SQUAWK (Simple QUasual AdWenture Kreator) for Unity and Bolt User Manual version 0.1

Contents

Installation guide

Step 1: Import and install Bolt

IMPORTANT: Squawk^{TM} requires Bolt, Unity's free visual scripting solution. You must import and install Bolt into your Unity project BEFORE you import the Squawk engine.

Read the "Unity Visual Scripting" page:
https://unity.com/products/unity-visual-scripting

Get Bolt on the asset store:
https://assetstore.unity.com/packages/tools/visual-scripting/bolt-163802

Read the Bolt manual:
https://docs.unity3d.com/bolt/1.4/manual/index.html

Unity have said that Bolt will be integrated into Unity at some point in 2021.

Step 2: Install Text Mesh Pro

Squawk also uses TMPro. Though it is not essential to install TMPro before Squawk, I recommend doing so. TMPro is a standard package in unity.

Read the documentation:
https://docs.unity3d.com/Packages/com.unity.textmeshpro@2.1/manual/index.html

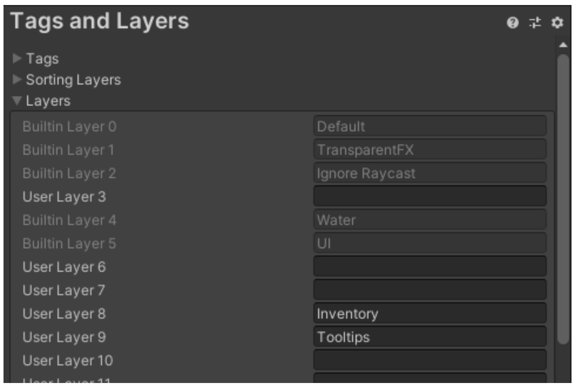
Step 3: Import Squawk

Import Squawk v 0.1 for free from the Unit Asset store. Select and import all files and folders.

Step 4: Set up layers

The engine needs two layers set up – "Inventory" on layer 8 and "Tooltips" on layer 9.

Do this by applying tag and layer settings using the "TagManager" preset located in the root Squawk directory.

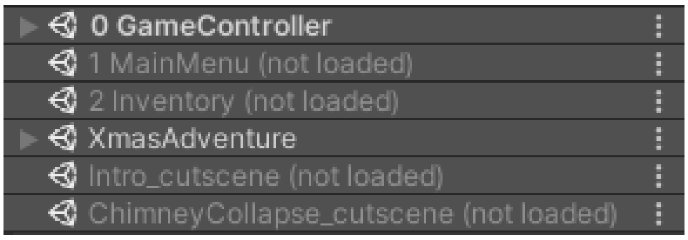


Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
User Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	UI
User Layer 6	
User Layer 7	
User Layer 8	Inventory
User Layer 9	Tooltips
User Layer 10	

Step 5: Set up scenes and play demo game

Then, from the Squawk->Scenes folder, add all six of the following scenes to your unity project:

0 GameController
1 MainMenu
2 Inventory
XmasAdventure
Intro_cutscene
ChimneyCollapse_cutscene


0 GameController
1 MainMenu (not loaded)
2 Inventory (not loaded)
XmasAdventure
Intro_cutscene (not loaded)
ChimneyCollapse_cutscene (not loaded)

The order doesn't matter except that the GameController must be the first scene in the "Build Settings" (step 6). The numbers "0", "1", "2" before the first three scenes are a suggested order and to remind you NOT to change the names of these scenes and to be careful about editing their contents until you know what you're doing. These first three scenes contain components and references that are part of the core Squawk engine. The other three scenes contain the main data for the specific XmasAdventure game.

Check the Unity manual pages on Multi-Scene Editing if you're not sure about adding multiple scenes:
https://docs.unity3d.com/Manual/MultiSceneEditing.html

Once added, you can remove the default scene that Unity creates for a new project. Then unload (don't remove!) all the scenes except the GameController. Press play in the unity editor to play "A Christmas Adventure" – a short game demonstrating the capabilities of the Squawk engine.

You can also play a WebGL build of A Christmas Adventure on Newgrounds:
https://www.newgrounds.com/portal/view/777377

Step 6: Build settings

To build the project, first load all the scenes back in the Unity editor. Then go to build settings and add all active scenes, making sure the GameController is the first in the list. Then build to your desired platform (currently WebGL and PC have been tested).

Introduction

Squawk enables the creation of 2D point-and-click adventure games and escape room games in first-person perspective or casual games produced by studios like Artifex Mundi and FIVE-BN. Using visual scripting (via Bolt, Unity’s visual scripting “solution”), it is possible to create simple games without writing a line of code.

Play "A Christmas Adventure" on Newgrounds, a jolly adventure game created with Squawk, and included as an example project:
https://www.newgrounds.com/portal/view/777377

Squawk comes with a library of C# classes and Bolt macros for handling interactable objects, complex puzzle dependencies and consequences, navigation between locations, inventory management, and diverting to cutscenes, all with visual scripting. Ready-made scenes and prefabs enable the rapid setup of a new game.

Scriptable objects (SOs) are used to simplify game development in three key areas:
* Inventory item SOs provide easy references for checking an item a player is using against requirements on target interactable objects.
* Consequence SOs provide a powerful and flexible system for checking whether defined consequences have been fulfilled.
* Conversation SOs provide a simple way to set up conversations and monologues for automatic playback during cutscenes.

Squawk 0.1 is functional but has a limited set of features and, in particular, does not include integrated game saving. Consequently, the current release of Squawk is best suited to short or episodic games or for rapid prototyping of single chapters in a larger game. Target build platforms are desktop computers or WebGL, not mobile devices, since interaction with a 3-button mouse (or 2 buttons and the spacebar) and a 1280x720 resolution are currently hard-coded into the engine.

Having said that, Squawk’s C# libraries are compact and heavily commented. A Unity developer of medium skill could easily expand the engine’s capabilities.

If feedback on this 0.1 release is positive, development will continue with the addition of features like conversation trees and more complex inventory management. It is intended that all releases will be free until version 1.0, at which point save game functionality will be added.

Key Squawk Features

Squawk has some important features to simplify the creation of complex functionality:

1. Library of Bolt macros to create game logic with visual scripting
2. Scriptable objects for Inventory Items
3. Scriptable objects for “Consequences” and complex dependency checking
4. Scriptable objects for simple Conversations

Bolt macros for creating game logic

A Squawk game consists of a Unity scene containing a single Chapter component. One or more Location components are created as children of the Chapter component, these being the locations a player can visit. Every Location has one or more BoltInteractable components (interactables). Interactables contain the majority of the game logic.

An interactable has an embedded Bolt Flow Machine with an Interactable macro/super unit inside it. This is used to initialise starting values for the interactable through its inputs. The outputs from the Interactable super unit are activated when a player clicks within the interactable's collider area in the game. The actions that are connected to those outputs (which will often be other Bolt macros from the Squawk library but could be any functions you like) define the gameplay.

Inventory Item Scriptable Objects

If you have an item that can be put into the inventory, you create an InventoryItemSO scriptable object for it. This is done by right clicking in the folder where you want to create the SO and, from the pop-up menu, selecting Create->Squawk->Inventory Item.

Once created, an InventoryItemSO is given an item display name and an item description. These are used when the item is in the inventory as the name of the item and the description used when the item is right clicked.

An InventoryItemSO also requires a 64x64 pixel sprite. This is the image used for the item one it is in the inventory. When importing image assets to use as inventory item sprites, you can apply the "import_inventory" preset to the image. This applies the key import settings. **IMPORTANT: graphics used by Squawk should be set to 1 Pixel Per Unit**.

To see some examples, inventory item SOs for A Christmas Adventure are stored in the Scriptable Objects->InventoryItems folder.

In interactable flow machines, the inventoryitemSOs are selected as references whenever a test against an inventory item is required. This permits easy checking of an item against a puzzle solution. It also simplifies the writing of different feedback when using different items with interactable objects rather than just default responses like "That didn't work". Of course, default responses can be set up using Bolt's scene variables.

Consequence Scriptable Objects

Good adventure and puzzle games have complex puzzle dependencies and multiple consequences may derive from completing an action. To keep track of these events, a ConsequenceSO can be created by right clicking in the folder where you want to create the SO and, from the pop-up menu, selecting Create->Squawk->Consequence.

ConsequenceSOs contain no data and are just used as named references for tracking and fulfilling consequences.

In interactable flow machines, the ConsequenceSO is used as an easy reference to create complex actions and reactions. When a consequence is fulfilled, it can cause reactions that occur immediately or that change the flow response of an interactable object the next time it is clicked on. See the sections below on the Bolt macros "Consequence" and "ConsequenceFulfilment".

Cutscene Conversation Scriptable Objects

If you want a conversation between multiple characters, or even a monologue, the CutsceneConversationSO simplifies its creation and the conversation can be played back using a Bolt CutsceneConversation flow macro during a cutscene.

CutsceneConversationSOs are created by right clicking in the folder where you want to create the SO and, from the pop-up menu, selecting Create->Squawk->Conversation.

Once created, a list of people in the conversation is created using rich text formatting codes to display their name in a chosen colour or with other desired formatting.

A second list of each line of dialogue is created together with the index value of who is speaking. When played back using the CutsceneConversation flow macro, the conversation continues automatically with each line of dialogue.

To see some examples, Cutscene Conversation SOs for A Christmas Adventure are stored in the Scriptable Objects->Conversations folder.

Scenes and components

This section of this manual goes through the six scenes in the example game, explains how they work and goes through their components. The first few scenes mainly focus on the operation of the Squawk engine which you will need to know at some point. To dive straight in and create new game content, turn to the section "XmasAdventure scene" and particularly its sub-sections "Location components" and "BoltInteractable components".

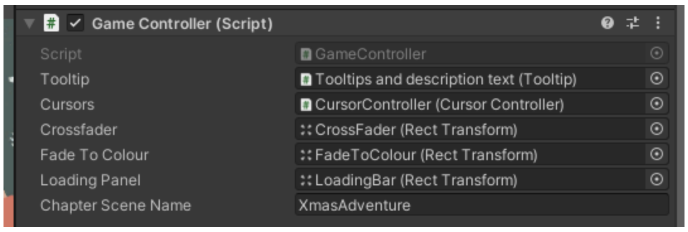
"0 GameController" scene

The GameController scene contains the main GameController component, which initialises and runs the game, plus components for managing the context responsive cursor, and displaying text windows, tooltips and UI overlays for fading or cross-fading between scenes.

The "0 GameController" scene initialises and starts the game and should therefore be the first scene in the build settings. For testing, however, it is possible to have all scenes unloaded except for the Chapter scene (see below) as the Chapter component will check whether there's a GameController and load the GameController scene if there isn't. This means you must not rename the "0 GameController" scene or this feature will break.

GameController component

A game has a single GameController (GC) that initialises and starts the game, first loading the "1 Main Menu" scene, and then loading and initialising the specified Chapter scene.


Game Controller (Script)

Script	GameController
Tooltip	Tooltips and description text (Tooltip)
Cursors	CursorController (Cursor Controller)
Crossfader	CrossFader (Rect Transform)
Fade To Colour	FadeToColour (Rect Transform)
Loading Panel	LoadingBar (Rect Transform)
Chapter Scene Name	XmasAdventure

The GC editor should be populated with references to the Tooltip and CursorController components and to the Crossfader, OverlayFader, and LoadingPanel objects. These can usually be left as they are.

The final parameter, chapterSceneName, is a string with the name of the scene containing your Chapter. Currently, Squawk only supports a single Chapter. When the Chapter scene is loaded, the Chapter object is assigned to the variable currentChapter.

After initialisation, most method calls go to the Chapter object. For some Bolt macros, though, where the flow or state machine can be placed on a number of different objects, it's easier to do a FindObjectOfType <GameController>() call to find the GameController, and then get the currentChapter reference from the GC. But make sure not to overdo FindObjectOfType calls if you write you own macros.

Tooltip component

The Tooltip component and its children contain various background panels and TMPro objects for displaying description text at the top of the screen and pop-up hints when you mouseover objects or activate the "show all hints" function with the spacebar or middle mouse click.

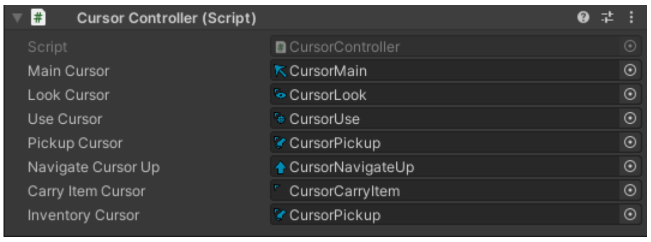
The Tooltip component and all its children should be on the "Tooltips" layer and the TooltipCamera child should have its culling mask set to "Tooltips". All other cameras in the game should exclude the Tooltips layer.

The Tooltip component has its transform set to (-650,360,0) to move it to the left of the UI overlay, making it visible for testing, aligning it with the Inventory display (see below), and to keep it out of the way of the Location components (see below)

This component can largely be left alone, though you should be able to do simple adjustments like changing the size, shape or colour of the background panel or delete the "click to continue" text without breaking anything.

All of Squawk's TMPro components refer to the same TMPro font material, so make sure to create new materials if you want to tweak them on any one instance.

CursorController component



Cursor Controller (Script)

Script	CursorController
Main Cursor	CursorMain
Look Cursor	CursorLook
Use Cursor	CursorUse
Pickup Cursor	CursorPickup
Navigate Cursor Up	CursorNavigateUp
Carry Item Cursor	CursorCarryItem
Inventory Cursor	CursorPickup

The CursorController contains references to the different shaped cursors used by Squawk based on in-game context.

For example, the "look" cursor has an eye in it for hovering over interactables you can only look at, and the "pickup" cursor has a hand for hovering over things you can pick up. Selecting cursor behaviour for different objects is done in a flow machine on a BoltInteractable component (see below).

There's also a "carry item" cursor, which is just a small triangle. This cursor is used when an item is picked out from the inventory and ensures that the icon of the carried item is not obscured.

There's also an "inventory" cursor for use in hovering over the inventory, but this isn't implemented in the current version of Squawk.

The CursorController makes used of an enum and dictionary, so is simple to expand if you want more flexibility. For example, the "navigate" cursor is currently limited to a single up arrow, but you could add arrows pointing in different directions.

If you want to import your own cursor, there is an "import_cursors" preset in the Squawk->Graphics folder to make it easy to apply the correct settings.

UI Overlay objects

Three different UI screen overlays are provided: a loading bar used while the Chapter is initialising; a crossfader object that uses a render texture to crossfade between locations; and a simple full screen overlay that can be faded in and out to a desired colour.

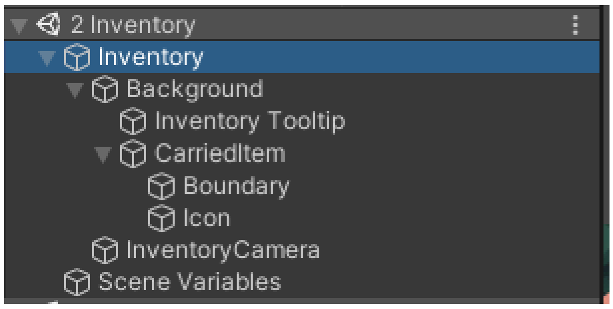
These don't need to be changed and be careful of breaking Bolt macros if you do, though you could easily tweak the look of the loading bar to fit your game's aesthetic.

"1 Main Menu" scene

There's not much here. Just an image to use as a background for the main menu and a "play" button that invokes the StartNewChapter method on the GameController.

"2 Inventory" scene

The inventory scene manages an inventory at the bottom of the screen, adding items that are picked up in game and allowing them to be used later with interactables. Current limitations: inventory items cannot be used together, and the inventory is limited to 12 items before it overflows.



The inventory is kept as a separate scene so that, when multiple Chapters are implemented, any chapter can refer to it. Also, this makes it easy for different chapters to use different inventories.

There are a lot of components in the inventory scene and this is just a brief summary. If you want to adjust or expand the inventory's capabilities, the C# code is heavily commented. The inventory related scripts are contained in the Squawk->Scripts->Inventory folder.

You can freely change the colour of the Background child component to fit your game's aesthetic. There is also a pre-built inventory prefab in the Squawk->Prefabs directory if you need to recreate the inventory from scratch. Be careful about changing or overwriting this prefab.

Inventory component

The inventory is the main inventory management component and needs an inventory camera childed to it. Every object under the inventory must be on the "Inventory" layer and the inventory camera should have its culling mask set to "Inventory".

The Inventory component has its transform set to (-650,360,0) to move it to the left of the UI overlay, making it visible for testing, aligning it with the Tooltips display (see above), and to keep it out of the way of the Location components (see below).

Most of the references in the Inventory script are to child objects of the Inventory component. The Cursors and InventoryCamera references are populated on initialisation, so you don't need to set those up in the editor.

The "blankBox" reference is to a "BlankInventoryBox" prefab located in the Squawk->Prefabs directory. Instances of this prefab are created when new items are added to the inventory.

Inventory Tooltip component

The inventory tooltip is a small text area that pops up a description text for inventory items when you hover over them or look at them.

Carried item component

When you select an item from the inventory, it is picked up and carried by the cursor. The Carried Item component is activated and the icon populated with the icon for the inventory item.

Inventory Camera component

This is a camera set up to see the inventory and should have its culling mask set to "Inventory" as mentioned above. The camera is turned on or off to show or hide the inventory.

Xmas Adventure scene (game Chapter scene)

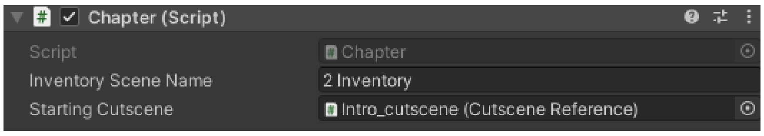
This scene, or any scene containing a Chapter object (preferably in a root game object in the scene), is where you "make" your own game. When you create your own Chapter scene, make sure to put the exact scene name in the chapterSceneName field in the GameController.

A Chapter scene is made from a single Chapter component having multiple Location components as children, and each Location component has multiple BoltInteractable components as its children.

Optionally, a Chapter Scene also has a single CutsceneController component having one or more CutsceneReference components as children.

Chapter component

A Chapter scene must have a single Chapter component and, in the current implementation, there must only be one loaded scene containing a Chapter component.



Chapter (Script)

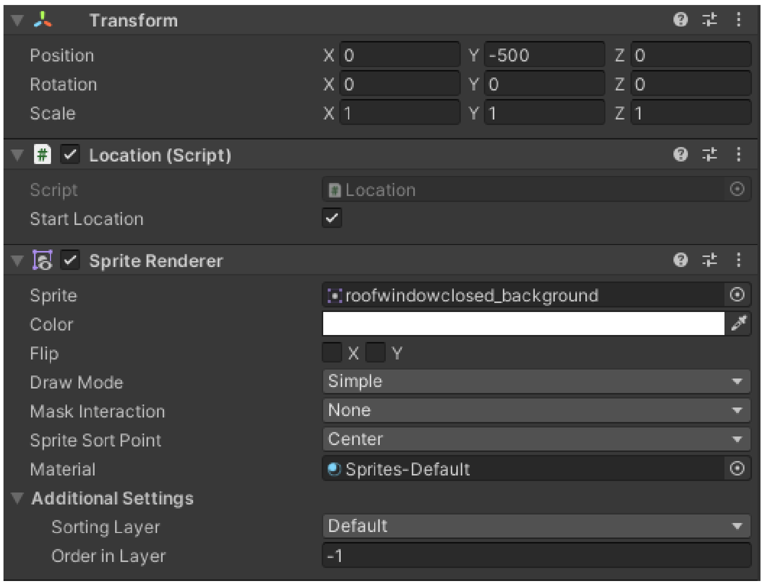
Script	Chapter
Inventory Scene Name	2 Inventory
Starting Cutscene	Intro_cutscene (Cutscene Reference)

The Chapter component handles initialisation and some other high level functions during the game so requires very little set up. The inventorySceneName must contain the name of the scene containing the Inventory component (see above). The startingCutscene is an optional reference to a cutscene that is played when the game starts (see the CutsceneController section below).

Location components

A Chapter is made up of one or more Locations, which the player navigates between as they explore your game world. Every Location component must be a separate child of the Chapter component.

There is a Location prefab in the Squawk->Prefabs directory to make it easy to add Locations to a scene. Drag and drop the prefab under the Chapter object, then right click->Prefab->Unpack completely to separate the scene instance from the prefab.



Transform

Position	X 0	Y -500	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Location (Script)

Script	Location
Start Location	✓

Sprite Renderer

Sprite	roofwindowclosed_background
Color	
Flip	X Y
Draw Mode	Simple
Mask Interaction	None
Sprite Sort Point	Center
Material	Sprites-Default
Additional Settings	
Sorting Layer	Default
Order in Layer	-1

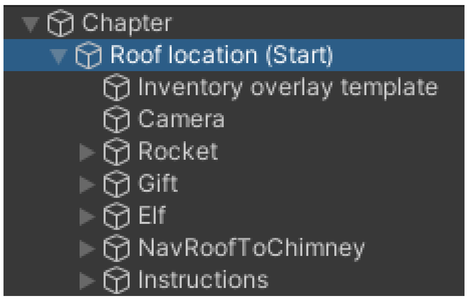
One and only one Location in the Chapter must be marked as a startLocation by ticking the box in the Location component. This is the Location where the game starts. It's worth putting the word "start" in the name of the Location component object to remind you.

The names you give the Location objects do not affect the game, but each should be unique. This is so that you can easily find the Location you want when selecting navigation targets in BoltInteractable components (see below).

The Location component should also have a SpriteRenderer containing a 1280x720 image that serves as the background to the Location (unless you want the background to be completely black!). To ensure that the background is behind other objects in the Location, you can set its layer order to "-1". You can set up extra sorting layers if you want, but leaving everything on the Default sorting layer should work fine for all but the most complex scenes.

When importing image assets to use as Location backgrounds, you can apply the "import_background" preset to the image. This applies the key import settings. **IMPORTANT: Backgrounds and all other graphics used by Squawk should be set to 1 Pixel Per Unit**.

You can use the transform of the Location component to arrange your different Locations in the editor scene view so they don't overlap. Arrange them to create a representative map of Locations in the scene editor view. If your map gets too big for the available space and starts overlapping the UI overlay or inventory/tooltips view, you can more easily move the whole map by changing the transform of the Chapter component.



The children of the Location include a location Camera, an optional inventory overlay to show which areas of the screen are covered by the inventory, and one or more interactable objects. For example, the Roof Location has five interactables as children.

The child Camera component should have its transform set to (0,0,-10). z=-10 is to give space to arrange colliders on BoltInteractable objects in the Location. The camera should also be orthographic and its culling mask should exclude the "Inventory" and "Tooltips" layers.

The inventory overlay template should be on the "inventory" layer so that it isn't "seen" by the Location camera.

BoltInteractable components (interactables)

These are the components that provide interactivity to your game. You will spend most of your time adding and setting up these components and references between them. You can create them from scratch, or you can drag and drop the Interactable prefab in the Prefabs folder under the Location object, then right click->Prefab->Unpack completely to separate the scene instance from the prefab.

You can name the game object containing the interactable component however you want. But each should be uniquely named. This is so that you can easily find the one you want when selecting interactables as targets for game functionality.

The BoltInteractable component itself looks very boring in the inspector. When you add one, it automatically adds a static RigidBody2D (so that colliders in child objects can be found) and a Bolt FlowMachine. We'll set up the FlowMachine and the interactable's transform component a little later, but for now it's worth setting the transform to (0,0,-1) to centre it within the Location. I'll also explain why z=-1 below.

Interactable appearance

The appearance of the interactable and its clickable area are defined by adding children to the interactable. The various interactables in the Roof location show a few different options.

The Rocket and the Chimney are the most basic examples – each has a single child with a BoxCollider2D component on it to define the interactable's clickable area. You can use circle colliders or a combination of both to define whatever size and shape of clickable area you need. No other children are needed because the rocket and chimney are part of the Location's background graphic.

The Elf and the Gift are not part of the background so, in addition to a box collider, each has a SpriteRenderer child containing a graphic for the interactable. The SpriteRenderer layer order is set to 0 (or higher) so that it appears above the background image in the Location.

When importing image assets to use as interactable sprites, you can apply the "import_interact" preset to the image. This applies the key import settings. **IMPORTANT: graphics used by Squawk should be set to 1 Pixel Per Unit**.

The Instructions interactable contains graphics and text, but doesn't have a collider because the player is not able to click on it or interact with it directly. Nevertheless, because it's an interactable component, it can be controlled via other interactables. For example, if you play the Christmas Adventure game and climb halfway down the chimney, then back to the roof, the instructions have disappeared because the interactable is deactivated when you navigate down the chimney.

Note that it's important for graphics or other components of the interactable to be on children of the interactable object itself because activating and deactivating an interactable affects only the children. The interactable component and the object it's attached to are always active and enabled.

Another important example is in the Living Room Location. Look at the "Window states" object. It's an empty object, but contains several different interactable children. You can group together related interactables under an empty gameobject like this to make it easier to keep track of them or, if needed, reposition them all as a group.

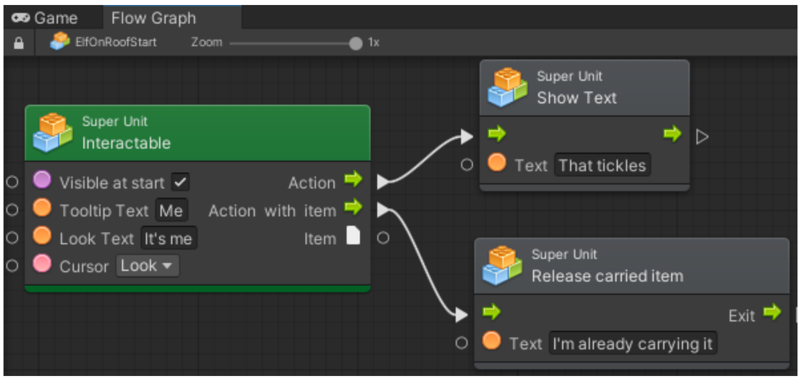
When first setting up an interactable, you should set the transforms for all the child components to (0,0,0) so everything is centred on the same place. The parent interactable can then be translated and rotated to the desired position within the Location. Then, if a graphic needs to be centred on a different place than its collider, or if the collider needs to be rotated to fit around a graphic, you can adjust them relative to each other. Having everything on separate child components gives you this level of control.

Returning to the interactable's transform, we set z=-1 to remind you that colliders stack in 3D space. The camera is location at z=-10, so you can place your interactables anywhere from z=-1 to z=-9. If interactables overlap, this is how you control which one's clickable area is in front. For layering graphics, you instead use the sort order on the SpriteRenderer component.

Interactable functionality

Now the appearance of the interactable has been set up, we can turn to the functionality. This is done using Bolt Flow Machines, so select the interactable object and open a Flow Graph tab.

If you created a BoltInteractable component from scratch rather than using the prefab, the Flow Machine on the interactable needs to be changed to the "embed" type. You can usually delete the Start and Update function. Instead, add an "Interactable" macro (or super unit) to the flow machine.



(Developer's Note: if anyone knows how to do this automatically from the C# script when you first add a BoltInteractable component, let me know! Thanks!)

The Interactable macro can be found in the Bolt Macros->Interactables folder, or you can right click in the flow graph ->Macros->Interactable to select it in the "fuzzy finder".

The Interactable super unit is used to set initial values for the interactable and to provide different functionality when the player clicks on the interactable or when they click on it with an inventory item in their "hand".

The four initialisation values are:

1. "Visible at start". A tick box for whether the interactable is visible at the start of the game or only appears due to subsequent actions. An example might be a door. At the start of the game, the interactable representing the closed door is visible, but the separate interactable representing the open door is hidden. When the door is opened, the closed door is deactivated and the open door is activated.
2. "Tooltip text". This is the text that appears when the player hovers the cursor over the interactable.
3. "Look text". This is the text that appears when the player right clicks on an interactable to display the default "look text".
4. "Cursor". This is the cursor that appears when the player hovers over the interactable. It is selected from a drop down list based on the enum defined in the CursorController class.

The three outputs from the Interactable super unit are:

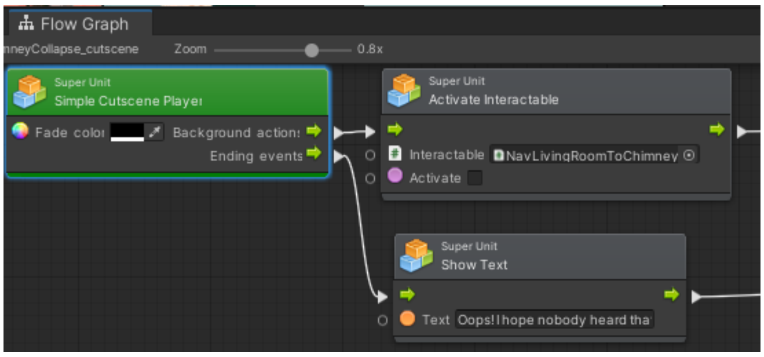
1. "Action". The action or chain of actions that occur when the player left clicks on the interactable. These actions will often be other Bolt super units but could also be standard Bolt flows or direct calls to Unity behaviours or C# script functions.
2. "Action with item". As above, but a different flow if the player is carrying an item from their inventory.
3. "Item". The item the user is carrying in the "Action with item" flow. Usually for comparing with one or more required item to choose the resulting behaviour.

Complex behaviours can be mapped out by using chains of actions and interactions from the Interactable super units outputs. Some of the more complex behaviours you might need in a game are already implemented in Squawk's various Bolt macros. A full list is provided below.

Cutscene Controller component

If you want to cut to a separate Unity scene, this can be done by adding a CutsceneController component to the Chapter scene. This enables extremely powerful expansion of Squawk's functionality since the separate cutscene could be anything that you can program in Unity, such as a minigame or other special functionality such as a puzzle box. Squawk comes with a basic conversation system if that's all you want from your cutscenes.

For each cutscene, you create a child object of the CutsceneController with a CutsceneReference component on it. The name of the game object with the CutsceneReference on it must be the same as the Unity scene that contains your cutscene. (Note that this is different from the naming conventions for Locations and Interactables where the name of game object can be anything you want).



The CutsceneReference component automatically adds a Bolt flow machine. Change this to an embedded flow machine and add the "CutsceneReference SimplePlayer" macro/super unit.

This SimplePlayer super unit starts the cutscene and controls other actions within the Chapter that happen in the background or when the cutscene is finished. In the ChimneyCollapse_cutscene, for example, the interactable for navigating to the chimney is deactivated and a pile of rubble is put in its place as a "background action". Then, when the cutscene is finished, the "ending events" include the screen shaking and displaying some text. You could even immediately open a different cutscene!

When you want to play the cutscene from a given Interactable, you add a Bolt macro "InteractableGoToCutscene" to that interactable's flow machine and use the CutsceneReference object as a reference. Or, if it's a cutscene that starts the game, you add the CutsceneRefernce to the Chapter component.

The last step is to set up your cutscene. You can do this however you want using whatever Unity tools and assets you prefer, but the cutscene must have a CutscenePlayer component on a root object in the scene. Preferably, place all other objects in the scene as children of the CutscenePlayer.

You can make your cutscene play however you want. One option is to add a Bolt machine component to the same object as the CutscenePlayer (either a flow machine or a state machine). When the CutscenePlayer starts in your separate scene, it triggers a Bolt Event named "playCutscene" and you can connect whatever functionality you want to that.

Alternatively, as demonstrated in the two cutscenes in Squawk's example project, you can use a Bolt State Machine that transitions from an empty Start state to the first state on clicking the left mouse button.

To leave the cutscene and return to the game, all that is needed is to call the EndCutscene() method on the CutscenePlayer component, optionally by using the "CutscenePlayer EndCutscene" macro.

Bolt Macro reference

Squawk comes with a small library of macros that can be used as super units on your BoltInteractable objects and elsewhere. Each of them is described below. The location of the macro asset under the Bolt Macros folder is listed. Also, pointers to helpful examples of the macro's use in Squawk's example project are given.

Consequence

Folder location: Interactables -> Consequences

Usage: Can be placed on any flow machine in a Chapter scene , but will usually be placed on a BoltInteractable flow machine

For complex behaviours, it can be useful to create a Consequence Scriptable Object (ConsequenceSO). The SO is used as an input to the Consequence macro and the ConsequenceFulfilment macro (see below) to control what happens when the consequence is fulfilled. By right clicking on the ConsequenceSO in the Unity project tab and selecting "Find references in scene", it is possible to find all references to that ConsequenceSO and track the behaviour.

The Consequence macro is added to a flow machine when a given action has a ConsequenceSO associated with it. You can mark whether the consequence is being fulfilled or un-fulfilled. For example, moving a box might fulfil a consequence, but moving it back might un-fulfil it again.

A reference to an affected interactable is also required. If a consequence affects several interactables, you must chain together a series of Consequence macros to apply the consequence to each of them.

Fulfilling a consequence can have both immediate and delayed effects, which are defined by using a ConsequenceFulfilment macro on the affected interactable.

Inputs:

1. "Consequence". The ConsequenceSO being (un)fulfilled
2. "Fulfil". Bool value. Fulfil or unfulfil the consequence = true/false = tick/untick
3. "Affected interactable". The BoltInteractable being affected – could even be the current interactable

Example of use: XmasAdventure scene -> Chapter -> Roof location (Start) -> GiftOnRoof
(The "PickedUpGift" consequence is fulfilled when the gift is picked up and affects the NavRoofToChimneyStart interactable)

ConsequenceFulfilment

Folder location: Interactables -> Consequences

Usage: Can be placed on any flow machine in a Chapter scene, but will usually be placed on a BoltInteractable flow machine.

This macro is used to check whether a ConsequenceSO has been fulfilled on clicking the interactable, and also provides output flows that are triggered immediately when a consequence is fulfilled by a Consquence macro elsewhere in the Chapter.

All ConsequenceSOs are assumed to be UN-fulfilled at the start of the game.

Inputs:

1. "Check consequence". Flow input to the macro to check whether the consequence has been fulfilled. This input is not required if you only want actions to occur immediately that the consequenceSO is fulfilled or un-fulfilled.
2. "Consequence". The ConsequenceSO being checked.

Outputs:

1. "Fufilled". Following a "Check consequence", the flow outputs here if the consequence is fulfilled.
2. "Not fulfilled". Following a "Check consequence", the flow outputs here if the consequence is NOT fulfilled.
3. "Fulfillment action". An immediate action that occurs when the consequence is fulfilled.
4. "UN-fulfillment action". An immediate action that occurs when the consequence is UN-fulfilled.

Example of use: XmasAdventure scene -> Chapter -> Roof location (Start) -> NavRoofToChimneyStart
(Left clicking on the chimney only results in a navigation action once the "PickedUpGift" consequence has been fulfilled)

Example of use: XmasAdventure scene -> Chapter -> Living room location -> Window states -> NavLivingRoomToWindow (locked window)
(a more complex example with chained Consequence Fulfilment macros)

CutsceneConversation

Folder location: Cutscenes

Usage: Must be used on a Bolt State machine

A CutsceneConversation is a State Machine macro. It receives a CutsceneConversationSO as input data and automatically populates a text window with the provided lines of dialogue, proceeding to the next line on a mouse click.

The inputs to the super unit are:

1. "On Enter state". The flow from the basic Bolt Event "On Enter State" for state machines used to initialise the conversation.
2. "Update". The flow from the basic bolt Event "Update". Used to check for user clicks.
3. "Conversation list". A CutsceneConversationSO object containing a list of characters having the conversation (or one character if it's a monologue) and a dialogue list of the text that is output on each click.
4. "Background panel". A reference to a game object having a SpriteRenderer sprite as a background for the dialogue text. A child of the background panel should have a TMPro component for the text.
5. "TMP text". The TMPro component the is the child of the background panel

The outputs to the super unit are:

1. “Conversation finished”. An action to take when the conversation has finished. For example, triggering a state transition or endinf the cutscene.
2. “Next string”. An action to take every time a new line of dialogue is displayed.
3. “List count”. What is the list index of the line or dialogue currently being displayed? This and the “next string” output could be used to change other graphics in the cutscene as the dialogue proceeds.
4. “Character index”. What is the character index of the character speaking the current line of dialogue. Outputs 2 to 4 could be used to change other graphics in the cutscene as the dialogue proceeds, such as displaying an image of the speaker.

Example of use: Intro_cutscene -> CutscenePlayer -> “Conversation” state

See also, the CutsceneConversationSOs in Scriptable objects->Conversations. Notice that the character name text and the dialogue text can use rich text formatting. You can create CutsceneConversationSOs under Assets->Create->Squawk->Conversation.

CutscenePlayerEndCutscene

Folder location: Cutscenes

Usage: Must be placed on a CutscenePlayer object in a cutscene

Simple super unit for use on the CutscenePlayer state machine to end the current cutscene and return to the Chapter.

Example of use: Intro_cutscene scene -> CutscenePlayer -> “End cutscene” state

CutsceneReferenceSimplePlayer

Folder location: Cutscenes

Usage: Must be placed on a CutsceneReference object

For use on the CutsceneReference flow machine. When the CutsceneReference is called by a flow machine on an interactable, it fades up a UI overlay to the input colour, the runs the cutscene with the same scene name as the CutsceneReference game object name.

Inputs:

1. “Colour”. What colour to fade in and out to when starting and ending the cutscene?

Outputs:

2. “Background actions”. Actions that occur in the Chapter in the background while the cutscene is playing (technically, just after the overlay has faded up)
3. “Ending events”. Actions or events that occur when the cutscene has finished.

Example of use: XmasAdventure scene -> Cutscenes -> ChimneyCollapse_cutscene

CutsceneReferenceStartAndEndFlow

Folder location: Cutscenes

Usage: Must be placed on a CutsceneReference object.

When a CutsceneReference object is called, it triggers three different events. This macro listens for those events and flow goes to the macro's outputs when triggered so that actions can be connected to the triggers.

This macro is not intended to be used directly, but is a key component inside the CutsceneReferenceSimplePlayer macro. If you want to use it yourself, check out the CutsceneReference C# script to understand when the different flows are triggered.

CutsceneTransitionOnLMB

Folder location: Cutscenes

Usage: Must be used on a Bolt State machine

Triggers a transition on a state machine when the player clicks the left mouse button.

Intro_cutscene scene -> CutscenePlayer -> "Update: Transition on LMB" transition

Interactable

Folder location: Interactables

Usage: Must be placed on a BoltInteractable flow machine

The core macro for every BoltInteractable. Initialises the interactable with starting values and provides flow outputs for when a user clicks on the interactable.

The four initialisation values are:

1. "Visible at start". A tick box for whether the interactable is visible at the start of the game or only appears due to subsequent actions.
2. "Tooltip text". This is the text that appears when the player hovers the cursor over the interactable.
3. "Look text". This is the text that appears when the player right clicks on an interactable to display the default "look text".
4. "Cursor". This is the cursor that appears when the player hovers over the interactable. It is selected from a drop down list based on the enum defined in the CursorController class.

The three outputs from the Interactable super unit are:

1. "Action". The action or chain of actions that occur when the player left clicks on the interactable. These actions will often be other Bolt super units but could also be standard Bolt flows or direct calls to Unity behaviours or C# script functions.
2. "Action with item". As above, but a different flow if the player is carrying an item from their inventory.
3. "Item". The item the user is carrying in the "Action with item" flow. Usually for comparing with one or more required item to choose the resulting behaviour.

Example of use: every single BoltInteractable!

InteractableActivate

Folder location: Interactables -> Actions

Usage: Can be placed on any flow machine in a Chapter scene

Activate or deactivate the specified interactable object. Note, this works by activating or deactiving the children of the BoltInteractable, not the BoltInteractable itself. This is why all graphics, colliders, etc of an interactable must be placed on child objects.

Inputs:

1. “Interactable”. The interactable to (de)activate
2. “Activate”. Bool value. Activate with true/tick. Deactivate with false/untick.

Example of use: XmasAdventure scene -> Chapter -> Roof location (Start) -> NavRoofToChimneyStart (the instructions are deactivated when the user navigates to the chimney)

InteractableChangeCursor

Folder location: Interactables -> Actions

Usage: Must be placed on the BoltInteractable object being changed

Changes the cursor of the current interactable to the cursor selected from the drop down list.

Example of use: XmasAdventure scene -> Chapter -> Living room location -> Window states -> NavLivingRoomToWindow (locked window)
(the cursor is changed when the chimney collapses)

InteractableChangeLookText

Folder location: Interactables -> Actions

Usage: Must be placed on the BoltInteractable object being changed

Changes the “look text” of the current interactable to the string provided.

Example of use: XmasAdventure scene -> Chapter -> Living room location -> Window states -> NavLivingRoomToWindow (locked window)
(the text is changed when the chimney collapses)

InteractableGoToCutscene

Folder location: Interactables -> Actions

Usage: Can be placed on any flow machine in a Chapter scene

Starts the cutscene identified by the provided CutsceneReference. Note that flow cannot continue from this super unit. The CutsceneReferenceSimplePlayer unit on the CutsceneReference object controls what happens while the cutscene is playing and when it is finished.

Example of use: XmasAdventure scene -> Chapter -> Living room location -> NavLivingRoomToChimney

InteractableInventoryContainsItem

Folder location: Inventory

Usage: Must be placed on a BoltInteractable flow machine

Checks whether a given InventoryItemSO is in the inventory or not.

Example of use: None. But could be used on the XmasAdventure scene -> Chapter -> Roof location (Start) -> NavRoofToChimneyStart interactable instead of using a Consequence.

InteractableItemsMatch

Folder location: Inventory

Usage: Usually placed on a BoltInteractable flow machine

Checks whether two inventory items are the same.

Often this macro will be used following an "Action with item" output flow from an Interactable super unit. It checks whether two InventoryItemSOs are the same. The output flows to True or False accordingly.

Example of use: XmasAdventure scene -> Chapter -> Roof location (Start) -> NavRoofToChimneyStart (Checks whether the Gift is being used with the Chimney is a gift and displays a comedy response)

InteractableItemPickup

Folder location: Interactables -> Actions

Usage: Must be placed on a BoltInteractable flow machine

An inventory item is added to the inventory when this flow is entered. Text is optionally displayed at the top of the screen. The interactable is optionally deactivated. Circumstances when you don't want to deactivate the interactable include picking up a single stick from a pile of wood.

Inputs:

1. "Item". An InventoryItemSO that is being picked up
2. "Text". Optional text to display when picking up the item. Leave blank for no text.
3. "Deactivate?". Bool value. Should the interactable be deactivated? True/tick = deactivate. False/untick = leave active.

Example of use: XmasAdventure scene -> Chapter -> Roof location (Start) -> GiftOnRoof

InteractableNavigate

Folder location: Interactables -> Actions

Usage: Must be placed on a BoltInteractable flow machine

Navigates from the current Location to the specified destination Location with a specified fade time (in seconds). Text is optionally displayed at the top of the screen.

1. "Destination". The destination Location.
2. "Text". Optional text to display when navigating. Leave blank for no text.
3. "Fade time". Time taken to cross fade from one location to the next. Use 0 to snap between locations.

Example of use: XmasAdventure scene -> Chapter -> Roof location (Start) -> NavRoofToChimneyStart

InteractableLocationEntered

Folder location: Interactables

Usage: Must be placed on a BoltInteractable flow machine

If you want a particular action to occur only on entering or leaving a location, this macro can be added to an interactable in that location. Examples would be an animation or sound effect that you only want to play while you're in the location.

There's also a bool output checking whether the interactable is currently active or not in case you want the flow to depend on that.

Outputs:

1. "Location entered". The flow when the location is entered.
2. "Location left". The flow when the location is left.
3. "Active?". Is the interactable currently active?

Example of use: XmasAdventure scene -> Chapter -> End credits location -> Animation

InteractableReleaseItem

Folder location: Inventory

Usage: Must be placed on a BoltInteractable flow machine

Depending on how you want your game experience to work, this could be the most common super unit on your interactables – even more common that the core Interactable super unit, since you could be using several at a time.

When an inventory item is used with an interactable, you may want it to stay in the player's "hand", or you may want to release it so it goes back into the inventory. That's what this macro does. Personally, I prefer to always release an item. I think it is a good practice to get into that all branches leading from an "Action with item" flow terminate at an InteractableReleaseItem macro.

There is an optional text to display when the item is released. This may be a default response such as "That didn't work", or you can combine this macro with an InteractableItemsMatch macro to give different responses depending on what item is being used. Leave the input blank if you don't what to display any text.

Example of use: every single BoltInteractable!

InteractableRemoveItemFromInventory

Folder location: Inventory

Usage: Can be placed on any flow machine in a Chapter scene

Removes the specified inventoryItemSO from the inventory. Usually used when the item is used with an interactable, so is connected to the output from an InteractableItemsMatch super unit connected to the "Action with item" output from an Interactable super unit

Example of use: XmasAdventure scene -> Chapter -> Under tree location -> Gap for gift
(The gift is removed from the inventory if it is used with the gap, along with many other actions)

InteractableShowLookText

Folder location: Interactables -> Actions

Usage: Must be placed on a BoltInteractable flow machine

Shows the interactable's "look text" at the top of the screen.

Example of use: XmasAdventure scene -> Chapter -> Chimney location -> Elf

InteractableShowText

Folder location: Interactables -> Actions

Usage: Must be placed on a BoltInteractable flow machine

Shows the provided string of text at the top of the screen.

Example of use: XmasAdventure scene -> Chapter -> Roof location (Start) -> RocketOnRoofStart

InteractableTwoWayCycle

Folder location: Interactables -> Actions

Usage: Can be placed on any flow machine in a Chapter scene

Toggles one interactable on and another off. It does not matter which order the interactables are placed, but one must always be active while the other is inactive. Useful for opening and closing door or windows.

Inputs:

1. "A". The first interactable of the cycle
2. "B". The second interactable of the cycle.

Outputs:

1. "Active interactable". Which of the two input interactables is currently active.

Example of use: XmasAdventure scene -> Chapter -> Living room location -> ClosedWindow
(cycles between the open and closed window interactables)

InventoryOpen

Folder location: Inventory

Usage: Can be placed on any flow machine in a Chapter scene

Opens or closes the inventory. Works by turning off the inventory camera.

Example of use: XmasAdventure scene -> Chapter -> Roof location (Start) -> GiftOnRoof
(Since the gift must be the first item picked up, I decided to have the inventory closed at the start of the Christmas Adventure game and open it only on picking up the gift)

LocationActivateCurrent

Folder location: Locations

Usage: Can be placed on any flow machine in a Chapter scene

Activates or deactivates the current location by turning on/off the inventory and location cameras. Was created for use in starting cutscenes, but the CutsceneReference class now handles this directly.

Example of use: None

LocationChangeBackground

Folder location: Locations

Usage: Can be placed on any flow machine in a Chapter scene

Changes the background of a specified Location to a specified sprite.

Example of use: XmasAdventure scene -> Chapter -> End credits location -> Animation
(The background changes from a congratulations message to the credit screen on each click)

LocationShakeScreen

Folder location: Locations

Usage: Can be placed on any flow machine in a Chapter scene

A classic “screen shaking effect” that works by randomly shaking a location’s camera.

Inputs:

1. “Location”. The location to shake.
2. “Vertical”. The amount of vertical shaking.
3. “Horizontal”. The amount of horizontal shaking.
4. “Durection”. The duration of the effect in seconds.

Example of use: XmasAdventure scene -> Cutscenes -> ChimneyCollapse_cutscene
(the screen shakes when the cutscene ends)