Minotaur Narrative Tool

Overview

Minotaur is a tool designed to make narrative design in Unity more accessible to designers. Core features of Minotaur include visually-scripted branching dialogue, easily exposed variables and events, a bark system built out of scriptable objects, and a visually-scripted quest system.

Table of Contents

Video Tutorials

1. Branching Dialogue tutorial
2. Barks tutorial

System Concepts

Below is a list of general concepts applied across the entire Minotaur system.

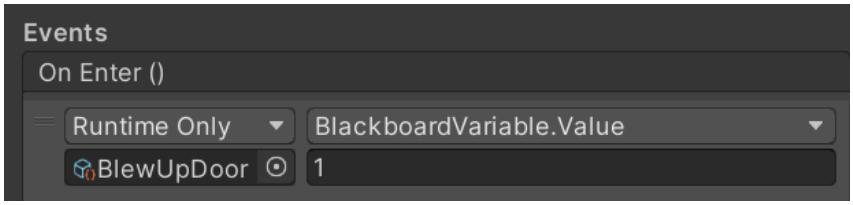
- *Conditional:* A logical unit requiring a blackboard variable, an operation, and a value to compare to.

- *Blackboard Variable:* A scriptable object representing a tracked value.
    - Blackboard variables are only stored as integers that default to 0. For boolean values, it's helpful to use the following shorthand:
        - 0 = false
        - 1 = true
- *Character:* A special type of scriptable object representing a character.
    - Note: The *icon* field of the character SO is only used in branching dialogue by default as a character portrait.
- *Scriptable Object Event:* A Unity pattern that Minotaur is built on. See below for more information on how it works.

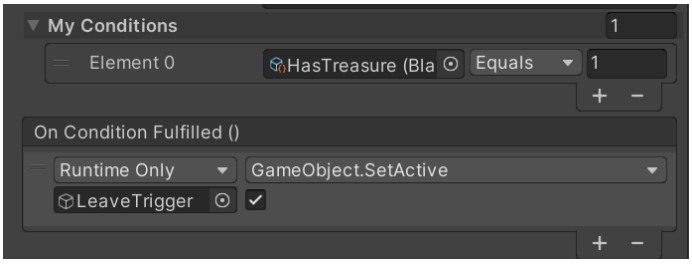
Blackboard Overview

The blackboard in Minotaur is composed of two primary parts: variables and conditionals.

Variables are scriptable objects used to contain a reference to a value to be used across the system - this can be things from the player's health to whether the player has completed a given quest or not. Variables are typically read automatically and they can be set through Unity Events.



Conditionals are logical statements that can be accessed from the inspector. Conditionals are used across every system in Minotaur and can be used outside of it as well.



[Back to Table of Contents]

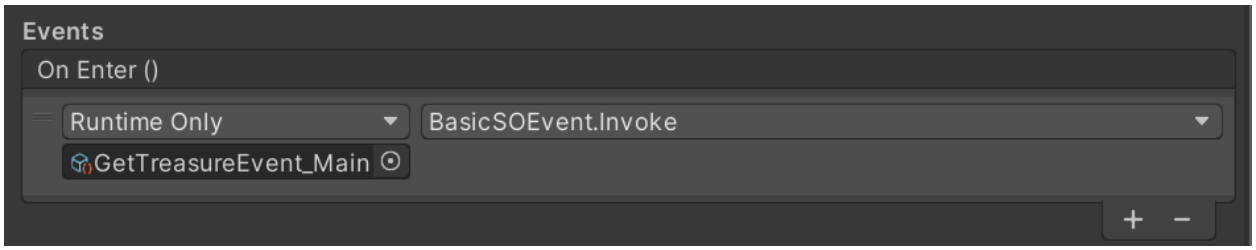
Scriptable Object Events Overview
This section assumes no prior knowledge of event systems.

Scriptable object events (SO events) are a Unity design pattern useful for putting more control in the hands of designers. Events, by their simplest definition, are things that happen. When an event is triggered, it often will trigger one to many different other things to occur - those other things that happen when an event is triggered are referred to as “listeners”.

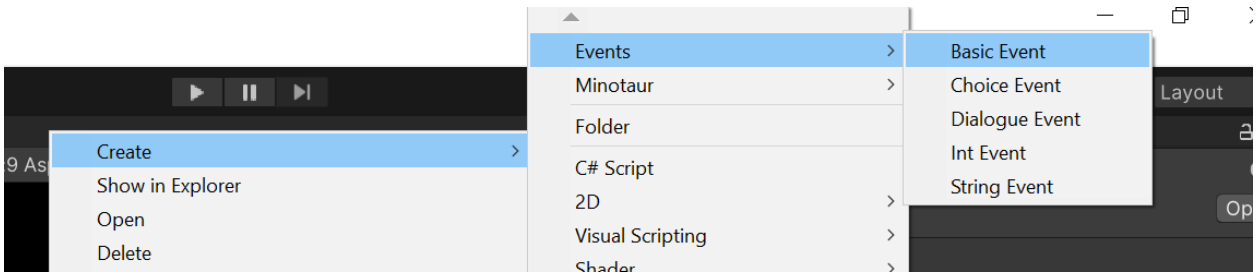
For example, say you have a character in a first-person shooter. When they’re shot, an event is triggered (OnBeenShot). OnBeenShot can have a lot of things listening to it; for example, it can reduce the player’s health, provide visual feedback in the form of screen jelly, and it can make the character say a bark.

Normally, events are handled through code, but the SO event pattern allows them to be drag-and-dropped in the Unity inspector, allowing them to be accessed and used by designers without fear of causing errors.

Triggering SO Events
SO events can be triggered through Unity events using the “Invoke” function on them, like so:



To do this, first create a BasicSOEvent (there are others available but they’re more advanced and less generally applicable) by right-clicking somewhere in the project and going to Create/Events/Basic Event.



Once you’ve created the event, you can drag it into the inspector field and trigger the invoke function like you would calling a function on any other script. Now, whenever that

Back to Table of Contents

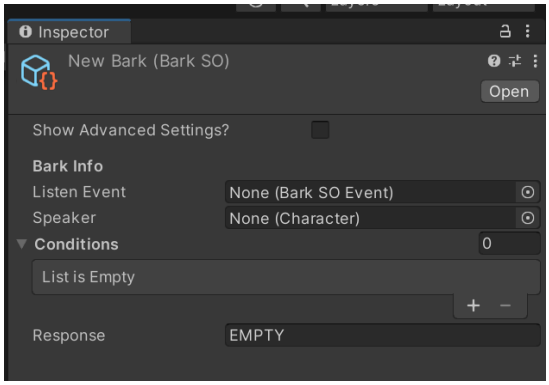
Unity Event triggers, it will also trigger your SO event, allowing you to reference that event in places where you wouldn't normally, for example…

Listening for SO Events

Minotaur uses SO events most prominently in the Barks and Quest systems, however, they're important to the dialogue system as well and can be used to great effect to trigger things over the course of a conversation.

For this example, we're going to look at the barks system.

When you create a new bark, it will look like the image on the right. *(The rest of this system is explained in the Barks section.)*



What we're most interested in right now is the "Listen Event" field.

The Listen Event for a bark is of type Bark SO Event, which can be created by right-clicking and going to Create/Minotaur/Barks/New Bark Event. They're used just like Basic SO Events, they're just a different type to reduce inspector clutter.

When you assign a Bark SO Event to a bark's Listen Event, that bark starts listening for it, and - assuming all of its conditions are fulfilled and the character is in range - the bark will be played when the event is triggered.

See the Barks section for more information on how to set up barks.

Back to Table of Contents

Branching Dialogue

Overview

Branching dialogue in Minotaur is built using script graphs in Unity Visual Scripting (UVS). It's worth noting that while Minotaur includes a lot of new functionality exclusive to branching dialogue systems, branching dialogue graphs still have the same functionality that UVS may otherwise have.

Key Concepts

Unity Visual Scripting (UVS) Concepts

- *Flow:* The execution order that's visually represented by arrows that connect every unit in a graph from start to finish.

An example of branching Flow using the dialogue system.

- *Unit:* A block that does something in the state/script graph view.
- *State Graph:* An overview graph for connecting big blocks of logic together.
- *Script Graph:* A graph for executing specific logic.
- *Subgraph:* A script graph prefab that has a specific function.

Branching Dialogue Concepts

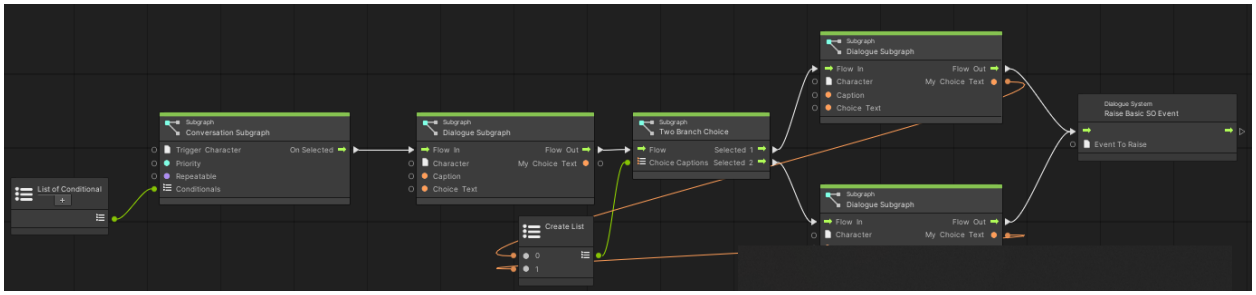
An explanation of how to use each concept is found in the following section.

- *Conversation:* A series of dialogue nodes preceded by a conversation subgraph. When you interact with an NPC, this is what you interface with.
    - Conversations need a "List of Conditionals" UVS unit, even if the list is empty.
- *Dialogue:* A single line of dialogue triggered by flow or a choice branch. This is what a character will actually say.
- *Choice Branch:* A special unit for offering choices to the player.
    - Choice branches need a list of all the dialogue branches' choice text.

Back to Table of Contents

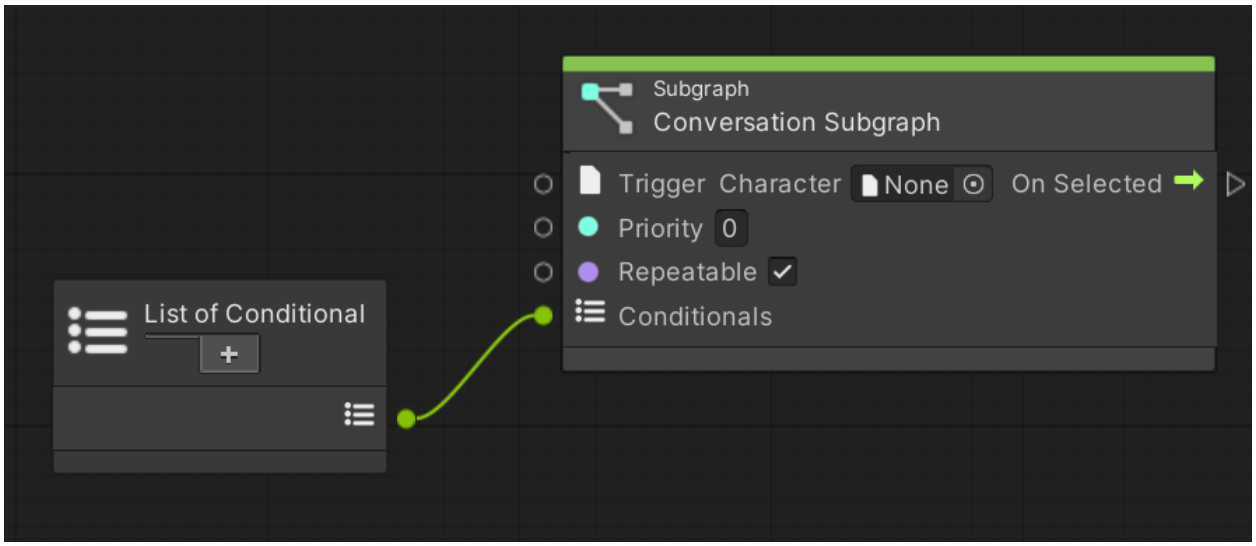
Quick Start Guide (Conversation Graphs)
A complete conversation includes one Conversation Subgraph, any number of Dialogue Subgraphs, optional Choice Branches, and eventually every branch must terminate in a Dialogue End Event.

A complete conversation looks basically like this:



We'll walk through each part briefly in the following sections.

Conversation Subgraph

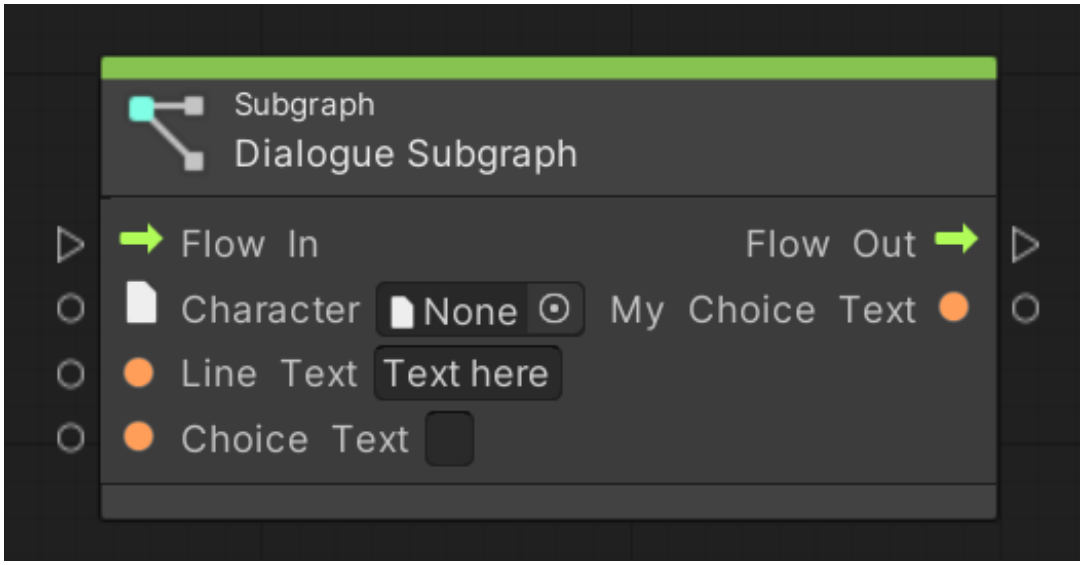


Conversations must start with a Conversation Subgraph (right) and that subgraph must have a "List of Conditional" unit (left) connected to the "Conditionals" input, even if it's empty.

The conversation subgraph only needs a character to be assigned to its Trigger Character field - this is the NPC that the conversation will be triggered from.

Back to Table of Contents

Dialogue Subgraphs



Dialogue subgraphs are lines of dialogue. It will be triggered when the flow enters the graph and, once progressed beyond, it will lead to whatever its flow outputs.

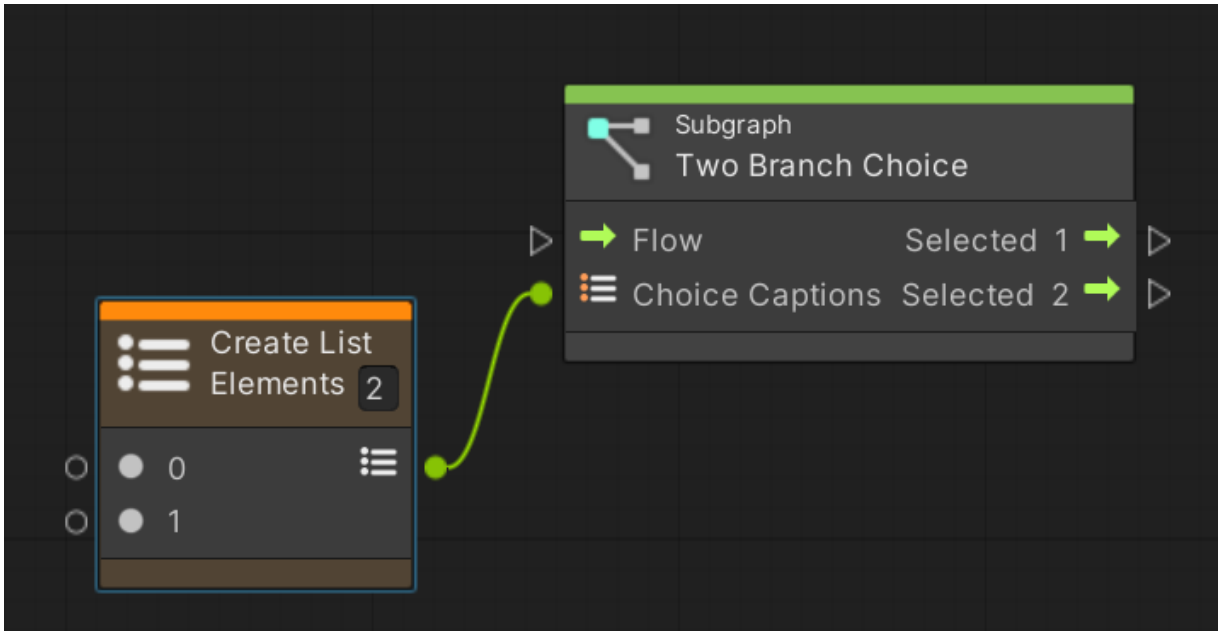
The Character field represents the speaking character. Out of the box, this will have that character's name across the top of the dialogue box and their icon to the left as a character portrait.

Line text is what the character says.

Choice text is only used in choices. See the Choice Branch section on the next page for more information.

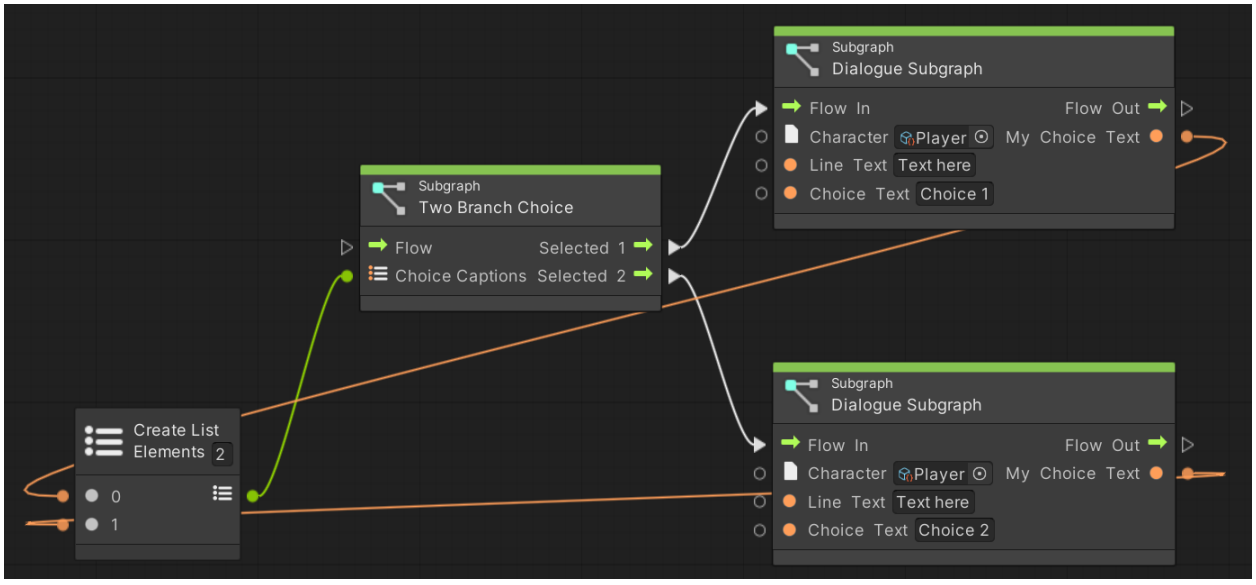
Back to Table of Contents

Choice Branch



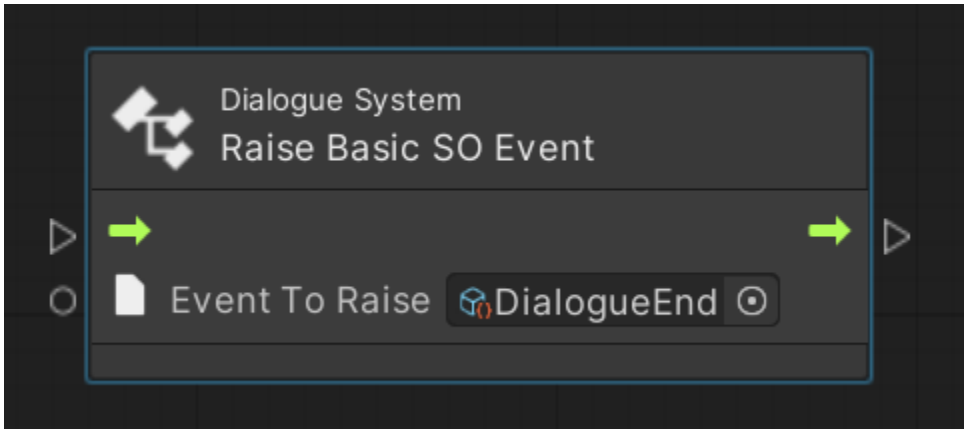
Branch subgraphs offer choices to the player. There are two, three, and four branch choice subgraphs and each one will need a list connected to its input of every branch's choice text. *(Note; Make sure that the "Selected 1" choice text feeds into option 0, "Selected 2" into option 1, and so on - order matters!)*

A complete choice branch will look like this.



[Back to Table of Contents]

Dialogue End Event



Whenever a dialogue terminates the conversation, it will need a “Raise Basic SO Event” unit with a “DialogueEnd” event plugged into it. Without this, a conversation will not close!

Additional logic can (and is recommended to be) triggered after a conversation terminates. Triggering quest progression events, environmental interactions, or barks should all happen *after* a conversation terminates.

Back to Table of Contents

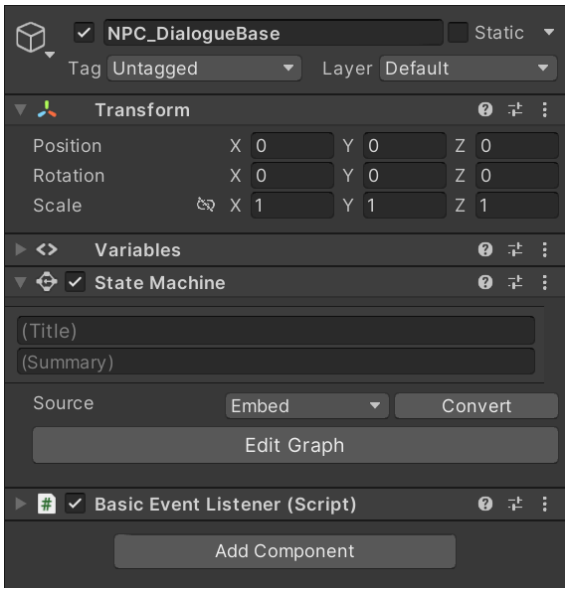
Quick Start Guide (Prefabs and Components)

NOTE!

Any conversation must be an enabled gameobject in the scene - without this, conversations will not be registered to the dialogue system and will not be able to be shown! There is a basic prefab for Conversation objects located at Assets/Minotaur/Prefabs/Gameobjects.

The Conversation Prefab

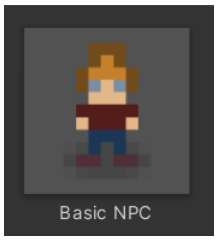
An instance of a conversation prefab looks like this:



To create a new conversation, drag the prefab, unpack the prefab by right-clicking on it and selecting Prefab/Unpack Completely, then click on the object and press “Edit Graph”.

Basic NPC Prefab

By default, the only component that will need to be changed on the Basic NPC prefab is the “Talkable NPC” component. *(Located at Assets/Minotaur/Prefabs/Gameobjects)*



On that component, change the “My Character” field to whichever character you’d like this NPC to trigger conversations for.

[Back to Table of Contents]

Barks

Overview

Barks in Minotaur are built using Scriptable Objects. Barks can be triggered from any event and allow for any amount of conditions to make them only trigger under specific circumstances.

Note: If you're unfamiliar with the Scriptable Object Event pattern in Unity, it's recommended you read the Scriptable Object Events Overview section before continuing.

Key Concepts

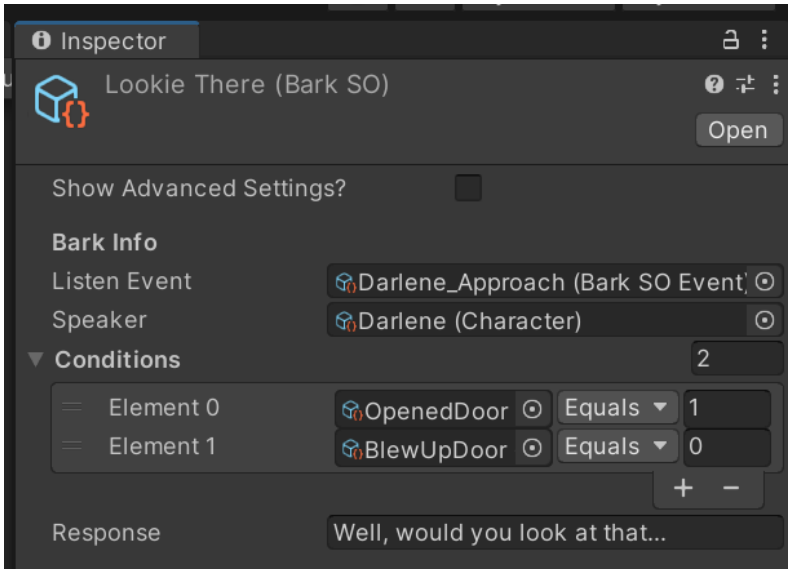
Bark Concepts

- *Remember:* Functionality that allows barks to change a variable when they complete.

Back to Table of Contents

Quick Start Guide (Bark SOs)
By default, barks need an event to listen for, a speaker, any number of conditions (can be zero!), and a response.

A complete bark looks basically like this:



Listen Event
The event that triggers this bark if all conditions are met. *(See Listening for SO Events for more information.)*

Speaker
The character this bark is registered to. *(See Barking NPC for more information.)*

Conditions
A list of conditionals.

Note! A bark can only be played when *all* conditions are met.

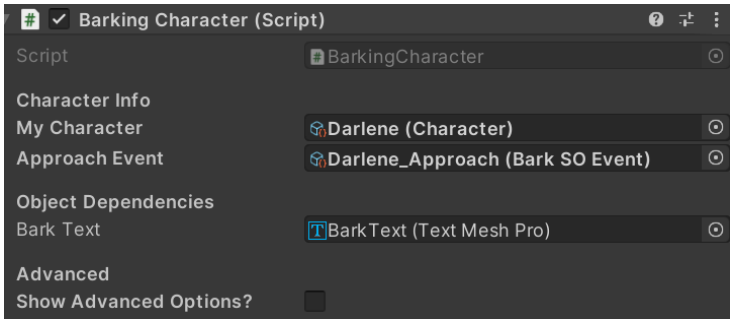
Response
The text to display over the barking character.

Back to Table of Contents

Quick Start Guide (Components)

Barking NPC

In order to have an NPC bark, they must have a Barking NPC component. The “My Character” field is the most important field - without it the script will not function, The “Bark Text” must also contain a valid reference to a TMP Text object otherwise this NPC will not display their barks.



Finally, “Approach Event” is an optional field for a Bark SO Event. This event is triggered when the NPC is on-screen and close enough to the center for it to be reasonable for them to bark.

Back to Table of Contents