be used in other scripts. Use the Tween.To(...) static method to create a new transition.

The To(gameObject, input) has two parameters: The Game Object that recieves the tweening, and an instance of a TweenInput class, which configures the animation.

Following the example from above, let's say we want to slide a "door" object 2 units up in the air. We can define the TweenInput class instance like this:

```
Vector3 valueSource = door.position;
Vector3 valueTarget = door.position + Vector3(0,2,0);
float duration = 5f;

ITweenInput tween = new TweenInput<Vector3>(
    valueSource,
    valueTarget,
    duration,
    (a, b, t) => door.position = Vector3.Lerp(a, b, t),
    Tween.GetHash(typeof(Transform), "transform"),
    Easing.Type.QuadInOut
);
```

†Pansition Type

In this example we use a Vector3 transition, but it accepts any value type, like numbers, colors, quaternions, ... It's up to the *updateCall* to interpolate between the initial and final value.

Let's break down each of these parameters in order:

```
TweenInput<Vector3>(
    Vector3 start,
    Vector3 end,
    float duration
    Update updateCall,
    int hash,
    Easing.Type easing
);
```

- $\boldsymbol{\cdot}$ $\boldsymbol{\text{start:}}$ A value indicating the starting position
- $\boldsymbol{\cdot}$ $\boldsymbol{end:}$ A value indicating the end position
- duration: The amount of time it takes to complete the transition
- updateCall: A method called every frame while the transition occurs. Contains 3 parameters: The starting value, the end value and the completion ratio between 0 and 1.
- hash: An integer that uniquely identifies this transition. If another transition with the same id starts, it cancels the previous one.
- · easing: An optional easing function. If none is provided, it will use a linear function.

1.7.8 Custom Installs

Game Creator comes with the **Install** window, which allows a user to install and uninstall examples and templates from all modules. This is something available to all module developers and here you'll learn how to create, step by step, a template for a module called "My Module".

Installer

The **installer** directory is where the compressed file with the information about it is located. This folder is usually found under the custom Module's path but can be anywhere on the project folder. It must contain two files:

- An **Installer** configuration file, which contains all the information related to the example, including its name, the module it belongs to, a description and the version of this package.
- · A Package.unitypackage file, which contains the compressed assets that will be unpacked upon installing.

Installation Location

The installed location is the directory where the example is decompressed after installing an example in order to be used by the user. This folder is always located at the following route:

```
Assets/Plugins/Game Creator/Installs/
```

An installed extension will always have a folder parent called after the name of the module, followed by a dot, followed by the name of the example, followed by an @ symbol and the semmantic version of the example. For example, if the example is called "My Example" and it's from a module called "My Module", the installation location of the example will be:

Assets/Plugins/Game Creator/Installs/MyModule.MyExample@1.0.0/

Creating a custom Installer

The example installer can be placed anywhere in the project. For simplicity it should be created where you have the rest of the module's assets. For example, if you are creating a module called "My Module" and an example of that called "My Example", at the root of the Unity project, you may want to place the installer inside the MyModule folder:

```
Assets/
MyModule/
Examples/
MyExample/
Scripts/
Textures/
...
```

THE INSTALLER ASSET

Now that there is a folder where we can drop in the installation files, we'll create an **Installer** asset inside the **MyExample** folder. To do so, right click on the aforementioned folder and select Create -> Game Creator -> Developer -> Installer. If the option doesn't appear, you can also duplicate any existing Installer asset. Once you have the Installer asset you can rename it so it makes sense for your project.



We recommend sticking to Game Creator's naming convention and name the asset following "[ModuleName]. [ExampleName]". This makes it easier to identify the asset and avoids conflicting names with other examples from other modules.

With the **Installer** in place, click on the *Configuration* button to expand the properties available and fill in the fields:

- · Name: Name of the Example. Following the example from above, this would be "My Example.
- Module: Name of the module. It is important to note that this determines the category of the example. In the use case from above, the name would be "My Module".
- Description: A thorough description of this example. Make sure to indicate any quirks the example may have or how to get started once the example is installed.
- Author: Name of the creator of this example. This has no implication other than giving credit to the
- **Version:** The semmantic version of this example. Make sure to increase the value every time you create a new version of the example.
- Complexity: How difficult it is for users to understand this example. This is for informational purposes only.
- Dependencies: A collection of ID (module name + example name) that this example depends on.

Paling with Dependencies

The **Install** window will automatically install any dependencies that an example may depend on, without prompting the user to do so. This allows to quickly resolve any conflicts between this example and others that are required to be installed.

For example, if the example Example A has Example B as a dependency, and this last one is not yet installed, attempting to install Example A will install both Example A and Example B.

If Example B cannot be found, it won't be possible to install Example A from the Install window and will prompt the user an error message telling which module could not be found.

MAKING THE SKELETON

Now that we have the installer in place it's time to create the skeleton from which to build our example. To do so, select the previously created **Installer** and in the Inspector, right click on the name of the installer. This will make a dropdown menu appear with a bunch of options:

- Install Package: Forces the installation of this example. However, it is recommended to use the Install window to perform any installation instructions.
- Delete Package: Deletes the installed example, if there's any.
- **Build Package:** Changes the name of the installation path to fit the version number and creates a *Package.unitypackage* file at the installation location.
- · Create Package: Creates the bare bones structure that allows to develop a new example.

In our case, we want to click on the "Create Package" option. This will create a new folder at:

Assets/Plugins/Game Creator/Installs/MyModule.MyExample@1.0.0/

Inside this folder you can place all prefabs, materials, scenes or any content that the example must have. To generate (or compress) this folder so it can be shared, select the option "Build Package" from the previous dropdown menu. This will export all assets inside the aforementioned folder and create a file called Package.unitypackage at the same directory as the Installer.

Sharing your example

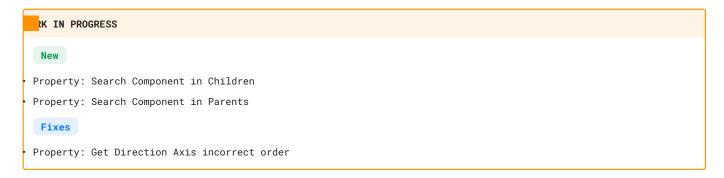
Once you have the example built, it is ready to be distributed. To share this example installer, you just need to export the folder with the installer and the Package unitypackage file generated.

If you (or the user) opens the Install window, the module will be displayed as a sub category of the specificed module with the option to install it, update it and/or delete it, depending on whether there is an installed version or not.

1.8 Releases

1.8.1 Releases

2.14.49 (WIP)



2.14.48 (Latest)



2.14.47

```
New
Property: Direction for Input Action
Property: Decimal for Input Action
Fixes
Regression: States no longer transition properties
Trigger: Flick uses magnitude of Vector 2
```

2.14.46

eased January 9, 2024 New Internal: New high performance spatial hashing Condition: Is Speech Playing by Target Condition: Character Fits · Condition: Check Capsule Instruction: Draw Gizmo Line Properties: Materials access by Index · Properties: Decimal Condition Enhances · State: Properties change over time during transition Changes Instruction: Enter State uses a animation clip Property Fixes Save/Load: Scenes loading twice when loading a game Save/Load: Invisible exception thrown saving the game · Driver: Navmesh Agent now detects collisions · Driver: Rigidbody radius not reacting on change · Driver: Rigidbody incorrect drag value when airborne · Tweening: Incorrect order when duration is zero Condition: Box 2D incorrect return value

2.13.45

eased November 12, 2023

This update does not come with any breaking changes. However it requires to uninstall Game Creator first before upgrading in order to replace the old search engine for the new one.

New

· Editor: New search engine

Enhances

· Runners: Toggle visibility in Project Settings

Fixes

- Script: Incorrect location of some scripts
 Property: Get Position Vector axis values
- Demos: Characters example wrong jump animation

2.13.44

New Instruction: Set Character Busy Instruction: Set Character Available Fixes Load: Ignore loading an empty save slot Instruction: Change Position incorrect use local/world space Instruction: Change Rotation incorrect use local/world space Input: Incorrect empty string check on Action Maps Camera: Avoid Clipping checks for null ignorers

2.13.43

leased October 31, 2023

This version comes with new features, but also breaks compatibility with the previous core and submodule versions.

We recommend updating only if you're in a prototyping phase or you're far from releasing your project(s).

In order to upgrade, be sure to backup your project(s) first and uninstall any previous versions of Game Creator and its modules.

New

- Instruction: Hotspots Active
- Instruction: Prewarm pool of Game Objects
- · Instruction: Destroy pool of Game Objects
- · Instruction: Set Animation Clip
- · Instruction: Enable Input System Action asset
- · Instruction: Disable Input System Action asset
- · Condition: Check OR Conditions list
- · Condition: Check AND Conditions list
- Property: Material type
- Property: String with ID format
- · Property: Decimal have Math operations
- · Property: Get and Set Sprite Renderer
- UX: Favorite Signals stored per-project
- · Props: Handle asset for handling multiple characters
- · Variables: Material value references
- Variables: Animation Clip value references

Enhances

- Performance: Improved performance of Instructions
- Camera: Decoupled Avoid Clipping systems
- · Pooling: Improved performance managing instances
- Variables: Drop zones to auto-fill List Variables
- · Instruction: Move To detects when it gets stuck on a path

Changes

- Hotspot: More flexible options
- Input: Input Action assets require manual enable/disable
- Properties. Faster and new Location system
- · Properties: Replaced Camera properties with Game Object
- · Properties: Replaced Shot properties with Game Object
- Properties: Rearranged Game Object properties
- Scripting: Audio Instructions use Audio Properties
- Variables: Use built-in polymorphic serialization
- Characters: Jump and Land no longer set Legs as Busy

Fixes

· Trigger: Interaction working when Trigger is inactive

· Saving: JSON File with Encryption throws error

· Staging: Forbid open in play-mode

Editor: Error uninstall module with Settings open
 UX: Vertical alignment of elements in Inspector

· UX: Use Raycast field expands in Inspector

Instruction: Submit UI uses all components in game object
 Character: Screen Center interaction with elements behind
 Character: Screen Cursor interaction with elements behind
 Saving: Exist differentiates between unloading and destroying

· Characters: Player input direction depending on Y axis

Removes

· Hotspot: Look at with Focus

· Instruction: Change Hotspot Radius

Characters: Twitching an Breathing layers deprecated
 Properties: Offset properties are Direction properties

2.12.42

leased August 29, 2023

New

Hotspot: Play Audio on Enter/Exit

Condition: Check Audio Effect is Playing

Condition: Variables List is Empty

· Condition: Is UI Sound Playing

· Condition: Is Music Playing

Condition: Is Ambient Playing

· Condition: Is Speech Playing

Property: Check Conditions

Property: Constant Boolean True

· Property: Constant Boolean False

Property: Boolean Local/Global List is Empty

· Property: Boolean Local/Global List has entries

Property: Rotation Euler by Axis

· Character: Rotation Towards Input

Character: Change Is Controllable from State

· Character: Option for Simple Ragdoll

Character: Option for No Ragdoll

IK: Ported Look Track system to Animator IK

Enhances

· UX: Sorting Lists in the Inspector

UX: Reorganized Axonometry dropdown

UX: Reorganized Hotspots dropdown

Changes

Hotspot: Option With Focus set as a Hotspot field

Character: Decoupled Skeletons and Ragdolls

Fixes

Conditions: Raycast better titles

Camera: Shots use correct orthographic size

Camera: Residual Zoom in Third-Person Shots

Camera: Head Bobbing throwing NaN value

Audio: Stop playing after suspending runtime app

Removes

Hotspot: Look at with Focus

Hotspot: Activate Object with Focus

Hotspot: Instantiate Prefab with Focus

Hotspot: SHow Text with Focus

· Camera: Removed preview of Camera Shots

· IK: Look Track system with Animation Rigging

· Editor: Dependencies with Animation Rigging

2.11.41

eased June 12, 2023. New Instruction: Change Axonometry Instruction: Change Audio Source volume Instruction: Change Audio Source pitch · Hotspot: Activate Game Object Hotspot: Activate Game Object on Focus Hotspot: Renamed Activate Prefab to Instantiate Prefab Enhances Hotspot: Optional infinite radius value Changes Local Variables: Access using Game Object Properties Fixes · Interaction: Near Mode using wrong Character Input: Wrong setup when using Input System Actions Input: Wrong Action selection when retrieving by Map Input Field not returning TMP component value · Property: Location Character with Offset values Mobile: Support for Point & Click and Follow Cursor · Global Variables: Not resetting after loading without save

2.10.40

New Events: New Command Args entry point Enhances Camera: Search for any Camera Shot after setup Fixes Editor: Error creating Player from context menu Input: Loading before scene is set up Removes Props: Removed obsolete api Signals: Removed obsolete api Location: Removed obsolete Property Tool class

2.9.39

eased May 25, 2023 New Instruction: Next Iterator for List Variable Instruction: Previous Iterator for List Variable · Instruction: Random Iterator for List Variable · Instruction: Calculate Modulus between two numbers · Instruction: Change Mannequin Position · Instruction: Change Mannequin Rotation · Instruction: Change Mannequin Scale · Instruction: Clear Looking Around · Property: Get String of Saved Slot Date · Input: Finger in Screen and World Space · Component: Text Property String UI component · Console: Save/Load Game commands Enhances • Execution: New execution order for certain components Fixes · Variables: Not collecting correctly during build · Variables: Reset Game does not reset variables

2.9.38

Leased May 13, 2023 Fixes Settings: Opening during domain reloads Variables: Initialize before subscribers Events: Initialize before subscribers

2.9.37

New Console: Runtime console for debugging Instruction: Log Text on Runtime Console Instruction: Submit Command to Runtime Console Instruction: Open/Close/Toggle Runtime Console Variables: Refresh button in Settings window Enhances Examples: New Console example scene Fixes Variables: Not loading on standalone builds Settings: Window missing entries on startup

2.9.36

```
eased May 8, 2023
 New
 Instruction: Change Local Variable ID
· Instruction: New Dash instruction parameters
· Variables: Set at Index uses dynamic property
 Driver: Axonometry field for Isometric games
 Driver: Axonometry field for Side-Scroller games
 Enhances
 Execution: Order in which components are initialized
 Fixes
 Build: Crash when compling to Windows or Android
 Font: Incorrect resource font name
 Instruction: Move to Transform rotation ignored
· Variables: Edge case Global Variables not loading

    Trigger: On Blur throws error if object destroyed

· Align: Name Variables pick field
· Align: Input System pick field
· Align: Navigation Area Mask field
· Align: Navigation Agent field
Align: Layer Mask field
· Repository: Deferred initializing database
 Combat: Not set default Target to first Candidate
 Touchstick: Icon not appearing on skin asset
```

2.9.35

leased March 28, 2023

Enhances

· Icon: Equated dropdown arrow to standard

Fixes

Build: Forbidden access of m_Dirty memberAlign: Labels in Instantiate instruction

· Align: List Variables pick field

eased March 24, 2023

New

- Settings: Custom scene when loading application
 Settings: Displays current and update version
 Settings: Notification when there is an update
- Instruction: Change Character ID
 Instruction: Change Hotspot Radius
- · Instruction: Set Sprite
- Instruction: Set Character Combat Target
 Instruction: Clear Character Combat Target
- · Instruction: Reset Character Vertical Velocity
- · Instruction: Set Character Invincibility
- · Instruction: Set Character Poise
- · Instruction: Cycle to Closest Target
- · Instruction: Cycle to Next Target
- · Instruction: Cycle to Previous Target
- · Instruction: Cycle to Target by Direction
- · Instruction: Enable/Disable Collider
- · Instruction: Enable/Disable Renderer
- Event: On Change Combat Target
- Event: On Change Invincibility
- Event: On Dash
- Event: On Dodge
- Event: On Input Flick
- · Condition: Is Character Invincible
- · Condition: Character Raycast Floor
- · Condition: Compare Vertical Distance
- · Condition: Compare Horizontal Distance
- Property: Character Combat Target
- Property: Game Object Floor Position
- · Property: Get Last Dodge Location
- · Property: Get Last Dodge Position
- Property: Get Poise
- Property: Get Defense
- · Input: Gamepad Left/Right Stick
- Input: Mouse Scroll-Wheel
- · States: Change Motion fields from within States
- · IK: Aim Towards Rig for First Person mode
- · Examples: New Interaction and redone some others

Enhances

- Variables: Access index with dynamic Property
 Instruction: Finer grain control over Dash
- Condition: Non-Alloc raycast methods for Physics
 Camera: First Person has smoother bobbing and sway
- Examples: New model for First Person mode
 QoL: Right click to Expand/Collapse items
 QoL: Facelift of the Character inspector

Changes

- · Variables: Save is disabled by default
- · Event: On Character Step renamed to Character On Step
- Event: On Input renamed to On Input Button
- Physics: Boolean result when checking raycasts

Fixes

- · Instruction: Dash wrong left/right fields
- · Condition: Physics reported opposite result
- · Footsteps: Avoid creating material instances
- · Character: Allow zero linear speed
- · Character: States dangling after Stop if delayed
- · Character: Missing Avatar changing model in Editor
- · Character: Incorrect bones when wearing Skinned Meshes
- · Variables: Conversion from Texture throws exception
- Font: Use LegacyRuntime.ttf for Unity 2022.2 or higher
- Property: Get/Set Sprite UI Image incorrect name
- Editor: Changing Character unit throws error if selected

2.8.33

eased January 31, 2023

Fixes

- Skeleton: Error in Unity 2022.2 and higher
- Camera: Moved execution to Late Update
- Camera: Incorrect execution order

2.8.32

eased January 30, 2023

New

· Skeleton: Revamped the whole workflow

· States: Root Motion for Entry/Exit clips

• Event: On Camera Change Shot

• Event: On Change to/from Camera Shot

· Property: Option to get Character Look Target

· Property: Get Character Bone position

Enhances

• IK: Feet on ground uses softer values

Hotspots: Show Text can use Text Mesh Pro components

Fixes

· Animim: More consistent Gestures system

· Animim: States spawned in layered order

· Shot: Lock On Camera jitters when close to target

· Audio: Forbids playing the same clip at the same frame

· Ragdoll: Humanoids without Neck or Chest bones

• IK: Looking at a target jitters spinal chain

· Variables: Local ones not correctly initialized

· Interaction: Control Labels do not properly display

2.8.31

eased January 8, 2022 New Dash: new Character settings under Motion IK: Align Body with Ground Instruction: Stop Character Dash · Instruction: Change Input Field · Instruction: Move List Variable Instruction: Change Orthographic Size Property: Unity Editor Version Property: Application Version Enhances Character: Not allowed to stand on top of others Instruction: Dash uses improved Dash feature Shots: Viewport changes use Transitions duration Icons: New UI icons for common components Fixes · Variables: Loading missing Global Name Variables · Variables: Loading missing Local Name Variables · Instruction: Cross product operator symbol · Property: Location Game Object uses Markers · Property: Location Game Object with Offset uses Markers Property: Display hidden labels on some Variables · Gizmos: Error when inspecting a prefab with Gizmo calls Removes · Property: Get Player Bone Location · Property: Get Player Bone with Offset

2.8.30

leased December 8, 2022

Fixes

· Props: Retrocompatibility with previous versions

2.8.29

leased December 8, 2022 New · Instruction: Drop Prop from Character Instruction: Reset Game Example: Nested variable access Enhances · Instruction: Attach Prop accepts direct instance Instruction: Remove Prop accepts direct instance Shots: Reduced the amount of unselected gizmos Changes · Faster Spatial Hash with better layout Prop system accepts prefabs and instances Fixes · Ragdoll: Joint constraints use projection · Variables: Error if two variables have same ID · Remember: Error if two Remembers have same ID · Deter IL2CPP from stripping certain assemblies · States example where character does not sit down

2.7.28

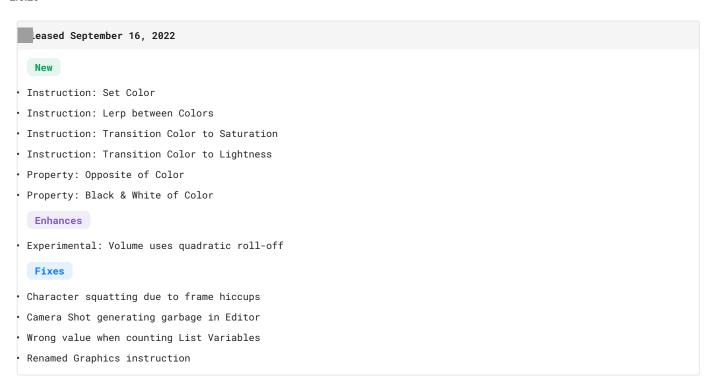
Consistent Label width in 2021 LTS

Time scale not reaching zero with transition

eased November 8, 2022 New Character: New feet phases curves Character: New Rigidbody driver controller Camera: Shots can override camera values Instruction: Set Vector X/Y/Z value Condition: Is Foot Phase on Ground Condition: Is Character Humanoid • Event: On Application Focus • Event: On Application Pause · Event: On Application Quit Property: Find Game Object by Tag · Property: Constant Zero · Property: Constant One Property: Round Decimal Property: Ceiling Decimal Property: Floor Decimal Example: Loop List Variables Changes Phases: New character foot planting Character: Jumps with the grounded foot · Removed 'In Background' field on Actions Removed 'In Background' field on Triggers Input: Using Enhanced Touch Support · Instructions: Removed activate Shot systems Camera: Overhauled Shot systems Character: Split gravity into up/down Fixes Condition: Has State in Layer Memory: Exists game object works again Hotspot: Not allowing multiple cursor icons · Instruction: Change Position incorrect space Brake velocity capped to at least 1 unit · Character: Look at feet instead of eyes

New Copy-runners for Visual Scripting Enhances Support for dropping 3D models improved States use new Copy-runners Sequencing uses new Copy-Runners Fixes Remember components run after initialization

2.6.26



eased September 6, 2022 New Instruction: Transform Direction Instruction: Inverse Transform Direction Instruction: Transform Point · Instruction: Inverse Transform Point Event: On Fixed Update Property: Camera Field of View · Property: Character Model Enhances · Slight Trigger design facelift · Faster Event System queries Rearranged Hotspot menu Redesigned welcome screenshots Fixes · Save/Load: Leak when destroying Remember Character: Unable to move/rotate when dead Inverse Kinematics: Deactivate when is Dead

2.6.24

New Instruction: Change Character Time Mode Instruction: Unfocus UI object Maximum collect radius set to 500 Screen Width/Height numeric Properties Enhances Keywords on visual scripting nodes Fixes Instruction: Add Force has space mode Shot: Third Person direction after transition Pool sets position before activating Save/Load wrong format when deleting slots Time scale affects Character animation Sequence tool creates empty clips

New Character can use Driver/Motion directions Property: Screen and World cursor positions Enhances Touchstick can subclass and override properties Fixes Instruction: Transform Change Position Instruction: Join Text incorrect values Serializable data structures null error Error thrown by polymorphic list items Scroll not appearing on Install window Order in which SaveLoad system is executed Copy & Paste multiple times does deep copy Update serialized object when getting managed

2.6.22

eased June 21, 2022 New New non-diegetic Music audio channel Instruction: Play Music · Instruction: Stop Music · Instruction: Change Music Volume · Instruction: Stop all Music · Instruction: Stop all Ambient • Event: On Change Music Volume · Property: Music volume getters and setters Fixes Gestures and States ignore time scale Character physical material crash · Character not saving position/rotation/scale Instruction: Change Position self space Condition: Raycasts wrong object reports · Props: Incorrect prefab scale

eased June 14, 2022. New Option to uninstall modules Input: Any button Input: Constant Motion · Instruction: Parent of Game Object Instruction: Root of Game Object Property: No Sprite · Character Rotation: Look at Pointer Variables: Show error if duplicate ID Changes Scene Entries decoupled from Instruction Fixes Global List Variables collect methods work Condition: Are Arms Busy incorrect spelling · Serialization error during domain reloads • IK: Deactivate IK Lean at runtime Navigation: Disable rotation in movement

2.5.20

eased May 24, 2022 New · Named Variables: Nested Access Settings: Choose how to save game data · Save: Json File option with simple encryption · States: Instructions run on every change · Instruction: Physics 3D Trace Line Instruction: Physics 3D Overlap Sphere · Instruction: Physics 3D Overlap Box Instruction: Physics 2D Overlap Circle Instruction: Physics 2D Overlap Box Enhances Spatial Hash algorithm performance Tweaks on padding on Editor UI Fixes Acceleration uses vertical speed States using wrong duration for entries

2.5.19

eased May 12, 2022

New

· Instruction: Increment Number

Instruction: Change RectTransform Width
 Instruction: Change RectTransform height

• Instruction: Transform Look At

· Instruction: Scale Product

· Instruction: Change Character Driver

Instruction: Swap List Elements
 Instruction: Start Looking At
 Instruction: Stop Looking At
 Event: On Collide Exit With
 Condition: Raycast 3D and 2D

Condition: Is EditorCondition: Is Batch modeCondition: Is Console

· Condition: Is Mobile

Condition: Check Runtime Platform

Property: Empty StringProperty: Rect TransformProperty: Random Audio Clip

Property: Last Collided Enter/Exit
 Property: Last Trigger Enter/Exit
 Input: Mouse Double Press/Release
 Input: Touchscreen Press/Release

Driver: NavmeshAgent has Agent Type exposed

Memory: Exists

Enhances

· Footsteps support LOD Groups

· Acceleration feels more natural

Keywords for boolean values

Fine-tuned Fuzzy Finder algorithm

· Lock characters on a 2D plane

· Increased scope of Breathing and Twitching

· Breathing and Twitching can be nullified

· Offset values default to zero

Fixes

- · Signals not initialized on AoT platforms
- · Picking component from a Property reference
- Null check before playing Audio Clips
- · States flicker before playing exit clip
- · Deep copy when getting Tree children
- · List picker has dropdown menu title
- · States using wrong entry Avatar Mask
- Exception with uncaught Kernel events
- · Crash with Rigidbody Driver caused by Material

2.5.18

eased March 25, 2022

New

· Condition: Is Character Controllable

Enhances

· Scope for changing character rotation state

- · Input: Holding contains release cycle
- · Input: Renamed properties to follow standard
- · Template character using incorrect height
- · Template player checks if is controllable

2.5.17

eased March 23, 2022

New

Condition: Is Input Pressed
Condition: Is Input Released
Condition: Is Input Held Down
Event: On Input Action Press
Event: On Input Action Release
Event: On Input Action Hold

· Icons for visual scripting nodes

Changes

· Character has a proxy object for model

Fixes

• Orphan object destroying a ragdoll character

· Settings window elements not appearing

· SceneReferene error when building a binary

· Skeleton throw null reference exception

2.4.16

eased March 21, 2022

New

· Player: Follow Pointer Unit

Instruction: Show/Hide Touchsticks

Instruction: Activate Feet IK
 Instruction: Activate Lean IK
 Instruction: Activate Look IK

· Condition: Has Save at Slot

Event: On Load GameEvent: On Delete GameProperty: Audio ClipsProperty: Random Strings

· Input: Detection for Press/Release

Support for TreeView API

Changes

- · Properties return editor references
- Smooth in/out landing compression

- · Sync physics with main thread transforms
- Changing units when character is selected
- · Changing model null exception
- Locating camera by tag overridden by FindByType
- · Looping variables works with all types
- · List selectors not being displayed
- · Changing character height stutters
- · Fields alignment in Inspector
- Signals throw an error when exitingNull error after deleting some objects
- · Spawn prefabs every frame in Point & Click
- · Changing Is Controllable stops any motion
- · Greedy systems automatically initialize
- Description of surface/volume properties
- Instruction: Quit Application exits playmode

2.3.15

eased January 28, 2022 New Game Creator Toolbar Signal dispatching Instruction: Character move to direction · Instruction: Character stop movement · Instructions: Camera Shots Instruction: Raise Signal • Event: Receive Signal • Event: On Change Audio Volume · Property: Audio Mixer Parameter Enhances Run Visual Scripting components from Unity events Easier to navigate dropdown menus · Footsteps textures mimic character rotation A Camera Shot can be assigned as the main one Improved performance of Editor UI elements Changes · Tree class renamed to Trie

- · Variables now accept integers, floats and doubles
- · Some events were invoked when the Trigger was disabled
- · Error thrown with inactive Local Variables

- · Dead characters don't twitch or breathe anymore
- · Async Manager exception throw exiting Play-Mode
- · Settings window compressing overflowing elements
- Event: On Click does not execute over UI elements
- · UI Controls have the UI layer as default
- · Locations allow to specify the rotation
- · Procedural animations take into account Time Scale

2.2.14

eased December 27, 2021

New

Shot: New Anchor Peek camera shot

• Marker: New Inwards type • Instruction: Play Footstep

· Spot: Look on Focus

· Shots: Can use easing functions

· New deep clone utility to duplicate instances

· Properties: Get and Set audio volumes

Enhances

· Spots: Disabled while interacting

Spots: Offset option for World and Self space
 Event: Lifecycle events have better description

Play button is contextually hidden

· Colors have HDR and non-HDR option

· Name Variables display non-available options

String Variables can get values from other types

Changes

· Instruction: Toggle Bool uses one single property

- · Shots interpolate based on its duration
- Event: Characters not registering changes
- · Prefab Variables error at runtime
- · Tweening UI elements uses unscaled time
- · Actions and Triggers catch exceptions
- · Spatial Hash queries on Markers

2.1.13

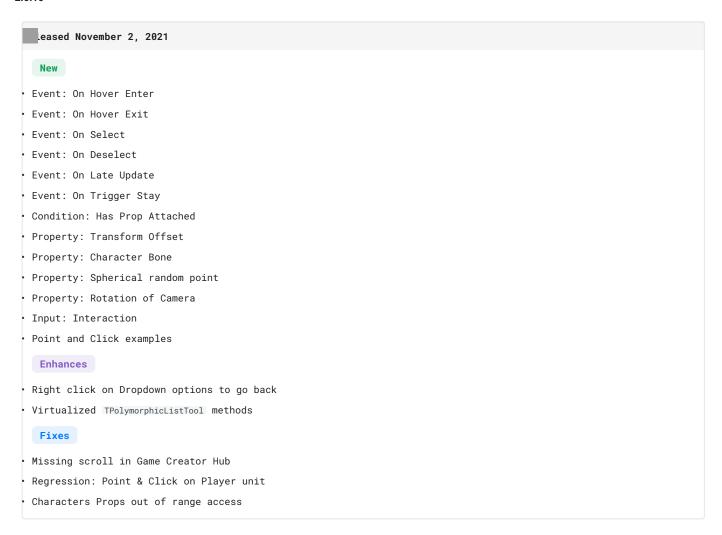
eased December 1, 2021 New Interaction system Condition: Can Interact Instruction: Interact • Event: On Focus • Event: On Blur · Event: On change NPC to Player Event: On change Player to NPC Spot: Text on Focus · Spot: Object on Focus Enhances Leaning IK default values Character inspector UX Conditions have more friendly names Changes · Name of Point and Click button Motion unit is more compartmentalized · Hide Character gizmos collapsing each unit Fixes Spatial Hash returning farther values

2.0.12

Rew Driver: Skin width exposed in Inspector Ragdoll animations Enhances Save/Load format does not use special characters Fixes Shot: Lock On ignores Anchor and Target clipping Event: On Enter NavLink not detected Event: On Exit NavLink not detected NavMesh: Agents move between Off-Mesh Links Scene asset were null in standalone builds Character: Sinking in ground when using Feet IK

New Paste button for Visual Scripting Instruction: Sort List alphabetically States: Weight uses a range control Enhances Conditions redesign Focus on search fields automatically Event: Input distance has offset option Shot: Lock on includes better default values Fixes Airborne animations did not loop correctly Null check for characters Bone Rack

2.0.10



eased October 20, 2021 New Input: Mobile virtual joystick support IK: Lean towards motion direction Event: On Hotspot Activate • Event: On Hotspot Deactivate Property: Light Intensity and Range Enhances Rendering Pipeline in documentation · Visual Scripting search engine precision · Renamed Example Manager to Install window · Renamed execution events to lifecycle path Fixes Character radius out of sync with driver unit · Crouch and Walk string input codes · Memory leak in Camera Shot preview window · Skeleton valid prefab type Conversion between float and double values · Test Runner using float values

2.0.8

New New getters for each Vector3 component Instruction: Clamp Vector3 Enhances Examples with higher contrasting textures Fixes Null check for gamepads and keyboards Null check for material _MainTex Input for walking using crouch settings Character footstep bones incorrect instance

eased September 27, 2021 New · Start State to Character component · Latest documentation PDF file · Option to run camera in Fixed Update • Dust FX on examples when character lands · Rigidbody character Driver · Instruction: Set Text · Instruction: Text Join · Instruction: Text Replace · Instruction: Text Substring Enhances · Handling on Character units · Performance on Reflective properties Fixes • Examples and improved their visuals · Physics engine methods being called every frame

eased September 22, 2021 New Tank Controls to characters Copy & Paste to all lists Duplicate button to all lists · Faster method to get managed reference values · Get and Set values from Input devices Get and Set fields using C# Reflection · Get and Set properties using C# Reflection · Event: On Navigation Link Enter · Event: On Navigation Link Exit Condition: Compare Child Count · Instruction: Remap Coordinates Instruction: Uniform Scale a Vector3 value · Instruction: Loop List Fixes Animator null when changing model in Editor · Audio not taking into account time scale Incorrect description on some Input methods Changing kernel units while in play-mode Global variable access in standalone builds

2.0.5

eased September 17, 2021 New IsRunning property to Actions and Conditions Property to search an object by name Memory: Name Memory: Tag Memory: Layers Memory: Is Active Memory: Light Color Memory: Light Intensity Instruction: Change name of Game Object Fixes Point & Click incorrect raycast order Point & Click ignore over UI game objects Memories not drawing some properties Date not parsing using system culture

New NavMeshAgent avoidance quality NavMeshAgent avoidance priority Fixes Material Sound error when texture is null Player not moving without a Main Camera Description of Usage Input buttons

2.0.3



eased September 13, 2021 New Option to create impacts for Material Sounds Model position offset to Character animation Complete & Basic Locomotion States · Instruction: Toggle Active Bool Property: Does not Exist · Bool Property: Is not Active · Input: Usage/Crouch · Input: Usage/Walk Fixes · Invalid Hub URL on Windows machines · Invalid Documentation URL · Skeleton asset error when using 3D models · Stop State instruction layer index · Primary motion input with joystick dead-zone • Foot IK disabled during gestures with root-motion · Look IK alignment with target's line of sight Animation time scale on characters

2.0.1

.eased September 10, 2021 New • First release

2. Inventory

2.1 Inventory

Inventory

Using items, combining them, crafting new ones or trading them with other characters is at the heart of many games.

The **Inventory** module has been meticulously crafted to support a wide variety of situations that involve the use and management of items.

Get Inventory



The ${\bf Inventory}$ module is an extension of ${\bf Game\ Creator\ 2}$ and won't work without it

2.2 Setup

Welcome to getting started with the **Inventory** module. In this section you'll learn how to install this module and get started with the examples which it comes with.

2.2.1 Prepare your Project

Before installing the **Inventory** module, you'll need to either create a new Unity project or open an existing one.



It is important to note that Game Creator should be present before attempting to install any module.

2.2.2 Install the Inventory module

If you haven't purchased the **Inventory** module, head to the Asset Store product page and follow the steps to get a copy of this module.

Once you have purchased it, click on Window Package Manager to reveal a window with all your available assets.

Type in the little search field the name of this package and it will prompt you to download and install the latest stable version. Follow the steps and wait till Unity finishes compiling your project.

2.2.3 Examples

We highly recommend checking the examples that come with the **Inventory** module. To install them, click on the *Game Creator* dropdown from the top toolbar and then the *Install* option.

The **Installer** window will appear and you'll be able to manage all examples and template assets you have in your project.

- Items: Template items ready to be used in your games
- UI: Samples for creating loot user interfaces, inventories, merchants and crafting windows
- Examples: A collection of scenes that will help you understand each and every option of the Inventory module, in an organized and tidy way.

Installer Inventory

The **Examples** requires both the **Items** and **UI** extensions in order to work.

There is also an extra skin for adventure games that allows to swap the default inventory for a typical old-school point and click inventory.



Clicking on the Examples install button will install all dependencies automatically.

Once you have the examples installed, click on the *Select* button or navigate to Plugins/GameCreator/Installs/Inventory.Examples/.

Inventory Examples

2.3 Items

2.3.1 Items

Items are in-game objects that can be added to a Bag, and represent the name and description, properties, visual representation, and other information that allows to craft, trade, use and equip them.

Creating an Item

Items are scriptable objects and to create one, you'll need to right click on the *Project Panel* and navigate to *Create Game Creator Inventory Item*.

Item

An **Item** asset will appear, with a list of sections that can be expanded or collapsed so it is easy for the user to modify and organize your items.

The ID value is a unique text that represents an item. When creating a new asset, it will be completely unique. However, duplicating an existing item will also duplicate the ID and a red message will appear above stating that there are two items with the same ID.

To solve that, expand the field and click on the *Regenerate* button to create a new unique ID. You can also type in a name if you follow a naming convention that ensures that all item IDs are unique.

The **Prefab** field is used to drop/instantiate an item onto the scene. If no prefab is provided, the item will not be instantiated.

INHERITANCE

The Parent field allows an item to inherit values from another item, such as Properties and Sockets.

Rem A equals Item B?

Comparing two items takes into account their parent-child relationship. For example, if Item A inherits from Item B and a Condition is trying to determine if an object is equal to another one:

Item A inherits from Item B

 $\it A$ will always return success when comparing if $\it A$ equals $\it B$ or equals $\it A$.

B will always return success when comparing if B equals B but not to A, because A is further down in the inheritance chain.

An Item will always return success if asked whether it is equal to itself or any of its parent items.

INFORMATION

This section allows to define the Name. Description, Sprite representation and Color of the Item.

Item Information



All these fields use dynamic properties so their values can be localized.

SHAPE

The shape of an **Item** determines the **Width** and **Height** the item occupies in the inventory bag, if it's a grid-based inventory.

It also determines the Weight of the item, in case the bag has a max weight limit.

The Max Stack field determines how many of the exact same item can be stacked one on top of another.

Item Information



If an **Item** has one or more **Sockets**, the **Max Stack** will be automatically restricted to 1, due to technical constraints.

PRICE

An **Item**'s trading value is determined by a Currency asset and a numeric value. This value is the total *pure* one, without any discounts or modifiers applied.

Item Price



Note that an item can only be traded using a single currency.

Sckets

The price of an Item that can have other Items attached is the result of the sum of the price of all Items attached, plus the price of the Item itself.

For example, if the item Sword has a price of 45 gold and a $Magic\ Rune$ costs 20 gold pieces, the value of the Sword with the rune attached will be 65 (45 + 20).

PROPERTIES

Properties define mutable values that an item defines. A Property is a data block that is identified by a name and contains a value and a text that can be used to display information about this item and use it in-game.

Item Properties

tse case of Properties

The most common use-case of a property is definining the attack power of a weapon. One could easily use an item that represents a *Sword* and add a property called attack and has a value of 35.

Item Attack Property

See more information about this in the **Properties** page.

SOCKETS

Sockets allow to attach items onto other items. The type of item that can be attached is determined using item inheritance.

Item Sockets

Attaching Runes

For example, a socket accepts the item Rune, then all items that inherit from the Rune item will be accepted.

See more information about this in the Sockets page.

EOUIPPING

Some items can be equipped by the wearer (usually the Character with the Bag component).

Item Equipment

See more information about this in the Equipping page.

USAGE

This section allows to define the behavior of an utility Item which can be used at any given time.

Ttem Use

A usable item can have a finite or infinite amount of usages. The **Consume on Use** toggle defines whether an item is consumed upon use or not.

Finite vs Infinite usages

For example, a Health Potion is consumed when used. However a Whistle can be used many times.

The **Can Use** conditions are executed every time a runtime item is attempted to be used. If the result is successful, the item is used.

When an **Item** is used, the **On Use** instructions are executed, where **Self** refers to the game object with the *Bag* component the item belongs to, and the **Target** is the references the wearer of the *Bag*.

ecute From Parent

Both the **Can Use** conditions and the **On Use** instructions can optionally execute the parent Item's *Can Use* and *On Use* instructions before executing itself.

This is very useful to avoid repeating the same logic over multiple items. For example, if drinking any potion results in the character executing a particular animation and playing a sound effect, these instructions can be placed in a parent Item called *Potions* so each child Item (Health Potion, Mana Potion, ...) does not have to.

CRAFTING

The **Crafting** section allows to define recipes to create new **Items** as well as dismantle them into multiple ingredients.

Item Crafting

See more information about this in the ${\bf Crafting}\ {\bf page}\,.$

2.3.2 Properties

Properties are mutable values that compose a runtime item. For example, an **Item**'s attack power, its durability or whether they apply a special effect, such as *Burn*.

Item Properties

Creating a new Property

To create a new **Property** all that needs to be done is to click on the Add Property button.

Item Attack Property

The **Property ID** field determines the unique ID of this Property. It is used to identify it, so make sure it's a name that's easy to remember and type.

Is Hidden determines if a Property is hidden in the UI. For more information, see the Hiding Properties section.

The rest of fields are all optional.

- Icon: Provides the Property with a Sprite to be used in user interfaces.
- · Color: Assigns a color to the Property. Useful to differentiate items in user interfaces.
- · Number: A mutable value that can be used in-game, such as increasing stats.
- Text A dynamic value that is usually used to represent the in-game name of the Property.

Mutable vs Immutable

Mutable is a programming concept which means that the value is dynamic and can be changed at runtime. Immutable, in contrast, means that its value can't be changed once a value is assigned.

Inheriting Properties

Checking the **Inherit Properties** toggle found at the top will automatically inherit all properties from its parent(s).

Item Inherit Properties

The value of an inherited **Property** can be overridden by checking its left toggle and changing the field value.

Taking advantage of inheritance

It is very common to have a type of item that shares the same properties with all its child items. Setting a base value for the parent item type will make it much easier to define what each sub-item does.

For example, let's say all *shield* items have a defense value. We could add this property on the base item "Shield" and propagate this property to all other shields that inherit from this item, and just change the final value, so a "Wooden Shield" has a lower defense value than a Steel Shield.

Hiding Properties

When displaying properties in the UI, these can be sequentially displayed, without having to manually set them one by one. If the **Is Hidden** checkbox is ticked, these properties will not be displayed in the user interface.

Item UI Properties

Suff behind the scenes

This is specially useful when a property represents something that the user should not be aware of.

For example, some items could have the is-metal property that determines if an item is a metallic one or not.

2.3.3 Sockets

Sockets allow to attach items onto other items. For example, a Sword can have a socket that allows to attach a Rune so it increases its properties.

Item Sockets



Ticking the Inherit from Parent checkbox will instruct the Item to inherit all Sockets from its parent(s).

The socket section is divided in two parts: The part that defines the object attached to the socket, and the part that accepts attachments.

Objects attached to Sockets

The **Socket Prefab** field accepts a prefab game object, which is instantiated when attaching this **Item** onto another Item's **Socket**.

Item Sockets Prefab instance

To configure where the prefab is instantiated, the scene prefab object must have a **Prop** component. This component automatically updates and correctly instantiates the attachment prefabs in the right places, defined in the component's Editor.

Item Sockets Prop component

In this case, the *Metal Shield* has a **Prop** component that inserts the instance of a prefab of any attached rune at the center of the socket.

Configuration of Sockets

To add a ${\bf Socket}$ to an item, simply click on the ${\it Add}$ ${\it Socket}$ button.

Item Sockets new Socket

A **Socket** is defined by a **Base** Item that determines which types of objects can be attached to, and a **Socket ID**, which is used by the *Prop* component.



It is important to note that the **Base** item determines the type of item that the Sockets accepts, not the specific item. In the example above, it accepts a *Rune* item, but will also accept any item that has a *Rune* item parent, such as the *Rune* of *Attack* and *Rune* of *Defense* included in the examples.

How Properties affect Sockets

When attaching an Item onto another one's Socket, only their shared Properties are added.

Sword with a Rune of Attack

Let's imagine we have a **Sword** with a single *Property*

attack = 10

And a Rune with the following Properties:

attack = 5

defense = 5

Attaching the Rune to the Sword results in the latter have an $\frac{1}{2}$ attack value of 15 (10 + 5), but will ignore the defense Property because it is not present in the Sword.

2.3.4 Equipping

To define an equippable Item, the **Is Equippable** checkbox must be ticked, which enables the rest of the options.

Item Equipment

When attempting to equip an Item, the Conditions Can Equip will first be checked.

If it succeeds, it will instantiate the prefab and execute the **On Equip** instruction list. The **Prefab** field is the game object prefab instantated when equipping this particular Item.



Attempting to equip an **Item** on a slot that is already filled by another **Item** will automatically unequip the current one so the new **Item** can be equipped.

When unequipping an Item it will execute the On Unequip instruction list.



To know more about how to define which **Equipment** slots are available for a character, see **Equipment** in the Bag section.

When executing the Can Equip conditions and the On Equip and On Unequip instructions:

- The Self property references the game object that contains the Item being equipped/unequipped.
- The Target references the wearer of the Bag (which usually is the same as the Bag object itself).

It is important to note that when a currently equipped item changes the value of one of its Sockets, it will first unequip it, change the **Socket** value and equip it again.

ecute From Parent

If the **Execute From Parent** checkbox is marked, the instructions and conditions from the item's parent item will be executed first (and its parent too, if the parent has *Execute From Parent* marked).

This is very useful to avoid repeating the same logic over multiple items. For example, if the parent type *Swords* contains a **Property** called attack and all sub-items from Swords have different attack values, there is no need for all sword sub-items to add a **Stat Modifier** with that property.

Instead, the Swords item can execute the common logic between all swords, and each sub-item just needs to have the Execute From Parent checkbox enabled.

2.3.5 Crafting

The **Crafting** section both defines a way to craft the **Item** being examined, as well as tear it apart and dismantle it into multiple **Items**.

Item Crafting

There are 3 distinct sections inside the Crafting tab.

Ingredients

Ingredients are Items that can be used to craft the current one, or dismantle it into these ingredients.

To create a new Ingredient click on the Add Ingredient.

Item Crafting Ingredients

This will create a new ingredient entry with an Item field and the amount of those necessary.

Afinite ingredients

There is no limit to the amount of **Ingredients** you can create.

Craft

When attempting to craft an **Item** it will first check if the **Conditions** are sufficient. If so, it will then require a certain amount of **Ingredients** defined.

If there are enough ingredients, these will be subtracted from the Bag.

the conditions

Leaving the **Conditions** field empty will always return success and means there are no conditions to craft it, outside from the **Igredients**.

Once the **Conditions** and **Ingredients** requirements are fulfilled, it will create a new instance of the **Item** and add it to the Bag.

Afterwards, it will call the **Instructions**, in case the designer wants to do something afterwards, such as increasing the proficiency of the Player in crafting.

Dismantle

Dismantling an **Item** is the inverse process of **Crafting**: Instead of creating the current **Item** from a collection of **Ingredients**, it destroys the **Item** and reclaim the **Ingredients**.

Reclaim Probability

When **Dismantling** an **Item** there is a *Reclaim Chance* value that determines the chance to recover each of the **Ingredients**. A value of 1 will always recover all ingredients, while a value of 0.5 will only have a chance to recover around 50% of them.

2.4 Bags

2.4.1 Bags

A Bag is a component that can be attached to any game object, and contains Items and Currencies.

Bac

The **Inventory** module comes with 2 types of **Bags**:

- · List: Sequentially displays the items one after the other and all occupy the same amount of space.
- Grid: Each item occupies a certain amount of cells and these can be manually arranged inside the inventory
 grid-view.



We recommend sticking with the **List** type, as it is easier to understand and manage. **Grid** inventory systems should be only used by experienced users.

To change the type of Bag click on the right-side arrow button and choose the type from the dropdown menu.

Bag Options

A Bag can define a Maximum Weight and a Maximum Height.

- If a maximum height is defined, there is a maximum amount of Items it can hold.
- If a maximum weight is defined, if the sum of all **Item**'s weight exceeds the maximum value, the **Bag** is considered overloaded.



It is important to note that a **Bag** can't exceed a maximum amount of height (if any is defined). However, a **Bag** will still accept new **Items** even if its content weight exceeds the maximum weight defined.

Equipment

The **Equipment** field is an optional value that accepts an **Equipment Asset**. If provided, it allows the wearer of the **Bag** to equip **Items**.

To know more about how to configure it, see the Equipment section.

Stock and Wealth

Some **Bags** may contain a certain amount of **Items** and **Currency** by default. For example, a Merchant may have some default stock available.

Bag Stock and Wealth

- · Clicking on the Add Stock button creates a new Stock option that accepts an Item and a certain amount of it.
- ullet Clicking on the $\emph{Add Wealth}$ button creates a new ullet option that accepts a ullet currency and its value.

Random Loot

A Bag can also be used as a Chest where the player loots its contents. To generate random loot, we recommend using Loot Tables, instead of Stock options.

Skin UI

The **Skin UI** field is a UI skin asset that displays a different type of user interface that depends on what the purpose of the Bag is. For example, a **Bag** attached to the Player character could display an Inventory UI, while a Chest displays a UI with its content and a button to transfer all of them to the Player's bag.



To know more about designing custom skins, see the User Interface section.

Wearer

The **Wearer** selector refers to the targeted game object that wears the **Bag**'s equipment. By default it is set to Self because the **Bag** is usually attached along the **Character** component. However, if for some reason that is not the case, you can choose which character should be targeted as the equipment wearer.

2.4.2 Equipment

The **Equipment** asset is a scriptable object that lives in the *Project Panel* which contains information about the amount of equippable slots and what bone matches each one of them.

The Equipment Asset

To create an **Equipment** asset, right click on the *Project Panel* and select Create Game Creator Inventory Equipment.

Equipment

An **Equipment** initially has no equipment. Click on the *Add Equipment Slot* button to add a new slot.

Equipment Slot

An equipment slot has a Base Item and a Bone reference.

- The **Base Item** is the type of **Item** it accepts. For example, if all *Helmets* inherit from a *Head* item, using the *Head* template item will allow to equip all helmets in this slot.
- The **Bone** is a reference to the chosen skeletal bone. If the targeted character is a *Humanoid*, the bone can be picked from a dropdown list. If the character is a non-humanoid, the bone must be referenced using its hierarchy path.

Using the Equipment

Once the **Equipment** asset is created, this can be linked to a **Bag** component so the character knows which equipment slots it has available and where each is mapped to which bone.

Example

For example, the equipment that comes with the **Inventory** module has 4 equippable slots (head, body, right and left hand), plus three extra slots for consumable items:

Equipment Example

We can assign this Equipment asset to a Bag and all available slots will appear below.

Equipment Example to Bag

After assigning an **Equipment** asset to a **Bag**, the bone that is linked to each slot can be overridden. This is specially useful for non-humanoids, where their bone hierarchy names might not match.

2.4.3 Loot Tables

Loot Tables are probablility sheets that when executed, pick an option from its entries based on a weighted chance and send the chosen element (if any at all) to a **Bag** component.

To create one, right click on the Project Panel and select Create Game Creator Inventory Loot Table.

Loot Table

To add a new loot entry, click on the Add Loot button. A new entry will appear with the following options:

- Rate: A number that represents the weight of the chance. The higher the value, the greater the chance.
- Loot: A dropdown that allows to pick an Item or a Currency.
- · Amount: The amount picked if the entry is chosen. It can either be a constant value or a random one.



It is important to note the distinction between a Rate (or weight) and a probability percentage.

The **Rate** depends on the total sum of all rates from all entries. For example, two entries with a **Rate** of 1 is equal to two entries with a **Rate** of 5. In both cases, the chance of picking them is 50%.

Optionally there is a No Drop Rate field that enables the Loot Table to pick nothing.

To execute a **Loot Table** it is as easy as using the **Loot Table** instruction and choosing both a **Loot Table** asset and the targeted **Bag** where the items/currency will be sent to.

Loot Table Instruction

Run multiple times

Note that each time a **Loot Table** is executed, it picks one entry from the table. A **Loot Table** can be used multiple times in sequence to fill, for example, a Chest with multiple items.

Chest with Random Loot

One easy way to randomize the loot of a level is to populate them with a Chest prefab that has an **On Start** Trigger. This Trigger then runs one or more times a **Loot Table** and sends its contents to the Chest's **Bag** component.

This allows to very easily populate all the Chests of a level with different content, while at the same time controlling the kind of content they contain.

2.5 Currencies

To determine the value of an ${\bf Item}$, ${\bf Game}$ Creator uses the concept of ${\bf Currency}$.

A **Currency** is an asset that contains one or more **Coins**. Each **Coin** has a value relative to a single unit. To create one, right click on the *Project Panel* and select Create Game Creator Inventory Currency.



Most games make use of a single **Currency**. However, some mobile games and hard-core resource management games use multiple ones.

Currency

In the example above, the **Currency** just has a single **Coin** called **Gold** which value is 1. This is the most simple currency one can create and it's the most commonly used in most games.



It is important to note that a currency cannot have a decimal value. If you wish to represent a value with 2 decimals, one can multiple the value x100 and then shift the comma two units left.

However, some games make use of a multi-coin Currency where each coin represents a different value.

copper, Silver and Gold

Let's say we are making a game where the currency has three different coins, each with a different value:

- A Copper coin is the smallest one.
- A Silver coin is equal to 25 of Copper coins.
- A Gold coin is equal to 5 Silver coins.

In that case, we would create a ${\bf Currency}$ asset with three coins:

Copper: Is the smallest possible value, so it has a value of 1.

Silver: Is equal to 25 copper coins, so it has a value of 25.

Gold: Is equal to 5 silver coins, which cost 25 copper coins each, so it has a value of 125.

Currency In-Game

It is important to note that when adding or subtracting a value of a particular **Currency** the value used is relative to the unit. Following the example above, if we want to give one *Gold Coin* to the Player, we simply increase its wealth by **125**.

2.6 Merchants

The **Inventory** module comes with a built-in system that allows two **Bags** to trade their contents in exchange for a specified Currency.

Merchant

2.6.1 Merchant Component

To initiate a trade between two **Bags**, one of them (the merchant) must have a **Merchant** component attached along a **Bag** component.

- The Bag component provides the stock of items available.
- The Merchant component determines the type of transactions made.

Merchant Component

Merchant Info

The Merchant Info section allows to give the Merchant a name and a description. This is completely optional, but can be useful to display the type of trading made by a certain Merchant.

Example

For example, having a merchant called *Herbologist* already gives a clue of the type of **Items** this merchant trades with

Configuration

- Infinite Currency: If checked, the Merchant will have an infinite amount of currency supply to buy Items from the client (Player). Otherwise it will use the Bag's wealth.
- Infinite Stock: If checked, the number of available Items will not decrease after the client (Player) purchases them. Otherwise, the available stock decreases with each purchase made.
- Allow Buy Back: If checked, every Item sold by the client (Player) is automatically added to the Merchant's stock. Otherwise, any Item sold cannot be recovered.
- Sell Niche Type: If checked, it allows to filter the type of Items sold by this merchant, regardless of its Bag content. For example, if a Merchant only sells *Herbs*, even if its Bag contains a Sword, it will not be available for sale.

The **Buy Rate** is the discount coefficient that the Merchant provides when buying Items from the client (Player). A value of 1 indicates the Items sold have no discount. To provide a 90% discount on all Items, this field should be set to 0.9.

The **Sell Rate** is the coefficient applied when the Merchant purchases Items from the client (Player). In most games, the selling price of an Item is lower (commonly half the price) than its real one.

The Bag field is a reference to the Bag component from where the Merchant takes its stock.



If your Bag is placed along another game object, you can change the value of this field from Self to Bag and manually reference the correct object.

 ${\bf Skin}\ {\bf UI}$ is the user interface skin used by this merchant.

2.7 Tinkering

Tinkering

The process of transforming items into other ones is called **Tinkering**, which includes:

- · Crafting: Creating a single item from multiple ones.
- ${\boldsymbol{\cdot}}$ ${\boldsymbol{Dismantling}}\colon$ Destroying an item in order to recover multiple ones.

To open a Crafting or Dismantle interface, use the Open Tinker UI instruction.

Open Tinkering UI

This instruction uses a **Tinker Skin** that determines whether the UI crafts new items or dismantles existing ones.

The **Input Bag** and **Output Bag** are the bags used by the tinker process. In most games, both bag references will match, but there might be some cases where the game outputs the new items onto another bag, from where the player can pick them.

The Filter Item field determines the type of items displayed.

Filtering by Type

Blacksmithing and brewing potions use the exact same process. The only difference between an Alchemy station and a Forge is that the first one filters the types of items to craft by *Potion* type and the latter filters by *Equipment* type.

To know more about how to create your own custom tinkering UI elements, see the Tinker UI section and the examples that come with the **Inventory** module.

2.8 Visual Scripting

2.8.1 Visual Scripting

The **Inventory** module symbiotically works with **Game Creator** and the rest of its modules using its visual scripting tools.

- · Instructions
- · Conditions
- Events

Each scripting node allows other modules to use any **Inventory** feature, and adds a list of **Properties** ready to be used by other interactive elements.

2.8.2 Conditions

Conditions

SUB CATEGORIES

• Inventory

Inventory

INVENTORY

Sub Categories

- Bags
- Cooldowns
- Equipment
- Merchant
- Properties
- Tinker
- Ui
- Wealth

Conditions

- · Can Add
- Has Item
- Has Runtime Item
- Is Overloaded
- Is Type Of Item
- Is Usable

CAN ADD

Inventory » Can Add

Description

Returns true if the item can be added to the $\ensuremath{\mathsf{Bag}}$ component

Parameters

Name	Description
Item	The item type to add
To Bag	The target destination Bag

Keywords

Inventory Give Put Set

HAS ITEM

Inventory » Has Item

Description

Returns true if the Bag component contains, at least, the specified amount of an item

Parameters

Name	Description
Item	The item type to check
Amount	The minimum amount of a particular item
Bag	The targeted Bag

Keywords

Inventory Contains Includes Wears Amount

HAS RUNTIME ITEM

Inventory » Has Runtime Item

Description

Returns true if the Bag component contains the Item instance $% \left(1\right) =\left(1\right) \left(1\right)$

Parameters

Name	Description
Runtime Item	The item instance to check
Bag	The targeted Bag

Keywords

Inventory Contains Includes Wears

IS OVERLOADED

Inventory » Is Overloaded

Description

Returns true if the Bag's maximum weight is surpassed

Parameters

Name Description
Bag The Bag component

Keywords

Inventory Weight Amount

IS TYPE OF ITEM

Inventory » Is Type of Item

Description

Returns true if the item is equal or a sub-type of another one $% \left(1\right) =\left(1\right) \left(1$

Parameters

Name	Description
Item	The item source
Compare To	The item compared to

Keywords

Inventory Compare

IS USABLE

Inventory » Is Usable

Description

Returns true if the chosen Item can be used

Parameters

Name Description

Item The item type to check

Keywords

Inventory Consume Drink

BAGS

Bags

Conditions

• Enough Space

Enough Space

Inventory » Bags » Enough Space

Description

Returns true if the item can be added to the $\ensuremath{\mathsf{Bag}}$ component

Parameters

Name	Description
Bag	The Bag to check
Min Space	The minimum amount of free spaces

Keywords

Inventory Has Free Available Full Empty

COOLDOWNS

Cooldowns

Conditions

- Is Item Cooldown
- Is Runtime Item Cooldown

Is Item Cooldown

Inventory » Cooldowns » Is Item Cooldown

Description

Returns true if the Bag's Item is currently on a cooldown state

Parameters

Name	Description
Bag	The Bag targeted
Item	The Item that checks its cooldown state

Keywords

Bag Cooldown Timer Timeout

Is Runtime Item Cooldown

Inventory » Cooldowns » Is Runtime Item Cooldown

Description

Returns true if the Bag's Runtime Item is currently on a cooldown state

Parameters

Name Description

Keywords

Bag Cooldown Timer Timeout

EQUIPMENT

Equipment

Conditions

- Can Equip
- Is Equipment Slot Free
- Is Equippable
- Is Equipped
- Is Runtime Item Equipped

Can Equip

Inventory » Equipment » Can Equip

Description

Returns true if the chosen Item can be equipped by the targeted Bag's wearer

Parameters

Name	Description
Item	The item type to check
Bag	The targeted Bag

Keywords

Inventory Contains Includes Wears Amount

Is Equipment Slot Free

Inventory » Equipment » Is Equipment Slot Free

Description

Returns true if the Bag's equipment slot does not have any Item assigned

Parameters

Name	Description
Bag	The targeted Bag component
Equipment Slot	The Equipment slot to check

Keywords

Inventory Wears Slot Hotbar

Is Equippable

Inventory » Equipment » Is Equippable

Description

Returns true if the chosen Item can be equipped

Parameters

Name	Description
Item	The item type to check

Keywords

Inventory Wear Equip

Is Equipped

Inventory » Equipment » Is Equipped

Description

Returns true if the Bag's wearer has an Item of that type currently equipped

Parameters

Name	Description
Item	The item type to check
Bag	The targeted Bag

Keywords

Inventory Wears

Is Runtime Item Equipped

Inventory » Equipment » Is Runtime Item Equipped

Description

Returns true if the Bag's wearer has the Runtime Item currently equipped

Parameters

Name Description

Keywords

Inventory Wears

MERCHANT

Merchant

Conditions

- · Can Buy
- Can Sell

Can Buy

Inventory » Merchant » Can Buy

Description

Returns true if the item can be bought from a Merchant

Parameters

Name	Description
From Merchant	The Merchant component
Item	The item type attempted to purchase
To Bag	The destination Bag for the item

Keywords

Inventory Purchase Get Bargain Haggle

Can Sell

Inventory » Merchant » Can Sell

Description

Returns true if the item can be sold to a Merchant

Parameters

Name	Description
From Bag	The Bag where the item is sold
Item	The item type attempted to sell
To Merchant	The Merchant target

Keywords

Inventory Vend Trade Exchange Part Bargain Haggle

PROPERTIES

Properties

Conditions

- Item Has Property
- Runtime Item Has Property

Item has Property

Inventory » Properties » Item has Property

Description

Returns true if the chosen Item has the specified item Property

Parameters

Name	Description
Item	The item type to check
Property	The item property

Keywords

Inventory Contains Exists

Runtime Item has Property

Inventory » Properties » Runtime Item has Property

Description

Returns true if the chosen Runtime Item has the specified item Property

Parameters

Name	Description
Runtime Item	The Runtime Item type to check
Property ID	The item property ID to check

Keywords

Inventory Contains Exists

TINKER

Tinker

Conditions

- · Can Craft
- Can Dismantle
- Enough Ingredients
- Is Craftable
- Is Dismantable

Can Craft

Inventory » Tinker » Can Craft

Description

Returns true if the item can be crafted

Parameters

Name	Description
From Bag	The Bag where ingredients are picked
Item	The item type attempted to craft
To Bag	The target destination Bag after creating the new Item

Keywords

Inventory Create Make Cook Smith Combine Assemble

Can Dismantle

Inventory » Tinker » Can Dismantle

Description

Returns true if the item can be dismantled

Parameters

Name	Description
From Bag	The Bag where item is picked
Item	The item type attempted to dismantle
To Bag	The destination Bag for all ingredients after dismantling the Item

Keywords

Inventory Apart Disassemble Deconstruct Tear Separate

Enough Ingredients

Inventory » Tinker » Enough Ingredients

Description

Returns true if the item can be crafted

Parameters

Name	Description
From Bag	The Bag where ingredients are picked
Item	The item type attempted to craft

Keywords

Inventory Create Make Cook Smith Combine Assemble

Is Craftable

Inventory » Tinker » Is Craftable

Description

Returns true if the chosen Item can be crafted

Parameters

Name	Description
Item	The item type to check

Keywords

Inventory Create Forge Alchemy Brew

Is Dismantable

Inventory » Tinker » Is Dismantable

Description

Returns true if the chosen Item can be dismantled

Parameters

Name	Description
Item	The item type to check

Keywords

Inventory Destroy Tear Break

UI

U

Conditions

- Is Bag Ui Open
- Is Merchant Ui Open
- Is Tab Ui Active
- Is Tinker Ui Open

Is Bag UI Open

Inventory » UI » Is Bag UI Open

Description

Returns true if the there is a Bag UI open $\,$

Keywords

Inventory Close Stash Loot Container Chest

Is Merchant UI Open

Inventory » UI » Is Merchant UI Open

Description

Returns true if the there is a Merchant UI open

Keywords

Shop Exchange Trader

Is Tab UI Active

Inventory » UI » Is Tab UI Active

Description

Returns true if the chosen Tab UI component is currently active

Keywords

Shop Exchange Trader

Is Tinker UI Open

Inventory » UI » Is Tinker UI Open

Description

Returns true if the there is a Crafting/Dismantling UI open

Keywords

Close Craft Dismantle Assemble Disassemble Smith Upgrade

WEALTH

Wealth

Conditions

· Compare Wealth

Compare Wealth

Inventory » Wealth » Compare Wealth

Description

Returns true if a comparison between the wealth and another integer is satisfied

Parameters

Name	Description
Bag	The Bag component with the Wealth being compared
Currency	The currency type to compare
Comparison	The comparison operation performed between both values
Compare To	The integer value that is compared against

Keywords

Price Money Cash Currency Coin Gold

2.8.3 Events

Events

SUB CATEGORIES

• Inventory

Inventory

INVENTORY

Sub Categories

- Currency
- Equipment
- Merchant
- Sockets
- Tinker
- Ui

Events

- On Add
- On Drop Item
- On Instantiate Item
- On Remove

ON ADD

Inventory » On Add

Description

Executes after adding an item to the specified $\ensuremath{\mathsf{Bag}}$

Keywords

Bag Inventory Item Add

ON DROP ITEM

Inventory » On Drop Item

Description

Detects when a Bag's item is dropped onto the Trigger

ON INSTANTIATE ITEM

Inventory » On Instantiate Item

Description

Executes after dropping an item from a Bag to the scene

ON REMOVE

Inventory » On Remove

Description

Executes after removing an item from the specified $\ensuremath{\mathsf{Bag}}$

Keywords

Bag Inventory Item Take

CURRENCY

Currency

Events

• On Change Currency

On Change Currency

Inventory » Currency » On Change Currency

Description

Detects when a Bag's Currency value changes

EQUIPMENT

Equipment

Events

- On Equip
- On Unequip

On Equip

Inventory » Equipment » On Equip

Description

Executes after equipping an item from the specified Bag

Keywords

Bag Inventory Item Add Wear

On Unequip

Inventory » Equipment » On Unequip

Description

Executes after unequipping an item from the specified Bag

Keywords

Bag Inventory Item Remove Wear

MERCHANT

Merchant

Events

- On Buy
- On Sell

On Buy

Inventory » Merchant » On Buy

Description

Executes after successfully purchasing an item from any Merchant

On Sell

Inventory » Merchant » On Sell

Description

Executes after successfully selling an item to any Merchant

SOCKETS

Sockets

Events

- On Socket Attach
- On Socket Detach

On Socket Attach

Inventory » Sockets » On Socket Attach

Description

Detects when an Item's Socket gets another Item attached

On Socket Detach

Inventory » Sockets » On Socket Detach

Description

Detects when an Item is detached from another Item's Socket

TINKER

Tinker

Events

- On Craft
- On Dismantle

On Craft

Inventory » Tinker » On Craft

Description

Executes right after successfully crafting any item

On Dismantle

Inventory » Tinker » On Dismantle

Description

Executes right after successfully dismantling any item

UI

Ui

Events

- On Close Bag Ui
- On Close Merchant Ui
- On Close Tinker Ui
- On Open Bag Ui
- On Open Merchant Ui
- On Open Tinker Ui

On Close Bag UI

Inventory » UI » On Close Bag UI

Description

Detects when a Bag UI is closed

On Close Merchant UI

Inventory » UI » On Close Merchant UI

Description

Detects when a Merchant UI is closed

On Close Tinker UI

Inventory » UI » On Close Tinker UI

Description

Detects when a Tinker UI is closed

On Open Bag UI

Inventory » UI » On Open Bag UI

Description

Detects when a Bag UI is opened

On Open Merchant UI

Inventory » UI » On Open Merchant UI

Description

Detects when a Merchant UI is opened

On Open Tinker UI

Inventory » UI » On Open Tinker UI

Description

Detects when a Tinker UI is opened

2.8.4 Instructions

Instructions

SUB CATEGORIES

• Inventory

Inventory

INVENTORY

Sub Categories

- Bags
- Cooldowns
- Currency
- Equipment
- Loot
- Sockets
- Ui
- Variables

BAGS

Bags

Instructions

- Add Item
- Add Runtime Item
- Drop Item
- Drop Runtime Item
- Increment Bag Height
- Increment Bag Width
- Move Content To Bag
- Move Wealth To Bag
- Remove Item
- Remove Runtime Item

Add Item

Inventory » Bags » Add Item

Description

Creates a new item and adds it to the specified Bag

Parameters

Name	Description
Item	The type of item created
Bag	The targeted Bag component

Keywords

Bag Inventory Container Stash Give Take Borrow Lend Buy Purchase Sell Steal Rob

Add Runtime Item

Inventory » Bags » Add Runtime Item

Description

Adds an existing instance of an Item and adds it to the specified Bag

Parameters

Name	Description
Runtime Item	The existing Item instance
Bag	The targeted Bag component

Keywords

Bag Inventory Container Stash Give Take Borrow Lend Buy Purchase Sell Steal Rob

Drop Item

Inventory » Bags » Drop Item

Description

Drops an Item type from a Bag onto the scene

Parameters

Name	Description
Item	The type of item created
Bag	The targeted Bag component
Distance	The distance from the Bag where the Item is dropped

Keywords

Leave Eliminate Take

Drop Runtime Item

Inventory » Bags » Drop Runtime Item

Description

Drops a Runtime Item from its Bag onto the scene

Parameters

Name	Description
Runtime Item	The instance of an Item dropped
Distance	The distance from the Bag where the Item is dropped

Keywords

Leave Eliminate Take

Increment Bag Height

Inventory » Bags » Increment Bag Height

Description

Increases the amount of rows a Bag has, if possible

Parameters

Name	Description
Bag	The targeted Bag component
Rows	The number of rows to increment by

Keywords

Bag Inventory Container Stash Column Size

Increment Bag Width

Inventory » Bags » Increment Bag Width

Description

Increases the amount of columns a Bag has, if possible

Parameters

Name	Description
Bag	The targeted Bag component
Columns	The number of columns to increment by

Keywords

Bag Inventory Container Stash Column Size

Move Content to Bag

Inventory » Bags » Move Content to Bag

Description

Moves all the contents of a Bag to another Bag

Parameters

Name	Description
From Bag	The Bag component where its contents are removed
To Bag	The targeted Bag component where the contents end up

Keywords

Bag Inventory Container Stash Chest Take All Give Take Borrow Lend Buy Purchase Sell Steal Rob

Move Wealth to Bag

Inventory » Bags » Move Wealth to Bag

Description

Moves all wealth from one Bag to another one

Parameters

Name	Description
From Bag	The Bag component where its wealth is taken from
To Bag	The targeted Bag component where the wealth ends up

Keywords

Bag Inventory Container Stash Chest Take All Give Take Borrow Lend Buy Purchase Sell Steal Rob Currency Cash Money Coins

Remove Item

Inventory » Bags » Remove Item

Description

Removes an Item from the specified Bag

Parameters

Name	Description
Item	The parent type of item to be removed
Bag	The targeted Bag component

Keywords

Bag Inventory Container Stash Give Take Borrow Lend Buy Purchase Sell Steal Rob

Remove Runtime Item

Inventory » Bags » Remove Runtime Item

Description

Removes an Item instance from its associated Bag

Parameters

Name Description

Keywords

Bag Inventory Container Stash Give Take Borrow Lend Buy Purchase Sell Steal Rob

COOLDOWNS

Cooldowns

Instructions

- Add Item Cooldown
- Add Runtime Item Cooldown
- · Clear Cooldowns
- Reset Item Cooldown
- Reset Runtime Item Cooldown

Add Item Cooldown

Inventory » Cooldowns » Add Item Cooldown

Description

Adds a cooldown timer for a Bag's Item

Parameters

Name	Description
Bag	The Bag where the Item belongs to
Item	The Item asset to add its cooldown

Keywords

Add Runtime Item Cooldown

Inventory » Cooldowns » Add Runtime Item Cooldown

Description

Adds a cooldown timer for a Runtime Item's Bag

Parameters

Name Description

Keywords

Clear Cooldowns

Inventory » Cooldowns » Clear Cooldowns

Description

Removes all cooldowns on a Bag

Parameters

Name	Description	
Bag	The Bag where all cooldowns are removed from	

Keywords

Reset Item Cooldown

Inventory » Cooldowns » Reset Item Cooldown

Description

Removes the cooldown timer of a Bag's Item

Parameters

Name	Description
Bag	The Bag where the Item belongs to
Item	The Item asset to reset its cooldown

Keywords

Reset Runtime Item Cooldown

Inventory » Cooldowns » Reset Runtime Item Cooldown

Description

Removes the cooldown timer of the Runtime Item's Bag

Parameters

Name	Description
Item	The Runtime Item instance to reset its cooldown

Keywords

CURRENCY

Currency

Instructions

· Change Currency

Change Currency

Inventory » Currency » Change Currency

Description

Modifies the value of a Bag's currency

Parameters

Name	Description
Currency	The currency type to modify
Amount	The value and operation performed
Bag	The targeted Bag component

Keywords

Bag Inventory Container Stash Give Take Borrow Lend Buy Purchase Sell Steal Rob Coin Cash Bill Value Money

EQUIPMENT

Equipment

Instructions

- Equip Item
- Equip Runtime Item
- Unequip Item
- Unequip Runtime Item

Equip Item

Inventory » Equipment » Equip Item

Description

Equips an Item from the Bag that inherits from the specified type

Parameters

	Name	Description
	Item	The parent type of item to equip
	Bag	The targeted Bag component
, –		

Keywords

Bag Inventory Equipment Put Wear Inventory Wield

Equip Runtime Item

Inventory » Equipment » Equip Runtime Item

Description

Equips the specified Runtime Item

Parameters

Name Description

Keywords

Bag Inventory Equipment Put Wear Inventory Wield

Unequip Item

Inventory » Equipment » Unequip Item

Description

Unequip an Item from the Bag that inherits from the specified type

Parameters

	Name	Description
	Item	The parent type of item to equip
	Bag	The targeted Bag component
, .		

Keywords

Bag Inventory Equipment Take Sheathe Inventory Remove

Unequip Runtime Item

Inventory » Equipment » Unequip Runtime Item

Description

Unequip an Item instance that is currently equipped

Parameters

Name Description

Keywords

Bag Inventory Equipment Take Sheathe Inventory Remove

LOOT

Loo

Instructions

- Instantiate Item
- Loot Table

Instantiate Item

Inventory » Loot » Instantiate Item

Description

Instantiates the prefab of an item on the scene

Parameters

Name	Description
Item	The type of item created
Location	The position and rotation where the item instance is placed

Keywords

Drop Inventory Instance

Loot Table

Inventory » Loot » Loot Table

Description

Picks a random choice from a Loot Table and sends it to the specified Bag

Parameters

Name	Description
Loot Table	The Loot Table that generates the Item instance
Bag	The targeted Bag component

Keywords

Bag Inventory Container Stash Give Take Borrow Lend Corpse Generate

SOCKETS

Sockets

Instructions

- Attach Runtime Item
- Detach Runtime Item

Attach Runtime Item

Inventory » Sockets » Attach Runtime Item

Description

Attaches a Runtime Item onto the first available Runtime Item socket

Parameters

Name	Description
Runtime Item	The item instance
Attach	The item instance attached to the other runtime item

Keywords

Bag Inventory Sockets Attach Enchant Embed Imbue

Detach Runtime Item

Inventory » Sockets » Detach Runtime Item

Description

Detaches a Runtime Item from another Runtime Item socket

Parameters

Name	Description
Runtime Item	The item instance with an occupied socket
Detach	The item instance to detach from the other runtime item

Keywords

Bag Inventory Sockets Detach Disenchant

UI

U

Instructions

- · Close Bag Ui
- · Close Merchant Ui
- · Close Tinker Ui
- Open Bag Ui
- Open Merchant Ui
- Open Tinker Ui
- Set Bag Ui
- Set Drop Amount
- Set Split Amount
- Set Transfer Amount

Close Bag UI

Inventory » UI » Close Bag UI

Description

Closes the current inventory UI

Keywords

Item Inventory Catalogue Content Sort Equipment Hotbar Consume

Close Merchant UI

Inventory » UI » Close Merchant UI

Description

Closes the current Merchant UI

Keywords

Trade Merchant Shop Buy Sell Junk

Close Tinker UI

Inventory » UI » Close Tinker UI

Description

Closes the current Tinker UI

Keywords

Craft Make Create Dismantle Disassemble Torn Alchemy Blacksmith

Open Bag UI

Inventory » UI » Open Bag UI

Description

Opens an inventory UI of a specific Bag

Parameters

Name	Description
Bag	The Bag component
Wait to Close	If the Instruction waits until the UI closes

Keywords

Item Inventory Catalogue Content Sort Equipment Hotbar Consume

Open Merchant UI

Inventory » UI » Open Merchant UI

Description

Opens a trading window for a specific Merchant

Parameters

Name	Description
Merchant	The currency type to modify
Client Bag	The client's Bag component
Wait to Close	If the Instruction waits until the UI closes

Keywords

Trade Merchant Shop Buy Sell Junk

Open Tinker UI

Inventory » UI » Open Tinker UI

Description

Opens an Tinkering UI for a specific Bag

Parameters

Name	Description
Tinker Skin	The skin that is used to display the UI
Input Bag	The Bag component where items are chosen
Output Bag	The Bag component where new items are placed
Wait to Close	If the Instruction waits until the UI closes

Keywords

Craft Make Create Dismantle Disassemble Torn Alchemy Blacksmith

Set Bag UI

Inventory » UI » Set Bag UI

${\tt Description}$

Changes the targeted Bag of a Bag UI component

Parameters

Name	Description
Bag UI	The Bag UI that changes its target
Bag	The new Bag component

Set Drop Amount

Inventory » UI » Set Drop Amount

Description

Changes whether a Bag drops a single item or the whole stack when dropping them

Parameters

Name	Description
Drop	Whether to drop one, or the whole stack

Keywords

Item Inventory Let Leave Take Place

Set Split Amount

Inventory » UI » Set Split Amount

Description

Changes whether a Bag splits by unstacking a single item or the whole stack is split in half

Parameters

Name	Description
Drop	Whether to split one, or the whole stack in half

Keywords

Item Inventory Stack Unstack Split Divide

Set Transfer Amount

Inventory » UI » Set Transfer Amount

Description

Changes whether a Bag moves a single item or the whole stack when transferring them

Parameters

Name	Description
Transfer	Whether to transfer one, or the whole stack

Keywords

Item Inventory Transfer Move Content Place

VARIABLES

Variables

Instructions

- Set Item
- Set Runtime Item

Set Item

Inventory » Variables » Set Item

Description

Saves an Item type on a Variable

Parameters

Name	Description
Set	The Variable that saves the Item
Item	The type of item saved

Keywords

Save Keep

Set Runtime Item

Inventory » Variables » Set Runtime Item

Description

Saves a Runtime Item on a Variable

Parameters

Name	Description
Set	The Variable that saves the Runtime Item
Runtime Item	The Item instance saved

Keywords

Save Keep

2.9 User Interface

2.9.1 User Interface

The **Inventory** module comes with a large collection of components so you have complete freedom to make your own game UI.



To get started, it is recommended to install the UI examples that come with this module, which include a HUD, a classic inventory, as well as a merchant and crafting/dismantle interfaces.

Skins

Skins are assets that contain a prefab with a specific UI component. There are three types of skins:

- Bag Skins: These skins are linked to Bag components and require a Bag UI component at the root of the prefab.
- Merchant Skins: These skins are linked to Merchant components and require a Merchant UI component at the root of the prefab.
- Tinker Skin: These skins are directly accessed when opening a Craft/Dismantle interface. They require a Tinker UI component at the root of the prefab.

Skins

The Inventory module comes with a lot of components that make it very easy to build a user interface that synchronizes with a Bag, Merchant or Tinkering object. Each component has a very specific use-case that is covered in each relevant sub-section.



Some UI components depend on others that feed information to them. For example, the **Coin UI** component depends on the **Price UI** component, that instantiates and reuses a prefab with a Coin UI component for each currency coin.

2.9.2 Bag UI

The **Bag UI** is the root component for any UI prefab that displays information about a Bag. There are two types of **Bag UI** components, which depend on the type of **Bag** used:

- Bag List UI: Used for list-like Bags
- Bag Grid UI: Used for grid-like Bags



This documentation focuses on Bags with a List-type, as they are most commonly used. The use of a Grid-type requires a deeper understanding on how each UI component works, but the concepts and components used are mostly the same.

Bag List UI

Prefab Cell is a prefab game object with a **Bag Cell UI** component. This component is automatically instantiated and updated by its parent, for each Item in the Bag displayed.

Filter by Parent is an optional Item-type filter. If none is provided, it will display all Items of all types. This is particularly useful when creating tabs or sections.

Content is the parent game object where all prefab cells will be instantiated - One for each Item in the Bag.

Can Drop Outside determines whether an Item can be dragged outside of the UI canvas to drop it into the scene world.

Max Drop Distance determines the maximum distance that an Item can be dropped from the Bag object.

Drop Amount determines whether a dropped object removes the whole stack of objects or just the top-most.



Note that only Items that have a ${\bf Prefab}$ object in their Item definition can be dropped.

Components

There are a few extra components that can synchronize a Bag's information with UI controls, which can either be linked to a Bag, or to the Bag linked to a Bag List/Grid UI component.

CELL UI

This component is automatically set up and refreshed by its ${f Bag\ List\ UI}$ or ${f Bag\ Grid\ UI}$ parent component.

Bag Cell UI

The **Cell Info** section contains an optional collection of UI control fields that can be plugged in order to be updated when the **Item**(s) associated with this inventory cell change.



This component requires a **Graphic** component (either an Image or a Text) in order to receive input events, such as clicks and drags.

The **Merchant Info** field is optional and only useful if the **Bag Cell UI** component is part of a **Merchant UI** component.

The Can Drag toggle determines whether an Item can be dragged and dropped.

On Drop and On Select defines the behavior when this Item cell is dragged and dropped, and when it is focused.

ected Cell UI, Socket UI and Property UI

When a Bag Cell UI is selected, any Selected Cell UI component will be refreshed with the information of the currently selected cell. This allows to display information about a particular cell outside from the cell itself.

In both Bag Cell UI and Selected Cell UI components, one can create a prefab with a Socket UI/Property UI component that displays the current sockets/properties.

EQUIP UI

This component is used for equipping items and assigning consumables to hotbars.

Bag Equip UI

The Bag and Equipment fields determine the targeted Bag and the equipment slot that this refers to.

There are two main sections:

- · Base UI: Allows to display a collection of optional controls that reference the base-type Item
- Equipped UI: Allows to display a collection of optional controls that reference the currently equipped Item (if there is one).

The rest of the fields define the behavior when the Bag Equip UI is interacted with.

WEALTH UI

The Bag Wealth UI component is used to display the selected Currency and how much of it the Bag carries.

Bag Wealth UI

This component requires a prefab that represents each coin's **Currency** value, and must contain the **Coin UI** component.

WEIGHT

This component displays the current and max weight of the selected Bag.

Bag Weight UI

2.9.3 Merchant UI

The Merchant UI is a very simple component that acts as a middle-man between two Bag UI components - Allowing both ends to transfer or trade their contents based on a particular set of rules.

Merchant UI

This component has two fields at the top:

- · Merchant Bag UI: A Bag UI component that contains information about the Bag that represents the merchant.
- · Client Bag UI: A Bag UI component that contains information about the Bag that represents the client (usually, the Player).



When a Bag UI component is referenced by a Merchant UI, the Bag UI obtains information about the trading rules, which cascade and can be accessed from the Merchant Info section on a Bag Cell UI component.

There are also a couple of Instruction lists at the bottom that are executed when this Merchant UI executes a transaction.



My and Sell

Note that Buy and Sell are from the client's perspective (aka the Player). So the On Buy instructions run when the client purchases an item, and On Sell run when the client sells an item.

2.9.4 Tinker UI

Tinkering involves both **Crafting** and **Dismantling** items, and the **Tinker UI** component allows to display a list of UI controls that handle the transformation.

Tinker UI

There are two distinct sections in this component, but both work very similarly: There is a container object where all available recipes/items are displayed, from where the user can pick one and begin the transformation process.

- Filter By Parent allows to display only those Items that inherit, at some point, from the selected type. If none is set, it will not filter any items.
- Selected UI references a Crafting UI or Dismantling UI component, which is used to display the currently selected Item from the list.

The following two fields allow to populate the list of Items:

- The **Content** field must reference a UI game object which will be populated by an instance of a prefab for each element in the list.
- The Prefab field references a prefab game object, which will be instantiated in the container object.



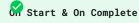
The **Prefab** field requires a **Crafting Item UI** or a **Dismantling Item UI** component in order to work. This will be automatically synchronized and refreshed with the information provided by the Tinker UI list.

Crafting Item UI

The **Crafting Item UI** component is both used when selecting an **Item** from the recipe list as well as to display each entry from the list.

Crafting Item UI

This component is automatically refreshed with the correct information about the current Item.



The **On Start** and **On Complete** instructions are executed when either a dismantle or crafting operation starts, and successfully finishes. This is the perfect place to add sound and visual effects.

Dismantling Item UI

The **Dismantling Item UI** component is both used when selecting an **Item** from the available item list as well as to display each entry from the list.

Dismantling Item UI

This component is automatically refreshed with the correct information about the current Item.

Recover Chance is a value between 0 and 1 that determines the chance to recover each and every one of the ingredients that constitute the dismantled **Item**.

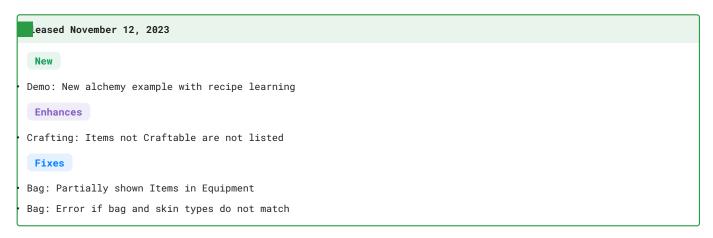
2.10 Releases

2.10.1 Releases

2.8.16 (WIP)

Fixes
Equipment: Missing options overriding slots

2.8.15 (Latest)



2.8.14

Leased October 31, 2023 This version breaks compatibility with previous versions and will only work with Game Creator 2.13.43 or higher. New Grid: New Grid Inventory system Condition: Has Enough Available Space Equipment: Optional use of Handles Property: Last Item Created UI: Option to hide equipped Items Examples: Three new grid inventory demos Changes Items: Instructions with optimized workflows Internal: Support for Core 2.13.42 version Fixes UI: Tooltips showing after destroying UI cell

2.7.13

eased September 1, 2023 New Item: Instructions On Create Item under Info Item: Hide number/text from item Properties Loot Table: Variable type Loot Table · Property: Loot Table instance Property: Loot Table from Variables Settings: Refresh button on Inventory window Enhances · UX: Auto detect when new Items are created UX: Items display parent hierarchy in Inspector Changes · Instruction: Loot Table uses a Property Fixes Item: Refresh ID if no Items available Property: Mixed Last Item Sold / Bought · Remember: Equipment error if no equipment

2.6.12

New Condition: Enough Ingredients to Craft Event: On Change Currency Fixes Trigger: On Drop Item has Bag object as Target Trigger: On Drop Item not working with previous API Property: Get Random Item returns a valid value Property: Get Random Runtime Item returns a valid value Variables: Incorrect initialization phase Examples: Support for core version 2.11.41

2.6.11

eased May 9, 2023 New Items: Can now have Cooldowns after using them Instruction: Add Item/Runtime Item Cooldown Instruction: Remove Item/Runtime Item Cooldown · Instruction: Clear Cooldowns Condition: Is Item/Runtime Item in Cooldown Property: Random Item and Runtime Item from Bag • UI: Option to disable exchanging equipped items UI: Option to split stack by one or in half Enhances Remember: Memorizes time left of Item cooldowns Fixes Instruction: Set Drop Amount incorrect settings · Items: Can Run conditions not running properly · UI: Bag UI component missing field throws error · Align: Equipment index alignement regression Align: Coin index alignement regression

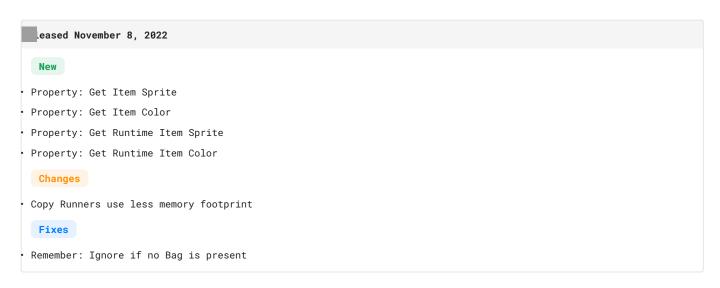
2.6.10

eased March 24, 2023 New · Instruction: Set Transfer Amount Instruction: Set Drop Amount · Condition: Is Tab UI Active · Property: Last Item attempted to Use Property: Last Item attempted to Equip/Unequip Property: Last Item attempted to Craft/Dismantle UI: Allow to split stack of Items · Settings: Displays current and update version Enhances UI: Tab support selection/gamepads Examples: Shortcuts to cycle through UI tabs Fixes • Equipment: Wrong Skinned Meshes bones · Triggers: Bags not detecting the Player • UI: Incorrect buy/sell conditions

2.6.9

Leased December 8, 2022 Enhances Performance when using On Drop Item Exposed Bag UI members for modification Fixes Support for new Props system Null reference when retrieving Item properties

2.6.8



2.5.7

New Instruction: Close Bag UI Instruction: Close Merchant UI Instruction: Close Tinker UI Example: Save and Load inventory Enhances Drag & drop swaps Items instead of shifting Merchant UI Cell: Field to check if cell is valid Fixes Save/Load: Preserves order of Items Grouping Items when stacking deletes source Item Failing to Load Equipment of previously saved game Instruction: Can Increase Width incorrect check Instruction: Can Increase Height incorrect check

2.4.6

New Instruction: Increment Bag Height Instruction: Increment Bag Width Condition: Is Equipment Slot Available Dropping Items use a LayerMask UI Items can be rearranged by default Enhances Loot Table redesign top plot Changes Rearranged Equipment Index class Fixes Detect new Items before enter Play Mode

2.3.5

```
eased June 29, 2022
 New
 Option to uninstall modules
· Condition: Item has Property
· Condition: Runtime Item has Property
· Condition: Is Runtime Item Equipped
 Property: Get Item/Runtime Item Sprite
· Property: Get Item Sprite
· Property: Get Runtime Item counterparts

    Property: Set Runtime Item counterparts

· Property: Get Current Open Bag
· Property: Get Current Merchant Bag
· Property: Get Current Client Bag
 Property: Get Current Tinker Bag
 Example: Storage Chest
 Enhances
· Reorganized Item dropdown
 Reorganized Runtime Item dropdown
 Fixes
Log error when exception in Item instructions
· Wrong Item tinkered when changing UI window
 Condition: Is Item Equipped with sub items
Serialization error during domain reloads
```

2.3.4

eased May 17, 2022 New New Runtime Item properties New Runtime Item Variable type Instruction: Add/Remove Runtime Item · Instruction: Drop Runtime Item · Instruction: Equip/Unequip Runtime Item · Instruction: Attach/Detach to Socket · Condition: Has Runtime Item · Checkbox determines if Item can be sold · Checkbox determines if Item can be bought Checkbox determines if Item can be dropped Enhances · Reorganized Inventory instructions Fixes • Edge case when saving Equipment and Wealth · Retrieving a Bag from a Property · Loot Table displays NaN with no drops · Bag wealth updated at runtime upon change · Selected item would show wrong one

2.2.3

eased March 25, 2022 New · Instruction: Drop Item · Condition: Compare Wealth · Property: Bag Set Wealth • Property: Item Get Property Text · Property: Item Get Property Value · Property: Item Get Property Color · Property: Item Get Property Sprite Property: Item Set Property Text · Property: Item Set Property Value Enhances • Editor: Properties have scene refs · Hide properties from within Item · Support Socketing from external sources Fixes · Fields alignment in Inspector · Missing Price UI editor drawer · Incorrect dropped item in Example scenes

2.1.2

eased January 28, 2022 New · Items have usage conditions Equip/Unequip can inherit logic from its parents · Using Items can inherit logic from its parents · Condition: Can Equip to Bag · Condition: Is Equippable · Condition: Is Equipped · Condition: Is Craftable · Condition: Is Dismantable · Condition: Is Usable · Instruction: Change target Bag of Bag UI • UI: Bag UI can have a default Bag · UI: Properties with a value of 0 can be skipped · Properties: Access to recent socketed Items Changes · Item price increments with socketed Items · Compatibility with Game Creator 2.3.15

2.0.1

Leased January 12, 2022 New First release

3. Dialogue

3.1 Dialogue

Dialogue

Most games allow verbal communication between the player and other characters - Whether that's using barks, cinematic sequences or dialogues where the player is prompted to choose between different choices.

The **Dialogue** module caters all these using simple and intuitive tools that help keep dialogues at a glance while allowing to fully tailor it to the user's needs.

Get Dialogue



The $\bf Dialogue$ module is an extension of $\bf Game\ Creator\ 2$ and won't work without it

3.2 Setup

Welcome to getting started with the **Dialogue** module. In this section you'll learn how to install this module and get started with the examples which it comes with.

3.2.1 Prepare your Project

Before installing the **Dialogue** module, you'll need to either create a new Unity project or open an existing one.



It is important to note that Game Creator should be present before attempting to install any module.

3.2.2 Install the Dialogue module

If you haven't purchased the **Dialogue** module, head to the Asset Store product page and follow the steps to get a copy of this module.

Once you have purchased it, click on Window Package Manager to reveal a window with all your available assets.

Type in the little search field the name of this package and it will prompt you to download and install the latest stable version. Follow the steps and wait till Unity finishes compiling your project.

3.2.3 Examples

We highly recommend checking the examples that come with the **Dialogue** module. To install them, click on the *Game Creator* dropdown from the top toolbar and then the *Install* option.

The **Installer** window will appear and you'll be able to manage all examples and template assets you have in your project.

- Examples: A collection of scenes with different use-case scenarios
- · Skin Default: A minimalist template UI skin for your dialogues
- · Skin Message: A UI skin that shows conversations like SMS/Text messages
- · Skin Pixel: A fantasy UI skin that displays floating conversations
- · Skin Cyberpunk: A futuristic UI skin with glitches and HUD portraits

Installer Dialogue

The Examples requires all the skins in order to work..



Clicking on the ${\it Examples}$ install button will install all dependencies automatically.

Once you have the examples installed, click on the *Select* button or navigate to Plugins/GameCreator/Installs/Dialogue.Examples/.

Dialogue Examples

3.3 Dialogues

3.3.1 Dialogues

All conversations are written in a **Dialogue** component. To create one, right click on the *Hierarchy Panel* and select *Dialogue* Dialogue.



Alternatively, you can select any existing game object and click on the Add Component button and search for **Dialogue**.

Dialogue Component

This is the basic view of the **Dialogue** component, and it's where all the text is written and configured. However, there are multiple sidebars and windows that can be hidden/shown in order to make it easier to work.

3.3.2 Anatomy of a Dialogue

The **Dialogue** component, fully expanded, has 4 different sections, two of which can be collapsed to increase the amount of space available when these are not needed.

Dialogue Sections

Top Toolbar

Dialogue Toolbar

The top toolbar has two distinct sections.

The buttons on the left allow to add new nodes to the conversation. These nodes can either be:

- A Text element, which is the most common type. It displays a text on screen.
- · A Choice element, which allows to present a choice to the player
- A Random selection element, which is similar to the Choice element, but automatically selects a random value.

Shortcuts

Holding the 'Shift' key while clicking on any of the buttons will perform the opposite operation stated next to the buttons.

For example, clicking on a **Text** node that is set as a **Sibling**, while holding the Shift key, it will create a new node as a **Child** of the current one.

To learn more about the different nodes, head to the Nodes section

This section also allows to select where to create the new element. By default, it will always create it right below the currently selected entry, as a sibling. However, this can be changed to create a new element as a child of the selection.

On the far right there are two toggle buttons.

- The first one with the gear icon, toggles the left sidebar, which is the Settings window.
- The second one with the square, toggles the right sidebar, which is the Inspector window.

Settings

The Settings window allow to configure the general values of the current conversation. There are 3 sections:

- Configuration: Determines the skin used by the Dialogue when displaying the conversation, as well as whether it is affected by the time scale or not.
- Actors: This section is automatically filled when new Actors are added or removed, and allows to link a scene reference with the Actor.
- Editor: This allows to customize how the Editor looks like, in order to have more real estate and work more comfortable. These options have no impact on gameplay.

Dialogue Settings

Conversations

This section is the most important one, which allows to overview the whole conversation flow at a glance. Each row is a spoken dialogue line, and they are executed from top to bottom, and examining the child nodes first, before jumping to the next sibling.



Because opening and closing the Inspector sidebar is a very common operation, double clicking on any node will open (and focus on the current node) or hide the sidebar.

Inspector

The inspector sidebar allows to set and modify the currently selected node of a Dialogue.

Dialogue Inspector

Starting from the top, any node allows to change its type, which can either be a Text, Choice or Random.



For more information about node types, see the Nodes section.

The Conditions list below determines whether this node can be executed or not.

The **Portrait** field allows to choose where the Actor's portrait is displayed (if any at all). It allows three options:

- ${\boldsymbol{\cdot}}$ ${\boldsymbol{\mathsf{None}}}\colon \ensuremath{\mathsf{No}}$ portrait is displayed. This is the default option.
- $\boldsymbol{\cdot}$ $\boldsymbol{Primary}\colon$ The primary position of all portraits.
- ullet Alternate: An alternate position where to show the portrait, if the skin supports it.

The Actor field allows to reference an Actor asset. If one is provided, it also allows to choose which expression to use for this dialogue line (if any are available).



Note that whenever an **Actor** field is modified, the **Dialogue** component re-scans the whole conversation tree and gathers which Actors are being used, which can be configured in the **Settings** sidebar.

The Text field is probably the most important one, and it defines the text displayed by the dialogue.

There's a button below that reads Add Value..., which allows to create a list of key-value pairs. These values can be used by the text to add dynamic values. For example, displaying the real name of the Player saved in a global variable.

Pre about Dynamic Values

Dynamic values are incredibly powerful. Read more on how to use them at the Dynamic Values section.

The Audio field, as it name implies, allows to use a voice clip while the text is being displayed.

The **Animation** field allows to choose an animation field, which is played on the object linked to the current Actor. If none is provided or the scene reference is empty, the animation is ignored.

Animation Timeline

The **Animation** field is more powerful than regular **Gestures**, as it allows to play instructions at any point of the animation.

Dialogue Animation Timeline

For more information about the animation timeline tool, see the Animation Timeline section.

The **On Start** and **On End** instructions are executed when the text starts to display and disappears, respectively.

The **Duration** field determines how long the text will stay on screen. By default, it waits until the user presses any button to jump to the next line. However, this can be changed with one of the following options:

- · Until Interaction: The default value. Waits until the element is ordered to skip to the next line.
- Timeout: Waits until the specified time has passed.
- · Audio: Waits until the specified Audio Clip finishes playing.
- Animation: Waits until the Animation Clip finishes playing.

Audio or Animation

It's important to note that if **Audio** or **Animation** are selected, but no asset for those values are present, the duration will be zero seconds and will skip immediately to the next text line.

The **Jump** field, by default, indicates the next dialogue line to play is the natural one (child if any, otherwise the next bottom sibling). However, this field can also be changed to jump to any arbitrary point marked with a specific Tag, or even exit the **Dialogue** after the current line is executed.

3.3.3 Nodes

A Dialogue is composed of nodes displayed from top to bottom, and can even be set as children of other nodes.

Dialogue Nodes Layout

Nodes can be dragged and dropped to change their position in the conversation tree. Dragging and dropping onto another node, will convert the dragged one into a child of the targeted.

There are three different node types: Text, Choices and Random nodes.

Text

Text nodes are the most common and used to display conversations. They display a text message on screen and simply jump to the next node when they are finished.

Dialogue Nodes Text



It's important to note that a **Text** node can contain children nodes. These will be executed if, and only if, the parent Text node's conditions are satisfied. This is specially useful if you want to display a conversation only after meeting certain conditions.

Any text can be enhanced with **rich text** tags, which allow to change the color, size and other properties of specific regions. For example, to display the word James in white in the phrase Hello James, you can surround the specified word between <color> tag:

Hello <color=#FFFFFF>James</color>



Read the official Unity documentation on Rich Text

Choices

Choice nodes prompt the user with a collection of choices. How these choices are presented and their behavior is configured below with a new set of fields that appear.



Since version 2.2.8 Choice options are configured in the Dialogue Skin by default. However, you can change the dropdown option from From Skin to From Node and a list of options that override those from the Skin will appear.

The available choices are picked from the direct children of the **Choices** node, which should usually be **Text** nodes.

Dialogue Nodes Choices

- **Hide Unavailable**: Determines whether unavailable choices (their Conditions return false) should be displayed (but greyed out) or hide them completely.
- · Hide Visited: Determines whether the choice is skipped if the line has already been visited.
- · Skip Choice: Allows to skip the execution of the Text choice selected, and skip to the next immediate one.
- · Shuffle Choices: When ticked, the choices order will be shuffled and displayed randomly.
- Timed Choice: Determines if the choice has a time limit. If checked, two new fields wil appear down below.
- Duration: Specifies the amount of time the user has to pick a choice, in seconds.
- **Timeout**: Defines what happens if the user fails to input a choice, which can either be picking one at random, the first option or the last one (both prior to shuffling, if enabled)



Choosing **Skip Choice** allows the player to not speak the dialogue line when picking it from the prompt. For example, let's say a bartender asks the player whether they want a drink. The Player could see the option "Yes, a Moonlight Specter". If left unchecked, the Player would then execute the **Text** node. Some games, however, don't repeat the choice made by the user and assume the player already said it when the user picked the choice.

Angle Choice

If the **Choices** does only have a single choice available, it will be automatically selected without requiring the user to choose it.

Random

Random picks are similar to **Choices**, except for the fact that the user is not prompted to pick them, and instead, they are randomly picked.

Dialogue Nodes Random



Since version 2.2.8 Random options are configured in the Dialogue Skin by default. However, you can change the dropdown option from From Skin to From Node and a list of options that override those from the Skin will appear.

The Random node also has the field Allow Repeat which determines whether the same choice can be picked in a row. or not.



Random choices are useful to allow characters to pick a random line from a collection. For example, a shop keeper could greet the player differently every time they talk.

3.3.4 Dynamic Values

There are times where a dialogue text must contain some sort of variable value. For example, displaying the player's name that has previously been prompted.

Dynamic Values allow to replace special symbols on the text with values that come from more dynamic sources, such as *Local Variables*, *Stats*, etc...

There are two types of dynamic values: Local and Global ones.

Local Dynamic Values

Local dynamic values are set up inside the Dialogue component, right under the Text field of a node.

Dialogue Local Dynamic Values

Each value is assigned an index value, starting from θ at the top. Using the index number between curly braces $\{$ and $\}$.

Player Name

In the screenshot above, the text {0} will be replaced by the **Global Variable** value called name.

A **Local** dynamic value can also have a specific **color** assigned to it, appear in **bold** and/or in **italic** characters.

Global Dynamic Values

Global dynamic values are very similar to the local ones, but their scope is project-wide, so they only need to be set up once. In the previous example, in order to display the player's name, we'd need to configure a **Local** dynamic value for each dialogue line that displays the player's name.

Instead, it's much more efficient to define a global value that any Dialogue can make use of.

To edit or create **Global** dynamic values, click on the top toolbar's Game Creator button and navigate to Settings. Click on the **Dialogue** tab and a list of all created values will appear, with a button to add new ones.

Dialogue Global Dynamic Values

Global dynamic values have an extra field called **Key**, which is the unique ID assigned to that particular value.

In order to use a **Global** dynamic value, one must type the **Key** value between brackets. For example, if the key value is "player-name", the symbol that replaces itself with the **Global** dynamic value is {player-name}.

Global dynamic values also allow to specify whether the replaced text should be printed in **Bold**, **Italic** and/or in a specific **color**.

3.3.5 Animation Timeline

The **Animation** field available in every **Dialogue** node is a fully-featured Timeline-like sequencing tool that allows to play, preview and add events at different timestamps of an animation.

Dialogue Animation Timeline

Dragging and dropping an Animation Clip onto the Animation field reveals a sequencing tool below.

The first section is called Configuration and contains all the setup options any other Gesture has.

The second one is the **Sequencing** tool, where the animation clip can be previewed in the scene view if the **Actor** referenced is present in the Editor scene. To disable scrubbing the preview, click on the squared blue button.

The timeline has rhomboid-like shapes called **Markers**, which execute instructions when the animation clip plays that specific point.

Moving Markers

Markers can be dragged and slide around the timeline. Doing so will automatically enter animation preview mode, so it's easier to adjust the exact point where the instructions should be called.

The sequence has two buttons with a $\overline{}$ and a $\overline{}$ at each end.

- · Clicking on the plus icon will place a new Marker on the timeline where the head is.
- The minus button removes the currently selected Marker and any instructions associated with them.

Recuting Instructions

It is very important to note that if the **Dialogue** line stops executing before the animation clips has finished, the **Animation** sequence will be canceled at that point and the rest of **Markers** won't be executed.

If there are some critical events that need to be executed before skipping to the next line, these should be placed inside the **On Start** or **On End** instruction lists of the **Node**.

3.3.6 Tags

Each Dialogue line can be marked with a Tag, which is a unique name that identifies that line from the rest.

This identification can then be used for:

- Jumping to a specific line after a node has been executed (useful for looping conversations).
- Using a Condition to check if a node has been executed or not.

To add a Tag to a node, right click it and select Tag... A pop up window will appear with a text prompt. After giving it a name, click Save and it will display on the right side of the node.

Dialogue Tags



Note that **Tags** should have unique names across the **Dialogue** component and their name can't contain any spaces or non-alphanumeric characters.

Here's an example of a Condition checking if the my-tag Tag has been executed or not.

Dialogue Tag Condition

3.4 Actors

3.4.1 Actors

Actors are assets that represent a character speaking in a **Dialogue** and allow to configure their name, how they speak, appear and writing effects.

Dialogue Actor

Actors Name

The first two fields allow to give the Actor a Name and a Description.

Both fields are optional, but can be used in the **Dialogue** component to automatically display the name of the speaker when a character linked to this actor says something.

Expressions

The **Expressions** list is a collection of states in which a character can be when speaking a line. You can use these to express anger, surprise, confusion or any other mental state when a line is spoken, with their respective animation, sound and visual queues.

More about Expressions

For more information about Expressions, see the Expressions section.

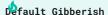
Effects

The **Typewrite** section allows **Dialogue** lines to appear word by word at a certain pace. This is very useful when different characters have different voice cadence and you want to reflect that without using voice-over.

Dialogue Actor

The Frequency field determines how many characters per second appear.

Gibberish is an audio effect played during non-voice acted characters that provide a cheap and easy way to imbue mood into each spoken line. Commonly used in older RPG games, each character plays a random collection of sounds with varying pitch and speed.



The **Dialogue** module comes with a built-in gibberish sound effect ready to be used and customized. Simply select it from the Audio Clip drop down and change the **Pitch** value to fit your needs.

Custom Skin

Most of the time, all characters will use the same speech bubble displayed in the user interface. However, some games require some characters to have a custom speech bubble, like a robot character using a different typography and speech bubble aesthetic.

The Optional Skin field allows a character to override its speech bubble whenever this Actor is used.

About Skins

For more information about how **Skins** work, see the **Skins** section.

The field **Default Portrait** allows to define a default position for this **Actor**'s portrait. This is used by the **Dialogue**'s component, when a node portrait is set to *Actor Default*.

3.4.2 Expressions

Expressions allow to deliver dialogue lines in a specific mood. For example, changing the *Sprite* character that represents the speaker with the associated emotion, show an onomatopoeia, sound effect and/or an animation.

To create a new Expression, click on the Add Expression button.

Dialogue Actor Expression

Lefault Expression

It's important to note that the top most **Expression** is considered the *default* expression, and thus it should be the most commonly used one.

The ID field determines the unique name that identifies this Expression among the rest.

The Sprite is a texture that is used as a portrait when the speaker uses this emotion.

The **Speech Skin** is an optional field that allows to override the speech UI skin used when the **Actor** uses this particular expression. If none is provided, the **Actor** skin is used. And if the **Actor** doesn't have any either, the default one is used.

Larn more about Speech Skins

To know more about what a Speech Skin is and how to use it, visit the UI section.

The **On Start** and **On End** instructions are executed at the very beginning and end of the **Expression**. This can be used to play a **Gesture** or even enter/exit a **State** when using a specific expression.

When instructions are executed

When a new Expression is used, the On Start instructions will be executed. However, the On End instructions won't be called until a new Expression is used, or the Dialogue is finished.

For example, let's say the Player delivers a new dialogue line with the *Angry* expression. If the next line also uses the *Angry* expression, the **On End** instructions won't be called until the Player delivers a new line with a different expression, or the dialogue finishes.

3.5 Skins

3.5.1 Skins

Skins are assets that allow a **Dialogue** to quickly change its looks by swapping them, as well as configure various aspects, such as sound effects and animations.

The Dialogue module has primarily two types of Skins:

- Dialogue Skins: Also known as Theme skins, are the most general ones.
- Speech Skins: They require to be part of a Dialogue Skin and can override the speech bubble of a speaker.

Built-in Skins

The **Dialogue** module comes with a collection of skins that you can use in your games. To install them, click on the toolbar and select Game Creator Install...

Dialogue Skins Install

You'll see a list of Skins to install with a short description next to them. Select the one you want to use (or all of them) and click on the *Install* button.

Dialogue Skins

Dialogue skins change the look and feel of a conversation, and contains all the necessary information to display speech bubbles (through the use of **Speech Skins**), history logs, present choices to the user and show or hide speaker's portraits.

To change a Dialogue's skin, select it and open the Settings sidebar

Dialogue Change Skin

You can drag and drop any available **Dialogue Skins** onto this field and it will automatically use it for this particular conversation component.

A **Dialogue Skin** contains a prefab field, which is the UI schematic with the different components that conform the interface.

creating a custom Dialogue Skin

To learn more about creating a custom skin, see the User Interface section.

Dialogue Skin

The **Animations** section allows to define which UI animations are played when a **Dialogue** component starts, loops and when it ends. These fields are optional and if none a provided, no animation will be played.

The Sound Effects section allows to define which sounds are played at different times.

- Start: Played when a dialogue starts.
- Finish: Played when a dialogue finishes and closes.
- Select: Played when the user hovers or selects any choice.
- Submit: Played when the user submits a choice.

Refault node configuration options

Since version 2.2.8 the Dialogue options are configured globally in each skin.

Dialogue Skin Nodes Options

At the end of the **Dialogue Skin** asset there's another section called **Nodes** where the default options for Text, Choice and Random nodes are configured. Unless a node overrides the values, these will be used.

Speech Skins

A Speech Skin is used by the Dialogue Skin to display a speech bubble by the current speaker.

A **Dialogue Skin** requires to have a default **Speech Skin**. However, this can be overridden by any speaker, assigning a new **Speech Skin** onto its **Actor** asset.

Dialogue Speech Skin

A Speech Skin contains a Prefab object field which defines the UI schema.

creating a custom Speech Skin

To learn more about creating a custom skin, see the User Interface section.

The **Animations** section allows to optionally define which animation clips will be played when a new dialogue line is spoken and a looped animation, if any is needed.

It also allows to configure whether the animation should be played only if a new speaker is delivering the line, or should the animation be played for every new line, even if the same character is delivering two or more of them in a row.

The **Sound Effects** section allows to play a sound effect when a dialogue line starts to be delivered, and when it finishes.

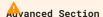
The **Override Log** field is an optional one that allows to customize the log entry (if any available). This is specially useful if you want, for example, the Player to have a different log design than the rest of the characters.

3.6 User Interface

3.6.1 User Interface

Creating custom interfaces for a **Dialogue** is fairly straight forward, although we recommend duplicating an existing one and modifying it in order to make the process easier.

Dialogue custom Skin



Customizing the UI requires certain degree of expertise with Unity and its UI system.

Custom Dialogue Skin

A custom **Dialogue Skin** interface must have, at the top level game object, a **Canvas** component and a **Dialogue UI** component.

The **Dialogue UI** component is te entry point of a conversation and delegates to its the rest of the child dialogue UI components what to do and when to do it.

Dialogue UI component

The Dialogue UI component has two fields:

- · Speech: A Rect Transform reference where the Speech Skin is instantiated.
- · Default: The default Speech Skin to use, if the current Actor speaker doesn't override it.

This is the bare minimum required to create a custom Dialogue Skin.

tional Components

There rest of the components mention below are all completely optional.

The **Dialogue Unit Timer UI** is a component that allows to display a countdown when a choice is presented to the user and has to make a selection before the time runs out.

The **Dialogue Unit Choices UI** is a component that allows to configure where the choices of an interface go and look like.

Moice by Index

Since version 2.1.7 the Dialogue Choice UI component contains a field called Index which references a Text or Text Mesh Pro Text component, which indicates the index of the choice, starting from 1.

You can use the **Choice Index** Instruction to attempt to choose an choice by its index. If a choice is available with that idex, it will automatically be chosen.

Wing Layout components

We recommend using a layout component, such as *Horizontal Layout Group* or a *Vertical Layout Group* in order to automatically align and distribute the choices.

The **Dialogue Unit Logs UI** is a component that collects and stores past lines delivered and choices, so the user can review them.

The Dialogue Unit Portraits UI is a component that displays Sprite of the current speaker, if any at all.

Custom Speech Skin

A Speech Skin UI prefab must contain, at the root of the game object, the Speech UI component.

Speech UI component

The **Active** field references a game object from itself that is set as active/inactive, depending on whether a dialogue text is being delivered.

Similarly, the **Actor Active** field is a optional game object reference that is set as active/inactive, depending on whether the currently delivered line contains an **Actor** reference or not.

Actor Name and Actor Description, as their name implies, reference a Text component which changes into the current Actor's name and description (if any).

The **Active Portrait** field is another optional one that sets the game object as active or inactive, depending on whether there **Actor** asset and chosen **Expression** contains a *Sprite* to be used. If it does, the **Portrait** Image field is used to fill it with the texture value.

The **Text** field is the most important one, and it references a Text component that changes with the text of the current line being delivered.

The **Skip** game object is an optional game object reference that is used to mark the end of a sentence. It indicates that the user can press any key to jump to the next dialogue line, and it usually has the shape of a small arrow pointing right or downwards.

3.7 Visual Scripting

3.7.1 Visual Scripting

The **Dialogue** module symbiotically works with **Game Creator** and the rest of its modules using its visual scripting tools.

- · Instructions
- · Conditions
- Events

Each scripting node allows other modules to use any **Dialogue** feature.

3.7.2 Conditions

Conditions

SUB CATEGORIES

• Dialogue

Dialogue

DIALOGUE

Conditions

- · Dialogue Played
- Tag Visited

DIALOGUE PLAYED

Dialogue » Dialogue Played

Description

Returns true if the Dialogue component has been played

Parameters

Name Description

Dialogue The Dialogue component

Keywords

Dialogue Text Line Choice

TAG VISITED

Dialogue » Tag Visited

Description

Returns true if the Tag of a particular Dialogue has ran

Parameters

Name	Description
Dialogue	The Dialogue component
Tag	The Tag name to check

Keywords

Dialogue Text Line Choice

3.7.3 Events

Events

SUB CATEGORIES

• Dialogue

Dialogue

DIALOGUE

Events

- On Finish Dialogue
- On Start Dialogue

ON FINISH DIALOGUE

Dialogue » On Finish Dialogue

Description

Executed when a specific Dialogue component finishes playing

Keywords

Node Conversation Speech Text End Complete

ON START DIALOGUE

Dialogue » On Start Dialogue

Description

Executed when a specific Dialogue component starts to play

Keywords

Node Conversation Speech Text Begin Play

3.7.4 Instructions

Instructions

SUB CATEGORIES

• Dialogue

Dialogue

DIALOGUE

Sub Categories

• Ui

Instructions

- Play Dialogue
- Stop Dialogue

PLAY DIALOGUE

Dialogue » Play Dialogue

Description

Plays a dialogue

Parameters

Name	Description
Dialogue	The Dialogue component to play
Wait to Finish	Whether to wait until the Dialogue is finished or not

Keywords

Dialogue Narration Speech Next Skip

STOP DIALOGUE

Dialogue » Stop Dialogue

Description

Stop playing a dialogue

Parameters

Name Description

Dialogue The Dialogue component to stop playing

Keywords

Dialogue Narration Speech Next Skip

UI

Ui

Instructions

- · Choice Index
- Skip Line

Choice Index

Dialogue » UI » Choice Index

Description

Attempts to choose a Choice node by its index (starting at 1), if it exists

Parameters

Name	Description
Index	The numeric index of the Choice, starting from 1

Keywords

Dialogue Narration Speech Choose Pick

Skip Line

Dialogue » UI » Skip Line

Description

Finishes a dialogue UI line or skips to the next one

Parameters

Name	Description
Speech UI	The Speech UI component associated

Keywords

Dialogue Narration Speech Next Skip

3.8 Releases

3.8.1 Releases

2.4.13 (Latest)

This version breaks compatibility with previous versions and will only work with Game Creator 2.13.43 or higher. New Expression: Sprites are dynamic Properties Changes Internal: Support for Core 2.13.42 version Fixes Examples: Compatible with the latest core version Expression: Not binding values when adding new ones Expressions: Incorrect clamping index value

2.3.12

Pixes Examples: Compatible with the latest core version

2.3.11

```
New

Skin: Customize behavior when there's one choice

Fixes

Choices: Skipping conditions when using keyboard

Examples: Compatible with the latest core version
```

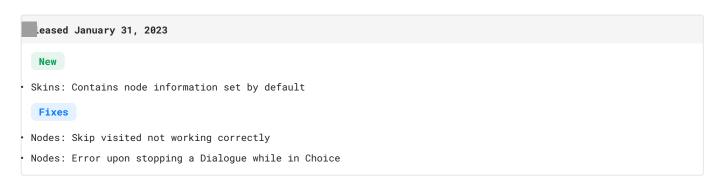
2.3.10

Portratis: Names and Descriptions cleared before playing

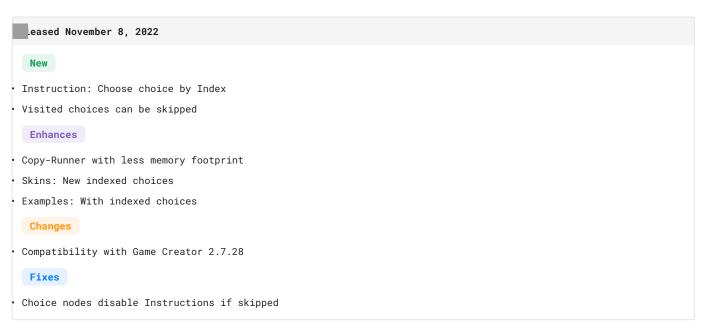
2.3.9

New Settings: Displays current and update version Enhances QoL: Align labels with Unity 2022.2 standard Fixes Edge case in which Dialogue would not save changes

2.2.8



2.1.7



2.0.6

Leased September 22, 2022 Enhances Editor: Dialogue remembers last selection Fixes Portraits exception when set to default

2.0.5



2.0.4

Pixes Sequencing tool not executing markers Using non-existing Tag throws error

2.0.3

New Option to uninstall Dialogue Icon for Dialogue Skin Enhances Default skin uses darker background Behavior of the Skip/Continue button Improved Cyberpunk Dialogue Skin Fixes Serialization error during domain reloads Null Actor reference in Log UI Typo in Example scenes

2.0.2

Pixes Marketing images fit better Conflicting meta files with other modules

2.0.1



4. Stats

4.1 Stats

Stats

Nearly all games one can play has some kind of *Stat* system; Whether it is a simple health bar with a fixed amount of hit points or a complex RPG with dozens of stats that influence the progress of the player and the outcome of any interaction.

The **Stats** module has been envisioned to help game designers more naturally and easily architect their games.

Get Stats



The Stats module is an extension of $Game\ Creator\ 2$ and won't work without it

4.2 Setup

Welcome to getting started with the **Stats** module. In this section you'll learn how to install this module and get started with the examples which it comes with.

4.2.1 Prepare your Project

Before installing the Stats module, you'll need to either create a new Unity project or open an existing one.



It is important to note that Game Creator should be present before attempting to install any module.

4.2.2 Install the Stats module

If you haven't purchased the **Stats** module, head to the Asset Store product page and follow the steps to get a copy of this module.

Once you have purchased it, click on Window Package Manager to reveal a window with all your available assets.

Type in the little search field the name of this package and it will prompt you to download and install the latest stable version. Follow the steps and wait till Unity finishes compiling your project.

4.2.3 Examples

We highly recommend checking the examples that come with the **Stats** module. To install them, click on the *Game Creator* dropdown from the top toolbar and then the *Install* option.

The **Installer** window will appear and you'll be able to manage all examples and template assets you have in your project.

- Examples: A collection of scenes with different use-case scenarios
- · Classes: A template with Stats, Attributes and Classes to kickstart your game
- UI: Samples for creating a HUD and a Character Stats menu

Installer Stats

The Examples requires both the Classes and UI extensions in order to work.



Clicking on the **Examples** install button will install all dependencies automatically.

Once you have the examples installed, click on the *Select* button or navigate to Plugins/GameCreator/Installs/Stats.Examples/.

Stats Examples

4.3 Classes

4.3.1 Classes

Taking inspiration from classic pen and paper RPG games, the **Stats** module lets you create character **Classes** which contain a collection of **Stats** and **Attributes**. On the other end, **Classes** can be assigned to any number of characters or game objects using the **Traits** component.

Stats Overview



This concepts are more easily understood with an example. Let's say we want to create a Warrior character. In this case, we would create a **Class** called "Warrior" which would contain the following **Attributes**:

- Health
- Stamina

And the following **Stats**:

- Strength
- Constitution

Now that we have the Warrior class, we can create a scene Character with the **Traits** component and assign it the Warrior **Class** defined above. This same class can be reused for other characters, such as enemies and NPCs.

4.3.2 Stats

Stats are objects that represent a particular numeric trait of a character. This value can evolve throughout the whole game and its final value can be modified using a **Formula**.

Common Stats

Common stat values on games are strength, dexterity, wisdom, luck, ...

To create a **Stat** asset, right click on the *Project panel* folder you want to create it and select Create Game Creator Stats Stat.

Stat Asset

The **ID** value must be unique throughout the whole project and it is used to identify this particular numeric trait. It is also used in **Formulas** so be sure to give it a name that's easy to remember.



We recommend sticking to acronyms or short and single worded names. For example, if the **Stat** represents the strength of the character, its ID should be str or strength.

The **Base Value** is the numeric value that the **Stat** starts with. It is worth noting this value is not necessarily the final value of the **Stat**, just a mutable numeric value.

The final value of a **Stat** is calculated applying a **Formula**. If none asset is provided, the final value is simply the **Base Value**.

Base and Formula

Let's say we have a stat with a **Base** value of 100 and a **Formula** that multiplies this value by the level (another stat value) of the character. In this case, the resulting final value of the stat would depend on the character's level.

For example, if the character is at level 1, the value would be 100 (100 \star 1). At level 2, it would be 200 (100 \star 2), at level 3 it would be 300 (100 \star 3), etc...

The **UI** dropdown contains a list of fields that can be used to display information about this particular **Stat** on the game scene, including a name, acronym, description, color and icon.

4.3.3 Attributes

Attributes are objects that represent a numeric trait of a character, but its value is clamped between a min/max range.

Common Attributes

The most common attribute is the health of a character. Its value could a value clamped between 0 and 100.

To create an **Attribute** asset, right click on the *Project panel* folder you want to create it and select Create Game Creator Stats Attribute.

Attribute Asset

The **ID** value must be unique throughout the whole project and it is used to identify this particular numeric trait. It is also used in **Formulas** so be sure to give it a name that's easy to remember.

Raming Attributes

We recommend sticking to acronyms or short and single worded names. For example, if the **Attribute** represents the health of the character, its ID should be hp or health.

The Min Value and Max Value are numeric values that represent the minimum and maximum range of the value. The Max Value comes from a Stat as this value can change at runtime.

Max Value is a Stat

For example, if the attribute represents the health of the player, levelling up could increase the maximum health. In this case, increasing a **Stat** called "Max_Health" would automatically increase the max cap of the health **Attribute**.

The **Start Percent** field defines the percent at which the character's attribute starts. By default most games should start with their attributes completely filled.

The **UI** dropdown contains a list of fields that can be used to display information about this particular **Attribute** on the game scene, including a name, acronym, description, color and icon.

4.3.4 Classes

Classes are objects that represent a type of character or object with RPG traits, and contains a list of Stats and Attributes.

Classes in an RPG

Just like in most RPGs, a **Class** defines a type character with different values. For example, a *Mage* will have the same **Stats** and **Attributes** as a *Knight*, but their values and progression may differ, making the *Mage* grow his magic abilities at a much higher rate than the *Knight*, which focuses on its physical ones.

Class

To create a **Class** asset, right click on the *Project panel* folder you want to create it and select Create Game Creator Stats Class.

By default, a **Class** has an empty list of fields. The image below represents a **Class** filled with a collection of **Stats** and **Attributes**.

Class Asset



The eye icon that appears next to all Attributes and Stats is a button that can be toggled. It has no impact on the game whatsoever. Instead it hides the option from the Traits component. This is useful if you have hundreds of Stats and Attributes and want to keep the important ones at a glance.

The **Class** and **Description** fields are used to display information about the current class in the game's user interface.

Attributes

The Attributes list defines all the attributes linked to this particular class.

To add a new **Attribute**, click on the "Add Attribute" button at the bottom and pick (or drag and drop) the desired **Attribute** asset.

Class Attributes

In this section, the selcted **Attribute**'s starting percent can be overriden, in case a particular **Class** has a different starting value than another.

Stats

The **Stats** list defines all the stats linked to this particular class, including the ones that define the max cap of Attributes.

To add a new **Stat**, click on the "Add Stat" button at the bottom and pick (or drag and drop) the desired **Stat** asset.

Class Stats

In this section, the selected **Stat** base value and formula can be overriden.

Herride Stat Base and Formula

When creating multiple RPG classes, such as Mages, Knights and Archers, it's a good practice to have the same Attributes and Stats. In order to change their progression rates, their values can be overriden within the **Class** asset itself.

For example, the wisdom base stat value may have a much higher one in a Mage class than in a Knight.

4.3.5 Traits

Traits are components that link a Class asset with a scene game object.



It is important to note that, although Characters will most likely be the objects with a **Traits** component, these can be attached to any game object.

For example, to assign the *Player* with the *Knight* **Class** one just has to click on the *Player* game object "Add Component" button at the bottom of the *Inspector* and look for the **Traits** component.

Traits in Editor

Once the Player has the Traits component a message appears prompting to assign it a Class asset.

Traits missing Class asset

Drag and drop any **Class** asset onto the designated field and it will change its appearance to display the asset's information.

Traits with Class asset

Each Attribute and Stat can be expanded and their values can be overriden, just like in the Class asset.

Traits at Runtime

Once the game object has a Traits component linked with a Class asset, it is ready to interact in play mode.

To help the designer understand what's happening in play mode and debug any possible problems, the **Traits** component changes its *Inspector* appearance to display real-time information about its current Attribute and Stat values.

Traits in Playmode

4.4 Formulas

Formulas are at the core of the Stats module; They allow the game designer to elaborate simple or complex systems that intertwine different stat and attribute values.

Math Expressions

Formulas are written using math expressions. For example the following formula:

source.stat[attack] - target.stat[defense]

Can be used to calculate the damage dealt to an enemy. It calculates the output taking into account the attack stat from the player and subtracting the defense stat from the enemy.

It is up to the game designer defining how simple or complex these formulas should be.

4.4.1 Creating a Formula

To create a **Formula** asset, right click on the *Project panel* folder you want to create it and select Create Game Creator Stats Formula.

Formula Asset

The Formula asset has a text field at the top, where the the math expression can be written.

The *Help* section contains a list of all possible symbols that can be used. For example, to retrieve the final value of a **Stat** called "strength" from the caller, use the source.stat[strength] symbol.

Each section can be expanded and collapsed to keep the important information at a glance.

Formula Help



Check the list of all symbols at the end of this page.

The *Table* field is an optional one, that can be used to reference a *Table* asset from within the formula expression.

4.4.2 Symbols

A formula expression is composed of a series of symbols, joined together by a math expression, such as the sum, subtraction, product and division.

For example, the attack power of a character could be it's base strength value multiplied by its level. In this case, the expression would be:

source.base[strength] * source.stat[level]

Stats

This section covers all values found inside a game object with a **Traits** component. A stat or attribute can either come from the **Source** object or the **Target** object. For example, when calculating the damage dealt to an enemy, **Source** references the attacker and **Target** the attacked object.

Surce and Target

In some cases, there may be no distinction between source and target. For example, when calculating the level of a character. In this case, we recommend ignoring the **Target** symbols and use **Source**.

To get the value of a **Stat** or **Attribute**, the target object of the query is first specified, followed by a dot (.) and the value type. Between brackets, the *id* of the stat or attribute is specified.

Stat Example

For example, to retrieve the attribute "mana" from the source object it's done using:

source.attr[mana]

- · base: The base stat value of the object.
- stat: The final stat value of the object.
- attr: The attribute value of the object.

Procular Formulas

It is up to the game designer to avoid circular dependencies, and Game Creator will not warn about them. A circular dependency happens when a formula requires a value, which must be calculated using the first formula. This locks the process in an infinite loop.

Variables

Variables work very similarly to retrieving Stats and Attributes. The targeted object is first specified, followed by a dot (.) and the keyword *var*. And between brackets, the name of the variable.

Example

For example, if a numeric Local Variable attached to the targetted object with the id "hit-counter" should be accessed, the expression would be:

target.var[hit-counter]

Lcal Variables

For the moment, a Formula can only access **Local Variables** by name. In a future update, **List Variable** access will be supported.

Random

Most skill checks use some sort of random values. The **Formula** analyzer provides three symbols to generate a random value

• random[min, max]: Returns a value between min and max, both included.

Random[min, max]

Using random[1, 4] returns a decimal value between these ranges.

dice[rolls, sides]: For those old-school game designers, you can roll X amount of dices of Y sides and this
symbol will return the sum of values.

Dice[rolls, sides]

Using dice[2, 6] returns the result of rolling 2 dices of 6 sides (the most common one).

· chance[value]: Returns 1 if a random value between 0 and 1 is lower or equal than the value specified.

chance[value]

Using chance[0.2] has a 20% chance of returning a value of 1 and an 80% chance of returning 0.

Arithmetic

Number manipulation is also useful and commonly used. For example, to round numbers or choosing between two.

- $\min[a, b]$: Returns the lowest value between two.
- max[a, b]: Returns the greatest value between two.
- round[value]: Returns the value rounded up or down to the closest integer.
- floor[value]: Returns the integer part of the value.
- ceil[value]: Returns the next integer of the input value.

Tables

Tables are mostly used for player progression, as they map a certain input value to another value. For more information about **Tables** see this link.

Able asset

It is required to provide the Formula with a Table asset.

Table symbols start with table followed by a dot (.) and the type of value to retrieve. The value is specified between brackets afterwards.

Level from Experience

For example, let's say we have a stat called experience and we want to calculate the character's level based on that. We can use a **Table** that transforms the accumulated experience points to a value that represents the level. In this case, the expression would be:

table.level[experience]

- · level[value]: Returns the level at from the table based on the input cummulative value.
- value[level]: Returns the cummulative value necessary to reach the input level.
- increment[level]: Returns the amount left to reach the next level.
- current[value]: Returns the value gained at the current level.
- next[value]: Returns the value left to gain to reach the next level.
- ratio[value]: Returns a unit ratio that represents the progress made at the current level.

4.5 Tables

Commonly used for character progression, Tables are charts that map a range of values to an integer.

4.5.1 Concepts

Here are some concepts to better understand how Tables work.

- Level: An integer value that is calculated based on the cumulative value.
- · Cumulative Value: This is the total amount of value (or experience) accumulated.
- · Value: The difference between the current level's cumulative value and the total cumulative value.

Table concepts

4.5.2 Creating a Table

To create a **Table** asset, right click on the *Project panel* folder you want to create it and select Create Game Creator Stats Table.

Table

A **Table** asset has a visual chart and a configuration box at the bottom. The chart can be scrubbed to reveal the different cumulative values at each level.



In the example above, at Level 13, the cumulative value is 1248 and it will require 208 more (for a total of 1455) to reach Level 14.

4.5.3 Types of Progressions

A character can progress linearly, exponentially, or at a custom rate. That's why Game Creator provides a range of different tables for the user to choose from.

Tables Progression

To change the type of progression, click onl the Table field and choose one from the dropdown menu:

- Manual: Each level requires a pre-defined amount of experience.
- $\boldsymbol{\cdot}$ $\boldsymbol{\mathsf{Constant}}\colon \mathsf{Each}\ \mathsf{level}\ \mathsf{requires}\ \mathsf{the}\ \mathsf{same}\ \mathsf{amount}\ \mathsf{of}\ \mathsf{value}\ (\mathsf{or}\ \mathsf{experience})\,.$
- Linear: Each level requires a value equal to the product of a constant and the current level.
- $\bullet \ \textbf{Geometric} \colon \ \textbf{Each level requires a value equal to the current level multiplied by a fixed coefficient rate. } \\$



We recommend using **Linear Progression** for most cases, as it's the one commonly used in games where the player progressively receives more experience. **Geometric Progression** is recommended for short games where power ramps up very quickly (like in MOBAS).

4.6 Stat Modifiers

We've seen so far that objects with a **Traits** component can change their **Stat** and **Attribute** values at runtime using **Formulas** and **Tables**. However, characters in games can also increment/decrement their stats when equipping weapons and other kinds of wearables.

This is where **Stat Modifiers** come into play: They increase or decrease a **Stat** value by a certain amount, and can be added and removed at any time.

4.6.1 Adding Stat Modifiers

To add a **Stat Modifier** to a **Traits** component, use the visual scripting Instruction **Add Stat Modifier**. This instruction allows to specify a target object, which must have a **Traits** component, a **Stat** to affect and a value.

This value can either be a percentage or a constant and can be displayed separately in the UI.

Stat Modifiers in UI

Percentage and Constants

You may have raised an eyebrow when **Stat Modifiers** can use constant and percentage values, as the result is different when applying a product after an addition or vice versa. The **Stats** module always applies percentage based modifiers first, and then adds any constant modifiers.

Add a Stat Modifier

4.6.2 Removing Stat Modifiers

Removing a **Stat Modifier** is as easy as adding one. All that needs to be done is to use the visual scripting instruction **Remove Stat Modifier** and input the same values as a previously added one.

Remove a Stat Modifier

4.7 Status Effects

Status Effects are temporal ailments that affect a character.

Most RPG games use the same **Status Effects**, such as *Poison*, which drains the character's health for a period of time. However, you can create your own and completely customize the afliction.

4.7.1 Creating a Status Effect

To create a **Status Effect** asset, right click on the *Project panel* folder you want to create it and select Create Game Creator Stats Status Effect.

Status Effects

A **Status Effect** has an ID which is used to uniquely identify it among all other afflictions. It is very important to keep this value unique across the whole project.

The **Type** field determines whether this effect is positive, negative or neutral for the targeted character. This is useful when using the instruction **Remove Status Effects**, where you can choose to remove only those that have a negative impact.

Max Stack determines how many of the same Status Effect can be active at a give time on a target.

By default, most **Status Effects** will have a stack of 1, and adding subsequent effects refresh the duration. However, it is entirely possible to stack multiple (for example) *Poison* aflictions, increasing their health drain.

The Save toggle determines whether the Status Effect persists after saving and loading back the game. Saving a Status Effect keeps track of the remaining time.

Has Duration allows the Status Effect to run for a certain amount of time (specified in the Duration field, in seconds).

If this field is unticked, the **Status Effect** will continue until it's manually removed, using the appropriate visual scripting instruction.

Status Effects UI section

The **UI** section allows the user to define any information displayable to the player, such as the name, a description of what the ailment does, its color and even an icon.

Status Effects Start End and While Active sections

Inside the **OnStart**, **On End** and **While Active** sections is where the logic of the **Status Effect** goes and it uses Game Creator's visual scripting tools.

- On Start: A list of instructions executed as soon as the Status Effect is added onto a target.
- · On End: A list of instructions executed when the Status Effect stops taking effect on a target.
- While Active: A list of instructions that runs every frame, as long as the Status Effect is active.

Poison

For example, a **Poison** status effect could start spawning a particle effect onto the targeted character using the **On Start** instruction list. To damage the player, it would use the **While Active** instruction list and subtract a bit of the Target's health every few seconds.

4.7.2 Adding a Status Effect

To add a Status Effect onto a target you can use the visual scripting instruction Add Status Effect.

Add a Status Effect

All that needs to be done is to select the targeted character, which must have a **Traits** component, and specify the type of **Status Effect**.

4.8 User Interface

4.8.1 User Interface

The Stats module makes it really easy to build flexible user interfaces (UI) using Unity UI.

Stat User Interface

It comes with a few components that work fairly similar. You can attach each component to any UI game object and drag and drop any Text and Images to each of its fields.

- · Stat UI
- · Attribute UI
- · Formula UI
- · Status Effects UI

These components are all found under the *Add Component* submenu on any game object and navigating to Game Creator UI Stats. For example, this is the **Stat UI** component.

Stat UI example

The first two fields are required: **Target** is the game object with a **Traits** component and **Stat** is the asset to be referenced by this UI component.

All other fields are optional and will only be updated if a change is detected.

Stat UI

For example, dragging a **Text** component onto the *Value* field will change the contents to a numeric value that represents the selected **Stat** value.

4.8.2 Stat UI

The **Stat UI** component allows to display the runtime information about a specific target's **Stat**. To create one, click on a game object's *Add Component* button and navigate to Game Creator UI Stats Stat UI.

Stat UI

All fields are optional and all that needs to be done is to drag **Text** and **Image** components to the corresponding fields.

Stat UI

For example, to display the Name of a **Stat**, drag and drop the **Text** component onto the Name field and it will automagically update its content, even if the targeted game object changes.

4.8.3 Attribute UI

The **Attribute UI** component allows to display the runtime information about a specific target's **Attribute**. To create one, click on a game object's *Add Component* button and navigate to Game Creator UI Stats Attribute UI.

Attribute UI

All fields are optional and all that needs to be done is to drag **Text** and **Image** components to the corresponding fields.

Attribute UI

For example, to display the Name of an **Attribute**, drag and drop the **Text** component onto the Name field and it will automagically update its content, even if the targeted game object changes.

Transitions are a feature that allow the **Image** fill progress to animate and stall for a certain amount of time.

Attribute UI Transitions



This is mostly used on health and mana bars, where getting hit makes the HP bar display a second bar below that decreases after a few seconds, in order for the player to get a sense of the amount of damage taken.

Ticking any of both options reveals two new options below.

- · Stall Duration: Amount of seconds debounced between the value change and the start of the transition
- Transition Duration: Amount of seconds it takes to animate towards the targeted value.

4.8.4 Formula UI

The Formula UI component allows to display the result of an expression between two game objects with a Traits component. To create one, click on a game object's Add Component button and navigate to Game Creator UI Stats Formula UI.

Formula UI

All fields are optional and all that needs to be done is to drag **Text** and **Image** components to the corresponding fields.

Formula UI

For example, to display the resulting value of a **Formula** applied to the Player and another character, drag and drop the **Text** component onto the *Value* field and it will automagically update its content, even if any of the targeted game objects changes.

4.8.5 Status Effects UI

Status Effects have two components to display their information.

Status Effect List UI

- Status Effect List UI: Gathers information about a targeted game object and manages the concrete list of
- Status Effect UI: Displays information about a particular affliction. It is spawned by the Status Effect List UI component.

Status Effect List UI

To create one, click on a game object's Add Component button and navigate to Game Creator UI Stats Status Effect List UI.

Status Effect List UI

The Target field should point at the game object with a Traits component.

Types allows to filter which status effects to display: Negative, Positive, Neutral, or any combination of them.

Container and Prefab Status Effect are the most important ones: For each afliction on the targeted character, the Status Effect List UI component will spawn (or reuse) an instance of a prefab. The spawn location is as a child of the Container rect transform.

Example

So if the Player has 3 ailments: *Poison, Paralyzed* and *Bleeding*, the **Status Effect List UI** component will spawn 3 instances of the prefab as a child of the **Container** transform.

Each spawned instance must have, at the root level, the component **Status Effect UI** component, which communicates with the **Status Effect List UI** which afliction to display.

Status Effect UI

To create one, click on a game object's *Add Component* button and navigate to Game Creator UI Stats Status Effect UI.

Status Effect UI

As can be seen, this component does not have a **Target** field. Instead, its the **Status Effect List UI** component that feeds it the target and concrete afliction.

All fields are optional and automatically update the values according to changes sent by the parent component.

4.9 Visual Scripting

4.9.1 Visual Scripting

The **Stats** module symbiotically works with **Game Creator** and the rest of its modules using its visual scripting tools.

- · Instructions
- · Conditions
- Events

Each scripting node allows other modules to use any Stats feature.

The **Stats** module also comes with a collection of custom **Properties**. Any interactive element can request the value of a **Stat**, **Attribute** and **Formula** using the value dropdown, as seen in the image below.

Properties

4.9.2 Conditions

Conditions

SUB CATEGORIES

• Stats

Stats

STATS

Conditions

- · Check Formula
- · Compare Attribute
- · Compare Stat
- Has Stat Modifiers
- · Has Status Effect

CHECK FORMULA

Stats » Check Formula

Description

Returns the comparison between the result of a Formula against another value

Parameters

Name	Description
Formula	The Formula used in the operation
Source	The game object that the Formula identifies as the Source
Target	The game object that the Formula identifies as the Target
Compare To	The value that the result of the Formula is compared to

Keywords

Skill Throw Check Dice Lock Pick Charisma Speech

COMPARE ATTRIBUTE

Stats » Compare Attribute

Description

Returns true if the Attribute comparison is $\operatorname{successful}$

Parameters

Name	Description
Traits	The targeted game object with a Traits component
Attribute	The Attribute type value that is compared
Value	The type of value from the attribute to compare
Comparison	The comparison operation performed between both values
Compare To	The decimal value that is compared against

Keywords

Health Mana Stamina Magic Life HP MP

COMPARE STAT

Stats » Compare Stat

Description

Returns true if the Stat comparison is successful

Parameters

Name	Description
Traits	The targeted game object with a Traits component
Stat	The Stat type value that is compared
Comparison	The comparison operation performed between both values
Compare To	The decimal value that is compared against

Keywords

Vitality Constitution Strength Dexterity Defense Armor Magic Wisdom Intelligence

HAS STAT MODIFIERS

Stats » Has Stat Modifiers

Description

Returns true if the targeted Stat component has a Stat $\operatorname{Modifier}$

Parameters

Name	Description
Target	The targeted game object with a Traits component
Stat	The Stat that checks if it has a Stat Modifier

Keywords

Skill Throw Check Dice Lock Pick Charisma Speech

HAS STATUS EFFECT

Stats » Has Status Effect

Description

Returns true if the game object has a particular Status Effect active

Parameters

Name	Description
Target	The targeted game object with a Traits component
Status Effect	The type of Status Effect that is checked
Min Amount	The minimum amount of stacked and active Status Effects

Keywords

Buff Debuff Enhance Ailment Blind Dark Burn Confuse Dizzy Stagger Fear Freeze Paralyze Shock Silence Sleep Silence Slow Toad Weak Strong Poison Haste Protect Reflect Regenerate Shell Armor Shield Berserk Focus Raise

4.9.3 Events

Events

SUB CATEGORIES

• Stats

Stats

STATS

Events

- On Attribute Change
- On Stat Change
- On Status Effect Change

ON ATTRIBUTE CHANGE

Stats » On Attribute Change

Description

Executed when the value of a specific game object's Attribute is modified

Parameters

Name	Description
Target	The targeted game object with a Traits component
When	Determines if the event executes when the Attribute increases, decreases or both
Attribute	The Attribute from which the event detects its changes

Keywords

Health HP Mana MP Stamina

ON STAT CHANGE

Stats » On Stat Change

Description

Executed when the value of a specific game object's Stat is modified. Including due to Stat Modifiers

Parameters

Name	Description
Target	The targeted game object with a Traits component
When	Determines if the event executes when the Stat increases, decreases or both
Stat	The Stat from which the event detects its changes

Keywords

Health HP Mana MP Stamina

ON STATUS EFFECT CHANGE

Stats » On Status Effect Change

Description

Executed when a Status Effect is added or removed from a Traits component

Parameters

Name	Description
Target	The targeted game object with a Traits component
Status Effect	Determines if the event detects any Status Effect change or a specific one

Keywords

Buff Debuff Enhance Ailment Blind Dark Burn Confuse Dizzy Stagger Fear Freeze Paralyze Shock Silence Sleep Silence Slow Toad Weak Strong Poison Haste Protect Reflect Regenerate Shell Armor Shield Berserk Focus Raise

4.9.4 Instructions

Instructions

SUB CATEGORIES

• Stats

Stats

STATS

Sub Categories

• Ui

Instructions

- Add Stat Modifier
- Add Status Effect
- Change Attribute
- · Change Stat
- Clear Status Effects Type
- Remove Stat Modifier
- Remove Status Effect

ADD STAT MODIFIER

Stats » Add Stat Modifier

Description

Adds a value Modifier to the selected Stat on a game object's Traits component

Parameters

Name	Description
Target	The targeted game object with a Traits component
Stat	The Stat that removes the Modifier
Туре	If the Modifier changes the Stat by a constant value or by a percentage
Value	The constant or percentage-based value of the Modifier

Keywords

Slot Increase Equip Fortify Vitality Constitution Strength Dexterity Defense Armor Magic Wisdom Intelligence

ADD STATUS EFFECT

Stats » Add Status Effect

Description

Adds a Status Effect to the selected game object's Traits component

Parameters

Name	Description
Target	The targeted game object with a Traits component
Status Effect	The type of Status Effect that is added

Keywords

Buff Debuff Enhance Ailment Blind Dark Burn Confuse Dizzy Stagger Fear Freeze Paralyze Shock Silence Sleep Silence Slow Toad Weak Strong Poison Haste Protect Reflect Regenerate Shell Armor Shield Berserk Focus Raise

CHANGE ATTRIBUTE

Stats » Change Attribute

Description

Changes the current Attribute value of a game object's Traits component

Parameters

Name	Description
Target	The targeted game object with a Traits component
Attribute	The Attribute type that changes its value
Change	The value changed

Keywords

Health HP Mana MP Stamina

CHANGE STAT

Stats » Change Stat

Description

Changes the base Stat value of a game object's Traits component

Parameters

Name	Description
Target	The targeted game object with a Traits component
Stat	The Stat type that changes its value
Change	The value changed

Keywords

Vitality Constitution Strength Dexterity Defense Armor Magic Wisdom Intelligence

CLEAR STATUS EFFECTS TYPE

Stats » Clear Status Effects Type

Description

Clears any Status Effects based on their type from the selected game object's Traits component

Parameters

Name	Description
Target	The targeted game object with a Traits component
Types	The type of Status Effects that are cleared

Keywords

Buff Debuff Enhance Ailment Blind Dark Burn Confuse Dizzy Stagger Fear Freeze Paralyze Shock Silence Sleep Silence Slow Toad Weak Strong Poison Haste Protect Reflect Regenerate Shell Armor Shield Berserk Focus Raise

REMOVE STAT MODIFIER

Stats » Remove Stat Modifier

Description

Removes an equivalent Modifier from the selected Stat on a game object's Traits component.

Parameters

Name	Description
Target	The targeted game object with a Traits component
Stat	The Stat that receives the Modifier
Туре	If the Modifier changes the Stat by a constant value or by a percentage
Value	The constant or percentage-based value of the Modifier

Keywords

Slot Decrease Unequip Weaken Vitality Constitution Strength Dexterity Defense Armor Magic Wisdom Intelligence

REMOVE STATUS EFFECT

Stats » Remove Status Effect

Description

Removes a Status Effect from the selected game object's Traits component

Parameters

Name	Description
Target	The targeted game object with a Traits component
Amount	Indicates how many Status Effects are removed at most
Status Effect	The type of Status Effect that is removed

Keywords

Buff Debuff Enhance Ailment Blind Dark Burn Confuse Dizzy Stagger Fear Freeze Paralyze Shock Silence Sleep Silence Slow Toad Weak Strong Poison Haste Protect Reflect Regenerate Shell Armor Shield Berserk Focus Raise

UI

Ui

Instructions

- Change Attributeui Attribute
- Change Attributeui Target
- · Change Statui Stat
- · Change Statui Target
- · Change Status Effects List Ui Target

Change AttributeUI Attribute

Stats » UI » Change AttributeUI Attribute

Description

Changes the Attribute from a Attribute UI component

Name	Description
Attribute UI	The game object with the Attribute UI component
Attribute	The new Attribute asset

Change AttributeUI Target

Stats » UI » Change AttributeUI Target

Description

Changes the targeted game object of an Attribute UI component $% \left(1\right) =\left(1\right) \left(1\right) +\left(1\right) \left(1\right) \left(1\right) +\left(1\right) \left(1\right) \left($

Name	Description
Attribute UI	The game object with the Attribute UI component
Target	The new targeted game object with a Traits component

Change StatUI Stat

Stats » UI » Change StatUI Stat

Description

Changes the Stat asset from a Stat UI component

Name	Description
Stat UI	The game object with the Stat UI component
Stat	The new Stat asset

Change StatUI Target

Stats » UI » Change StatUI Target

Description

Changes the targeted game object of an Stat UI component

Name	Description
Stat UI	The game object with the Stat UI component
Target	The new targeted game object with a Traits component

Change Status Effects List UI Target

Stats » UI » Change Status Effects List UI Target

Description

Changes the targeted game object of an Status Effects List UI component

Name	Description
Status Effects List UI	The game object with the Status Effects List UI component
Target	The new targeted game object with a Traits component

4.10 Releases

4.10.1 Releases

2.5.14 (Latest)

eased October 31, 2023

This version breaks compatibility with previous versions and will only work with Game Creator 2.13.43 or higher.

Changes

Status Effects: More performant instructions

UX: Different layout for Stat overrides

UX: Different layout for Attribute overrides

Internal: Support for Core 2.13.42 version

Fixes

Stat Modifiers: Clear method does not zero value

UI: Attribute scale on Y axis scales on X

2.4.13

eased August 31, 2023

New

· Status Effect: Option to make them Hidden

Fixes

Examples: Compatibility with latest core version

2.4.12

eased June 27, 2023

Fixes

· Formulas: Source and Target only use Traits components

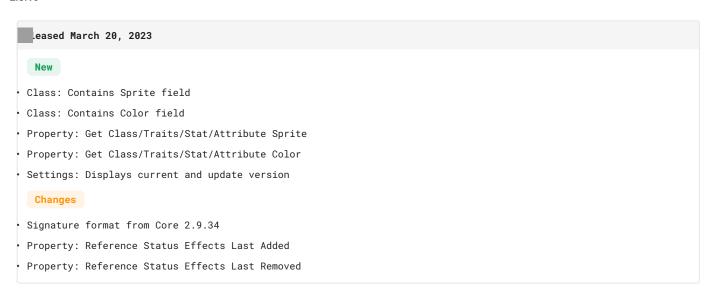
Formulas: Incorrect Regex expressions in parsing clauses

• Examples: Not working with latest core version

2.4.11

New UI: Display attributes as independent units Condition: Has Stat Modifiers Property: Stat and Attribute Sprite values Examples: New Attribute Unit example Changes Formulas: New precomputable math library Fixes Formulas: Chance ratio is inverted Formulas: Error when Tables are missing

2.3.10



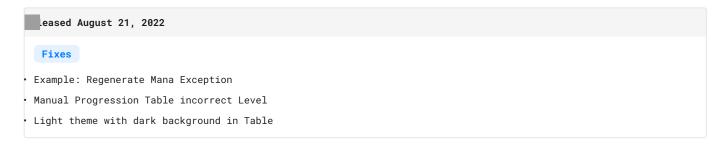
2.2.9



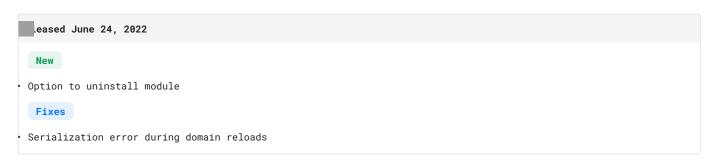
2.1.8

Leased September 21, 2022 Fixes Constant Table: Experience calculation Geometry Table: Experience calculation

2.1.7



2.1.6



2.1.5

Pixes Incorrect Stat Modifiers application order Crash when overriding Trait component values Math expressions support line breaks Removed duplicate internal method

2.1.4

New Property: Stat Modifiers value Enhances Formulas can have multiple lines Moved UI components to submenu Changes Example scenes compatibility Fixes Attribute UI: Scale options disappeared Incorrect caching of Status Effects Alignment of elements in Inspector

2.0.3

Enhances Classes installer has no dependencies Easier to understand examples

2.0.2

```
New

Instruction: Change AttributeUI Attribute

Instruction: Change StatUI Stat

Enhances

UI instructions are now found under Stats/UI/
Disallow multiple Traits component per object

Fixes

Event: Attribute Change not running
```

2.0.1

```
New

First release
```

5. Quests

5.1 Quests

Quests

Between main quests, side quests, bestiary and flora information gathering, lore, ... Managing the progress of the game can quickly become a daunting task (no pun intended).

The **Quests** module aims to help automatizing the creation and management of quests using a simple set of rules. These rules allow to easily create any type of quests while keeping it intuitive and easy to modify and iterate over.

Moreover, the **Quests** module also comes with common user interface tools, such as a Minimap, visual Indicators and a Navigation Compass system that automagically displays active Tasks and where the destination is.

Get Quests



The Quests module is an extension of Game Creator 2 and won't work without it

5.2 Setup

Welcome to getting started with the **Quests** module. In this section you'll learn how to install this module and get started with the examples which it comes with.

5.2.1 Prepare your Project

Before installing the Quests module, you'll need to either create a new Unity project or open an existing one.



It is important to note that Game Creator should be present before attempting to install any module.

5.2.2 Install the Quests module

If you haven't purchased the **Quests** module, head to the Asset Store product page and follow the steps to get a copy of this module.

Once you have bought it, click on Window Package Manager to reveal a window with all your available assets.

Type in the little search field the name of this package and it will prompt you to download and install the latest stable version. Follow the steps and wait till Unity finishes compiling your project.

5.2.3 Examples

We highly recommend checking the examples that come with the **Quests** module. To install them, click on the *Game Creator* dropdown from the top toolbar and then the *Install* option.

The **Installer** window will appear and you'll be able to manage all examples and template assets you have in your project.

- ${\boldsymbol{\cdot}}$ ${\boldsymbol{Examples}}\colon$ A collection of scenes with different use-case scenarios
- UI: A bundle of common user interface elements

Installer Ouests

The ${\bf Examples}$ requires all the skins in order to work.



Clicking on the Examples install button will install all dependencies automatically.

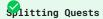
Once you have the examples installed, click on the *Select* button or navigate to Plugins/GameCreator/Installs/Quests.Examples/.

Quests Examples

5.3 Ouests

5.3.1 Quest

The Quest asset contains a collection of Tasks that are required to be completed in order to consider the Quest fulfilled.



A naive approach is to consider a single Quest as the main quest, while having multiple Quest assets for each side-quest of a game. However, because the main quest of a game might quickly become very big, it's advisable to split it into multiple Quests and activate these when completing the previous ones.

At the end though, it's you who decides how to organize the Quests of your game.

To create a new Quest asset, right click on the Project Panel and select Create Game Creator Ouests Ouest.

Ouest Asset Overview

Overview

The Quest asset has three very distinct sections:

The top section includes general information about the Quest such as its Name or a Description (if any). It also optionally allows to determine a Color and a Sprite image used in UI.

The Type field determines whether the Quest is a hidden quest, or a normal one.

Hidden Quests

Hidden quests can be hidden from UI elements and are useful for setting up missions that should not be displayed to the user. For example, an achievement system.

The Sorting Order determines the priority of the Quest compared to the rest, when being displayed as a list on UI elements. A Quest with a higher value will be displayed above other Quest assets.

The ID is a unique identifier that distinguishes a Quest from others.



Two Quests with the same ID

If there are two Quest assets with the same ID value, an error message will appear above. To resolve it, click on any of the fields and it will reveal a button that regenerates the current value with a unique one.

Quest Tasks Hierarchy

The second section of the Quest asset is the Tasks Hierarchy, which controls how the Quest runs. We cover this section in detail on the Tasks page.

The last section contains a collection of Instructions that are executed when the Quest changes its state.

Quest Callbacks

When a Quest is Completed

For example, the **On Complete** instructions will be executed as soon as the **Quest** is successfully completed. This can be used to give the Player some rewards, display a notification, etc...

States

A **Quest** starts in an **Inactive** state. In order to start a quest, the instruction **Quest Activate** can be used, which will enable it in a particular **Journal** component.

Quest Activate

When activating a **Quest**, the first root **Task** is also activated. This process cascades to any other subtasks the **Task** may have. Once the first **Task** is completed, its next sibling is **Activated**. This process is repeated until all root **Tasks** are finished.



See the Tasks States for more information about running tasks.

An Active quest can then either transition to Inactive, or one of the following Finished states:

- · Completed
- Abandoned
- Failed

A quest is automatically **Completed** if all of its root tasks are completed (in sequence, from top to bottom). If a root task is **Abandoned** or **Failed**, the quest will also be automatically **Abandoned** or **Failed** respectively.

At any point, a quest can be deactivated using the Quest Deactivate instruction.

Quest Deactivate

5.3.2 Tasks

At the bottom section of the **Quest** asset there's the *Tasks Hierarchy* panel, which controls the logic behind the **Quest**.

Quest Tasks Hierarchy



A Task is a node that can contain a series of Subtasks, which in turn may contain other Subtasks.

The two buttons at the left of the top toolbar allow creating a new **Task**: The left one creates a *Task* node as a sibling of the current selected one, while the right one creates a child *Task*.

The right button toggles the Task Inspector tab, which allows to edit the currently selected Task details.



You can hold the left mouse button over a Task and drag it somewhere else to reorganize your Tasks

Task States

- A Task can be in one of the following states at any given time:
- Inactive: The default state.
- · Active: An active task is currently being executed and can transition to a finished state.
- Completed: The task has been successfully resolved.
- · Abandoned: The task has been abandoned, with similar effects to the failed state.
- Failed: The task has been failed.

Tasks Transitions

A Task can't transition to and from any state. Instead, there's a set of rules that define those.

- · An Inactive Task can only transition to an Active state.
- · An Active Task can transition to either Inactive, or to any finished state.
- A Finished state can only transition to an Inactive state.

Finished States

A $\it Finished$ state means either $\it Completed$, $\it Abandoned$ or $\it Failed$.

Switch to an invalid State

If an **Inactive** task tries to change its state to **Completed**, the command will be ignored because only an **Active** task can be completed.

Task Anatomy

To modify the properties of a **Task**, select it from the *Tasks Hierarchy* and reveal the *Inspector* on the right side by clicking on the top-right button of its toolbar.

Task Details

SETTINGS

The Completion mode field determines how this Task is completed, if it has any subtasks.



More information about Subtasks at the Running Subtasks section.

The **Is Hidden** field determines whether this particular task should be considered as hidden. This is used to skip displaying a particular task-line in the UI.

The Name and Description fields are also used by the user interface to communicate the information about this particular task.

The next fields, **Color** and **Sprite**, are optional and can be used to customize the appearance of different tasks

using Sprites for Tasks

For example, it may be desirable to display a different icon on the HUD depending on the task at hand. Some investigation tasks might display a magnifier, while an assassination task could display a skull icon.

COUNTERS

The **Use Counter** allows to define a task as a countable one or not. The options available are *None*, *Value* and *Property*.

No Counter

By default, a task is set to *None* by default. This means that the task must be completed using the *Complete Task* instruction. However, tasks can also include a counter that automatically completes the quest when the value and the counter become equal.

Example of Counter task

The most common example of a Counter task is when an NPC asks the player to kill a certain number of enemies. The counter would be the amount of enemies to kill, while the value would be the enemies killed so far.

As soon as the value and the counter are the same, the task is automatically completed.

Value Counter

This option displays a single Count To field, which is the value to reach in order to complete this task.

Task Counter type Value

Kill 5 boars

For example, if an NPC tasks the player to kill 5 boars, the **Count To** field would be 5. The starting value is zero at the beginning, and can be changed using the **Task Value** instruction.

In this case, the instruction would increment in +1 the value of the task, automatically completing it after defeating 5 boars.

Property Counter

The Property option is a bit more advanced, and allows to also count up to a certain amount in order to automatically complete the task, but the value is synchronized with a dynamic property.

Task Counter type Property

The Count To field, just like in the previous option, defines the desired value to reach.

The Value From field is a dynamic property that allows to choose the source from which the current value is taken. For example, a Global Variable.



From other Game Creator modules

This option allows to seamlessly combine Quests with other Game Creator modules. For example, a quest giver may ask to collect a certain amount of Potions, which is defined as an item in the Inventory module. The Value From, in this case, would be the amount of Potions.

The Detect When event is used to determine when the synchronization should be executed. For example, if the dynamic value comes from a Global Name Variable, the detection should be set to run when a global variable changes.

Follow-up with the Inventory module

Similarly, if we are using the amount of a particular Item of the Inventory module as the value of a counter task, the detection should be set whenever the Bag component changes.

INSTRUCTIONS

A Task, just like a Quest, has a collection of Instructions that can be executed whenever a task changes its state.

- The On Deactivate is executed when a task changes its state to Inactive.
- The On Activate is executed when a task changes its state to Active.
- The On Complete is executed when a task changes its state to Completed.
- The On Abandon is executed when a task changes its state to Abandoned.
- The On Fail is executed when a task changes its state to Failed.

Running Subtasks

A Task that has one or more child Subtasks will be automatically Completed, Abandoned or Failed, depending on the value of its Completion field.

Tasks Completion Options

SLIBTASKS IN SECUENCE

This type of Task activates the first Subtask as soon as it is activated, leaving any subsequent subtasks inactive.

Subtasks in Sequence

When the Subtask is completed, the next sibling task is activated. This is repeated until all Subtasks are completed, at which point the Task is automatically completed too.

If any Subtask is abandoned or failed, the Task is also abandoned or failed, respectively.



Running a series of tasks in order is the most common type. For example, a quest-giver asking to find its *Magic Sword* and return it to them. In this case, finding the *Magic Sword* would be the first subtask, and completing it would activate the second subtask: returning the item to the quest-giver.

SUBTASKS IN COMBINATION

This type of Task activates all Subtasks as soon as it is activated.

Subtasks in Combination

These **Subtasks** can be completed in any order, and as soon as all of them are completed, the **Task** will also become completed.

If any Subtask is abandoned or failed, the Task is also abandoned or failed, respectively.



This type is mostly used during investigation segments: The player arriving at a crime scene and having to investigate multiple clues, in any order. For example, talking to a witness, investigating the footprints and doing a preliminary autopsy on the victim. After all these subtasks have been completed, the task will be completed too.

ANY SUBTASK

This type of Task activates all Subtasks as soon as it is activated.

Any Subtask

As soon as any **Subtask** is completed, the **Task** will automatically be completed too and leave the rest of the **Subtasks** as active.

Because it only requires a single **Subtask** to be completed in order to complete the **Task**, the **Task** won't be abandoned or failed unless there are no other inactive **Subtasks**.



This type is used when making branching decisions where completing one subtask determines a different path than completing another subtask. For example, killing a targeted enemy or sparing its life. Once a decision has been made (aka a subtask has been completed), it locks the player from doing the other one.

MANUAL

This type of $\boldsymbol{\mathsf{Task}}$ does not activate any $\boldsymbol{\mathsf{Subtasks}}$ when activated.

Manual Subtask

If the other modes do not fit a particular quest flow, this one can be selected in order to customize each step, as it doesn't automatize any changes.



There aren't any particular use cases. However, if you want to take full control over when a task is completed (for example, despite its children subtasks not being completed), this might be useful.

5.3.3 Tracking

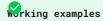
Tracking a Quest means the player will prominently see that particular quest highlighted among the rest.



For example, by default, the **HUD** will only display those Quests being tracked, so the player is not overwhelmed having too many quests active at a time.

The **Quests** module allows to either limit the amount of quests tracked to a single one, or multiple ones. You can customize this behavior by changing it in the dropdown menu field in the **Journal** component.

To start **Tracking** a quest, you can either use the **Quest Track** instruction, or let the **UI** components that the **Quests** module comes with, handle it.



The UI Journal template that the **Quests** module comes with, contains an example where a list of active quests are displayed on the left side, and selecting one allows to toggle its tracking state.

Quest Tracking Example

To **Untrack** a quest, you can either toggle it from the UI elements or use the **Quest Untrack** instruction. Alternatively, you can also stop tracking all quests by using the **Quests Untrack All** instruction.

5.3.4 Points of Interest

A Point of Interest is a highlighted location that is of interest to the player.

Setup

Defining one is done by adding either the Task Point of Interest or the Custom Point of Interest.

- Task Point of Interest: Defines a point of interest linked to a specific *Task*. When the task is in an *Active* state, the point of interest is enabled. Otherwise, it's disabled.
- Custom Point of Interest: Defines a point of interest not bound to any specific task or quest. Useful for positioning objects that are not related to quests, such as enemies, collectibles, etc, ...

Point of Interest on Hotspot

Showing Points of Interest

A Point of Interest is automatically displayed using one of the Points of Interest UI components.

The **Quests** module comes with a collection of game-ready systems that you can drag and drop onto your game and they will automagically work.

For example, the **Minimap** prefab from the examples displays a rectangle on the bottom right corner of the screen.

Point of Interests Minimap

The **Compass** from the examples shows a minimalist line at the top of the screen with elements that fade in and out as they are shown on-screen.

Point of Interests Compass

The **Indicators** prefab displays the icon of the Task directly on top of the screen space position of the scene object.

Point of Interests Indicators



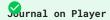
By default, the **Indicators** prefab displays off-screen elements at the closest edge of the screen, with an arrow indicating its direction.

However, this can be disabled unticking the Keep in Bounds field from the Indicators UI component.

5.4 Journal

The Journal is a component that keeps track of the current state of Quests and its tasks.

Quest Journal Component



It is usually attached to the **Player** character object so it's easy to access. However, you can decide to attach it to some other object or even have multiple characters, each with their own quests log.

5.4.1 Tracking

The **Journal** component determines whether it can track only one **Quest** at a time, or multiple quests. If the value is set to **Single Quest**, attempting to track a quest will untrack any previous tracked one.

However, if the value is set to **Multiple Quests**, tracking another one will insert it to the list of tracked quests, without untracking any others.

More information

To know more about tracking quests and how is it used, see the **Tracking** section.

5.4.2 Debugging

After entering play-mode, the **Journal** component changes its appearance and will display real-time information about the current state of *Quests* and *Tasks*.

Quest Journal Debug in Real-Time

This allows to easily debug whether a Quest has been properly activated, which Tasks are completed, and so on.



You can click on the **Quest** and **Task** to toggle its expand state, in case there is a lot of visual noise due to the amount of elements.

5.4.3 Saving the Game

The **Journal** doesn't automatically store the quests and tasks states. In order to do so, simply add the **Remember** component to where the Journal component is and add the **Journal** memory.

Quest Journal Remember Memory

This will automatically handle saving the state of Tasks and Quests, and loading them back when a previously saved game is loaded.

5.5 Visual Scripting

5.5.1 Visual Scripting

The **Quests** module symbiotically works with **Game Creator** and the rest of its modules using its visual scripting tools.

- · Instructions
- · Conditions
- Events

Each scripting node allows other modules to use any ${\bf Quests}$ feature.

5.5.2 Conditions

Conditions

SUB CATEGORIES

• Quests

Quests

QUESTS

Sub Categories

• Groups

Conditions

- Are Quests Equal
- Is Quest Abandoned
- Is Quest Active
- Is Quest Completed
- Is Quest Failed
- Is Quest Inactive
- Is Task Abandoned
- Is Task Active
- Is Task Completed
- Is Task Failed
- Is Task Inactive

ARE QUESTS EQUAL

Quests » Are Quests Equal

Description

Returns true if two given Quest assets are the same $% \left(1\right) =\left(1\right) \left(1\right) \left$

Keywords

Journal Mission Task

IS QUEST ABANDONED

Quests » Is Quest Abandoned

Description

Returns true if a Quest from a Journal is abandoned

Keywords

IS QUEST ACTIVE

Quests » Is Quest Active

Description

Returns true if a Quest from a Journal is active

Keywords

IS QUEST COMPLETED

Quests » Is Quest Completed

Description

Returns true if a Quest from a Journal is completed

Keywords

IS QUEST FAILED

Quests » Is Quest Failed

Description

Returns true if a Quest from a Journal is failed

Keywords

IS QUEST INACTIVE

Quests » Is Quest Inactive

Description

Returns true if a Quest from a Journal is inactive

Keywords

IS TASK ABANDONED

Quests » Is Task Abandoned

Description

Returns true if a Task from a Journal is abandoned

Keywords

IS TASK ACTIVE

Quests » Is Task Active

Description

Returns true if a Task from a Journal is active

Keywords

IS TASK COMPLETED

Quests » Is Task Completed

Description

Returns true if a Task from a Journal is completed

Keywords

IS TASK FAILED

Quests » Is Task Failed

Description

Returns true if a Task from a Journal is failed

Keywords

IS TASK INACTIVE

Quests » Is Task Inactive

Description

Returns true if a Task from a Journal is inactive

Keywords

GROUPS

Groups

Conditions

- Are All Quests Completed
- Is Any Quest Completed

Are all Quests Completed

Quests » Groups » Are all Quests Completed

Description

Returns true if at least one Quest from a List is Complete

Keywords

Journal Mission Group

Is any Quest Completed

Quests » Groups » Is any Quest Completed

Description

Returns true if at least one Quest from a List is Complete

Keywords

Journal Mission Group

5.5.3 Events

Events

SUB CATEGORIES

• Quests

Quests

QUESTS

Events

- On Any Quest Track
- On Any Quest Untrack
- On Quest Abandon
- On Quest Activate
- On Quest Complete
- On Quest Deactivate
- On Quest Fail
- On Task Abandon
- On Task Activate
- On Task Complete
- On Task Deactivate
- On Task Fail
- On Task Value Change

ON ANY QUEST TRACK

Quests » On Any Quest Track

Description

Executes after a Quest from a Journal starts being tracked

Keywords

Journal Mission Follow

ON ANY QUEST UNTRACK

Quests » On Any Quest Untrack

Description

Executes after a Quest from a Journal stops being tracked

Keywords

Journal Mission Follow

ON QUEST ABANDON

Quests » On Quest Abandon

Description

Executes after a Quest from a Journal is abandoned

Keywords

ON QUEST ACTIVATE

Quests » On Quest Activate

Description

Executes after a Quest from a Journal is activated

Keywords

ON QUEST COMPLETE

Quests » On Quest Complete

Description

Executes after a Quest from a Journal is completed

Keywords

ON QUEST DEACTIVATE

Quests » On Quest Deactivate

Description

Executes after a Quest from a Journal is deactivated

Keywords

ON QUEST FAIL

Quests » On Quest Fail

Description

Executes after a Quest from a Journal is failed

Keywords

ON TASK ABANDON

Quests » On Task Abandon

Description

Executes after a Task from a Journal is abandoned

Keywords

ON TASK ACTIVATE

Quests » On Task Activate

Description

Executes after a Task from a Journal is activated

Keywords

ON TASK COMPLETE

Quests » On Task Complete

Description

Executes after a Task from a Journal is completed

Keywords

ON TASK DEACTIVATE

Quests » On Task Deactivate

Description

Executes after a Task from a Journal is deactivated

Keywords

Journal Mission

ON TASK FAIL

Quests » On Task Fail

Description

Executes after a Task from a Journal is failed

Keywords

Journal Mission

ON TASK VALUE CHANGE

Quests » On Task Value Change

Description

Executes after a specific Active Task from a Journal changes its value

Keywords

Journal Mission

5.5.4 Instructions

Instructions

SUB CATEGORIES

• Quests

Quests

QUESTS

Instructions

- Quest Activate
- Quest Deactivate
- Quest Track
- Quest Untrack All
- Quest Untrack
- Set Quest
- Task Abandon
- · Task Complete
- Task Fail
- · Task Value

QUEST ACTIVATE

Quests » Quest Activate

Description

Changes the state of a Quest on a Journal component to Active

Parameters

Name	Description
Journal	The Journal component that changes the state of the Quest
Quest	The Quest asset reference
Wait to Complete	Whether to wait until the Quest finishes running its Instructions

Keywords

Mission Start Active Enable

QUEST DEACTIVATE

Quests » Quest Deactivate

Description

Changes the state of a Quest and its Tasks on a Journal component to Inactive

Parameters

Name	Description
Journal	The Journal component that changes the state of the Quest
Quest	The Quest asset reference
Wait to Complete	Whether to wait until the Quest finishes running its Instructions

Keywords

Mission Start Deactivate Inactive Disable

QUEST TRACK

Quests » Quest Track

Description

Starts tracking a Quest if it is active

Parameters

Name	Description
Journal	The Journal component that starts tracking the Quest
Quest	The Quest asset reference

Keywords

Mission Follow Bookmark

QUEST UNTRACK ALL

Quests » Quest Untrack All

Description

Stops tracking all Quests that are being tacked $% \left(1\right) =\left(1\right) \left(1\right$

Parameters

Name Description

Journal The Journal component that tracks the Quest

Keywords

Mission Follow Bookmark Track

QUEST UNTRACK

Quests » Quest Untrack

Description

Stops tracking a Quest if it is being tacked

Parameters

Name	Description
Journal	The Journal component that tracks the Quest
Quest	The Quest asset reference

Keywords

Mission Follow Bookmark Track

SET QUEST

Quests » Set Quest

Description

Sets a Quest value equal to another one

Parameters

Name	Description
Set	Where the value is set
From	The value that is set

Keywords

Change Task Variable Asset

TASK ABANDON

Quests » Task Abandon

Description

Abandons the state of an active Task on a Journal component

Parameters

Name	Description
Journal	The Journal component that changes the state of the Task
Quest	The Quest asset reference
Task	The Task identifier from the Quest
Wait to Complete	Whether to wait until the Task finishes running its Instructions

Keywords

Mission Leave Forget Stop Restart

TASK COMPLETE

Quests » Task Complete

Description

Completes the state of an active Task on a Journal component

Parameters

Name	Description
Journal	The Journal component that changes the state of the Task
Quest	The Quest asset reference
Task	The Task identifier from the Quest
Wait to Complete	Whether to wait until the Task finishes running its Instructions

Keywords

Mission Finish Finalize

TASK FAIL

Quests » Task Fail

Description

Fails the state of an active Task on a Journal component

Parameters

Name	Description
Journal	The Journal component that changes the state of the Task
Quest	The Quest asset reference
Task	The Task identifier from the Quest
Wait to Complete	Whether to wait until the Task finishes running its Instructions

Keywords

Mission Stop Restart

TASK VALUE

Quests » Task Value

Description

Sets, Adds or Subtracts a value from a Task

Parameters

Name	Description
Journal	The Journal component that changes the state of the Task
Quest	The Quest asset reference
Task	The Task identifier from the Quest

Keywords

Mission Increment Change Add Set Progress

5.6 User Interface

5.6.1 User Interface

The **Quests** module comes with a collection of components designed to streamline the creation of UI windows and elements.

All examples that come with the module have been created with them and are flexible to accommodate any type of window.

Quest List UI

This is one of the most important components and allows to display a list of **Quests** in a list fashion.

Quest List UI

The Journal field determines which component the quests are taken from.

The following fields act as filters to display those quests.

- The **Show** dropdown allows to display only quests that are in a particular state. For example, display only those that are complete and active.
- The Show Hidden toggle determines whether hidden quests should be displayed or not.
- The Hide Untracked determines if the quests that aren't tracked should be visible or not.
- The **Filter** dropdown allows to define whether to only display those quests that are present in a Global or Local List Variable. This is useful to display achievements or non-standard quests.

The **Content** field defines the Rect Transform where each prefab instance will be instantiated, for every visible quest.

Layout component

The **Content** value should contain an auto-layout component, such as Vertical Layout Group, Horizontal Layout Group or Grid Layout Group.

The **Prefab** is the prefab instantiated inside the *Content*. It must contain a **Quest UI** component, which is automatically configured by its parent.

Quest UI

This component is used in tandem with the **Quest List UI** to display a list of quests based on a set of rules and filters.

Quest UI

The **Title**, **Description**, **Color** and **Sprite** fields are all optional and reference the indexed quest's homonymous values.

The **Style Graphics** section contains a collection of color codes to change the graphics based on different conditions, such as whether a quest is *Active*, *Inactive*, *Completed*, *Tracked*, etc...

The **Active Elements** section defines a set of optional game objects that are activated/deactivated according to different conditions.

show a Tracking Bookmark

It is common to mark the currently tracked quest with an icon or a different color. You can do this by selecting a game object that contains a bookmark image, and drag and drop this element onto the Active if Tracking field.

This will deactivate the bookmark if the quest is not being tracked, and activate it otherwise.

The Interactive elements allow to define different types of interactions performed by the player.

For example, the Button Track field instructs a button to toggle the tracking state of the quest when clicked.

The Select Quest field allows to define a selection element as a button to select this particular quest.

Nore about Selections

More information about selecting quests and tasks below at the Selection UI section.

The **Show** and **Show Hidden** fields work exactly like the ones from **Quest List UI** but instead of quests, it refers to tasks.

The **Tasks Content** and the **Task Prefab** are two optional fields that allow to define a place where to list the tasks of this quest based on the previous filters.

Ask UI required

Just like the **Quest List UI** component requires a prefab with a **Quest UI** component to configure, the latter requires a prefab with a **Task UI** component.

Task UI

This component is very similar to Quest UI but instead of working with quests, it does work with tasks.

Task UI

As seen in the upper screenshot, most fields are exactly the same, and only a handful differ.

Selection UI

Upon selecting a quest, any Quests UI component with the Selection keyword will be automatically updated.

The components affected are:

- · Selected Quest UI
- · Selected Task UI

Both components have the exact same interface as **Quest UI** and **Task UI** respectively. But instead of targeting a specific quest or task, they target the currently selected one, and automatically change upon receiving any change or selecting a new one.

Points of Interest UI

The points of interest UI components are all related to the highlighting and location of specific Tasks and scene objects around the scene.

For example, displaying a minimap where dots appear around a certain radius, or floating indicators as an overlay over the camera.

MINIMAP UI

This component is used to display a rectangle and displays nearby points of interest within a certain radius.

Minimap UI

Each Prefab field must contain a Minimap Item UI component, which is configured by this component.

Changing Radius

The radius of the minimap can be changed at runtime, and can be increased when the player goes at a high speed, or even as an unlockable skill that allows to view further away.

COMPASS UI

The **Compass UI** is a thin line that usually appears at the top of the screen, and displays the direction of points of interest from the camera's point of view.

Compass UI

The **Character** field determines the origin of the compass, and the **Camera** field the forward direction to be considered.

Each **Prefab** field must contain a **Compass Item UI** component, which is automatically configured by this component.

INDICATORS UI

The **Indicators UI** component displays floating images on top of the interface that shows the exact position of the point of interest.

Indicators UI

The **Keep in Bounds** field determines whether indicators should stay at the edge of the screen when the world space instance is off-screen.

Each **Prefab** field must contain a **Indicator Item UI** component, which is automatically configured by this component.

5.7 Releases

5.7.1 Releases

2.3.7 (Latest)

eased October 31, 2023

This version breaks compatibility with previous versions and will only work with Game Creator 2.13.43 or higher.

Changes

Tasks: Better performance on Task property checks

Internal: Support for Core 2.13.42 version

Fixes

Quests: Destruction of Events with same Task ID

Condition: Group all Complete with latest core version
Condition: Group any Complete with latest core version

2.2.6

eased August 29, 2023

Fixes

· Quests: Destruction of Events with same Task ID

Condition: Group all Complete with latest core version
 Condition: Group any Complete with latest core version

2.2.5

eased June 13, 2023

Fixes

Variables: Quest type initialized in correct phase

2.2.4

New Hotspots: Allow to define a fade in/out distance Compass: Display distance with units Compass: Fade in/out based on distance/direction Minimap: Fade in/out based on distance Indicators: Fade in/out based on distance Property: Get Quest/Task Sprite Property: Get Quest/Task Color Settings: Displays current and update version Changes Signature format from Core 2.9.34

2.1.3



2.0.2

```
New

Trigger: On Task Value Change

Enhances

Editor: Quests remembers last selection

Fixes

Indicators: Wrong position when off-screen
```

2.0.1

Leased September 15, 2022

New

First release

6. Behavior

6.1 Behavior

Behavior

The **Behavior** module allows to easily create and manage all your game's intelligent agents using a wide variety of industry-standard tools:

- · State Machines
- · Behavior Trees
- GOAP
- · Utility AI

Choosing one or another is a matter of preference and what makes more sense. It's a the right tool for the job kind-of situation where there's not one definitive answer.

This documentation goes over them in detail from the most basic to the most complex systems.

Get Behavior



The Behavior module is an extension of Game Creator 2 and won't work without it

6.1.1 The Processor

The **Processor** component is responsible for executing the logic of any of the aforementioned AI systems and can be added to any game object in the scene, not just characters.

Processor component

This component has a **Graph** field which accepts a **State Machine**, **Behavior Tree**, **Action Plan** or **Utility Board** graph asset.

The Loop option determines whether the graph should be ran once or start over when it finishes executing.

The **Update** option determines whether the graph is executed every frame, manually via a script or at a custom interval, which can be specified using a dynamic property.

Whenever a graph is added it collects all the Blackboard parameters and displays them as fields, which can be dragged and dropped or set via visual scripting.

bisplay using a Parameter

If we create a parameter called <code>my-target</code> on a Blackboard and assign this graph onto a Processor component, it will display like in the screenshot below so its value can be assigned.

Processor Blackboard Parameters



Parameters can be changed at runtime but a Processor's graph cannot be changed when in play-mode.

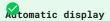
6.1.2 The Graph

All AI systems included in the Behavior module use a similar graph window with some common elements.

Graph Tools

The top toolbar's left side contains a button that allows to focus on the currently selected element(s) on the view as well as toggle the grid-mode. On the right side there's a collection of toggles that allow to show and hide other sections of the window.

The bottom toolbar tracks which graphs have been opened so you can quickly go back to editing a parent graph.



The breadcrumb toolbar will automatically be displayed whenever you edit a sub-graph asset from a parent graph asset.

The Blackboard

It's a collection of parameters with a name and a type that allow to interface between the agent running the AI system and the graph itself.

Patrolling

Let's say we have an AI system that requires guards to patrol an area, each having their own route. We can create a **Blackboard** entry called patrol-route and change its type to *Game Object*. By doing so, every guard that uses this AI system will have a *Patrol Route* field that can be used to define its individual route.

To create a new entry simply type in the name and press *Enter* or click the + button.

By default a parameter doesn't have any type and thus won't appear in the **Processor** component. To change its type click on the icon and select it from the dropdown.

Blackboard Entry Change Type

After creating or editing a parameter of a **Blackboard** you'll need to refresh the **Processors** using this graph so the parameters are re-sync.

Inspector

The Inspector panel allows to edit any nodes created inside the graph.

To edit a node simply select it and open the *Inspector* panel if it isn't already. Each node type will have its own configuration options.

Inspector

6.1.3 Parameters

After assigning an AI system to a **Processor** component it will display all available parameters from the **Blackboard** at the bottom of the component.

Processor with Parameters

These values can be set by dragging and dropping values from the scene or the project panel or using *Game Creator*'s **Visual Scripting**.

Changing Parameters

For example, let's say there's a parameter called energy and we can to subtract 1 unit from it every second. We can create a **Trigger** component with an *Interval* value of 1 second, and use the **Subtract Numbers** instruction.

Processor with Parameters

We assume it's the Player who has the **Processor** component, so we use it as the targeted game object and as the name of the parameter we use $\frac{1}{2}$ energy.

Each Property dropdown will have a Behavior/ section with the corresponding parameter value.

6.2 Setup

Welcome to getting started with the **Behavior** module. In this section you'll learn how to install this module and get started with the examples which it comes with.

6.2.1 Prepare your Project

Before installing the **Behavior** module, you'll need to either create a new Unity project or open an existing one.



It is important to note that Game Creator should be present before attempting to install any module.

6.2.2 Install the Behavior module

If you haven't purchased the **Behavior** module, head to the Asset Store product page and follow the steps to get a copy of this module.

Once you have bought it, click on Window Package Manager to reveal a window with all your available assets.

Type in the little search field the name of this package and it will prompt you to download and install the latest stable version. Follow the steps and wait till Unity finishes compiling your project.

6.2.3 Examples

We highly recommend checking the examples that come with the **Behavior** module. To install them, click on the *Game Creator* dropdown from the top toolbar and then the *Install* option.

The **Installer** window will appear and you'll be able to manage all examples and template assets you have in your project.

The **Behavior** module comes with four different AI systems, and each one has its own demos. If you're new to AI, we recommend starting in the following order, which is from the most basic to the most complex system.

- · State Machine: A very simple example of a patrolling guard using a finite state machine.
- · Behavior Tree: A behavior tree example of a patrolling guard playing hide and seek with the player
- GOAP: An example using goal-oriented action planning where characters work with each other to pick branches and keep a fire.
- Utility AI: An example that uses a needs-based AI system where characters go in a dance club, dance, drink and go home when they are tired.

Installer Behavior

Once you have the examples installed, click on the *Select* button or navigate to each one (for example Plugins/GameCreator/Installs/Behavior.StateMachine/).

6.3 State Machines

6.3.1 State Machines

State Machines (also known as Finite State Machines or FSM) are the most basic form of AI.

Simple State Machine

As its name implies, an entity can be in just one state at a time and can only transition to another state which is linked to the current one.

Simple FSM

For example, a very simple State Machine could define the behavior of a guard. We could define two states:

- Patrolling
- Attacking the Player

The guard would start in the *Patrolling* state and only transition to the *Attacking Player* state if the player is in sight.

- ${f \cdot}$ The ${f Nodes}$ section details all available node types.
- ullet The $oldsymbol{\mathsf{Logic}}$ page details how a $oldsymbol{\mathsf{State}}$ $oldsymbol{\mathsf{Machine}}$ works and how it's executed.

6.3.2 Nodes

There are 4 different node types and they are created by right clicking anywhere on the graph, apart from the **Enter** and **Exit** ones:

- The Enter node is unique and determines which State will be the first one when starting to run the graph.
- The **Exit** node is optional and allows the **State Machine** to *finish* running. Finishing running allows a **State Machine** graph to be used as a subgraph of another AI tool so it has a beginning and an end.

State

The State nodes are the backbone of a State Machine and is where the magic happens.

State in a State Machine

The **State** contains a **Name** field that allows giving the node a name. This has no effect on the execution and is just for information purposes.

The **Conditions** list determines whether this **State** can be transitioned to from another **State** with an edge pointing at this. If the conditions are not successful this **State** won't be transitioned to.

The Check field accepts two options:

- Every Cycle means that when this node is being executed, it will wait till its On Update instructions are completed before checking whether it can transition to another node.
- Every Frame means that when this node is being executed, it will check every frame if it can transition to another node.

Efault to Every Cycle

It is tempting to check every frame whether the state should transition to another one. However, checking Conditions comes with a (very small) performance overhead. We recommend using *Every Cycle* when possible, so Conditions are called less frequently.

The **On Enter** instructions are called whenever this node starts being executed because of a transition.

The On Exit instructions are called whenever this node finishes executing and transitions to another node.

The **On Update** instructions are called every frame while the node is being executed, and will restart again automatically if the node is still being executed after finishing running the instructions.



It is important to node that the **On Update** instructions can be interrupted at any moment because of a transition. It's better to add the initialization instructions on the **On Enter** and the post-run instructions on the **On Exit**, which are guaranteed to be executed from start to finish.

Conditions

Conditions serve as a gate to move from one state to another. If the conditions return false when attempting to switch states, the transition won't happen.

Why Conditions?

You might be wondering why there's a **Conditions** node when a **State** already has a *Conditions* list that do exactly the same thing.

This is because you might have multiple **States** that require the same *Conditions* and funnel them to a single output **State** node. In order to not repeating the same *Conditions* list on all **States** you can relay them to a single **Conditions** entry point.

Sub Graphs

Sub Graph nodes allow to encapsulate another graph as if it were a State.

Subgraph in a State Machine



It is important to node that a **Sub Graph** accepts any kind of graph, not just **State Machine** graphs. You can, for example, execute a **Behavior Tree** as a **State**.

Elbows

Elbows don't do anything and just allow to improve the readability of graphs by creating corners.

Their shape is determined by the direction after dragging and dropping an edge from another node, but it can also be changed again by selecting the **Elbow** node and changing the direction in the *Inspector*.

6.3.3 Logic

State Machines are one of the easiest AI tools to understand and work very intuitively.

There is a starting **State** node which connects to other **State(s)** in a specific order. This order can be seen selecting a **State** node and sorting the transitions at the bottom of the *Inspector* panel.

State Transitions in a State Machine

When a **State** is running, depending on the *Check* field value (which can be *Every Cycle* or *Every Frame*) it will check whether it can transition to another **State**.

The order in which it tries to change to another node is from top to bottom, and will move as soon as it finds a new suitable **State** that successfully passes its *Conditions* list.

If not a single connected **State** successfully passes the *Conditions*, the transition won't happen and the current **State** will remain as the running one.

Example

Here's an example of a **State Machine** with two states: One that makes a character patrol around and one that tracks the player when it's in its line of sight.

State Machine Running

Upon seeing the player, the current **Patrol** state transitions to the **Track Player** state. If the guard loses sight of the player, it transitions back to **Patrol**.

6.4 Behavior Trees

6.4.1 Behavior Trees

Behavior Trees are tree-like structures that stem from a single root node and are evaluated from top-to-bottom, and following a left-to-right priority order, having the right one the highest priority.

Simple Behavior Tree

Lenavior Trees vs State Machines

Although **State Machines** are better suited for simple AI systems, they can quickly become messy with lots of connection edges and hard to maintain. **Behavior Trees** are slightly more complex but offer much more flexibility and are easier to read at a glance when having lots of nodes.

- The Nodes section details all available node types.
- $\boldsymbol{\cdot}$ The \boldsymbol{Logic} page details how a $\boldsymbol{Behavior}$ \boldsymbol{Tree} works and how it's executed.

6.4.2 Nodes

There are 4 types of nodes in **Behavior Trees** plus a special node called **Entry**, which is a single node that can't be deleted and marks the root of the execution.

Tasks

A Task node is in charge of executing a specific set of Instructions when it runs.

It also contains a *Conditions* list, which is executed every time the graph is evaluated. If their value is not successful and the *Instructions* list are running it will return a \bigcap *Failure*.

While the Instructions are running, the **Task** node returns \bigcirc Running and upon finishing them it returns \bigcirc Success.

Task node of a Behavior Tree

A simple Task

For example if a Task node is a direct child of the Entry node and it waits 1 second before finishing, running the graph will put the whole graph under the \bigcap Running state and change to \bigcap Success after one second.

Task node of a Behavior Tree

Composites

Composite nodes allow to branch and determine the order in which its child nodes are executed. There are multiple types of composites, which can be chosen selecting the *Composite* node and clicking on the field in the *Inspector*.

Composite node of a Behavior Tree

SELECTOR

The **Selector** composite executes the left-most child first. If the conditions return a \bigcirc Success the composite node also returns \bigcirc Success.

However if the child returns \bigotimes Failure it attempts to execute the next child node.

Composite Selector type node of a Behavior Tree

If all nodes return \bigcirc Failure the composite also returns \bigcirc Failure.

Selector as an OR

The ${f Selector}$ composite type can be seen as an ${\it OR}$ operator with their children.

SEQUENCE

The **Sequence** composite executes the left-most child first and follows to the next if the execution of the previous one is \bigcirc Success.

If the execution of a child node returns \bigcirc Failure the composite node will also return a \bigcirc Failure.

Composite Sequence type node of a Behavior Tree

If all nodes return \bigcirc Success the composite also returns \bigcirc Success.



The Sequence composite type can be seen as an AND operator with their children.

PARALLEL

The **Parallel** composite, as its name implies, executes all of its children at the same time and it can configure when the composite should be considered as Success by choosing the option from the field that appears in the *Inspector*.

Composite Parallel type node of a Behavior Tree

All Successful

For example selecting All Successful will make the composite node return \bigcirc Success if and only if all of its children have finished with a \bigcirc Success result. Otherwise it will return \bigcirc Failure.

RANDOM SEQUENCE

The **Random Sequence** composite works exactly the same way as the **Sequence** composite, except that the order in which its children are evaluated is chosen at random right before the graph is evaluated.

RANDOM SELECTOR

The **Random Selector** composite works exactly the same way as the **Selector** composite, except that the order in which its children are evaluated is chosen at random right before the graph is evaluated.

Decorators

Decorator nodes don't *do* anything but can transform the results of its child node. For example, the **Invert** node returns \bigcirc Success or \bigcirc Failure depending on the value of its child node.

Decorators node of a Behavior Tree

There are also multiple **Decorator** types of nodes

FAIL

Returns \bigcirc Failure regardless of the result of its child.

SUCCESS

Returns $\hfill \bigcirc$ Success regardless of the result of its child.

RUNNING

Returns \bigcap Running regardless of the result of its child.

INVERT

Returns Success if its child node is Failure.

Returns \bigcap Failure if its child node is \bigcap Success.

Returns α Running otherwise.

REPEAT

Returns \bigcap Running as long as the amount of times that its child has ran is below a specific number.

In other words, allows to execute its child a certain amount of times before returning its last result.

WHILE FAIL

Returns \bigcirc Running as long as its child is either \bigcirc Running or \bigcirc Failure.

Returns \bigcirc Failure otherwise.

WHILE SUCCESS

Returns \bigcirc Running as long as its child is either \bigcirc Running or \bigcirc Success.

Returns \bigcirc Failure otherwise.

Sub Graph

A **Sub Graph** node allows to execute other **Behavior Trees** or even other types of AI systems, such as **State Machines**.

Sub Graph node of a Behavior Tree

This type of node behaves exactly the same as the **Task** node, except that instead of executing a collection of *Instructions* it executes another AI graph.

6.4.3 Logic

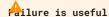
A **Behavior Tree** always starts its execution from the root **Entry** node and trickles down following a top-to-bottom fashion, looking for a single **Task** or **Sub Graph** node type.

Once one of these nodes has been found it executes them and the result of the execution bubbles up following the same path. During this phase, *Decorators* might change the returning value.

The return value of a node (and therefore, the return value of a behavior tree) can be:

- Success: The node has ran and successfully completed
- ullet Failure: The node has attempted to run but unsuccessfully completed
- $\boldsymbol{\cdot}$ $\boldsymbol{}$ $\boldsymbol{}$ Running: The node is running and has not finished

Apart from these states, a node can also be in a ready state, which means it has yet to be executed.



A common misconception is that a \nearrow Failure state is due to an error, which is not true. A \nearrow Failure state could be due to a character not being able to see the Player and moving on to checking another branch on a Selector node.

Every time a **Behavior Tree** is evaluated, whe whole tree structure is checked, taking into account the state of all nodes, which are carried over. For example, a node that has been \bigcirc Success ran won't run again unless the tree finishes with a \bigcirc Success or a \bigcirc Failure state.

Once a **Behavior Tree** has finished running and has a \bigcirc Success or \bigcirc Failure state it is considered as finished.

Ordering Nodes

Composite nodes can have multiple children branching from them. The order in which these are executed is denoted by a numeric value at the top of their children, and it's automatically calculated when moving a node around.

Ordering nodes in a Behavior Tree

The left-most branching node will be the first one, and the last one will be the one found at the right-most position.

6.5 GOAP

6.5.1 GOAP

GOAP systems, also known as **Goal Oriented Action Planning**, are an AI system that automatically builds **Plans** based on a list of **Requisites** and **Effects** that affects the agent's **Beliefs**.

The **Goal** of a **GOAP** system is to build **Plan** that changes the current **Beliefs** into one that satisfies a specific **Requisite**.

GOAP example Graph

- Plan: A sequence of nodes executed in order to achieve a goal.
- Beliefs: The current knowledge if the agent about the state of the world.
- Requisites: A collection of boolean conditions that need to be satisfied to run a node.
- Effects: A collection of changes that occur after executing a node.

Smilar to Domino

The best analogy to this AI system is by comparing it to the *Domino* game. Each piece has a number of dots on one side (which would be the **Requisites**) and a number of dots on the other side (**Effects**).

The **Beliefs** would be current number of dots required after placing a piece next to another one, and the **Plan** the sequence of pieces placed in order.

The beauty of **GOAP** is that you can add as many nodes as you want without drawing connections between them and the AI system will come up with the best plan possible. This has the drawback (or an advantage) that it may come up with a plan that isn't foreseen by the game designer.

If there are multiple possible **Plans** the AI system will always prioritize the one that has the lowest overall **Cost**.

Simple Example

Let's say we have the following Action Plan and we ask it to build a Plan that satisfies the Goal: be-in-hotel.

GOAP simple example Graph

There are two nodes, both of which have zero ${\it Requisites}$ but both have an ${\it Effect}$:

- The first one satisfies be-in-hotel
- The second one satisfies be-in-restaurant.

If we ask the **Action Plan** to build us a **Plan** it would give us the first node, because running it satisfies the **Goal** be-in-restaurant.

A more complex Example

Let's imagine we now have the same case scenario, but in order to enter the *Hotel* we need to walk through the *Restaurant*. This can be represented by adding the **Requisite** be-in-restaurant on the first node.

GOAP complex example Graph

If we ask the AI system to build us a plan, it will quickly realize the plan can't be made of a single node, because it requires be-in-restaurant to be satisfied. However, the second node satisfies this condition, and thus it will give us the **Plan** sequence: Task 2 Task 1

Thoughts

Beliefs are the agent's local knowledge about the world state at any given time, which can be transformed applying **Effects** in order to reach a goal.

By default when an **Action Plan** attempts to build a **Plan** it starts with a blank slate of **Beliefs**. However we can change that by giving it some default values whenever the plan starts being calculated.

GOAP Thoughts turn into Beliefs

To do so, select the **Action Plan** asset from the *Project Panel* and click on the *Add Thought* button. Thoughts are the initial values of an agent's **Beliefs**.

Tam thinking whether I see the Player or not

For example, we could have the initial **Belief** (or **Thought**) of whether the agent can see the player or not. To do so we can use the *Visual Scripting* boolean option of a **Condition** and check whether there's an obstacle between the *Player* and the *Self* (the agent itself).

GOAP Belief can see Player

6.5.2 Nodes

GOAP has two types of nodes:

- Tasks
- · Sub Graphs

Both of them accept a Requisites node on the left and an Effects node on the right.

Tasks

The Task node is the main node of a GOAP AI system.

GOAP Task node overview

It contains a Name that helps identify what this node does. This has no effect at runtime.

The **Cost** is a numeric value that determines how difficult it is to run this node. When coming up with a plan, the AI system will pick the plan with the least cost value, adding up all costs of all nodes in each plan and comparing the resulting value.

Conditions determine whether this node can be executed or not.

Onditions are not Beliefs

It is extremely important to note that **Conditions** are not the same as **Beliefs**. **Conditions** have no effect when coming up with a plan and will only be evaluated when executing a plan, and cancel the plan if they evaluate to False.

It's highly recommended using **Beliefs** instead of **Conditions** unless you want to abort your plans mid-way through. For example, a plan that approaches a character and attacks it, an aborting **Condition** could be checking if the character is already dead when reaching it.

The **On Execute** instructions are where the bulk of the logic happens. A node will be considered finished after all its instructions have been executed in order.

Sub Graph

The **Sub Graph** node works exactly the same way as a **Task** node does, but instead of executing instructions, it runs another AI graph system, such as a **State Machine** or a **Behavior Tree**.

GOAP Sub Graph node overview

Requisites and Effects

The **Requisites** and **Effects** nodes are exactly the same, but the first one links to a **Task** or **Sub Graph** node from their left side, and the former links to them on the right side.

GOAP Effects node overview

There can be any number of **Beliefs** attached to each one of these nodes, and it can check whether the **Belief** is *True* or *False*.

A False Belief

A **Belief** that is marked as *False* means the belief is not present, which is the default value when coming up with a plan (unless the Thoughts list states otherwise).

6.5.3 Logic

A GOAP AI system doesn't do anything by itself. It requires one or multiple Goals and come up with a plan to solve them and pick the most beneficial.

Setting up Goals

The instruction Add Goal allows to add a new one to a specific Processor component and Action Plan.



The Action Plan asset must be specified because the AI system might contain multiple Action Plan assets running as sub graphs.

GOAP Add Goal Instruction

The Name field determines the Belief that will be required to satisfy (meaning it evaluates to *True*) in order to consider the Goal as completed.

The **Weight** is a value that is multiplied to the total cost when coming up with a plan that resolves this goal. This allows to prioritize some goals higher than others.

Goal Priorities

Let's say we have two goals: Stay-Alive and Kill-Enemies and the AI system comes up with a plan for each with the following total costs:

- Stay-Alive With a total Cost of 5
- Kill-Enemies with a total Cost of 3

In this case, because killing enemies has a lower cost, it would be picked up over staying alive. However this doesn't make sense, because a character should focus on self-preservation first.

In this case, we can give the Stay-Alive goal a Weight of 0.5, which is a coefficient that multiplies the total cost by this value, yielding the following new results:

- Stay-Alive with a total Cost of (5 * 0.5) = 2.5
- Kill-Enemies with a total Cost of (3 * 1) = 3

Now staying alive will be prioritized unless killing an enemy comes at a very small cost. Note that we could have also doubled the weight of the killing enemies goal instead and it would result still work.

Just like adding goals, to remove a Goal use the Remove Goal instruction.

Planning

Once an **Action Plan** has one or more **Goals** it can come up with a plan, which is only done if there's not an ongoing plan being executed.

The GOAP algorithm starts coming up with a plan for each Goal, and if there are multiple plans for the same goal, stores the one with the lowest cost.

GOAP running a plan

Once it has zero or one plan for each **Goal** it multiplies each one by their **Weight** value and picks the one with the lowest cost. After than it starts executing the chosen plan until it finishes or is aborted by a *Condition*.

6.6 Utility AI

6.6.1 Utility AI

Utility AI, also known as **Needs-Based AI**, is a behavioral artificial intelligence technique that defines a collection of *needs* that have a necessity curve assigned to them and a **Score** value that can grow or decrease over time.



This AI technique is mostly suited for organic simulation games, such as The Sims.

At any point the *need* task to use is determined by comparing all **Scores** and applying the easing curve to it. The *need* task with the highest resulting value will be picked and executed.

Two Needs

Let's say we have an Utility Board with just two needs, with very different curves:

Cook some Rice: The need to cook and eat grows over time, as the hunger increases.

Call a Friend: The need to call a friend decreases as the socialization value is met.

Note that the **Call a Friend** curve slightly increases a bit at the end. This is because once a character is socializing, it keeps wanting to socialize, unlike with hunger, where once a human is full, it doesn't want to eat anymore.

Utility AI graph example

If we were to execute this graph, if the hunger level was high it would likely pick the **Cook some Rice** task. However if the socialization needs are very low it would likely pick the option to **Call a Friend**.

Curves allow to tweak the resulting value and model different responses based on a set of dynamic values that change over time.

Game Maker's Toolkit on Utility AI

Mark Brown from Game Maker's Toolkit put up a very nice and rounded video covering how the AI of *The Sims* work. It's filled with good ideas and high-level knowledge that designers can use to get started.

Game Maker's Toolkit video on Utility AI



Utility AI pairs very well with other AI systems, such as Behavior Trees or GOAP. Each need node can contain other sub graphs from other systems that delegate the complexity of executing the action that fulfills the need.

6.6.2 Nodes

Because Utility AI systems don't have any connections, there are just two possible types of nodes. Both of them do the exact same thing but one has the logic embedded on them and the other one delegates it to another AI system, such as a Behavior Tree, a GOAP, a State Machine or even another Utility AI system.

Tasks

The Task node has the logic of the need embedded on itself and is suitable for basic interactions, such as playing a guitar.

Utility AI Task node

The top part of the node contains a Name field which doesn't have any effect on the execution and is used to easily identify it across all other nodes.

The Score section determines the easing Curve as well as the value used as input on the curve.



The bounds of the Value

By default the bounds of the value range from 0 to 1, meaning that the curve's left-most position is zero and the right-most is 1.

However if your game requires negative values or another range of values, you can modify the minimum and maximum value by selecting the Utility Board asset on the Project Panel and changing the corresponding fields.

Utility AI Minimum and Maximum limits for Score

Bear in mind that the curve will always be constrained by 0 an 1, but the resulting value will be transformed to fit the limits defined in the Minimum and Maximum fields.

Curves allow to transform the input Value into something that's usable. Here are some common use-case scenarios:

LINEAR CURVE

Utility AI Linear Curve

The most basic curve is the linear one, which takes the input Value and returns it as-is.

INVERSE LINEAR CURVE

Utility AI Inverse Linear Curve

The inverse linear curve reverses the value given, meaning that the resulting value decreases as the input value increases, and vice-versa.

EASING CURVE

Utility AI Easing Curve

The smooth easing curve is similar to the linear curve because it grows as the input value grows. However the growth is much more visible as it approaches the end of the range.

BELLY CURVE

Utility AI Belly Curve

The belly curve (aka Bell curve or Gauss curve) is a very interesting one in which the growth happens in the middle but decreases at the edges. This is usually used for low-priority tasks that are executed out of boredom, such as scratching one's head.

The **Conditions** field allows to determine whether this *need* is valid and can be executed. If the conditions can't be satisfied, the node will not be taken into account.

The **On Execute** instructions run as soon as the *need* node starts executing. After it finishes, the task is considered to have finished.

Because some tasks might exit early due to unforeseen events, such as attempting to play a guitar that no longer exists, the **On Exit** instructions are executed after finishing running a *need*. These instructions are guaranteed to be called from start to finish and can be used to finalize the execution and restore the state of the agent back to before it started.

Sub Graph

The **Sub Graph** node works exactly the same way as the **Task** one, but instead of executing some **On Execute** and **On Exit** instructions, it delegates the responsibility of running the *need* node to another AI system, such as a **Behavior Tree**, a **State Machine**, an **Action Plan** or even another **Utility Board**.

Utility AI Sub Graph node

6.6.3 Logic

When an Utility Board is executed it starts by gathering all Task and Sub Graph nodes.

It then gets their different **Value** values and applies them to their corresponding **Need Curve**, which returns a value constrained between the **Minimum** and **Maximum** value defined in the **Utility Board** asset.

Once their **Score** value is calculated, it compares all values and picks the one with the highest score and starts running it.



An **Utility Board** will not recalculate a new *need* node unless the one currently being executed finishes. This means that when a *need* task cannot be completed, it should either early exit or abort itself.

Utility AI running

When entering Play-Mode and selecting an agent that has a **Processor** component will display real-time information about the current execution. In the screenshot above the title of each node shows the resulting value of applying the **Value** onto the **Curve**:

- Go Home has a value of 0.072
- Drink has a value of 6.48
- · Dance has a value of 9.856

Thus the node chosen to execute the AI system is Dance because has the highest score value.

6.7 Visual Scripting

6.7.1 Visual Scripting

The **Behavior** module symbiotically works with **Game Creator** and the rest of its modules using its visual scripting tools.

- · Instructions
- · Conditions
- Events

Each scripting node allows other modules to use any **Behavior** feature.

6.7.2 Conditions

Conditions

SUB CATEGORIES

• Behavior

Behavior

BEHAVIOR

Conditions

• Is Processor Running

IS PROCESSOR RUNNING

Behavior » Is Processor Running

Description

Returns true if the Processor is in a Running state

Parameters

Name Description

Processor The reference to the Processor component

Keywords

AI Behavior Tree State Machine Utility Need Goal Plan GOAP

6.7.3 Events

Events

SUB CATEGORIES

• Behavior

Behavior

BEHAVIOR

Events

- On Processor Finish
- On Processor Start

ON PROCESSOR FINISH

Behavior » On Processor Finish

Description

Executed when the Processor finishes its current execution

Keywords

AI Behavior Tree State Machine Utility Need Goal Plan GOAP

ON PROCESSOR START

Behavior » On Processor Start

Description

Executed when the Processor starts a new execution

Keywords

AI Behavior Tree State Machine Utility Need Goal Plan GOAP

6.7.4 Instructions

Instructions

SUB CATEGORIES

• Behavior

Behavior

BEHAVIOR

Sub Categories

• Action Plan

Instructions

• Processor Update

PROCESSOR UPDATE

Behavior » Processor Update

Description

Manually executes a new iteration on a Processor

Parameters

Name Description

Processor The targeted Processor component

Keywords

AI Behavior Tree State Machine Utility Need Goal Plan GOAP

ACTION PLAN

Action Plan

Instructions

- · Add Goal
- Remove Goal

Add Goal

Behavior » Action Plan » Add Goal

Description

Adds a new Goal to the specified Action Plan

Parameters

Name	Description
Processor	The targeted Processor component
Action Plan	The Action Plan asset to set the goal
Name	Name identifier of the goal
Weight	The weight the goal has when calculating the plan
Name	Name identifier of the goal

Keywords

AI Action Goal Plan GOAP

Remove Goal

Behavior » Action Plan » Remove Goal

Description

Removes an existing Goal from the specified Action Plan

Parameters

Name	Description
Processor	The targeted Processor component
Action Plan	The Action Plan asset to remove the goal
Name	Name identifier of the goal
Weight	The weight the goal has when calculating the plan
Name	Name identifier of the goal

Keywords

AI Action Goal Plan GOAP

6.8 Releases

6.8.1 Releases

2.1.3 (Latest)

eased October 31, 2023

This version breaks compatibility with previous versions and will only work with Game Creator 2.13.43 or higher.

Changes

Demos: Examples support latest core version Internal: Support for Core 2.13.42 version

2.0.2

eased August 31, 2023

Enhances

· UX: Paste from shortcut places node nearby

· UX: Creating a node automatically selects it

Fixes

· Graph: Duplicating special nodes

· Example: Removed State Machine unused object

2.0.1

eased August 30, 2023

New

· First release

7. Perception

7.1 Perception

QuePerceptionsts

Video game characters can read their world through a wide variety of virtual sensors, such as sight, hearing and smell. These sensors have parametrized values that try to emulate the ones from the real world.

The **Perception** module aims to help non-playable characters understand what happens around them and organically react to different stimulus.

Get Perception



The Perception module is an extension of Game Creator 2 and won't work without it

The **Perception** component can be attached to any game object in order to make it aware of their surroundings. This component admits four different sensors that help read the world around them:

- · Sight: Detects other objects within a vision cone and its peripheral vision
- · Hearing: Detects noises above a certain threshold
- Smell: Tracks scent(s) and the direction where they come from
- Feel: A sixth sense that detects other objects by proximity

Using the sensors above, the **Perception** component can track a wide variety of scene objects and keeps a log of how aware it is of them.

Moreover the **Perception** module comes with an **Evidence** system that allows agents to notice changes made to the world by other characters.

7.2 Setup

Welcome to getting started with the **Perception** module. In this section you'll learn how to install this module and get started with the examples which it comes with.

7.2.1 Prepare your Project

Before installing the **Perception** module, you'll need to either create a new Unity project or open an existing one.



It is important to note that Game Creator should be present before attempting to install any module.

7.2.2 Install the Perception module

If you haven't purchased the **Perception** module, head to the Asset Store product page and follow the steps to get a copy of this module.

Once you have bought it, click on Window Package Manager to reveal a window with all your available assets.

Type in the little search field the name of this package and it will prompt you to download and install the latest stable version. Follow the steps and wait till Unity finishes compiling your project.

7.2.3 Examples

We highly recommend checking the examples that come with the **Perception** module. To install them, click on the *Game Creator* dropdown from the top toolbar and then the *Install* option.

The **Installer** window will appear and you'll be able to manage all examples and template assets you have in your project.

- Examples: A collection of scenes with different use-case scenarios.
- UI: A bundle of common user interface elements, such as enemy awareness, luminance, smell and hearing.

Installer Perception

The **Examples** requires all the skins in order to work.



Clicking on the Examples install button will install all dependencies automatically.

Once you have the examples installed, click on the *Select* button or navigate to Plugins/GameCreator/Installs/Perception.Examples/.

Perception Examples

7.3 Awareness

Even though there are multiple ways to detect characters and objects, the ultimate goal is to determine whether an object A is aware of another B or not.

The Awareness is a value that aims to simplify how an agent communicates and reacts with other game objects.

Perception Awareness

Example of Awareness

Let's say we have an enemy agent that can detect the player using *Sight*, *Hearing* and *Smell*. It would be very difficult to react when the enemy detects the player if we didn't have the **Awareness** meter, because we would need to control what happens when any of the possible combination of all three sensors detect or not the player.

For example, what would happen if the enemy sees the player? What happens if the enemy can see the player, but also hear it? What if it hears it but can't see it? Any of these permutations increase the complexity of building a robust detection system.

In this module the **Awareness** is a unique value stored in the **Perception** component that tracks how aware the agent is of each tracked object. This value ranges between 0 (not aware at all) and 1 (fully aware of the target).

The **Perception** component has a list of **Sensors** that feed the awareness value independently from each other. By doing so, one can create an enemy AI that reacts according to how aware it is of the player, instead of relying on information from each of the sensors.

Awareness relation with Sensors

Mareness as Stages

The **Awareness** can be read as a value between 0 and 1, but in some cases, it may be easier to give a name to awareness ranges. The **Perception** module provides 4 ranges:

None: The Awareness value is between 0 and 0.05

Suspicious: The Awareness value is between 0.05 and 0.5

Alert: The Awareness value is between 0.5 and 0.95

 $\textbf{Aware}\colon$ The Awareness value is between 0.95 and 1

7.3.1 Tracking Objects

The **Perception** component does not automatically track all scene objects. The user can define which objects are tracked (or untracked) at any moment using the **Track Awareness** and **Untrack Awareness** instructions respectively.

Tracking the Player

The most common use-case is for an enemy to track the Player. To do so, simply add a **Trigger** with the *On Start* event with the following **Track Awareness** instruction:

Perception Track Object

Once an object is being tracked it will appear at the bottom of the **Perception** component during play-mode along with a progress bar. This is useful for debugging how aware the component is of the tracked game object.

Perception Debug Awareness

7.3.2 Increasing Awareness

One a Perception component tracks a game object there are multiple ways to increase its Awareness. The easiest one is using the Increase Awareness instruction, which apart from incrementing the awareness, it also allows to cap its increment up to a certain maximum.

Increase Awareness Instruction



The Max Level field allows to determine a maximum level in which the Awareness can increase. This is useful, for example, when the player breaks a crystal glass.

In this case, the enemy will increase its awareness, but even if the player breaks more glasses, the enemy will never reach the point of highest awareness, until it has a clear line of sight.

The other way to increase Awareness is using the Sensors, where each one works differently.

7.3.3 Forgetting Awareness

A Perception component requires a rather frequent stream of stimulus in order to keep its awareness on tracked game objects. The Perception component has the following fields that helps control how it behaves when stimulus stop being received:

- Forget Speed: The speed at which Awareness decreases over time.
- Forget Delay: The minimum amount of seconds it takes without receiving any stimulus before the Awareness starts decreasing.

Perception Forget Speed and Forget Delay

The exception is when the Perception reaches the Aware stage. In this case, if the Can Forget field is disabled, the Perception will never decrease its Awareness of the game object.

However if the Can Forget field is ticked, instead of the Forget Delay it will require the amount of seconds specified in the Duration field in order to start decreasing the Awareness.



www. Aware stage is different

The Aware stage can have a different amount of seconds in order to forget the specified game object because in most games, an enemy being aware of the Player means entering combat mode.

When characters are in combat, the player is likely to seek cover, and enemies could easily lose sight of the player. To avoid enemies being too forgetful it's a good practice to add a high amount of seconds before enemies start forgetting the player was seen.

7.3.4 Relaying Awareness

Perceptions can also send Awareness information to other Perception components in order to make them appear like they work as a team.

Inform when the Player is spotted

A common practice of an enemy spotting the player is to inform nearby guards that the player has been found, raising their **Awareness**. This can be easily done using the **Relay Awareness** instruction, which transfers all **Awareness** knowledge from one **Perception** component to another, without the second one losing any knowledge.

Relay Awareness between Perception components

7.4 Sensors

7.4.1 Sensors

The **Perception** component can make use of different sensors which emulate real-world senses.

Perception component with sensors

- Sight: Use a vision cone.
- **Hearing**: Hear noises made by other entities.
- Smell: Detect and track scents emitted by entities.
- Feel: Sense other entities by proximity.

7.4.2 Sight

The **Sight** is one of the most useful sensors of the **Perception** component and it allows to check whether there's a line of sight between the agent and another tracked object.

Settings

Perception Sight Settings

The **Update** field determines whether the vision cone is updated every frame, at a specified interval, or manually done (using a custom *Instruction* or a script). By default we recommend leaving it to *Every Frame* unless there are performance reasons.

The **Detection Speed** determines how fast the **Awareness** value increases when the tracked object is in its line of sight.

Aximum Detection Speed

Note that it is possible that the detection speed during runtime is slower than **Detection Speed**. This is because this field determines the maximum speed at which the **Perception** component will increase awareness, but there might be factors, such as low light and distance, that dampens the detection speed.

The **Layer Mask** is a physics mask used during the ray-cast phase and allows to determine what is an obstacle between the **Perception** component and the tracked object, and what is not.

The **Optics** field references a humanoid or generic bone from the game object with the **Perception** component, and it's the position where the *eye* should be. Characters will always ray-cast to the center of the tracked game object. When tracking characters, such as the player, the position is also the hips.

Eyes on the hips

Notice that by default, characters see the world from their hips. This is because it's easier to hide from enemies when their eyes are at hip-level.

The **Use Luminance** checkbox determines whether light conditions affect the detection speed or not. Unless you're making a stealth game, we recommend leaving the checkbox unticked, since it slightly increases performance.

If this field is ticked, two new fields will appear below:

- **Dim Threshold**: The minimum amount of luminance required for the **Perception** component to barely see a tracked object.
- Lit Threshold: The maximum amount of luminance required for the Perception to detect tracked objects at full speed.

Example of Dim and Lit thresholds

Let's say we have a **Perception** component with:

Dim Threshold = 0.2

Lit Threshold = 0.5

This means that any tracked object will require to be illuminated with an intensity of at least **0.2** in order for this **Perception** component to detect it. However the detection speed will be very slow, and will gradually increase until it's illuminated with an intensity of **0.5**. Higher light intensities won't further increase detection speed.

Mght Vision

Thanks to the **Dim** and **Lit** thresholds it's very easy to make enemies wear night vision goggles. You can simply turn the **Dim Threshold** all the way down to zero and **Lit Threshold** to something very small, like 0.1. This should be enough to detect the player in poor light conditions.

For more information about setting up lights when tracking objects, see the Luminance page.

Primary & Peripheral

At the bottom of the **Sight** sensor has two expandable boxes called **Primary** and **Peripheral** which represent the primary and peripheral vision cone.

The **Primary** cone is where the **Perception** can fully detect tracked objects. However if the tracked object is inside the **Peripheral** cone the speed at which it is detect is reduced by proportionally.

This means that the further away it is from the Primary cone, the slower the detection speed will be.

Perception Primary and Peripheral Vision

Both boxes contain identical fields:

- Angle: The angle extension in front of the Perception component.
- Radius: How far the vision cone extends.
- Height: The vertical size of the vision cone.

Perception Primary and Peripheral Cone

You can see in real-time a gizmo representing the **Primary** and **Peripheral** vision cones in the scene view.

Pripheral extends Primary

The **Peripheral** values extend the **Primary** ones. That means that if the **Primary Radius** is 5 and **Peripheral Radius** is 2, peripheral's radius is actually 5 + 2 = 7.

7.4.3 Hearing

The Hearing sensor represents the ability of a Perception component to hear noises above a certain threshold.

Settings

Perception Hearing Settings

The Use Obstruction field determines whether sound can be blocked by Obstruction components. Ticking this field will reveal a layer mask that determines which Obstruction components are considered as such.



Not using Obstructions

If your game is set in open spaces or you don't need noises to be blocked by nearby rooms you can untick the Use Obstruction field. Enabling this option comes with a slight performance overhead.

More information about these components can be found in the Obstruction page.

The Min Intensity and Max Intensity fields determine the range in which noises can be heard. Lower and higher noises will not be acknowledged by the Perception component.

Using intensities

It is known that dogs can hear higher pitched noises than human. You could model a situation where the player can use high intensity noises that only dogs can hear in order to distract them, without alerting other guards.

Decay Time is the amount of seconds a noise takes to fade out. Although noises are emitted as a one-shot effect, the hearing of a Perception components keeps hearing that noise due to the resonance inside a human's skull. This is more commonly known as Tinnitus effect in high-pitched noises.

Emitting a Noise

In order to emit a Noise the instruction Emit Noise can be used.

Perception Hearing Emit Noise

The Position field determines the origin of the noise emitted, while the Radius indicates how far it reaches.

The Intensity determines whether the noise stimulus can be heard by the Perception component or not. And the Tag is a custom value that can be used when building the AI to react differently depending on its value.

A whistle vs a gun shot

We could conclude that when an enemy hears a whistle it will change into a suspicious mode and approach the origin of the noise. However if it hears a gun shot it may turn into aware mode.

Reacting to Noises

The Perception component doesn't inherently increase its Awareness upon hearing a noise, unlike the sight sensor. This is because each noise tag can mean something different.

To react to noises a Trigger must be used with the On Hear event. Upon hearing any or a specific noise, the resulting reaction can be specified, like increasing the Awareness of the Perception component towards the player.

Trigger Hearing Noise



Ambient Noise)

It is worth noting that noises can also be masked by the \dim or ambient noise. The \dim is a more advanced concept and it is covered in depth in the Din section.

7.4.4 Smell

The **Smell** sensor works by emitting a **Scent** at a specific point with a *tag*. Emitting more scents links them together creating a chain similar to a breadcrumb trail that **Perception** components can pick to react or follow.

Settings

Perception Smell Settings

The **Update** field allows to choose whether the **Perception** component updates its *smell* sensor every frame, at a specific interval, or manually using a custom script.



If you are starting out we recommend leaving the update mode to *Every Frame*. *Interval* and *Manual* can be used to improve performance at the expense of precision.

The **Radius** determines how far the **Perception** component can smell a scent. Increasing this value allows agents to smell farther scents.

The Use Obstruction field allows scents to be blocked by Obstruction components.

Without Obstruction

If your game is set on on an open area or don't want scents to be blocked by **Obstruction** components, untick the **Use Obstruction** checkbox. It will save some performance overhead.

The **Min Intensity** field allows to define a minimum threshold at which scents are caught. See the **Emitting** Scents section for more information about how **scents** work.

Emitting Scents

Scents are one or more points in space connected by their lifetime value, identified by a tag name.

Each time a new **Scent** is emitted, it is connected to the previous scent with the same tag, allowing to traverse the scent trail from the oldest to the newest point.

Perception Smell Scent Trail

The instruction **Emit Scent** can be used to emit a new scent at a specific point.

Perception Smell Scent Trail

The **Source** field is a game object emitting the scent and its position is equal to that game object's point in space.

The **Tag** field is a string that identifies the scent. If there already is one or more scents with the same tag, they will be automatically connected forming a trail of breadcrumbs that other characters can track.

The Duration field specifies the time it takes to dispel the scent.

The Intensity field is a numeric value that indicates how strong the scent is when it starts.

Attensity fades over time

Note that the **Intensity** of a scent fades over time, similar to how gas dispels in real-life. This means that a scent with an intensity of 1 and a duration of 5 seconds will have an intensity of 0.5 after 2.5 seconds have passed.

Catching Scents

The **Perception** component doesn't inherently increase its **Awareness** upon smelling a scent, unlike the sight sensor. This is because each scent *tag* can mean something different.

To react to scents the **On Smell** Trigger can be used, which is executed whenever the **Perception** component enters a zone with a scent.

Perception On Smell Trigger

Once a **Perception** component catches a whiff of a scent it can increase the **Awareness** using the **Increment Awareness** instruction or even follow the scent trail.

Following Scents

A character can query the current point in space of a specific scent, as well as the next scent point (if it exists), using the **Position** property.

This allows characters to follow a scent trail by attempting to go to the next scent point from the one whiffed.

Move To...

Perception Move To with Follow Scent

For example you can use the Move Character To instruction with the Follow Scent option, which selects the next scent point from the closest one, effectively making the character follow the scent like a trail of breadcrumbs.

7.4.5 Feel

The Feel sensor works similar to sight sensor but it uses the entire space around the Perception component.



The **Feel** sensor is meant to be used as a sixth sense in which characters can *feel* another character being very close, almost touching, the character.

However you can also tailor it to your own needs. For example you can increase the radius and change the update mode to regular intervals and simulate a sonar.

Settings

Perception component sensor Feel

The **Update** field determines when the sensor is updated, which can be either *Every Frame*, at a regular *Interval* or manually from an external script.

Update every frame

If you are starting out we recommend leaving the update mode to *Every Frame*. *Interval* and *Manual* can be used to improve performance at the expense of precision.

The **Detection Speed** is at which rate a tracked object will increase the **Perception**'s **Awareness** when being within the radius.

The **Radius** field determines how far the sixth sense reaches, and the **Layer Mask** allows to filter out (or not) obstacles between the **Perception** component and the tracked object(s).

Automatic Awareness

When a tracked object is inside the detection radius of the **Feel** sensor it will automatically start increasing the **Awareness** meter of the **Perception** component.

7.5 Luminance

The **Sight** sensor of a **Perception** component has the option to **Use Luminance** in order to account for an object's reflected light when determining whether it is visible or not, and how much should the **Awareness** increase based on its intensity.

Luminance

Unity uses the **Light** component to specify light entities, which can take different shapes (Spot lights, Directional lights and Point lights) as well as affecting the scene during the light baking process or in real-time.

7.5.1 Component

The **Perception** module supports all these shapes and lighting modes through the use of the **Luminance** system, which is a component attached to any light source which indicates that this specific light should affect how visible an object is in the scene.

Luminance and Light components

The Luminance component will automatically detect the type of light and shape.

Luminance on Lights

As a general rule of thumb, all **Light** components should also have a **Luminance** component attached to them. However you might also want some lights not to affect player's visibility.

For example, if the player puts on some night vision goggles, you could enable a spotlight in front of it with a green tint that doesn't have any **Luminance** component attached to it. This will prevent this light from affecting visibility.

The Multiplier field allows to modify the luminance value compared to the Light's intensity.

Dim Light

For example, let's say our game is set in a very dark room and one of the light sources is a small lamp. But because it's very dark, the intensity of the light must be increased, but we would like the **Luminance** value to stay low.

In that case we can use a 0.5 multiplier so that if our **Light** intensity is 2, the resulting luminance value is 2 * 0.5 = 1.

The Layer Mask allows to pick which objects do affect luminance. It is important to note that luminance is calculated based on *colliders*, not on general geometry. If an object (like a fence) doesn't have a collider, although light will be blocked by the opposite side of the fence, the luminance value will still reach the other side.

Add colliders to geometry

It's advised to add colliders to every piece of geometry in the level so that both light and luminance result in similar results.

*Pansparent geometry

If you have a glass door in your game, light will pass through it and hence luminance should lit objects behind it. In order to achieve this, simply place the collider of the glass door in a **Layer** that isn't affected by the **Luminance**'s **Layer Mask**.

7.5.2 Environment Luminance

The **Environmental Luminance** can also be set using the **Change Global Luminance** instruction. This instruction changes the minimum light intensity the scene has, even there are no lights in the scene.

This is useful if the Light Window has a bright environment light color or sky-box.

7.5.3 Checking Luminance

Once all **Luminance** components are set up the resulting **Luminance** value can be queried at any point in the scene using the **Luminance** property, which is a decimal value that represents the intensity of the overall light hitting that spot.

Luminance Property



There is a UI component that allows to display UI information about the **Luminance** in real-time to the player. For more information, visit the **Luminance** UI section.

7.6 Camouflage

The **Camouflage** component can be attached to any game object in order to dampen their visibility when observed by other **Perception** components.

Luminance

The **Sight Damping** field is a coefficient that multiplies the resulting visibility value of a **Perception** component tracking this game object.

camouflage Calculation

Let's say an enemy is tracking the Player character and it's inside its *primary* vision cone. This means that the character will fully see the character if the light conditions are sufficient. For the sake of this example, let's say there's a dim global light with an intensity of 0.5, so the resulting visibility of the player is 0.5.

If the player has a **Camouflage** component with a **Sight Damping** value of 0.25, the resulting visibility of the player will be 0.5 * 0.25 = 0.125. If the *Dim* threshold is a bigger number than the resulting visibility, the player will be invisible to the enemy.

namic Camouflage

Because the **Sight Damping** value is a dynamic property, you have the flexibility to create a wide variety of game mechanics.

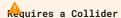
- You can bind the value to a Local Variable to turn on or off their invisibility skill.
- You can also bind the property to a Stat to increase their stealth skill as the player levels up.
- You can bind it to the player's movement, so that when it's standing still the player is less visible than when moving.

7.7 Obstruction

Some **Perception** sensors can be partially or completely blocked by scene obstacles, like sound-proof rooms or frozen-glass walls.

The **Obstruction** component allows to dampen the amount of *Sight* and *Hearing* of visual and audible stimulus passing through the object.

Obstruction



The **Obstruction** component requires a Collider component in order to work.

The **Noise Damping** field defines a coefficient that multiplies the resulting noise intensity heard by a **Perception** component when listening for a noise at the other side of one or multiple **Obstruction** components.

The Sight Damping does exactly the same for the Perception's sight sensor.

Obstruction Dampen Sight and Hearing

Example

A **Noise Damping** of 0.5 means that any noise coming from the other side of an **Obstruction** component will be muffled by half. If the **Noise Damping** is set to zero the noise is completely muffled.

blocking Sight

The Sight sensor already gets blocked by collider geometry. The **Obstruction** component should only be added if you want to partially block objects at the other size. For example, for semi-transparent walls.

7.8 Din (Ambient Noise)

There is an exception to when a **Perception** will not hear a noise emitted, even though it's between the **Min Intensity** and **Max Intensity** range, which is when the **din** (also commonly known as *Ambient Noise*) is loud enough to mask the noise emitted.

Thunderclap vs Whistle

Let's say we have an agent with a **Perception** component that can hear noises with an intensity between θ and 1. If the Player emits a noise with an intensity of θ .5, the agent should be able to hear the noise if it is within the noise radius.

However let's say there is a thunderclap at the same time, which temporally changes the din (or ambient noise) to 0.75. In this case, the whistle won't be heard because the **Perception** component will be deaf to noises between 0 (**Min Intensity**) and 0.75 (current din). However any noise above 0.75 and below 1 will be heard.

The scope of a din source can either be local or global.

Global Din

Global Din affects the entire scene equally and can be set using the **Set Global Din** instruction. A **Duration** can be specified that determines the time it takes to change from the current value to the target one.

Global Din Instruction

Local Din

The Local Din affects only a region of the scene and it is configured using the Din component.

Local Din Component

The **Radius** field determines how far the ambient noise reaches, fading out the intensity at the edges of the sphere.

The **Roll off** slider allows to change how slow or fast the intensity fades. Setting the slider in the middle means the intensity fades out linearly, while moving the knot to the left will take longer for the intensity to fade the closer to the edges the **Perception** component is.

Anear when possible

It is recommended to leave the **Roll off** value at 6. Although in real life sound propagates logarithmically, players will feel more comfortable knowing that the **din** intensity changes linearly the closer they are to the source of the ambient noise.

The **Use Obstruction** field allows to use **Obstruction** components to block the ambient noise from affecting nearby sound-proof rooms.

The **Level** field is the intensity at which the **din** is at its highest point, gradually fading out towards the perimeter of the sphere.

To more consistently represent how sound and ambient noise affects the computed **din**, there is an **Audio Source** box field at the bottom of the component. This optional field allows to play an audio clip at a volume equal to the component's **din**.

This makes it much easier to synchronize the audio clips played as noise masks and the din values.

7.9 Fyidence

The **Perception** module comes with a unique **Evidence** system that allows scene agents to detect changes in objects made by other characters (like the player) and react accordingly.



Let's say we are making a stealth game where the player is sneaking through a series of rooms. If the player leaves a door open, guards may notice this change and react accordingly. For example, raising their awareness.

The Evidence component can be added to any game object.

Evidence

The Tag field is a string that identifies the Evidence type.

Multiple Evidences with the same Tag

You can have multiple **Evidence** components with the same **Tag** value. Discovering one of them will mark all of them as already seen. This is very useful if you want guards to only notice evidence once.

For example, if the player leaves three doors open, a guard should only raise its awareness once and notice there's someone around just once. Otherwise a guard might shout three times in a row the same line and look a bit too goofy.

The On Sight allows the Evidence to be seen by other agents.

The **Emit Noise** checkbox allows the **Evidence** to emit a noise when it is tampered, and allows other agents to notice it using the *Hearing* sensor.

Noise with Tag

It is worth noting that when an **Evidence** emits a noise, it will also make use of the *Tag* field as the identifier of the noises. Apart from noticing the **Evidence** agents can also react to the noise emitted just like any other noise, using the **On Hear** Trigger.

The **Emit Scent** allows the **Evidence** component to emit a scent identified by the *Tag* field that other agents can catch when the component is tampered.

Scent with Tag

Just like with noise, the scent emitted by the **Evidence** component can be used just like any other scent. The **On Smell** Trigger can catch its whiff identifying it by its *Tag* field.

7.9.1 Tampering

An **Evidence** does nothing by itself. Instead it must be **Tampered** in order for the **Evidence** to be noticeable by other agents.

Evidence at Runtime with Tampered field

Debug Evidence at Runtime

When entering Play-Mode you can select any **Evidence** component and a new field will appear at the bottom called *Is Tampered*, which indicates the current value.

In order to **Tamper** or **Reset** an already tampered evidence you can use the **Tamper Evidence** instruction and **Restore Evidence** instruction, respectively.

Evidence Tamper Instruction

Once an Evidence is tampered, it will start being noticeable by other Perception components.

7.9.2 Detection

The Trigger On Notice Evidence can be used to react when a tampered Evidence component is detected.

On Notice Evidence Trigger

The **On Notice Evidence** component detects when a specific **Perception** component detects a new **Evidence**, which can be optionally filtered by a specific *Tag* or not.

7.9.3 Relaying Knowledge

In order to make multiple **Perception** agents look like they work as a team, you can also relay **Evidence** knowledge between them. This is done using the **Relay Evidence** instruction, which fills in the knowledge gaps of the targeted agent.

Relay Evidence Knowledge

When a **Perception** has new relayed **Evidence** knowledge transferred, it can also react using the **On Relayed Evidence** Trigger.

Relayed Evidence Knowledge Trigger

Communication between Guards

Let's say we have two guards and one of them notices that a door has been opened. In that case, the first guard will receive the **On Notice Evidence** trigger event, upon which it will update its internal knowledge about the current state of **Evidence** components.

After the guard shouts something like We have company! Someone left a door open! it could use the Relay Evidence instruction in order to communicate to nearby guards that they should be alert.

Nearby guards can use the **Relayed Evidence** Trigger in order to know that a new **Evidence** component has been discovered by another guard nearby and raise their awareness level and shout something like *Understood!*.

7.10 Visual Scripting

7.10.1 Visual Scripting

The **Perception** module symbiotically works with **Game Creator** and the rest of its modules using its visual scripting tools.

- · Instructions
- · Conditions
- Events

Each scripting node allows other modules to use any **Perception** feature.

7.10.2 Conditions

Conditions

SUB CATEGORIES

• Perception

Perception

PERCEPTION

Sub Categories

- Evidence
- Hear
- See
- Smell

Conditions

- Compare Awareness
- In Awareness Stage

COMPARE AWARENESS

Perception » Compare Awareness

Description

Compares the Awareness value with another value

Parameters

Name	Description
Perception	The Perception component
Target	The Game Object checked
Value	The comparison to the Awareness value

Keywords

Awareness Track See Alert Suspicious Aware

IN AWARENESS STAGE

Perception » In Awareness Stage

Description

Returns true if the awareness of a target is in any of the specified stages $% \left(1\right) =\left(1\right) \left(1\right$

Parameters

Name	Description
Perception	The Perception component
Target	The Game Object checked
Stage	The stage(s) to check

Keywords

Awareness Track See Alert Suspicious Aware

EVIDENCE

Evidence

Conditions

• Is Evidence Tampered

Is Evidence Tampered

Perception » Evidence » Is Evidence Tampered

Description

Determines whether an Evidence game object has been tampered or not

Parameters

Name	Description
Evidence	The Evidence component

Keywords

Notice Change Tamper Modify Fiddle

HEAR

Hea

Conditions

- Can Hear Noise
- Hears Noise Tag

Can Hear Noise

Perception » Hear » Can Hear Noise

Description

Checks whether the Perception component can hear a new Noise stimulus

Parameters

Name	Description
Perception	The Perception component
Position	The position of the Noise stimulus
Radius	The radius of the Noise stimulus
Intensity	The intensity of the Noise stimulus

Keywords

Sound Noise Bell Intensity Stimulus

Hears Noise Tag

Perception » Hear » Hears Noise Tag

Description

Checks whether the Perception component is hearing a Noise Tag

Parameters

Name	Description
Perception	The Perception component
Noise Tag	The Noise Tag to check
Value	The comparison to the noise value

Keywords

Sound Noise Tag Bell Intensity

SEE

Se

Conditions

- · Can See
- Compare Luminance At

Can See

Perception » See » Can See

Description

Returns true if object can be seen by the Perception component

Parameters

Name	Description
Perception	The Perception component
Target	The Game Object checked

Keywords

See Sight Vision Detect

Compare Luminance At

Perception » See » Compare Luminance At

Description

Compares the Luminance value with another value

Parameters

Name	Description
Target	The object reference that checks its Luminance
Value	The comparison to the Luminance value

Keywords

Light Dim Lit Expose Sun

SMELL

Smel

Conditions

- · Can Smell Scent
- Smells Scent Tag

Can Smell Scent

Perception » Smell » Can Smell Scent

Description

Checks whether the Perception component can smell a new Scent stimulus

Parameters

Name	Description
Perception	The Perception component
Position	The position of the Scent stimulus
Radius	The radius of the Scent stimulus
Intensity	The intensity of the Scent stimulus

Keywords

Aroma Scent Smell Sniff Nose Trace

Smells Scent Tag

Perception » Smell » Smells Scent Tag

Description

Checks whether the Perception component is smelling a Scent Tag

Parameters

Name	Description
Perception	The Perception component
Scent Tag	The Scent Tag to check
Value	The comparison to the scent value

Keywords

Aroma Scent Smell Sniff Nose Trace

7.10.3 Events

Events

SUB CATEGORIES

• Perception

Perception

PERCEPTION

Sub Categories

- Awareness
- Evidence
- Hear
- See
- Smell

Events

- On Change Awareness Level
- On Change Awareness Stage

ON CHANGE AWARENESS LEVEL

Perception » On Change Awareness Level

Description

Executed when the Awareness value of a target changes

Keywords

Perceive Alert Aware Suspicious Curious Detect

ON CHANGE AWARENESS STAGE

Perception » On Change Awareness Stage

Description

Executed when the Awareness value of a target changes

Keywords

Perceive Alert Aware Suspicious Curious Detect

AWARENESS

Awareness

Events

• On Relayed Awareness

On Relayed Awareness

Perception » Awareness » On Relayed Awareness

Description

Executed when an agent with Perception receives new Awareness information from another agent

Keywords

Detect Bark Info Receive Propagate Transmit Communicate

EVIDENCE

Evidence

Events

- On Notice Evidence
- On Relayed Evidence

On Notice Evidence

Perception » Evidence » On Notice Evidence

Description

Executed when an agent with Perception notices a new Evidence component

Keywords

See Detect

On Relayed Evidence

Perception » Evidence » On Relayed Evidence

Description

Executed when an agent with Perception receives new Evidence information from another agent

Keywords

Detect Bark Info Receive Propagate Transmit Communicate

HEAR

Hear

Events

• On Hear

On Hear

Perception » Hear » On Hear

Description

Executed when the Perception hears a Noise

Keywords

Sound Noise Distract Alert Aural Hear

SEE

See

Events

• On See

On See

Perception » See » On See

Description

Executed when the Perception sees the specified (tracked) game object

Example 1

This Event will only execute on game objects that are being tracked

Keywords

Track Vision Sight

SMELL

Smel

Events

• On Smell

On Smell

Perception » Smell » On Smell

${\tt Description}$

Executed when the Perception smells a Scent

Keywords

Odor Smell Aroma Nose

7.10.4 Instructions

Instructions

SUB CATEGORIES

• Perception

Perception

PERCEPTION

Sub Categories

- Awareness
- Evidence
- Hear
- See
- Smell

Instructions

- · Track Awareness
- Untrack Awareness

TRACK AWARENESS

Perception » Track Awareness

Description

Starts tracking a game object in order to become aware of it

Keywords

Perceive Alert

UNTRACK AWARENESS

Perception » Untrack Awareness

Description

Stops tracking a game object and forgets about it

Keywords

Perceive Alert

AWARENESS

Awareness

Instructions

- Decrease Awareness
- Increase Awareness
- Relay Awareness Knowledge

Decrease Awareness

Perception » Awareness » Decrease Awareness

Description

Decreases the awareness of a target on a Perception component

Parameters

Name	Description
Decrement	The decreasing value of awareness
Perception	The Perception component that changes its awareness
Target	The target game object that changes its awareness

Keywords

Remove Less Know Detect Alert See

Increase Awareness

Perception » Awareness » Increase Awareness

Description

Increases the awareness of a target on a Perception component

Parameters

Name	Description
Increment	The increment value of awareness
Maximum Level	The maximum Awareness this increment can reach
Perception	The Perception component that changes its awareness
Target	The target game object that changes its awareness

Example 1

Use the Maximum Level if you want to increase the Awareness up to a certain threshold. For example, throwing a bottle nearby will make guards suspicious but never reach the state of Aware

Keywords

Add Sum Know Detect Alert See

Relay Awareness Knowledge

Perception » Awareness » Relay Awareness Knowledge

Description

Relays the Awareness knowledge of a game object to another Perception agent

Parameters

Name	Description
Perception	The Perception component that transmits its Awareness knowledge
Target	The Perception component that receives the Awareness knowledge
Perception	The Perception component that changes its awareness
Target	The target game object that changes its awareness

Keywords

Communicate Shout Tell Inform Transmit Propagate Know Detect Alert See

EVIDENCE

Evidence

Instructions

- Relay Evidence Knowledge
- Restore Evidence
- · Tamper Evidence

Relay Evidence Knowledge

Perception » Evidence » Relay Evidence Knowledge

Description

Relays the Evidence knowledge of a Perception component to another

Parameters

Name	Description
Source	The Perception component that transmits its Evidences
Target	The Perception component that receives the Evidence knowledge

Keywords

Communicate Shout Tell Inform Transmit Propagate

Restore Evidence

Perception » Evidence » Restore Evidence

Description

Restores the state of the evidence so that other agents do not notice it

Parameters

Name Description

Evidence The Evidence reference

Keywords

Change Modify Manipulate Clue

Tamper Evidence

Perception » Evidence » Tamper Evidence

Description

Tampers the evidence so that other agents notice it

Parameters

Name Description

Evidence The Evidence reference

Example 1

If a door is closed and the player opens it, the door can be considered as tampered and enemy agents will be able to notice the change

Keywords

Change Modify Manipulate Clue

HEAR

Цоо

Instructions

- Emit Noise
- Global Din

Emit Noise

Perception » Hear » Emit Noise

Description

Emits a Noise Stimulus that other Perception components can process

Parameters

Name	Description
Position	The center of the noise emitted
Radius	The radius of the noise emitted
Tag	The name identifier of the noise
Intensity	The strength value used for the noise emitted

Keywords

Sound Noise Distract Alert Aural Hear

Global Din

Perception » Hear » Global Din

Description

Changes the Global ambient din value

Parameters

Name	Description
Din	The new value for the ambient background noise
Transition	A set of options that allow to change the value over time

Keywords

Sound Noise Ambient Aural Hear Deaf

SEE

Sec

Instructions

· Change Global Luminance

Change Global Luminance

Perception » See » Change Global Luminance

Description

Changes the global ambient Luminance value

Parameters

Name	Description
Luminance	The new value for the global ambient Luminance
Transition	A set of options that allow to change the value over time

Keywords

Light Bright Dark Dim Night Sun

SMELL

Smel

Instructions

- Emit Scent
- Set Dissipation

Emit Scent

Perception » Smell » Emit Scent

Description

Emits a Scent Stimulus that other Perception components can process

Parameters

Name	Description
Source	The game object emitting the scent
Dispel Duration	The seconds it takes for the odor to fade out
Diffusion Rate	The growth factor of the smell per second
Tag	The name identifier of the noise
Intensity	The strength value used for the noise emitted

Keywords

Odor Smell Distract Alert Diffuse Dispel

Set Dissipation

Perception » Smell » Set Dissipation

Description

Changes the global ambient Dissipation value

Parameters

Name	Description
Dissipation	The new dissipation value. Values over zero reduce the smell dispel duration
Transition	A set of options that allow to change the value over time

Example 1

Dissipation values increase the time a scent dispels. A value of θ means it doesn't affect the dispel time. A value of 1 means it doubles the time it takes to dispel it

Keywords

Odor Smell Scent Nose Wind

7.11 User Interface

7.11.1 User Interface

The **Perception** component comes with multiple UI (user interface) components that help communicate with the player the current state of the **Perception** components around.

- · Awareness UI: Allows to show a floating indicator with the amount of awareness towards a certain object.
- Luminance UI: Allows to track how much luminance (light intensity) is affecting the chosen target.
- · Noise UI: Determines the intensity of the highest noise being heard compared to the current din value.
- \cdot Smell UI: Determines the scent with the highest intensity perceived by an agent.

Ready-to-use Examples

The demos include three examples of UI component that can be dragged and dropped onto your scene(s) which you can use or modify and tailor to your needs.

7.11.2 Awareness UI

The **Indicator Awareness UI** component is used to communicate the amount of **Awareness** a group of agents has towards a specific game object (usually the Player).

Awareness UI Indicator Editor

The component shows the **Awareness** value taking as a point of reference a specific game object and gathering all agents around a specific radius.



If your game has **Perception** agents that you don't want to display their awareness, you can filter them at the bottom of the component. For example, you can filter them by their game object tag and only display those that are marked as *enemies*.

Awareness UI Indicator at Runtime

The demos included in the **Perception** module come with a prefab that is ready to be dragged and dropped onto your scene.

7.11.3 Luminance UI

The **Luminance UI** component can be used to communicate with the player the amount of light intensity (also known as **Luminance**) received by a game object.

Luminance UI Editor

The **Lit** and **Dim** boxes allow to either draw how much light intensity has a specific game object or how much engulfed in darkness it is.



If your game is struggling with performance it is worth changing the **Update** field from *Every Frame* to *Interval*. This will allow to save some precious cycles and improve the performance of the game without barely affecting the gameplay.

Luminance UI at Runtime

The demos included in the **Perception** module come with a prefab that is ready to be dragged and dropped onto your scene.

7.11.4 Noise UI

The **Noise UI** component allows to display a progress bar of the noise with the highest intensity heard by a **Perception** component, as well as the amount of din.

Noise UI Editor



If your game is struggling with performance it is worth changing the **Update** field from *Every Frame* to *Interval*. This will allow to save some precious cycles and improve the performance of the game without barely affecting the gameplay.

Noise UI at Runtime

The demos included in the **Perception** module come with a prefab that is ready to be dragged and dropped onto your scene.

7.11.5 Smell UI

The **Smell UI** component allows to display a progress bar of the smell with the highest intensity caught by a **Perception** component.

Smell UI Editor

date at an Interval

If your game is struggling with performance it is worth changing the **Update** field from *Every Frame* to *Interval*. This will allow to save some precious cycles and improve the performance of the game without barely affecting the gameplay.

Smell UI at Runtime

The demos included in the **Perception** module come with a prefab that is ready to be dragged and dropped onto vour scene.

7.12 Releases

7.12.1 Releases

2.0.1 (Latest)



8. Shooter

8.1 Shooter



This module is currently under developement

9. Melee

9.1 Melee

Melee

Creating a deep combat system requires much more than playing animations and detecting what enemies pass through the blade of the attacker.

The **Melee** module aims to provide a generic framework to build your own combat system, whether it's a methodical and deliberate or fast paced with crazy combos.

Get Melee



The Melee module is an extension of $Game\ Creator\ 2$ and won't work without it

9.2 Setup

Welcome to getting started with the **Melee** module. In this section you'll learn how to install this module and get started with the examples which it comes with.

9.2.1 Prepare your Project

Before installing the Melee module, you'll need to either create a new Unity project or open an existing one.



It is important to note that Game Creator should be present before attempting to install any module.

9.2.2 Install the Melee module

If you haven't purchased the **Melee** module, head to the Asset Store product page and follow the steps to get a copy of this module.

Once you have bought it, click on Window Package Manager to reveal a window with all your available assets.

Type in the little search field the name of this package and it will prompt you to download and install the latest stable version. Follow the steps and wait till Unity finishes compiling your project.

9.2.3 Examples

We highly recommend checking the examples that come with the **Melee** module. To install them, click on the *Game Creator* dropdown from the top toolbar and then the *Install* option.

The **Installer** window will appear and you'll be able to manage all examples and template assets you have in your project.

- ullet Examples: A collection of scenes with different use-case scenarios
- Brawl: A deliberate and realistic combat system where characters fight with punches and kicks similar to Souls-like games
- Sword: A more complex and free-flow combat system with fast-paced animations similar to Devil May Cry and Kingdom Hearts
- ullet Sword FPS: A simple combat system tailored to first-person view

Installer Melee

The Examples requires all combat systems in order to work.



Clicking on the ${\bf Examples}$ install button will install all dependencies automatically.

Once you have the examples installed, click on the *Select* button or navigate to Plugins/GameCreator/Installs/Melee.Examples/.

Melee Examples

9.3 Weapons

When a **Character** has a **Weapon** equipped, it can be used to execute **Skills** using the **Combo** asset defined and the **Input** fed or directly using the **Play Skill** instruction.

Melee Weapon Overview

9.3.1 The Weapon Asset

To create a new **Weapon** asset, right click on the *Project Panel* and select *Create Game Creator Melee Weapon*.

Melee Weapon Asset

Title and **Description** allow to give it a name and description, which can later be used to display on the user interface. **Icon** and **Color** provide similar functionality for changing the graphic and its tint color.



Weapons are identified by their unique ID value, not by the asset name. Make sure all your weapons have a unique ID value!

Shields & Reactions

The **Shield** field is only necessary if you're using any kind of blocking mechanic. How blocking attacks work is covered more in depth in the **Shields** section.

Melee Weapon Asset

The **Hit Reaction** and **Parried Reaction** fields allow to define a **Reaction** asset that plays an animation when the character gets hit or its attack is parried by another character.

States

When a **Weapon** is equipped, the character using it can automatically switch to the defined animation **State**. This allows, for example, to hold a sword with the right hand when equipping a *Sword Weapon* and change the gait when wielding it.

Melee Weapon State



It's very important to be conscious about the **State Layer** field! By default, all characters equipping a weapon will use the layer index 5 as the layer index. **Shields** use the layer index 7, which is two units higher than the **Weapon** layer index so that the blocking state overrides the weapon one during that state. Layer 6, by default, is reserved for **Charged Skills**.

Combos

The **Combos** field allows to define which **Skills** will be executed under which conditions when inputting *Charge* and *Execute* commands (see **Input** for more information).

Melee Weapon Combos Asset

There are two ways to define the Combos of a weapon: Using a Combo reference asset, or embedding one directly.

use Combo references

We recommend using **Combo** asset references when possible as they are more flexible and allow to reuse the same move-sets for multiple weapons. However if you're certain you'll only use one move-set for a particular weapon, you can embed it directly and save up space in your project folder.

To embed a **Combo** directly you can use the *Embedded* option and the field below will turn into a **Combo Tree** where you can define the order and conditions in which attacks are executed.

Melee Weapon Combos Embedded

Instructions

The **On Equip** instructions are executed whenever this **Weapon** is equipped by a *Character*. This is the perfect place to instantiate and attach a prop representing the weapon onto the targeted character.

The On Unequip instructions are executed when the Weapon is unequipped by the Character.

The **On Dodge** instructions are executed when the **Weapon** wielder dashes through an attack and is invulnerable during those frames.

9.4 Shields

Shields are optional assets that allow to block incoming attacks.

Socking, Parrying and Breaking

Because there is no standard nomenclature throughout all games we decided to pick the following terms. However you can choose to name them in your game as you see fit:

Block: Stop any incoming attack while shielding that isn't a Parry.

Parry: Stop an attack where the time window between raising the shield and blocking is shorter than a certain amount. Some games call this *Deflect* or *Perfect Blocking*.

Break: Whenever the defense position of a character is broken, due to receiving too many impacts.

9.4.1 The Shield Asset

To create a new **Shield** asset, right click on the *Project Panel* and select *Create Game Creator Melee Shield*.

Melee Shield Asset

Defense

The upper-most values determine how the weapon can defend from incoming attacks.

Melee Shield Defense

The **Angle** field determines how aligned must the character be towards the attack in order to block it, starting from the front. For example, if the angle is 180 degrees, the character will defend from any attack that comes from the front and sides. A value of 360 will block any attacks from any direction.

The **Parry Time** field determines the maximum time window, in seconds, between the block starts and an attack connects to be considered a **Parry**. If the time between raising the shield and blocking the attack is higher than this value, it will be considered a normal **Block**.

The **Defense** value is a number that decreases with each blocked attack. If the value reaches zero, the defense will be **broken** and any other attack will hit the *Character*.

♥sualizing the Defense value

To visualize the current **Defense** of a character, enter *Play Mode* and select the character. Once a weapon is equipped, a few new fields will appear at the end showcasing the currently equipped weapons and combat values.

Melee Shield Defense at Runtime

The **Cooldown** determines how long it takes to start recovering **Defense** after decreasing it from a blocked attack.

The **Recovery** field determines the pace at which **Defense** is recovered, in seconds. For example, a value of 2 means it will recover 2 units per second.

States

A **Shield** allows to change the animation **State** of the character when rising the shield and stop it automatically when lowering the defense.

Melee Shield States



It's very important to be conscious about the **State Layer** field! By default, all characters equipping a weapon will use the layer index 5 as the layer index. **Shields** use the layer index 7, which is two units higher than the **Weapon** layer index so that the blocking state overrides the weapon one during that state. Layer 6, by default, is reserved for **Charged Skills**.

The **Speed** field allows to run the whole **State** animations faster or slower using a coefficient. A value of 1 means the animation will run normally and a value of 0.5 means the animations will be in slow-motion.

Transition In and **Transition Out** fields determine the time in seconds it takes to blend into the state. We recommend using small values, between 0.1 and 0.5 seconds.

Blocking

Blocking happens whenever the character blocks an incoming attack that isn't perfectly blocked (also known as parried).

Melee Shield Blocking

The **Block Reaction** field allows to play an animation gesture when blocking an attack. See **Reactions** for more information.

The **Effect** field allows to instantiate a prefab at the point of impact. For example, a particle effect that highlights the blocking.

The **On Block** instructions are executed every time an attack is blocked, regardless of the type of attack. When these instructions are executed, *Self* refers to the character blocking the attack, and *Target* to the attacker.

Parrying

Parrying happens whenever the character blocks an incoming attack and the time between raising the shield and the impact is less than a certain amount of time. This mechanic is also commonly known as *perfect blocking*.

Melee Shield Parrying

The Parry Reaction field allows to play an animation gesture when parrying an attack. See Reactions for more information.

The **Effect** field allows to instantiate a prefab at the point of impact. For example, a particle effect that highlights the parry.

The **On Parry** instructions are executed every time an attack is parried, regardless of the type of attack. When these instructions are executed, *Self* refers to the character blocking the attack, and *Target* to the attacker.

Breaking

Breaking happens whenever the character's blocking defense is broken. When a character starts blocking, it has a defense gauge that decreases with every attack blocked. If the gauge value reaches zero, the defense becomes broken and the character is open to attacks.

Melee Shield Defense Broken

The **Break Reaction** field allows to play an animation gesture when the defense is broken. See **Reactions** for more information.

The **Effect** field allows to instantiate a prefab at the point of impact. For example, a particle effect that highlights the point where the defense is broken.

The $On\ Break$ instructions are executed every time the defense is broken, regardless of the type of attack. When these instructions are executed, Self refers to the character blocking the attack, and Target to the attacker.

9.5 Skills

9.5.1 Skills

Skills are assets that define an action performed by a character.



The most common use-case of a **Skill** is a single attack, but it can also be a synchronized takedown, an attack cutscene or even a non-attack animation.

The Skill Asset

To create a new Skill asset, right click on the Project Panel and select Create Game Creator Melee Skill.



The **Skill** asset is a very complex one. This page goes over each one of its parts without much detail. There will be a link annexed to each sub-section that deep-dives into its features.

Melee Skill Asset

Title and **Description** fields allow to optionally give the skill a name and an explanation about what it does. These values, along with **Icon** and **Color** are used to display information on the user interface.

CHARGE

The **Charge** section defines the behavior of the skill when using it as a charge. A charge happens when the character holds a pose during a certain amount of time before executing the animation and its effects.

Melee Skill Charge

The Charge section allows to specify an animation State that the character enters before executing the Skill.



For more information about Charged Attacks, see the Charges section.

STRIKE

The Strike section is relevant for Skills that deal damage to other characters.

Melee Skill Strike

The **Direction** field determines the direction, from the attacker's perspective, in which the **Skill** affects the enemy.

The **Predictions** field allows to define how many inter-frame physics predictions are performed. In most cases, a single prediction should be enough. However if the animations are so fast that the weapon ghosts-through enemies, this value can be fine-tuned until the attack detects all possible enemies.

The **Use Strikers** dropdown allows to define which weapon **Strikers** to use. This is especially useful if your game allows to equip multiple weapons and you only want one of them to hit enemies during this attack.



For more information about Strikes and how to configure them, see the Strikes sub-section.

TRAIL

The **Trail** section allows to override the default trail left by **Strikers** when attacking an enemy. Each checkbox allows to change the value of the trail option.

Melee Skill Trail



For more information about Trails and what each option does, see the Striker's Trail section.

EFFECTS

The Effects section allows to define what happens when the Skill is used.

Melee Skill Effects

For example, the Sound fields allow playing sound effects when using, hitting, getting blocked and parried.



In order to avoid playing the exact same sound effect over and over again, **Melee 2** sound effects have a slight random pitch variation.

The **Hit Pause** checkbox enables hit-pause (also known as Hit Freeze) when the **Skill** successfully hits something.

Time Scale determines the time coefficient at which time slows during the hit.

Delay allows to introduce an unscaled time delay. This is useful if you want to slow time after a slash to showcase the trail better.

Duration is the unscaled time in seconds that the hit-pause effect takes effect.

The Hit Effect field allows to instantiate a prefab object at the point of impact after a successful hit.

SEQUENCER

The **Sequencer** is the most important section of a **Skill**. It determines the animation played and all the events that are executed when it plays.

Melee Skill Sequencer

The **Animation** field is required in order to run a **Skill**. It will play an animation using the *Gestures* system from the character and allows to attach events and phases in the timeline tool below.

An **Avatar Mask** can also be provided in order to play the animation on certain bones of the character. For example, slash with a sword with the right hand while letting the rest of the body play the locomotion animation.

Gravity determines how strong the downward force on a character is when executing the **Skill**. The most common scenario will have a value of 1, where the character is affected by gravity. However, if the **Skill** allows the character to jump, reducing or setting the gravity value to zero will help the character snap out of the ground.

The **Transition In** and **Transition Out** fields allow to define the blend-time between the character's current animation and the **Skill**'s animation. It's recommended to use small values such as 0.1 or 0.3.

Melee Skill Sequencer in Detail

The **Motion** field defines the type of motion the character will use when playing the **Skill**. There are three possible values:

- · None: The Skill doesn't take over the motion of the character and it's free to move as it plays out.
- Root Motion: The Skill overrides the character's locomotion and uses the animation clip's root motion.
- Motion Warping: Similar to root motion, but also allows to define a range in which the character interpolates its position and rotation towards a destination.



For a complete deep-dive into Motion's details, see the Motion page.

The **Enter Skill Mode** button allows to preview the animation on the scene-view and tweak the different **Phases** and **Motion** values from the **Sequencer** below.

Once in Skill Mode the default character will appear on the screen and the animation can be previewed by scrubbing the timeline below.



You can change the character model by dragging and dropping your own model onto the corresponding field and clicking the *Change Model* button. The **Skill** will remember which model is used for this one.

formation about Sequencer

For a complete deep-dive into the **Sequencer**'s options, see the **Sequence** page.

Melee Skill Sequencer Speeds

The three **Speed** fields allow to determine the speed coefficient of each one of the *Attack Phases*. If the **Skill** doesn't have any attack phases, the **Anticipation Speed** will be used as a coefficient of the animation's speed.

A value of 1 means the animation plays at its normal rate, while a value of 2 means it will play twice as fast.

POISE

A Skill has two Poise values:

- · Poise Armor: Determines the poise defense value when a character uses this Skill.
- · Poise Damage: Determines the poise damage it inflicts when a character uses this Skill.

Melee Skill Poise

When a character starts executing a **Skill** it starts with its full *Poise Armor* value. If during the execution of the **Skill** it receives an attack, it will get its armor damaged by the attacker's *Poise Damage* skill value.

If after receiving an attack its *Poise Armor* reaches zero, the current **Skill** will be interrupted and a **Hit Reaction** will be played instead.

Aformation about Poise

For a complete deep-dive into the Poise system, see the Poise page.

POWER

The **Power** of a **Skill** can be used to play different animations depending on its strength. For example, a normal attack could have a value of 1 while a powerful blunt attack could have a value of 2.

Melee Skill Power

These values can be used in the **Reaction** assets to play a different reaction in each case. For example, the normal attack could execute a small flinch while the powerful blunt attack could make the victim play a knockback animation.

INSTRUCTIONS

There are three Instruction Lists called under different circumstances:

- On Start: These instructions are called as soon as the Skill starts to play. The Self value references the character playing the skill.
- On End: These instructions are called whenever the Skill stops playing, even if it's interrupted by a poise-break or another skill.
- On Hit: These instructions are called whenever the Skill lands a successful attack onto another character.

 The Self value references the character attacking and Target references the victim.

Melee Skill On Start, On End and On Hit Instructions



The **On Hit** instruction list is the perfect place to deal damage to the enemy, either using a **Formula** from the Stats module or using **Local Variables**.

Running Skills

At any given point a Skill can be forced on a character using the Play Melee Skill instruction.

Melee Skill Instruction

All you need to specify is the character that's going to play the **Skill** and information necessary for the correct execution.

9.5.2 Charges

The **Skill** asset allows to very easily define *Charged Attacks*.

When a *Charge* skill starts it will enter an animation **State** which may (or may not) change the character's locomotion properties.

Melee Skill Charge

charging a Slash

In this example, the player holds the left mouse button and the character enters a sheathed state in which it can't move. Upon releasing the button, the character performs the **Skill** attack.

Melee Skill Charge Attack Example

It's important to note that the Combo asset defines whether a Skill is a charged one or not.

Melee Skill Charge in Combo

When selecting the **Charge** option from the **Combo** asset, you'll be prompted whether to auto-release the **Skill** after the minimum charge or let the user specify when to release it.

It will also allow to define the minimum amount of time to charge the attack in order to have any effect.



For more information about ${\bf Combos},$ see the ${\bf Combos}$ section.

9.5.3 Strikes

When a character attacks with a **Skill** there is a phase called *Strike* in which the current weapon(s) collect any hit enemies and pass the information to the **Skill** in order to determine if each of the enemies hit was successful, blocked, parried or ignored.

Upon entering the *Strike* phase, the **Skill** looks up all the **Striker** components of the weapon object(s) and gathers any overlapping enemies.

Using multiple Weapons

If the character has equipped more than one weapon and the **Skill** requires one of them to hit the enemies, you can specify the ID of the **Striker** by changing the dropdown value from *All* to *By ID*.

Melee Skill Strike by ID

A new ID field will appear below where the Striker id value can be specified

The Striker Component

Strikers are components attached to game objects (usually props that represent weapons) that detect hit enemies when the character attacks.

Melee Striker Component

The ID allows to hand-pick Strikers by their unique identifier in each Skill.

The **Section** game object reference allows to define where the *Blade* of the weapon is. By default, it should be the game object where the **Striker** component is attached to, but some weapons, like a whip, might require mobile parts to work as the hitting part.

Melee Striker Scene View Configuration

The **Shape** dropdown allows to define the shape of the physics volume that captures enemies when passing through them. There are different options:

- Sphere: The most basic shape. It captures enemies within a defined radius.
- Capsule: A pill-like shape that captures enemies along its length and is defined by a radius and a height value.

The **Trail** section defines two points in which a trail is drawn whenever the weapon is used during a *Strike* phase.

It also determines the maximum amount of **Quads** allowed to draw the trail mesh, it's total **Length** and the **Material** used.

Melee Skill Strike Weapon Trail

The Trail

When a **Skill** enters the *Strike* phase, all **Striker** components that are involved start drawing the **Trail** effect, which is automatically faded upon finishing the phase.

The **Trail** is drawn using a Catmull-Rom spline with regular intervals so that it looks smooth regardless of the speed at which the animation plays.

Point A and Point B define the segment from which the trail will be drawn.

Visualize the Trail

During play-mode you can visualize the **Trail** by selecting the character executing the **Skill**. It will automatically draw the quads on top of the trail's material so you can visually see whether that particular skill needs more or less quads in order to look good.

Melee Striker Trail visualization

Predictions

When capturing enemies that the weapon passes through there is a chance that enemies *ghost-through* if the **Striker** shape is very small and the animation plays very fast.

Melee Skill Strike

The **Predictions** field allows to determine inter-frame physics casts that are performed each frame to avoid this problem.

For example, setting a value of 5 means it will capsule-cast 5 times every frame between the weapon position at the last frame and the current one. You can visualize the predictions by selecting the character executing the skill and seeing how each colored volume appears:

- Green Volume: Shows the current frame weapon's cast volume position.
- Blue Volume: Shows the predicted volumes positions.

Melee Skill Striker Predictions



While it's easy to increase the number of predictions to minimize the chance of ghosting-through enemies, each prediction has a direct performance impact on the physics engine. It's better to keep them at a minimum if there are going to be lots of enemies fighting at the same time.

9.5.4 Motion

The **Motion** field determines how the character moves when the **Skill** is executed. There are three possible values:

- · None: The Skill doesn't take over the motion of the character and who's free to move as it plays out.
- · Root Motion: The Skill overrides the character's locomotion and uses the animation clip's root motion.
- Motion Warping: Similar to root motion, but also allows to define a range in which the character interpolates its position and rotation towards a destination.

None

Setting the Motion value to None allows the character to move normally during the execution of the Skill.

Melee Skill Motion None

This is useful if you have an upper-body Avatar Mask masking the lower-body so that the attack animation only plays on the torso and arms, but not on the legs. This allows the character to move while attacking.

Root Motion

Selecting the **Root Motion** option allows the **Skill** to take over the character's control for the duration of the animation, translating and rotating it using its root motion values. This allows very fine-grain control over how far the character moves during an attack combo and where it ends up looking.

Switching to **Root Motion** also adds two new tracks onto the **Sequence** timeline: One with the letter P, which stands for *Position*, and another one with the letter R, which stands for *Rotation*.

Melee Skill Root Motion Sequencer

These sliders define a range where the root motion doesn't take effect onto the character's animation.

using Root Motion sliders

For example, let's say you have a **Skill** where the character curls the weapon close to itself for a couple of seconds before launching forward.

If you were not to use the sliders, the character would not be able to track the enemy during the *Anticipation* frames and they could simply slightly step out of the way of the attack.

Melee Root Motion Example

Using the **Rotation** slider one can let the character pivot around itself during the first frames in order to keep tracking the enemy during the *Anticipation* frames, before sling-shotting itself towards the enemy.

Motion Warp

The **Motion Warp** also uses the *root motion* animation of the character, but instead of allowing to define the *position* and *rotation* frames at which the **Skill** takes over the locomotion control of the character, it defines a *warp* range during which the character will smoothly change its position towards the targeted one.

Melee Skill Motion Warp Sequencer

Using the Warp slider

The Warp slider is useful for combat systems that require characters to snap towards enemies, like in *Kingdom Hearts*, *Batman Arkham* game series or *Spiderman*.

Melee Motion Warp Example

At the beginning of each attack, the character attempts to close in the distance to the enemy during a few frames. The distance amount and skill used depends on how far the character is from its target.

The Warp slider can be selected to reveal option settings to configure how the warp is performed.

The **Conditions** list at the top allows to check whether the warp should happen or not. The most common use-case is checking whether there is a target available or not.

The **Easing** field specifies the easing curve used when moving the character from its starting position to its destination.

Self and **Target** fields define the final position of both the character executing the **Skill** (Self) and the targeted character receiving the attack (Target).

Using Melee Locations

For both Self and Target locations, we recommend using the values found under the Melee section.

Melee Skill Motion Warp Sequencer

There is a collection of options, each with its own description of what it does. The most useful ones are:

- **Self/Target Close**: Moves the character close to the target, keeping a specified distance, in a straight line. This is mostly used to close-in an attack.
- · Look at Self/Target: Rotates the character towards the target. Useful for synchronizing takedown attacks.

The rest of the options allow moving the character at each cardinal position of the opponent.

If you don't want either the Target or Self to change its location, simply set the value to None.

Selecting Motion Warp also adds a new field called Sync Reaction. This option allows to play a synchronized animation onto the *Target* character along with the Skill. This is especially useful when performing takedowns or playing animations that require enemies to react in a certain way.

Takedown Skill

Melee Motion Warp Takedown

In the video above, the unaware enemy plays a *Takedown* animation as soon as the **Skill** starts. In order to synchronize the position of both the attacker and the victim, the **Warp** slider defines the following:

Melee Motion Warp Takedown settings

The Self character (which is the attacker) moves in close to the enemy, keeping a distance of 1.5 units.

The Target character (which is the victim) simply looks at the attacker.

These values allow to define a sync point at which the animation can play synchronized animation on both characters that looks seamless.

9.5.5 Sequence

The **Skill** asset contains a *sequencing* timeline tool called **Sequencer** that allows to configure every little detail that happens during the execution of the animation.

To edit the **Sequencer**, click on the **Enter Skill Mode** button. The *Scene View* and *Hierarchy Panel* will change its appearance to one similar to when editing a prefab.

Melee Skill Sequencer Stage Mode

Entering into **Skill Mode** displays the default character at the center of the screen with nothing else. This character is automatically bound to the sequencer, which you can scrub to play the animation forward and backwards, frame by frame.

change Character

To change the preview character model, simply drag and drop the model onto the corresponding field and click on the **Change Character** button. This will change the model for this particular scene and it will remember to use it in the future.

Melee Skill Sequencer Change Model

Sequencer Anatomy

The **Sequencer** is composed of horizontal **Tracks** and each one defines information using **Clips**, which are the little rhomboids that can be dragged along the timeline.

Melee Skill Sequencer Anatomy Track and Clip

On the left side of each **Track** there's a rhomboid with a *minus* sign and on the right side, there's a similar one with a *plus*. Each of these buttons allow to remove or add a new **Clip** at the position of the play-head.

Tack Behavior

Each **Track** allows to create or remove different amounts of clips. When it's not possible to add new ones, the buttons will be grayed out.

The Skill Sequencer has three different Tracks by default:

- · Attack Phases Track: This determines the Anticipation, Strike and Recovery phases of any attack.
- · Animation Cancel Track: Determines which frames the Skill can be canceled by another action.
- Instructions Track: Allows to execute Instructions at different points of the timeline.

Root Motion and Motion Warp Tracks

Setting the Motion of the Skill to either Root Motion or Motion Warp will add other tracks to the Sequencer. These ones are covered in their corresponding section under Motion.

Attack Phases

When executing any Skill that is an attack, there will always be three phases:

- · Anticipation: Also known as wind-up. In this phase, the character prepares to execute an attack.
- Strike: Also known as activation. In this phase, any enemies passing through the edge of the weapon will be hit.
- · Recovery: Also known as follow-through. In this phase the character is exposed to enemy attacks.

Melee Skill Sequencer Attack Phases

Chaining Combos

When executing an attack that is the follow-up of another one (known as *combo*), the *Recovery* phase of the first attack is skipped in favor of the anticipation of the new one. This makes combos feel faster and more responsive, without having to wait to idle the pose before starting the new attack.

Animation Canceling

The **Animation Canceling** track determines at which frames the user can cancel the **Skill** in order to execute something else.

Melee Skill Sequencer Animation Canceling

The **red** portion of the **Track** are the frames that the character isn't allowed to cancel. This means that the character may be able to cancel the start and/or the end of the animation.

Executing a Faint

Some games allow to cancel the start of an attack using the *block* during the first few frames of an attack. This can be easily done by dragging the start of the **Animation Canceling Clip** a few frames from the start.

Roll Cancel

Some games also allow to cancel the recovery phase of an attack using a *roll*. This can be easily done by leaving some empty space between the end-clip of the **Animation Canceling Clip** and the last frame of the track.

Instructions

The Sequencer allows to run arbitrary Instructions at any frame of the Skill animation.

Melee Skill Sequencer Instructions Track

To add a new **Instruction** list, simply move the play-head at the frame you want to do something and click the right-most plus button from the **Instructions** track. Select the new **Clip** that appears to reveal the **Instructions** list below, where you can create any logic that will run at that point in time.

Melee Skill Sequencer Instructions Track Node with Sound Effect

Anceling Skills

Note that a **Skill** can be canceled at any point and thus some **Instructions** won't be executed if these are further away in time from the canceling point. Do not run critical logic in this **Track**. Instead, use the **On Start** and **On End** instructions callback, which are guaranteed to run, even if the **Skill** is canceled.

9.5.6 Poise

The **Poise** refers to the ability of a character to withstand other attacks during the execution of a **Skill** without canceling it.



In some games, this is also known as Hyper-Armor.

Melee Skill Poise

When a character starts playing a **Skill**, its **Poise Armor** is filled. If during the execution of this **Skill** the character receives any attack from an enemy, the **Poise Armor** will be reduced by the **Poise Damage** of the enemy's **Skill**.

Any subsequent attacks will also damage the **Poise Armor**. If it reaches zero or less, the character's **Skill** will be automatically canceled at that frame and will play a **Hit Reaction**.



By default, all **Skills** have a **Poise Armor** of 1 and a **Poise Damage** of also 1 unit. This means that any attack can be interrupted by an enemy's attack. If you want an enemy to withstand more than one attack, increase the **Damage Armor**.

The **Poise** value of a character can be visualized at runtime by simply selecting the character object and looking at the *Inspector* window.

Melee Skill Poise at Runtime

At the very bottom of the character component, there's a **Poise** gauge that is filled whenever the character executes a **Skill**. This gauge diminishes with each attack received.

9.6 Reactions

The **Reaction** assets allow characters to react to different stimulus, such as being hit, blocking an attack, being the victim of a takedown, etc... It allows to define a list of *Animation Clips* and one of them is picked based on different conditions.

For example, a character being attacked from the front might play a random flinch animation, but the same attack hitting its back might make it stumble or even get knocked down.

9.6.1 The Reaction Asset

To create a new **Reaction** asset, right click on the *Project Panel* and select *Create Game Creator Melee Reaction*.

Melee Reaction Asset

The **Reaction** asset has a **Transition In** and a **Transition Out** field that define how long it takes for a character to blend in and out the selected animation from the reaction. It is recommended to have small values, such as 0.1 up to 0.5 seconds.

The **Use Root Motion** checkbox defines whether the animation being played takes over the character's locomotion and will use the animation clip's root motion to move the character. In most cases, this checkbox should be checked.

The **Speed** field is a coefficient that is used to speed up or down the animation played. By default it's set to 1, which indicates that the animation will play at its original velocity. Setting a value of 2 means the animation will play twice as fast, and a value of 0.5 will play the animation in slow-motion.

Clicking on the Add Reaction button creates a new entry, of which you can create as many as needed.

Melee Reaction Entry

When a **Reaction** decides which animation is played, it starts checking all entries, from top to bottom. If the conditions of an entry are successful, then the animation played is picked randomly between the ones provided by that entry.

Entry Conditions

The **Min Power** checkbox determines whether the entry requires a minimum power in order to be considered successful. Upon ticking the toggle, a decimal field appears on the right side, which is used to define the minimum power threshold.



The power is provided by the attacker's **Skill**. This allows to play different hit reactions depending on the power of the attack received, and playing a knock-back animation when the strength of the attack taken is higher than a certain value.

The **Direction** field allows to execute a particular entry only if the direction of the attack received matches the direction of the attack. The *From Any* option ignores the direction of the skill.

On top of these conditions, you can also specify visual scripting **Conditions** such as checking stats, and other kinds of data from both the attacker and the victim.

\$€1f and Target

When checking Conditions the Self value references the character attempting to play a Reaction and the Target the attacker character.

Entry Behavior

The **Cancel Time** toggle allows to define a maximum time at which the character playing the **Reaction** can cancel the reaction and play another **Skill**, *dash* or do any other action.



A Stun Lock is what happens when all attacks of an attacker character are faster than the reaction animation of its victim. If the aggressor constantly attacks a character, it can't break free because each new attack locks it in a new flinching animation.

To avoid that, try ticking the **Cancel Time** checkbox and give it a small threshold time. This will allow characters to attempt dashing out of the way after being hit but won't allow them to move during the reaction time

This is especially useful for the player character.

The **Rotation** field allows the character to either look away from the attack direction, towards the attack direction or not rotate the character at all. This is usually useful when all your animations are frontal ones and you don't want to create directional animations.

Gravity determines the influence this reaction entry will have on the character's own gravity. This is mostly used when doing airborne hit reactions, where the character stays up in the air while playing the hit animation.

The Avatar Mask allows to play the entry animation clips on just a few bones.

The **Animation Clip** list below determines all animation clips that are part of this **Reaction Entry**. If the conditions are successful, a random clip will be picked (without repetition) from the list.

Instructions

A **Reaction** asset also allows to execute **Instructions** upon starting and/or exiting the animation. These instructions are guaranteed to be executed, even if the **Reaction** being played is canceled.

- $\boldsymbol{\cdot}$ The $\boldsymbol{0}\boldsymbol{n}$ $\boldsymbol{E}\boldsymbol{n}\boldsymbol{ter}$ instructions play as soon as the animation starts playing.
- The On Exit instructions play as soon as the animation stops, or the reaction is canceled.

9.6.2 Running Reactions

At any given point a Reaction can be forced on a character using the Play Melee Reaction instruction.

Melee Reaction Instruction

All you need to specify is the character that's going to play the **Reaction** and information necessary for the correct selection of the entry.

9.7 Combos

Combos define the order in which Skills are executed using different inputs and conditions.

Melee Combos Asset Example

A **Combo** can be either defined as a separate asset or embedded onto the **Weapon** itself, although we recommend the first option in order to reuse the same combos for multiple weapons.

9.7.1 Combo Asset

To create a new **Reaction** asset, right click on the *Project Panel* and select *Create Game Creator Melee Combo*.

Melee Combo Asset

9.7.2 Combo Anatomy

A Combo asset or embedded value have both the exact same layout:

- A left panel with a tree-like structure that defines each Combo entry.
- · A right panel that displays more information and options for the selected Combo entry.

Combos are executed from top to bottom, and upon successfully validating an entry from the left panel, that *Skill* is executed.

Combo entries can be nested into other **Combo** entries in order to create attack combos. For example, having a *Light Attack 2* under the *Light Attack 1* entry will allow to execute the second attack right after finishing the first attack, and skipping the *Recovery Phase*.

To reorder **Combo** entries, simply drag and drop the entry from the left panel where you want it to go. Dropping an entry onto another entry will add it as a child of it.

Combo Input

The first fields allow to define the input type of a particular **Combo** entry. There are 8 possible keys to use and 2 modes: *Tap* and *Charged* **Skills**.

- Tap: Tap skills are executed as soon as the Execute input is detected and don't require any waiting time.
- Charge: Skills start being charged as soon as the Charge input is detected and fully execute when the Execute input is input.



For more information about input commands, see the ${\bf Input}$ section.

If the Charge option is selected, two fields will be revealed below:

- Timeout: The minimum amount of time it needs to pass before the charge can be executed.
- Auto Release: Whether the charge can be hold indefinitely or should it execute as soon as the minimum timeout expires.

In both **Charged** and **Tap** inputs there's another checkbox field called **Has Delay**. This field is only available when the **combo** entry is a child of another **combo** entry and requires the **Execute** input to happen after a delay.

Melee Combo Has Delay

Delayed Input Commands

Delayed inputs usually execute much more powerful attacks but require precise timing. Hence why these are usually only used by seasoned players that don't button-mash the controller. For example, *Devil May Cry* was one of the first action games to make use of delayed input attacks.

Combo Execution

The Conditions list at each Combo entry allows to determine whether this entry should be executed or not.

When to use Conditions

A common use-case of **Conditions** inside **Combo** entries is when the character executes different attack animations when being airborne or grounded.

The **Execution Order** defines whether the **Combo** entry is taken into account in order (option *In Order*), or it can interrupt any combo chain (option *Anytime*).

Interrupting Combos with Skills

Combo entries that are marked as Anytime can run even when the character is playing a combo attack.

The last field in the Combo entry is the Skill itself, which is required in order to execute it.

9.8 Input

There are two ways to execute **Skills**:

- · Using the Play Melee Skill instruction.
- Using the Input mechanism and let the Combo decide which Skill to play.

This section focuses on the second one.

9.8.1 How to Input

When a character has at least one **Weapon** equipped with a **Combo** asset or embedded value, it is ready to receive input commands, and its combat module will decide whether it can play a **Skill** and which one to run.

There are two instructions to feed characters with:

- Input Charge Instruction should be executed when pressing an attack button/key
- · Input Execute Instruction should be executed when releasing an attack button/key

Melee Input Example Charge and Execute

No Charge? No Problem

If your game does not have any charged attacks you might prefer executing the **Input Execute** directly upon pressing the key, instead of the release, and skipping the **Input Charge**. This will make the controls feel snappier and responsive, but on the other hand, you won't be able to charge attacks.

The **Input** mechanism has been engineered to support multi-platforms out of the box. There are 8 abstract bindable keys which are called A, B, C, ... all the way to H.

Keyboard and Gamepads

To allow multi-platform support, simply use Triggers that execute each of the abstract keys from different inputs.

 $For example, to support both Keyboard and Gamepad controllers, you can use the following {\it Triggers} for Keyboards:$

Trigger On Mouse Press [Left Button]: Charge Input A on Player

Trigger On Mouse Release [Left Button]: Execute Input A on Player

And also add the following Triggers for gamepads:

Trigger On Gamepad Press [Square Button]: Charge Input A on Player

Trigger On Gamepad Release [Square Button]: Execute Input A on Player

9.8.2 Input Buffer

The **Input** mechanism supports a technique called *input buffering* which allows players to input their commands with a slight time window error margin before executing the **Skills** from the input keys.

Melee Input Buffering

Input Buffer for Combos

For example, let's say the player is doing a 3-hit light attack combo. The first time they press the $\bf A$ key, the player character starts playing the skill Light Attack 1, which takes around 1 second to execute.

However, around 0.75 seconds have passed, the player presses the A key again to send the command to do a followup *Light Attack* 2 skill. However, because the first attack hasn't finished, without *input buffering* the second attack would never start.

With input buffering the input keys are remembered for a maximum amount of time before being consumed.

The Input Buffer window duration can be changed at any time using the Set Buffer Window instruction.

By default it uses a 0.5 seconds window, but if you feel that's too much, you can easily change it with the aforementioned instruction.

9.9 Targets

Characters can focus on a targeted character in order to track it and direct their attacks towards them.

On top of that, the Melee module also allows to cycle through a list of candidate targets.

To add new candidate targets, use the Add Target Candidate instruction. To remove it, use the Remove Target Candidate instruction.



When to Add/Remove Candidates

In most cases you'll want to set Player target candidates as soon as the enemy appears, and remove the candidate when the enemy dies. To do so, simply add an On Start Trigger on the enemy that adds itself to the Player's candidates, and another Trigger set to On Destroy that removes itself from the Player's target Candidates.

Melee Target add Candidate



If your game doesn't require to cycle through targets, you can skip all this and simply use the Set Target instruction to focus a character onto a specific enemy. You can also use the Clear Target to remove its current targeted character.

9.9.1 Cycling Through Candidates

Once a character has more than 1 candidate target, it can cycle through its list to focus on a particular one. There are multiple ways to cycle through a list of candidates:

Closest Candidate

Focuses on the closest candidate to the character. This is useful for auto-targeting enemies in fast-paced games where the enemy that's closest to the player should be prioritized.

Use the Cycle Closest instruction to automatically select the one that's closest to the character.

Next / Previous Candidate

Focuses on the next or previous character from its internal list. This is usually done in games with few enemies on screen where the right joystick is used for something else other than targeting. For example, pressing the left shoulder button to cycle through the list of enemies.

Use the Cycle Next and Cycle Previous instruction to automatically select the next or previous target.

circular Cycles

Repeatedly calling the Next or Previous candidate will circle back to the first or last position upon reaching the other end. For example, if there's a 3 candidate list of enemies, and we're currently focusing on the 3rd and run the Cycle Next instruction, it will jump back to the first position and focus on the first candidate on the list.

Cycle by Direction

Focuses on the next candidate that most closely matches the direction given in screen-space from the character's position and the camera reference provided. This is intended to be used with gamepads that have a right-stick that you can use to select the next target.

Use the **Cycle Direction Target** instruction to select the next candidate based on the camera's perspective and direction provided.

9.10 Visual Scripting

9.10.1 Visual Scripting

The **Melee** module symbiotically works with **Game Creator** and the rest of its modules using its visual scripting tools.

- · Instructions
- · Conditions
- Events

Each scripting node allows other modules to use any **Melee** feature, and adds a list of **Properties** ready to be used by other interactive elements.

9.10.2 Conditions

Conditions

SUB CATEGORIES

• Melee

Melee

MELEE

Conditions

- · Has Equipped Melee
- In Attack Phase
- Is Blocking
- · Last Cancel Successful
- Time Since Last Block
- Time Since Last Break
- Time Since Last Parry

HAS EQUIPPED MELEE

Melee » Has Equipped Melee

Description

Returns true if the Character has a specific Melee Weapon equipped

Parameters

Name	Description
Character	The targeted Character
Weapon	The Melee Weapon to check if it is equipped

Keywords

Combat Melee

IN ATTACK PHASE

Melee » In Attack Phase

Description

Returns true if the character is in any of the specified attack phases

Parameters

Name	Description
Character	The targeted Character
Phases	The attack phases the character might be in

Keywords

Combat Melee Attack Anticipation Strike Activation Recovery

IS BLOCKING

Melee » Is Blocking

Description

Returns true if the specified Character is blocking attacks

Parameters

Name Description

Character The Character that might be blocking attacks

Keywords

Combat Melee Block Defend

LAST CANCEL SUCCESSFUL

Melee » Last Cancel Successful

Description

Returns true if the last attempt to cancel a skill was successful

Parameters

Name Description

Character The Character that might have attempted to cancel its skill

Keywords

Combat Melee Attack

TIME SINCE LAST BLOCK

Melee » Time since Last Block

Description

Returns true if the time since the last blocked attack is less than a value

Parameters

Name	Description
Character	The Character targeted
Time	The maximum time for this condition to be true

Keywords

Combat Melee Block Defend

TIME SINCE LAST BREAK

Melee » Time since Last Break

Description

Returns true if the time since the last broken attack is less than a value

Parameters

Name	Description
Character	The Character targeted
Time	The maximum time for this condition to be true

Keywords

Combat Melee Block Defend Broken Destroy

TIME SINCE LAST PARRY

Melee » Time since Last Parry

Description

Returns true if the time since the last parried attack is less than a value $% \left(1\right) =\left(1\right) \left(1\right$

Parameters

Name	Description
Character	The Character targeted
Time	The maximum time for this condition to be true

Keywords

Combat Melee Block Defend

9.10.3 Events

Events

SUB CATEGORIES

• Melee

Melee

MELEE

Events

- On Equip Weapon
- On Input Charge
- On Input Execute
- On Melee Hit
- On Unequip Weapon

ON EQUIP WEAPON

Melee » On Equip Weapon

Description

Executed when the Character equips a new Melee Weapon

Keywords

Equip Unsheathe Take Sword Melee

ON INPUT CHARGE

Melee » On Input Charge

Description

Executed when the Character starts to run a Charge input command

Parameters

Name	Description
Key	The key being used as a Charge command

Keywords

Charge Input Melee Execute Hold Load

ON INPUT EXECUTE

Melee » On Input Execute

Description

Executed when the Character starts to run the Execute input command

Parameters

Name	Description
Key	The key being used as an Execute command

Keywords

Charge Input Melee Attack Strike

ON MELEE HIT

Melee » On Melee Hit

Description

Executed when the Trigger receives a hit from a melee ${\tt Skill}$

Keywords

Active Disable Inactive

ON UNEQUIP WEAPON

Melee » On Unequip Weapon

Description

Executed when the Character removes a new Melee Weapon

Keywords

Unequip sheathe Take Sword Melee

9.10.4 Instructions

Instructions

SUB CATEGORIES

• Melee

Melee

MELEE

Sub Categories

- Defense
- Equip
- Input
- Skills

DEFENSE

Defense

Instructions

- Set Defense
- Set Shield
- Start Blocking
- Stop Blocking

Set Defense

Melee » Defense » Set Defense

Description

Sets the current defensive value of a Shield on a Character

Parameters

Name Description	
Character The Charact	er that has a defensive combat value
Value The new def	ense value, clamped between 0 and the maximum defense value

Keywords

Melee Combat Shield Defense Block

Set Shield

Melee » Defense » Set Shield

Description

Sets the Shield value

Parameters

Name	Description	
То	The location where to store the Shield	
Shield	The Shield asset reference	

Keywords

Start Blocking

Melee » Defense » Start Blocking

Description

Attempts to start blocking with the Melee stance

Parameters

Name Description

Character The Character that starts blocking

Keywords

Melee Combat Shield Parry Deflect Block

Stop Blocking

Melee » Defense » Stop Blocking

Description

Attempts to stop blocking with the Melee stance

Parameters

Name	Description	
Character	The Character that stops blocking	

Keywords

Melee Combat Shield Parry Deflect Block

EQUIP

Equip

Instructions

- Equip Melee Weapon
- · Unequip Melee Weapon

Equip Melee Weapon

Melee » Equip » Equip Melee Weapon

Description

Equips a Melee Weapon on the targeted Character if possible

Parameters

Name	Description
Character	The Character reference equipping the weapon
Weapon	The weapon reference to equip
Model	The optional 3D model instance

Keywords

Unequip Melee Weapon

Melee » Equip » Unequip Melee Weapon

Description

Unequip a Melee Weapon from the targeted Character if possible

Parameters

Name	Description
Character	The Character reference unequipping the weapon
Weapon	The weapon reference to unequip

Keywords

INPUT

Input

Instructions

- · Input Charge
- Input Execute
- Set Buffer Window

Input Charge

Melee » Input » Input Charge

Description

Queues a charging Melee input command on a Character

Parameters

Name	Description	
Character	The Character reference	
Key	The Input key value	

Keywords

Input Execute

Melee » Input » Input Execute

Description

Queues an execution Melee input command on a particular Character

Parameters

Name	Description
Character	The Character reference
Key	The Input key value

Keywords

Set Buffer Window

Melee » Input » Set Buffer Window

Description

Sets the maximum time for an input to register before it can be executed

Parameters

Name	Description
Character	The Character reference
Buffer Window	The time of the Buffer Window, in seconds

Keywords

Melee Combat Buffer Window

SKILLS

Skills

Instructions

- Play Melee Reaction
- Play Melee Skill
- Reset Block Time
- Reset Break Time
- Reset Parry Time
- Reset Skill Hits
- Set Skill
- Try Cancel Skill
- Wait Until Phase

Play Melee Reaction

Melee » Skills » Play Melee Reaction

Description

Plays a Melee Reaction on a Character

Parameters

Name	Description
Character	The Character that plays the Melee Reaction
Attacker	The Character set as the attacker
Reaction	The Melee Reaction asset played

Keywords

Play Melee Skill

Melee » Skills » Play Melee Skill

Description

Plays a Skill on a Character regardless of the weapon or state

Parameters

Name		Description
Chara	acter	The Character that plays the Skill
Targe	et	Optional reference object set as the Target of the Skill
Skil	1	The Skill asset reference to run

Keywords

Reset Block Time

Melee » Skills » Reset Block Time

Description

Resets the registered time of the last blocked attack

Parameters

Name	Description	
Character	The Character reference resetting the value	

Keywords

Reset Break Time

Melee » Skills » Reset Break Time

Description

Resets the registered time of the last broken attack

Parameters

Name	Description		
Character	he Character re	eference resetting	the value

Keywords

Reset Parry Time

Melee » Skills » Reset Parry Time

Description

Resets the registered time of the last parried attack $% \left(1\right) =\left(1\right) \left(1\right)$

Parameters

Name	Description
Character	The Character reference resetting the value

Keywords

Reset Skill Hits

Melee » Skills » Reset Skill Hits

Description

Resets the hit performed by the ongoing ${\tt Skill}$

Parameters

Name	Description
Character	The Character reference resetting the hit buffer

Keywords

Set Skill

Melee » Skills » Set Skill

Description

Sets the Skill value

Parameters

Name	Description
То	The location where to store the Skill
Skill	The Skill asset reference

Keywords

Try Cancel Skill

Melee » Skills » Try Cancel Skill

Description

Attempts to cancel an ongoing Charge, Skill or Reaction being executed by a character

Parameters

N	lame	Description	
C	Character	The Character reference using a Charge, Skill or Reaction	

Example 1

If you want to cancel only a specific phase (like a Reaction) check whether the current phase being played is that one

Keywords

Melee Combat Skill Stop Reaction Charge

Wait until Phase

Melee » Skills » Wait until Phase

Description

Waits until the current Skill's phase reaches the chosen one

Parameters

Name	Description
Character	The Character reference
Phase	The Phase which waits to

Keywords

Melee Combat Anticipation Strike Recovery Finish Combo Skill

9.11 Releases

9.11.1 Releases

2.1.8 (Latest)

Fixes Examples: Missing hit detections Examples: Focus using gamepad

2.1.7

Pixes Skill: On Hit instructions run before victim Reaction Editor: Compatibility with Core 2.14.46

2.1.6

```
eased October 31, 2023
 This version breaks compatibility with previous versions and will only work with Game Creator 2.13.43 or higher.
 Component: Can Hit for hitting non-Character objects
 Property: Melee Self to Target Location
 Property: Melee Target to Self Location
 Instruction: Set Skill
· Instruction: Set Shield
 Skill: On Strike sound effect option
 Changes
· Internal: Support for Core 2.13.42 version
 Fixes
 Sequencer: Instructions called after cancelling
 Sequencer: Default empty values
 Sequencer: Duplicate value when creating new Clip
 Reactions: No reaction does not stop Gestures
 Skills: Audio Clips use caster time scale
· Skills: Motion Warping not detecting collisions
 Removes
Properties: Location Melee Properties
```

2.0.5

Pixes

Examples: Missing variables in some scenes

2.0.4

Leased June 13, 2023

Fixes

Trigger: On Hit has attacker as Target

Examples: Missing Local Name Variables

2.0.3

New
Skill: Condition Can Hit when striking targets
Variables: Melee Weapon, Shield and Skill types

Fixes
Trigger: On Hit calls every frame during attack
Trail: Not rendering on URP/HDRP
Examples: Dash not working correctly

2.0.2

New

Uninstall: Added option to uninstall the module

Fixes

Example: First-Person example missing character

Version: Not showing current module version

Misspell: Example scene typographic error

2.0.1

New First release

10. Traversal

10.1 Traversal



This module is currently under developement

11. Extensions

11.1 Welcome to Extensions

Extensions are free packages that can be downloaded from the official Game Creator site. As its name implies, **Extensions** add new features that can be easily used.

Download Extensions

11.1.1 Installation

To install an **Extension**, download it from the **Downloads** page. With your Unity project open, double click the ..unitypackage file and a screen will prompt you to choose which files you want to add to your project.



If it's the first time you're installing an extension, we recommend installing all files. Once you have more experience with it, you can cherry pick which examples to install and which ones to ignore.

An **Extension** will appear like any other normal module and can be uninstalled clicking on the top toolbar Game Creator Uninstall... and picking the desired module or extension to delete.

11.2 Transitions

11.2.1 Transitions

The **Transitions** extension allows to load a new scene using a custom loading screen, that may include game tips, random backgrounds, animations and other kinds of visual elements.

Transitions Example

These loading screens can be easily interchanged using the **Transition** asset, which is used in one of the Instructions.

11.2.2 Transitions

Transitions are configured in an asset, which accepts a prefab with a **Transient** component, and a collection of optional animations that are used to fade in, out and an idle the interface screen.

Transitions Asset



We recommend duplicating any of the built-in **Transition** assets and modifying it to create a custom one for your game.

This asset can be used with any of the Instructions available.

11.2.3 Instructions

There are a few Instructions available in this module.

Transition to Scene

To transition from one screen to another one, use the **Transition to Scene** instruction, which can be found under the *Transitions* category in any visual scripting dropdown list.

Transition to Scene Instruction

The first field must reference a valid **Transition** asset, which determines the type of screen that appears when loading the scene.

Ticking the **Wait Activation** checkbox will load the scene, but won't activate it until something executes the **Transition Complete**.



Ticking the **Wait Activation** allows long loading screens to remain after they've been loaded, so it's the user who decides when they are ready to play the next level.

By default, all built-in **Transitions** wait until the player presses any input key to continue, though this can be modified inside the transition prefab.

The Scene field allows to choose which scene to load next.

You can also specify wether to load the next scene Additively or unload every other scene and just load a new one, using the Single option from the Mode field.

The **Entries** section below, just like the **Load Scene** instructions, allow to define where each object from the next scene should be position. This is mostly used to post the Player at the correct door entrance when using this method.

kunning Time

It's important to note that the loading screen doesn't pause/stop the game while it starts loading.

This means that the Player could get hit and die while the loading screen's animation is starting to run, so it's worth considering adding mechanisms that prevents having gameplay issues with it.

Transition Start

The **Transition Start** instruction is used to play a specific **Transition**, just like the previous instruction. However, it won't load a new scene. This can be used to move between cameras. For example, playing a kill-cam using a transition.

Transition Complete

The Transition Complete instruction allows to resume and finish the current Transition being played.

This is most commonly used to transition out from a current loading screen, that's waiting for a Player input in order to activate the newly loaded scene.

11.3 Localization

11.3.1 Localization

The Unity Localization asset allows to manage and show a game translated to two or more languages.

bocumentation for Unity Localization

The **Localization** asset comes packed with a lot of out-of-the-box components that make it effortless to add translations to your games. This page assumes you're a bit familiar with its workflows. You can learn more about it at the **Unity Localization Documentation**

Display translations

This integration allows to localize not only texts, but also *Sprites*, *Textures* and even entire *Game Objects*, and in all cases it's the exact same workflow.

To display a localized Text use the Localized Text option from any of the text properties dropdown.

Localization Property Text

Once the property is created, you can choose a text entry from the Localization Table or create a new entry.

Localization Property Details



After selecting the table entry or creating a new entry, the **Instruction** node title might not update its text. This is because Unity 2022.3 and previous versions use *IMGUI* to render some sections of Unity Localization. You can simply collapse and expand the instruction again to manually refresh the title.

Change the Language

To change the language (also known as *Locale*), you can use the **Change Language** instruction. It allows to type in a string value with the locale of the language to display.

Localization Change Language Instruction

About Locales

Locales are short strings that represent a language. For example, English is usually typed as en and Spanish as es . There are also regional locales, such as en-CA for Canadian English or en-US for United States English.

Detect Language Changes

To detect when the currently selected locale has changed, use the **On Change Language** event Trigger, which is executed every time a new language is selected.

Localization On Change Language Trigger Event

** Refresh

Depending on the scene, it might not be possible to update the texts displayed on screen to the new language. In these cases, it's best to simply reload the scene after changing the language, so all text and assets that are localized are re-constructed again with the new locale values.

11.3.2 Examples

The **Localization** integration kit comes with an example scene that shows how to change a *Sprite* and a text based on the currently selected language.



Because Unity Localization uses *Addressables* to load in the translation tables, there is unfortunately no way to install the example with everything set up. This page will guide you on how to properly configure it.

Setup

Once the **Localization** package and integration kit have been installed, open the Game Creator's *Install* window and proceed to install the **Localization Examples**.

Localization Install Examples

Select and open the example scene. The *Hierarchy* view shows a canvas with a white square, and two buttons that read *English* and *Spanish*. These buttons change the language of the game, using the **Change Language** instruction

The Trigger_Start is called as soon as the game starts, and simply executes the instructions from the trigger below.

The **Trigger_Set_Language** is executed as soon as the game language is changed. This updates the *Image* component with a localized one, as well as the text of the *Title*.

Head to the top toolbar and open/create the **Localization Tables** by selecting *Window Asset Management Localization Tables*.

Localization Create Table

If you haven't created any tables yet, the window will prompt you to create a *Settings* asset and where you want to save it.

After that, you'll be able to create a new **Localization Table** by clicking on the left corner of the window, where it says "New Table Collection". This will create an asset that stores either string (texts) or asset translations.

Before creating any tables, the Locales need to be defined.



Locales are short texts that represent a language. For example es means *Spanish* and en represents *English*. There are also regional locales, such as en-UK which is the *English* spoken on the United Kingdom.

Click on Locale Generator and select English and Spanish (en and es respectively).

Let's create two Table Collections:

- · MyTexts: Select the String Table Collection to store the localized texts of the game.
- MyAssets: Select the Asset Table Collection to store the localized assets of the game.

Select the MyTexts and add a new entry called game-title and set the following texts:

- en: Dawn of the Wolf
- es: El Amanecer del Lobo

Select the MyAssets and add also a new entry called game-poster with the following sprites:

• en: Poster_EN.png

• es: Poster_ES.png

You'll find these sprites inside the example folder at Assets/Plugins/GameCreator/Installs/Localization.Examples/Localization/Sprites/.

Localization Assets Table with Posters

Example

Now that the example is set up, all that needs to be done is to link the table collection data with the *Image* and *Text* components from the scene.

To do so, select the ${\bf Trigger_Set_Language}$ trigger and expand both instructions.

- On the Set Sprite instruction, change the Sprite field to Localized and select the key game-poster key.
- · On the Set Text instruction, change the Text field to Localized and select the key game-title key.



Alternatively you can also set the entry from inside the Property of the Instruction.

Localization Instructions Complete

Click play and see how clicking on the buttons changes the game's language as well as the *Image* and *Title* texts.

11.4 Addressables

11.4.1 Addressables

The **Addressables** integration allows to better manage your game's memory footprint and have control over when objects are loaded on memory and when they are released.

About Addressables

This page assumes you are familiar with Unity's **Addressables** workflow. If not, check the official documentation on **Unity Addressables**.

Addressables Load from Property

The easiest way to load an object from an Addressable Group is choosing the Addressable option from a property field, and dropping in either the Addressable ID or the Asset Reference object.

When attempting to retrieve this object, the main thread will be blocked until the object is loaded. This option is perfect for small objects that are found inside the executable and do not need to be downloaded through the internet.

Automatic Release

When using addressables via properties, the object loaded is automatically scheduled to be released on the next frame. If you want to keep the object in memory so it can be used without loading it back again, use the **Load Addressable** instruction.

If you prefer to decide when to load an addressed object and not unload it just afterwards, you can use the instruction **Load Addressable Asset**.

Addressables Load Asset Instruction

This instruction allows to load an addressable object using three mechanisms:

- Synchronous: Blocks the main thread and won't resume it until the object is loaded. This should not be used unless the object is very small and is bundled with the executable.
- Asynchronous Wait: Starts loading the asset in the background, and the instruction waits until it's completed. The next instruction will either have the asset loaded (unless it has failed, and the value is then null).
- Asynchronous Forget: Starts loading the asset in the background but does not wait until it has been completed. We do not recommend using this method unless you know what you're doing.

Once the asset is loaded it can be instantiated without worrying about bringing it from disk to memory (or server to memory).

To release an asset from memory, use the **Release Addressable Asset** instruction. This will automatically remove it from memory.

Addressables Release Asset Instruction

11.4.2 Examples

The **Addressables** integration kit comes with an example scene that shows how to instantiate a game object from an addressable asset, as well as another one that pre-loads it in the background.



Due to how Unity Addressables is coded, it is not possible to share addressable groups, and thus the example scenes require some minor setup. This page explains the steps to do so.

Setup

After installing the **Addressables** package as well as the **Addressables Integration** kit from the **Downloads** page, open the *Install* window and proceed with the installation of the **Addressables** Example.

Addressables Install Examples

Once installed, click on the *Select* button or navigate to Assets/Plugins/GameCreator/Installs/Addressables.Examples/, where there are three items:

- The scene **1_Instantiate_GameObject**, which shows how to easily instantiate a game object using an addressable.
- The scene 2_Load_And_Instantiate, which shows also how to instantiate a game object, but preloading it beforehand.
- The prefab Cube_Prefab that is used as an example object to be used in both examples.

Before opening any scene, the **Addressables** settings must be configured. To do so, open the *Addressables Group* window by selecting from the top toolbar *Window Asset Management Addressables Groups*.

If you don't have any **Addressables** settings configured, the window will prompt you to create them. Click on the *Create Addressable Settings* and wait until it completes.

Addressables Setup Window

Now that the settings are complete, we can proceed to configure the examples.

Examples

Open the first example scene **1_Instantiate_GameObject**. This scene has a UI that allows to instantiate a game object by clicking a button.

Select the **Actions** object at the bottom, which is responsible for instantiating the **Cube_Prefab** object. Before doing so, we need to register this asset as an **Addressable Asset**.

To do so, it's very easy. Simply open the **Addressables Group** window just like we did in the previous section, and drag and drop the **Cube_Prefab** prefab onto the window.

Addressables Addressables Groups

Now that the **Cube_Prefab** is marked as an addressable, head to the **Actions** object and click on the right-most object picker from the *Addressable Reference* field and choose the prefab we just set up.

Addressables Instantiate Addressable Game Object

That's it! Entering play-mode will allow to click onto the *Instantiate* button, which loads and instantiates the game object at the center of the screen.

The second example's configuration is pretty much the same, except there's another **Actions** component called *Actions_Load*, which preloads the prefab before instantiating it. All that requires is to choose the **Cube_Prefab** addressable reference from the object picker, just like in the previous example.

Addressables Instantiate Addressable Game Object

Entering play-mode will not allow to instantiate the object directly. Instead, the object must be loaded before hand. Once it's finished, the *Instantiate* button will become enabled and ready to be used.

11.5 Footsteps

11.5.1 Footsteps

The **Footsteps Generator** kit is an extension that allows to setup existing animations and turn them into Game Creator compatible clips with correct foot placement.

How Footsteps work

Game Creator uses the animation parameters Phase-0, Phase-1, Phase-2 and Phase-3 to detect which feet and when a foot is on ground.



Humanoid characters use Phase-0 and Phase-1 for their left and right leg respectively. Non-humanoids can use phases in any other order.

For example, when the Phase-0 curve point has a value of 0 means the foot is on air. If a point has a value of 1 means the foot is in a grounded phase.

Creating Phases

To add or modify an animation curve phase, open the **Settings** window by clicking on the top toolbar Game Creator Settings, and navigate to the **Footsteps** section.

Footsteps Window

To start creating or modifying an **Animation Clip** phase group, click on the button *Enter Footsteps Mode*. The scene view and hierarchy panel will change into one similar to the ones when editing a prefab.

change Character

If you're working on a non-humanoid or a different character than the default one, drag and drop your prefab model onto the field below the previous button and click the **Change Character** button. This will change the preview character.

Drag and drop the Animation Clip onto the animation field. After doing so, the rest of options will be available

You can use the slider below to scrub through the animation in order to preview it in the scene view.

Footsteps Configuration

There are a few options below:

- Set as Grounded button allows to automatically set all 4 curves at a constant value of 1. This is meant for idle poses where the character doesn't lift its feet from the ground.
- Set as Airborne button does the opposite and sets all 4 curve phases to a constant value of θ . This is meant for any airborne animations, such as falling, jumping and such.

The 4 animation curves can also be manually edited below, and committing the changes by pressing the **Set Animation Curves**.

Creating Humanoid Phases

To speed up the workflow, this tool also allows to detect when the feet are above or below ground level and set the curve values automatically. To do so, simply click the **Automatic Human Footsteps**.

The **Ground Threshold** value determines an offset vertical value where the ground would be. If the curves appear to be jittering, try playing with values between 0.01, 0.05 and 0.1.