

# GAME MAKER TUTORIAL: TETRIS (GameMaker Studio)

Joe Weisbrod

February/April 2016

Faculty Advisor: Dr. J. W. Jaromczyk

Department of Computer Sciences

University of Kentucky

## Abstract

This tutorial is prepared as an introduction to game development with GameMaker. It includes a walkthrough guide for creating a Tetris game in GameMaker Studio, with all the necessary sprites and images. In addition, a “square one” (starting point) version of the game is provided. The development and testing has been done on Windows system with GameMaker Studio (Windows Free version) that can be downloaded from <http://www.yoyogames.com/get> .

## Overview

There are several basic elements which make up a video game: objects, sprites, and rooms.

**Objects** are the actual “things” that are in your game. These could be anything from your character, a block, bullets, doorways, etc.

**Sprites** are the images that represent objects. So while you might have a character object in your game, it is the sprite assigned to it that determines what the character would look like.

**Rooms** are the areas where the objects get placed and stuff actually happens. The game can only be in one room at a time, but multiple **instances** of each object can exist in a room. This means that you may have a standard block object, but you can place many copies of that block (instances) in the room to design your levels.

# Events – how we make stuff happen

Events are defined inside an object, and they are essentially groupings of instructions that are followed whenever certain things occur.

**Create Event:** This event is executed whenever an object is created. Note that objects that are put in a room before the game is run trigger their create event when the room starts.

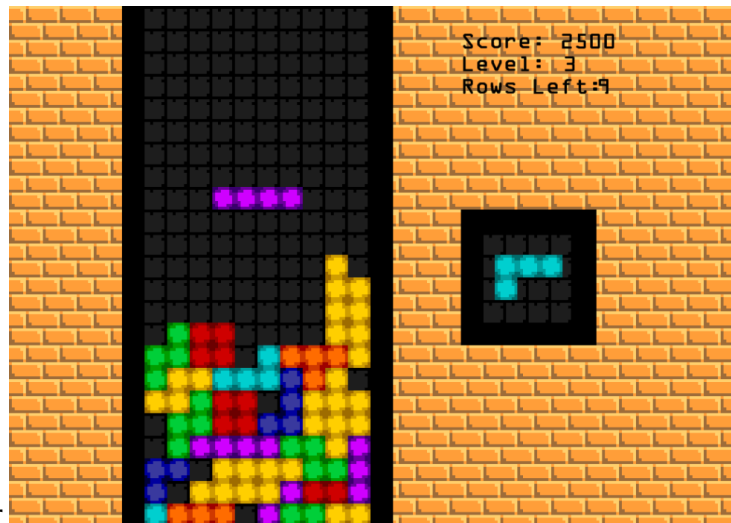
**Step:** This event is executed at regular intervals every so many fractions of a second. Just like a movie is has a certain frame rate, so video games have steps. This event is used when things need to be constantly calculated, especially for movement.

**Collision:** This event is executed whenever one object's sprite overlaps with another object's sprite. This is useful for keeping certain objects from passing through other objects (like a player through a wall).

**Alarm:** This event is essentially a timer. It is set to a certain number of steps that it will wait for, and then at the end of its waiting, will execute its code.

**Draw:** The draw event is where the object defines how it draws its sprite and/or any other graphical stuff on the screen.

**Keyboard Input:** These events occur whenever the user presses keys on the keyboard. These are useful for giving the player control over the pieces (i.e., allowing you to actually play the game).



## Tetris – A Basic Implementation

For this tutorial, we will be implementing a basic version of the classic game *Tetris*. All the sprites necessary for the game have already been created, so all you have to do is create the objects, rooms, and write the code.

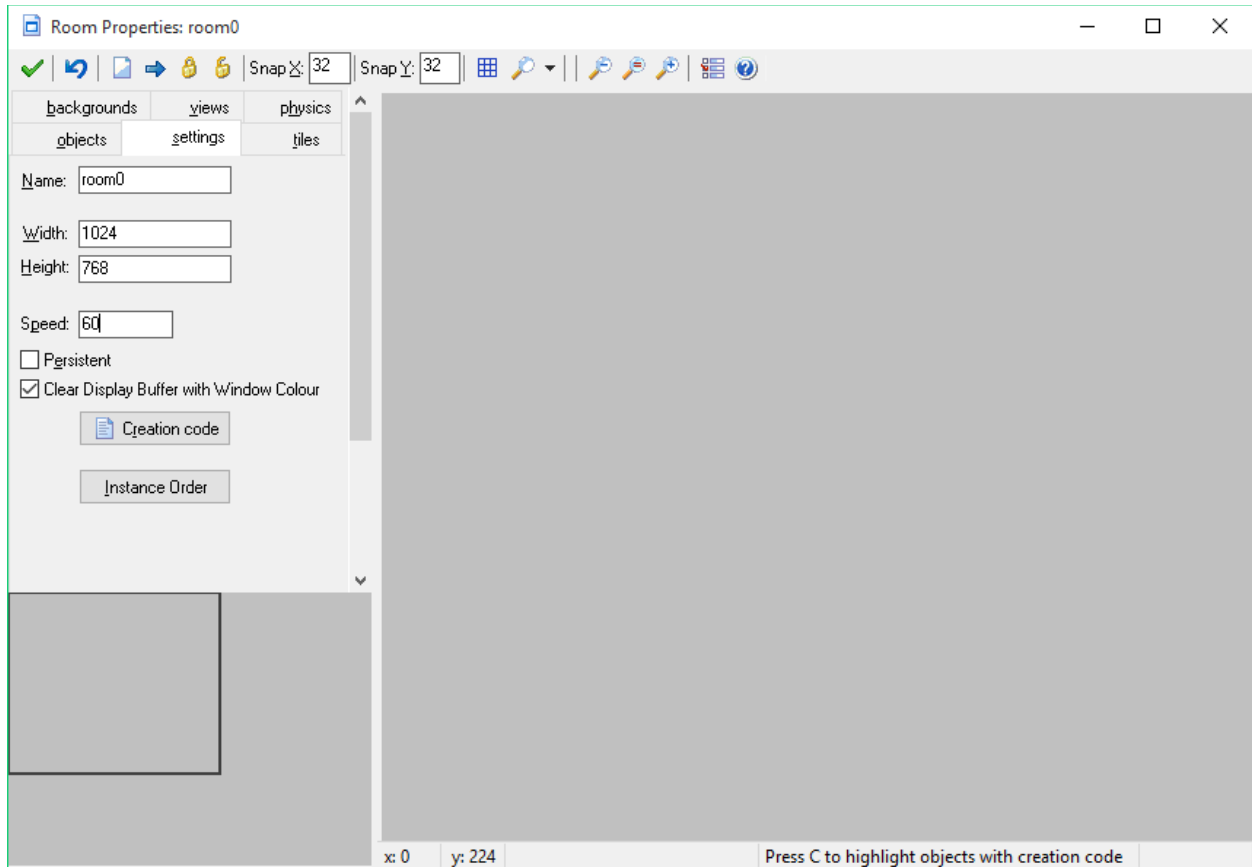
In building this game, we are going start by designing something simple and then adding things to it until it has all the features we want it to have.

Let's get started!

## The Level Design

The first thing we want to do is simple: let's have blocks fall from the top of the screen and stop when they hit the bottom.

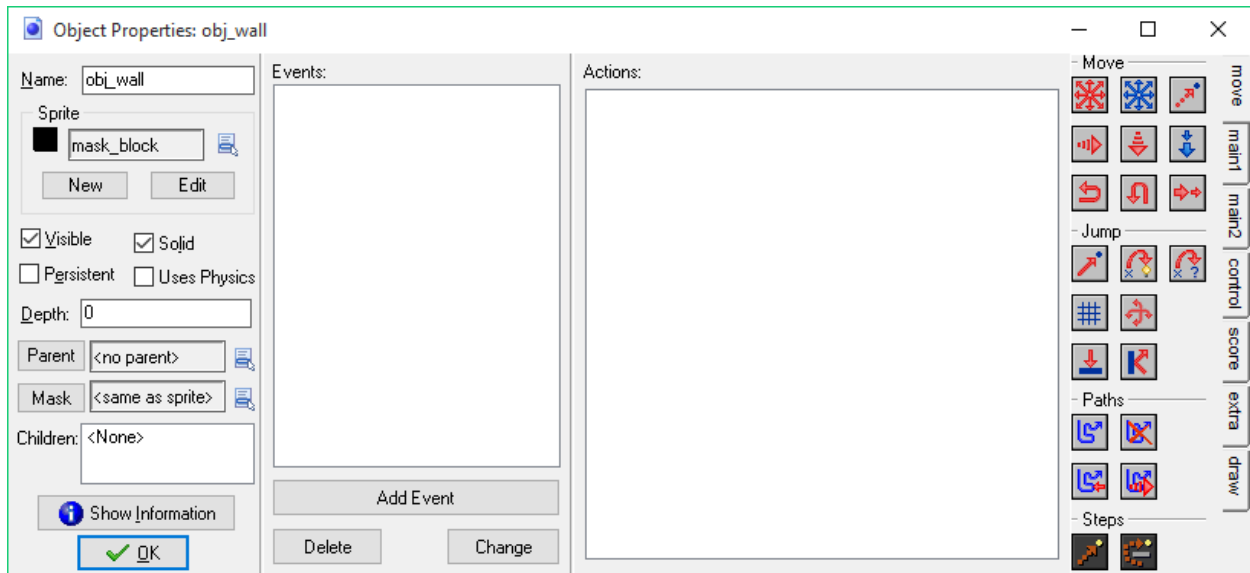
In order to do this, we will first need a room with the playing area defined. Create a room by right-clicking the "Rooms" folder and selecting "Create room":



This is what the room will look like when you first create it. You can call the room whatever you like, but we are just going to leave ours as room0.

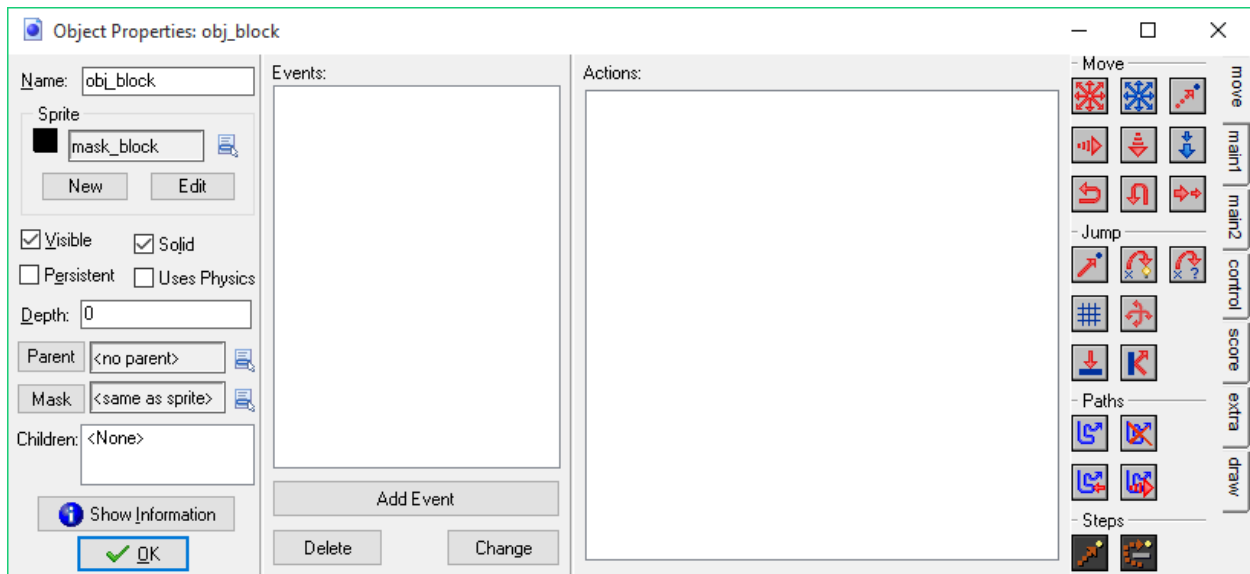
Note that to make the gameplay smoother, we changed the room "Speed" from 30 to 60. This means that the game will execute 60 step events per second, rather than just 30.

To define the playing area, let's create some wall objects. Right-click on the "Objects" folder and select "Create object":

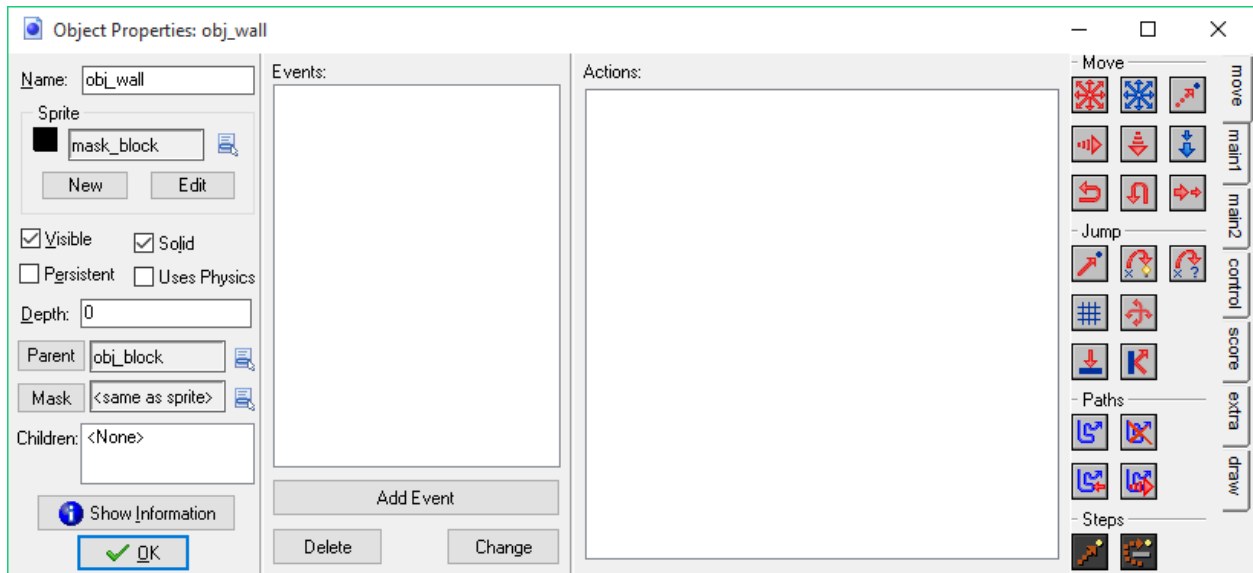


Note that we named the object “obj\_wall”, set the sprite to “mask\_block”, and checked the box that says “Solid”. Solid objects are going to be the ones that the Tetris pieces can’t fall through.

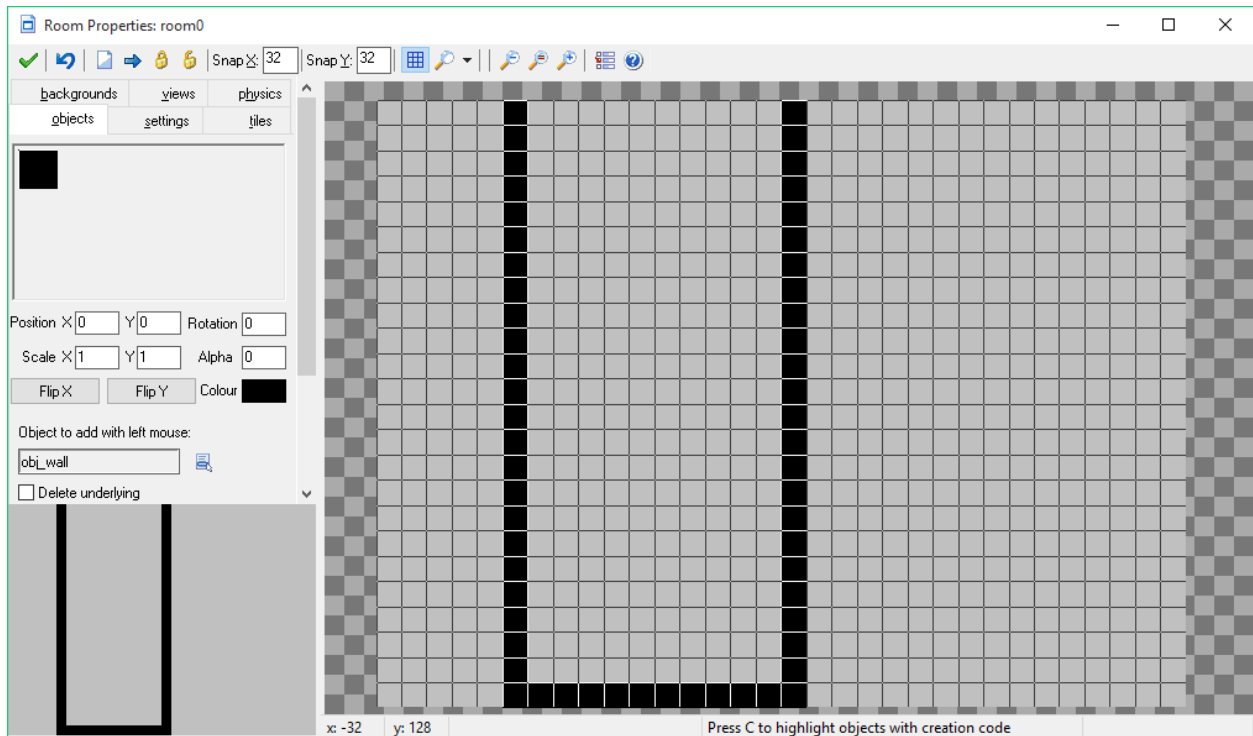
Next, we also need to create a parent “block” object. This object will represent all objects that the Tetris shapes cannot move through. Call the object “obj\_block” and make sure that you set the “Solid” checkbox.



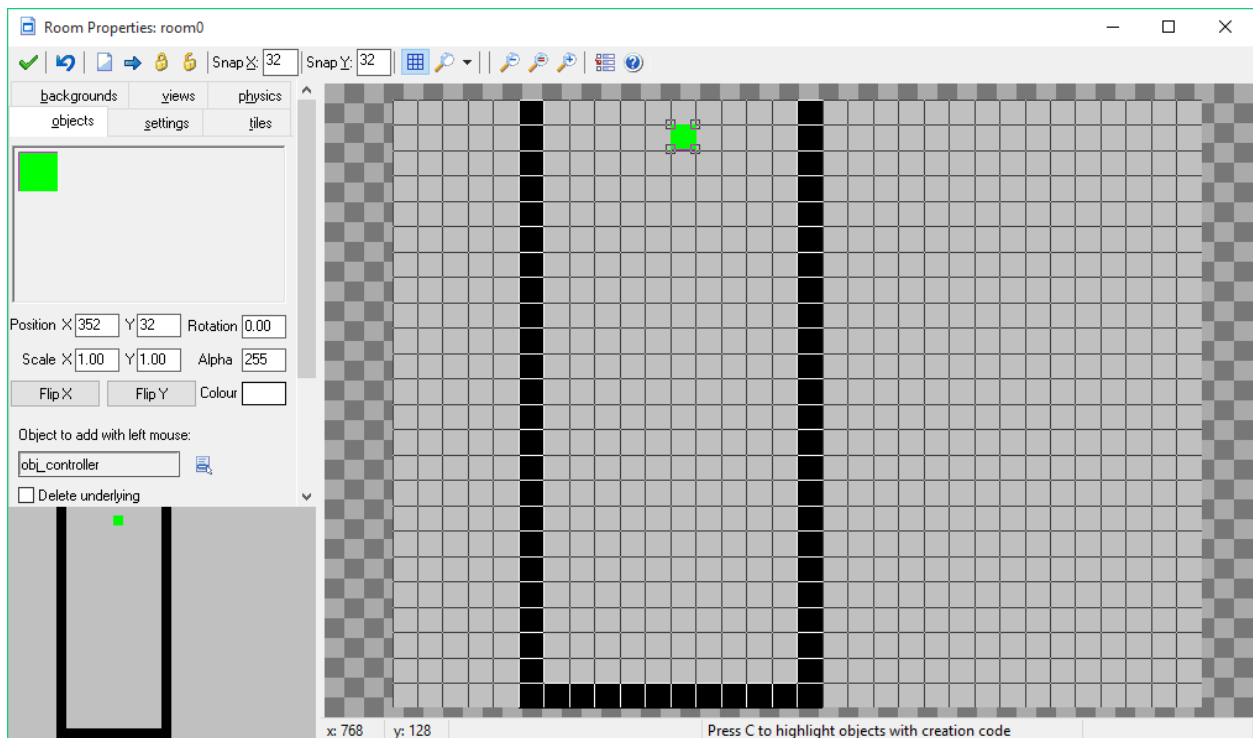
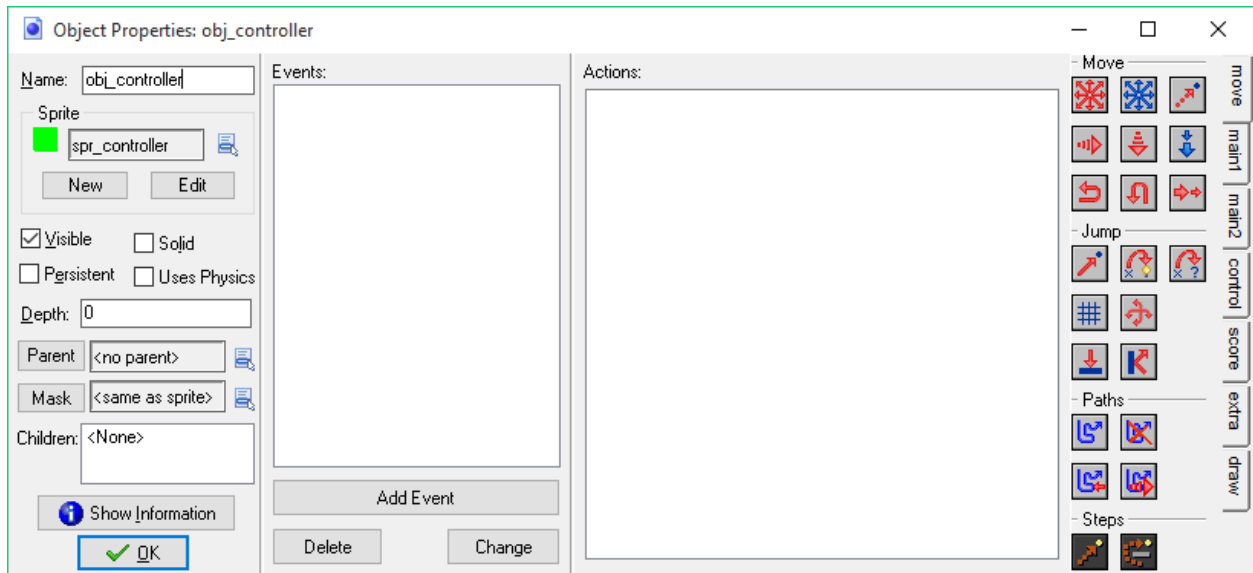
Now go back into “obj\_wall” and change the parent to “obj\_block”. This will help us later on with collision events.



Let's go ahead and put the walls in the room. Open "room0", click on the "objects" tab, and select "obj\_wall". Then, just place the wall objects like so in the room to create the shape of the Tetris playing field:

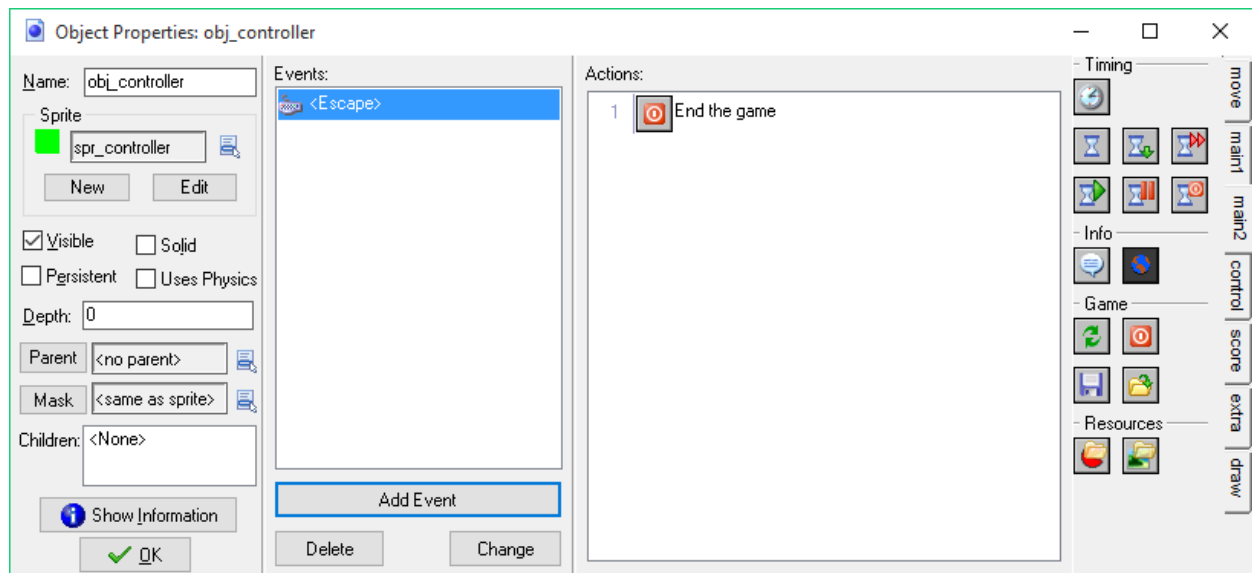


The next thing that we need is a controller object that will decide when to drop new Tetris blocks from the ceiling. Go ahead and make another new object and add it to the room:



Now, in order to be able to quit the game after we start it, we need to add an event to the controller object that will end the game. Open up “obj\_controller”, click the “Add Event” button, click on “Keyboard”, go down to “Others”, and select “<Escape>”.

Next, go down to the “main2” tab on the right of the object windows and draw the “End the game” box into the Actions window:



If you go ahead and run it, you'll notice that not much happens.

So let's do some things!

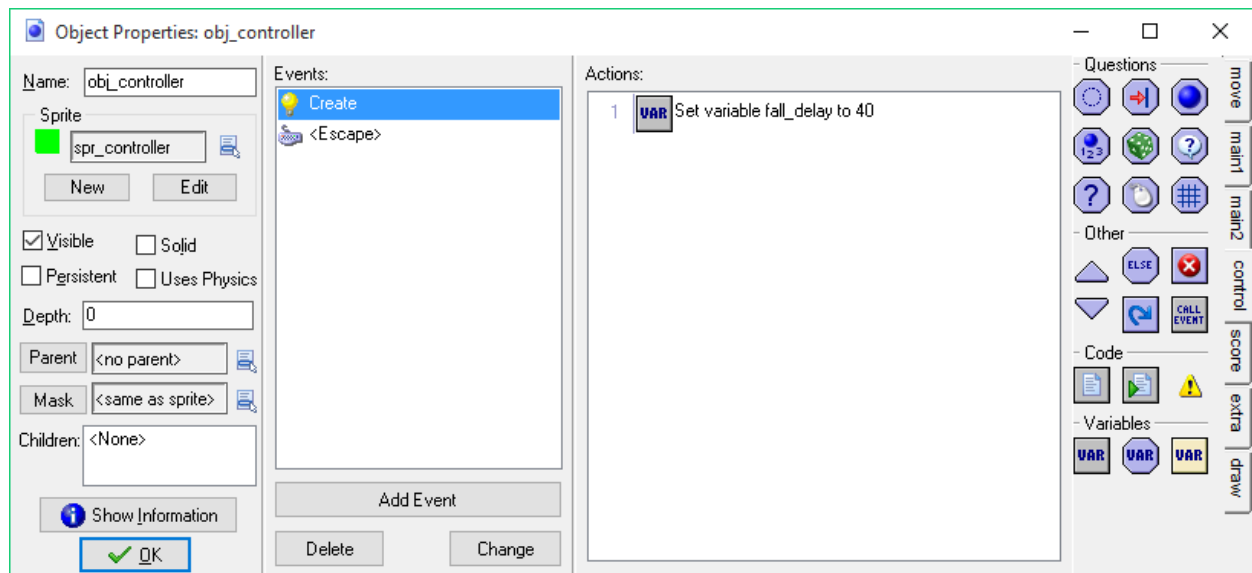
## Falling Blocks

First things first, let's make some Tetris blocks start falling from the top of the screen. Add an "obj\_tetris\_shape" object and for now, set its sprite to "spr\_tetris\_block".

Now in the controller, we need to do a bit of coding. First, let's define a few variables.

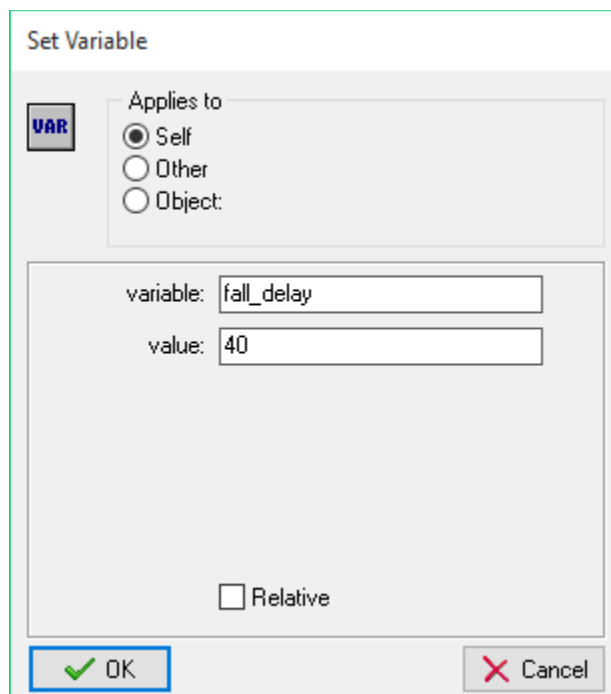
In "obj\_controller", add a "Create" event. Everything in here will happen once at the beginning of the room, so it's very good for just defining some things up front.

Go to the control tab and drag a set variable icon into the Actions window:



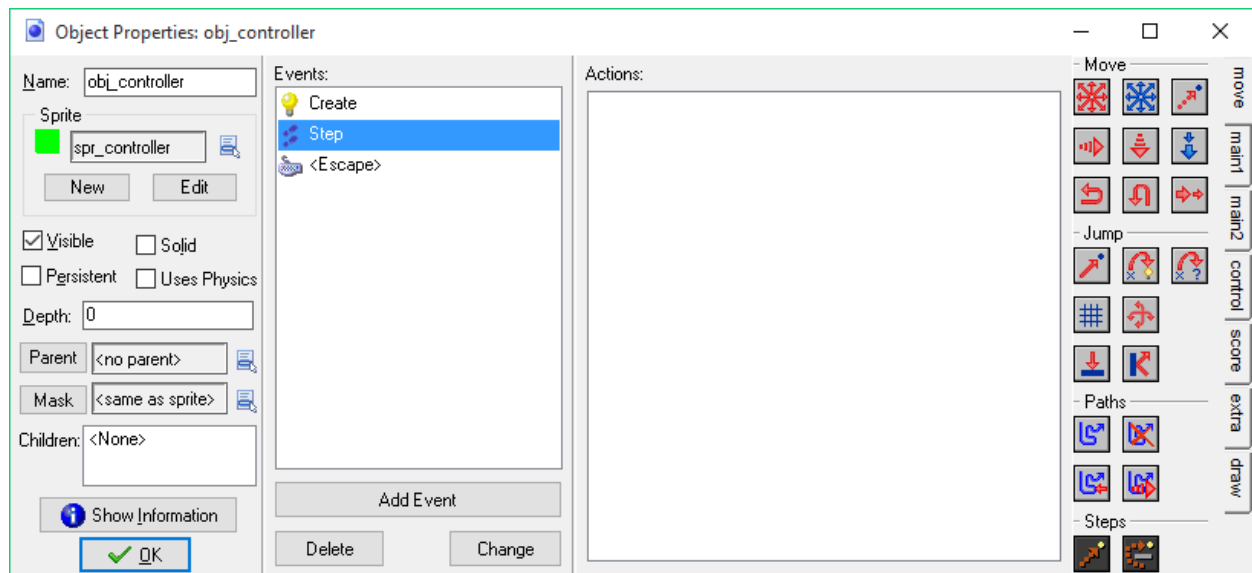
This variable will define how many steps are going to elapse between every drop of the Tetris block. So the higher the number, the slower the block will fall, and vice-versa.

To define it, just enter the variable name “fall\_delay” and the value “40” and hit OK. This will have the Tetris blocks fall one block every 2/3 of a second (remember that the game runs at 60 steps/second, so  $40/60 = 2/3$  second).

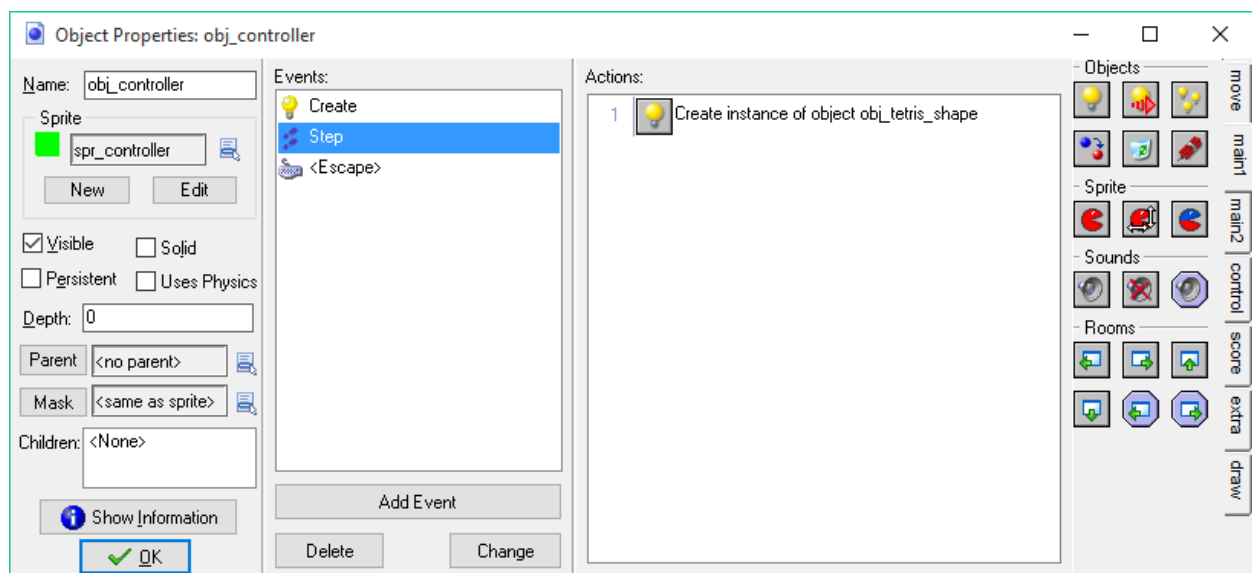


Next, we need to create the first Tetris shape! Add a “Step” event to obj\_controller.





To create a Tetris shape, go to the “main1” tab and add a create instance action:



Create Instance

Applies to

☒ Self

☐ Other

☐ Object:

object:

x:

y:

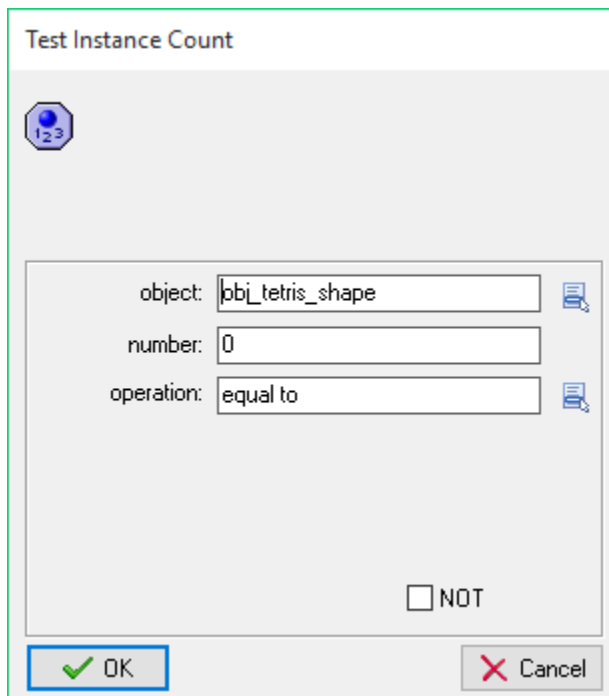
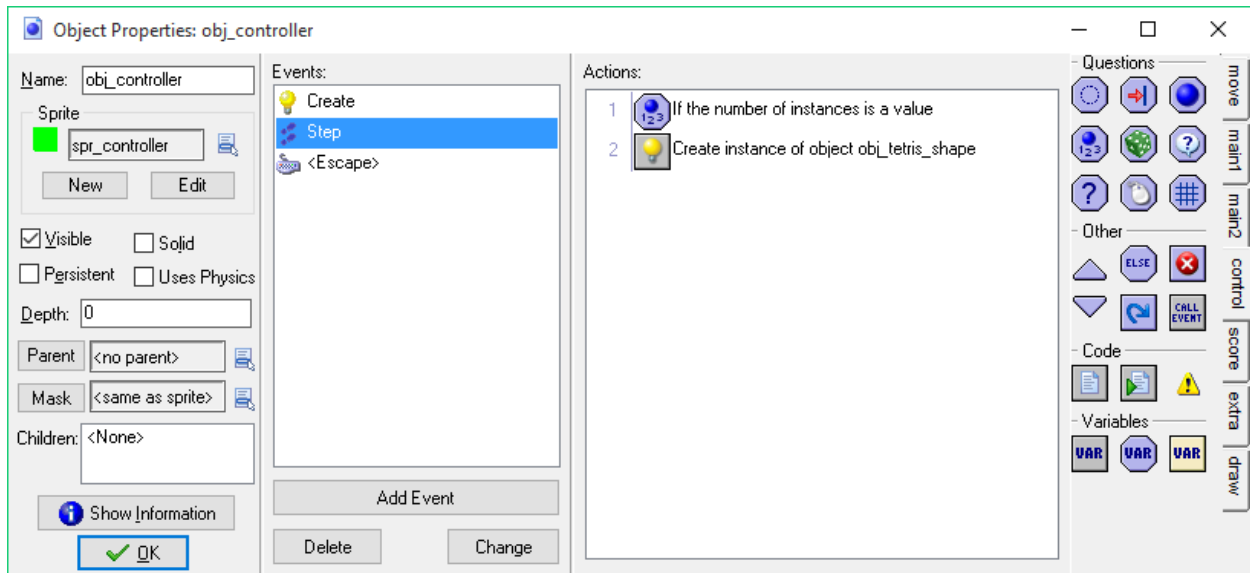
☒ Relative

OK Cancel

Set the object field to “obj\_tetris\_shape” to specify that it is a Tetris shape object that we want to create an instance of. Also, set x and y to 0 and make sure you check the “Relative” checkbox. The “Relative” checkbox specifies that this action will create the specified instance at a position relative to where the controller object is. So (0,0) relative to obj\_controller means the Tetris shape will be created right on top of the controller object.

Now in Tetris we only want there to be one block falling at any given time. So we need to add a “check” to see how many falling blocks there are currently in the room.

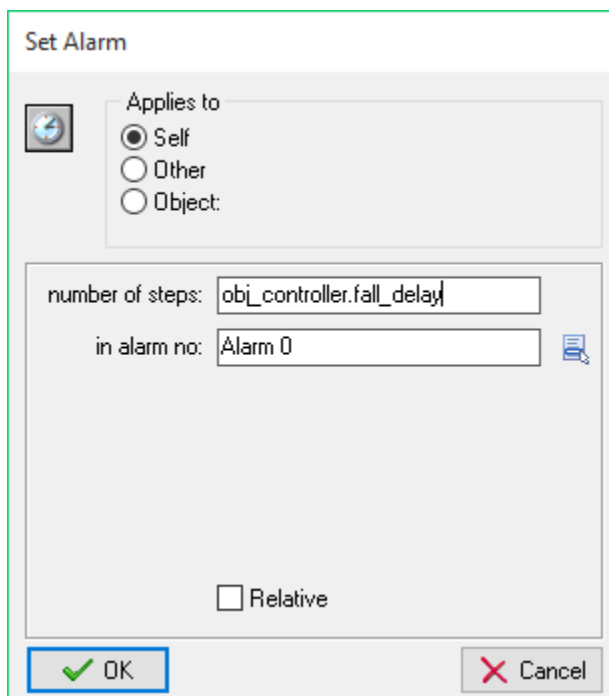
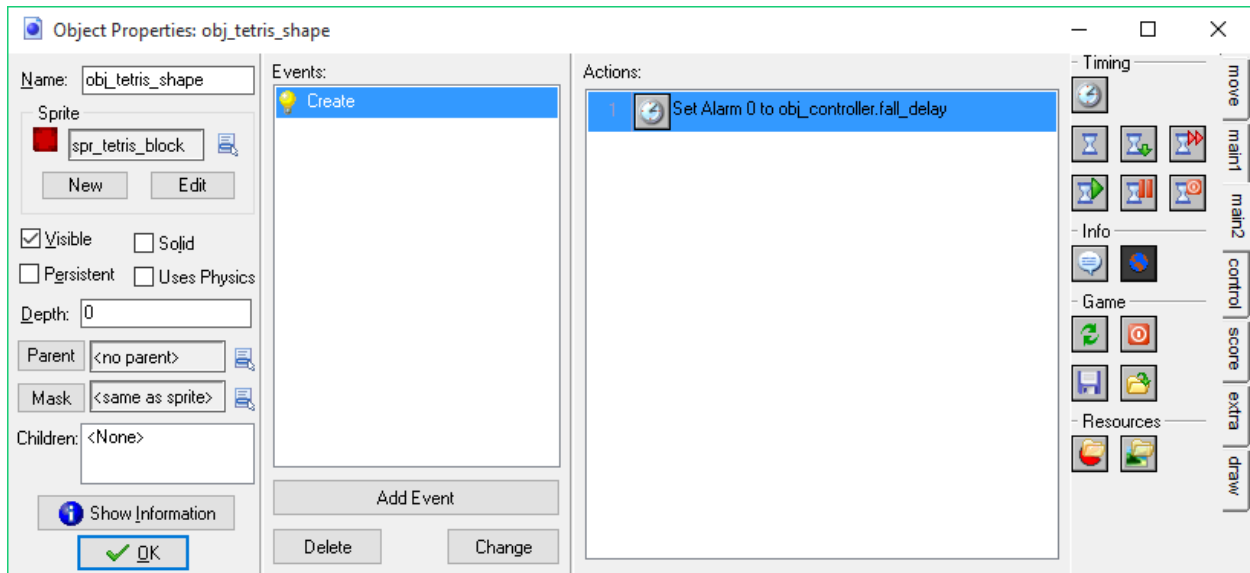
Go to the “control” tab and drag a “Test instance count” icon right above the create action:



What this action does is it checks to see if the number of instances of “obj\_tetris\_shape” is equal to 0. If it is, then the program moves on to do the next action (creating a new Tetris shape). If, however, there is more than that then the program will not make a new Tetris shape.

So far so good! However, the Tetris shape still doesn’t do too much. We need it to fall.

In “obj\_tetris\_shape”, let’s make a “Create” event for it as well. In this one, we are going to set an alarm to the value we just defined in the controller so that it will drop every 40 steps. Go to the “main2” tab and drag the alarm icon into the actions window.

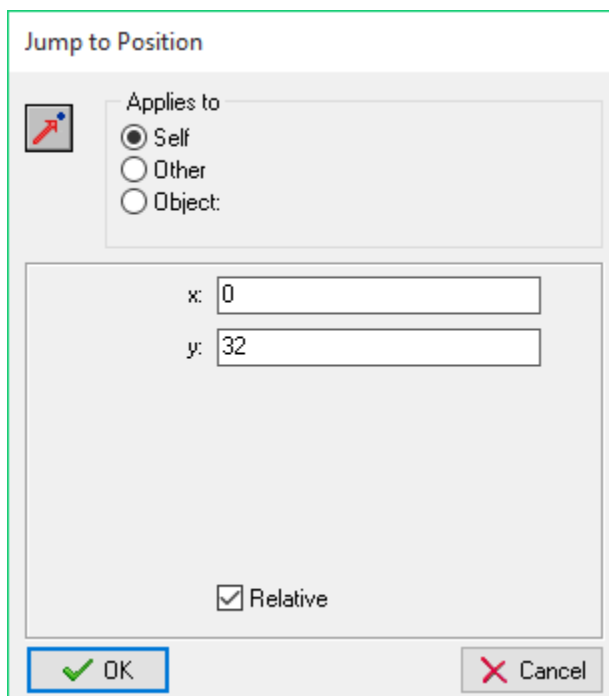
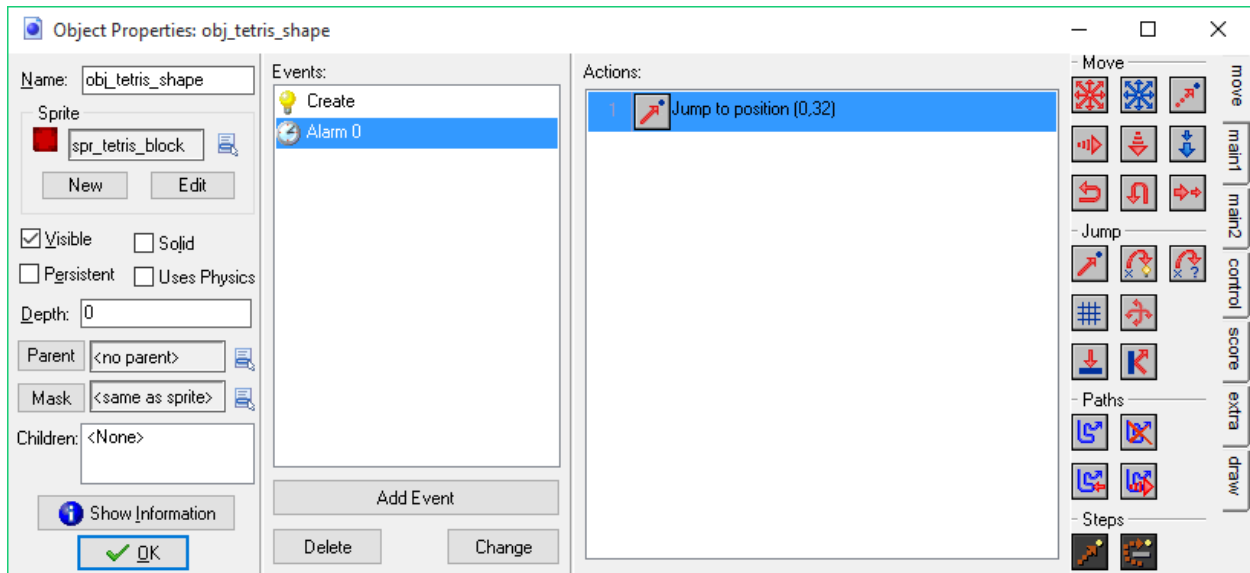


Notice anything unusual? In the “number of steps” field, we have “fall\_delay” (the name of the variable we set to define how long the alarm should be set for) prefaced with an “obj\_controller.”. This is because in order for objects to access the values of variables defined in other objects, it has to define which object it’s taking the value from. Saying “obj\_controller.fall\_delay” copies the value of “fall\_delay” in “obj\_controller” into “alarm0” of the tetris shape object.

Now let’s make something actually happen when alarm 0 goes off!

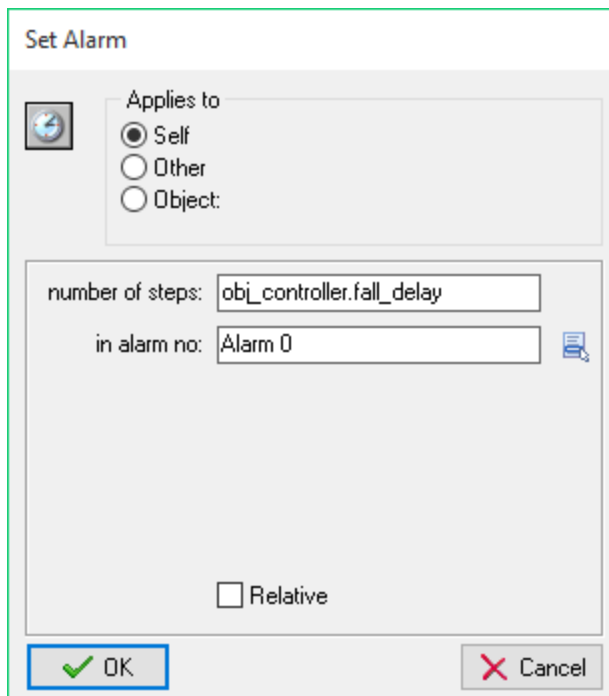
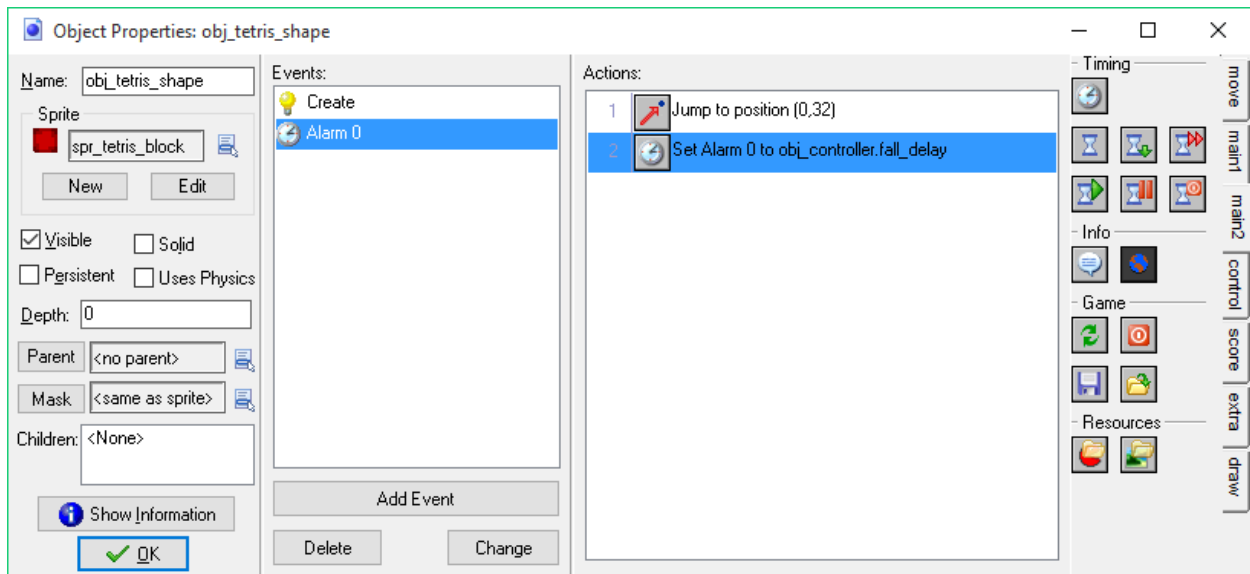
To do this, add an “Alarm 0” event to “obj\_tetris\_shape”. For this alarm we want two things to happen: first, it needs to move the Tetris shape down a block and second, it needs to reset the alarm so it will go off again in another 40 steps.

To do the first one, go to the “move” tab and drag the “Jump to position” icon into the “Actions” window.



In the settings for it, put in 0 for x, 32 for y, and make sure you check the “Relative” checkbox. Now just to reiterate, the “Relative” checkbox means that this jump action specifies where the object is going to move to relative to where it already is. So with these settings, the action will move the Tetris shape 0 pixels in the x (horizontal) direction and 32 pixels down in the y (vertical) direction *from where it is currently*. If the relative checkbox is left unchecked, it will move the object to the *absolute position* of (0,32).

Next, add another “Set Alarm” command. This command should be specified exactly the same as the one in the create event, so have it set alarm 0 to “obj\_controller.fall\_delay” so that it will be consistent.

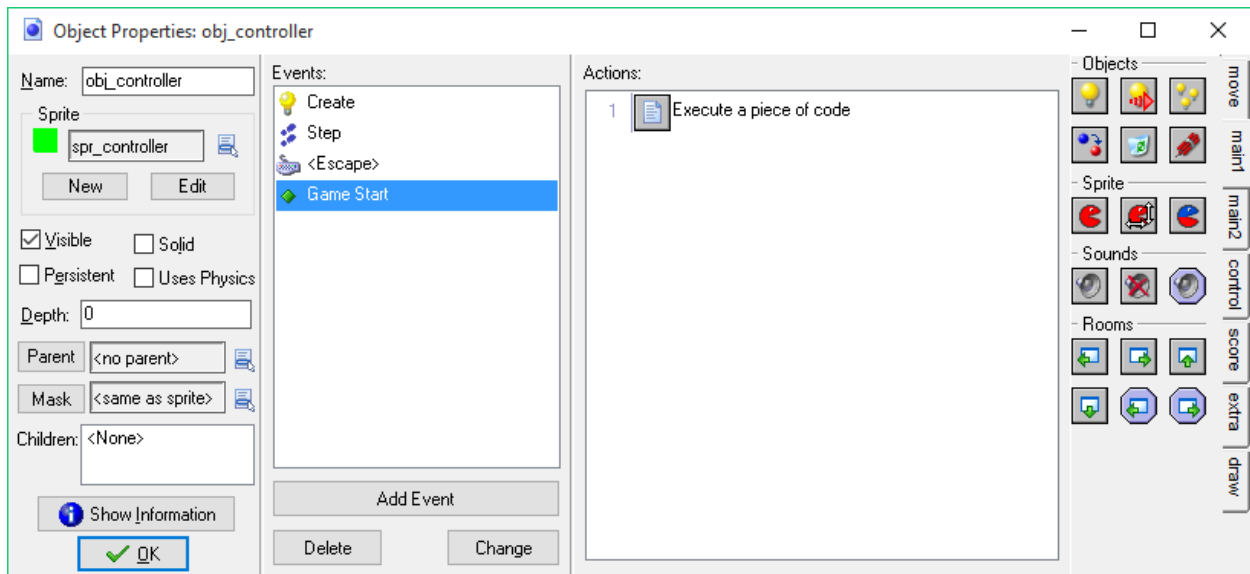


## Tetris Shapes

In Tetris, there are no single block shapes. However, there are several shapes made up of four smaller square pieces. These shapes are randomly selected and dropped throughout the game.

In order to implement this, each subimage of spr\_tetris\_shape is actually a different shape.

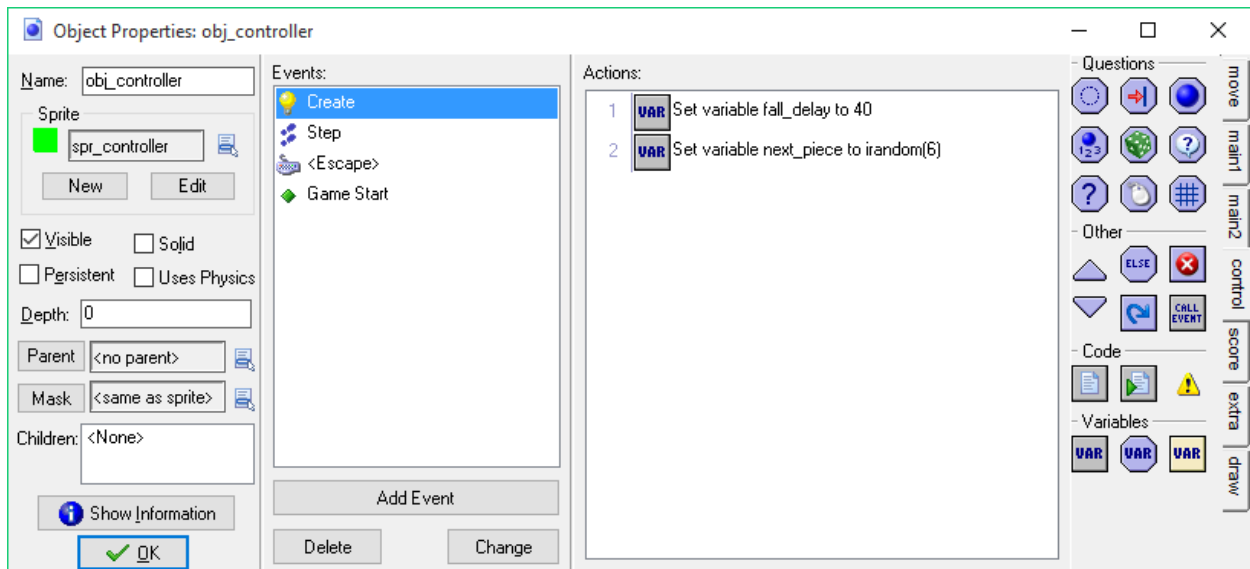
To have the controller randomly select a piece, first we have to randomize the random seed for the game. This is done by adding a “Game Start” event and adding a short “execute code” block from the control menu:

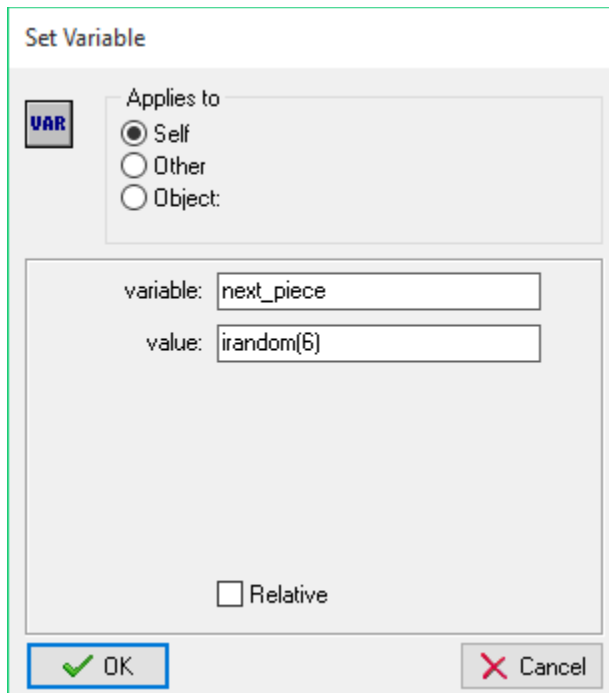


```
randomize();
```

This will allow the game select a *different* order of pieces every time the game is played.

We also need to initialize a variable to store what the next piece we are going to choose actually is. This is done in the create event:

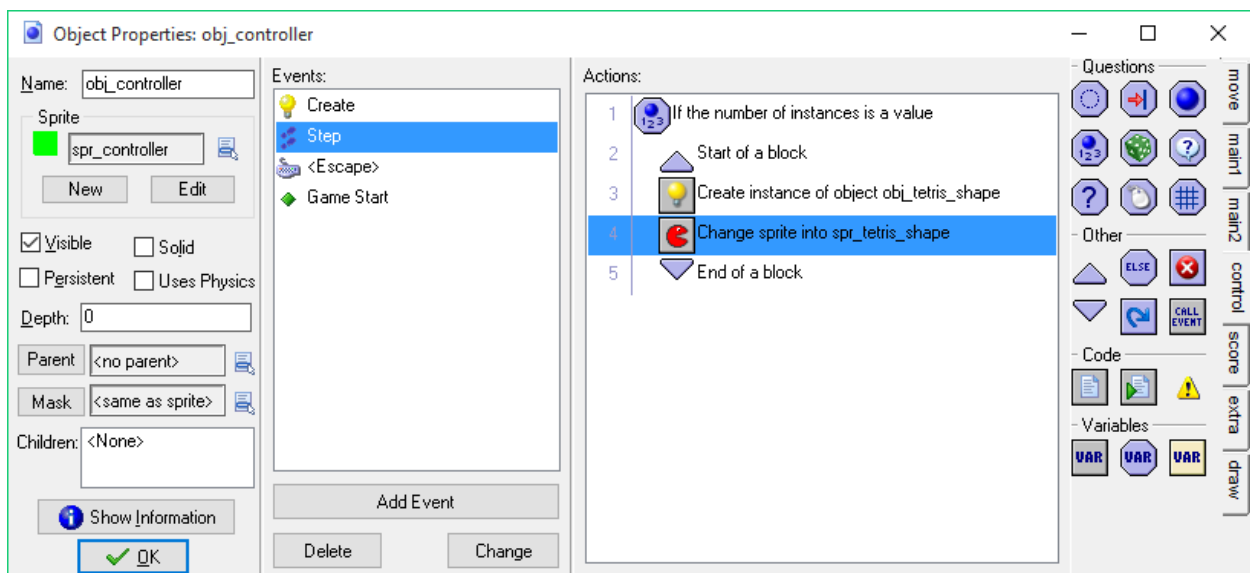




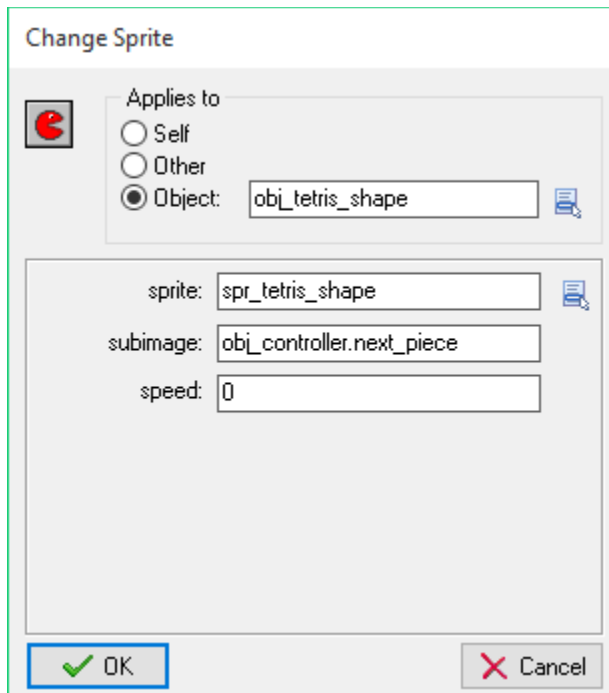
Note that the “irandom(6)” generates a random integer between 0 and 6, so this is exactly the number of shape types there are in Tetris, and therefore all we need.

Now to actually assign the random piece type to the Tetris piece, we need to add two things to the block in the step event. First, we need to assign the sprite to stay on a specific subimage. Then, we need to randomize the next piece to be chosen.

To assign the sprite, we add a set sprite action to the step event inside the block:



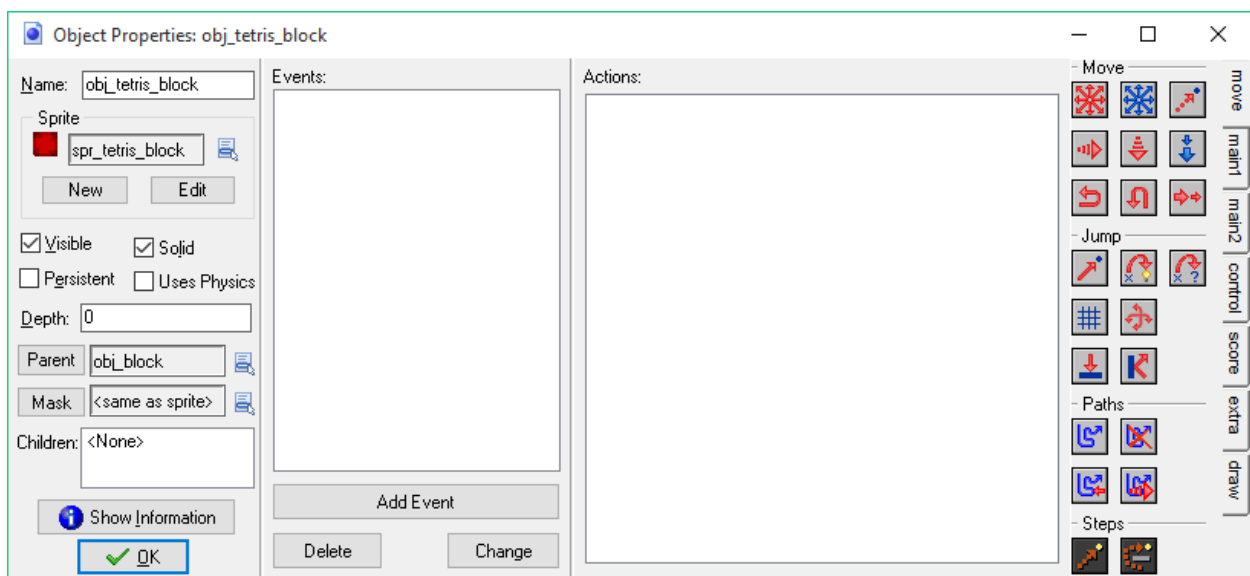




## Hitting the Ground

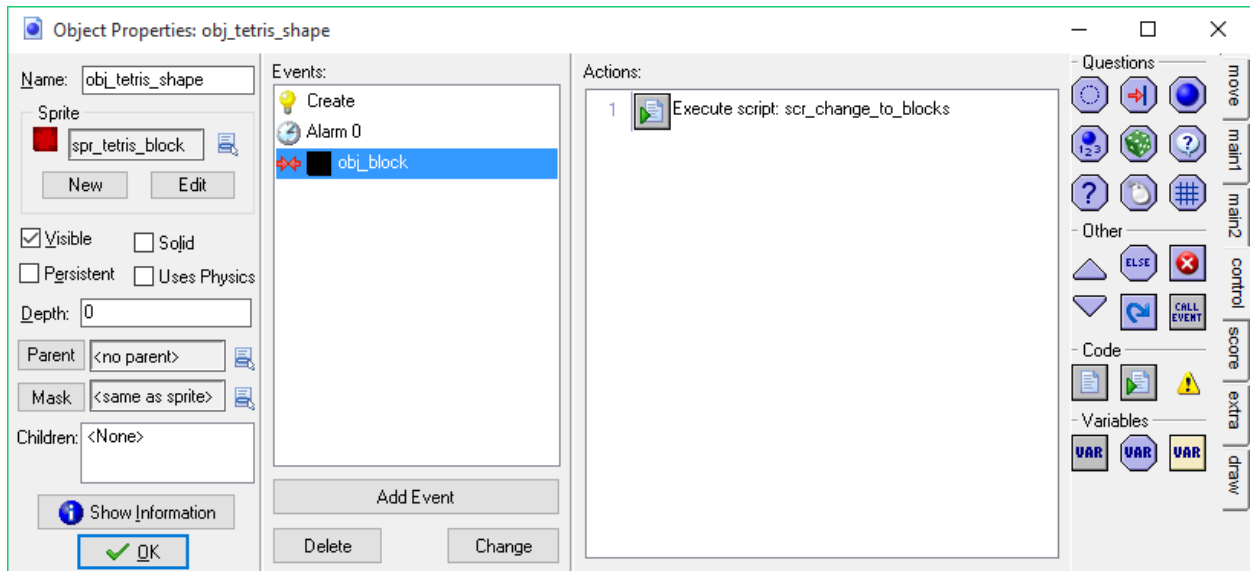
Next on this list, we need to have the Tetris shape stop falling when it hits the ground. To do this, we're going to have a different object that represents the Tetris shape after it has stopped moving.

Create an object called "obj\_tetris\_block". Set its sprite to "spr\_tetris\_block" and make sure for this object, you check the "Solid" checkbox so that falling Tetris shapes in the future will know that it is actually a solid object. You also have to make sure that it has "obj\_block" as its parent so that the shape will properly detect collisions with it.



For this next part, rather than using the drag-and-drop coding style, we are going to actually code up a few scripts to help handle the converting of Tetris shapes into individual block pieces. Create a new script by right-clicking on the “Scripts” folder and selecting “Create script”.

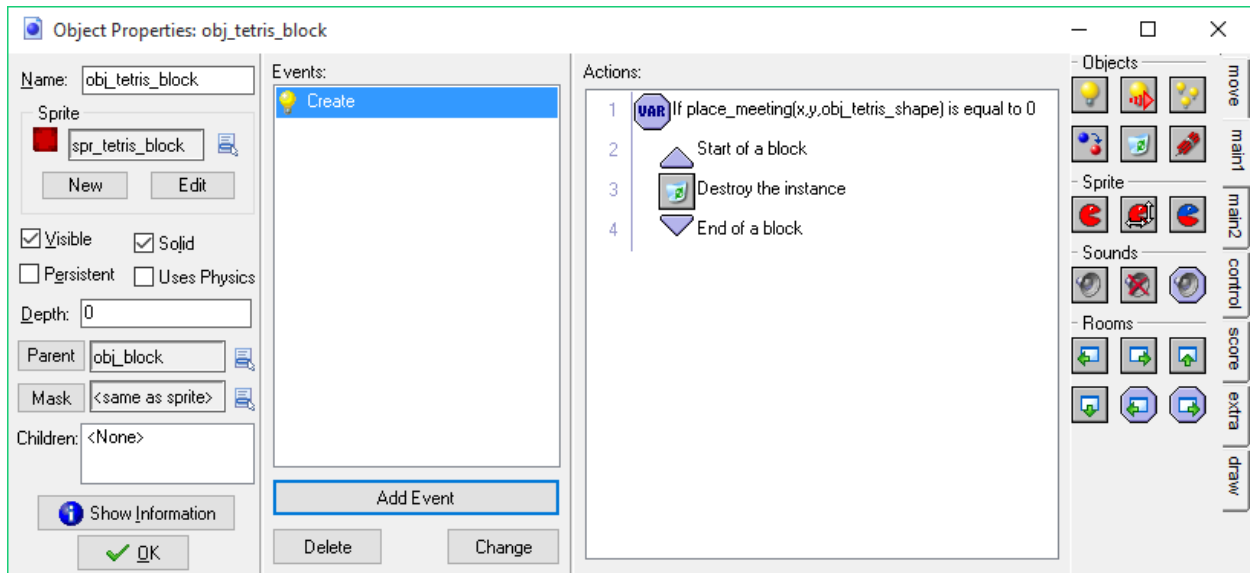
Name the script “scr\_change\_to\_blocks”. In obj\_tetris\_shape, add a script execution action from the control tab to a collision event with obj\_block:



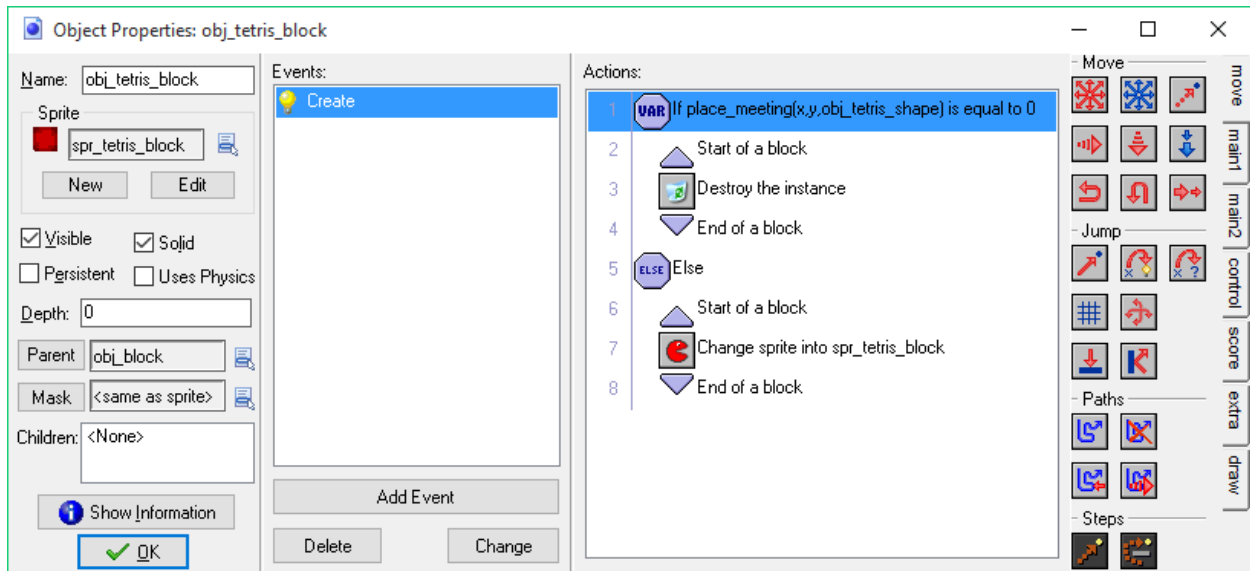
Now for the actual script. What the script is essentially going to do is create a Tetris block at all the positions that are on top of and directly surrounding the Tetris shape:

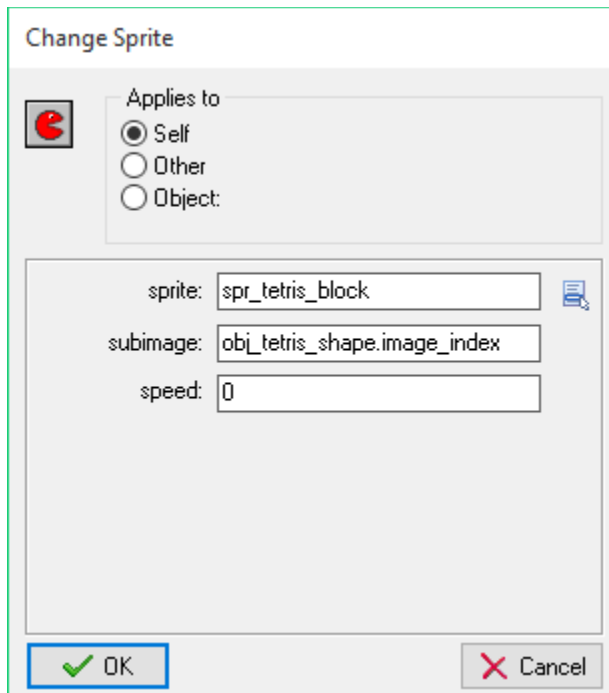
```
for(xx = -2; xx<=1; xx = xx+1) {
    for(yy = -2; yy<=1; yy = yy+1) {
        instance_create(x+xx*32,y+yy*32,obj_tetris_block);
    }
}
instance_destroy();
```

Next we need obj\_tetris\_block to check to see whether it is actually part of the shape or not. This is done by using a variable checker action in the create event of obj\_tetris\_block to check the expression `place_meeting(x,y,obj_tetris_shape)` which, basically just checks to see if there is a collision where the object currently is. If there is not, then it destroys the block.



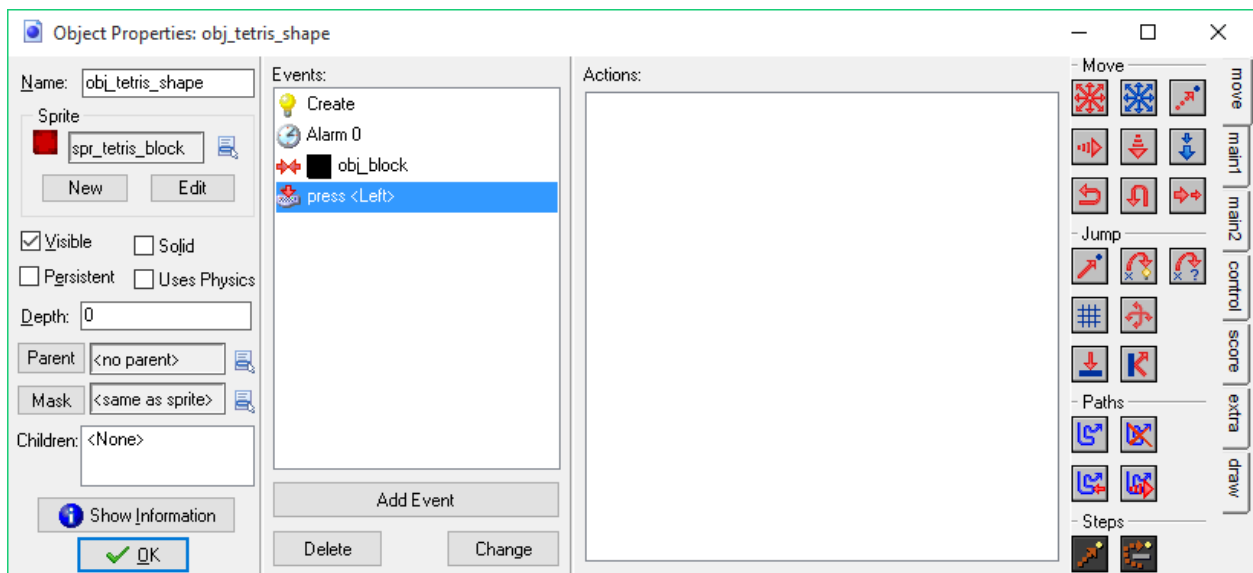
If, however, there is a collision with obj\_tetris\_shape, we need to change the sprite so that it is the same color as the animation. This is done by adding an else statement and a sprite\_set action:



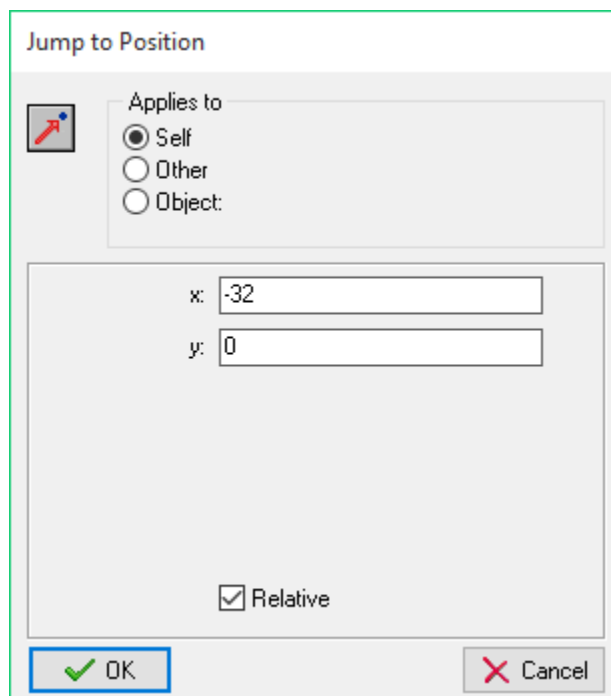
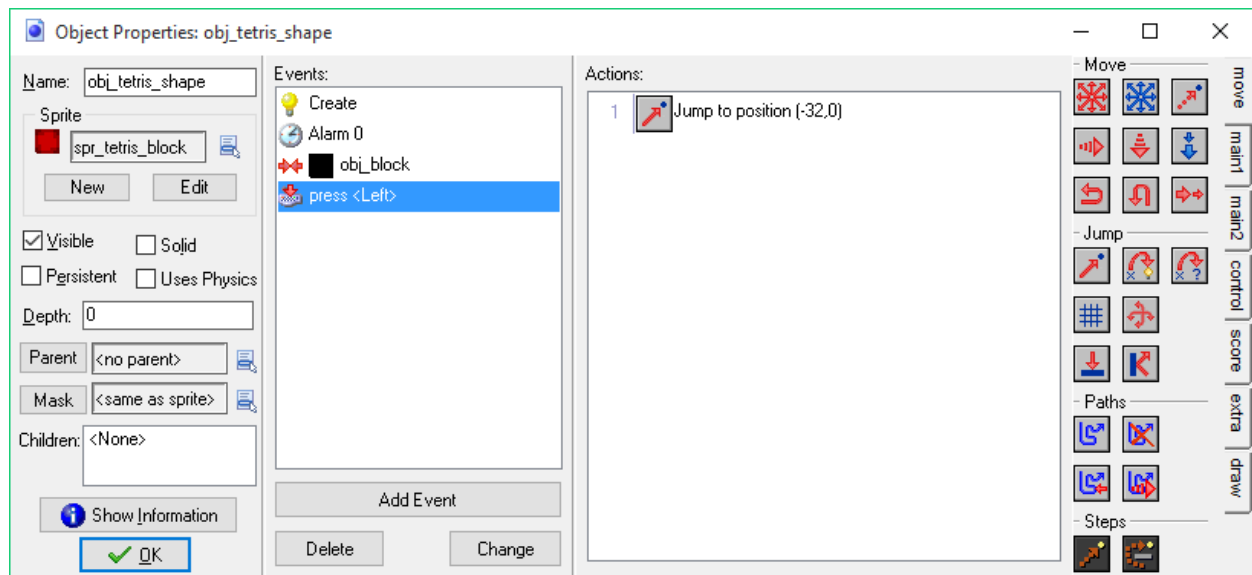


## Controls

In Tetris, you move the pieces side to side by pressing either left or right. In order to implement this, let's add a keyboard "Press Left" event to `obj_tetris_shape`.

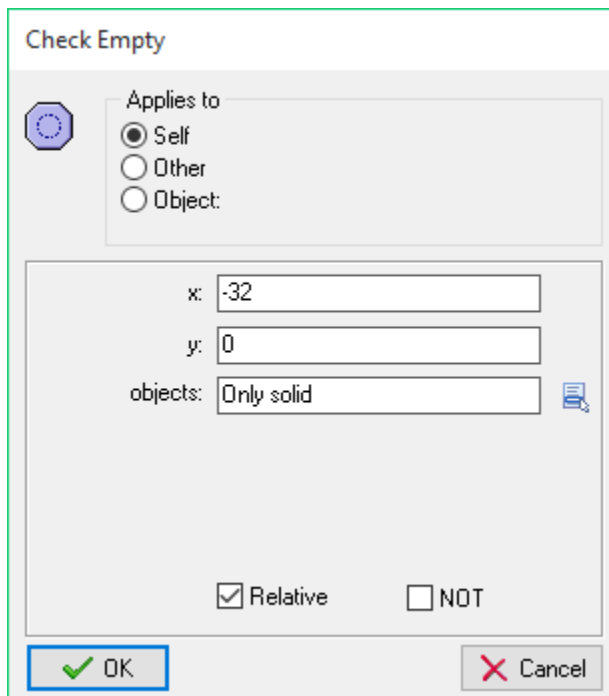
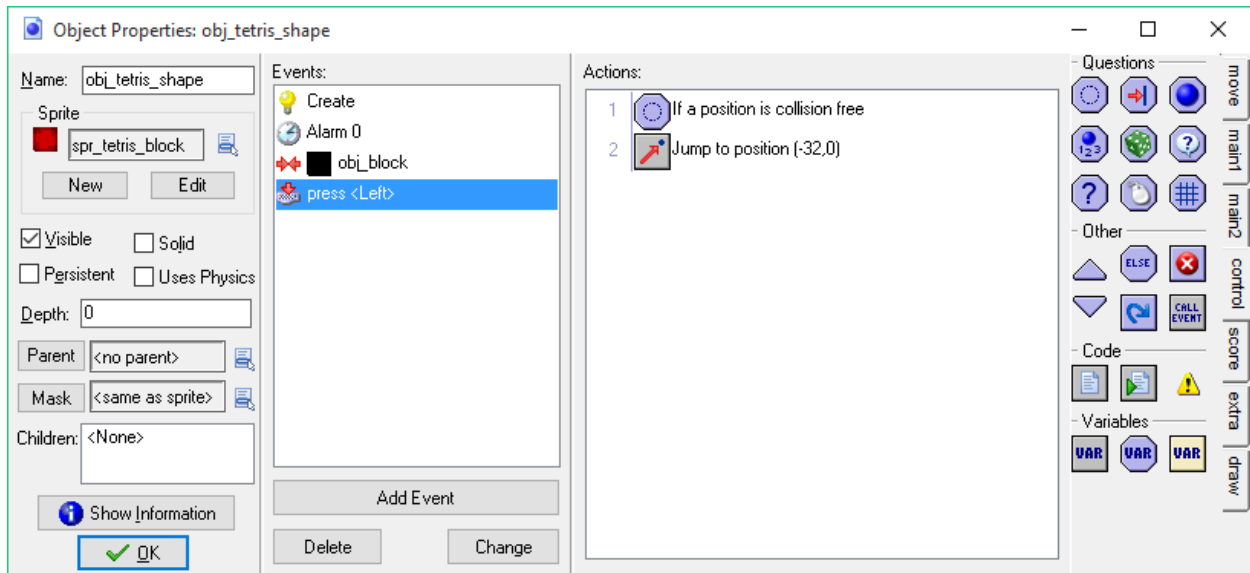


In order to make the piece move left, add a jump action from the "move" tab.



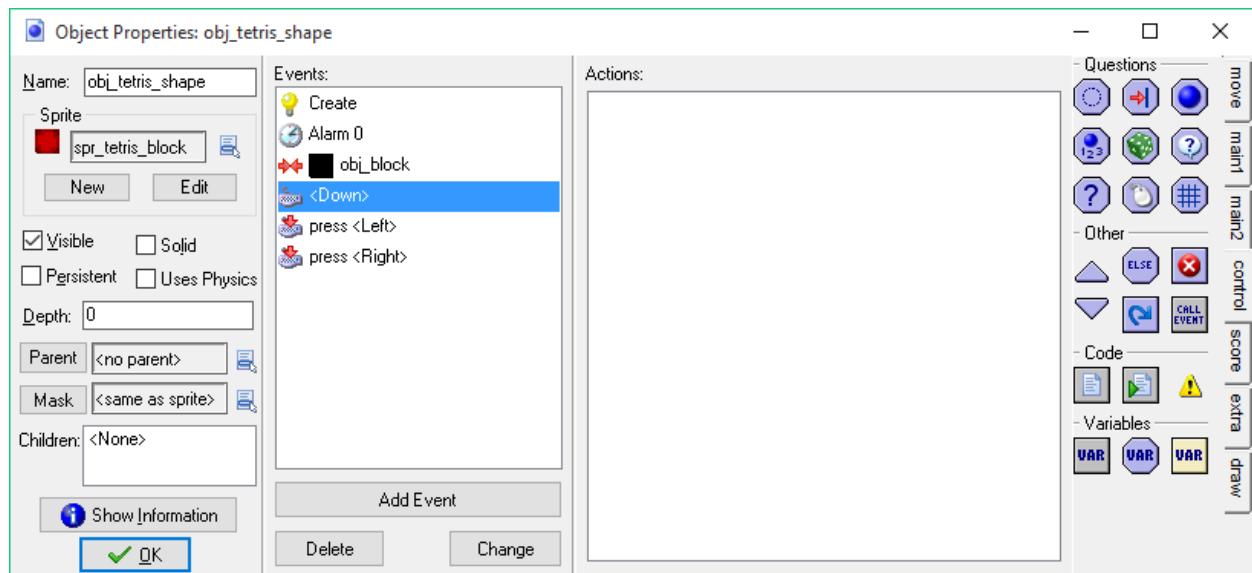
However, just because we press the left button doesn't mean that the piece should necessarily move left. If there is a block in the way, it should stay where it is.

To fix this, add a "Check Empty" event for the space immediately to the left.

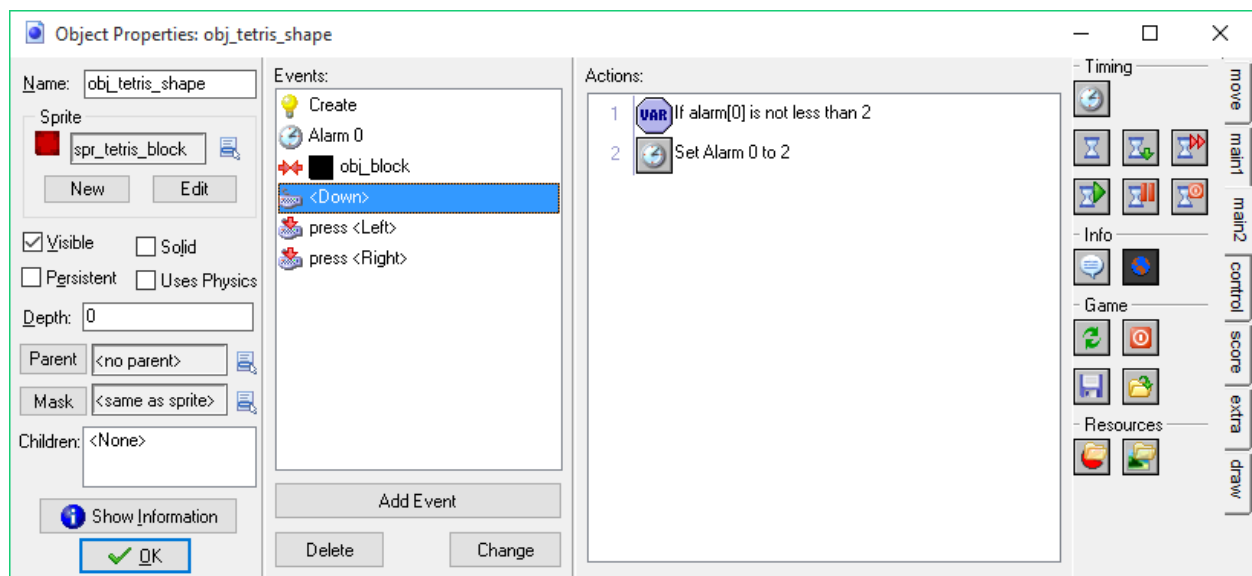


The same basic approach applies also to the right key. Right-click on the <Press Left> event and select “Duplicate” to <Press Right>. The only thing that needs to be changed is the -32 to 32.

Also, when you press the down arrow the piece should fall faster (so you don’t have to wait for so long when the fall speed is slow). Add a keyboard “Down” event (not pressed):



The way this will work is holding the down arrow will set alarm 0 to 2 if alarm 0 is anything higher than 2:



### Test Variable

VAR

Applies to  
☒ Self  
☐ Other  
☐ Object:

variable:   
value:   
operation:

☒ NOT

### Set Alarm

Applies to  
☒ Self  
☐ Other  
☐ Object:

number of steps:   
in alarm no:

☐ Relative

## Rotation

Having the pieces spin is another slightly more complicated portion of the game. Therefore, we are going to use a GML script once again to handle these equations.

In `obj_tetris_shape`, add an X keyboard press event and drag an empty code action into it from the control tab:

#### Object Properties: obj\_tetris\_shape

Name:

Sprite

☒ Visible
☐ Solid

☐ Persistent
☐ Uses Physics

Depth:

Parent:

Mask:

Children:

Events:

Create

Alarm 0

obj\_block

<Down>

press <Left>

press <Right>

press X-key

Actions:

Execute a piece of code

Questions

Other

Code

Variables

The actual code consists of a for loop that checks to make sure that when the pieces are rotated, they are not rotated in such a way that they end up overlapping one another:



```

image_angle-=90;
for (var i = 0; i < 3; i++) {
    if (place_meeting(x,y,obj_block)) image_angle-=90
    else break;
}

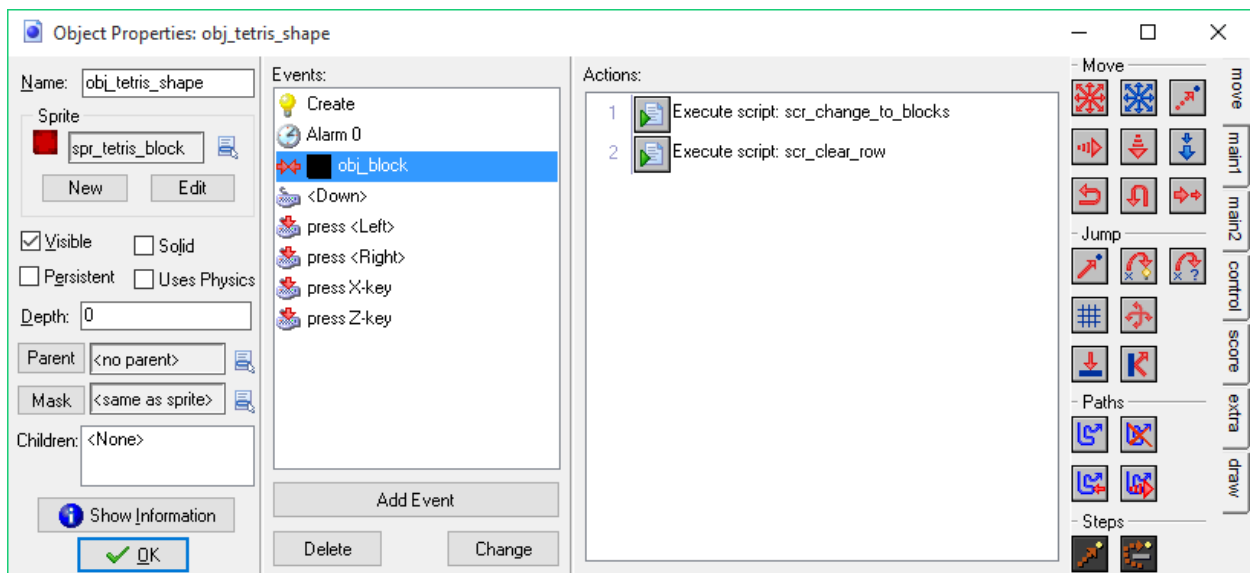
```

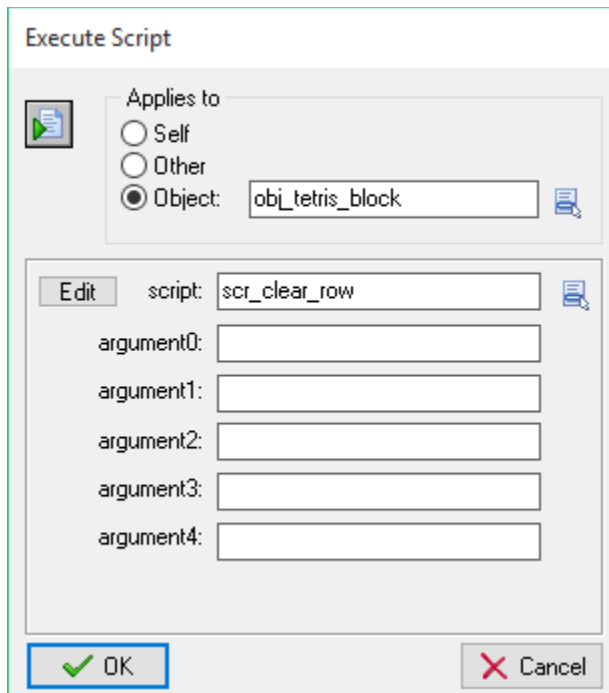
This code rotates the object clockwise. To change it for the Z key, you just have to change the -=90 to a +=90.

## Row Removal

In order to remove the rows when they are completely full, obj\_tetris\_shape will call a script *on behalf of* obj\_tetris\_block. This means that all of the instances of obj\_tetris\_block will each execute the script.

Create another script called scr\_clear\_row and add the script calling action at the end of the obj\_block collision event in obj\_tetris\_shape.





The code for `scr_clear_row` consists of essentially three parts. First, it checks to the left for holes in the row until it reaches a wall object. Then it does the same thing to the right, and if it didn't find a hole then the row can be cleared.

To clear the row, it tells all of the blocks above this row to jump down one block and all of the blocks in this row to destroy themselves.

```
var xx = x, yy = y, row_full = true;
// check to the left
while (place_meeting(xx-32,y,obj_wall)==false) {
    xx -= 32;
    if (place_meeting(xx,y,obj_tetris_block)==0) {
        row_full = false;
        break;
    }
}
// check to the right
xx = x;
while (place_meeting(xx+32,y,obj_wall)==false) {
    xx += 32;
    if (place_meeting(xx,y,obj_tetris_block)==0) {
        row_full = false;
        break;
    }
}
// if the row is full, clear it
if (row_full == true) {
    with obj_tetris_block {
```

```

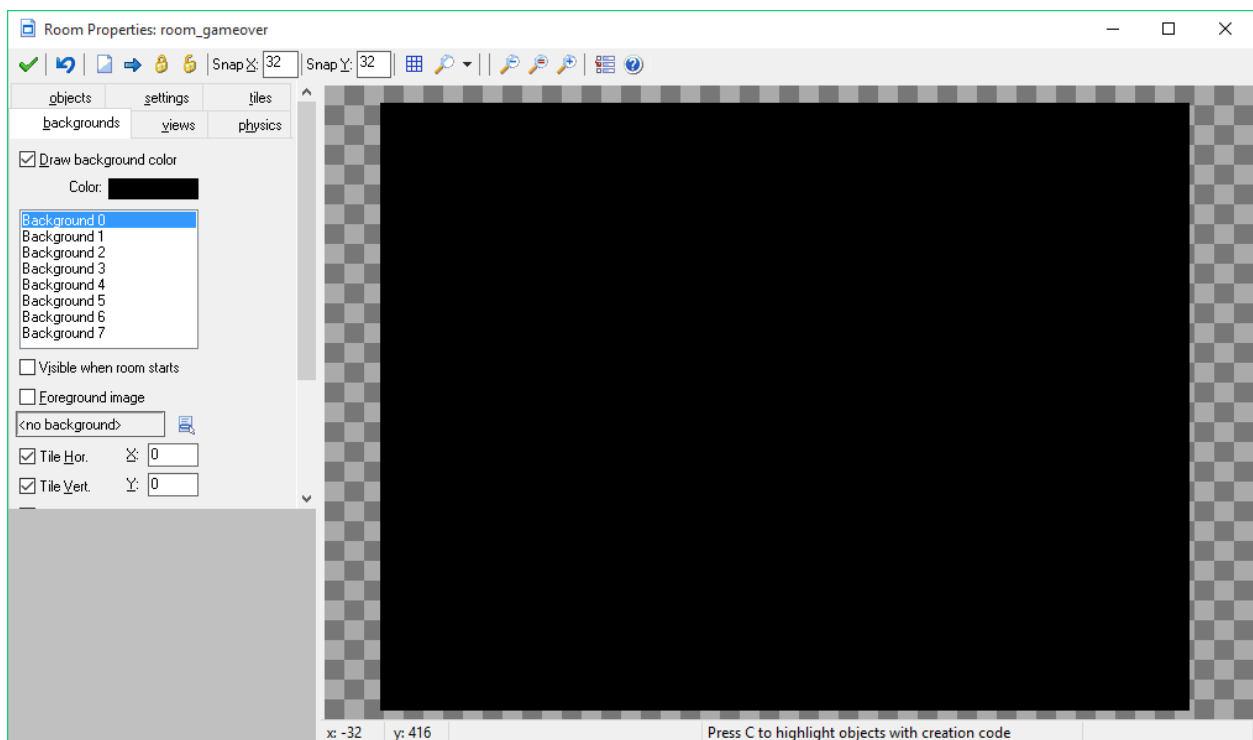
        if (y==yy) instance_destroy()
        else if (y<yy) y+=32;
    }
}

```

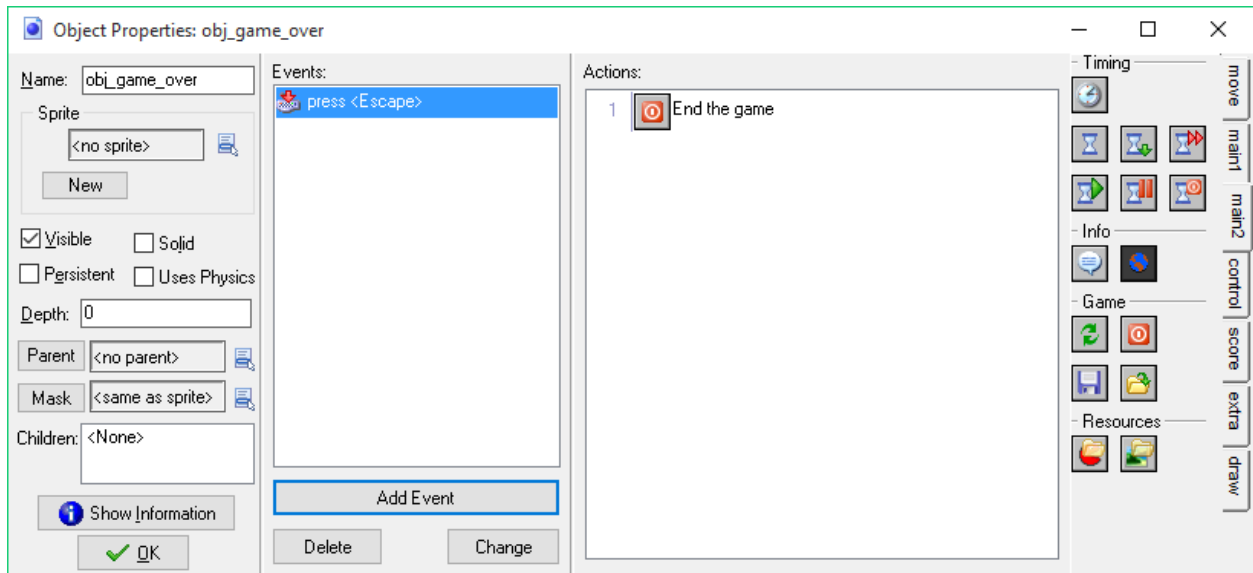
## Game Over Screen

The only thing left for the bare-bones implementation of Tetris is to have a Game Over Screen show up when the screen gets completely filled up and you lose. We will take care of this by having a second room that the game will go to.

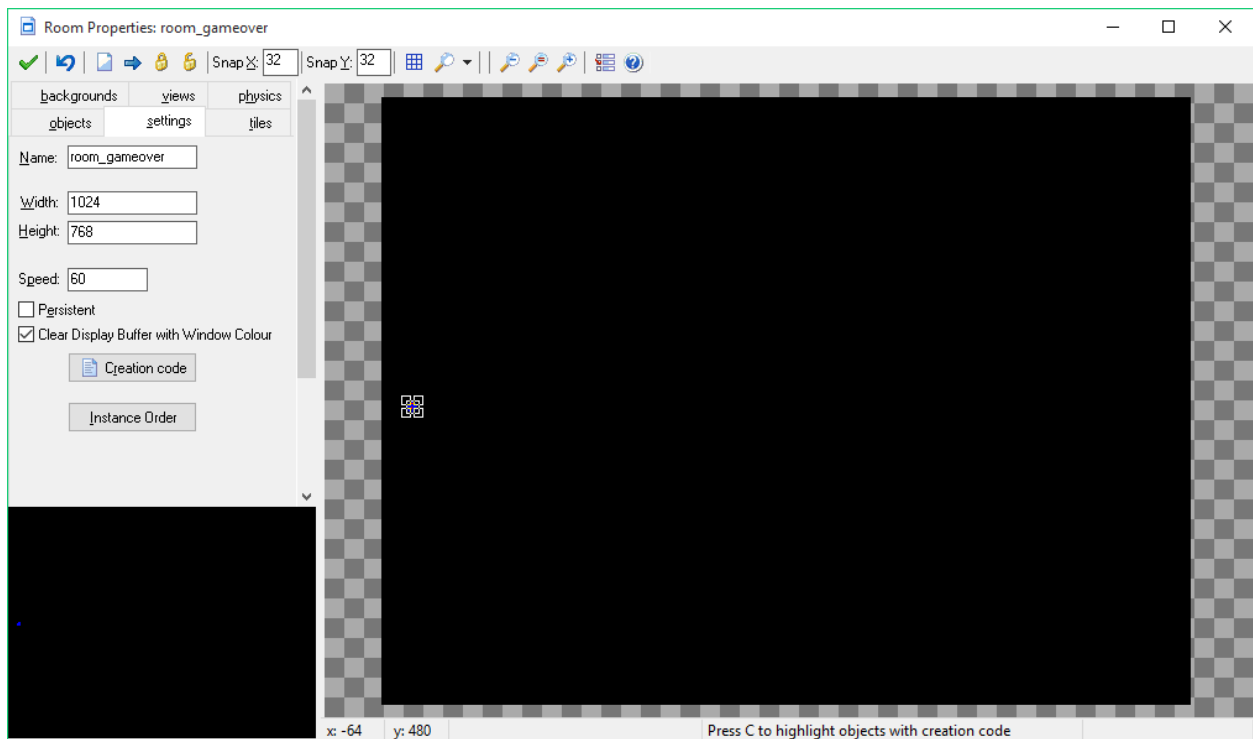
Make a new room called “room\_gameover” and be sure to set the speed to 60 again for this room. Also, change the background of the room to black by clicking on the backgrounds tab and clicking on the color box to select a different color.



Now to have it actually display the Game Over message, we need to create a new object. Call it obj\_game\_over, and once again we need to add an “Escape” key event that will end the game so that we can exit out if something is wrong.

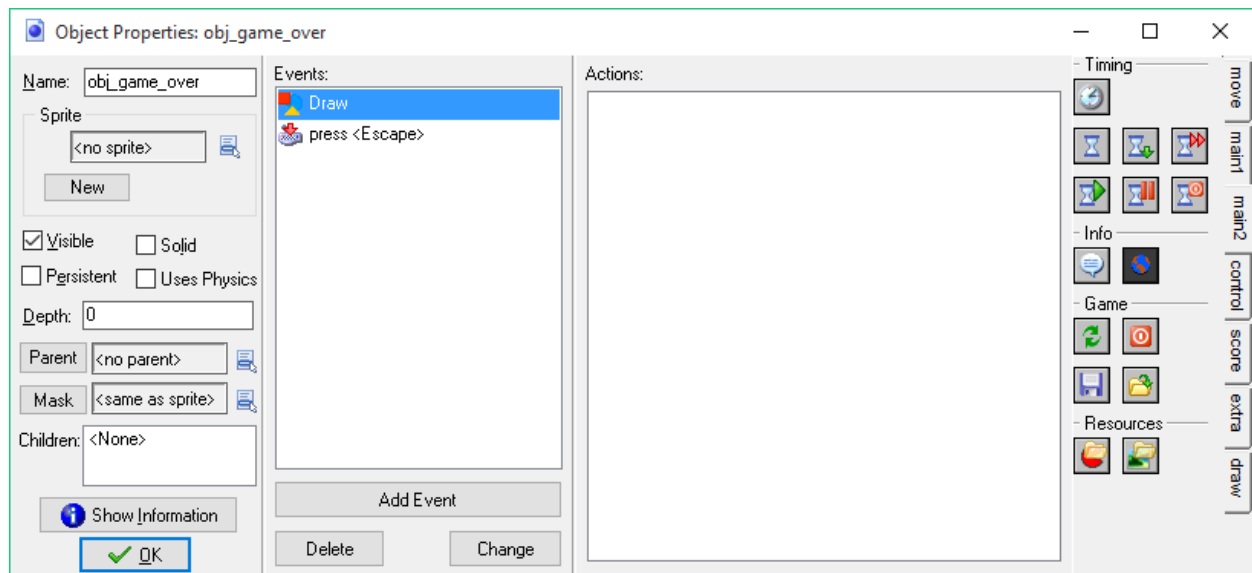


Open up room\_gameover again and add obj\_game\_over to it (the location doesn't matter).

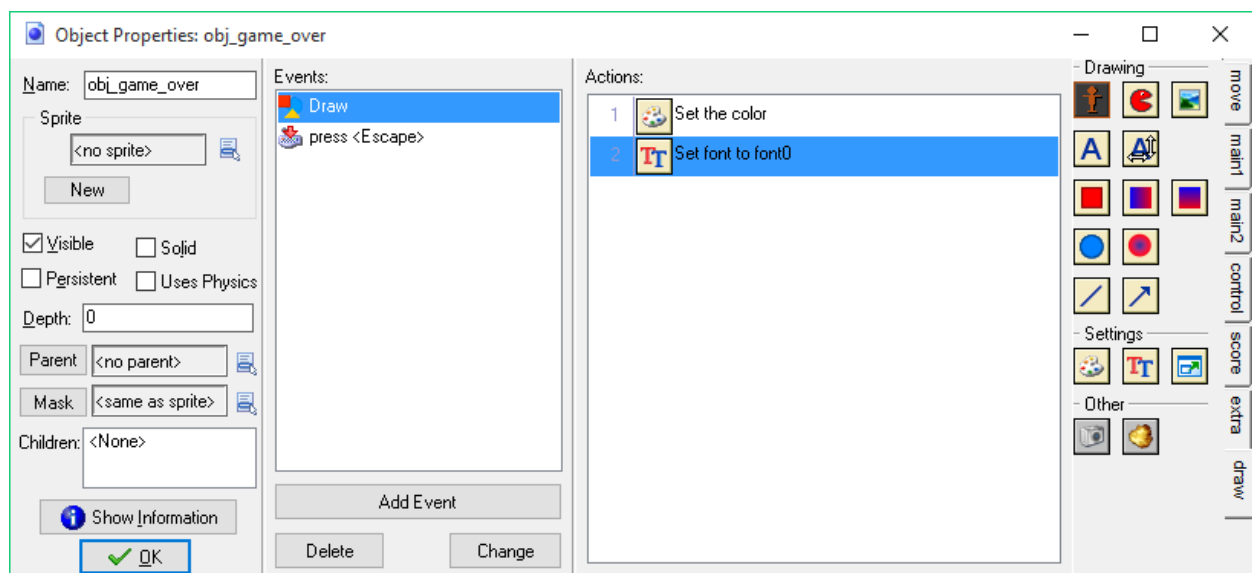


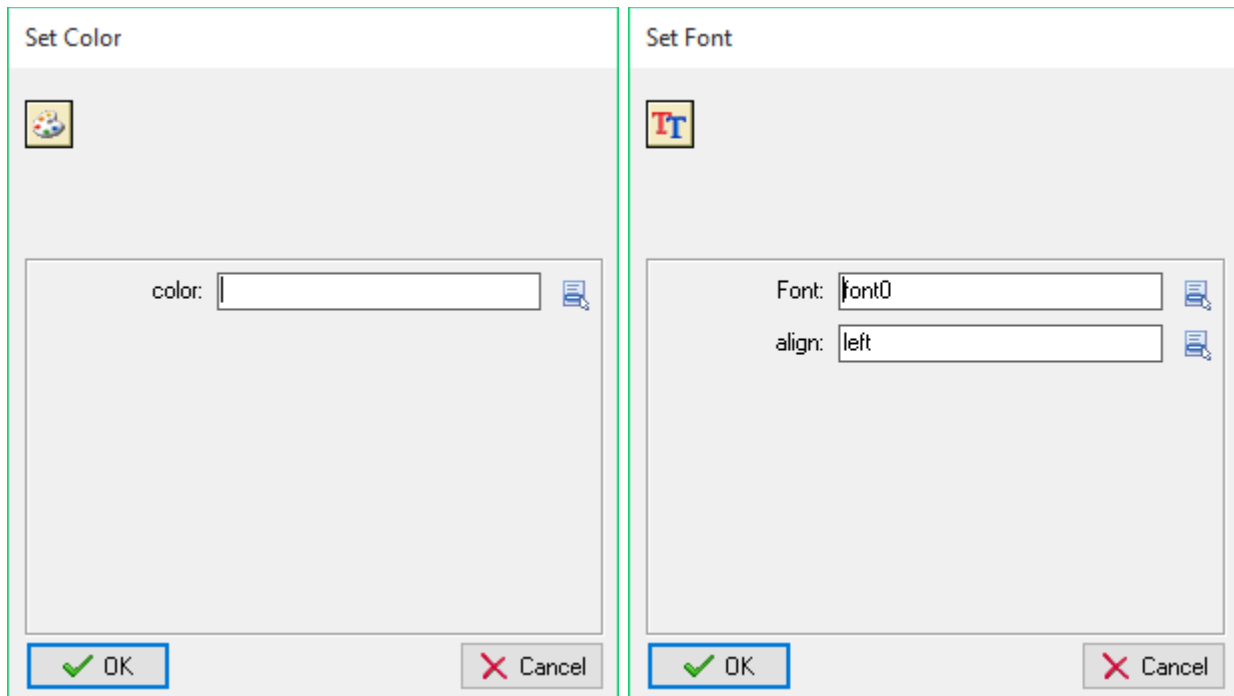
You can specify the font that text will be displayed in by right clicking on the "Fonts" folder and hitting "Create font". Change the font to "OCR A Extended", the size to 24, and mark the "Bold" checkbox.

Now to actually display the message, we need to add a draw event to obj\_game\_over:

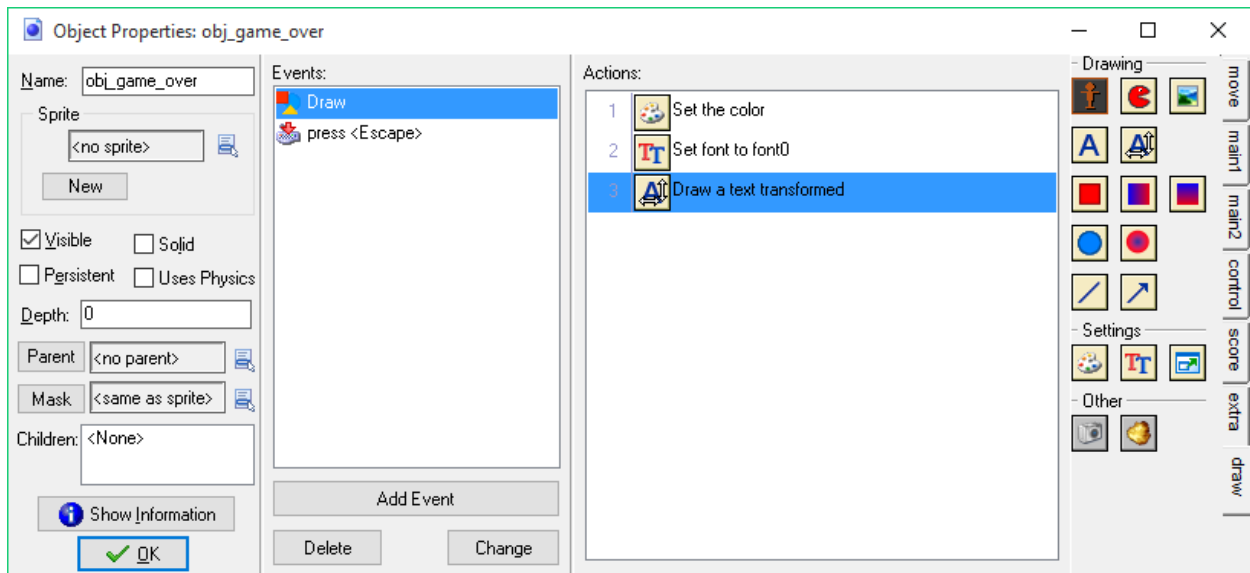


Set the font and draw color by adding the appropriate actions from the draw tab:



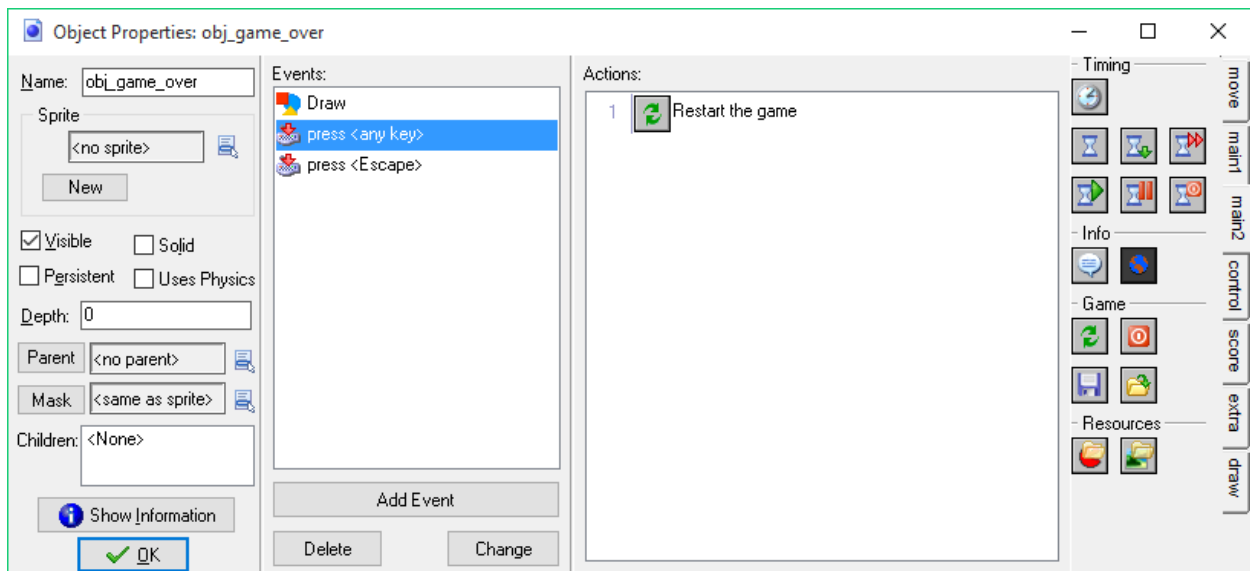


Then display the message by drawing the test:



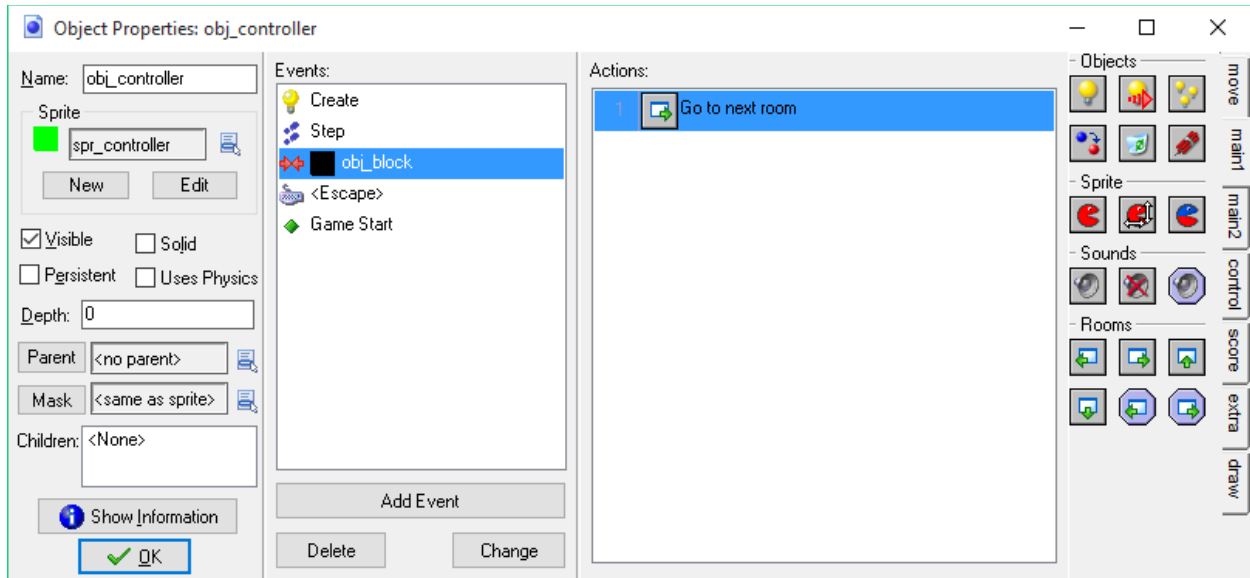


Finally, add a “Keyboard Pressed <Any Key>” event and have it restart the game so that you can continue playing even after you die.



The only thing that is left now is to set up the trigger to send the game from the starting room to room\_gameover.

In obj\_controller, add a collision event with obj\_block and have it take the user to the next room.



And that is it—the basic implementation!

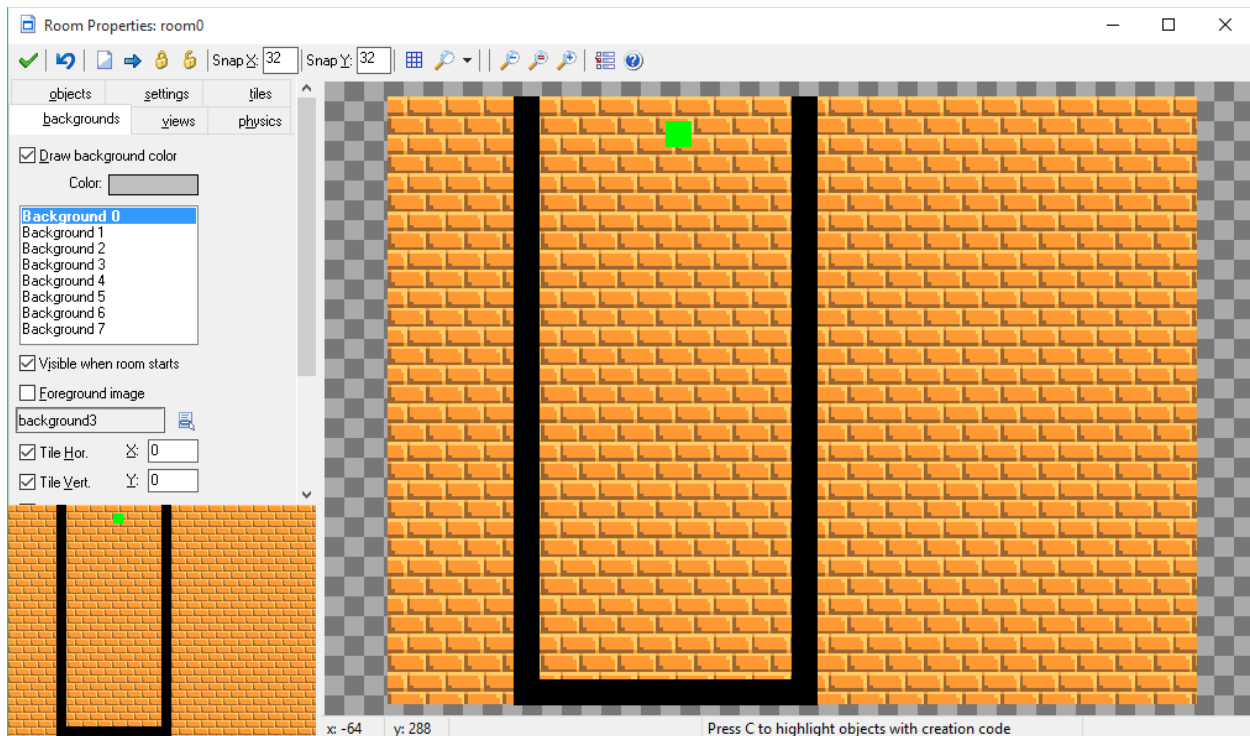
## Refinements

Although everything above will give you a working Tetris game, there are still quite a few things that could make the game nicer. They just take a little more time to implement.

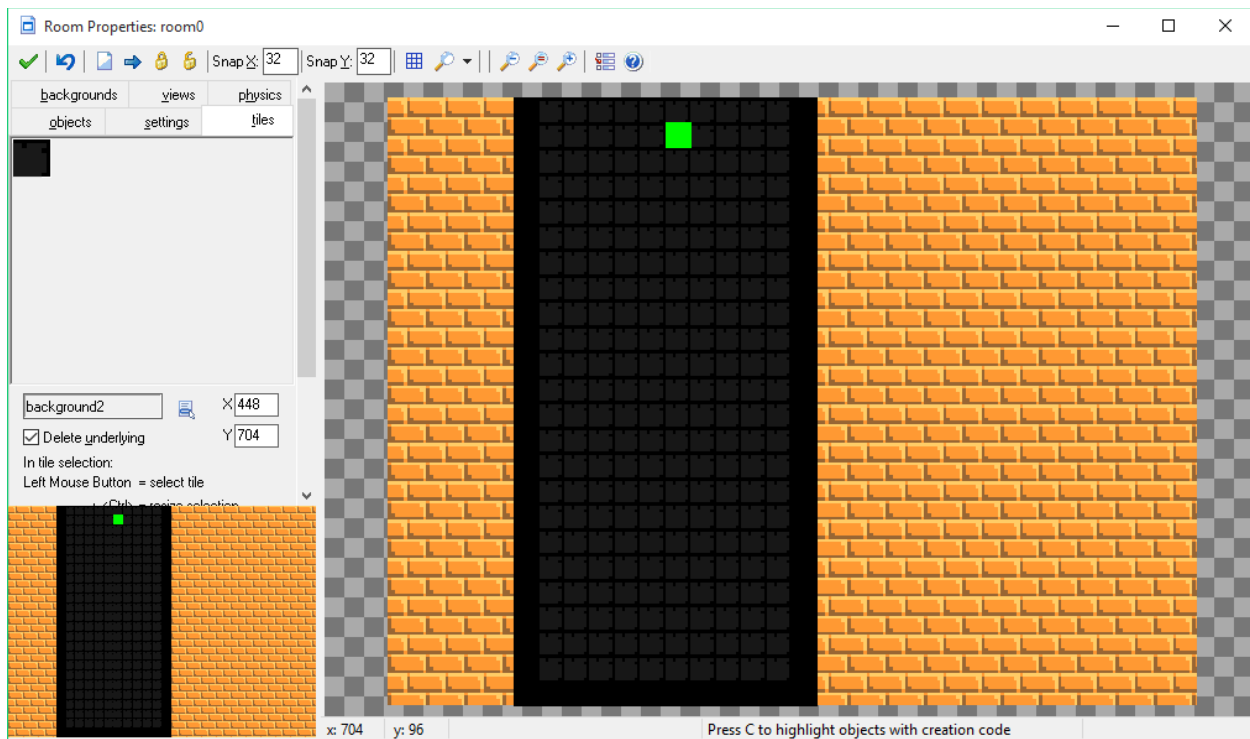
## Background

In room0, go to the background tab and change the background to “background3”





Then go to the “tiles” tab and select “background2”. Then fill in the playing area of the room.



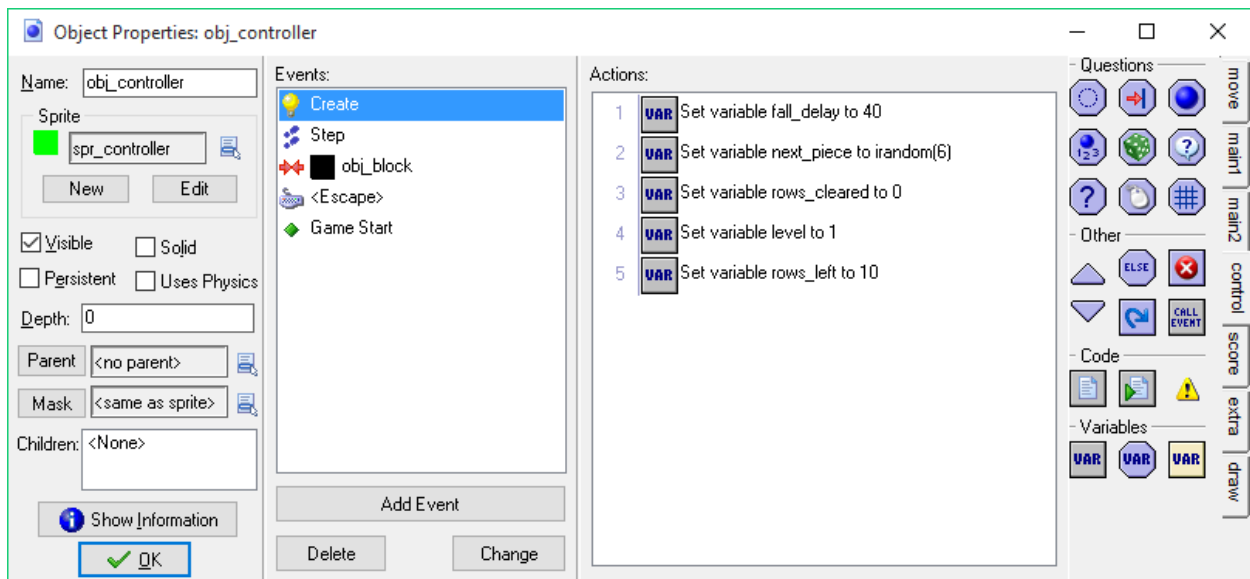
Scoring and Levels

The way levels work in Tetris is that every time ten rows are cleared, you progress one level and the pieces fall a little bit faster.

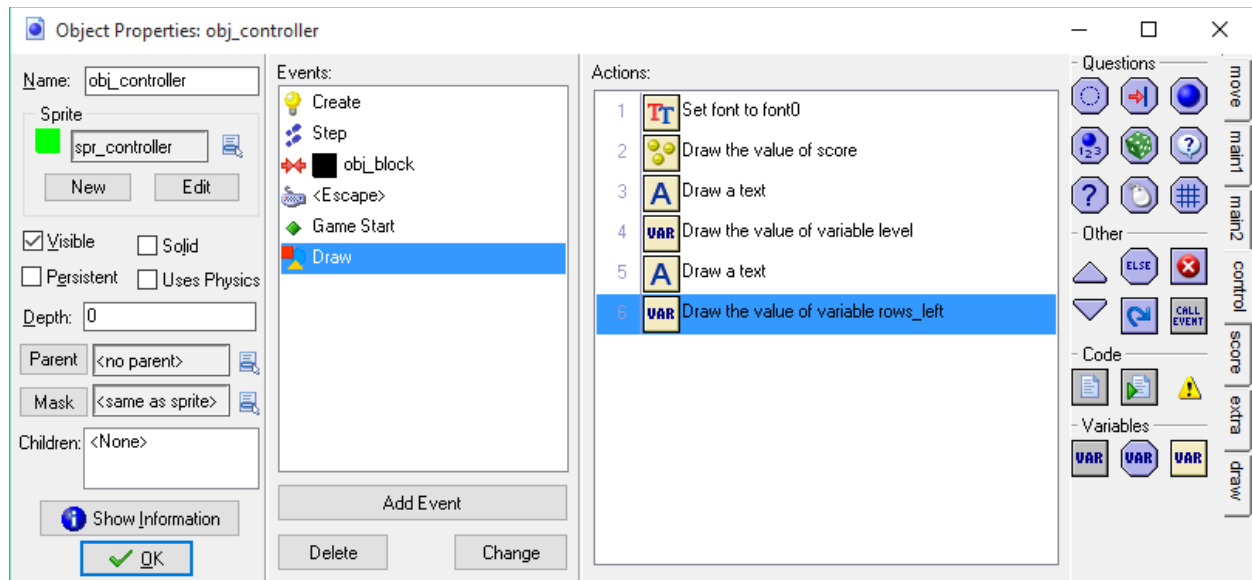
For scoring, you get 100 points for each row cleared, but if you clear more than one with just one block you get extra points.

To implement these features, initialize three variables in the create event of obj\_controller. “rows\_cleared” will refer to how many rows have been cleared by the current block (for scoring), “level” will keep track of the level, and “rows\_left” will keep track of how many rows still have to be cleared before you advance to the next level.


Initialize “rows\_cleared” to 0, “level” to 1, and “rows\_left” to 10.





Now, add a “Draw” event to obj\_controller so that the player will be able to see the current status of their playing during the game. Set the font to font0, draw the score, and draw headings for “Level” and “Rows Left” along with their values.



### Set Font




Font:  

align:  

☒ OK ☐ Cancel

### Draw Score



x:


y:

caption:

☐ Relative

☒ OK ☐ Cancel

### Draw Text



Applies to  
☒ Self  
☐ Other  
☐ Object:

text:


x:

y:

☐ Relative

☒ OK ☐ Cancel

### Draw Variable



Applies to  
☒ Self  
☐ Other  
☐ Object:

variable:

x:

y:

☐ Relative

☒ OK ☐ Cancel

### Draw Text

Applies to:

☒ Self  
☐ Other  
☐ Object:

text:   
x:   
y:

☐ Relative

☒ OK ☐ Cancel

### Draw Variable

Applies to:

☒ Self  
☐ Other  
☐ Object:

variable:   
x:   
y:

☐ Relative

☒ OK ☐ Cancel

Now, to get the leveling to work, modify the step event as shown:

Object Properties: obj\_controller

Name:

Sprite:

☒ Visible ☐ Solid  
☐ Persistent ☐ Uses Physics

Depth:

Parent:

Mask:

Children:

☒ OK

Events:

- Create
- Step**
- obj\_block
- <Escape>
- Game Start
- Draw

Actions:

- 1 **VAR** Set variable rows\_cleared to 0
- 2 **VAR** If rows\_left is less than or equal to 0
- 3 Start of a block
- 4 **VAR** Set variable rows\_left to 10
- 5 **VAR** Set variable level to 1
- 6 **VAR** Set variable fall\_delay to fall\_delay\*.9
- 7 **VAR** If fall\_delay is less than 1
- 8 Start of a block
- 9 **VAR** Set variable fall\_delay to 1
- 10 End of a block
- 11 End of a block
- 12 **If** the number of instances is a value
- 13 Start of a block
- 14 Create instance of object obj\_tetris\_shape
- 15 Change sprite into spr\_tetris\_shape
- 16 **VAR** Set variable next\_piece to irandom(6)
- 17 End of a block

Also, modify the scr\_clear\_row function to be the following:

```
var xx = x, yy = y, row_full = true;
// check to the left
while (place_meeting(xx-32,y,obj_wall)==false) {
    xx -= 32;
```

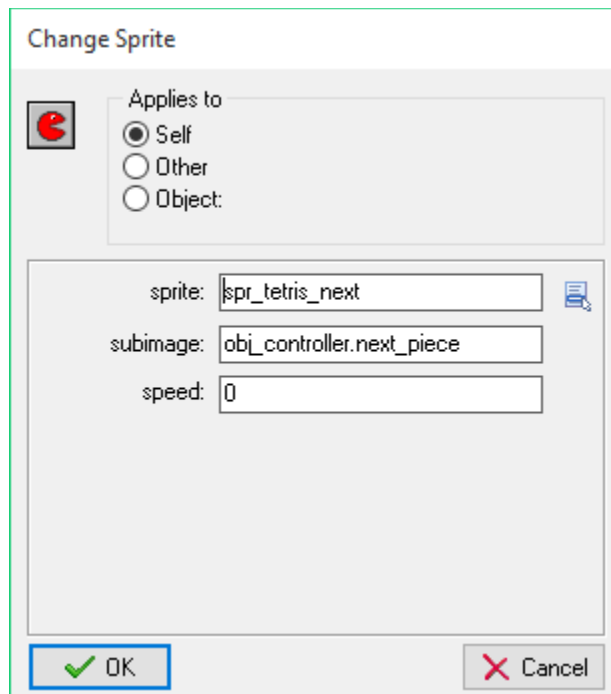
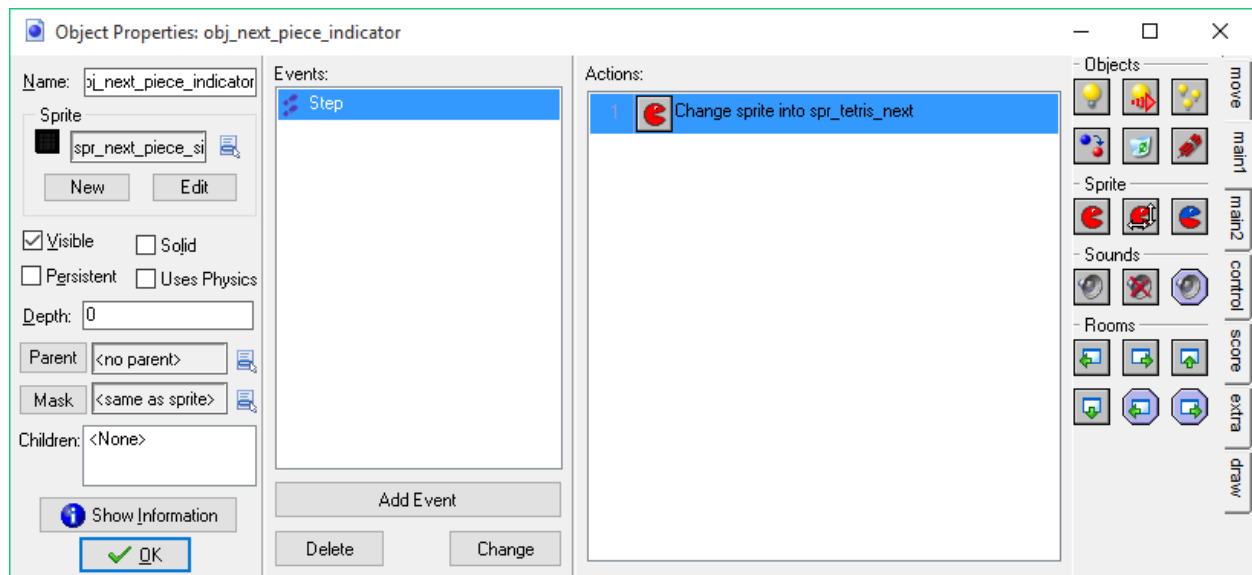
```

        if (place_meeting(xx,y,obj_tetris_block)==0) {
            row_full = false;
            break;
        }
    }
    // check to the right
    xx = x;
    while (place_meeting(xx+32,y,obj_wall)==false) {
        xx += 32;
        if (place_meeting(xx,y,obj_tetris_block)==0) {
            row_full = false;
            break;
        }
    }
    // if the row is full, clear it
    if (row_full == true) {
        obj_controller.rows_left--; // new rows
        obj_controller.rows_cleared++;
        score += 100*obj_controller.rows_cleared;
        with obj_tetris_block {
            if (y==yy) instance_destroy()
            else if (y<yy) y+=32;
        }
    }
}

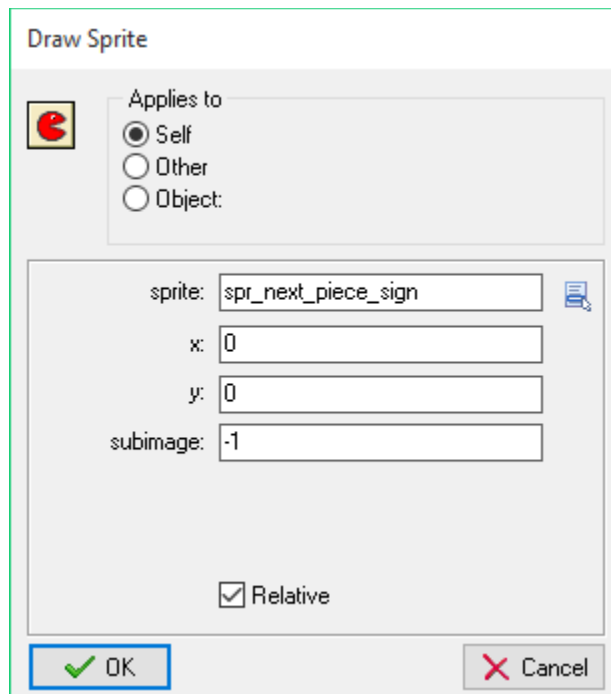
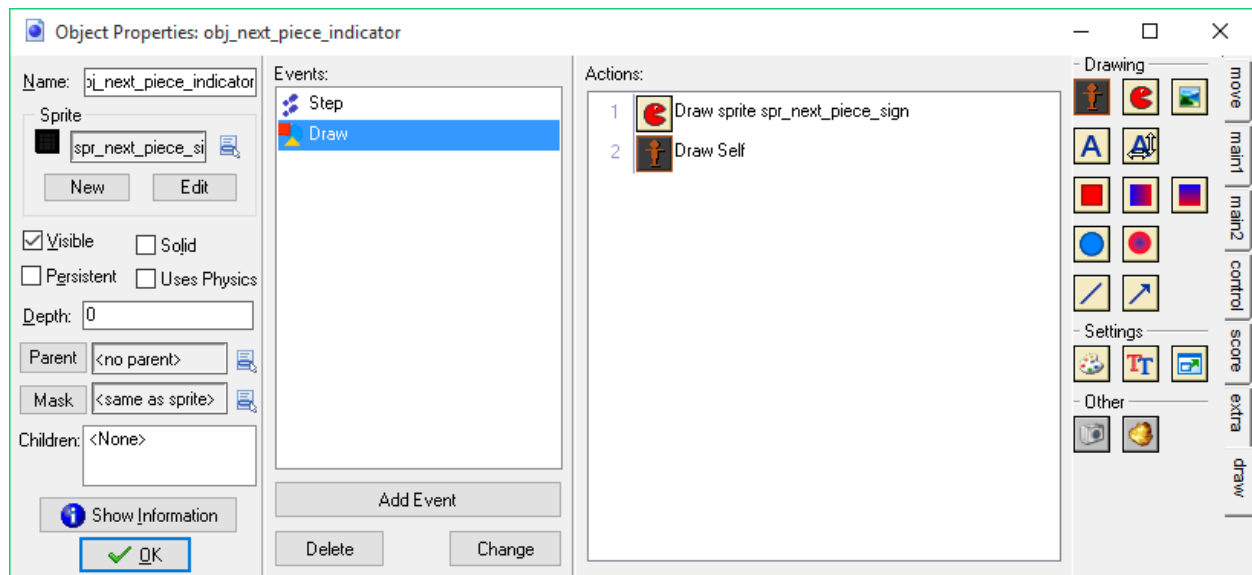
```

## Next Piece Indicator

To add a little icon that shows what the next piece will be is, create an object called “obj\_next\_piece\_indicator”. In the step event, set the sprite to be “spr\_tetris\_next”, but have the subimage be obj\_controller.next\_piece.

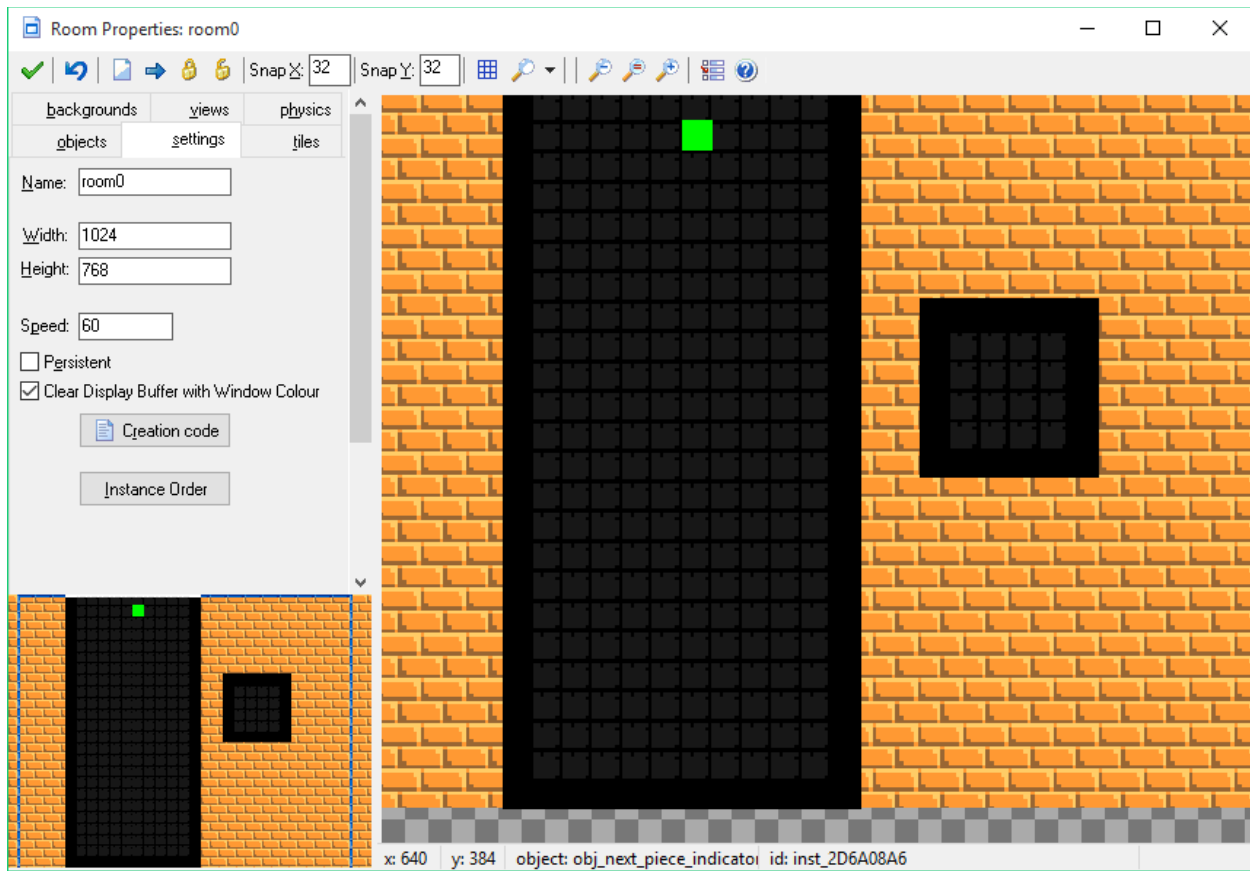


Then add a draw event which draws the background for the icon (spr\_next\_piece\_sign) and the next piece (i.e., obj\_next\_piece\_indicator's sprite).



Then place the object in room0.

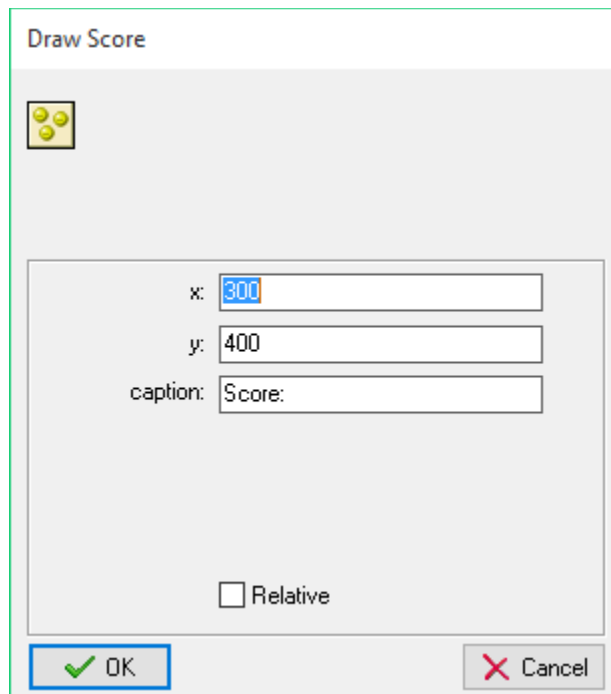
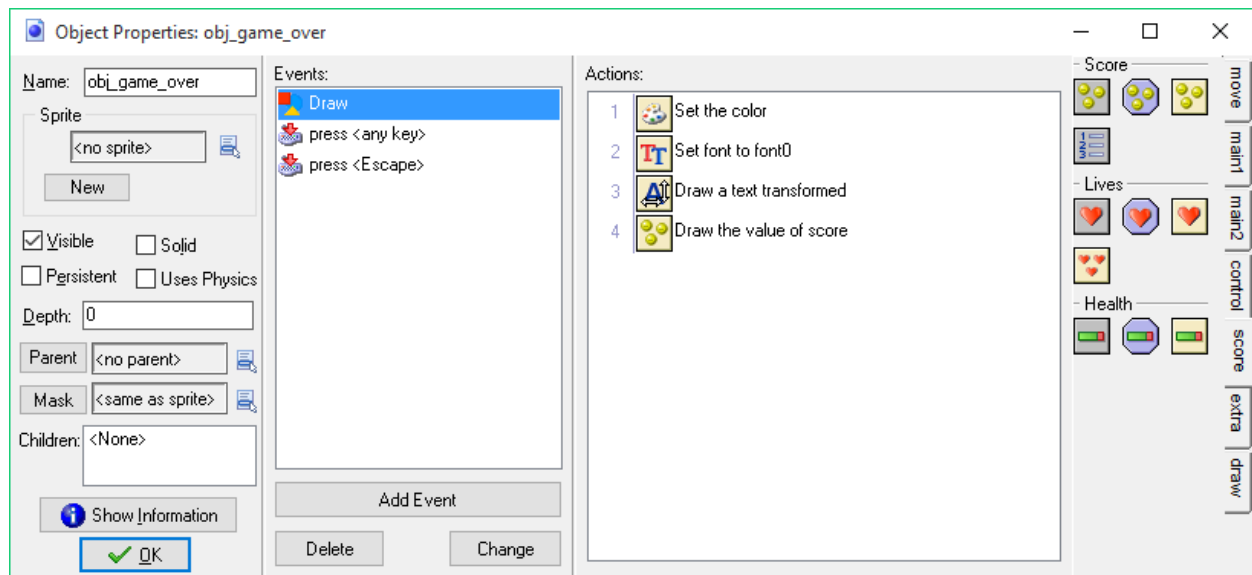




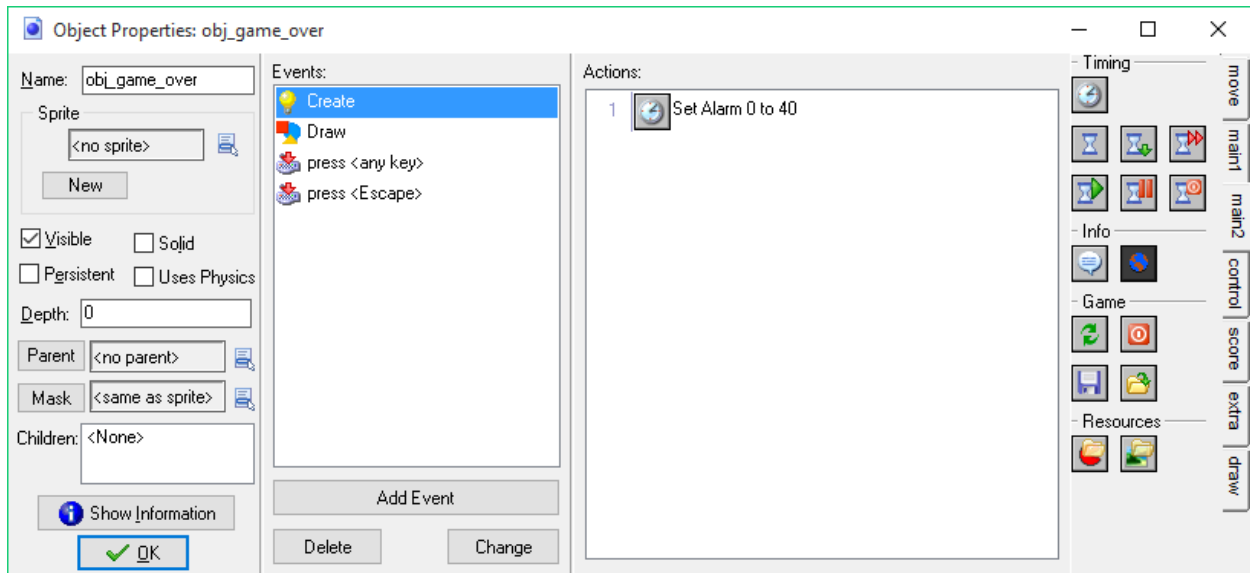
## Improve the Game Over Screen

Two things can be done to improve the game over screen. One is to have the player's final score shown on it, and another is to add a slight delay time during which the player can't restart the game (to give the player a chance to actually see the score).

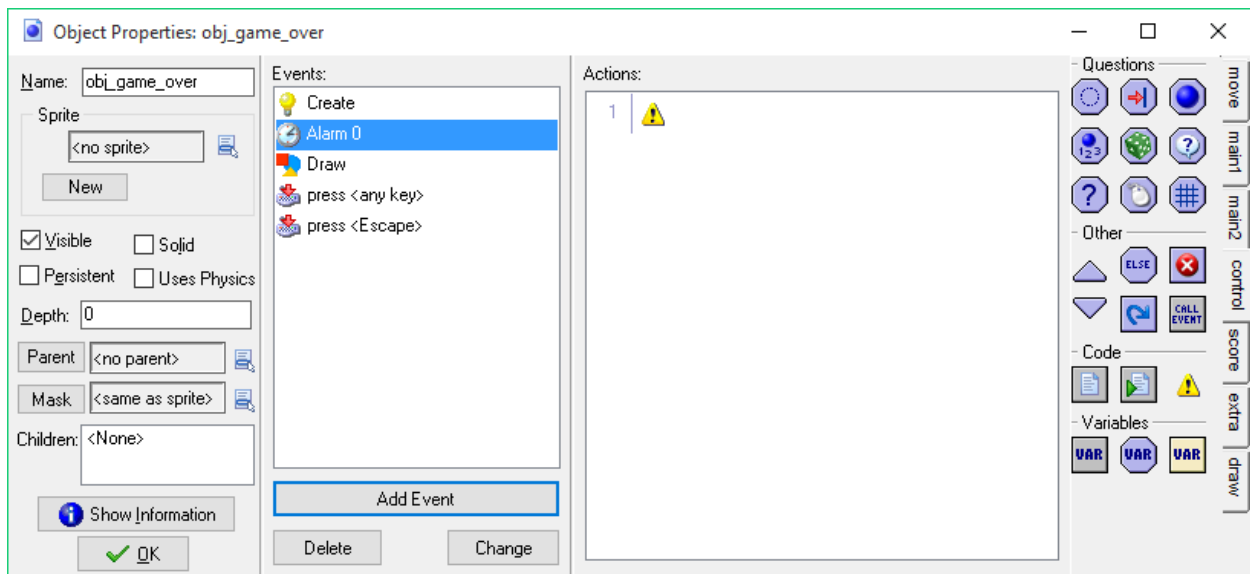
To add the final score is very easy ... just add a draw score action.



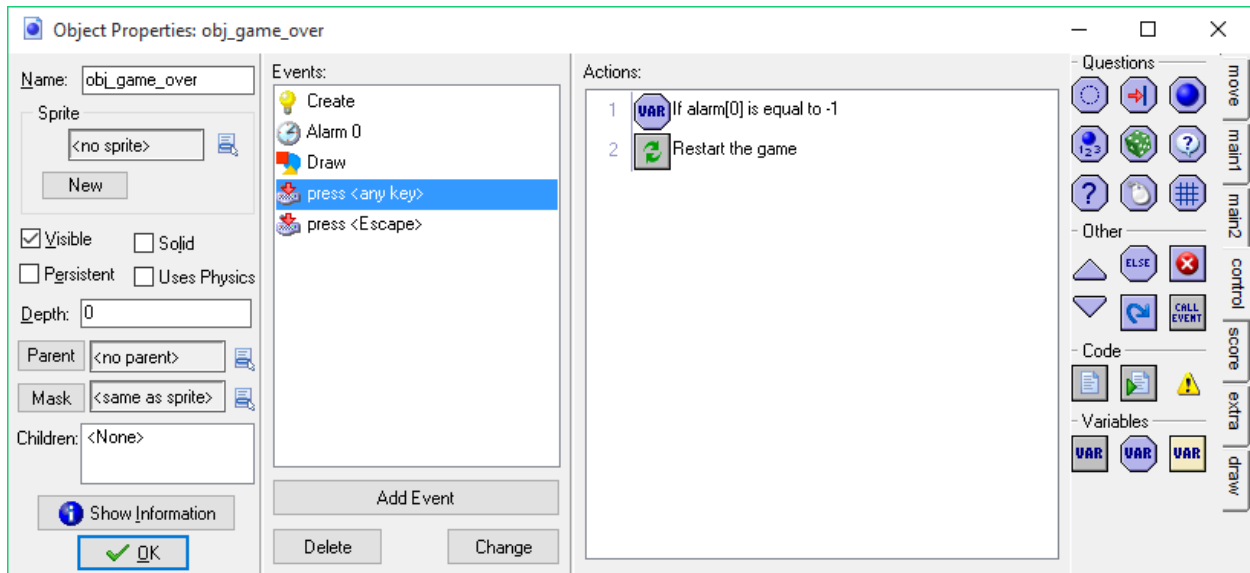
To add the delay is slightly more difficult. First, add a create event that sets alarm 0 to 40:



Then add an alarm 0. Note that it doesn't actually have to do anything because it's just a timer, but you can get it to remain in the object by adding a comment:

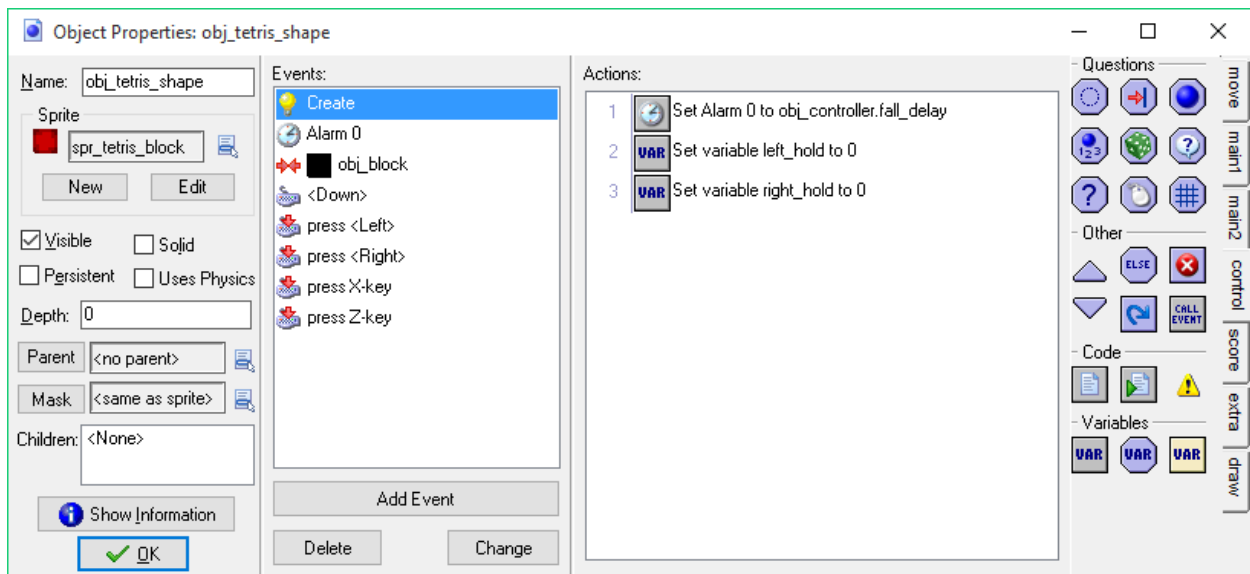


Then just add a conditional to the restart statement in the press <any key> event:

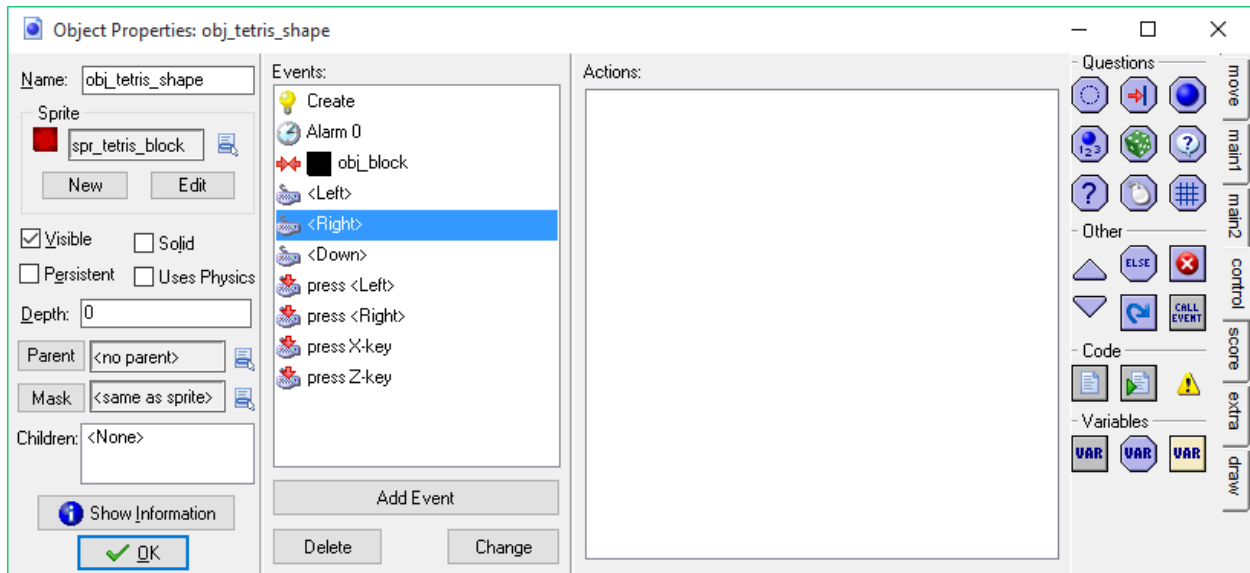


## Hold down Left and Right buttons

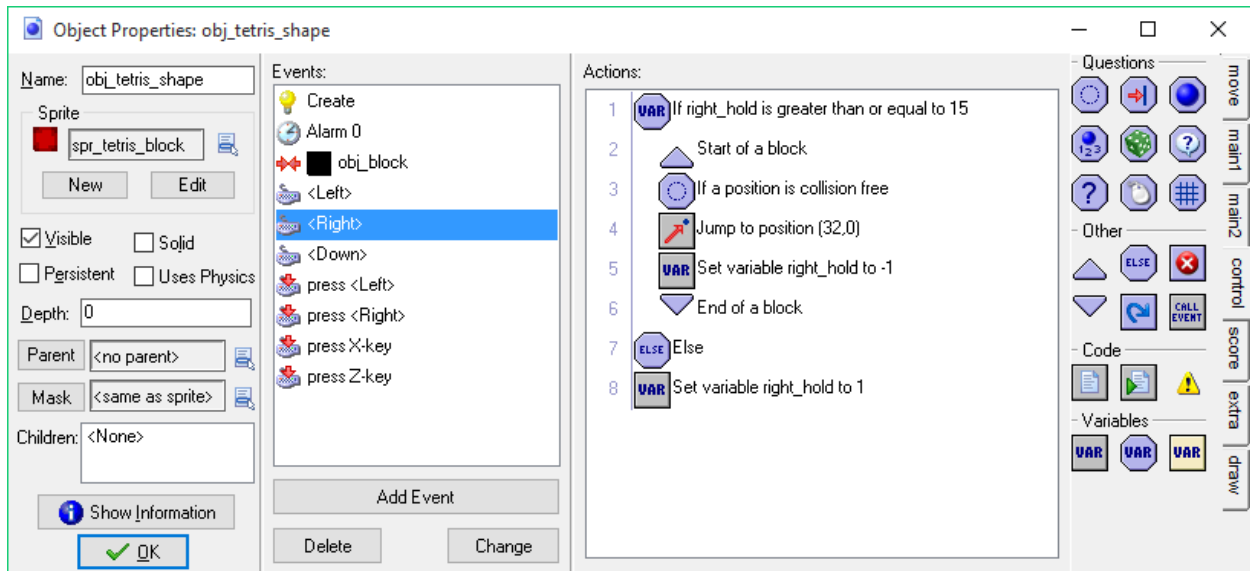
To make it so that you can hold down the left or right arrow keys for a second and then the piece will fly in the direction you were indicating, we need to initialize some variables. “left\_hold” and “right\_hold” are really just counters to see how long you have been holding the key down.



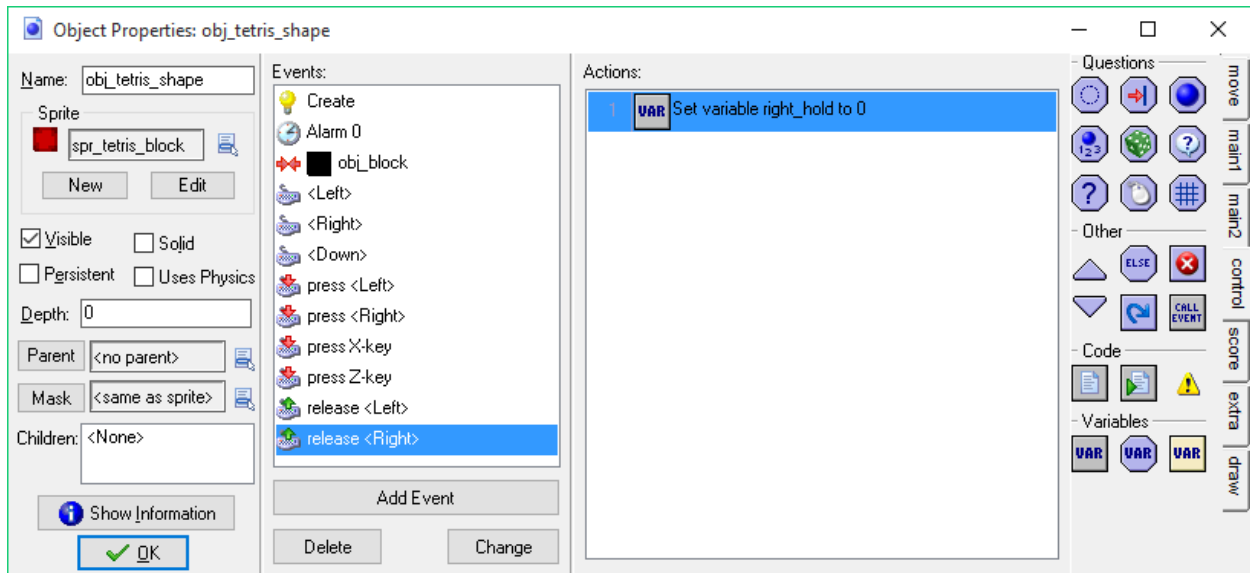
Then add regular keyboard left and keyboard right events:



In here, we have to check to see if the key has been held down long enough. If it has, then you act just as you would for a pressed up or down. Otherwise, you increment the counter.



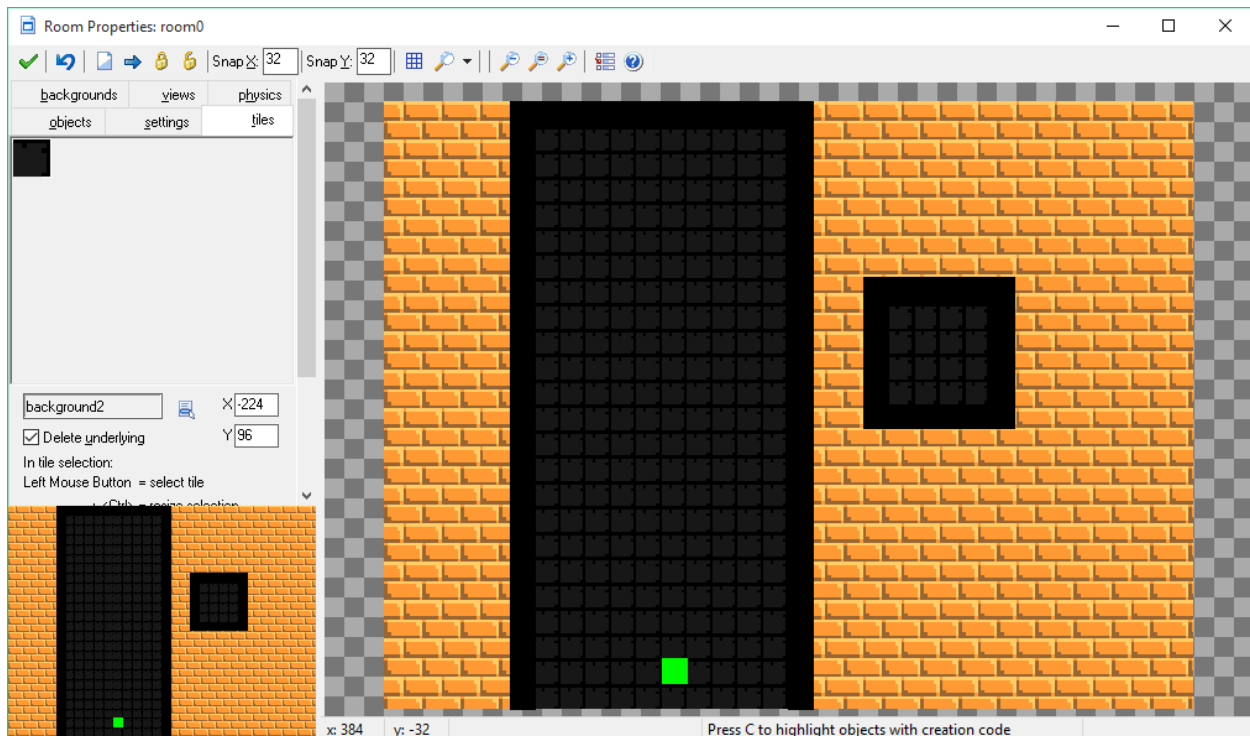
Lastly, you have to reset the right\_hold and left\_hold variables to 0 when you release the left and right keys:



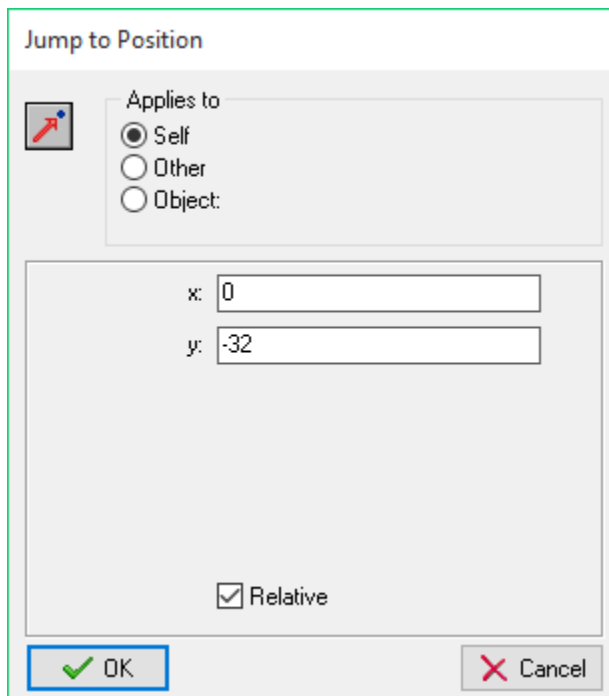
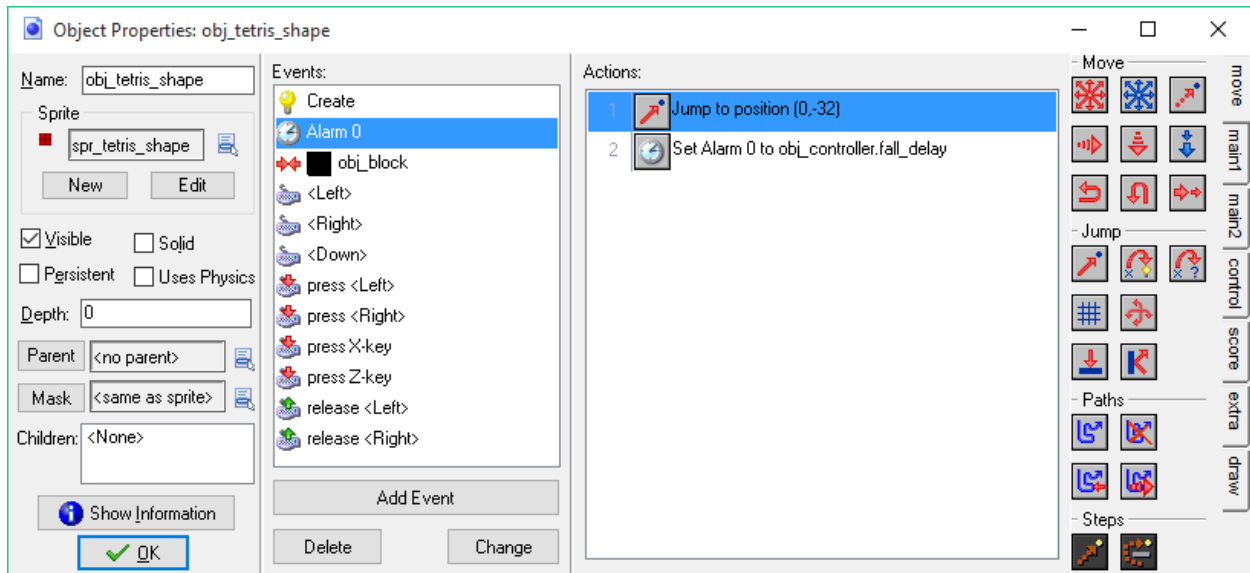
## 180 Degree Twist

To add an extra “twist” to the game, you can make the Tetris blocks fall *up* instead of down. This is actually surprisingly easy to do—only a couple of variables need to be changed.

First, we need to move the controller to the bottom of the room and the floor to the top:



Then, we need to change `obj_tetris_shape` so that the blocks will fall upwards. In the `alarm0` event, change the jump action to be -32 instead of 32:



Lastly, the script that clears the rows needs to be altered so that it moves the blocks that are *below* the cleared row *up*:

```
var xx = x, yy = y, row_full = true;
// check to the left
while (place_meeting(xx-32,y,obj_wall)==false) {
    xx -= 32;
    if (place_meeting(xx,y,obj_tetris_block)==0) {
        row_full = false;
        break;
    }
}
```

```
// check to the right
xx = x;
while (place_meeting(xx+32,y,obj_wall)==false) {
    xx += 32;
    if (place_meeting(xx,y,obj_tetris_block)==0) {
        row_full = false;
        break;
    }
}
// if the row is full, clear it
if (row_full == true) {
    obj_controller.rows_left--;
    obj_controller.rows_cleared++;
    score += 100*obj_controller.rows_cleared;
    with obj_tetris_block {
        if (y==yy) instance_destroy()
        else if (y>yy) y-=32;
    }
}
```