For more extensive documentation, please visit https://canvasflow.xyz/.

3. Storyboards

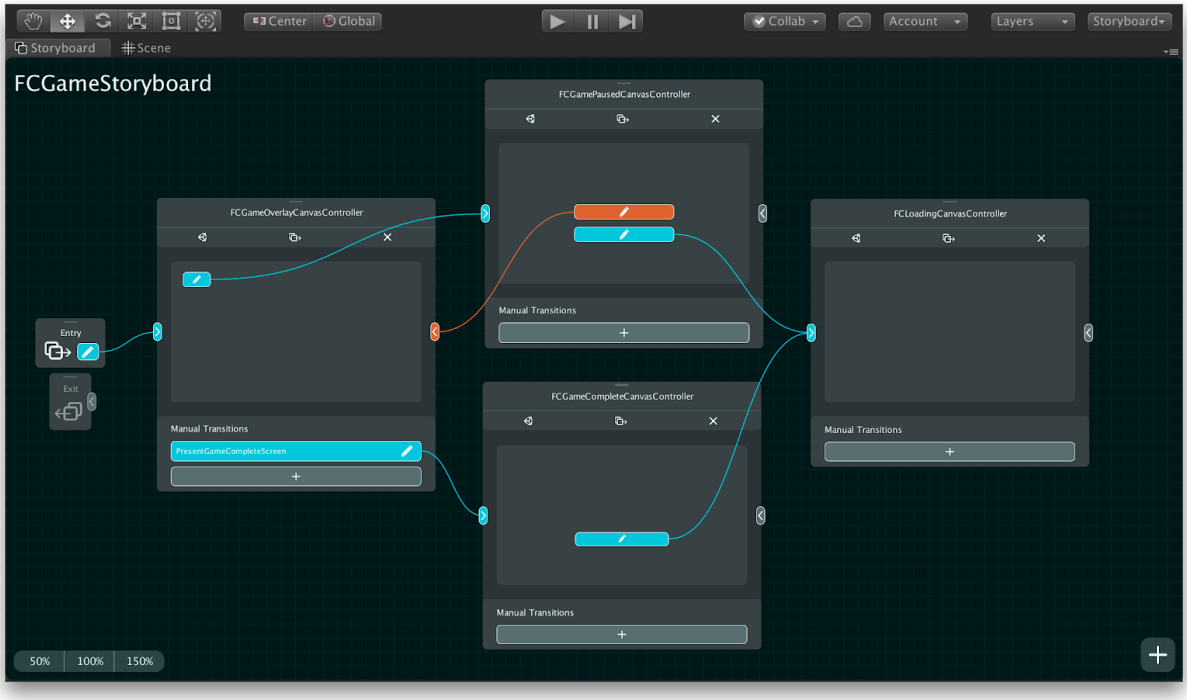
Contents

Introduction

Storyboards allow you to visually create the flow of your user interface by defining the canvas controllers involved (nodes) and the connections between them (transitions).

Instead of writing code to present and dismiss canvas controllers, you can create a storyboard to describe the flow between your screens. Canvas Flow can then automatically present and dismiss your canvas controllers as the relevant buttons, or *'hooks'*, are triggered.

Storyboards are designed in the Storyboard Editor window, shown below, which can be opened by either double-clicking on a storyboard asset in the project inspector or selecting *Window/Canvas Flow/Storyboard* in the menu bar.

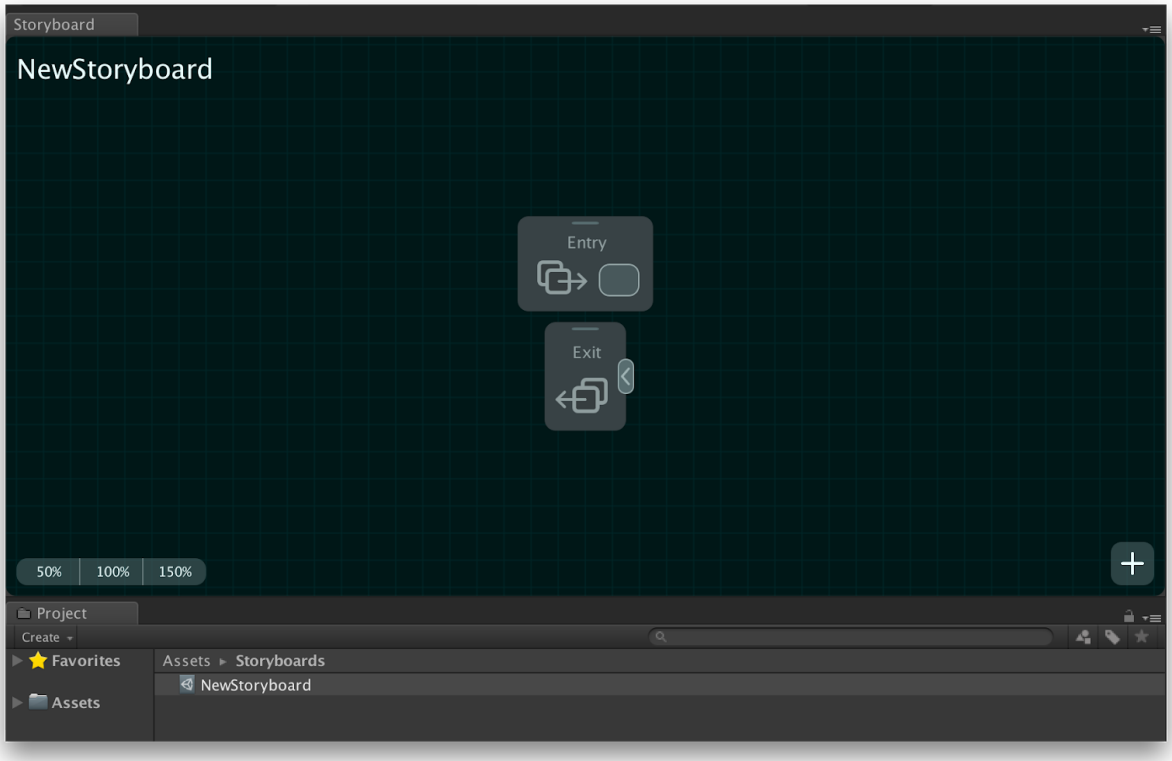


The in-game storyboard from Canvas Flow's Floaty Cube example, containing four screens.

Creating A Storyboard

To create a new storyboard, open Unity's *Create* menu - by either right-clicking in the Project window or selecting Assets in the menu bar - and selecting *Create/Canvas Flow/Storyboard*. Enter a name and a directory for your new storyboard.

This will create a new storyboard file at the specified location and open it in the Storyboard Editor window.



A newly-created, empty storyboard.

Basic Controls

To **pan the storyboard editor window**, you can either:

- Scroll.
- Click and drag.

To **zoom the storyboard editor window** in and out, you can either:

- Hold the *ALT* key and scroll.
- Use the *zoom* buttons in the lower-left corner of the window.

To **move a storyboard node**:

- *Click & drag* its title bar.

To **move an entry or exit node**:

- *Click & drag* it.

Storyboard Nodes

A storyboard node represents a single instance of a canvas controller in the storyboard. Nodes can then have connections made between them, called transitions, which define the flow of your user interface.

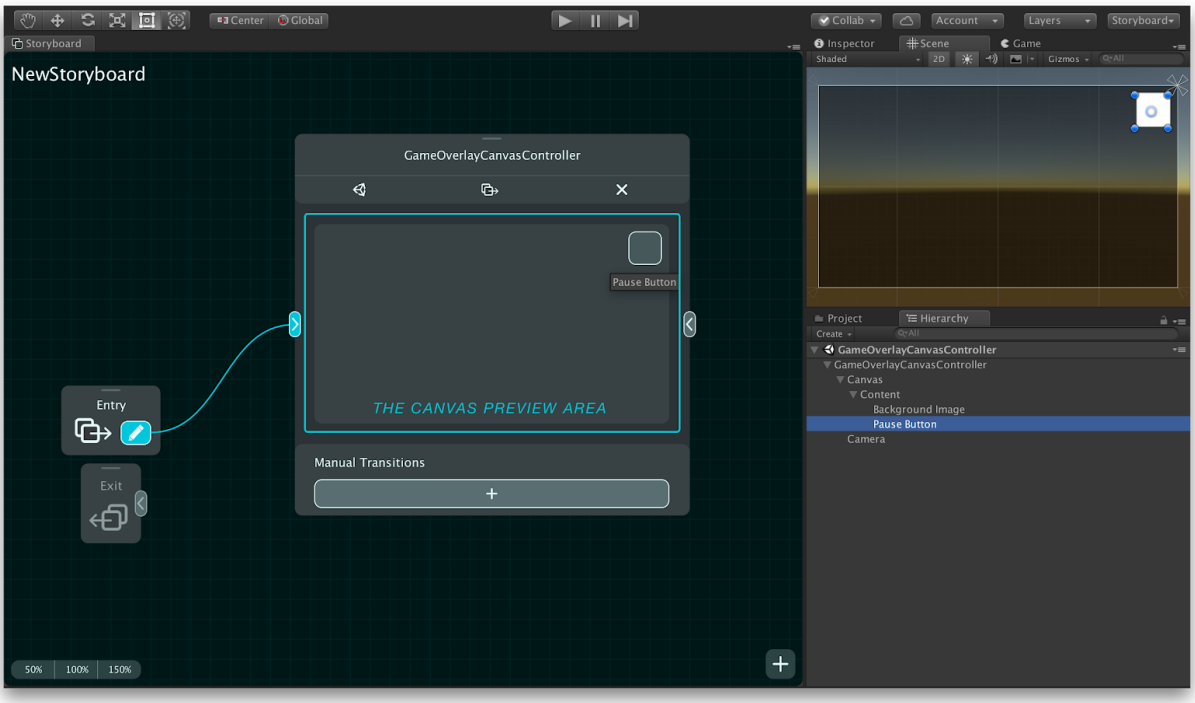
Adding Nodes

To add a node to a storyboard, simply drag the canvas controller's scene file onto the storyboard window.

Alternatively, you can use the '+' button in the lower-right corner of the Storyboard Editor window to select an existing canvas controller in the file inspector.

Node Details

A node represents a single instance of a canvas controller in the storyboard. It will display any *hookable* elements - such as Unity's UI Buttons - in the canvas preview area of the node, as shown below. A *hookable* element is one that a transition can be created from, in order to present or dismiss a canvas controller.

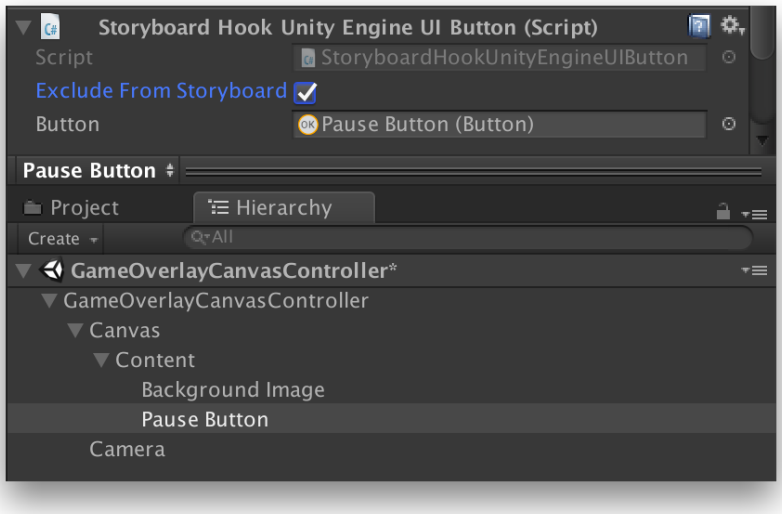


A node's canvas preview area displays its 'hookable' elements, such as the pause button shown here.

Note: The canvas preview area is sized according to the Canvas Scaler's reference resolution. See the canvas controller's canvas object for more information.

By default, all Unity UI Buttons are supported as *hookable* elements in a storyboard. This means that any `UI Buttons` in a canvas controller can be used in a storyboard to trigger a transition to another canvas controller. To support additional UI elements, such as a custom button type, please see the ***Custom Storyboard Hooks*** section of the manual.

When a canvas controller's scene is saved, all *hookable* components have a `StoryboardHook` added to them, causing them to be displayed in the canvas preview area. A storyboard hook can be excluded from a storyboard by selecting the `Exclude From Storyboard` flag on the hook in your canvas controller's scene.



Exclude a hook from a storyboard.

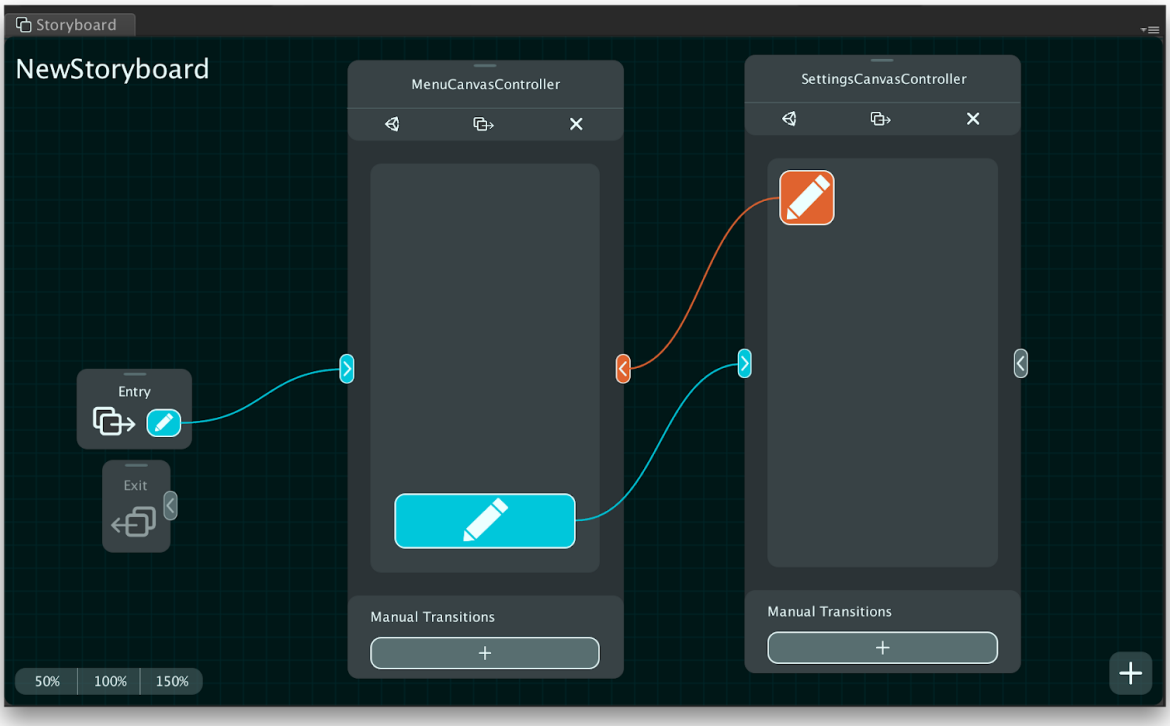
Additionally, the automatic adding of `StoryboardHook` components can be disabled in Canvas Flow's preferences by unchecking '*Auto-Add Hooks On Scene Save*', found in the menu bar at *Unity/Preferences/Canvas Flow.*

Storyboard Transitions

Storyboard transitions define the flow of nodes in the storyboard. A connection from one node to another represents a transition between those two nodes - a presentation or dismissal. The hook from which the transition originates defines the trigger for that transition.

Direction

All transitions in a storyboard have a direction, which can be one of two values - downstream or upstream. A downstream connection is equivalent to a presentation and is represented by a blue color. An upstream connection is equivalent to a dismissal and is represented by an orange color.



A downstream transition (presentation) from the menu to the settings screen and an upstream transition (dismissal) from the settings screen back to the menu.

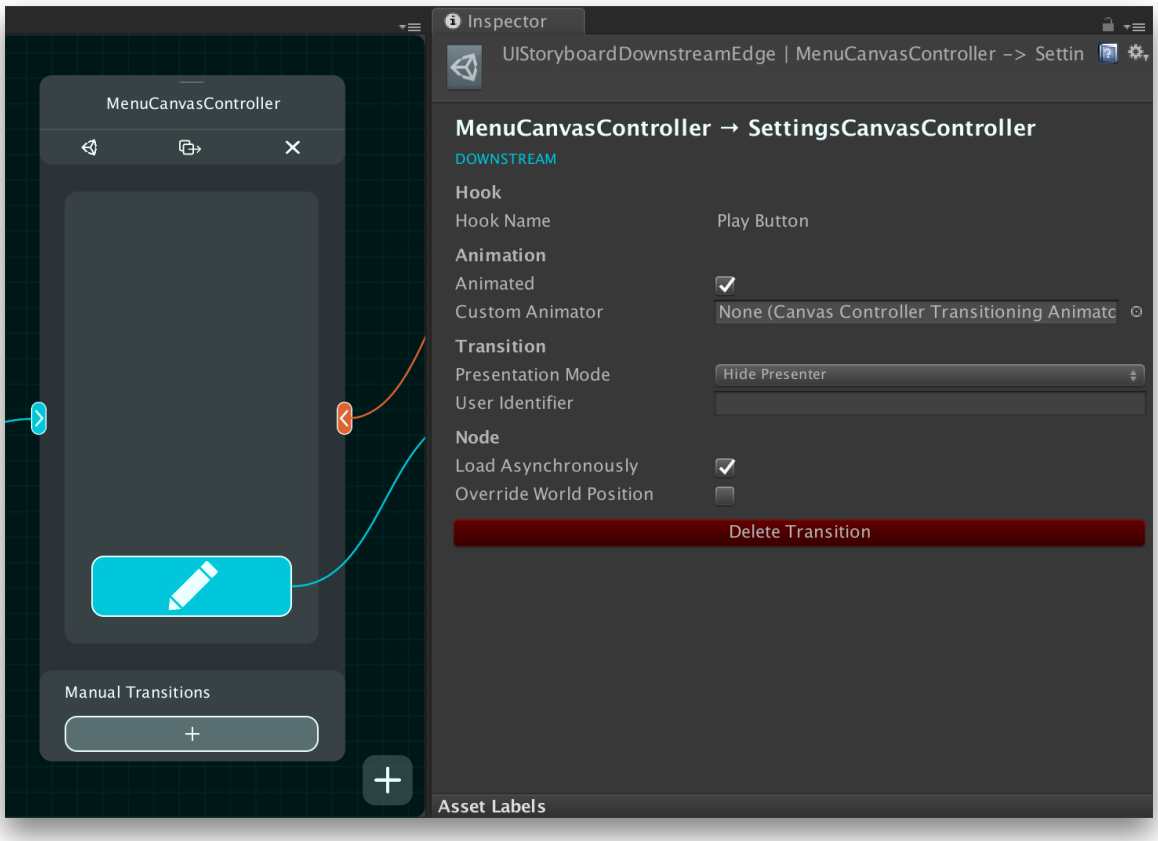
Creation

To create a transition between two nodes, drag from the source node's hook to the destination node. Canvas Flow automatically detects the direction - downstream or upstream - based on the current state of the flow graph.

Note: An upstream transition, or dismissal, is created when a connection is made in which the destination node has a downstream path to the source node. Therefore, to create an upstream transition a downstream path between the nodes must exist.

Configuration

A transition's properties can be configured by selecting its origin hook. This will open the transition inspector, which is shown below and followed by a description of each property.



The transition inspector.

Property	Description
Hook Name	The name of the hook that will invoke this transition. This is taken from the game object that the hook component is attached to in the canvas controller.
Animated	Is the transition animated?

Custom Animator	The transition's animator. For more information, please see the transition animators section of the online manual at https://canvasflow.xyz/.
Presentation Mode	The presentation mode of the destination canvas controller (only available on downstream transitions). Please see the canvas controller's presentation mode in the online manual at https://canvasflow.xyz/ for more information.
User Identifier	The transition's user identifier. This is used to identify the transition when triggering it manually, such as with a ***manual transition***.
Load Asynchronously	Should the canvas controller be loaded asynchronously? By default, canvas controllers are always loaded asynchronously.
Override World Position	Specify a world position at which to place the loaded canvas controller. This can be specified as an absolute world position with the 'Position' setting, or as a spacing multiplier with the '*World Spacing Multiplier*' setting. If '*World Spacing Multiplier*' is selected, the world position for the new canvas will be the source canvas' world position plus the canvas size scaled by the multiplier.

Passing Data Between Canvas Controllers

When a transition occurs in a storyboard, the transition's source canvas controller will have its `PrepareForStoryboardTransition()` method called, passing in a `StoryboardTransition`.

This gives an opportunity to pass data between canvas controllers, just as you might use a configuration action when presenting a canvas controller from script.

For example, in Canvas Flow's Floaty Cube example, the menu canvas controller uses the `PrepareForStoryboardTransition()` method to detect when the loading screen is being presented and to configure it with the name of the scene to load, like so:

```
public override void
PrepareForStoryboardTransition(StoryboardTransition transition)
{
    var destination = transition.DestinationCanvasController();
    if (destination is FCLoadingCanvasController &&
        transition.direction ==
StoryboardTransitionDirection.Downstream)
    {
        // We are presenting the loading screen. Configure it to
present the Game scene.
        var loadingCanvasController =
(FCLoadingCanvasController)destination;
        loadingCanvasController.SceneToLoad = "FCGameScene";
    }
}
```

Manual Transitions

Manual transitions are transitions that have no origin hook - i.e. they must be triggered manually. These can be used to invoke a storyboard transition in response to non-UI events, such as a collider being intersected or a timer expiring.

To create a manual transition, drag from the source node's '*Manual Transitions*' element to the destination node and enter a user identifier.

To trigger a manual transition, call `PerformTransitionWithIdentifier()` from the source canvas controller, passing in the transition's `userIdentifier`.

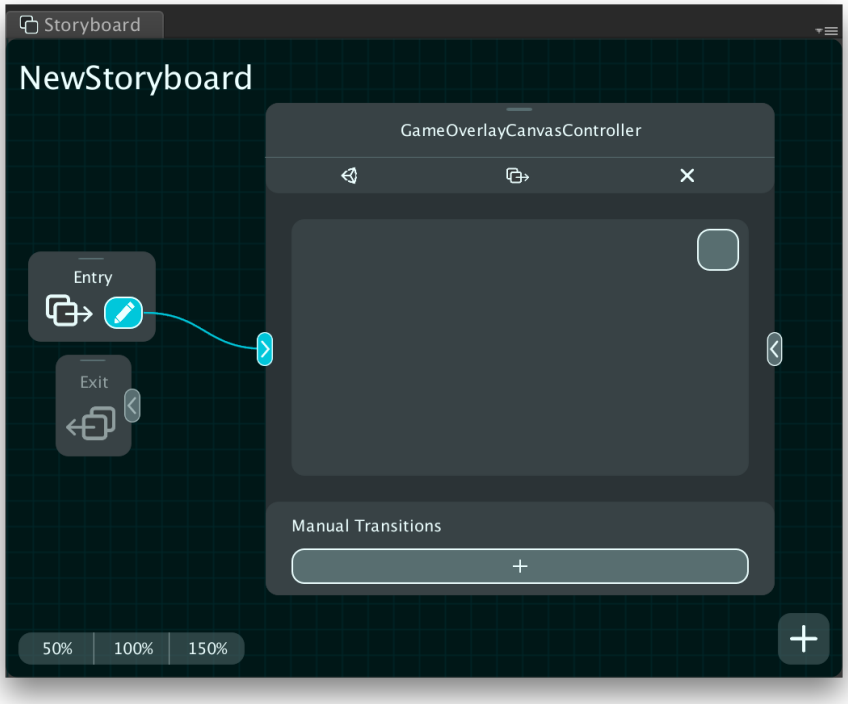
For example, in Canvas Flow's Floaty Cube example, an event is raised when the player reaches a target height. The game overlay screen subscribes to this event, and in the callback it triggers its manual transition named "*PresentGameCompleteScreen*", like so:

```
public void OnPlayerReachedHeight()
{
    PerformTransitionWithIdentifier("PresentGameCompleteScreen");
}
```

Note: Any storyboard transition - not just manual transitions - can be triggered by passing its *User Identifier* to `PerformTransitionWithIdentifier()`.

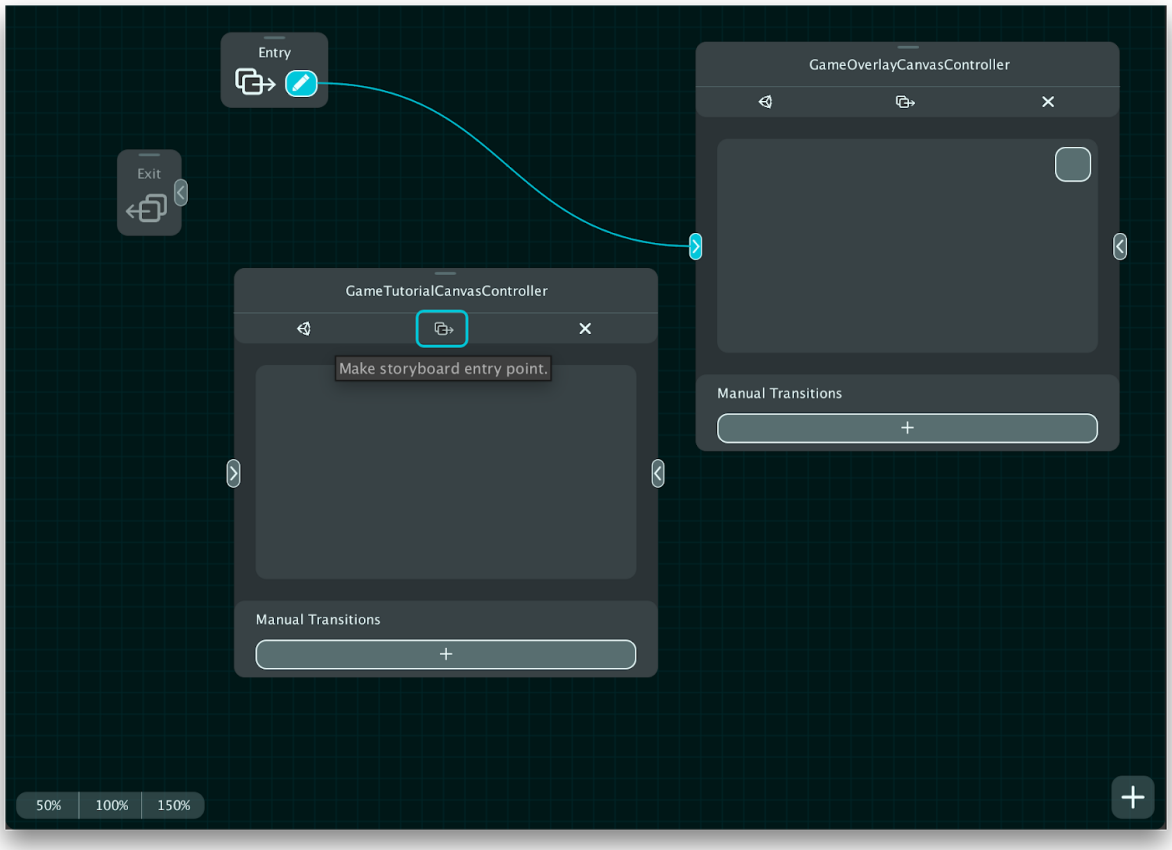
Entry Transition

When the first canvas controller is added to a storyboard, an entry transition pointing to the new canvas controller is automatically created. This is shown by the blue wire connecting the 'Entry' node to the newly-added canvas controller.



The storyboard's entry transition.

The storyboard's entry transition determines which canvas controller is presented when the storyboard is presented - i.e. the storyboard's entry point. To change the storyboard's entry point, select the '*Make storyboard entry point*' button on the canvas controller's node, as shown below.



Set a storyboard's entry point.

Note: A node can only become the entry point if it has no downstream transitions pointing to it. Similarly, if a node is the entry point, a downstream connection cannot be made to it.

The entry transition can be configured just like any other transition, as described above, with the exception that it cannot be deleted; a storyboard must have an entry point.

Exit Transition

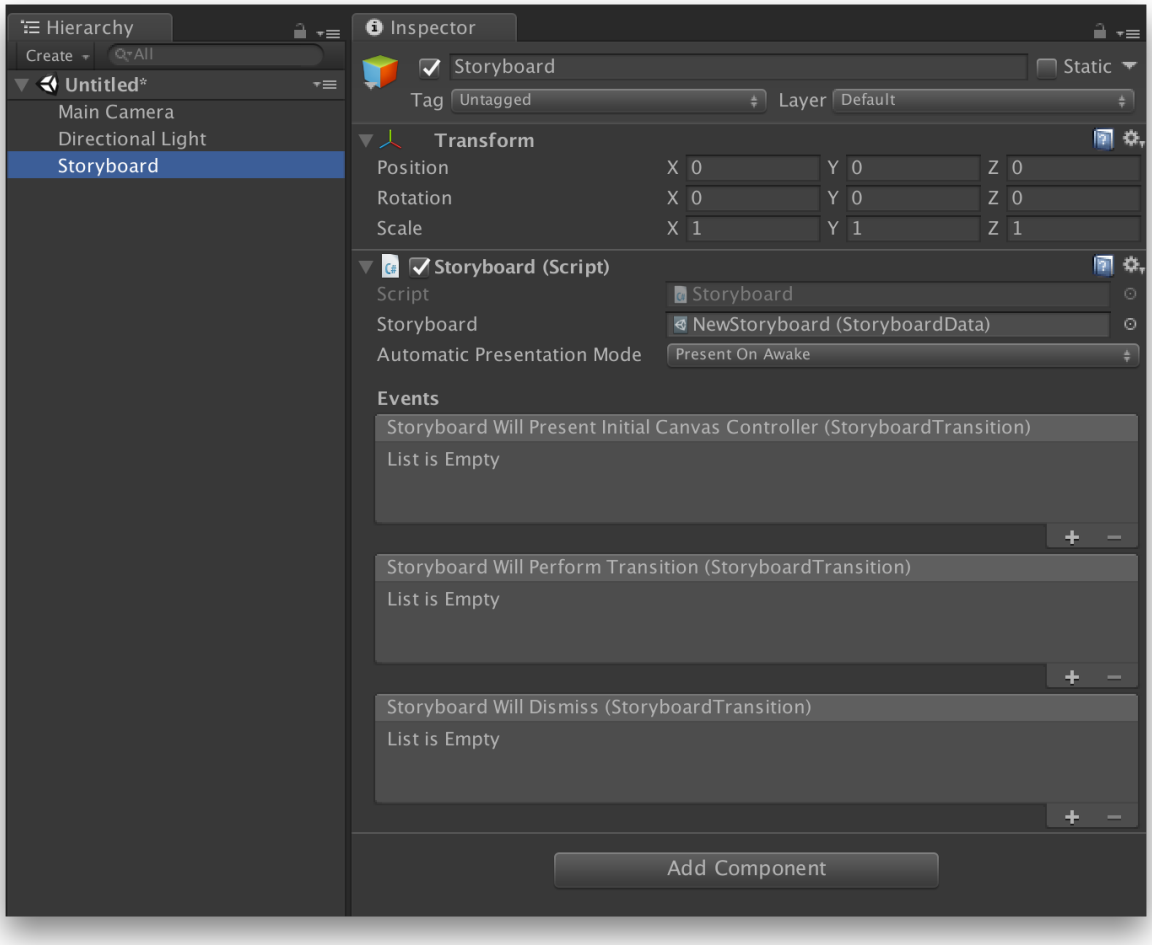
A storyboard exit transition is optional and is not always required. It is used in the event that you want to dismiss the entire storyboard. It will dismiss all canvas controllers that are in the storyboard's initial canvas controller's presentation hierarchy, including the initial canvas controller itself. It is the equivalent of calling `DismissAllCanvasControllers`.

To create an exit transition, drag from a hook or manual transition element to the *Exit Transition* node.

Presenting A Storyboard

Storyboards are presented from a scene. This could be a game scene, a menu scene, or any other content scene, over which the user interface is presented.

To present a storyboard, create a new game object in the scene, add the Storyboard component, and select your storyboard asset in the *Storyboard* field, as shown below. Run the scene and the storyboard will be presented.

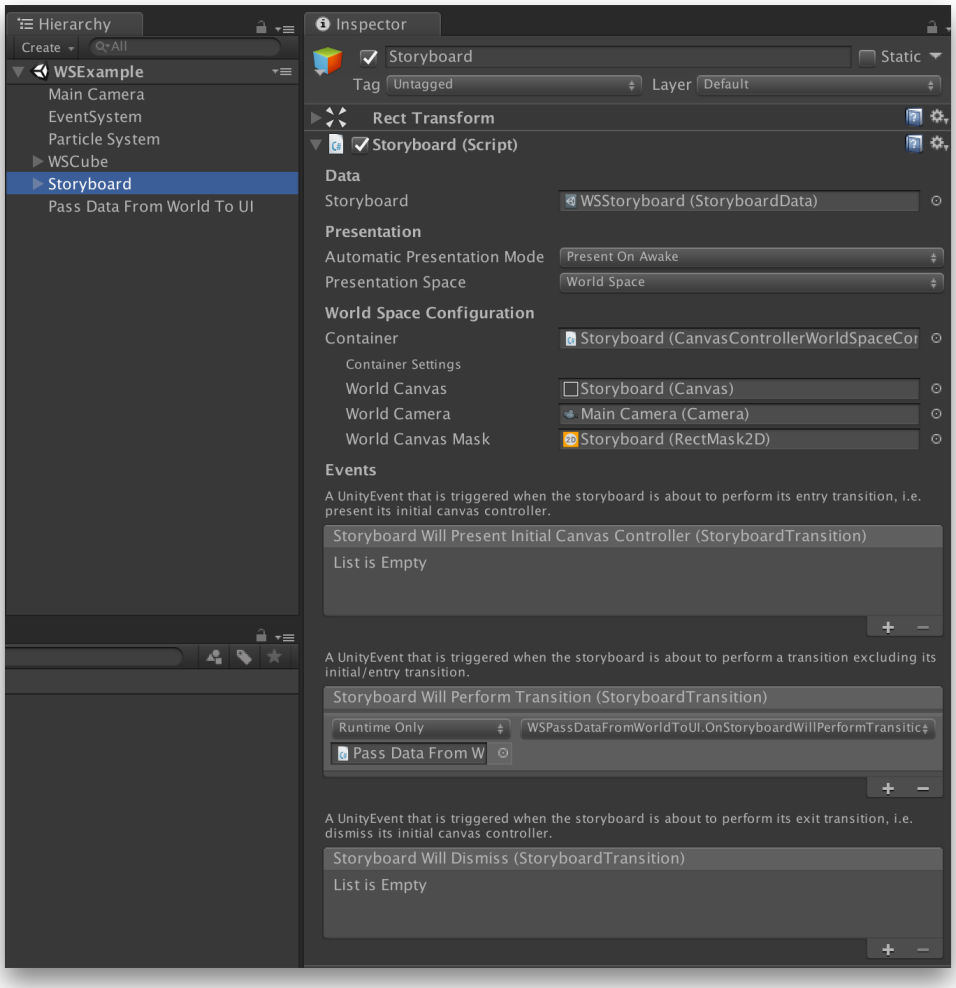


Presenting a storyboard from an empty scene.

Note: A single Unity EventSystem is required to be present for canvases to receive input events. Therefore, depending on your game's structure, you may wish to place an Event System (*Game Object/UI/Event System*) in the scene presenting your storyboard.

Presenting In World Space

By default, storyboards are presented in screen space. To present a storyboard in world space, simply select the storyboard component in the scene and change its presentation space to `World Space`. This will cause the storyboard to be presented in world space and embedded within the specified container.



Note that any subsequent storyboard transitions, as well as manual calls to `PresentCanvasController<T>()`, will automatically detect whether they should be presented in world space or screen space. If the presenting canvas controller is in world space, the presented canvas controller will also be in world space. Likewise, if the presenting canvas controller is in screen space, the presented canvas controller will also be in screen space.

Callbacks

As shown above, three UnityEvent callbacks are available on a Storyboard component.

These callbacks can be particularly useful for passing scene data to the user interface, such as references to scene objects. For example, Canvas Flow's Floaty Cube example uses the `StoryboardWillPresentInitialCanvasController` callback to pass a reference to the player object to the initial game overlay screen. This allows the game overlay screen to display the player's current height. It is implemented like so:

```
public void
OnGameStoryboardWillPresentInitialCanvasController(StoryboardTransiti
on transition)
{
    var gameOverlayCanvasController =

transition.DestinationCanvasController<FCGameOverlayCanvasController>
();
    gameOverlayCanvasController.ConfigureWithPlayer(player);
}
```

Note: The callbacks pass a `StoryboardTransition` object, allowing you to access the canvas controllers involved.

For more information, please see the Storyboard component's documentation in the online manual at https://canvasflow.xyz/.

Custom Storyboard Hooks

Custom Storyboard Hooks allow you to use your own button types in storyboards. They will show up in the canvas preview area as *hookable* elements, just as native UI Buttons do.

See also: The online tutorial Creating a Custom Storyboard Hook For UI.Toggle.

Overview

A storyboard hook is a component that is attached to game objects in your canvas controller's scene. Any game object in the canvas controller's scene with a `StoryboardHook` component will become *hookable* in a storyboard.

You do not use the `StoryboardHook` component directly, however. Instead, a subclass is created that adds support for a single target component type - the component that you wish to make *hookable*. It is this `StoryboardHook` subclass that is then added to any game objects with the target component type. Additionally, the process of adding `StoryboardHook` subclasses to their relevant game objects can be performed automatically for you by Canvas Flow, each time a canvas controller's scene is saved.

For example, Canvas Flow includes one such `StoryboardHook` subclass - `StoryboardHookUnityEngineUIButton` - which is a storyboard hook for native UI Button components. It can be (and is automatically) added to any game object with a `UI.Button` component in order to make it hookable in a storyboard.

In order to make your own buttons *hookable* in a storyboard, you must create a similar `StoryboardHook` subclass for your custom button type, as documented below.

Note: Canvas Flow's default storyboard hook for native UI Button components is included as a .cs file. This means you can easily inspect the source code, which is recommended when creating your first custom storyboard hook. It can be found in the Project inspector at *Canvas Flow/Storyboard Hooks/StoryboardHookUnityEngineUIButton.cs*.

1. Create A New Storyboard Hook Subclass

To begin creating a new storyboard hook, open Unity's Create menu - by either right-clicking in the Project window or selecting Assets in the menu bar - and selecting *Create/Canvas Flow/Storyboard Hook*. Enter a name and a directory for your new hook. This will generate a new `StoryboardHook` subclass at the specified location.

The generated script contains three methods for you to implement.

- Connect()
- Reset()
- AutoAddComponentType

2. Implement The Connect Method

When a canvas controller is loaded from a storyboard at runtime, its connected hooks will be provided with a callback via the `Connect()` method. Your hook's responsibility is to simply invoke this callback when it deems itself to have been triggered. Invoking the provided callback is all that is required to trigger a storyboard transition.

So, in the case of the `StoryboardHookUnityEngineUIButton`, the callback passed to the `Connect()` method is attached to the button's onClick handler. This means that when the button is clicked, the callback will be invoked, triggering the storyboard transition.

The source code for the `StoryboardHookUnityEngineUIButton`'s `Connect()` method is therefore:

```
public override void Connect(System.Action<StoryboardHook>
invokeTransition)
{
    // For a Unity UI Button we invoke our storyboard transition when
    // our Button is clicked.
    button.onClick.AddListener(() => {
        invokeTransition(this);
    });
}
```

You may wish to do some additional logic here. For example, if we were making a hook to be used with a UI Toggle switch, we might only wish to trigger the transition when the toggle is switched on, like so:

```
public override void Connect(System.Action<StoryboardHook>
invokeTransition)
{
    toggle.onValueChanged.AddListener(() =>
    {
        // Only invoke the transition if the toggle is switched on.
        if (toggle.isOn)
        {
            invokeTransition(this);
```

```
        }
    });
}
```

Note: You can pass your hook to the callback with `this`, as above. This causes it to be included in the subsequent `StoryboardTransition` object that is created and passed to methods such as `PrepareForStoryboardTransition`.

3. Implement The Reset Method

The `Reset()` method is called on your hook in the Unity Editor when the component is added to a game object or the user resets it in the inspector. A hook can be automatically added to a game object when the scene is saved or manually added in the inspector. In either case, the `Reset()` method is where you must do any configuration you require.

In the case of the `StoryboardHookUnityEngineUIButton`, the `Reset()` method obtains the required reference to the `Button` component that is used in the `Connect()` method above, like so:

```
protected override void Reset()
{
    // Always call base.Reset() when overriding Reset() in a custom
hook.
    base.Reset();

    // Store a reference to the Button component.
    button = GetComponent<UnityEngine.UI.Button>();
}
```

You might wish to follow a similar pattern, storing a reference to your custom button type that can later be used in the `Connect()` method.

4. Implement The Auto-Add Component Type Property (Optional)

When a canvas controller scene is saved in the Editor, Canvas Flow will ensure that storyboard hooks are present on any game objects with the hook's 'auto-add' component type. Here you can return the target type of your hook, such as your custom button type, to have your hook automatically added to any game objects with the target component type.

For example, Canvas Flow's default storyboard hook for native UI Button components returns the `UI.Button` component here, causing it to automatically be added to any game objects with a `UI.Button` component.

```
public override System.Type AutoAddComponentType
{
    get
    {
        // Automatically add this hook to game objects with a Unity
UI
        // Button component when the scene is saved.
        return typeof(UnityEngine.UI.Button);
    }
}
```

Note: This is an optional step. You may return `null` here if you do not want your custom storyboard hook to automatically be added to the target component type when the scene is saved. Additionally, you can turn off *auto-add* for all storyboard hooks in Canvas Flow's preferences, found in the menu bar at *Unity/Preferences/Canvas Flow*.