

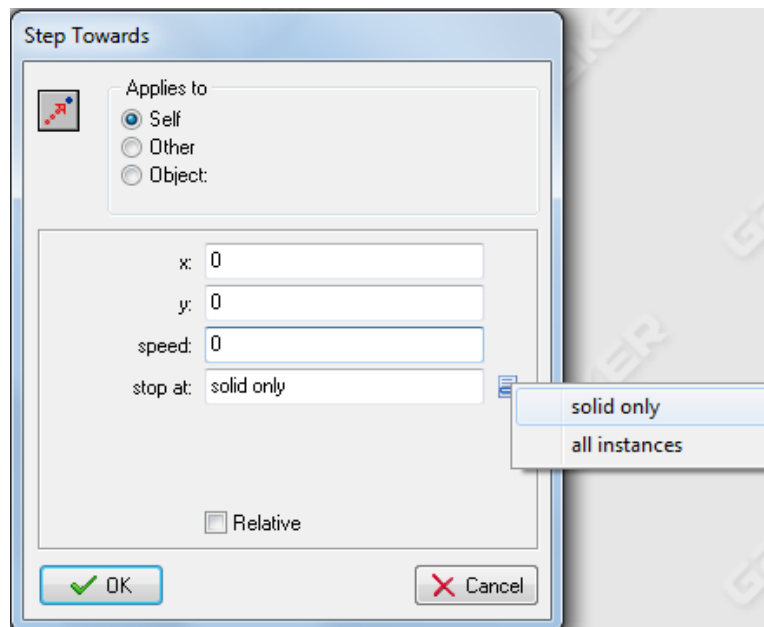
## STEP ACTIONS: STEP TOWARDS AND STEP AVOIDING

Oftentimes programmers want to create characters that either avoid other characters or move towards them. Think, for example, of a game where the objective is to avoid mummies or zombies that are chasing you. Well, GameMaker has two very useful actions called **Step Towards** and **Step Avoiding** that help you create objects that both avoid other object or move towards other objects.



### THE STEP TOWARDS ACTION

The **Step Towards** action is an action you would use if you want an object to move in a specific direction or toward another object. The difference between **Step Towards** and **Move Towards** is that in **Step Towards**, you can set a parameter for stopping the object when it encounters either solid object or all objects.



When setting the parameters of the **Step Towards** action, you need to specify the following:

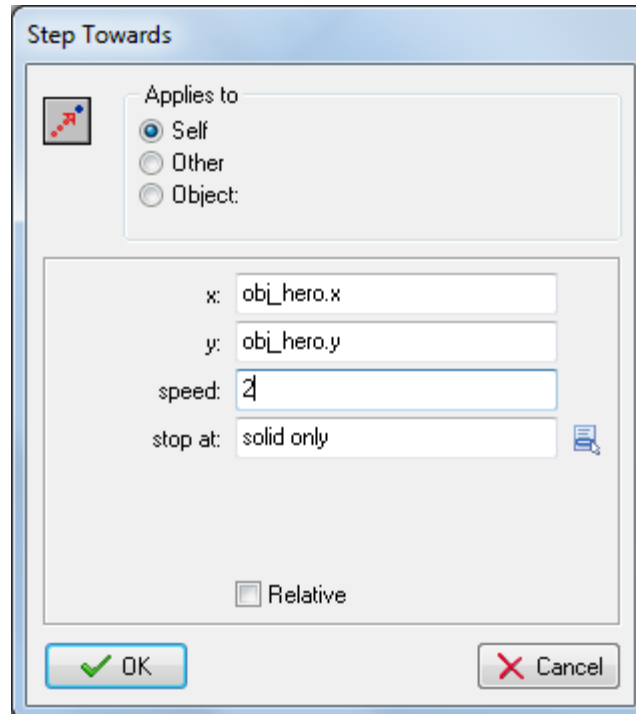
- 1. x:** This would be the x-position in the room that you would want the object to move towards.
- 2. y:** This would be the y-position in the room that you would want the object to move towards.
- 3. speed:** This would be the speed at which you want the object moving as it moves towards the point specified.

4. **stop at:** Here you can specify whether you want the object to stop at solid objects only or all objects.

## STEP TOWARDS A SPECIFIC OBJECT

Of course, it's possible to have an object step towards an object instead of a specific point in the room. In order to have an object move towards an object, you would simply need to specify the x- and y-position of the object you want it to follow.

In the following example, I want the object to step towards an object I have called **obj\_hero**:

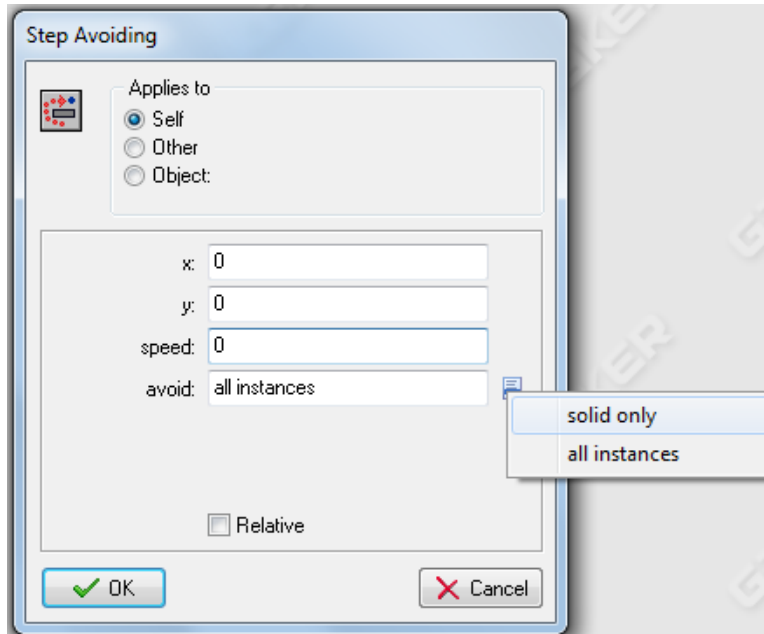


Since we would want the object to follow the other object repeatedly, we would need to add this action in a **Step Event**. Otherwise, the object will only step towards the other object when an instance of the object is created.

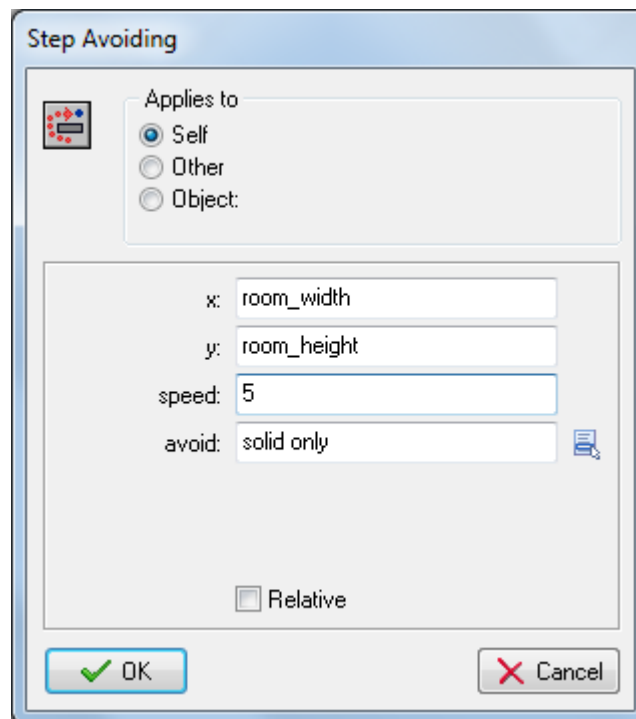
## THE STEP AVOIDING ACTION

The **Step Avoiding** action is similar to **Step Towards**, except you can tell the object to avoid solid objects or all objects.

So just like **Step Towards**, when setting the parameters for a **Step Avoiding** action, you need to specify the location in the room that you want the object to step towards, the speed at which you want it to move, and whether or not it should avoid solid objects or all objects.



In the following example, I am telling the object to step towards the bottom-right corner of the room (i.e. **room\_width** and **room\_height**) and avoid all objects.



## AVOIDING SPECIFIC OBJECTS

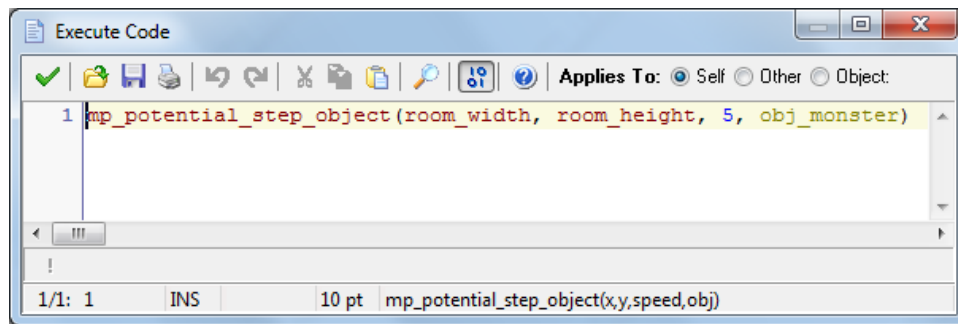
Sometimes avoiding solid objects or all instances of objects may not be what you're after because perhaps there are certain solid objects or certain instances of objects that you do not want to avoid. In that case, you can add some code to your Step event and use a function called

**mp\_potential\_step\_object** that allows you to specify specific objects to avoid. The syntax for the function is as follows,

```
mp_potential_step_object(xgoal, ygoal, stepsize, obj)
```

where **xgoal** is the target x-position, **ygoal** is the target y-position, **stepsize** is the speed the object moves in pixels per step, and **obj** is the object that you want to avoid.

So if wanted an object to avoid a specific object (let's call the object **obj\_monster**), we would add an **Execute Code** action (which you can find in the **control** tab) and write the following line of code:



Now the object will still try to step towards the bottom-right corner of the room, but will avoid the particular object called **obj\_monster**.

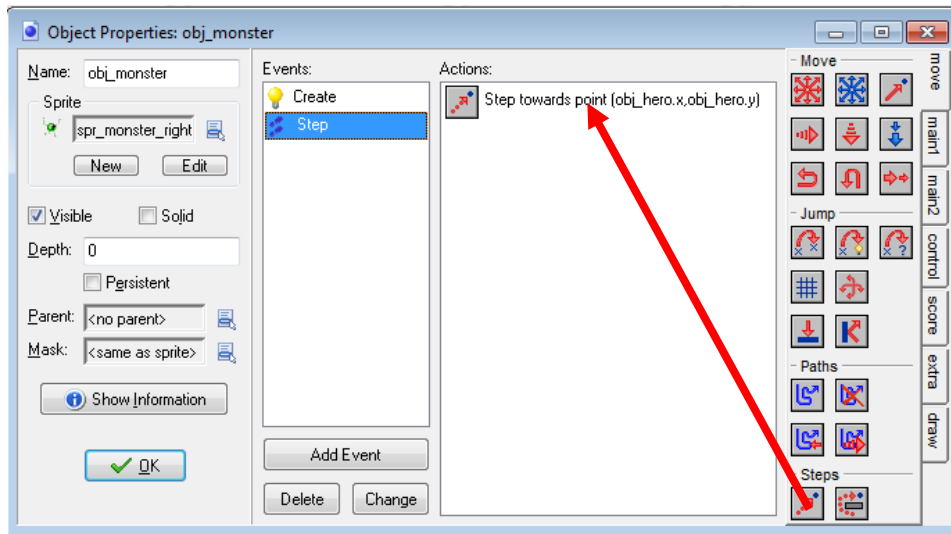
## CHANGING A SPRITE BASED ON THE DIRECTION AN OBJECT IS FACING

When more than one image is used to represent an object, it is often important to know what direction the object is facing. For example, if I am using a sprite that is comprised of four images representing the four directions the object is facing (up, left, down, right), I will need to constantly change the picture depending on its direction.

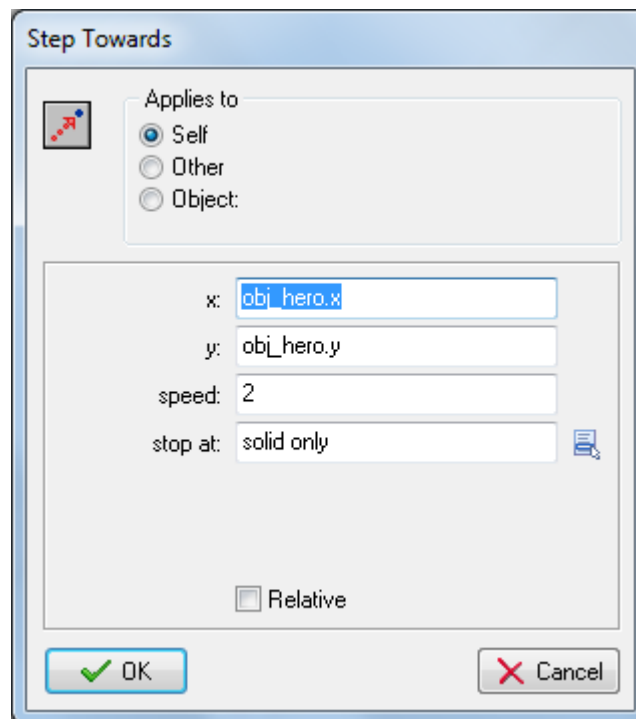
So let's say I have an object who's following another object. While following an object, it is very likely the direction of the object is going to change. For example, sometimes it may be moving right then have to change directions because the direction of the object it is following has changed.

In order to make this happen, we will need to write our first lines of code using the **Execute Code** action included in the **control** tab.

The first thing we will need to do though is create a **Step Event** because it is here that we are going to include a **Step Towards** action that will instruct the object to follow another object. In this example, I will be adding a Step Towards action to an object called **obj\_monster** that will step towards an object I have called **obj\_hero**.

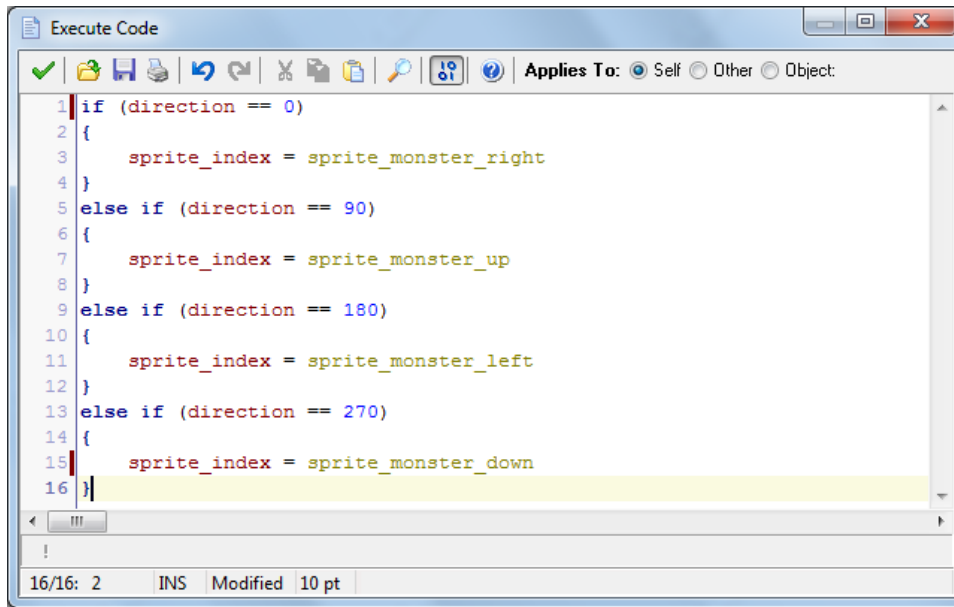


The **Step Towards** action requires you to specify the x- any y-position of the object you want it to follow, the speed at which you want it to move towards that object and whether you not you want it to stop at only solid objects or all instances of the object. These are the settings I will be using to have the monster object follow the hero object.



Next we will need to add an **Execute Code** action where we will write the code required to change the image based on the direction the image is facing.

In order to make this work, we will need to remember how direction works in GameMaker. When an object is facing right, the direction is 0. If an object is facing up, the direction is 90. If an object is facing left, the direction is 180, and if it is facing down, the direction is 270.



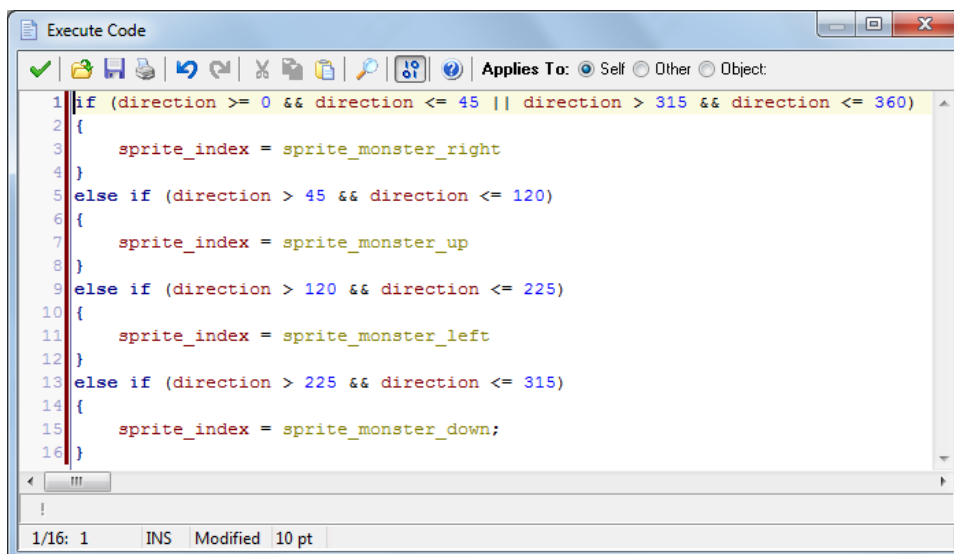
```
1 if (direction == 0)
2 {
3     sprite_index = sprite_monster_right
4 }
5 else if (direction == 90)
6 {
7     sprite_index = sprite_monster_up
8 }
9 else if (direction == 180)
10 {
11     sprite_index = sprite_monster_left
12 }
13 else if (direction == 270)
14 {
15     sprite_index = sprite_monster_down
16 }
```

Of course, the problem here with this code is that we're assuming that the direction will exactly equal one of these four values (0, 90, 180, or 270). But what if the direction does not exactly any of these values? What if, for example, the object's direction is 95? What will happen then? Given the code we have here, it won't do anything – the image won't change. So we need to account for other potential values representing the object's direction.

What we're going to do is set the image based on the following values:

- |            |  |
|------------|--|
| FACE RIGHT | If the direction is between 0 and 45 degrees or 315 and 360 degrees. |
| FACE UP    | If the direction is between 45 and 120 degrees.                      |
| FACE LEFT  | If the direction is between 120 and 225 degrees.                     |
| FACE DOWN  | If the direction is between 225 and 315 degrees.                     |

Our code will have to be changed in order to have the sprite changed based on these values:



```
1 if (direction >= 0 && direction <= 45 || direction > 315 && direction <= 360)
2 {
3     sprite_index = sprite_monster_right
4 }
5 else if (direction > 45 && direction <= 120)
6 {
7     sprite_index = sprite_monster_up
8 }
9 else if (direction > 120 && direction <= 225)
10 {
11     sprite_index = sprite_monster_left
12 }
13 else if (direction > 225 && direction <= 315)
14 {
15     sprite_index = sprite_monster_down;
16 }
```

