

## BiScheme Level JSON Guide

### Level Files

The JSON files for levels can be specified in one of two formats:

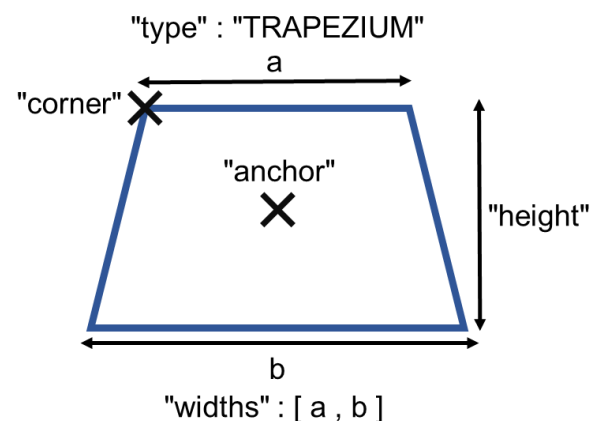
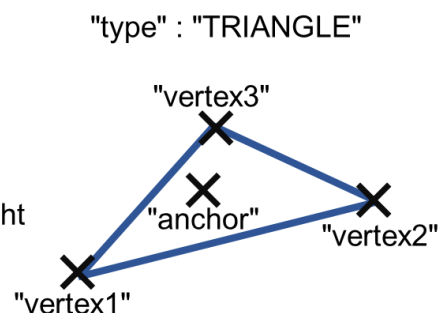
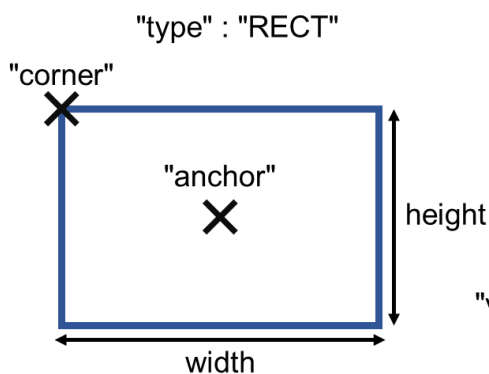
- As files within the `levels\` directory  
e.g. `levels\example.json`
- As their own directory within the `levels\` directory. When a level is specified within its own directory the core file for the level should be named `info.json`

e.g. `levels\example1\info.json`

### Level JSON Guide

The following are required within the Level JSON:

- **"name"** – The in-game display name for the level.  
e.g. `"name": "Example Level",`
- **"id"** – The unique integer identifier for the level.  
e.g. `"id": 0,`
- **"prerequisites"** – An array of ids of levels that must be completed prior to the level becoming available.  
e.g. `"prerequisites": [1, 2],`
- **"colourPrimary"** – Hex code for the level's PRIMARY colour.  
e.g. `"colourPrimary": "ffffff",`
- **"colourSecondary"** – Hex code for the level's SECONDARY colour.  
e.g. `"colourSecondary": "000000",`
- **"initRoomID"** – The id for the room of the level where the player starts.  
e.g. `"initRoomID": 0,`
- **"rooms"** – An array of either Room JSON or an array of Strings denoting the file names which hold the level's Room JSON.  
e.g. `"rooms": ["room0.json",  
"room1.json", "room2.json"]`



## Room JSON Guide

Whether they are specified within the "rooms" array within Level JSON or within separate room files, the Room JSON requires the following:

- **"id"** – The unique integer identifier for the room.  
e.g. `"id": 0,`
- **"dimensions"** – An array of numbers specifying the width (x-axis) and height (y-axis) of the room. Minimum dimensions of [16, 10] are recommended but not enforced.  
e.g. `"dimensions": [16, 10],`
- **"spawnPosition"** – An array of two floats specifying the position within the room at which the player will spawn/respawn.  
e.g. `"spawnPosition": [1.5, 7.5],`
- **"geometry"** – An object containing two arrays, "primary" and "secondary" which each contain geometry objects (see Geometry Guide)  
e.g. `"geometry": {  
"primary": [],  
"secondary": []  
}`
- **"adjacent"** – An array of adjacent objects (see Adjacency Guide)  
e.g. `"adjacent": [],`
- **"objects"** – An array of room objects (see Room Object Guide)  
e.g. `"objects": [],`

### Geometry Guide

The geometry of a room concerns the two monochromatic regions which comprise its structure, the primary region coloured with the level's primary colour and the secondary region coloured with the level's secondary colour. These two regions are determined by the geometry objects of the arrays "primary" and "secondary".

Each geometry object requires a "type" which is either "RECT", "TRIANGLE" or "TRAPEZIUM". Each type requires its own parameters.

## "RECT"

RECT defines a rectangle; each rectangle takes a **"dimension"** vector defining the width and height of the rectangle. The rectangle may also take a **"orientation"** value which defines the rectangle's rotation around its centre.

To define the position of the rectangle, either **"corner"** or **"anchor"** must be supplied. If **"anchor"** is provided then the rectangle is positioned with its centre at **"anchor"**. If **"corner"** is provided then the rectangle is positioned with its top-left corner at **"corner"** (then it is rotated around its centre by a provided "orientation").

e.g. The following two specifications describe the same rectangle:

```
{
  "type": "RECT",
  "anchor": [5, 6],
  "dimensions": [2, 3],
  "orientation": 0.7854
}
{
  "type": "RECT",
  "corner": [4, 4.5],
  "dimensions": [2, 3],
  "orientation": 0.7854
}
```

## "TRIANGLE"

TRIANGLE defines a triangle and such triangles may be specified in two ways. Firstly 3 vertices **"vertex1"**, **"vertex2"**, and **"vertex3"** may be specified relative to a central **"anchor"**. Or 2 vertices **"vertex1"** and **"vertex2"**, may be specified relative to a third vertex **"corner"**.

e.g. The following two specifications describe the same triangle:

```
{
  "type": "TRIANGLE",
  "anchor": [5, 5],
  "vertex1": [0, 1],
  "vertex2": [1, -1],
  "vertex3": [-1, -1],
}
{
  "type": "TRIANGLE",
  "corner": [5, 6],
  "vertex1": [1, -2],
  "vertex2": [-1, -2]
}
```

## "TRAPEZIUM"

TRAPEZIUM defines a trapezium; each trapezium takes a **"widths"** vector (describing the widths of its upper and lower parallel edges) and a **"height"** value (**"height"** is optional and defaults to 1 if not provided). The trapezium may also take a **"orientation"** value which defines the trapezium's rotation around its centre.

To define the position of the trapezium, either **"corner"** or **"anchor"** must be supplied. If **"anchor"** is provided then the trapezium is positioned with its centre at **"anchor"**. If **"corner"** is provided then the trapezium is positioned with its top-left corner at **"corner"** (then it is rotated around its centre by a provided "orientation").

e.g. The following two specifications describe the same trapezium:

```
{
  "type": "TRAPEZIUM",
  "anchor": [4, 4],
  "widths": [2, 3],
  "height": 1,
  "orientation": 0.5
}
{
  "type": "RECT",
  "corner": [3, 3.5],
  "widths": [2, 3],
  "height": 1,
  "orientation": 0.5
}
```

When defining a piece of geometry, additional **optional** physical properties may be defined.

- **"restitution"** – Determines the elasticity of collisions, defaults to 0.0 when not specified.  
e.g. "restitution": 0.0,
- **"staticFriction"** – Defaults to 1.0 when not specified.  
e.g. "staticFriction": 0.8,
- **"dynamicFriction"** – Defaults to 1.0 when not specified.  
e.g. "dynamicFriction": 0.5,

Lastly, it is important when defining the **"primary"** and **"secondary"** geometry arrays that the ENTIRE room is covered in geometry AND that there is NO OVERLAP between the two regions (geometry of the same region can overlap). Such constraints cannot be checked (easily) and therefore it is not enforced during parsing.

## Adjacency Guide

Rooms are connected through Adjacent objects which are specified in a room's **"adjacent"** array. Adjacency objects are both the entrances and exits to remove, each Adjacency is linked to another Adjacency and when a player reaches the Adjacency (on a collision) the room is switched to the linked Adjacency's room and they are positioned in the new room relative to the linked Adjacency.

Adjacency objects are positioned at the edges of rooms so it appears to the player that they are simply walking between rooms. Adjacency objects do not need to be linked in pairs; multiple Adjacency objects may link to a single exit Adjacency.

Each adjacency requires 4 parameters:

- **"id"** – A unique integer to identify the Adjacency from all other Adjacency objects and Room objects.  
e.g. "id": 0,
- **"colour"** – The colour which the player must be to pass through the Adjacency. Two linked Adjacency objects must have the same colour. The values of **"colour"** can be either PRIMARY or SECONDARY.  
e.g. "colour": "PRIMARY",
- **"side"** – The side of the room which the Adjacency lies along. Side is either LEFT, RIGHT, TOP or BOTTOM. Linked Adjacency objects must have opposite sides (LEFT links to RIGHT, TOP links to BOTTOM).  
e.g. "side": "TOP",
- **"range"** – Is a vector describing the start and end positions of the Adjacency along its respective side. Linked Adjacency objects must have the same difference between their start and end range positions. ([0, 5] and [3, 8] would link but [0,5] and [3,6] would not)  
e.g. "range": [0, 5]

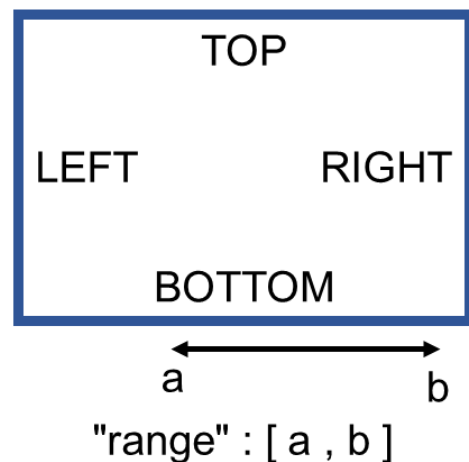
To link an adjacency to another room, 2 additional parameters must be provided.

- **"destRoomId"** – The unique identifier of the room which the adjacency links to.  
e.g. "destRoomId": 1,
- **"linkId"** – The unique identifier of the Adjacency in the destination room which the Adjacency links to.  
e.g. "linkId": 0,

e.g. Two rooms with a one-way Adjacency link would be.

```
"rooms": [
  {
    "id": 0,
    "adjacent": [{
      "id": 0,
      "colour": "SECONDARY",
      "side": "LEFT",
      "range": [4, 8],
      "destRoomId": 1,
      "linkId": 5
    }]
  },
  {
    "id": 1,
    "adjacent": [{
      "id": 5,
      "colour": "SECONDARY",
      "side": "RIGHT",
      "range": [2, 6]
    }]
  }
]
```

### Adjacency



## Room Object Guide

Room objects are the components of a room which display properties beyond the static geometry that has been described.

Each room object requires 4 parameters:

- **"id"** – A unique integer to identify the Room Object from all other Room Objects and Adjacency objects.  
e.g. "id": 0,
- **"type"** – The type of Room Object, the following options are available and described later. GEOMETRY, BLOCK, DOOR, LEVER, SPIKE, PORTAL, EXIT, and CUSTOM.  
e.g. "type": "BLOCK",

- **"colour"** – The colour of the Room Object. The PORTAL and EXIT types of Room Object do not use or require **"colour"**. The values of colour can be either PRIMARY or SECONDARY.  
e.g. `"colour": "PRIMARY",`
- **"anchor"** – The position of the Room Objects centre in the room. For some types, BLOCK, DOOR, and some **"gType"** variations of GEOMETRY and CUSTOM, **"anchor"** may be replaced with **"corner"**. **"anchor"** is not required for the EXIT **"gType"**.  
e.g. `"range": [0, 5]`

When defining any Room Object, additional **optional** physical properties may be defined.

- **"restitution"** – Determines the elasticity of collisions, defaults to 0.0 when not specified.  
e.g. `"restitution": 0.0,`
- **"staticFriction"** – Defaults to 1.0 when not specified.  
e.g. `"staticFriction": 0.8,`
- **"dynamicFriction"** – Defaults to 1.0 when not specified.  
e.g. `"dynamicFriction": 0.5,`

## "GEOMETRY"

GEOMETRY Room Objects allow for creating additional pieces of geometry which are not held to the constraints described in Geometry Guide (i.e. they can overlap with the geometry of different colours).

The type of the GEOMETRY object is specified by **"gType"**, which takes the values RECTANGLE, TRIANGLE, TRAPEZIUM, CIRCLE, ELLIPSE and POLYGON. Depending on the value of **"gType"**, additional fields must be provided.

RECTANGLE, TRIANGLE and TRAPEZIUM use the same parameters described for their respective shapes in Geometry Guide.

CIRCLE uses the provided **"anchor"** and a **"radius"**.

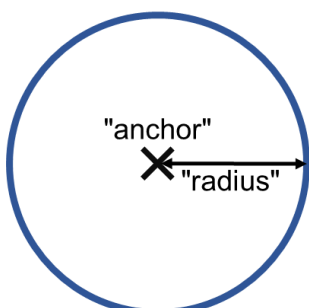
ELLIPSE uses the provided **"anchor"** and a **"dimensions"** vector to describe the width and height of the ellipse. The ellipse may also take an **"orientation"** value which defines the ellipse's rotation around its centre.

POLYGON defines a regular polygon defined around the provided **"anchor"**. It takes **"sides"** value to denote the number of sides it possesses and a **"dimensions"** vector to describe its width and height. The polygon may also take an **"orientation"** value which defines the polygon's rotation around its centre.

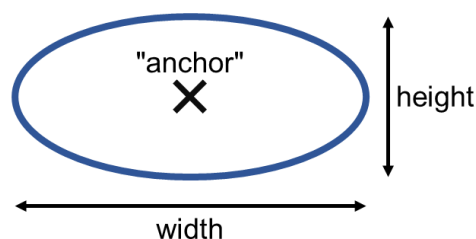
e.g. Some GEOMETRY objects using gType would be:

```
{
  "id": 0,
  "colour": "PRIMARY",
  "type": "GEOMETRY",
  "gType": "CIRCLE",
  "anchor": [2, 4],
  "radius": 1.5,
},
{
  "id": 1,
  "colour": "PRIMARY",
  "type": "GEOMETRY",
  "gType": "ELLIPSE",
  "anchor": [6, 4],
  "dimensions": [3, 1],
  "orientation": 0.0
},
{
  "id": 2,
  "colour": "PRIMARY",
  "type": "GEOMETRY",
  "gType": "POLYGON",
  "anchor": [10, 4],
  "sides": 6,
  "dimensions": [3, 3],
  "orientation": 0.0
}
```

**"gType" : "CIRCLE"**



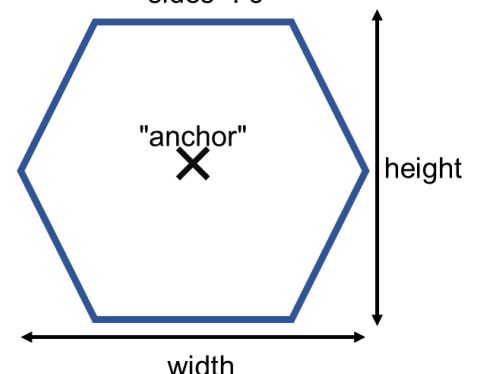
**"gType" : "ELLIPSE"**



**"dimensions" : [ width , height ]**

**"gType" : "POLYGON"**

**"sides" : 6**



**"dimensions" : [ width , height ]**

## "BLOCK"

A BLOCK is a rectangle that may be pushed around. A BLOCK requires a **"dimensions"** vector to describe its width and height. It also takes a **"mass"** value which defines its mass for in the context of physics interactions (if **"mass"** is not provided it defaults to 1). As the BLOCK is a rectangle, the **"anchor"** vector may be replaced with a **"corner"** vector instead (at which the BLOCK is position with respect to the corner instead of its centre).

The BLOCK optionally takes a boolean value **"initState"**. The BLOCK has two states, when the state is true, the BLOCK cannot be pushed or moved. When the state is false, the BLOCK can be pushed and moved. This state may be switched by LEVERs or special Behaviours. **"initState"** defines the initial state of the BLOCK. If **"initState"** is not provided it defaults to false.

e.g. A BLOCK definition would look like:

```
{
  "id": 3,
  "colour": "PRIMARY",
  "type": "BLOCK",
  "anchor": [5, 6],
  "dimensions": [2, 2],
  "mass": 5.0,
  "initState": true
}
```

## "DOOR"

A DOOR is a rectangle with two states, opened/closed (defined as by an internal state as true/false). A DOOR requires a **"dimensions"** vector to describe its width and height. As the DOOR is a rectangle, the **"anchor"** vector may be replaced with a **"corner"** vector instead (at which the DOOR is position with respect to the corner instead of its centre).

The DOOR optionally takes a boolean value **"initState"** which the initial state of the DOOR (true = open, false = closed). If **"initState"** is not provided it defaults to false.

e.g. A DOOR definition would look like:

```
{
  "id": 4,
  "colour": "PRIMARY",
  "type": "DOOR",
  "anchor": [5, 6],
  "dimensions": [1, 5],
  "initState": true
}
```

## "LEVER"

A LEVER is a 1x1 Room Object which the player may interact with. When interacted with the LEVER changes the state of other Room Objects in the room.

The only additional parameter required for a LEVER is **"linkedTo"** which is an array of integer ids to identify which Room Objects the LEVER will change the state of. The LEVER may also take an **"orientation"** value which defines the LEVER's sprite's rotation around its centre.

e.g. A LEVER definition would look like:

```
{
  "id": 5,
  "colour": "PRIMARY",
  "type": "LEVER",
  "anchor": [3, 3],
  "linkedTo": [3, 4]
}
```

## "SPIKE"

A SPIKE is a row of one or more 1x1 triangular spikes in a row that cause a player to return a spawn point (die and respawn) upon contact. The number of spikes in a row of SPIKE is determined by **"length"** (which defaults to 1 if not provided). The SPIKE may also take an **"orientation"** value which defines the SPIKE's orientation.

e.g. A SPIKE definition would look like:

```
{
  "id": 6,
  "colour": "PRIMARY",
  "type": "SPIKE",
  "anchor": [6, 3],
  "length": 5
}
```

## "PORTAL"

The PORTAL Room Object allows for a player, upon interaction, to switch between two geometry regions, switching their colour and potentially their direction of gravity.

PORTAL requires several parameters:

- **"width"** – The width of the PORTAL (or height if the portal is aligned vertically)
- **"startOpen"** – A boolean to state whether the PORTAL is initially open or must be opened by a LEVER or special Behaviour. If **"startOpen"** is not provided, it defaults to true.

- **"isOneWay"** – A boolean to state whether the PORTAL is only one way or not. If false, two parallel PORTALS will exist to allow bidirectional teleportation. If **"isOneWay"** is not provided, it defaults to false.

If **"isOneWay"** is true, two additional parameters are required:

- **"colour"** – The colour of the PORTAL (the colour which it switches the player to upon teleportation).
- **"side"** – The side an edge which the PORTAL protrudes from. Takes either LEFT, RIGHT, TOP or BOTTOM.

If **"isOneWay"** is false, 2 additional parameters are required.

- **"isVertical"** – Defines whether the PORTAL teleports the player vertically or horizontally (if true the PORTAL sits along a horizontal edge; if false the PORTAL sits along a vertical edge)
- **"topColour"** or **"leftColour"** – Defines the colour of either the top or left PORTAL or the two PORTALS, **"topColour"** is required if **"isVertical"** is true, otherwise **"leftColour"** is required.

e.g. PORTAL definitions could look like:

```
{
  "id": 7,
  "type": "PORTAL",
  "anchor": [8, 6],
  "width": 2,
  "isOneWay": true,
  "startOpen": false,
  "colour": "PRIMARY",
  "side": "LEFT"
},
{
  "id": 8,
  "type": "PORTAL",
  "anchor": [8, 6],
  "width": 2,
  "isVertical": true,
  "topColour": "PRIMARY"
}
```

## "EXIT"

The EXIT Room Object defines the end of a Level. Once an EXIT is reached, the Level ends and is considered completed.

Other than **"id"**, two parameters are required to define an exit:

- **"side"** – The side of the room which the EXIT lies along. Side is either LEFT, RIGHT, TOP or BOTTOM.
- **"range"** – Is a vector describing the start and end positions of the EXIT along its respective side.

e.g. **"side"**: "TOP",

e.g. **"range"**: [0, 5]

e.g. An EXIT definition could look like:

```
{
  "type" : "EXIT",
  "id" : 9,
  "side" : "RIGHT",
  "range" : [0, 4]
}
```

## "CUSTOM"

CUSTOM Room Objects are flexible and customisable Room Objects which may use any of the provided shapes of GEOMETRY and possess an array of Behaviours (see Behaviour Guide) to define their in-game properties.

To define the shape of the CUSTOM, **"gType"** is required. This **"gType"** possesses the same options as **"gType"** in GEOMETRY (and depending on the provided type, the same additional parameters are required/allowed).

The properties of the CUSTOM's rigid body may also be customised, through an optional **"rbType"** parameter which takes GEOMETRY (static and unmoving), NO COLLISION (no physics properties), ROTATEABLE (can only rotate), MOVEABLE (can move but not rotate) or BLOCK (can move and rotate). If ROTATEABLE, MOVEABLE or BLOCK is used then **"mass"** must be provided to specify the mass of the CUSTOM. If **"rbType"** is not provided it defaults to GEOMETRY.

The final, and core, component of CUSTOM is the **"behaviours"** array. Each object of this array is a Behaviour (see Behaviour Guide).

e.g. A CUSTOM definition could look like:

```
{
  "type": "CUSTOM",
  "id": 10,
  "gType": "RECT",
  "anchor": [5, 6],
  "dimensions": [2, 3],
  "orientation": 0.7854
  "rbType": "MOVEABLE",
  "mass": 2f,
  "behaviours": []
}
```

## Behaviour Guide

Behaviours determine a Room Object's capabilities outside of standard physical properties. The Room Objects BLOCK, DOOR, LEVER, SPIKE, PORTAL and EXIT as well as Adjacency objects all start with some predefined Behaviours. When creating CUSTOM Room Objects, Behaviours are included in the **"behaviours"** array. Each Behaviour requires a **"bType"** to specify the type of Behaviour it is. There are 11 available Behaviour types that may be specified.

### **"BHitKill"**

Whenever a Player collides with the Room Object they are returned to the room's spawn point (they die and respawn).

### **"BHitStateSwitch"**

Whenever something collides with the Room Object its internal state switches (either from true to false or false to true).

### **"BInteractStateSwitch"**

Allows for a nearby Player to interact with the Room Object, upon interact the Room Object's internal state is switched.

It requires additional parameters to specify when a Player is nearby. The Player is counted as nearby if they are within a provided radius (**"radius"**) or if the x and y distances of the Player to the Room Object's anchor are less than a provided x distance and y distance (**"xDist"** and **"yDist"**).

If the optional vector parameter **"indicatorOffset"** is provided, a small icon will appear (offset from the Room Object's anchor by the provided vector) when the player is able to interact.

If the optional boolean parameter **"activeOnState"** is provided, then the Room Object will only be interactable when it is in the provided boolean state.

### **"BStateBlock"**

This Behaviour causes a Room Object to become immovable when its state is true and movable when its state is false.

An icon will be added to the Room Object to specify whether it is currently movable or not. The size of this icon can be specified by **"iconSize"**. If not provided, **"iconSize"** defaults to 1.

If the optional boolean parameter **"initialState"** is provided, then the initial boolean state of the Room Object is set to the provided boolean value.

### **"BStateFlip"**

This Behaviour causes a Room Object's initial visual attribute to be flipped horizontally upon changing state. This only affects the visuals of the Room Object and not its physics mesh.

If the optional boolean parameter **"initialState"** is provided, then the initial boolean state of the Room Object is set to the provided boolean value.

### **"BStateHide"**

This Behaviour causes a Room Object to be hidden whilst its internal state is true.

If the optional float parameter **"iconSize"** is provided, then a locked icon will be added to the Room Object whilst its internal state is false. The size of this icon is specified by **"iconSize"**.

If the optional boolean parameter **"initialState"** is provided, then the initial boolean state of the Room Object is set to the provided boolean value.

### **"BStateSwapColour"**

This Behaviour causes a Room Object to have its colour swapped between primary/secondary whenever its internal state changes.

If the optional float parameter **"iconSize"** is provided, then a swap icon will be added to the Room Object. The size of this icon is specified by **"iconSize"**.

### **"BStateSwitchStates"**

This Behaviour causes the states of *other* room objects to switch whenever the Room Object's state switches.

It requires a parameter **"linkedTo"**, which is an array of integer ids to identify the other Room Objects in the room whose state will be switched whenever the Room Object changes state.

### **"BUpdateTimer"**

This Behaviour causes a Room Object's state to switch after periods of time.

The periods of time at which the Room Object's state switches can be specified in one of two ways. Either with a integer **"period"** parameter (which specifies how long (in frames) between each state change) or with an array of integers **"periods"** (which specify how long between each state change in a sequence of state changes).

If the optional integer parameter **"offset"** is provided, then the starting time (in frames) for the timer will be increased by the provided amount.

If the optional boolean parameter **"activeOnState"** is provided, then the timer will only run when it is in the provided boolean state

### **"BHitTeleport" and "BInteractTeleport"**

Both Behaviours allow for other Room Objects to be teleported to a different location after some event with the Behaviours' Room object. Both Behaviours share several parameters.

There is a single required parameter, a vector **"link"**. **"link"** which specifies where Room Objects will be teleported to.

If the boolean parameter **"swapColour"** is provided, then colour of the teleporting Room Object will be swapped between primary/secondary upon teleporting. If not provided **"swapColour"** defaults to false.

If the optional id parameter **"destRoomId"** is provided, then the teleporter will only apply to the Player and will teleport the Player to the room identified by the provided id.

If the optional boolean parameter **"flipGravity"** is provided, then the gravity of the teleporting Room Object will be flipped.

If the optional boolean parameter **"activeOnState"** is provided, then the Room Object will only be able to teleport when it is in the provided boolean state.

If any of the optional boolean parameters **"offsetX"**, **"offsetY"**, **"mirrorX"** or **"mirrorY"** are provided then the other three must also be provided. **"offsetX"** and **"offsetY"** specify whether the coordinates a Room Object teleports to should be offset by the distance between the Room Object's and the teleporter's locations. **"mirrorX"** and **"mirrorY"** specify whether, the Room Object should be teleported to the opposite side of the destinate relative to its positioning to the teleporter (e.g. if **"mirrorY"** is true, then a Room Object on the left side of the teleporter will be teleported to the right side of the destination)

### **"BHitTeleport"**

This Behaviour performs the above-described teleportation behaviour. This teleportation occurs to a Room Object when it collides with the Behaviour's Room Object.

If the optional boolean parameter **"playerOnly"** is provided, then the Behaviour will only apply on collision with a Player.

If the optional float parameter **"iconSize"** is provided, then a teleport icon will be added to the Room Object. The size of this icon is specified by **"iconSize"**.

### **"BInteractTeleport"**

This Behaviour performs the above-described teleportation behaviour. This teleportation occurs to a Player when it interacts with the Room Object.

If the optional float parameter **"iconSize"** is provided, then a teleport icon will be added to the Room Object. The size of this icon is specified by **"iconSize"**.

If the optional boolean parameter **"addIndicator"** is provided, a small icon will appear when the player is able to interact with the Room Object. The offset of this indicator from the Room Object's anchor may be specified by adding an optional vector parameter **"indicatorOffset"**, when not specified, this vector defaults to [0, 0]. The type of indicator displayed can be specified with **"indicatorType"**, three types of indicators are available INTERACT, TELEPORT and COLOURSWITCH. If not specified, INTERACT will be used.