

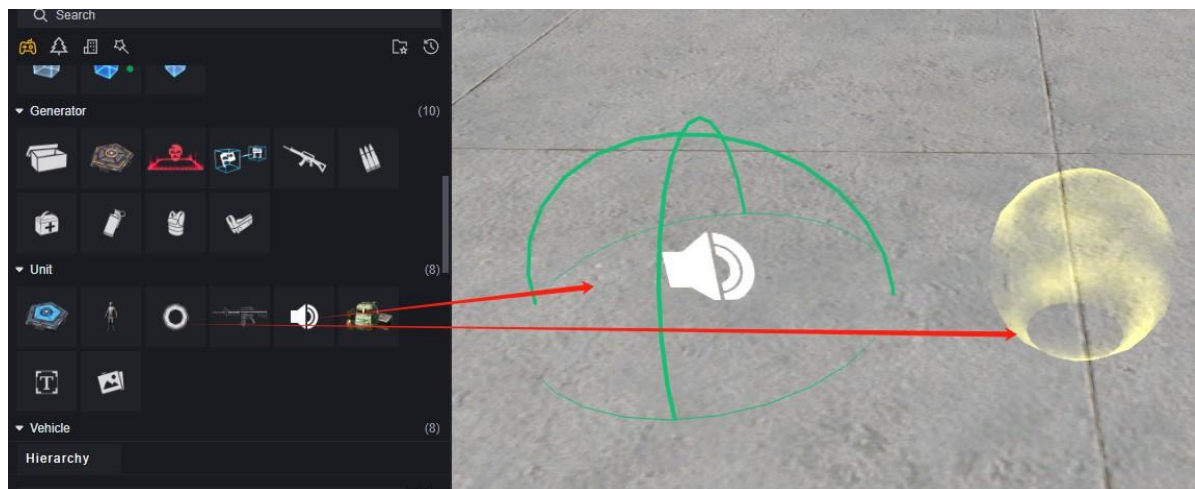
# Playback System-Owner's Manual

A playback system is a collection of a class of concepts that can be played, including playback sound effects, playback special effects, and playback motion. With the help of playback systems, a number of customized representations can be accomplished.

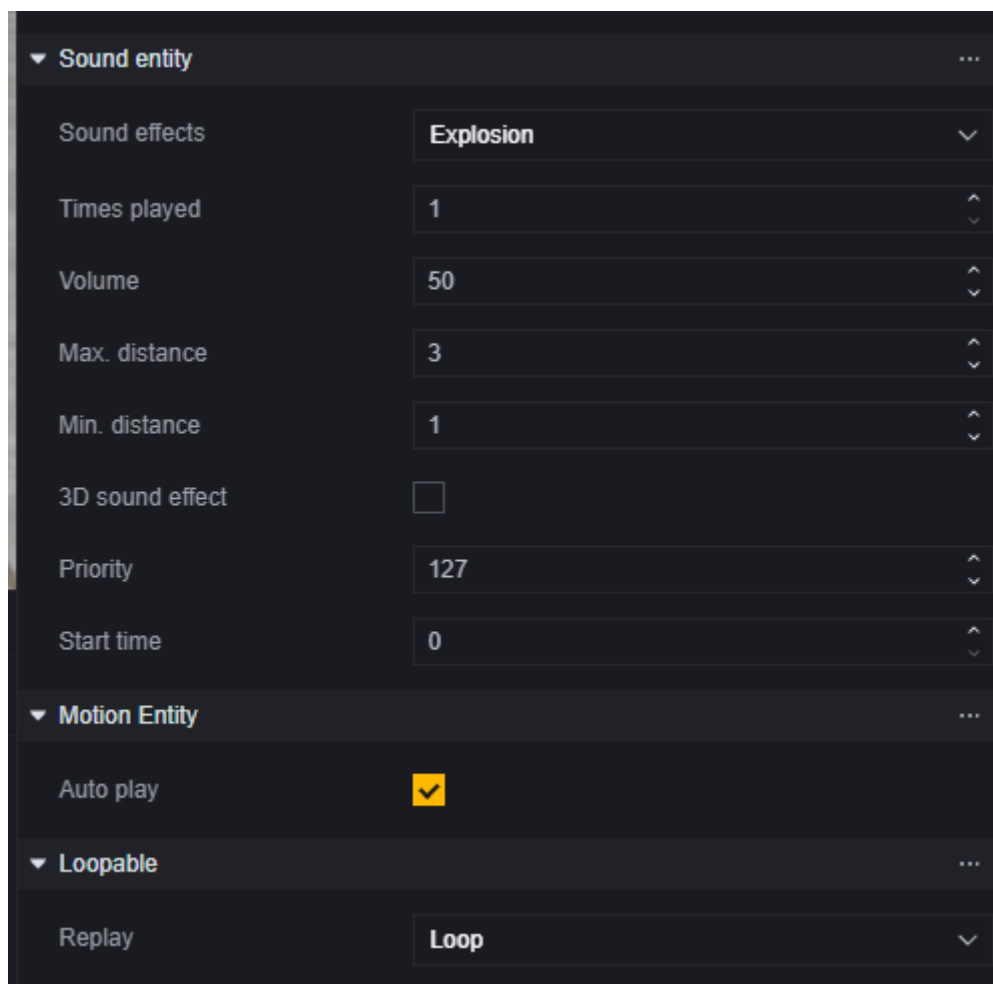
One of the campaigns is more complex and will be highlighted.

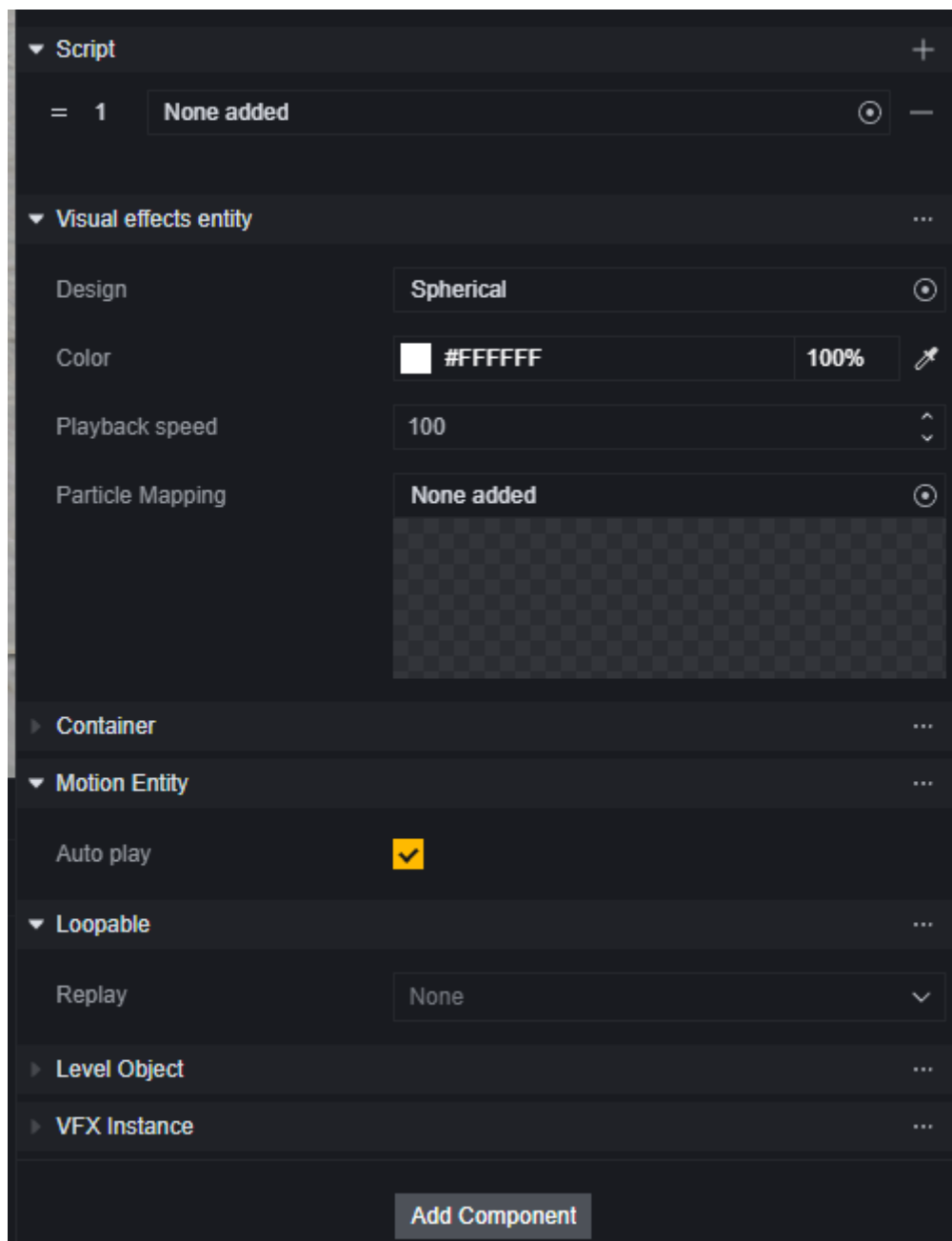
## Sound & Effects

Sound and FX entities can be created by unit entities.



and modify the configuration through the Properties panel.





Configuration of special effects entities

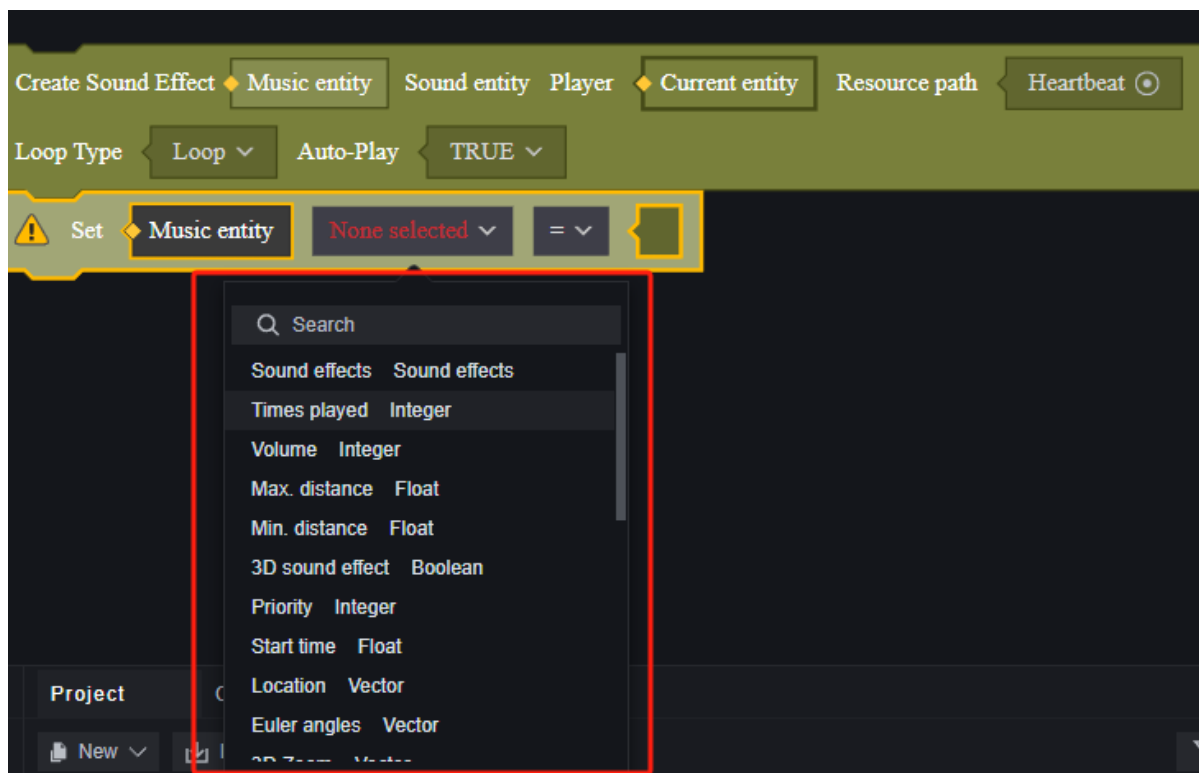
Entities created in this way are static.

Scripts can be used when dynamic modification of sound/effects is required. Sound and FX entities are usually created and controlled through graphical scripts.



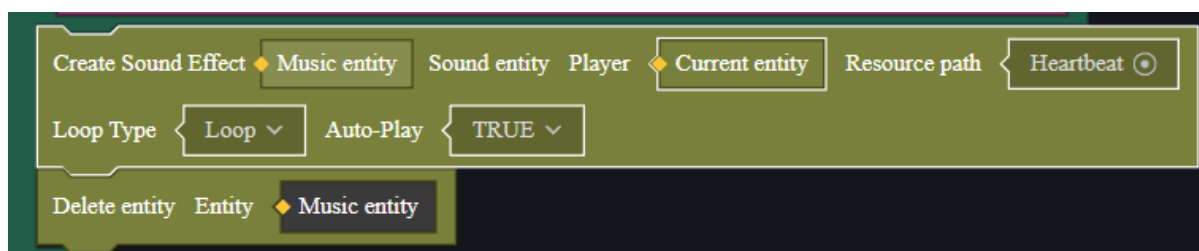
When creating via server script, you need to specify the player, and the created effects/sound effects can only be seen/heard by that player. When created via client script, it is only created on the client running the logic.

The position of the created entity defaults to (0, 0, 0), and the position and other modifiable properties can be adjusted by setting properties.



Some properties of sound effects

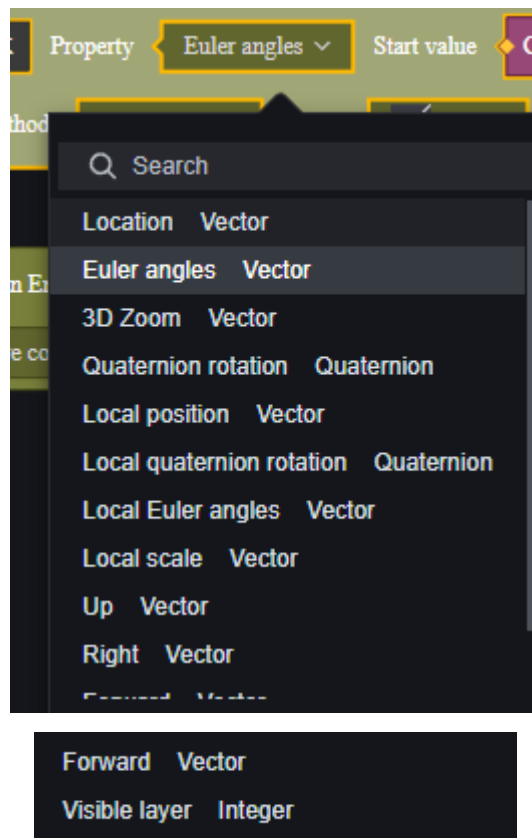
When you wish to end an effect/sound effect, you can choose to delete the corresponding entity.



The application of sound effects and special effects will be explained in the final example section along with the campaign.

## campaigns

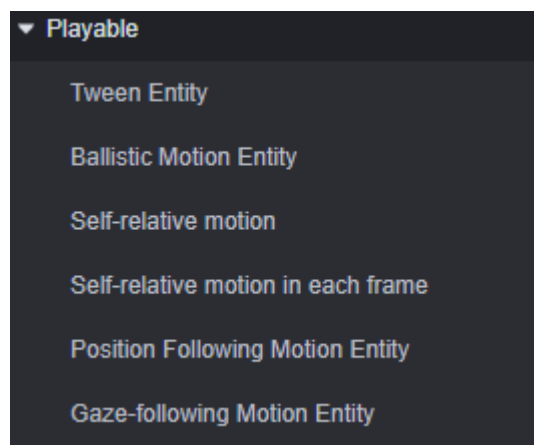
Motion is the process of an entity's certain motion attribute constantly changing. In addition to the common position, rotation, scaling, etc. are also attributes that can be controlled by the motion playback system.



Properties that can be exercised

A motion entity must be created by a graphical script and mounted on some entity. This entity will move with prescribed rules to realize motion patterns such as acceleration and rotation.

The campaign has six playback modes:



#### 1. interpolation campaign

The rate of movement of the entity varies in a prescribed curve that requires a prescribed start and end point.

#### 2. ballistic movement

Linearly accelerated motion.

#### 3. (math.) motion relative to itself

Similar to the interpolation movement, but always starting after its own last change.

#### 4. Motion per frame relative to itself

Similar to motion relative to itself, but the frequency of change is once per frame.

#### 5. position-following motion

position to follow the target entity.

## 6. gaze-following motion

Orientation is always towards the target entity.

# Motion Playback System Elements

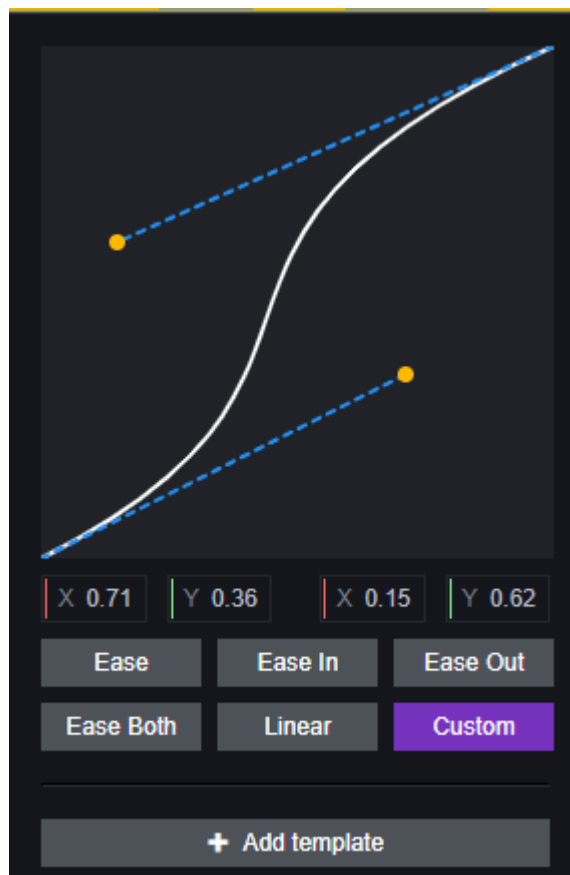
## Creating Motion Entities

Each motion playback mode has a corresponding creation element with a different configuration. The following parameters are presented in order from left to right, top to bottom

### interpolation campaign

Create Interpolation Motion ◆ Tween Entity Tween Entity Entity Entity Property None selected Start value Entity Target value Entity Method None selected Curve Ease Time 0 Auto-Play FALSE

1. Interpolated Motion Entity: outputs the parameters that create the interpolated motion entity.
2. Entity: The entity that is mounted so that the entity interpolates the motion.
3. Attributes: which attributes of the mounted entity are made to move, which need to be filled in before the entity can read the attributes it can move. Typically position, rotation, scaling, etc.
4. Starting value: what value the attribute of the mounted entity uses as the starting point of the movement.
5. Target value: what the properties of the mounted entity will change to after a single campaign.
6. Mode: the playback mode, you can select Single, Loop and Reciprocal. Loop indicates that after each movement is completed, the entity will move again from the start value. Reciprocating means that after the movement from the start point to the end point is completed, the entity will move from the end point to the start point one more time and then repeat the process.
7. Curve: a rate curve for property changes. Rate curves are linear Bessel curves, consisting of a fixed start point, end point, and two control points.



Several templates are provided for the curves, and the coordinates of the control points can also be customized.

You can think of the horizontal coordinates of the curve in the graph as the time and the vertical coordinates as the attribute values, with the **4. starting value at time 0** at the beginning, the end point being the **5. target value at time 8. time**, and the rate of change being the tangent line to the curve.

The rate is constant if the curve is straight, i.e. if the control point coordinates are all (0, 0). In the case shown in the figure the property changes will slow down, then speed up, then slow down again.

8. Time: the duration of a change in milliseconds.

9. Autoplay after creation: whether or not playback starts automatically after creation.

#### ballistic movement

1. Ballistic Motion Entity: the created ballistic motion entity.

2. Entity: The entity that is mounted, causing that entity to move ballistically.

3. Initial Velocity: the initial velocity, a three-dimensional vector indicating the direction and magnitude of the velocity.

4. Acceleration: acceleration, a three-dimensional vector that indicates the direction and magnitude of acceleration.

5. Maximum distance: the distance of the ballistic movement, the movement is completed after reaching the distance.

6. Autoplay after creation: whether or not playback starts automatically after creation.

#### (math.) motion relative to itself

1. Relative Self Motion: the relative self motion entity created.

2. Entity: a mounted entity that causes the entity to move relative to itself.

3. Attribute: which attribute of this entity is made to move.

4. Delta: the amount of change per movement, required to be of the same attribute data type as the **3. attribute** selection. For example, when selecting the position, the Delta is to be filled with a three-dimensional vector representing the change value of the position from itself.

5. Curve: a curve illustrating the interpolated motion with reference to a curve indicating the rate of change throughout the change.

6. Mode: With reference to the interpolated motion, decide the mode of the motion: single, cyclic or reciprocal.

7. Time: The time spent on each movement in milliseconds.

8. Autoplay after creation: whether playback starts automatically after creation.

#### Motion per frame relative to itself

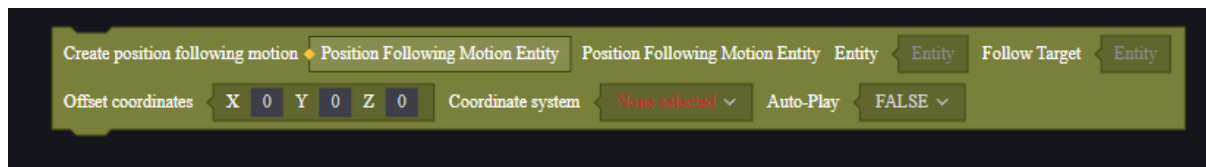
1. Relative to self per frame motion: the relative to self per frame motion entity created.

2. Entity: a mounted entity that causes the entity to move relative to itself per frame.

3. Attribute: which attribute of this entity is made to move.

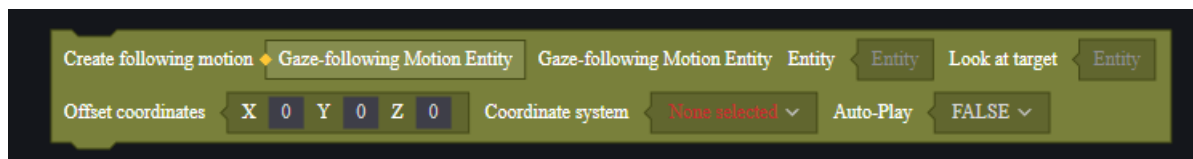
4. Delta: the amount of change in motion per frame, required to be of the same attribute data type as the **3. attribute** selection. For example, when selecting the position, the Delta should be filled with a three-dimensional vector inside, indicating the change value of the position compared to itself.
5. IsSwitch: the configuration when use in combination with other motion modes, can be used in combination with relative motion when it is true, can be used in combination with absolute motion when it is false. Please see below for the rules of combination.
6. Autoplay after creation: whether playback starts automatically after creation.

#### position-following motion



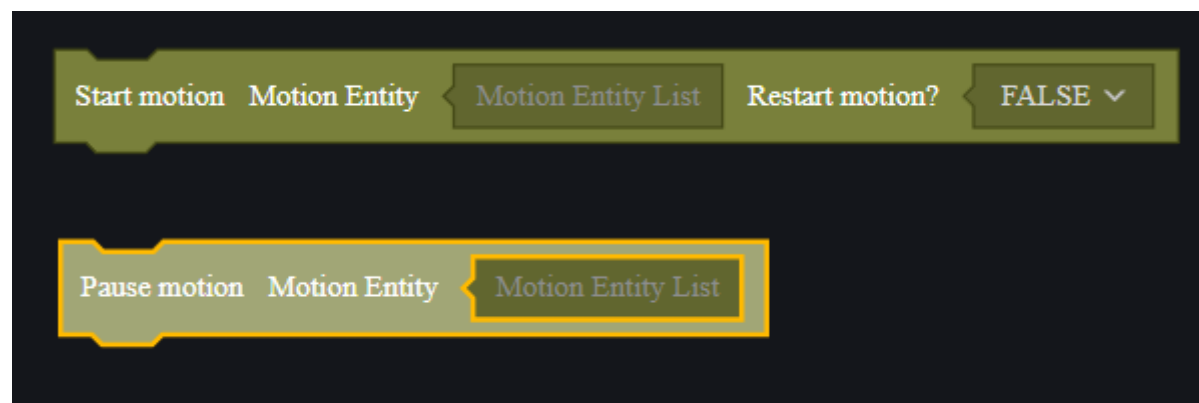
1. Position-following Motion Entity: The position-following motion entity created.
2. Entity: The entity that is mounted so that the position of the entity follows the motion.
3. Follow the target: the target to follow.
4. Offset Coordinates: fill in the offset coordinates, the position of the moving entity will keep the direction and size of the offset coordinates with the following target.
5. Coordinate system: you can choose between world coordinates or relative coordinates. This determines in which coordinate system the **4. offset coordinates** are offset.
6. Autoplay after creation: whether or not playback starts automatically after creation.

#### gaze-following motion



1. Note-following Motion Entity: the created Note-following Motion Entity.
2. Entity: The entity that is mounted so that the entity gaze follows the motion.
3. Target of attention: the target of attention.
4. Offset Coordinates: fill in the offset coordinates, the center of gaze will be offset from the target center by the corresponding direction and size.
5. Coordinate system: you can choose between world coordinates or relative coordinates. This determines in which coordinate system the **4. offset coordinates** are offset.
6. Autoplay after creation: whether playback starts automatically after creation.

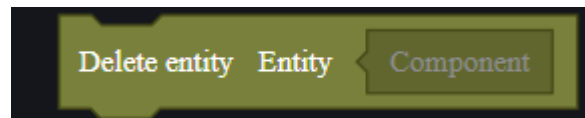
#### Control motion entity playback



Motion entities can be played or paused using the API.

Whether or not to replay in playback means whether or not to make the motion of the entity restart from the starting value. Pausing the motion does not delete the motion entity.

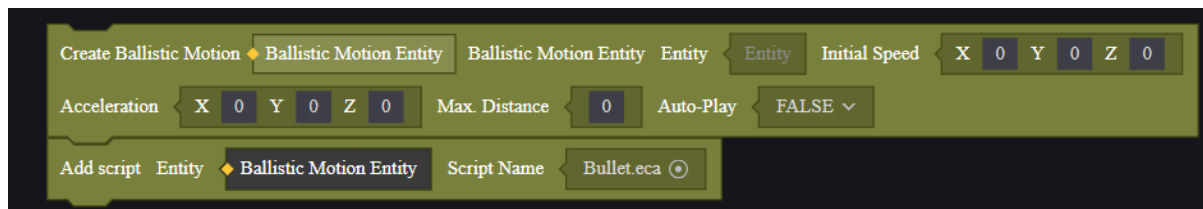
Deleting a campaign entity requires the use of the Delete Entity interface, though it is important to note whether the parameter filled in is a campaign entity or a mounted entity, as the former will delete the created campaign, and the latter will delete the mounted entity itself.



## sporting entity event



Motion start, motion end, motion pause and motion resume will trigger the corresponding events. However, these events depend on the motion entity, so you should create the motion entity and then mount the corresponding script on the motion entity when you use it.



## Portfolio of Campaign Entities

Motion entities can be used in combinations, and in general there are no restrictions on these combinations; entities that are mounted with multiple motion entities will perform multiple motions at the same time, but you need to pay attention to the type of motion that is being used in combination with them when using **motion relative to each of their own frames**.

When creating **per-frame motion relative to itself**, there is an IsSwitch option, which can be used in combination with relative motion when True is selected, and in combination with absolute motion when False is selected.

**Relative to self motion, relative to self per frame motion, position follow motion, and gaze follow motion** are relative motions, and **interpolated motion and ballistic motion** patterns are absolute motions.

Using the wrong combination can result in movement that does not match expectations.

## typical example

An example is shown below to illustrate the use of motion entities, special effects and sound effects.

We will create four orbs that always rotate around the player and add effects, play sound effects, and shoot out towards the player when they fire, and refresh the shot out orbs after 3 seconds.





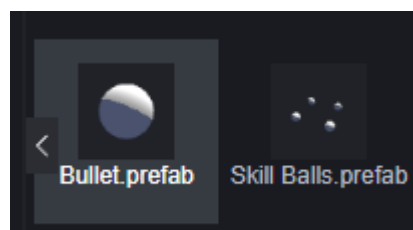
## preliminary

First create a Prefab, which is composed of a parent object and four orbs, which facilitates the configuration of the rotation.

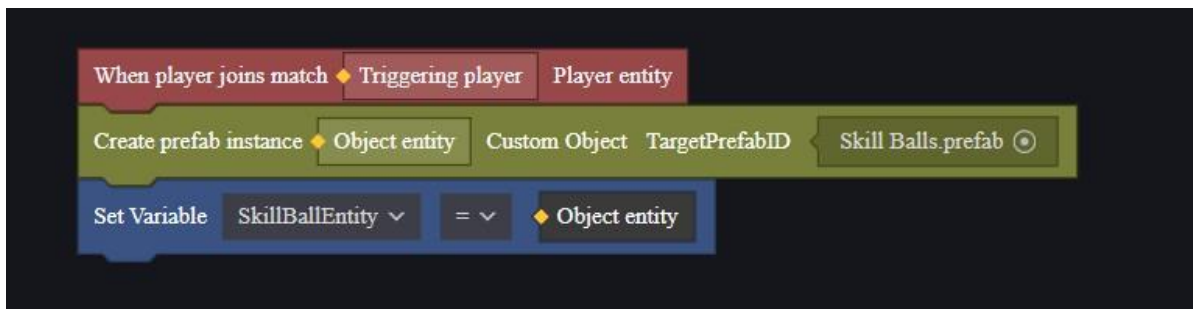
The orb uses the white orb of the base object, with the scaling set to 0.3 and the position relative to the parent object ( $\pm 0.6, 1, \pm 0.6$ ), respectively, and collision turned off.



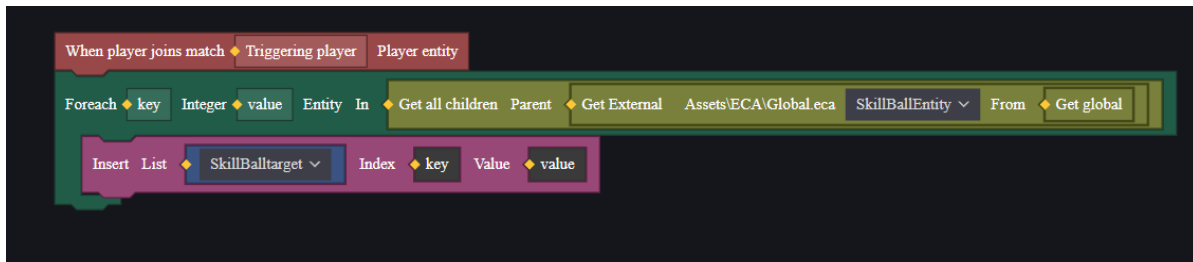
Create another bullet entity, configured to match the ball of the rotating entity, for special handling of the logic that fires out.



Sound and effects are chosen to use the default configurations of the officially provided resources and are not prepared in advance. A global script is created that will create the parent object and store it using script variables.



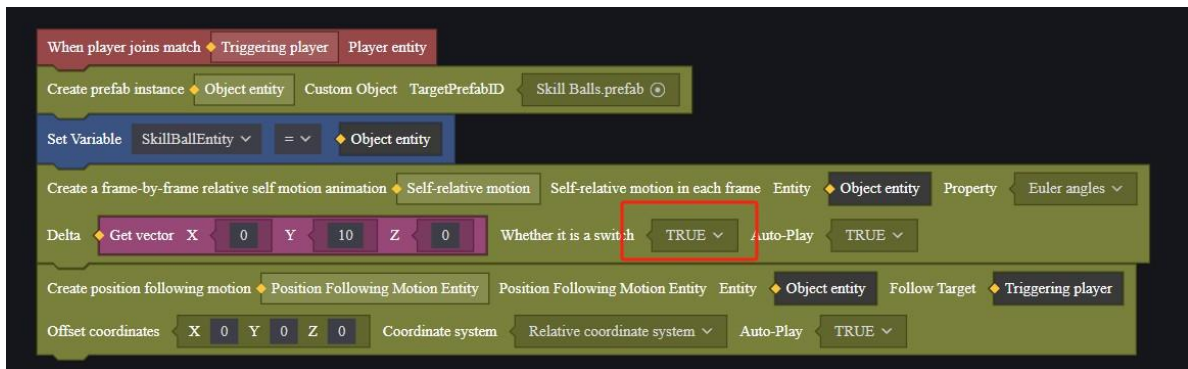
Create a player script that fetches the parent object created out of the global script using an external link and registers the four child objects in a list to facilitate subsequent handling of the bullet firing logic.



## Creating rotary motion

The parent object needs to follow the player at all times and rotate itself, which is a combination of two motions. We choose to use **per-frame motion relative to itself** and **position-following motion** in the global scripts

The first motion can also be satisfied using interpolated motion or motion relative to itself.



The motion property for **each frame of motion relative to itself** is set to Angle of Rotation, and Delta is set to (0, 10, 0), which means that the object will rotate by 10 degrees per frame, or 330 degrees per second, which is slightly less than one revolution.

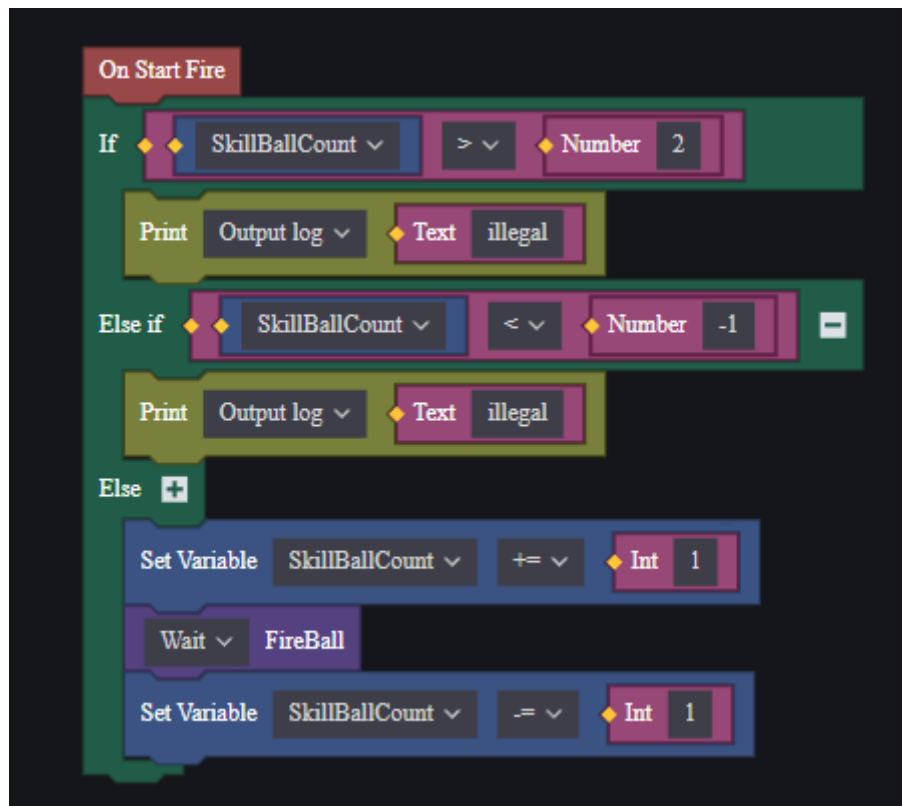
Because of the combination with another relative motion, IsSwitch is set to true.

Follow the player whose goal is set to trigger the event, i.e. every player who joins the game gets these four magic balls. Because the parent object is used and the child object already has an offset, no offset is set.

## Creating Shooting Sports

Shots need to be triggered by player shot events, and we return to the player script.

Here the balls that have been fired are detected by an integer variable, and if they are greater than four then they may not be continued.



This variable adds 1 each time it fires, ranging from -1 to 2. In practice, the values obtained in the FireBall method are 0 to 3, corresponding to the 4 balls stored in the list.

And each time the FireBall method ends, the value is decremented by 1 so that the smallest numbered of the balls around it will always be fired, preventing out-of-bounds.

In the FireBall method, we use a little trick to handle complex multimovement combinatorial logic: each time we should fire, we hide only the entity that is going to be fired out, and create a bullet entity at the location of that entity, mount a ballistic motion to it, and fire the created bullet out of the

Go. And after a period of movement, destroy this bullet and show the hidden entity back again.

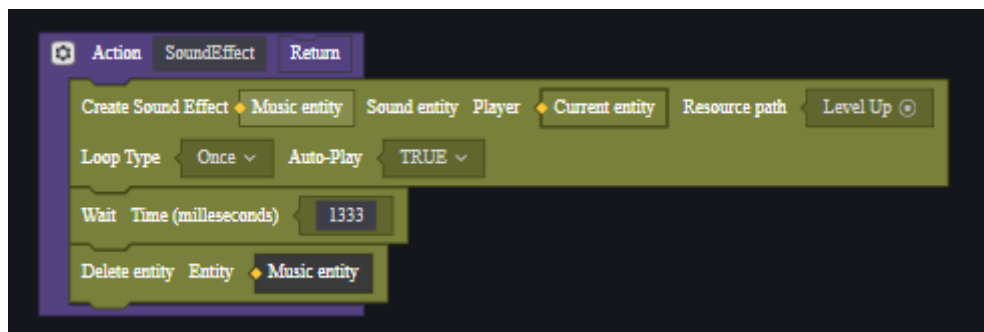


To attach another effect to the fired bullet, we'll use the Spherical effect and set the effect as a sub-object of the bullet, modifying the position and size of the effect a bit.

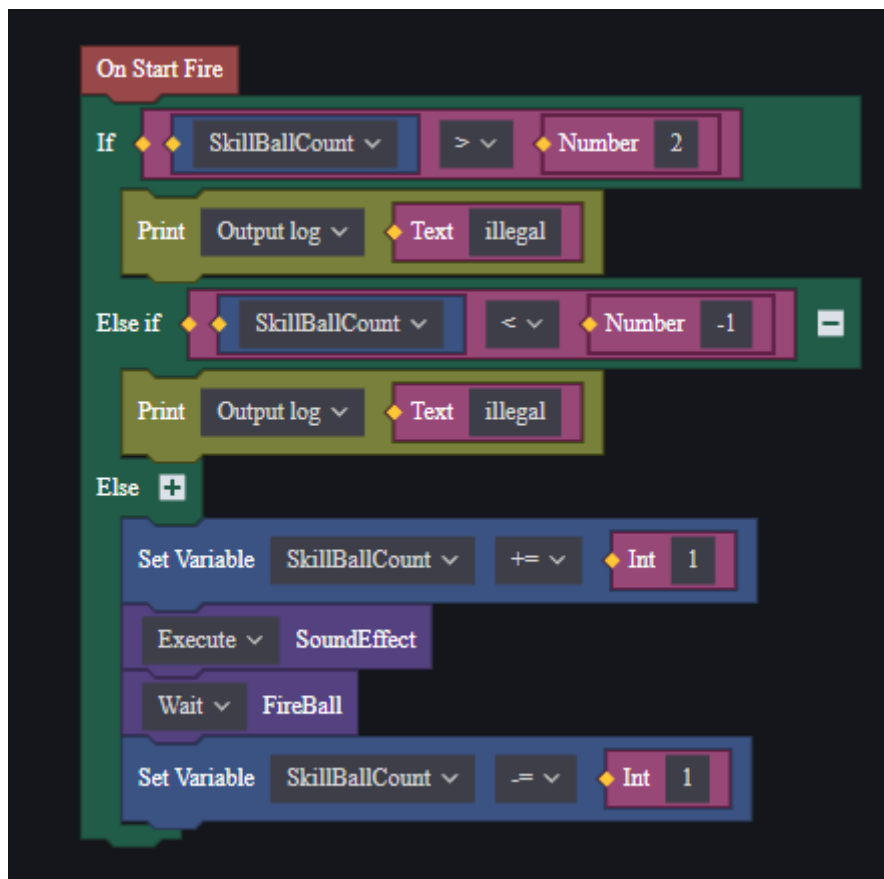


Since the effects entity is a sub-object of the bullet, it will be destroyed as the bullet is destroyed, so there is no need to specially handle the logic for destroying the effects. But the sound effects we chose to create separately and give a destruction logic.

Just for teaching demos, sound effects can be handled the same way as special effects.



Calls the sound method once when the player fires.



There is a function ordering issue to note here, both methods use an asynchronous tuple: wait. So both functions are asynchronous, but the requirement is that the sound effect must be played in parallel with the firing of the bullets, and the bullets must be fired immediately after the firing of the next moment, and the logic of each bullet is independent. So the sound method selects the execution method as Execute, and before the bullet method, the bullet method execution method is set to Wait, which means that the bullet is destroyed before the counter is incremented by 1 so that it can be fired again.

## runtime detection



In line with expectations.

This example is only used to show the content of the playback system, the actual similar needs can have other ways to realize.