Daimahou Gameplay Modules

Documentation

Gerard Nicolas

Table of contents

1. Core

1.1 Welcome to Daimahou

Daimahou is a *collection of modules* specifically designed to blend with the Game Creator 2 eco-system. The **philosophy of Daimahou** is to create tools to **rapidly prototype** particular game features in a very short amount of time.

The main *goal* of Daimahou is to ***remove all the technical steps*** to add features to your game so that you can **focus on the design aspect** which will make your game shine. By removing all the setup friction, you are free to experiment with feature combination, customize their behaviour and quickly converge to the fun part of your gameplay idea.

With this in mind, welcome to the core documentation.

1.1.1 Errata

If you find a mistake or omission in the documentation, please send us an email at daimahou.studio@gmail.com 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 Systems

1.2.1 Main Systems

The core module is mostly there to extend Game Creator's core and provide a basic framework for the other modules to rely on. This documentation will present the main feature that might be useful to you, as you embark on this journey.

- **Pawn** : main entry point to daimahou systems. Used to enable functionality for each game object and configure its settings. At runtime useful information will be displayed in the inspector.
- **Reactive Gestures**

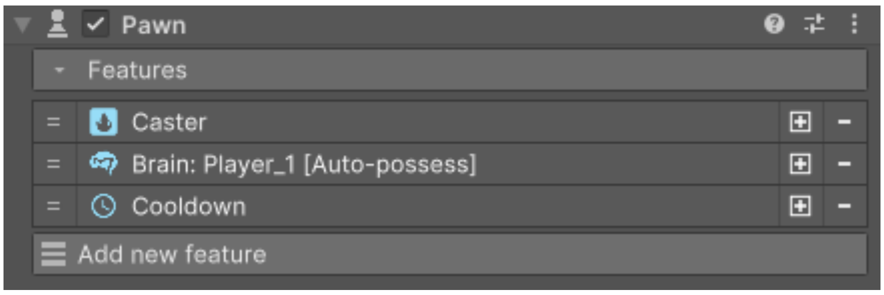
1.2.2 Pawn

Pawn Component

Pawn is the main component used to add functionality and configuration to game objects. Through the pawn inspector, you can add and remove various features to customize the behaviour of your game object very easily.

During play, the inspector will display useful information to allow you to understand exactly what is happening at any given time.

You can add a pawn component to your game objects that interact with Daimahou systems by clicking *Add component* in the inspector and navigating to *Game Creator -> Daimahou -> Pawn*



Debugging info

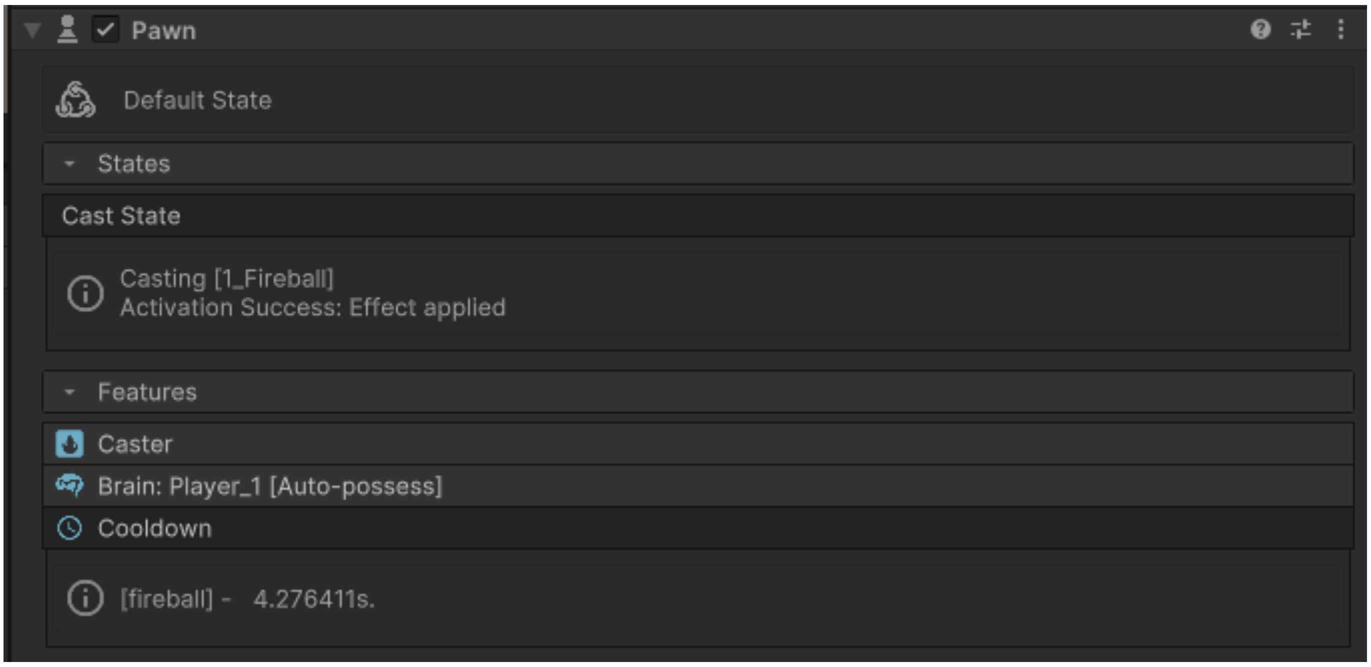
The pawn component can show useful debugging information during runtime. In case of a problem, feel free to add a screenshot of it to add to your support request as this might help fixing bugs !

States

An important thing to understand is that a Pawn can have various behaviours depending on the features that are enabled. For example, the motion feature gives the pawn the ability to move while the cast feature will give the ability to cast spells. When engaging in one behaviour, the pawn will enter a corresponding state. Each state will determine if it can be canceled potentially preventing other behaviour to start while the pawn is already busy.

The current state and previously used states can be inspected in the pawn inspector at runtime. Useful information can be found, for example, the reason that prevents a pawn from entering a given state will be written in the inspector to facilitate the debugging and understanding of the system.

States have internal rules that specify when they can or cannot be overridden. For example, the move state can be freely overridden by the cast state, but the other way around.


Pawn

Default State

States

Cast State

Casting [1_Fireball]
Activation Success: Effect applied

Features

Caster

Brain: Player_1 [Auto-possess]

Cooldown

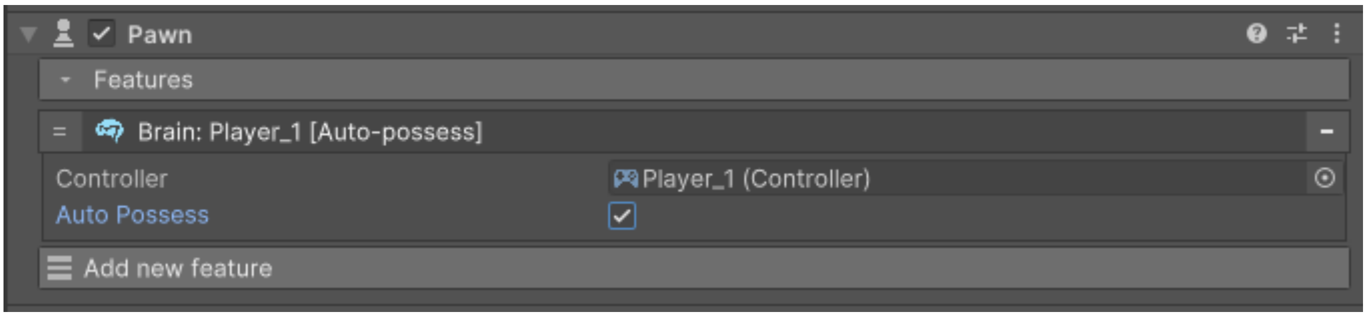
[fireball] -   4.276411s.

Features

BRAIN

Brain is a feature that allows a pawn to be bound to a controller.

- **Controller** : Specify the controller that will be used to take control of the pawn.
- **Auto Possess** : when activated, the pawn will be automatically possessed by the controller upon starting the game.



Controller: Player_1 (Controller)
Auto Possess: ✓

Only one pawn can be possessed at a time

Only one pawn can be possessed at a time, so if multiple pawns are set to be possessed automatically by the same controller, then a random pawn will be possessed instead.

States

DEFAULT STATE

Default empty state.

On Enter

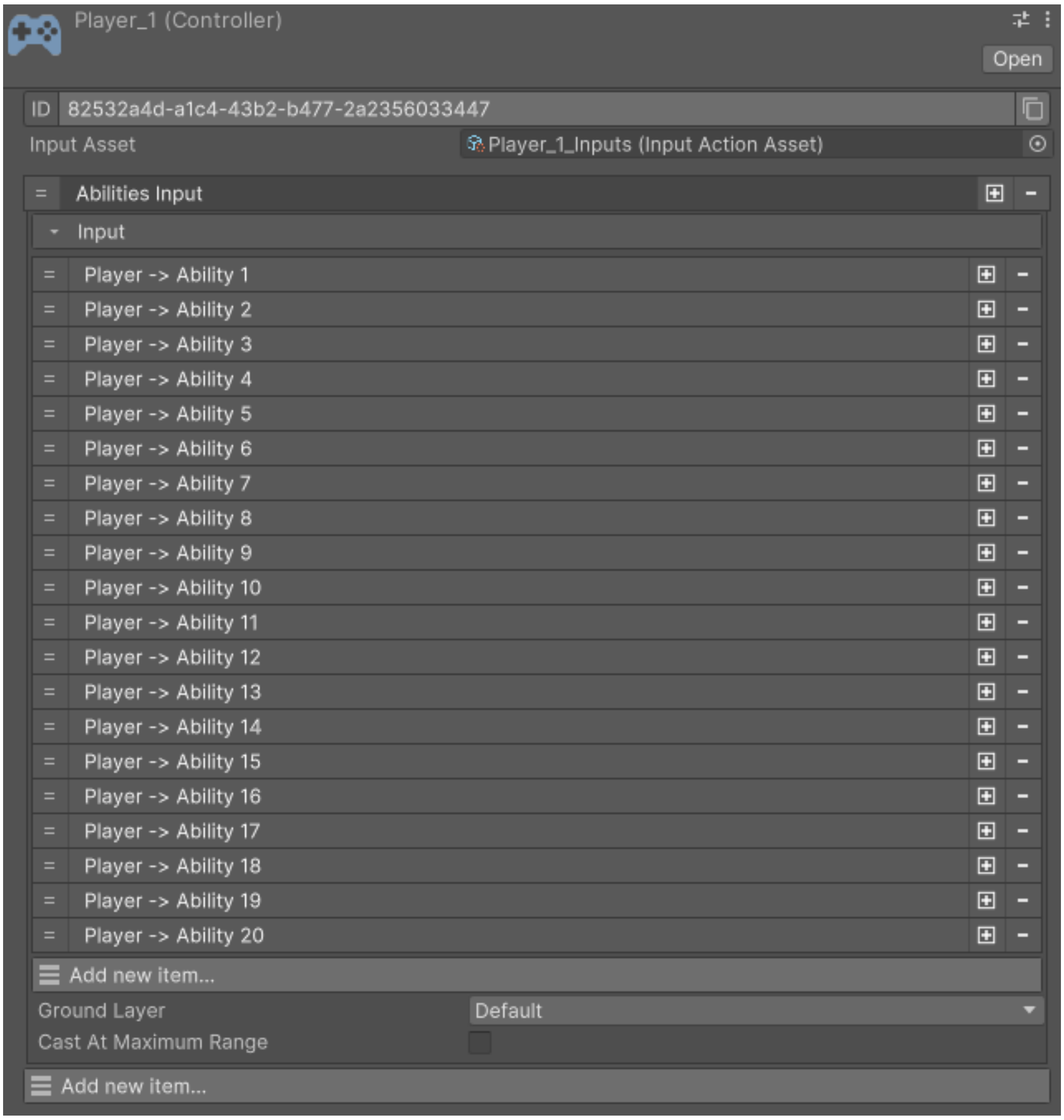
Upon entering this state the character is set to be controllable if it was not already the case.

1.2.3 Controller

A controller is an asset that represents a player controller in game. It is used to easily bind controls to functionality.

A controller holds a series of input mappings which can be interpreted by each respective module to create gameplay behaviour. Each button can be simply bound to a regular unity input action to customize which button corresponds to which action.

For a concrete example, have a look at the Abilities input module.


Player_1 (Controller)

Open

ID 82532a4d-a1c4-43b2-b477-2a2356033447

Input Asset: Player_1_Inputs (Input Action Asset)

Abilities Input

Input

Player -> Ability 1
Player -> Ability 2
Player -> Ability 3
Player -> Ability 4
Player -> Ability 5
Player -> Ability 6
Player -> Ability 7
Player -> Ability 8
Player -> Ability 9
Player -> Ability 10
Player -> Ability 11
Player -> Ability 12
Player -> Ability 13
Player -> Ability 14
Player -> Ability 15
Player -> Ability 16
Player -> Ability 17
Player -> Ability 18
Player -> Ability 19
Player -> Ability 20

Add new item...

Ground Layer: Default

Cast At Maximum Range: ☐

Add new item...

Controller example

Creating a new controller

Controllers are scriptable objects and to create one, you'll need to right click on the Project Panel and navigate to Create → Game Creator → Input → Controller.

1.2.4 Reactive Gesture

Reactive Gesture

Reactive gestures allows to synchronize logic events and store data directly with the animation. This is particularly useful to trigger sound effects and visual effects at the exact timing.

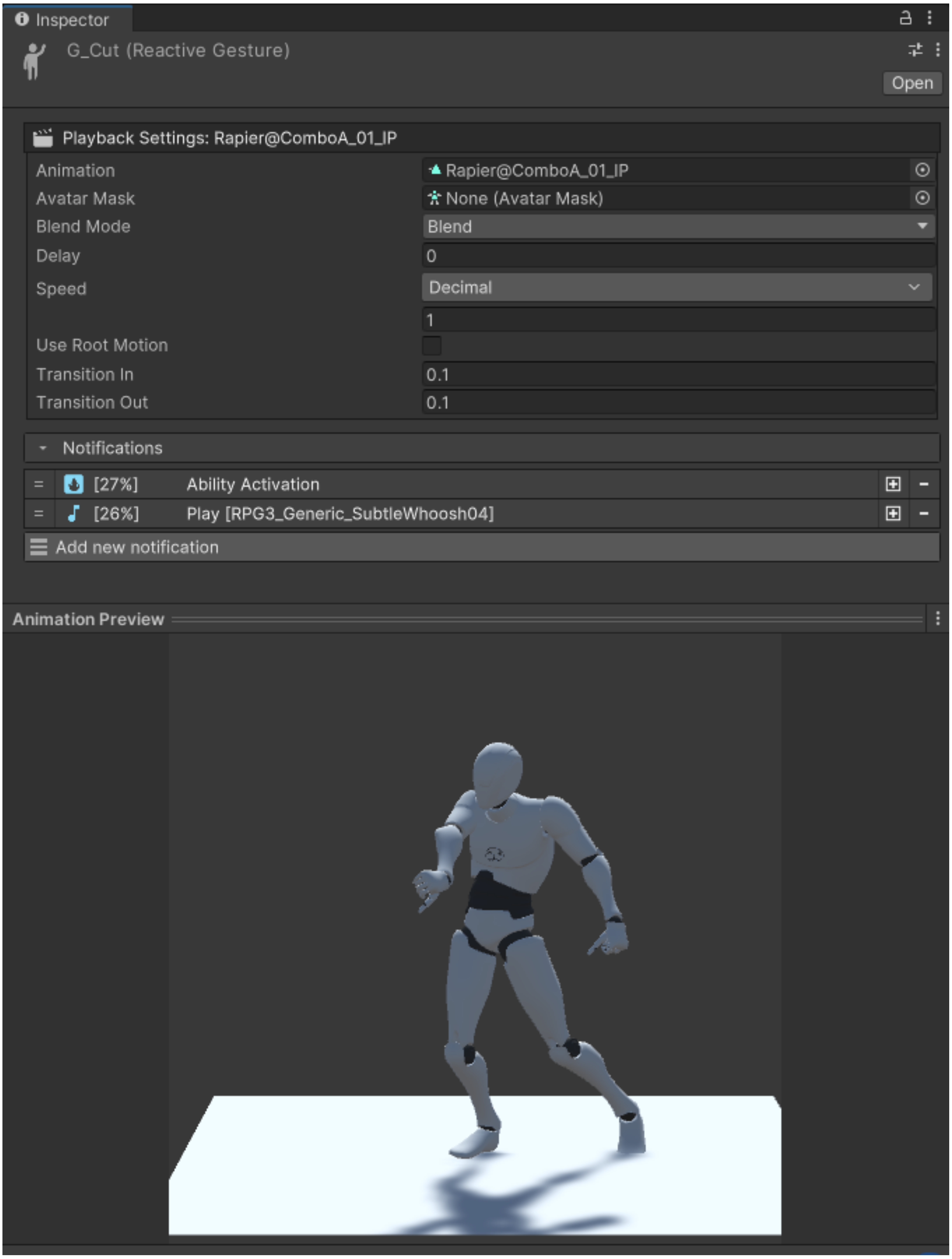
The system is also used to trigger the activation of an ability so that, for example, the damage is only delivered at the exact time the sword connects with the target. Moreover, the system also allows the user to execute custom logic and synchronize its execution with the animation.

Creating a reactive gesture

Reactive gestures are scriptable objects and to create one, you'll need to right click on the Project Panel and navigate to *Create → Game Creator → Character → Reactive Gesture*.

Reactive Gesture Inspector

A reactive gesture is a container for an animation. In it, you can define notifications that will be triggered when the animation is played. Each notification can be set to trigger at a specific time during the animation. This system is used to easily synchronize the visual element of the animation with its logic.



Inspector

G_Cut (Reactive Gesture)

Open

Playback Settings: Rapier@ComboA_01_IP

Animation	Rapier@ComboA_01_IP
Avatar Mask	None (Avatar Mask)
Blend Mode	Blend
Delay	0
Speed	Decimal
	1
Use Root Motion	
Transition In	0.1
Transition Out	0.1

Notifications

[27%]	Ability Activation
[26%]	Play [RPG3_Generic_SubtleWhoosh04]

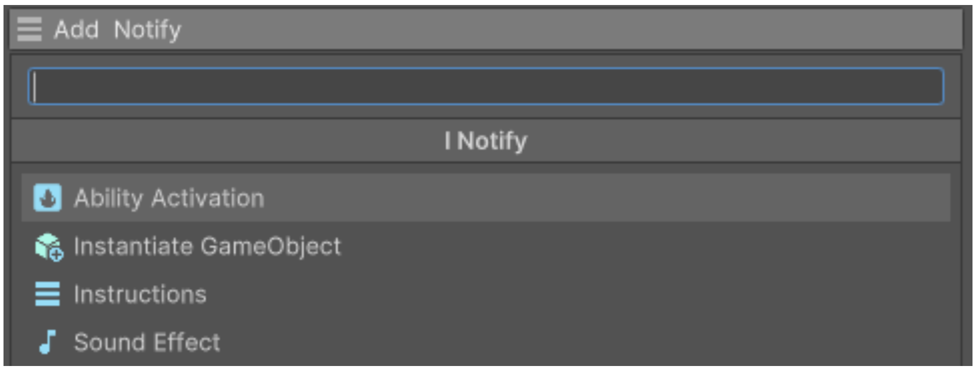
Add new notification

Animation Preview

Animation preview

When selected, if the animation clip field is filled with an animation, the animation preview will show the animation in real time. When dragging the slider of a particular notification, the animation will adjust to show the character position at this time frame.

Notifies



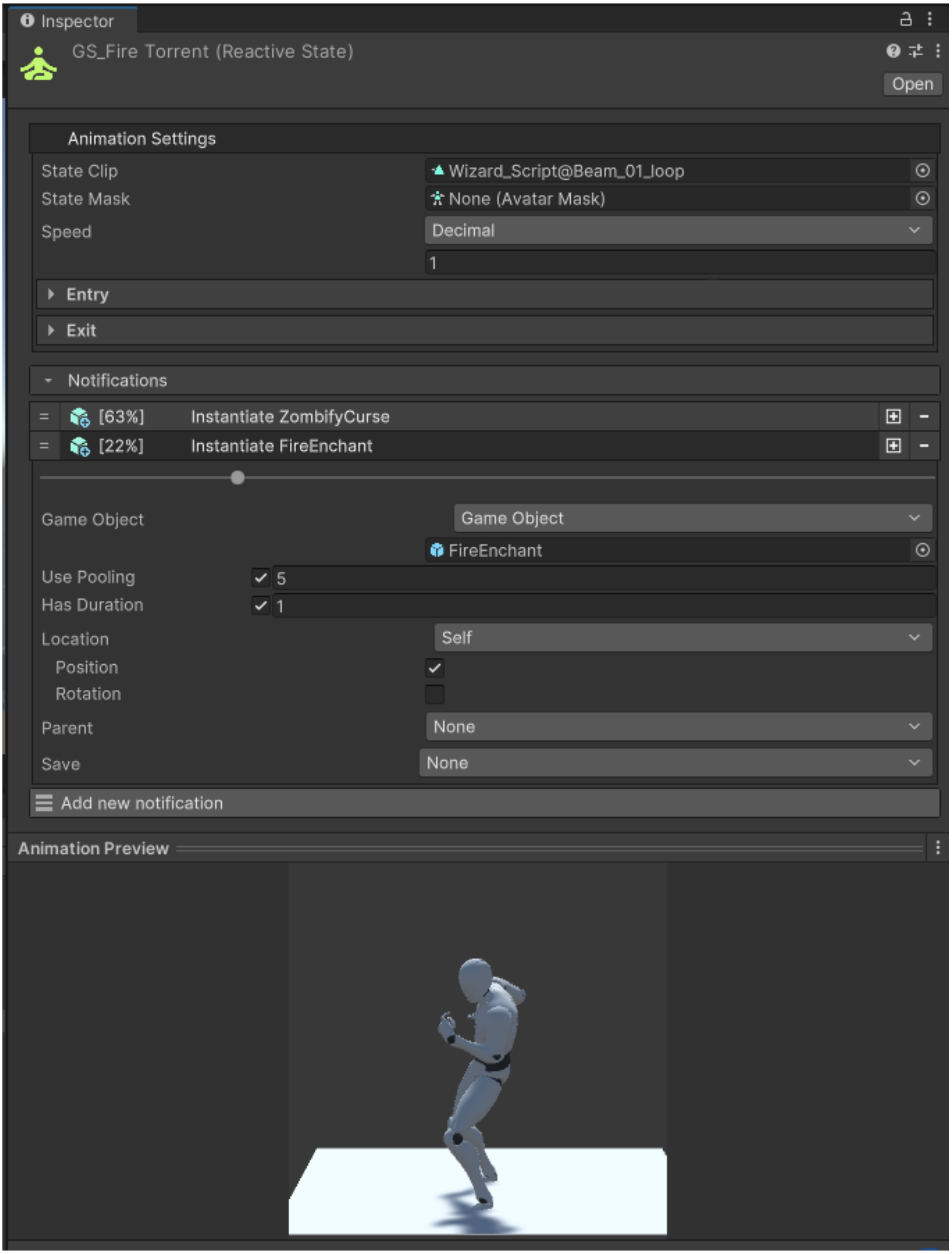
There is 4 different notifications at this time with relatively straightforward behaviour.

- **Ability Activation**: sends a notification to the ability system that will complete its activation.
- **Instantiate Gameobject**: works exactly like the Game Creator instruction. This one can be seen as a shortcut to spawn VFX.
- **Sound effects**: works exactly like the Game Creator instruction. This one can be seen as a shortcut to spawn SFX.
- **Instructions**: a list of Game Creator instructions that can be used to trigger custom logic.

Reactive Gesture State

Reactive gesture states are the reactive gesture counterpart of Game Creator State.

The only difference is that they include a timeline for the notifications. There is only one notification timeline which covers the entirety of the state, including the optional entry and exit animations. The portion of the notification timeline which corresponds to the looping animation is repeated each loop automatically.



Inspector

GS_Fire Torrent (Reactive State)

Open

Animation Settings

State Clip	Wizard_Script@Beam_01_loop
State Mask	None (Avatar Mask)
Speed	Decimal
	1

Entry

Exit

Notifications

[63%]	Instantiate ZombifyCurse
[22%]	Instantiate FireEnchant

Game Object	Game Object
	FireEnchant
Use Pooling	✓ 5
Has Duration	✓ 1
Location	Self
Position	✓
Rotation	
Parent	None
Save	None

Add new notification

Animation Preview

1.3 Game Creator Extensions

1.3.1 Character

Rotation Units

LOOK AT LOCATION

Specify a location for the character to look at.

PAWN - PIVOT

Has the same behaviour as the Pivot rotation, except that it also can lock its rotation to look at a target. The purpose of this pivot is to avoid a runtime error that happens when changing the facing unit of a character at runtime while inspecting said character. It is recommended to use this rotation unit when using pawns, especially with the Abilities module.

1.4 Visual Scripting

1.4.1 Instructions

DEACTIVATE CHARACTER

Simple instruction that deactivate a character completely. Deactivated character have no more physics attached to them. This is useful when killing a character, you can set the character state to be a death animation and then deactivate it so that it no longer blocks other characters.

1.5 Releases

1.1.0

Overall polish of all inspectors

- Pawn inspector
- Feature configuration
- Pawn state inspector
- Reactive Gesture State

1.5.1 1.0.0 Release version 1

Initial release

2. Abilities

2.1 Welcome to Abilities

Every RPG lover has dreamt of their ideal ability systems, thinking about how spells would interact with each other to create interesting dynamics. Creating such systems is a difficult task, this is why we developed Abilities, *a series of tools that will allow you, the designer, to implement YOUR ideas intuitively in a short amount of time*.

2.2 General information

2.2.1 Who is it for ?

Abilities have been created for **designers**. The philosophy behind abilities is to create a wide set of tools that designers can use to craft their own abilities without the need for additional code. Built upon Game Creator 2, you get access to an incredible wealth of tools to supercharge your creativity.

Abilities is packed with features of its own, targeting system, animation synchronization, projectiles and more. Designed to play well with other modules, you can **stop worrying about technicalities and focus on making games**.

2.2.2 What is it ?

A suite of tools that allows the designer to create abilities with synchronized animation and effects.

- **Reactive Gesture**: allows you to synchronize animation effortlessly
- **Abilities**: allows you to create abilities with customizable components.
- **Projectiles & Impacts**: allows you to configure projectiles with custom trajectory, spawn pattern and impact effects.

Abilities are defined by 5 different sub-systems designed to be flexible and easy to use.

- **Targeting system**: Let you choose how the targets are selected.
- **Activation system**: Let you choose how the ability is executed - e.g. is it a direct spell or a channelled spell.
- **Requirement system**: Let you define conditions that need to be met for the ability to be casted - e.g. cooldown, mana cost, or even arbitrary conditions.
- **Filter system**: Let you define conditions that need to be met for a target to be valid. Useful for removing friendly fire, or to create conditional effects - e.g. targets need to be on fire to be affected.
- **Effect system**: Let you specify what the ability will do once valid targets have been selected - e.g. applying damage, spawning projectiles, etc.

2.2.3 How does it work ?

Very simple, yet extremely powerful approach. There are only 4 steps to create all the abilities you dreamt of.

1. Configure your animations using the reactive gesture system to synchronize your effects and instructions precisely how YOU want.
2. Configure your ability strategies so they apply to the targets of YOUR choice.
3. Specify the effects of your ability to match YOUR design.
4. Spice up your ability by adding projectiles and impact which you can customize to create the abilities YOU envision.

2.2.4 Errata

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

2.3 Installation

2.3.1 Installation

This page will guide you through the installation of the **Abilities** module for Game Creator.

Creating a new project

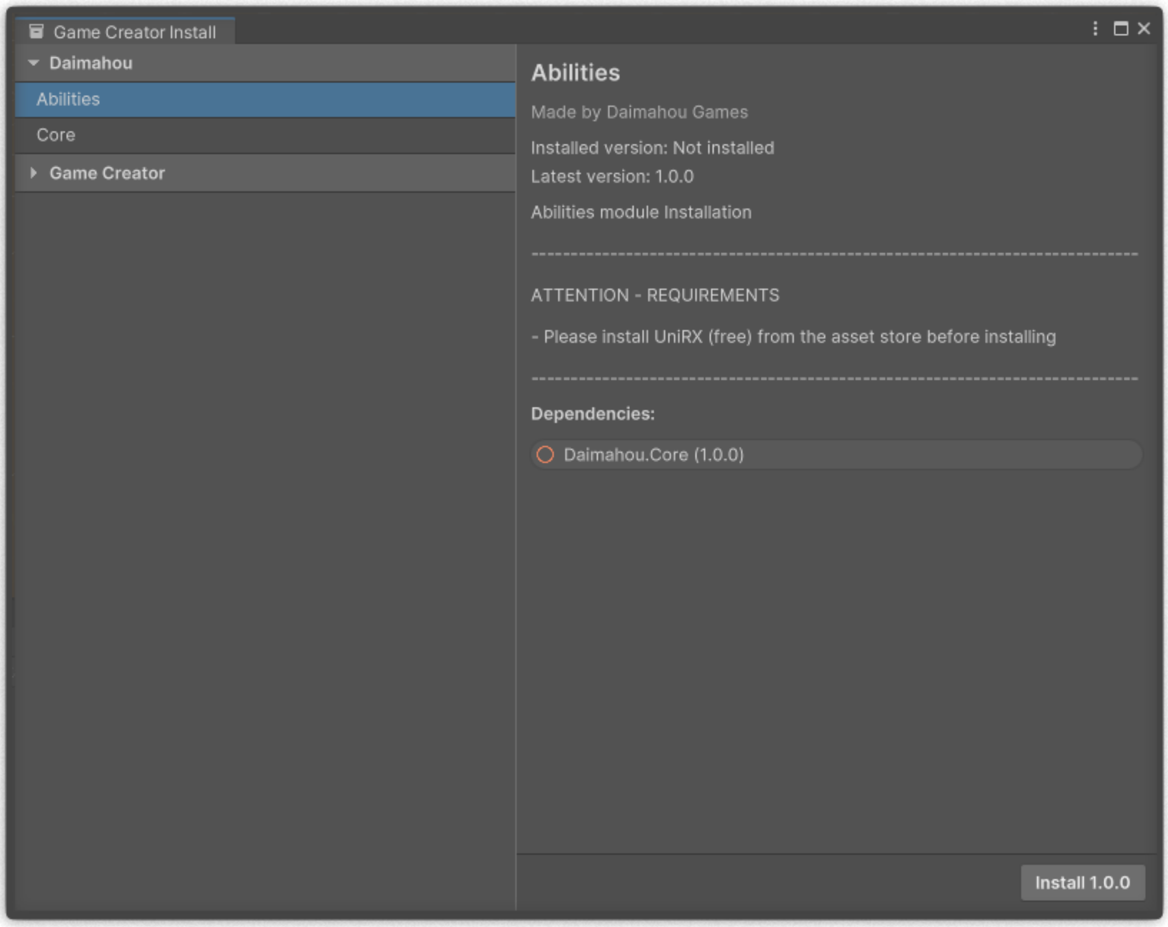
To use abilities, first you need to install the latest version of **Game Creator**. You can find the detailed procedure by clicking on the documentation link.

Get the **Abilities** package from the Unity Asset Store following the link below:

Get Abilities

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 **Ability** package selected. Click on Download and Import afterwards.

Once this process is completed, you will have a new "Game Creator" menu at the top-toolbar. Open the menu and click on *Install...* The following window will open.



Abilities

Made by Daimahou Games

Installed version: Not installed

Latest version: 1.0.0

Abilities module Installation

ATTENTION - REQUIREMENTS

- Please install UniRX (free) from the asset store before installing

Dependencies:

Daimahou.Core (1.0.0)

Install 1.0.0

Install window - Core

Select **Abilities** in the menu tree, and then click on *Install x.y.z* button. This will automatically install all the necessary packages.

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.

Using Git ? Add the following lines to your .gitignore file

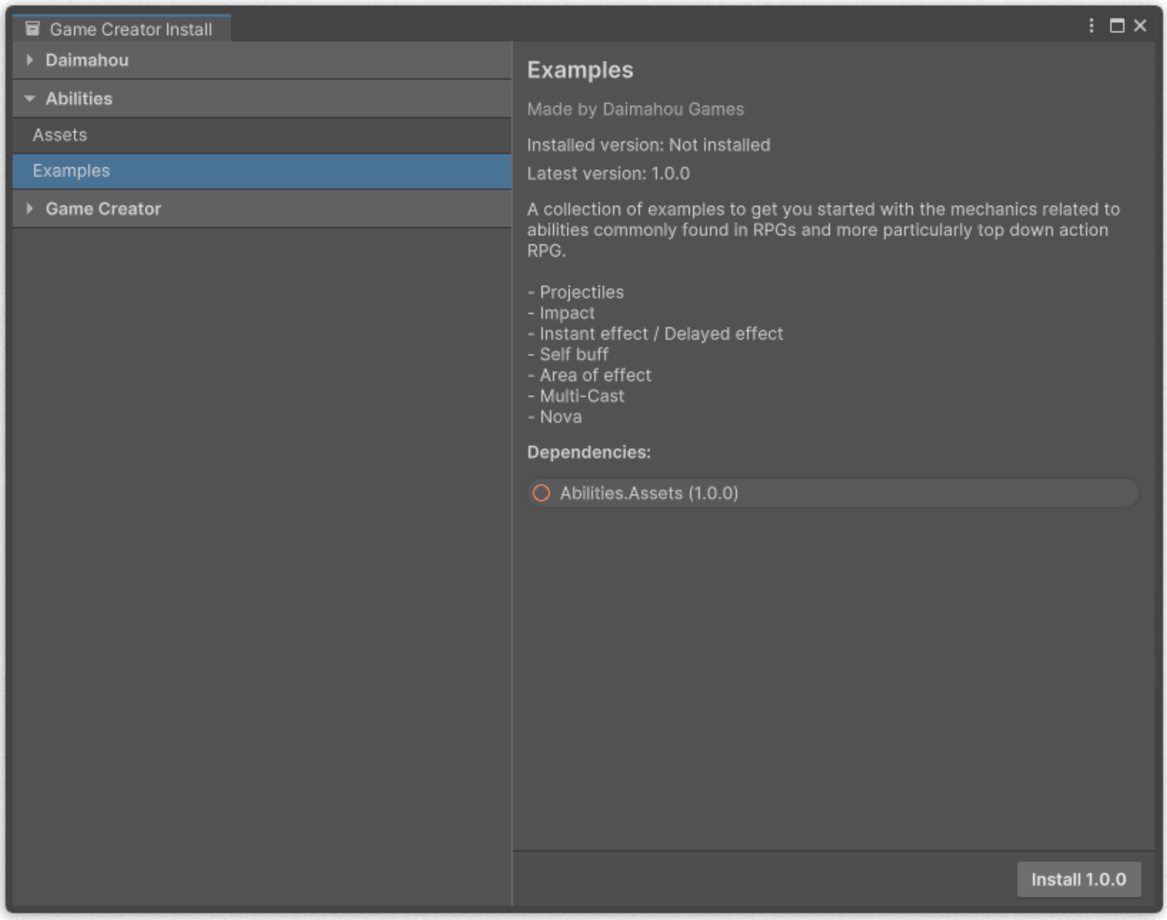
```
# Daimahou Games

/Assets/Plugins/DaimahouGames/Packages
```

Examples

After the successful installation of both **Game Creator** and **Abilities**, another menu will appear in the Installation window. This menu will provide you with additional content in the form of *examples* to get you familiar with the different concepts that make up **Abilities**.

For more information on how to install *examples*, please refer to the following link.

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



Examples

Made by Daimahou Games

Installed version: Not installed

Latest version: 1.0.0

A collection of examples to get you started with the mechanics related to abilities commonly found in RPGs and more particularly top down action RPG.

- Projectiles
- Impact
- Instant effect / Delayed effect
- Self buff
- Area of effect
- Multi-Cast
- Nova

Dependencies:

Abilities.Assets (1.0.0)

Install window - Examples

Uninstallation

Although examples can be uninstalled by simply clicking the Uninstall button. To uninstall Abilities and the Core module of Daimahou, simply pressing the uninstall button will NOT be enough. AFTER pressing uninstall, please delete the following folder :

```
/Assets/Plugins/DaimahouGames/Packages
```

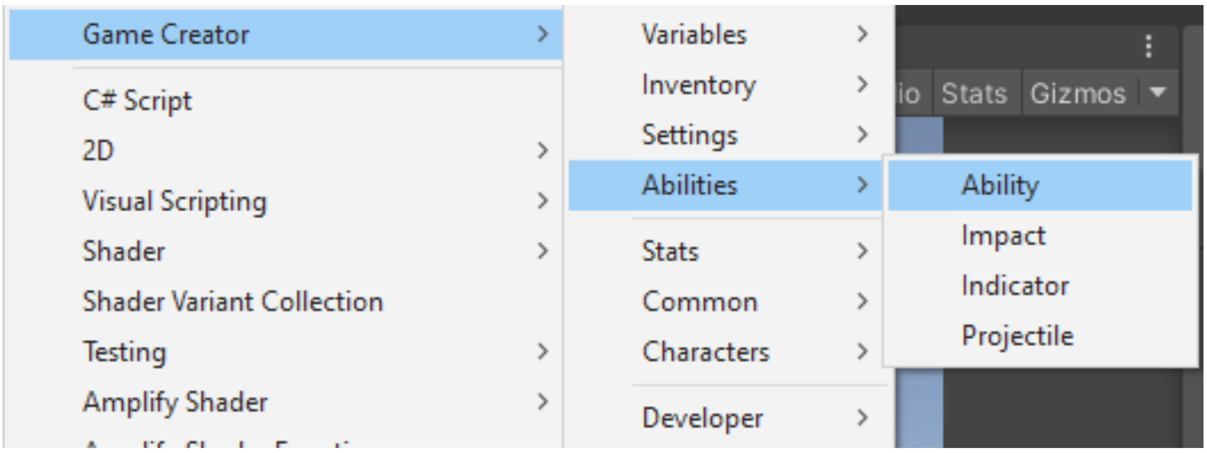
2.4 Ability Asset

2.4.1 Ability

The most fundamental part of Abilities is the Ability asset. It represents an action a character can perform in-game and comes packed with a collection of flexible and independent features that can be used to customize and control the way abilities work and feel in your game.

Creating an ability

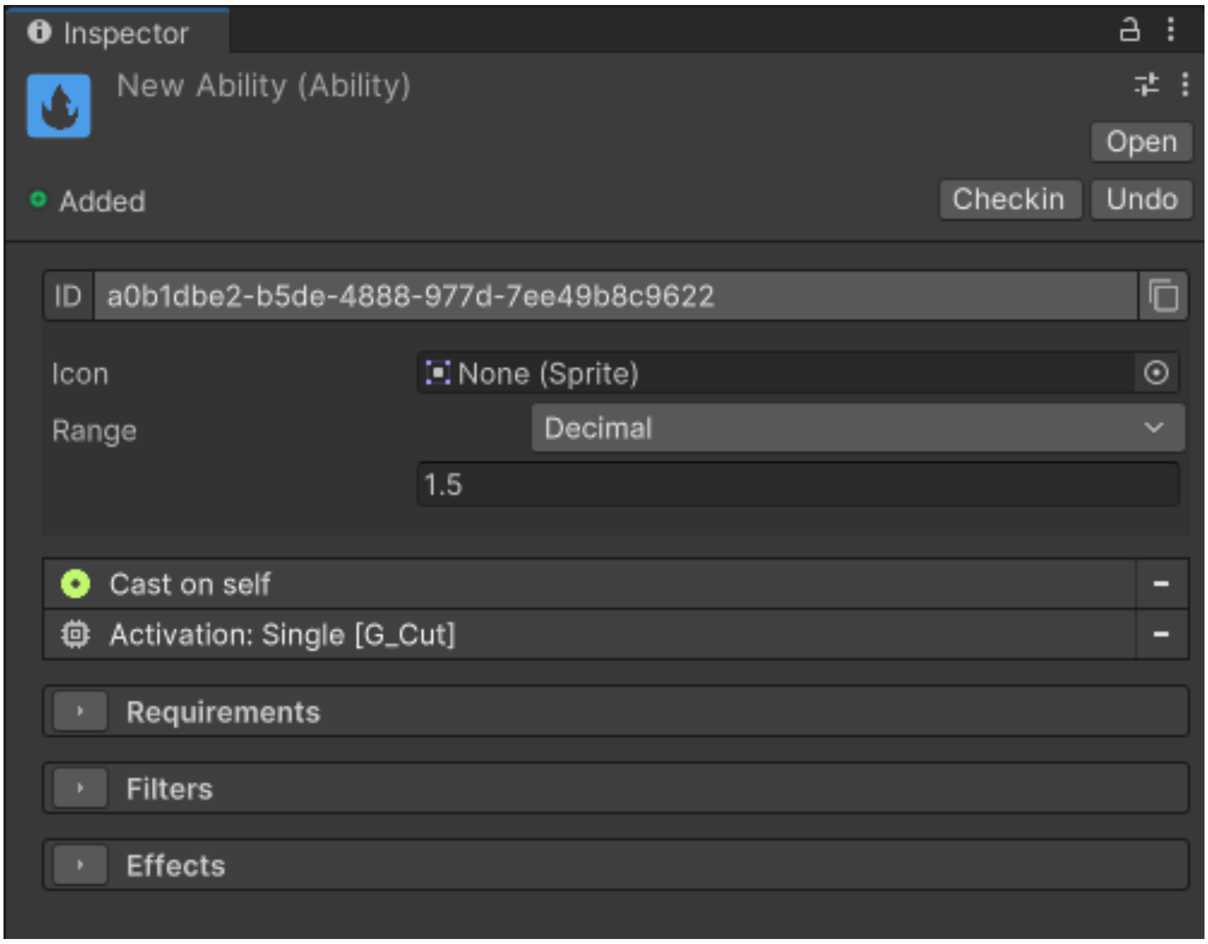
Abilities are scriptable objects and to create one, you'll need to right click on the Project Panel and navigate to Create → Game Creator → Abilities → Ability.



Creating an ability

Ability Inspector

An Ability 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 their abilities.



ID a0b1dbe2-b5de-4888-977d-7ee49b8c9622

Icon None (Sprite)

Range Decimal

1.5

Cast on self

Activation: Single [G_Cut]

Requirements

Filters

Effects

Ability inspector - folded

The **ID** value is a unique text that represents an ability. When creating a new asset, it will be completely unique. However, duplicating an existing ability 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.

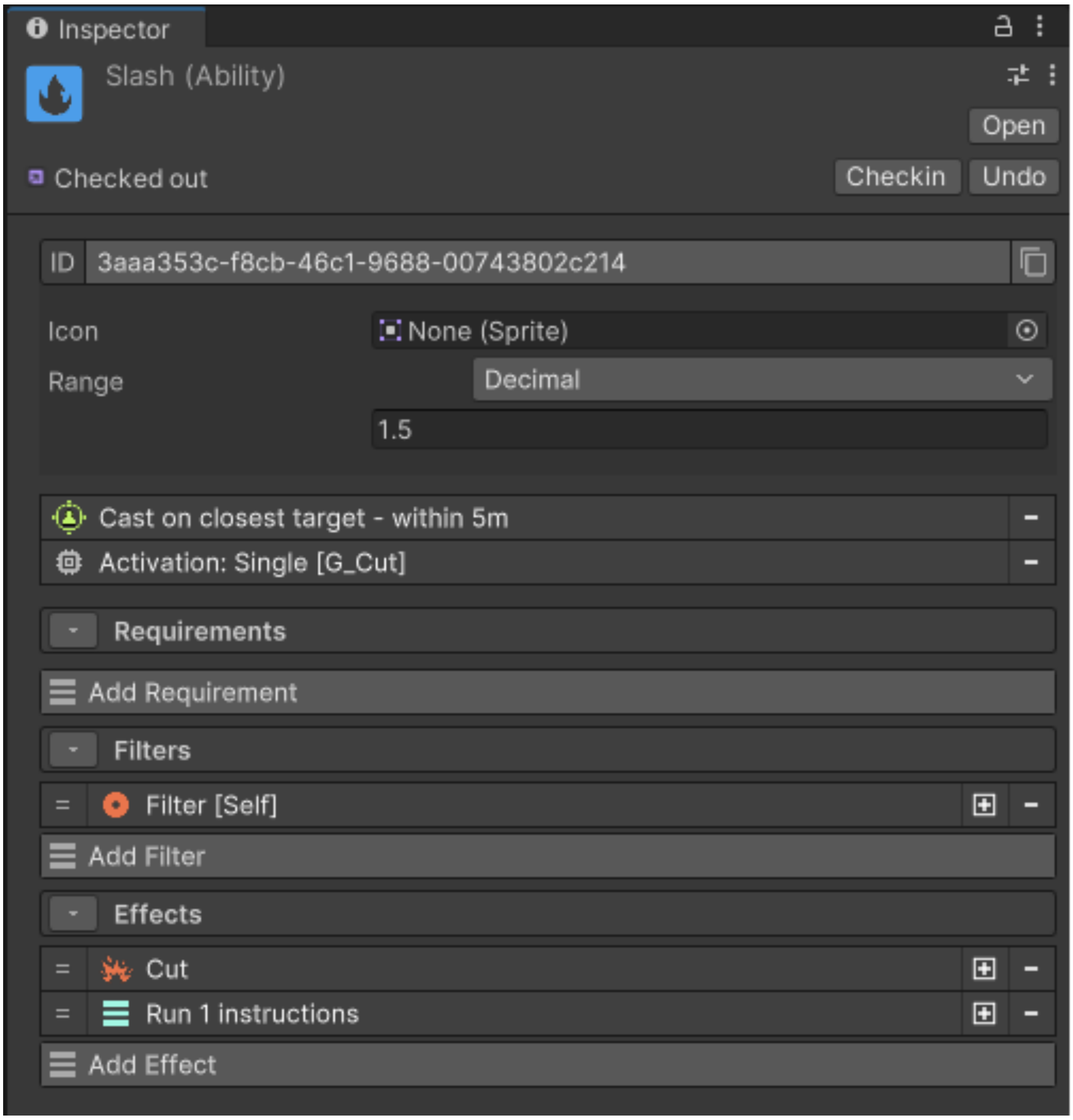
The **Icon field** is used to represent the ability in UI, if not set, a white square will be used instead.

The **Range field** represents the distance within which the ability can reach.

Main features

An Ability is defined by a scriptable object inside your project folder. It is organized into multiple collapse-able sections, each of which controls a very specific feature of this system.

- **Targeting** : An input system that allows you to change how the targets are acquired by the system. Automatically find the closest target, centers around the caster or at the mouse click are some of the different options.
- **Activation** : Controls how the ability activates its effect on the target and how input is processed. Some options are a single activation driven by an animation, or a channeling effect that stays active as long as it is on.
- **Requirements** : Define conditions which need to be met in order for an ability to be cast-able. Some requirements can also prevent an ability from activate and/or apply a cost to it, *e.g. cooldown or mana cost.*
- **Filters** : Define conditions that are used to filter the valid targets for your ability, *e.g. filtering the caster to prevent friendly fire.*
- **Effects** : Controls what the ability actually do, once it has activated and found valid targets, *e.g. damaging a target, spawning projectiles or explosion impacts and more.*



Inspector

Slash (Ability)

Open

Checked out

Checkin Undo

ID 3aaa353c-f8cb-46c1-9688-00743802c214

Icon: None (Sprite)

Range: Decimal

1.5

Cast on closest target - within 5m

Activation: Single [G_Cut]

Requirements

Add Requirement

Filters

Filter [Self]

Add Filter

Effects

Cut

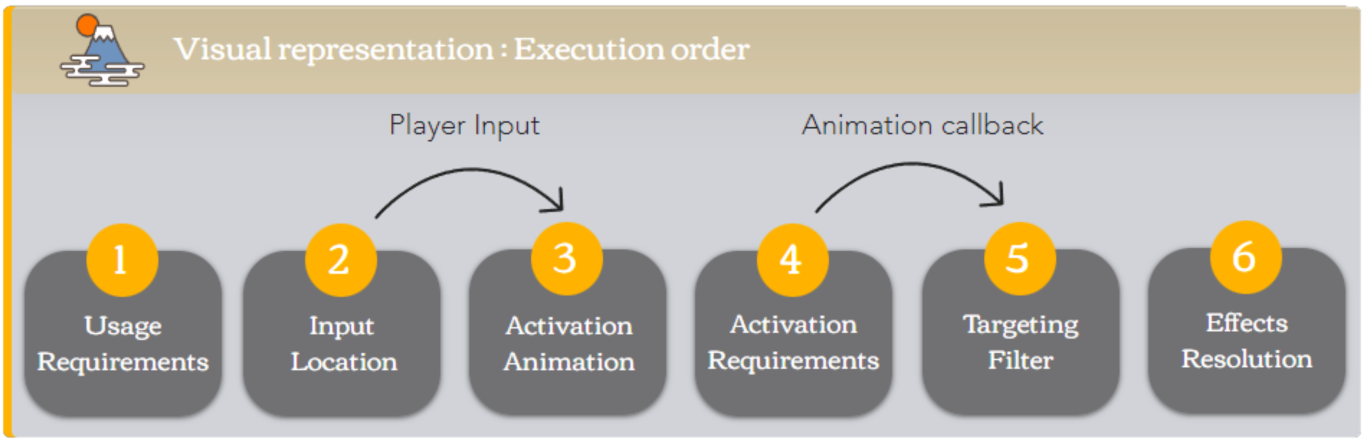
Run 1 instructions

Add Effect

Ability inspector - unfolded

Execution sequence

1. When the ability is triggered, the **Requirement system** determines if the ability can be executed.
2. The **Targeting system** triggers the input and determines the location the ability will activate from.
3. The **Requirement system** determines if the ability can be activated and activation costs are paid.
4. The **Activation system** decides how the ability will play out. Typically an animation is played using the Reactive Gesture system. Then, the animation sends a signal back to complete the activation.
5. The **Targeting system** works together with the Filter system to determine the valid target(s).
6. Finally, the **Effect system** applies the effects on the valid targets.



Context Menu

You can access a context menu by right clicking on most elements. Elements can be replaced or disabled and documentation can be accessed through it.

Systems

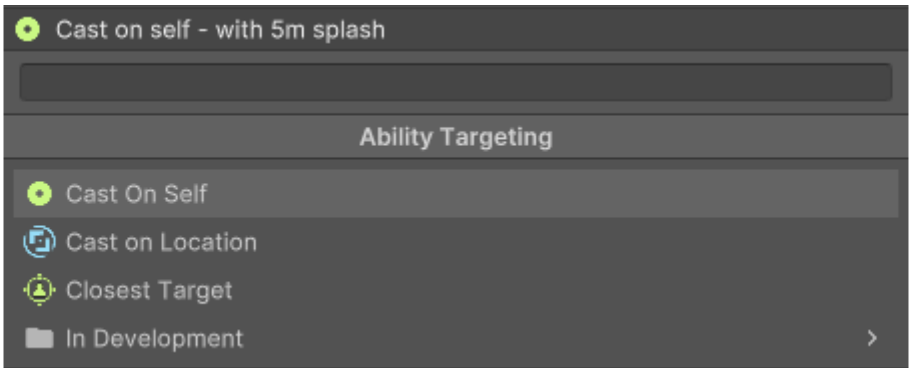
TARGETING SYSTEM

The targeting system is responsible for handling player *input* and *acquiring targets*.

A *target* can be any game object or location.

Depending on the system used, targeting happens in two phases. Once before activation to find the target location, and once after the activation to confirm the targets (c.f. #2 & #5 in the above graph).

Some settings have automatic input, *e.g. Cast on Self will automatically use the caster as a target*, while others will require the player input, *e.g. Cast on Location requires the player to click on the ground*.



Targeting systems

AI Input

A target can be submitted directly to the ability using an instruction to allow AI to use abilities as well. Refer to the Instructions section for more information.

ACTIVATION SYSTEM

The activation system is the centerpiece and dictates how the ability works.

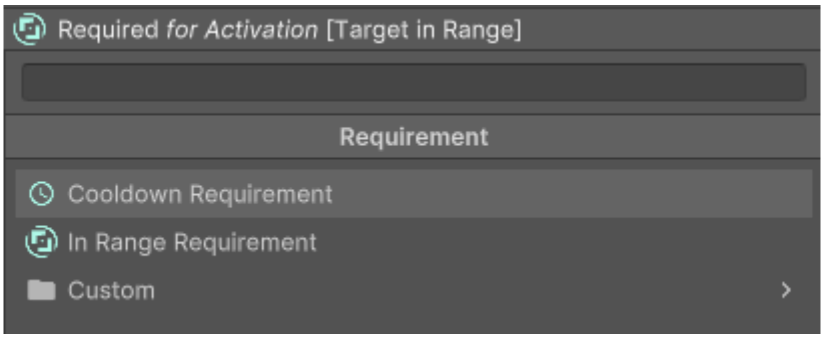
The most basic activation type is the Single activation, the ability starts with an animation, applies its effects once, and then completes its execution with the end of the animation.

Other activation types are Channeled and Charged activation.

At the moment only the Single activation type is implemented, but more will be added in future updates.

REQUIREMENT SYSTEM

The requirement system is responsible for controlling when the ability is use-able as well as managing its success conditions. Additionally it is also responsible for effects that are considered costs, *e.g. cooldowns and mana costs*.



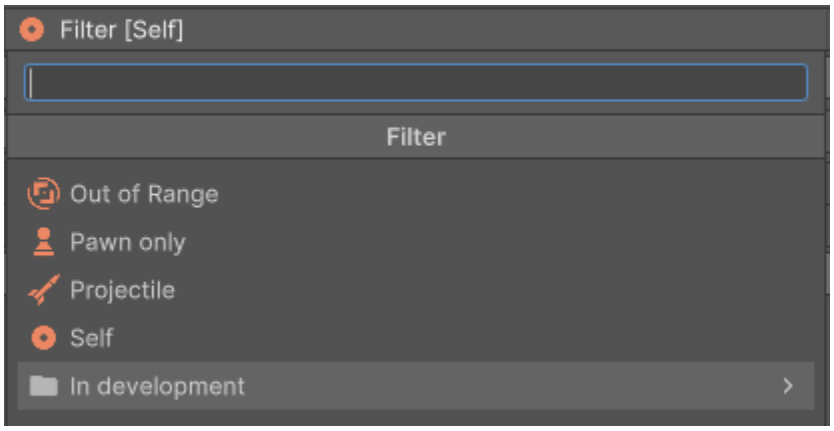
The requirements defined by this system are of 2 types :

- Requirement for use (#1): Condition that needs to be met in order to cast the ability. In case of failure the ability will simply not start.
- Requirement for Activation (#4): Condition that needs to be met in order for the effects to resolve. In case of failure, the animation will stop and the ability will fail.

Additionally, a requirement can sometimes have a *side-effect* (which is applied during the activation (#4)), e.g. entering cooldown, or paying mana cost.

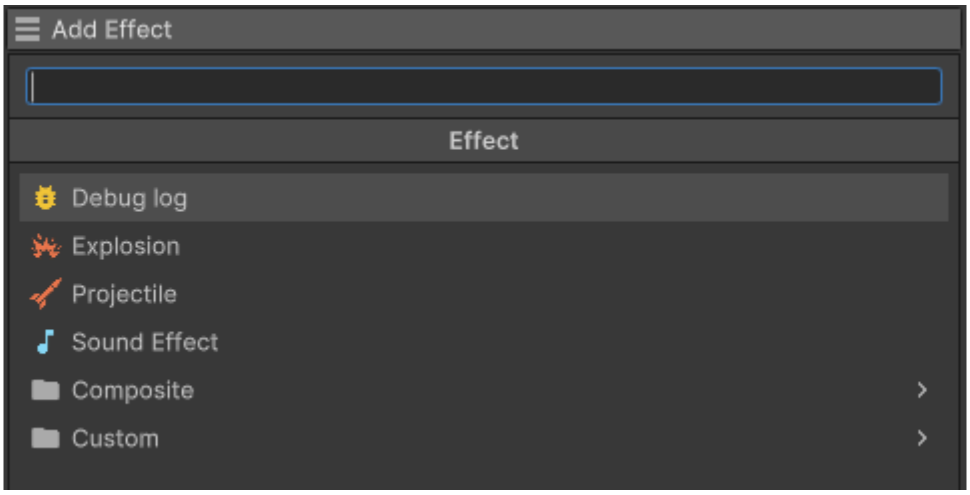
FILTER SYSTEM

The filter system is responsible for filtering targets according to filtering conditions. The most basic one is to filter out the caster to make sure the ability cannot backfire.



EFFECT SYSTEM

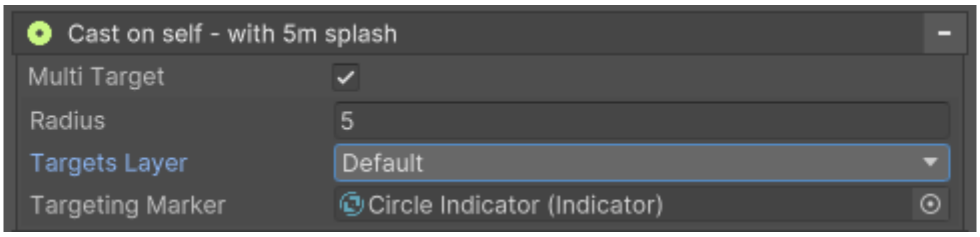
The effect system defines what the ability will eventually do to its targets. This system is further enhanced by the projectiles & impacts system.


Add Effect
Effect
Debug log
Explosion
Projectile
Sound Effect
Composite
Custom

2.4.2 Target System

Cast on Self

This targeting system will automatically use the caster as target for the ability at input time (#2). It will optionally target multiple targets at activation time (#5).



Cast on self - with 5m splash

Multi Target	✓
Radius	5
Targets Layer	Default
Targeting Marker	Circle Indicator (Indicator)

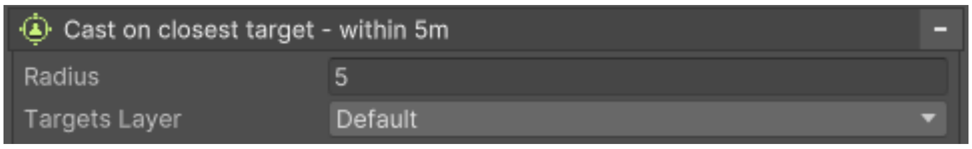
- **Multi Target** : Enable multi target.
- **Radius** : Maximum distance from the target
- **Target Layer** : Physics layer used to detect targets. Targets that are not part of the specified layers will be ignored.
- **Targeting Marker** : Optional indicator which spawns at the target location while the input is being processed. Using the UI input, the indicator will be visible as long as the corresponding button is pressed.

Physics detection

Any game object can represent a valid target for the ability, the only requirement is the presence of a collider on it. In case of problem, please double check that a collider exist on the target and that the ability is properly set to detect that layer

Cast on Closest Target

This targeting system will look for each potential target in the specified radius around the user location and select the closest one. This method acquires the target at input time (#2), meaning that once the animation starts, the target will always get hit. This can be avoided by using requirements.



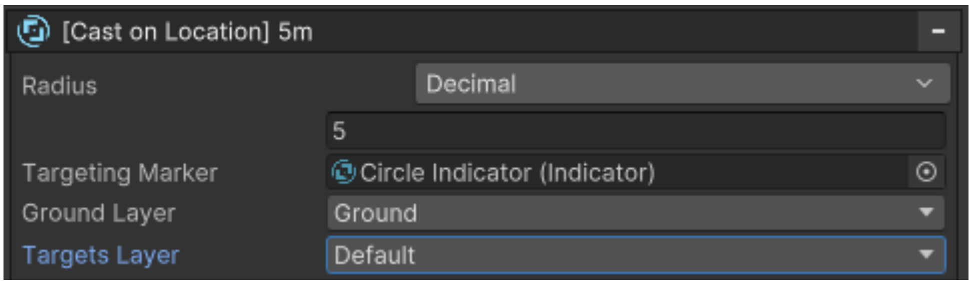
Cast on closest target - within 5m

Radius	5
Targets Layer	Default

- **Radius**: Maximum distance from the target location
- **Target Layer** : Physics layer used to detect targets. Targets that are not part of the specified layers will be ignored.

Cast on Location

This targeting system will automatically use the caster as target for the ability. It will optionally target multiple targets and display a marker on the ground while the input is kept pressed.



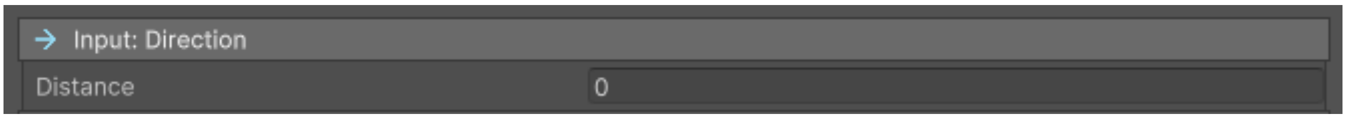
[Cast on Location] 5m

Radius	Decimal
	5
Targeting Marker	Circle Indicator (Indicator)
Ground Layer	Ground
Targets Layer	Default

- **Radius** : Area of effect - Distance from the center within which targets are selected.
- **Targeting Marker** : Optional indicator which spawns at the target location while the input is being processed. Using the UI input, the indicator will be visible as long as the corresponding button is pressed.
- **Ground Layer** : Physics layer used to draw the indicator on the floor.
- **Target Layer** : Physics layer used to detect targets. Targets that are not part of the specified layers will be ignored.

Direction

This targeting system will use the direction of the mouse cursor relative to the caster's position.



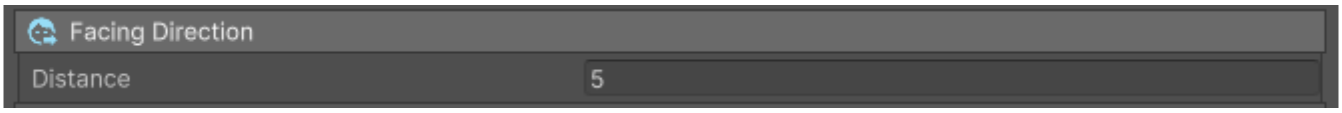
Input: Direction

Distance	0

- **Distance** : distance from the caster's position in the direction of the cursor. If left at 0, then the exact cursor position will be used instead.

Facing

This targeting system will use the direction of the caster.



Facing Direction

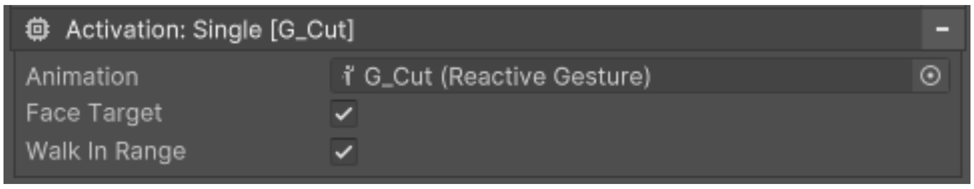
Distance	5

- **Distance** : distance from the caster's position in the forward direction.

2.4.3 Activation System

Single

The single activation starts an animation and waits for it to finish, ending the ability. Effects will be applied after receiving an activation notification from the animation.



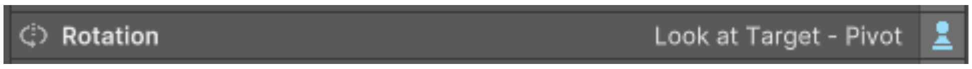
Activation: Single [G_Cut]

Animation	G_Cut (Reactive Gesture)
Face Target	✓
Walk In Range	✓

- **Animation** : Reactive Gesture containing the animation to play.
- **Face Target** : When enabled, turn the character to face the target. Rotation speed is defined by the Motion Unit.
- **Walk in Range** : When enabled, this will cause the character to first walk within range of the ability

Runtime Error

When **Face Target** is enabled during the execution of an ability, if the character is selected in the inspector, there will be an error. This is due to how Unity & Game Creator handle serialization and this will only happen in the editor. To avoid this, we recommend that you use our custom Rotation Unit **Look at Target - Pivot**.



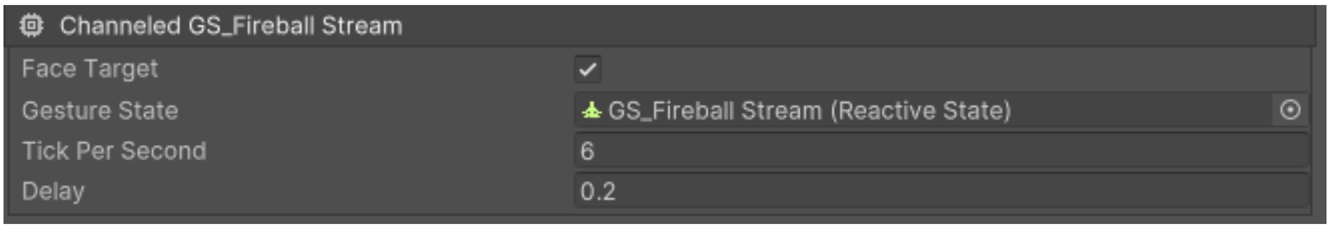
Rotation: Look at Target - Pivot

This unit is a hybrid between **Pivot** and **Look at Target**, it acts as a Pivot when no targets are available, and acts as Look at Target when no targets are available. The need for this custom Rotation Unit is only to avoid the runtime error in the editor.

Channeling

The channeling activation starts a looping animation and executes its effects a fixed number of times every second while the key is pressed down.

- **Face Target** : When enabled, turn the character to face the target. Rotation speed is defined by the Motion Unit.
- **Gesture State** : Reactive Gesture State containing the animation to play.



Channeled GS_Fireball Stream

Face Target	✓
Gesture State	GS_Fireball Stream (Reactive State)
Tick Per Second	6
Delay	0.2

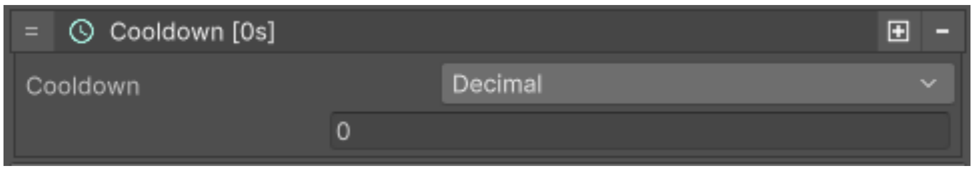
Charging

Coming soon

2.4.4 Requirement System

Cooldown

This requirement adds a cooldown to the ability. When a cast is requested (#1), the ability will fail if it is currently on cooldown. In case of success, the ability will enter its cooldown period at the activation time (#4).


Cooldown [0s]

Cooldown: Decimal

0

• **Cooldown**: cooldown duration in seconds.

Stats Synergy

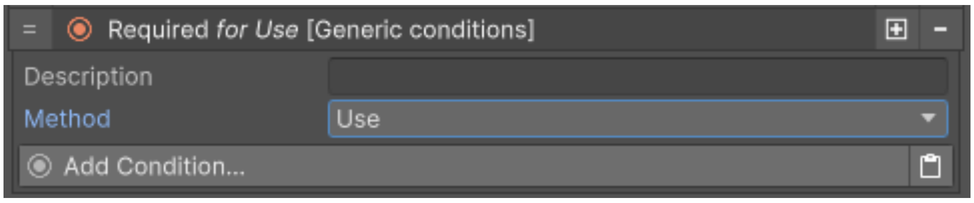
Owning the Stats module ? More power to you ! You can set the cooldown to use a formula and with a little set up, you can have abilities that have smaller cooldown as they level up !

In Range

This requirement will cancel the ability if the target is out of range at activation time (#4).

Custom

This requirement allows you to use any Game Creator conditions with the requirement system.


Required *for Use* [Generic conditions]

Description:

Method: Use

• **Description**: string field used to customize the title in the inspector. Useful to keep organized.

• **Method**: defines the type of requirement.

• *Use* : conditions to start the ability

• *Activation* : conditions to apply the effects.

2.4.5 Filter System

Filter Self

Removes the caster from the potential targets.

Filter Pawn Only

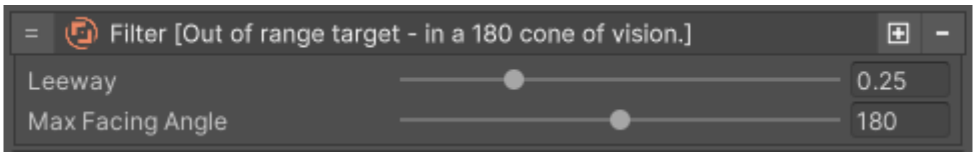
Targets can be locations, or they can be any game object. This will ensure that only Pawns are valid targets.

Filter Projectile

Filter used in projectiles for abilities that spawns multiple projectiles. This prevents a projectile from colliding with another.

Filter Out of Range

Filter targets that are not in range of the ability at activation time (#4).


Filter [Out of range target - in a 180 cone of vision.]

Leeway 0.25

Max Facing Angle 180

- **Leeway**: controls how strictly we are checking ability range. Leeway is added as a percentage to the actual ability range to determine if a target should be hit or not.
- **Max Facing Angle**: defines a cone in front of the caster, targets that are outside of it will be considered out of range. The angle is centered on the character facing direction, e.g. with a value of 180°, the half circle formed by the character's shoulder will be the valid area.

Melee ability avoidance

You can set up Melee Abilities using the Closest Target Targeting System.

With this setup, once the tharget is acquired and the animation initiated, the target will always be hit. However, between the start of the animation, and the resolution of the effects, there might be a time lapse during which the target may move away.

This filter will remove any target that moved out of range.

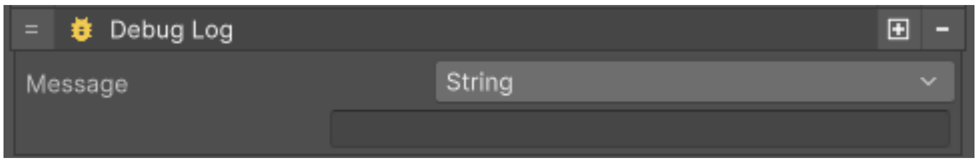
2.4.6 Effect System

Targets & effects

An important thing to understand is that each effect applies once per target, *e.g. if you set up the Targeting System to capture multiple targets and set an Effect to spawn one projectile, you will effectively spawn one projectile per target*.

Debug log

Prints a log message in the console. The message will be prefixed with the target name or location. Useful to understand what targets have been captured by the system.

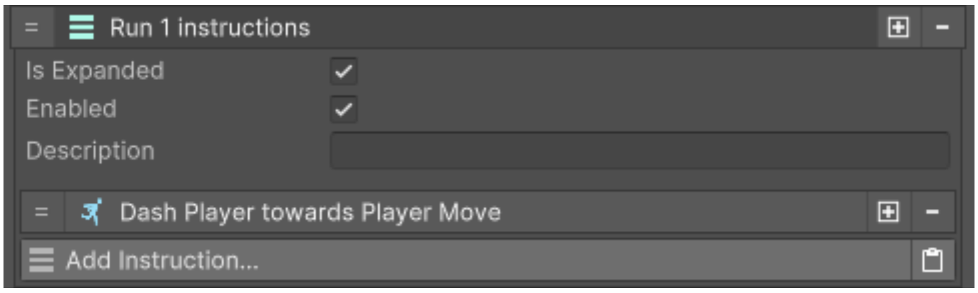


- **Message** : the message to print.

Instructions

Execute a generic Game Creator instruction list. Note that the arguments for the instructions are as follow :

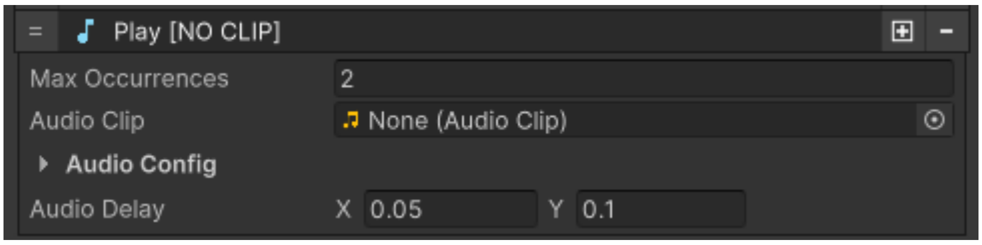
- **Self** : the caster
- **Target** : the current ability target



- **Description** : replace the title in the inspector. Useful for organization.

Sound effect

Play a sound effect.


Play [NO CLIP]
Max Occurrences: 2
Audio Clip: None (Audio Clip)
Audio Config
Audio Delay: X 0.05 Y 0.1

- **Max Occurences** : limits the number of time this sound effect will be trigger by this ability
- **Audio Clip** : the clip being played
- **Audio Config** : see Game Creator audio configuration for more info.

Projectile

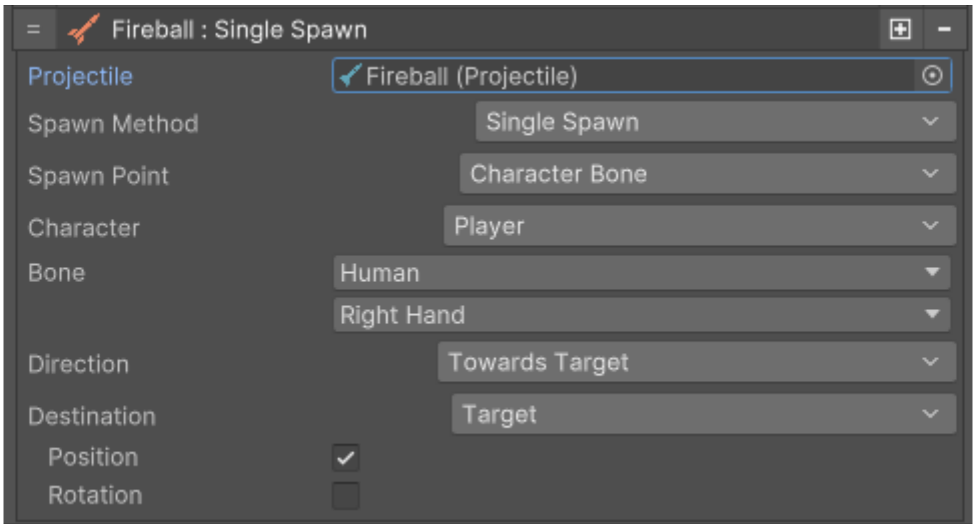
Spawns one or more projectile(s). See the projectile documentation for more details.

- **Projectile** : the projectile to be spawned.
- **Spawn Method** : the way projectiles are spawned.
- **Spawn Point** : the origin of the projectile

SPAWN METHODS

Single Spawn

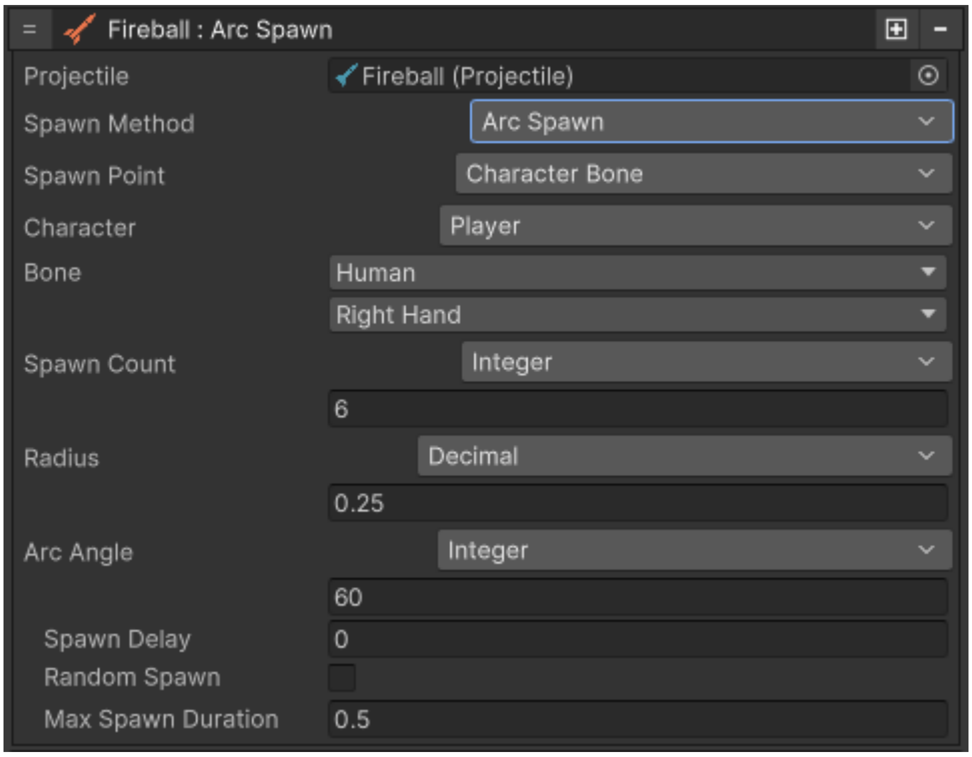
Spawn a single projectile.


Fireball : Single Spawn
Projectile: Fireball (Projectile)
Spawn Method: Single Spawn
Spawn Point: Character Bone
Character: Player
Bone: Human
Right Hand
Direction: Towards Target
Destination: Target
Position: ✓
Rotation:

- **Direction** : direction the projectile is spawned in.
- **Destination** : planned destination of the projectile.

Arc Spawn

Spawn projectiles distributed around a cone shape.

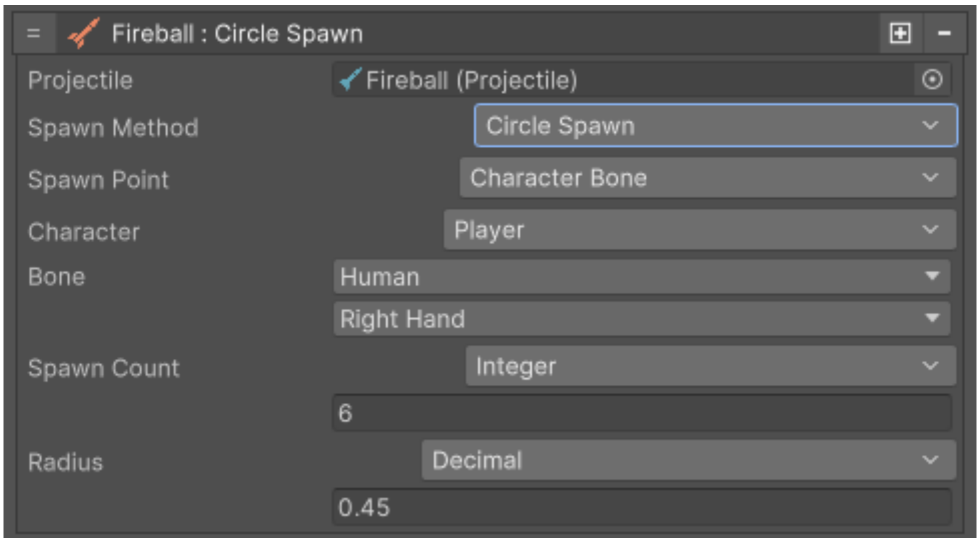

Fireball : Arc Spawn

Projectile	Fireball (Projectile)
Spawn Method	Arc Spawn
Spawn Point	Character Bone
Character	Player
Bone	Human
	Right Hand
Spawn Count	Integer
	6
Radius	Decimal
	0.25
Arc Angle	Integer
	60
Spawn Delay	0
Random Spawn	☐
Max Spawn Duration	0.5

- **Spawn Count** : number of projectiles to be spawned.
- **Radius** : radius of the cone shape, controls how far the projectiles are spawned from the spawn point.
- **Arc Angle** : angle of the cone shape. Controls the width of the spawn.
- **Spawn Delay** : spawn rate in seconds. If 0, all the projectiles will be spawned at a time.
- **Max spawn duration** : will reduce the spawn delay to allow all the projectiles to be spawned within the duration.
- **Random Spawn** : Will randomize the position of each projectile within the cone.

Circle Spawn

Spawn projectiles distributed around a circle.

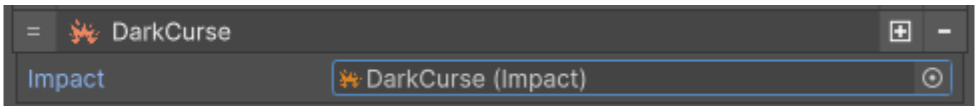

Fireball : Circle Spawn

Projectile	Fireball (Projectile)
Spawn Method	Circle Spawn
Spawn Point	Character Bone
Character	Player
Bone	Human
	Right Hand
Spawn Count	Integer
	6
Radius	Decimal
	0.45

- **Spawn Count** : number of projectiles to be spawned.
- **Radius** : radius of the cone shape, controls how far the projectiles are spawned from the spawn point.

Impact

Spawns an impact at the target location. See the impact documentation for more details.



DarkCurse
Impact: DarkCurse (Impact)

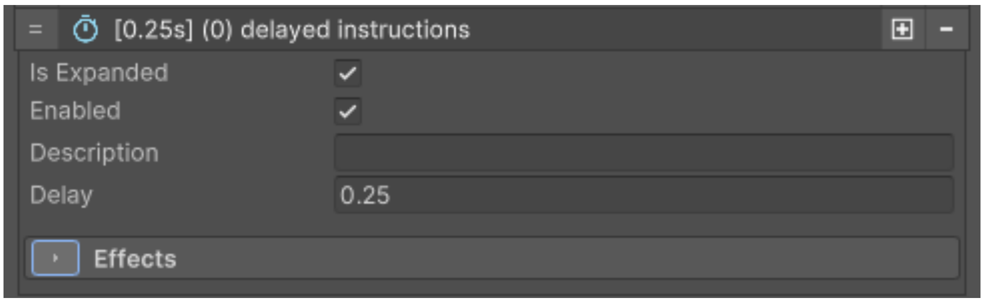
• **Impact** : the impact to be spawned.

Composite effects

Composite effects are special effect containers that add additional functionality to the previous effects. Composite effects can be combined with one another to create an arbitrarily complex setup.

DELAYED EFFECTS

Add a delay to the effects.



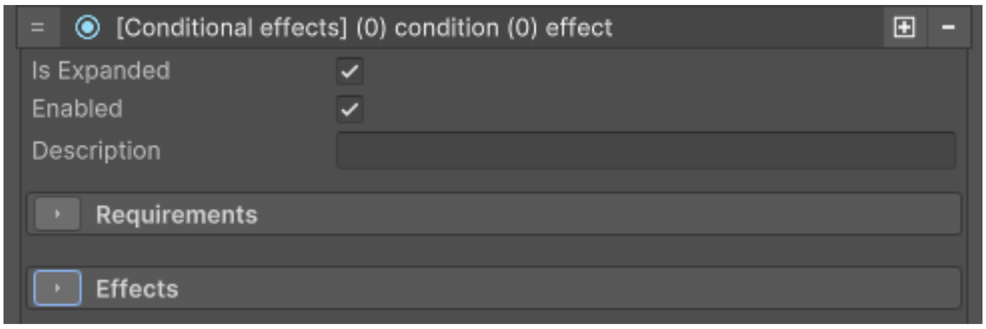
[0.25s] (0) delayed instructions
Is Expanded: ✓
Enabled: ✓
Description:
Delay: 0.25
Effects

• **Description**: string field used to customize the title in the inspector. Useful to keep organized.

• **Delay**: delay after which the effects are applied.

CONDITIONAL EFFECTS

Add requirements to the effects to be applied. If the requirements are not met, the effect will simply be skipped.



[Conditional effects] (0) condition (0) effect
Is Expanded: ✓
Enabled: ✓
Description:
Requirements
Effects

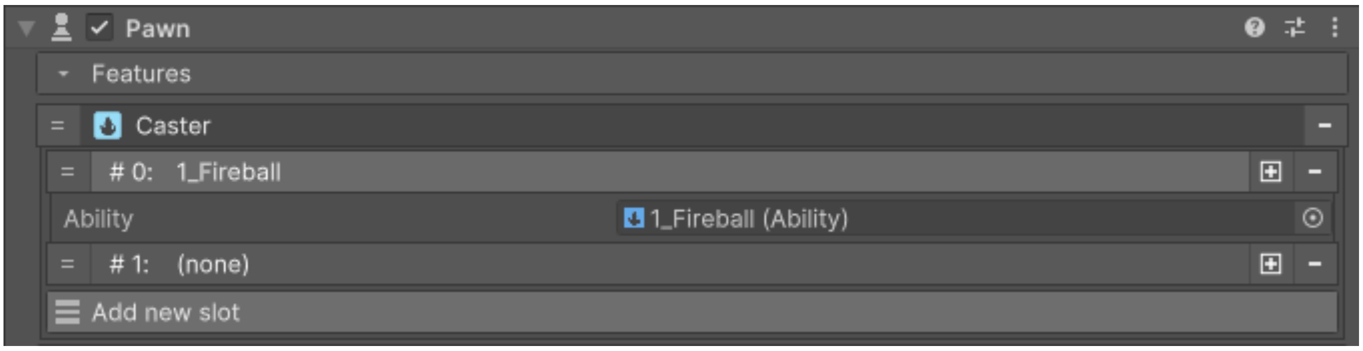
Burn with Fire !

You can easily use these conditional effects to add additional damage on a fireball if the enemy is affected by the status effect "Burn". Using the module Stats will help a long way to create this kind of effect.

2.5 Pawn feautures

2.5.1 Caster

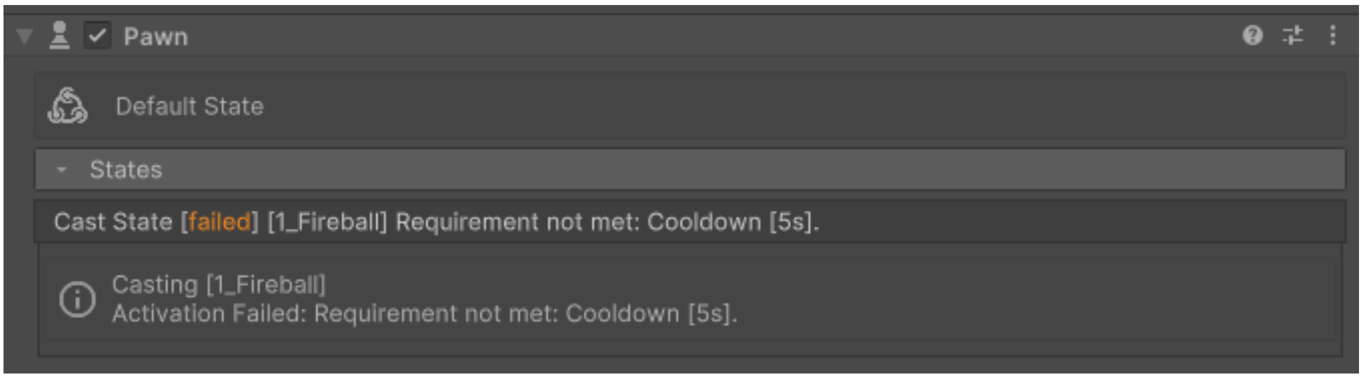
Feature that allows the pawn to cast spells. The feature contains a list of ability slots that can be bound to an input action of a controller. Simply adding the feature to a pawn will allow it to cast spells. Slots are used to bind each spell to an input key.



Cast State

When casting a spell, the caster feature will try to set the pawn state of "Cast State". This is an internal concept which the user does not need to be too concerned about. The interesting part though is that at runtime, useful information isdisplayed in the inspector which can give clues to why the system does not react as expected.

For example, any number of reasons could stop a spell from activating as expected, the cast state will reveal exactly what prevented the cast.

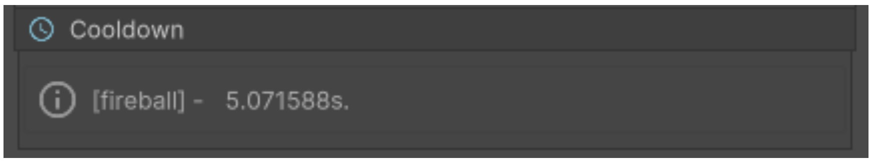


Cast State [failed] [1_Fireball] Requirement not met: Cooldown [5s].

Casting [1_Fireball]
Activation Failed: Requirement not met: Cooldown [5s].

2.5.2 Cooldown

Feature that allows the pawn to have cooldowns. Required to cast spells with cooldowns.



[fireball] - 5.071588s.

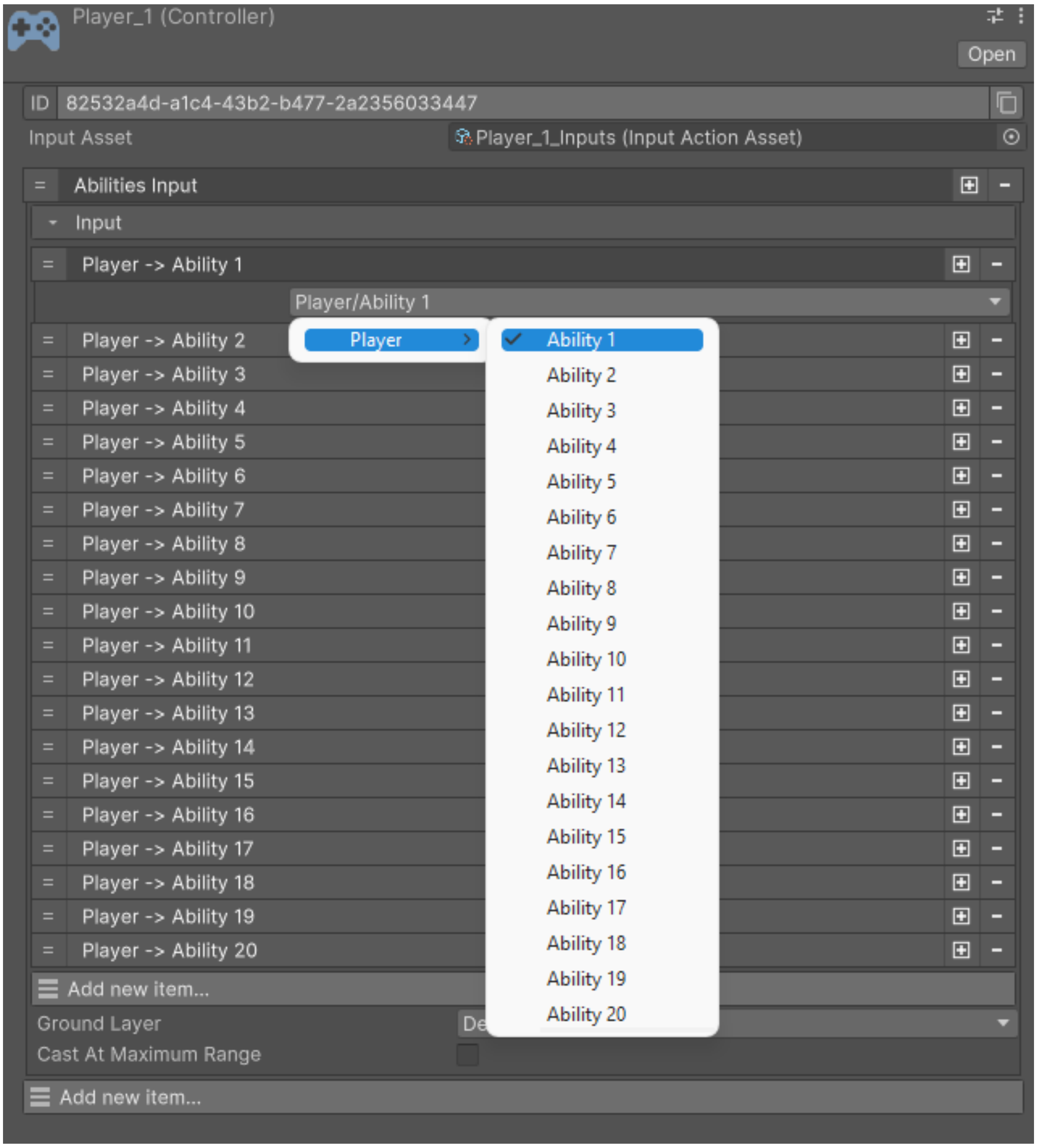
Runtime info

You can consult which abilities are on cooldown and for how long in the inspector at runtime.

2.6 Ability Input Module

The ability input module allows the user to define key binding to cast abilities that have been slotted in the caster feature.

- **Input:** List of input mappings. Adding a new entry will bind a slotted ability from the caster feature to the specified input action.
- **Ground layer:** Layer used for raycasting on the ground.
- **Cast at maximum range** When casting an ability to an out of range location, the controller will instead pick a new nearby location at max range.


Player_1 (Controller)
Open
ID 82532a4d-a1c4-43b2-b477-2a2356033447
Input Asset: Player_1_Inputs (Input Action Asset)
Abilities Input
Input
Player -> Ability 1
Player/Ability 1
Player -> Ability 2
Player -> Ability 3
Player -> Ability 4
Player -> Ability 5
Player -> Ability 6
Player -> Ability 7
Player -> Ability 8
Player -> Ability 9
Player -> Ability 10
Player -> Ability 11
Player -> Ability 12
Player -> Ability 13
Player -> Ability 14
Player -> Ability 15
Player -> Ability 16
Player -> Ability 17
Player -> Ability 18
Player -> Ability 19
Player -> Ability 20
Add new item...
Ground Layer: De
Cast At Maximum Range
Add new item...

Controller example

2.7 Projectiles & Impacts

2.7.1 Projectile & Impacts

Projectiles

A projectile is a moving game object that works as a "targeting system" that is used to deliver ability effects to the poor soul who ends up colliding with it.

A projectile defines the behavior of a single missile that can be reused in different abilities and be spawned in various ways using the projectile ability effect.

CREATING A PROJECTILE

Projectiles are scriptable objects and to create one, you'll need to right click on the Project Panel and navigate to Create → Game Creator → Abilities → Projectile.

projectile

Controller example

- **Prefab**: the prefab representing the projectile. The projectile is a physics object and is moved through physics, therefore it requires a trigger collider and a rigidbody to work properly. The system will automatically set it up for you, but you can customize the projectile further by adding them yourself.
- **Acceleration**: an animation curve that can be used to control the acceleration of the projectile over the distance between its spawn point and its target.
- **Speed**: the speed at which the projectile travels
- **Projectile range**: range of the projectile, used to determine the maximum travel distance.
- **Pierce Target**: determine if the projectile will be destroyed on contact or if it can hit more targets after the first one.
- **Always Explode**: if enabled, the projectile will always be destroyed, and its effect resolved after reaching its defined *range* (see above)
- **Constant Range**: the projectile will normally adjust its range to the distance between its spawn point and its target. If enabled, the range will always be the one set above (useful for enabling various effects).
- **Filters**: same as abilities, see documentation.
- **Effects**: same as abilities, see documentation.

Chain effects

Although it is possible to add an effect on the projectile to spawn itself, it is not recommended as it can get out of hand and spawn an exponentially increasing number of projectiles. A better way is to spawn a different projectile or an impact instead, this way you always control what a projectile can spawn. Always keep organized and avoid circular dependencies that might crash Unity.

DEVIATION SETTINGS

The settings under this fold out let you customize the trajectory of your projectile using animation curves. The animation curve can be set from 0 to 1 and is mapped to the range of the projectile.

Constraints

Always keep the time values between 0 and 1. Any values before 0 and after 1 will be ignored. It is also recommended to keep the Y value between 0 and 1. You can play with the intensity using the Deviation Multiplier field.

projectile-deviation

Controller example

- **Loop Deviation**: if enabled, after traveling a distance equal to its range, repeats the deviation curve in a loop over its lifetime.
- **Frequency**: compressed the curve so that it repeats more often over the projectile range. Requires *Loop Deviation*.
- **Randomize Deviation**: pick a random start time on the curve each time the projectile is spawned. Requires *Loop Deviation*.
- **Horizontal Deviation**: curve controlling the horizontal deviation.
- **Vertical Deviation**: curve controlling the vertical deviation.
- **Backward Deviation**: curve controlling the backward deviation.
- **Deviation Multiplier**: multiplies each deviation curve by the corresponding multiplier value.

Be creative

With the deviation settings you can easily create projectiles that spiral towards their target, or lob unto them.

Adaptive trajectory

If you keep the *constant range* setting off, the deviation will always be mapped from the origin to the target. So if you set the projectile to move along a bell shaped curve, you can be sure that your projectile will always hit the target as expected.

HOMING SETTINGS

The settings under this fold out let you modify your projectile to home to its target. The projectile will gradually adjust its direction until it hits its target. The *homing precision* allows you to control how aggressive it adjusts its direction.

projectile-homing

Controller example

Impact

An Impact represents another tool in your toolbelt to acquire targets and apply effects.

An Impact will gather targets all around it, up to a certain distance and apply the impact effect on them.

CREATING AN IMPACT

Impacts are scriptable objects and to create one, you'll need to right click on the Project Panel and navigate to Create → Game Creator → Abilities → Impact.

impact

Controller example

- **Prefab**: prefab that is spawned at the impact position. Usually meant to be VFX only. We *highly* recommend using the pooling system on that one.
- **Radius**: distance within which targets are acquired.
- **Delay**: delay after which targets are acquired and effects applied.
- **Filters**: same as abilities, see documentation.
- **Effects**: same as abilities, see documentation.

Impact delay

Sometimes, VFX have a windup/buildup to their visuals, you can use the delay to apply the impact effect at the exact time you impact VFX detonates for maximum impact (pun intended).

2.8 Quality of life

2.8.1 Indicators

Indicators are simply a scriptable object wrapper meant for visual indicators. The only field is for the game object prefab, optionally you can set it up for pooling (which is recommended with a value of 1).

Its purpose is to make it easier to pick when creating abilities.

Creating an indicator

Indicators are scriptable objects and to create one, you'll need to right click on the Project Panel and navigate to Create → Game Creator → Abilities → Indicator.

2.9 Tutorials

2.9.1 Coming soon

2.10 Releases

2.10.1 1.1.0

Overall polish of all inspectors

- New targeting system (Directional, Facing direction)
- New Activation mode : channeled
- New input system

2.10.2 1.0.0

Initial release