Goals

1. Spawning an invention is not destructive to existing content in a room.
2. Spawning an invention is not destructive to the invention.
    - If the invention contains a definition or reference, the definition or reference cannot be changed in a way that changes the inventions functionality or how it interacts with other inventions.
3. Inventions with defining entities can bring those definitions into other rooms.
4. Inventions with referencing entities can bring those references into other rooms.
5. Rules for all kinds of spawning (invention, clone, maker pen) are the same.
6. Enable future proposals that depend on scoping.

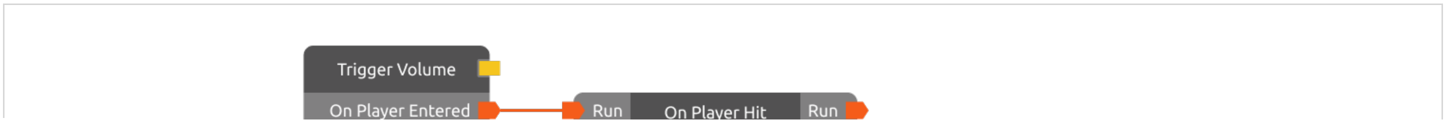
Design

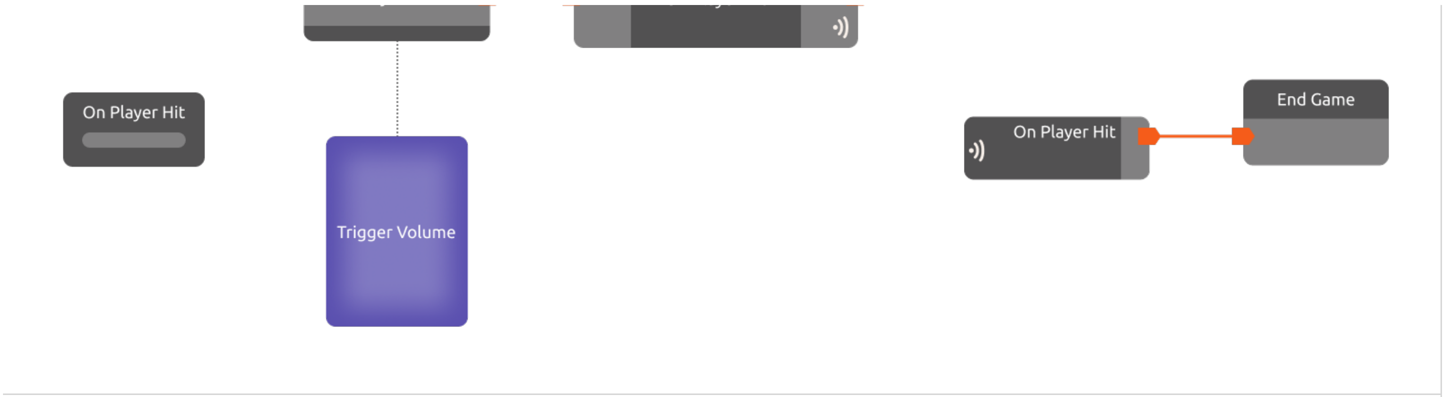
Terms

- **Guids** are universally unique ids. No other id on any computer anywhere in the universe is the same.
- **Defined entities** provide reusable logic or data to other gameplay systems.
    - Some examples are functions, events, and variables.
- A **binding** is the point-of-definition for logic or data.
    - An example of a binding is an event definition.
- An **instance** is the point-of-use for a binding.
    - Some examples of instances are event senders, event receivers.
- A **binding instance** is an instance that is also a binding.
    - An example of a binding instance is a variable.
- **Orphaning** is when an instance is spawned that references a binding that doesn't exist.
- **Ambiguity** is when two bindings exist with the same key and are accessible from the same scope.

Problem

Let's build some reusable killzones. If anyone touches the killzone, it ends the game.

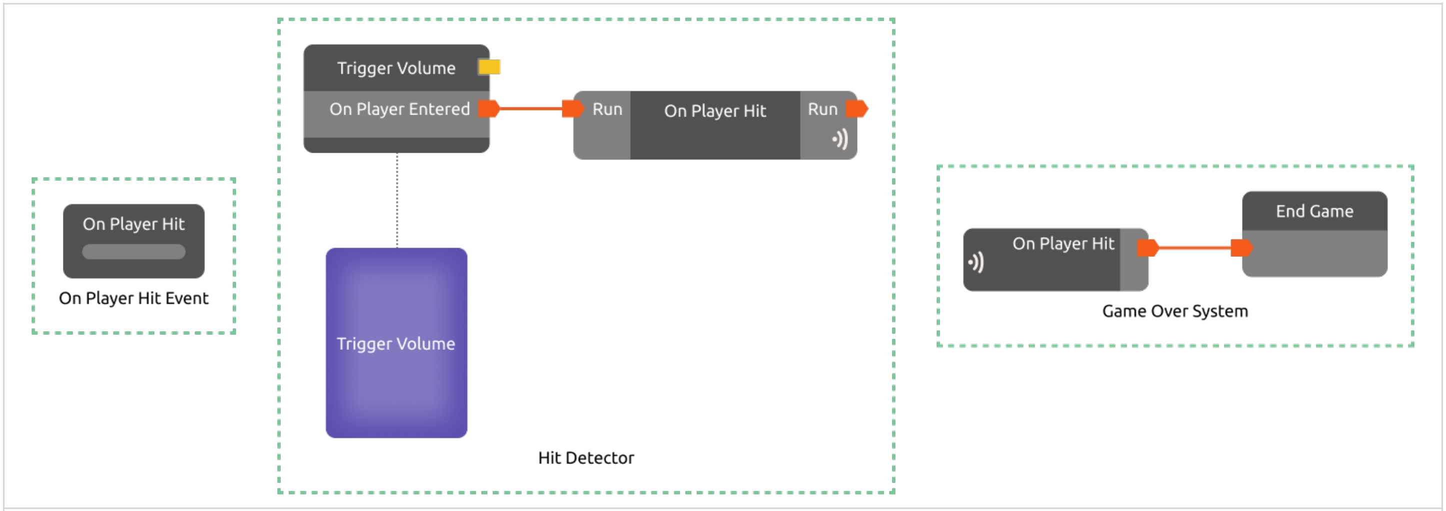




Game Setup

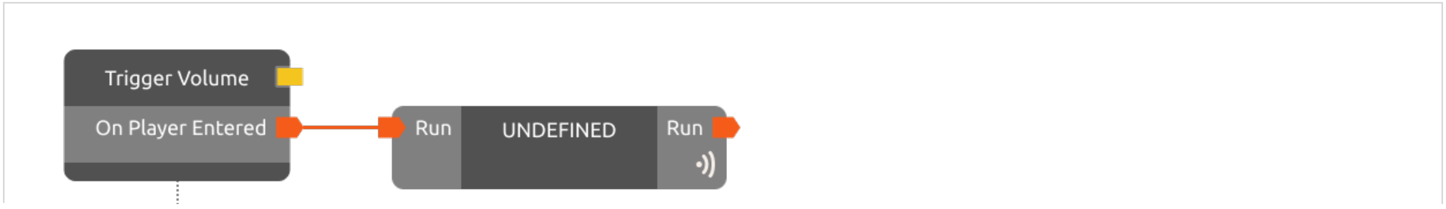
Our setup is pretty nice. We have an event which is triggered when a player is hit and we can create as many hit detectors as we want. Our game becomes critically acclaimed and other creators beg us to share our secrets. After some coaxing we decide to turn our game into a collection of reusable inventions:

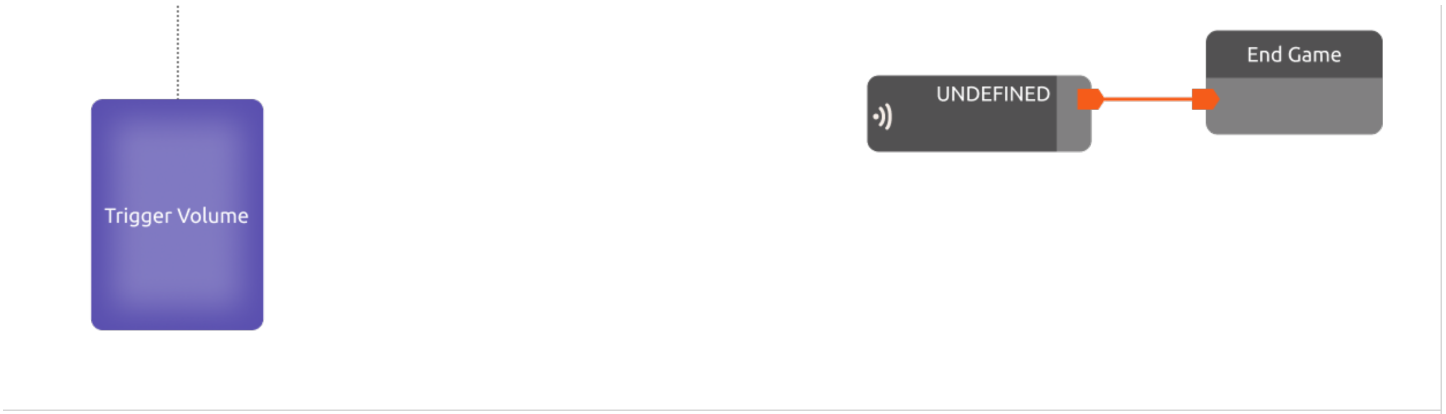
- An **On Player Hit Event** invention.
- An **Hit Detector** invention which uses **On Player Hit Event**.
- An **Game Over** invention which uses **On Player Hit Event**.



Inventions

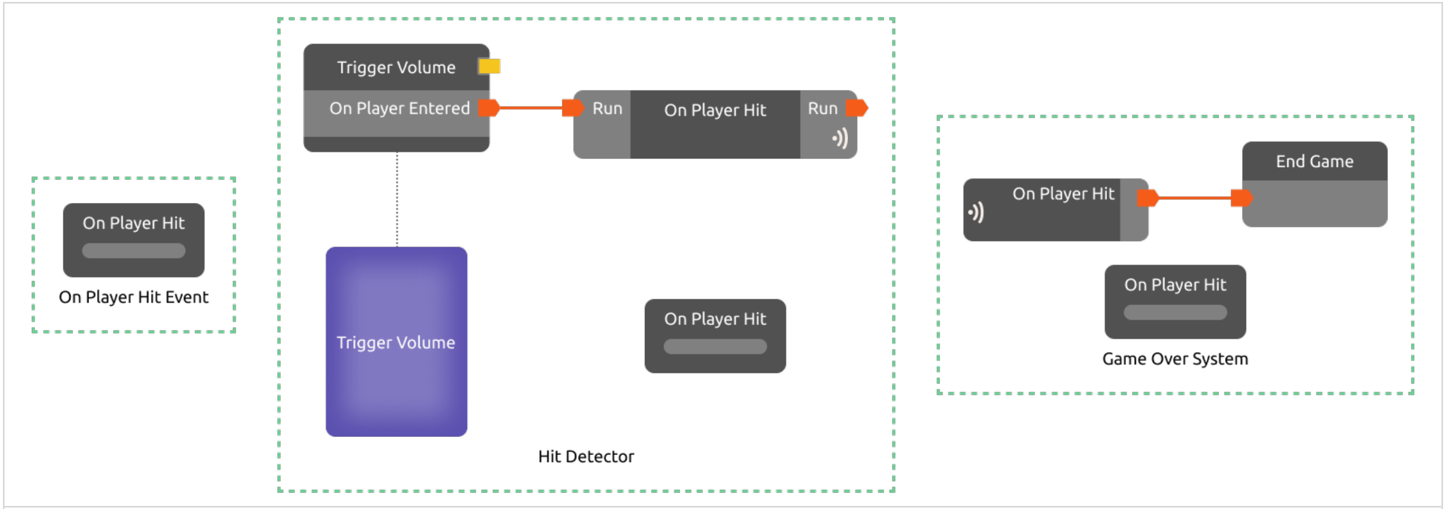
Creators start using our inventions. They download the hit detectors and put them in their rooms. However, some creators run into a problem: the hit detectors and game over system don't seem to be working.





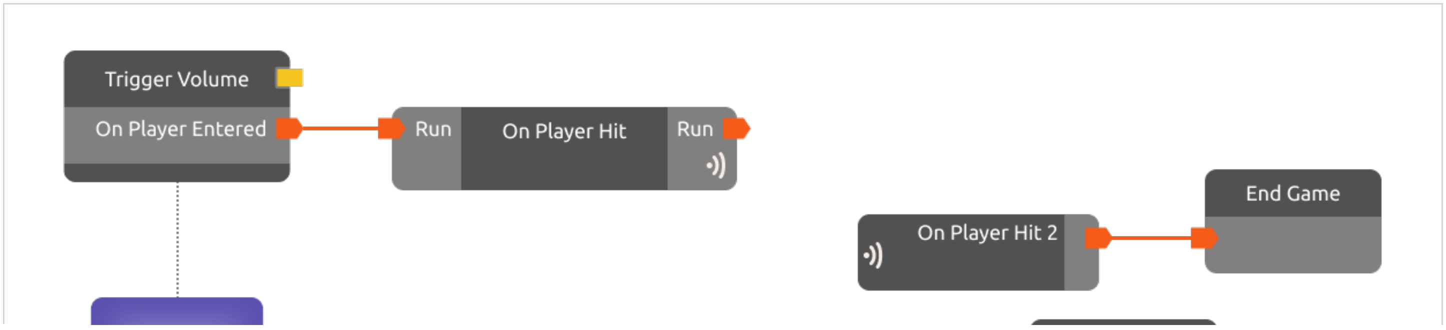
Orphaning

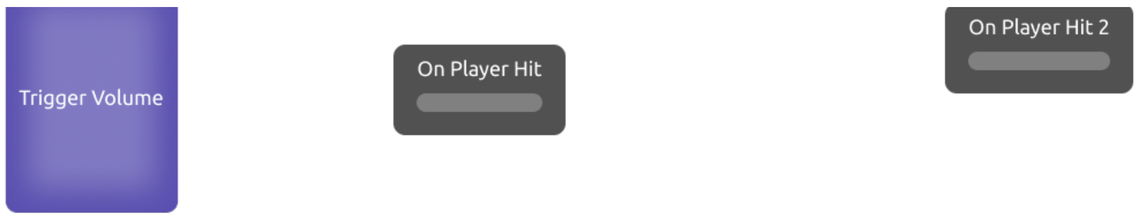
It seems that creators don't understand that they need to spawn the **On Player Hit Event** if they want to use the **Hit Detector** or **Game Over** inventions. When creators spawn inventions which use the event, without having the event defined, event senders and receivers cannot find it. This is called orphaning. Okay, lets go back to the drawing board. Perhaps there is a better way to package the inventions to avoid this problem. Maybe we can include the **On Player Hit Event** in the other inventions.



Inventions Improved

Now when creators spawn a **Hit Detector** or **Game Over System** they are guaranteed to have the event. But when we release our new invention, we still have a problems.





Ambiguity

Having multiple events in the same scope causes conflicts. We call this problem ambiguity. The inventions can't work together because every spawn causes another conflict.

Isn't there a better way?

A Better Way

Lets define the rules for event definitions before generalizing them for all entities in CV2. The specific example of event definitions is easier to understand than the general rules which are more abstract.

1. Every event definition gets a GUID
2. Every event receiver/event sender references the GUID from an event definition
3. If an event receiver/event sender is cloned while referencing an event definition, it will only work with the same event definition when spawned
    - If the event receiver/event sender is in an invention, it will not work if it is spawned in a room without the corresponding event definition
4. If two event definitions with the same GUID are spawned, they are the same event definition
    - The definitions have the same GUID and backend data
    - There is only one copy of the backend data and only one event available to users
    - The definition chips function as ref-counted pointers to the event data, the data is deleted when all of the definition chips are deleted
5. An event definition cannot be spawned in any level of hierarchy which already has the same event definition in an ancestor or descendant unless it is local only

With some clever replacement of words, we can rewrite the rules above in a general way that works with all objects.

1. Every binding gets a GUID
2. Every instance references the GUID provided by a binding
3. If an instance is cloned while referencing a binding, the clone will reference the same binding
    - If the instance is in an invention, it will not work if it is spawned in a room without the

corresponding binding

4. If two bindings with the same GUID are spawned in the same scope, they are the same binding
    - The bindings have the same GUID and backend data
    - There is only one copy of the backend data and only one binding available to users
    - The bindings as ref-counted pointers to the data, the data is deleted when all of the bindings are deleted
5. A binding cannot be spawned in any level of hierarchy which already has the same binding in an ancestor or descendant

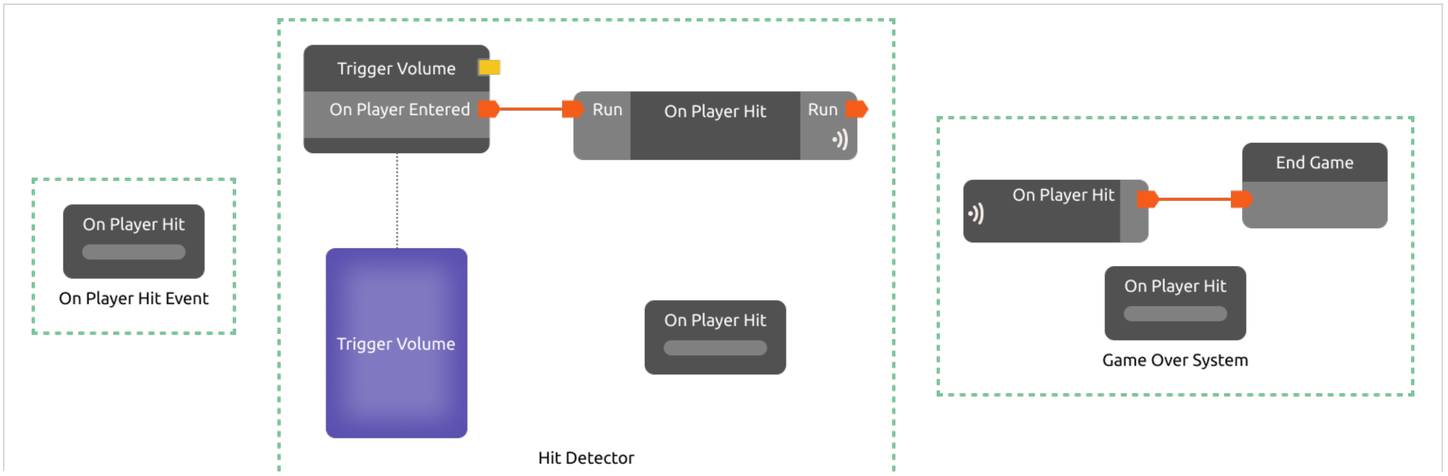
Name Resolution

With the rules above it is possible to spawn two entities in the same scope, with the same name, but different GUIDs. While this is non-destructive, it doesn't provide a great UX. Many of our menus display names but if multiple objects have the same name they are difficult to distinguish. The following rules define how objects are renamed to ease this ambiguity.

1. Name resolution happens if two bindings with the same name but different GUIDs are spawned in the same scope.
2. The name of the spawned binding will automatically have a number appended to it if it conflicts with an existing name.
    - Ex: A "Hit Detector" invention with an "On Player Hit" event definition may have the definition changed to "On Player Hit 2".
3. Binding names are configurable no matter what the invention permissions are.

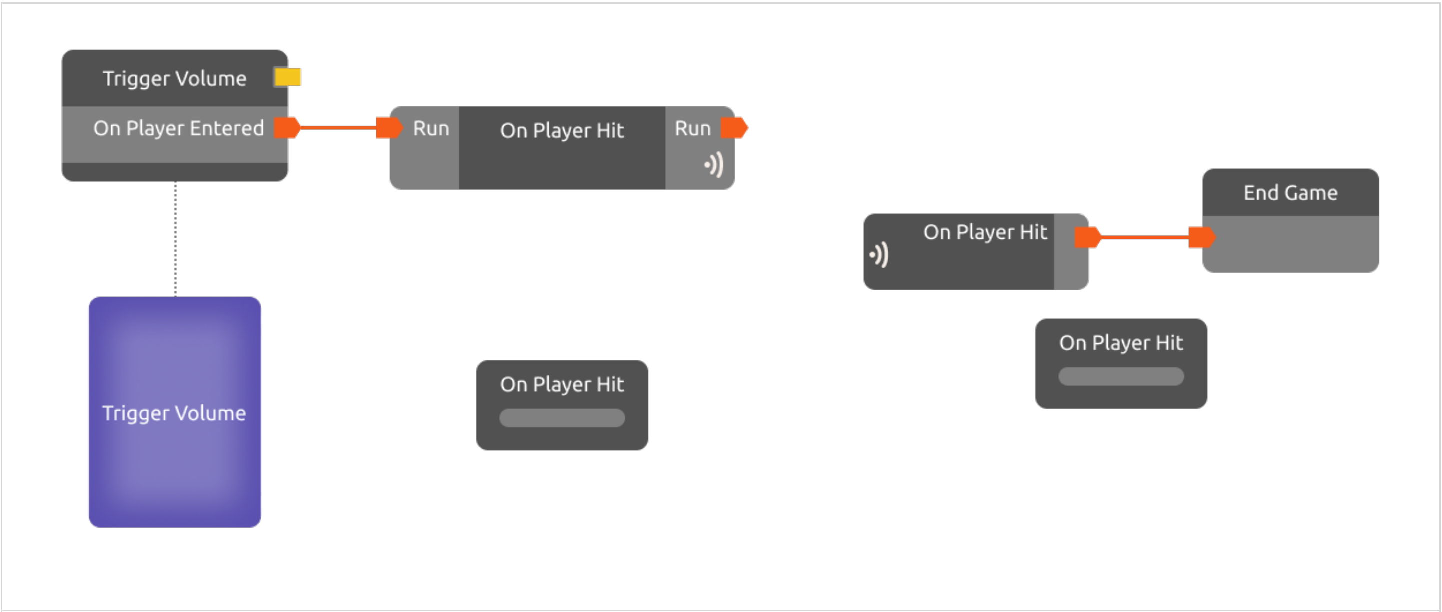
Scenarios

Now that we have these rules in place, lets try building our inventions again. We will start with the latest version of our inventions.



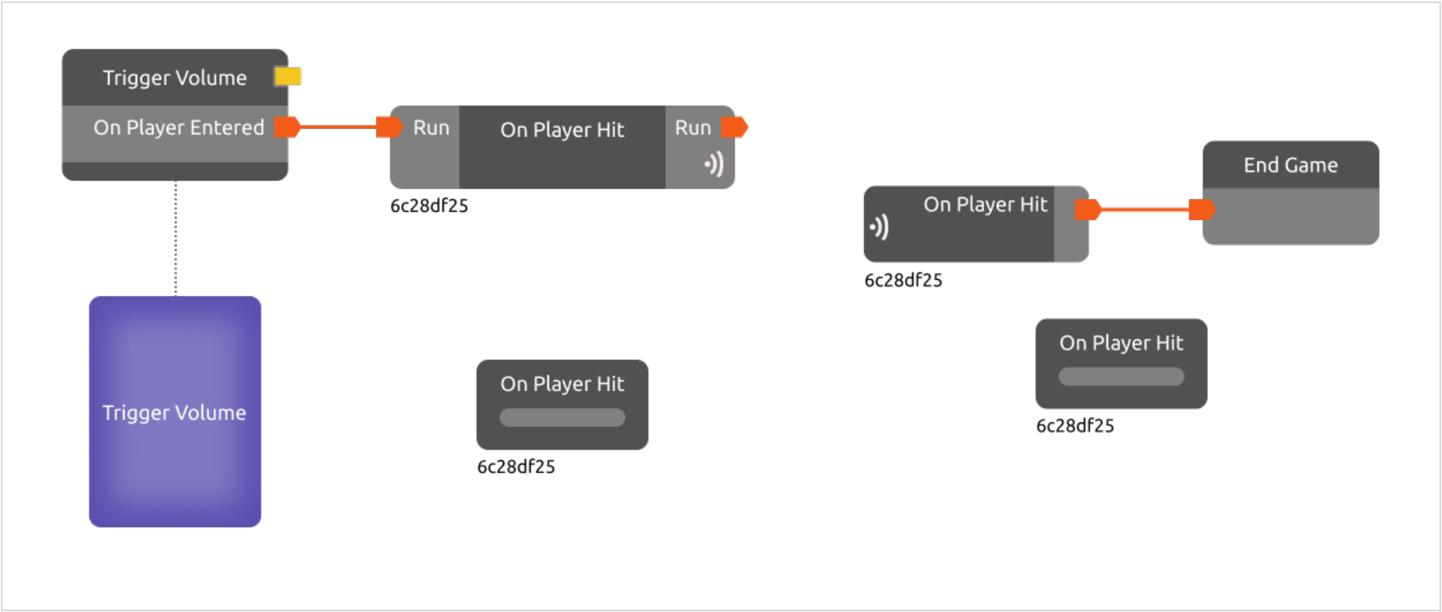
Inventions Improved

After sending these back out to the public we are lauded for their simplicity. They don't seem to break down at all and can happily be together without causing confusion.



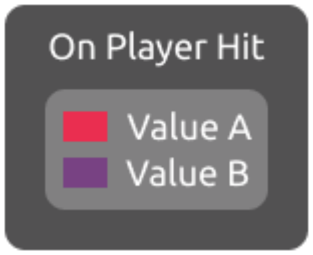
Successful Usage

This works because of rule 4: **if two bindings with the same GUID are spawned, they are the same binding**. If we superimpose the GUIDs over our image we can see that every event is linked up to the save definition because they all share a GUID, even if there are multiple definition chips in the room.



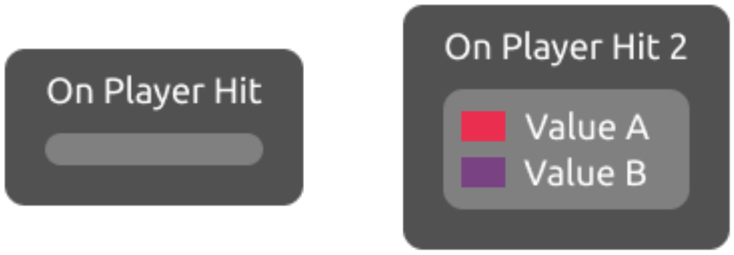
Successful Usage with GUIDs

The GUIDs here do some heavy lifting and help resolve more problems with inventions than just the ones we experienced above. For example, suppose someone else creates an "On Player Hit" invention.



Other On Player Hit Event

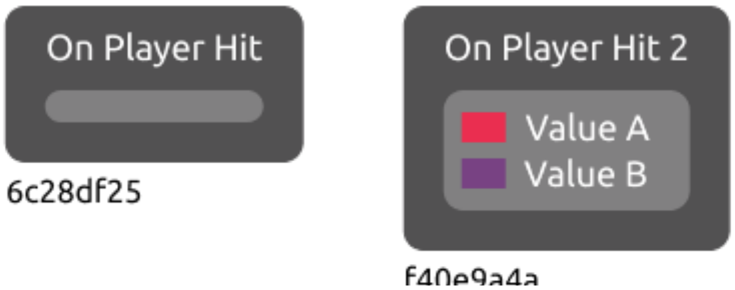
Lets spawn this in a room with our other event.

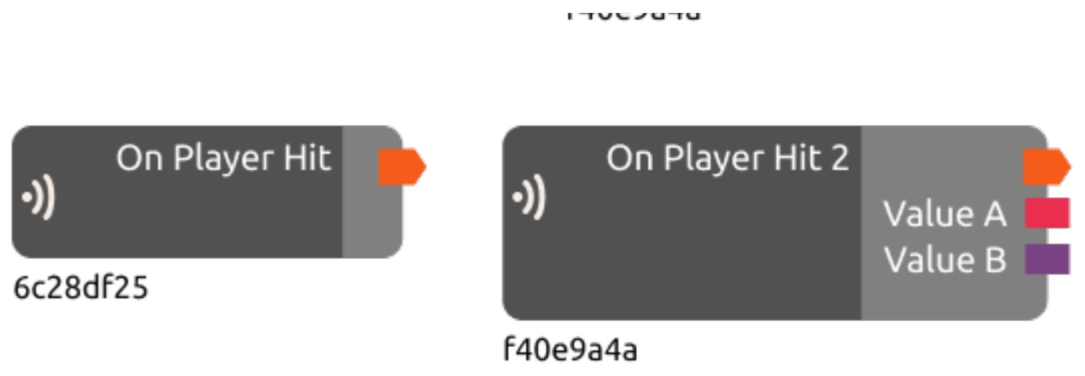


Both Events

Because of name resolution, our new event's name must be changed. Names must be unique per scope: **name resolution happens if two bindings with the same name but different GUIDs are spawned in the same scope**. We change the name by appending a number: **the name of the spawned binding will automatically have a number appended to it if it conflicts with an existing name**.

Previously, this change in name would prevent inventions with event senders and receivers from working with the renamed event. Now, each event has a unique GUID. Event senders and receivers only depend on the GUID so they can still find the definition if they are spawned in a room with it, even though the name has changed.

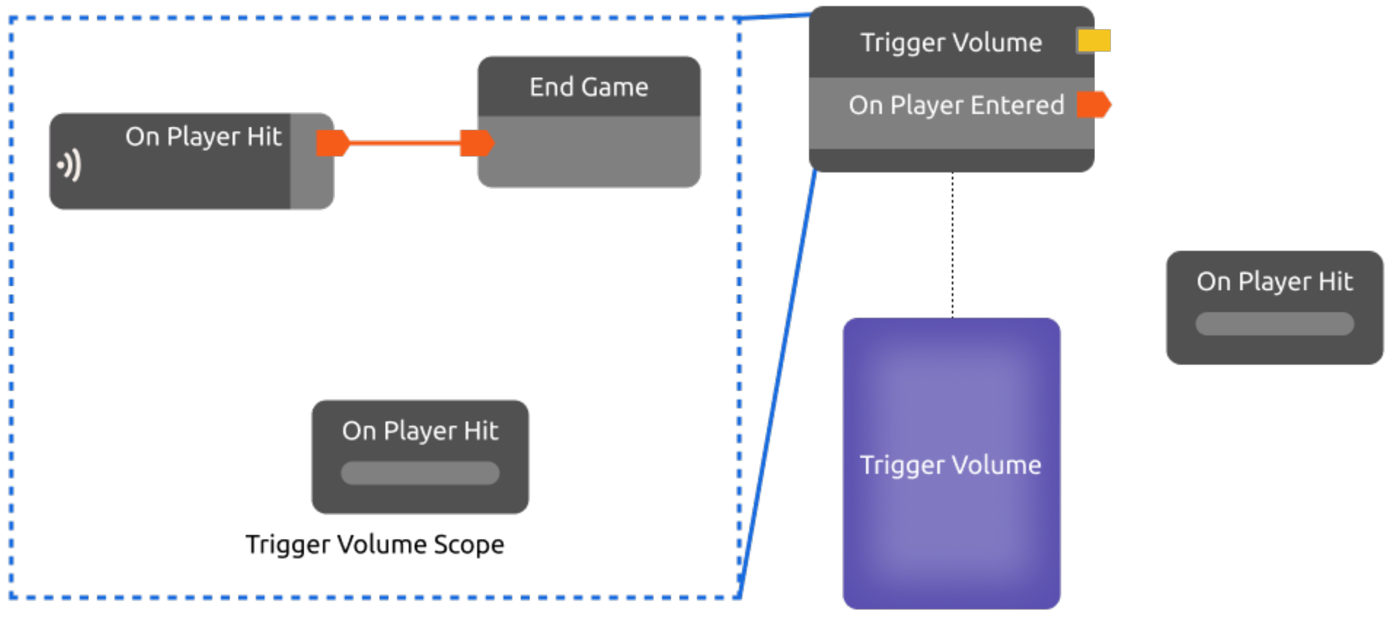




Both Events with GUIDs

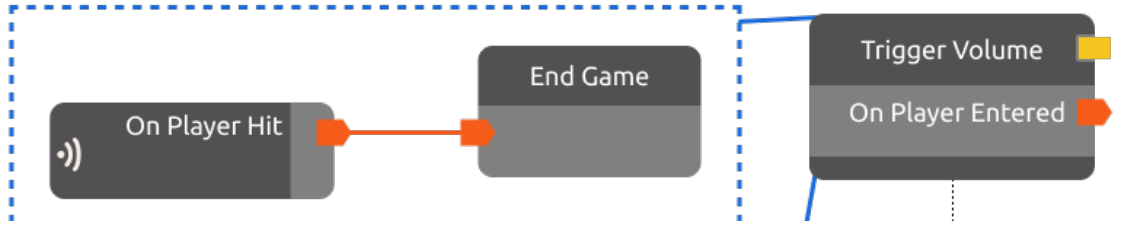
Nested Scopes

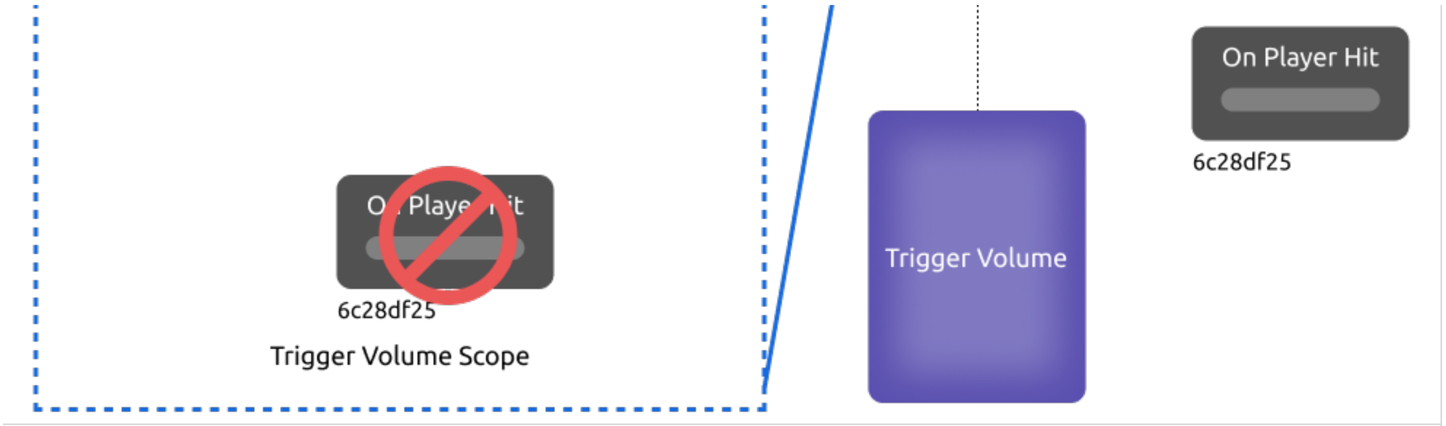
Suppose we wanted to move the **Game Over** invention to within the **Hit Detectors** control panel.



Hit Detector with Game Over

We may initially try to build something like the image, above, but we'd find that the current set of rules don't allow it: **A binding cannot be spawned in any level of hierarchy which already has the same binding in an ancestor or descendant**. Since the On Player Hit event has a matching GUID in both the Trigger Volume scope and at room scope, the entire invention cannot spawn.



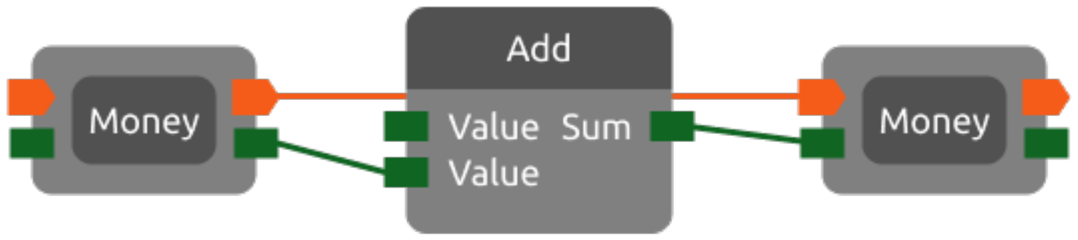


Game Over System Spawn Failure

By preventing the same definition in multiple scopes, we avoid ambiguity. We make it clear to creators that a definition chip implies the scope of definition and spawning additional inventions in different scopes in the future has no impact on the visibility of the definition.

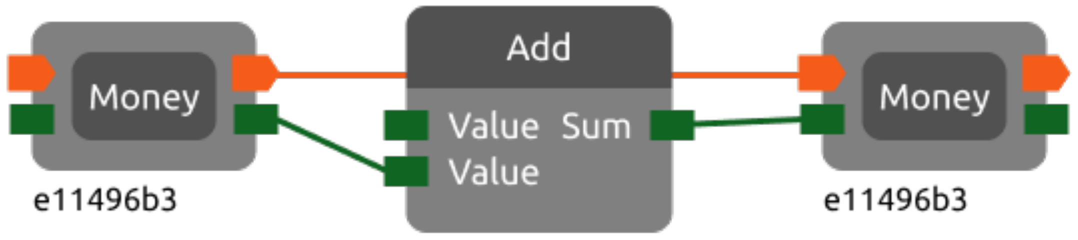
Binding Instances

Lets take a look at some examples that use variables now. We will build a simple system that adds money to a player.



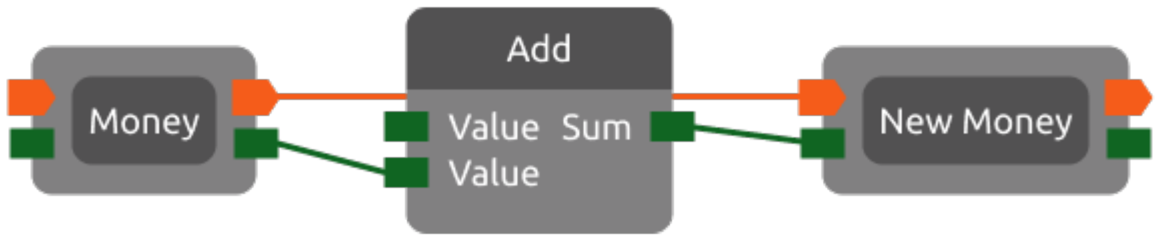
Add Money

In this case the variables are binding instances rather than just bindings or instances: they provide a definition for data but they are also the point of use for the data. Since these variables have the same name, and they are in the same scope, they must be the same variable. Indeed, they share the same id.



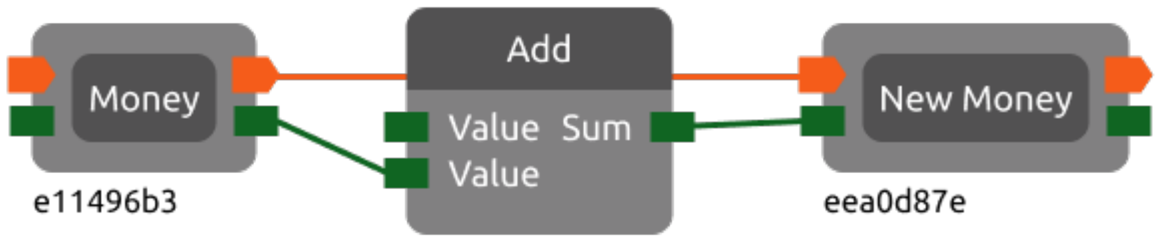
Add Money Ids

Suppose we want to store our money in a different variable than the one we spawned it in. We can create a new variable by changing an existing variable to a new name.



New Money

Since the different variables have different ids, they also have different values.



New Money Ids

When a binding instance has it's name changed it creates a new definition. This keeps variables working in the way that creators expect: uniquely named variables have unique values.

Binding Instances vs Bindings and Instances

Right now, the only way to use a variable in Rec Room is as a binding instance: there is no variable definition chip that only provides a binding. On the opposite end of the spectrum, the only way to use an event is with separate bindings and instances: you have to have an event definition to use an event instance. In the future we plan to expand both kinds of entity to cover the full spectrum.

Creators find great value in using variables as they are now. It's nice to be able to spawn the variable chip and use it immediately, rather than spawning two chips and changing to the config tool to select a variable definition. On the other hand, binding instances create problems for cross-scope variables.

If an instance is placed in both a parent and child scope, the definition technically exists in multiple places and deleting a variable will move the definition across scopes. This behavior may be unexpected for creators and creates complications for inventions which is why binding instances (and bindings in general) cannot exist in direct hierarchy if they share the same id.

Related Features

- Functions: functions will need a sane definition of scoping. By consolidating the rules for events and variables we can ensure that we have firm foundation in place when implementing functions in the future instead of 3 distinct scoping solutions.
- Variable definitions: variable definitions enables access to variables from child scopes. The rules we have right now, which treat names as primary keys and change names in certain spawning scenarios, make the design for variable definitions difficult to implement.
- Object model: Circuits entities will eventually exist in a hierarchy alongside object model entities. We should ensure the scoping rules for circuits are sane by then so that object model hierarchies are not broken.

Other Considerations

- Why do we let multiple of the same bindings in the same scope?
    - This provides an easy way for creators to clone a binding and change it. This clone-and-customize operation is pretty common for our creators and the easiest way to support it is by allowing the multiple bindings. We have to do it for binding instances anyways or you wouldn't be able to have multiple variables with the same name get the same value.
- Why can't I put the same definition in ancestral or descendant scopes?
    - This creates ambiguity and a lot of headaches about which scope is actually providing the definition. Additionally, it causes scope transitions on definition deletion which may not be obvious to creators and may break some inventions.