Gunpoint Crosslink Deconstruction

Evan Daveikis 991 721 245

What does it do? What problem(s) in the game does it solve?

As the blog states, the crosslink system allows players to rewire elements of the level to open/close doors, turn lights on and off, etc. It allows players to solve levels in unique, creative ways. Examples of problems it solves could be “how should/do I open this door?” or “how can I beat this level?” or “what makes this game unique?”.

What Unity ingredients are required to do those things? What components and GameObjects are needed?

There is a GameObject for each different electrical element, i.e. an object for each light switch, door, hand scanner, light, etc. There are also objects/components for the rendering of the connections, i.e. LineRenderers. There are then specific components for elements such as colliders, lights, etc. where needed.

What’s the C# recipe: what’s the functionality of the script components? Write the pseudocode.

I envision two basic scripts necessary for this system:

```
abstract class ElectricalComponent
	var connection
	// Called when we receive an electrical input
	abstract fn HandlePulse(sender)

abstract class InteractableElectricalComponent : ElectricalComponent, IInteractable
	abstract fn OnInteract()

class LightSwitch : InteractableElectricalComponent
	override fn HandlePulse(sender)
		// Maybe toggle, or just do nothing
	override fn OnInteract()
		// When clicked, send a pulse to all other electrical components
		// TODO: Maybe some way to make this common behaviour easier?
		connection.SendPulse()

class Door : ElectricalComponent
	var open
	override fn HandlePulse(sender)
		open = !open
// etc for other classes
```

```
class ElectricalConnection : IDragStart, IDragEnd, IDragDrop
	static var currentlyDragging
	var connections
	var electricalComponent
	var lineRenderer

	fn DragStart() -> currentlyDragging = this
	fn DragEnd() -> currentlyDragging = null
	// Connect the dragged connection to us
	fn DragDrop()
		if currentlyDragging another connection
			AddConnection(currentlyDragging)

	// Make both of us reference each other
	fn AddConnection(otherConnection)
		connections.add(otherConnection)
		otherConnection.connections.add(this)
		UpdateLineRenderer() // Not implemented for brevity
	fn RemoveConnection(otherConnection)
		connections.remove(otherConnection)
		otherConnection.connections.remove(this)
		UpdateLineRenderer() // Not implemented for brevity

	fn SendPulse(originalSender = null by default)
		foreach connection in connections
			if connection isnt the originalSender
				connection.HandlePulse(this)
	fn HandlePulse(sender)
		electricalComponent.HandlePulse(sender)
		// Continue sending pulse along network
		// TODO: Avoid infinite loop if there are loops in the network
		SendPulse(sender)
```

Which bits of this system can be achieved by composition?

Elements would naturally be composed of several built-in Unity components for basic functionality: colliders and Rigidbodies to detect being clicked on, SpriteRenderers and LineRenderers for graphics, etc. The electrical components would have elements of composition as well, with functionality like the connection and the component itself being split into different scripts.

Which bits of this system can be achieved by inheritance? Draw the inheritance "tree" showing parent & child classes and where the functionality goes.

The electrical components would all derive from a base class providing functionality for receiving an electrical input, and another commonly used base class would be one that detects players/enemies interacting with it as well. The connections themselves could use a form of inheritance as well – for example, there could be a connection that cannot be dragged/changed until some condition is met.