

## Lazarus: Stages 3 & 4

In the world that we live in, we are a subject to the laws of **physics**. The law of **gravity** brings objects down to earth. Actions have equal and opposite reactions. Some objects have more **mass** than other objects, as measurable by weight. We can continue to list some basic laws and principles that you can connect to in everyday life. In developing video and computer games, sometimes we have to consider how things behave in the real world based on **physics**. The study of **physics** is a science that deals with matter and energy and the way they act on each other in heat, light, electricity, and sound, and motion. If I am hit by an acorn that falls out of a tree from ten feet high, it will not hurt quite as bad as a bowling ball that falls from a shelf that is ten feet high. This hypothesis is validated by laws of physics. The two objects will fall at the same speed, but the **mass** of the bowling ball gives it more momentum, hence it hits me with more force.


The example above is what you would expect in the real world. As studied in STAGES 1 & 2, animated action in a game can occur in a way that looks familiar to us, or it can be *exaggerated* over what may typically happen in reality. In either case, animation in computer and video games can be created in way that feels comfortable or sensible to us, even though the characters and the action are fantasy. In programming your properties, you can consider the way things behave in nature based on **physics**. Even though movements and action can be *exaggerated*, the behaviors become more *plausible* when they occur in a way that is recognizable. If you apply **physics** in game building, the physical behaviors of events and actions that you see every day can be used as a reference to plan what will happen in the gameplay.

Game Maker has numerous properties that can be used to simulate **physics**. In STAGES 3 & 4 you will create a logic system that uses collision events to establish **physics** properties for objects. You will show, through animation, the varying **mass** of objects and what happens when a heavy object collides with something light. As boxes fall, they will either stack up on equal weight or heavier boxes, or crush lighter boxes. This constant dropping of boxes will be controlled by a special object called a **controller object**. This object is invisible, but has properties that control what happens to other objects. In this game, it controls the constant dropping of boxes. Fixed move actions will simulate **gravity**. Other physics principles for game design, like *friction*, will be explored in future tutorials.

It is impossible that falling boxes in a warehouse will always find someone to fall on. Yet they always seem to find Lazarus. You will include a **variable** within a "Jump to" action that will always put a box over Lazarus. A **variable** is an unknown quantity. It can assume any one of a set values. The set of values might be a *domain* for x values, or a *range* for y. In the box properties, the **variable** will always be equal to the location of Lazarus as he moves and jumps left or right along the *domain* (x axis) of the room.

Through the use of animation, you will show how Lazarus behaves when he knows that something heavy is coming down on him. Even though this would never happen in a warehouse, it will make sense to the player because of the magic of animation properties, a connection to **physics**, and the use of a **variable** within a property.

Be sure to complete the vocabulary definitions on your tutorial guide before proceeding.

- **BE SURE TO COMPLETE HYPOTHESIS STATEMENTS FOR EACH STAGE.**
-  **DO NOT USE THE TEST BUTTON IN THIS ACTIVITY UNTIL THE END!**
- **OPEN YOUR FILE YOU SAVED AS *initials\_lazarus***
- **THERE ARE NO SAVE PROMPTS IN THIS TUTORIAL, SO SAVE REGULARLY.**

**RESUME GAME BUILD ON NEXT PAGE.**

### STAGE 3:

#### Creating new box sprite and object resources for the game

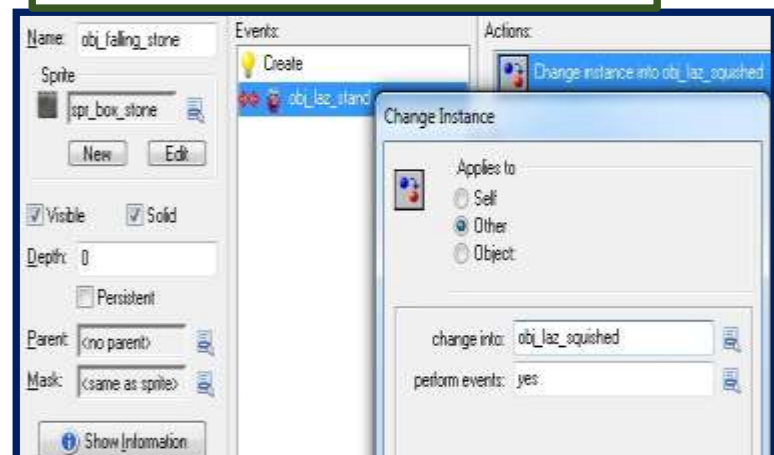
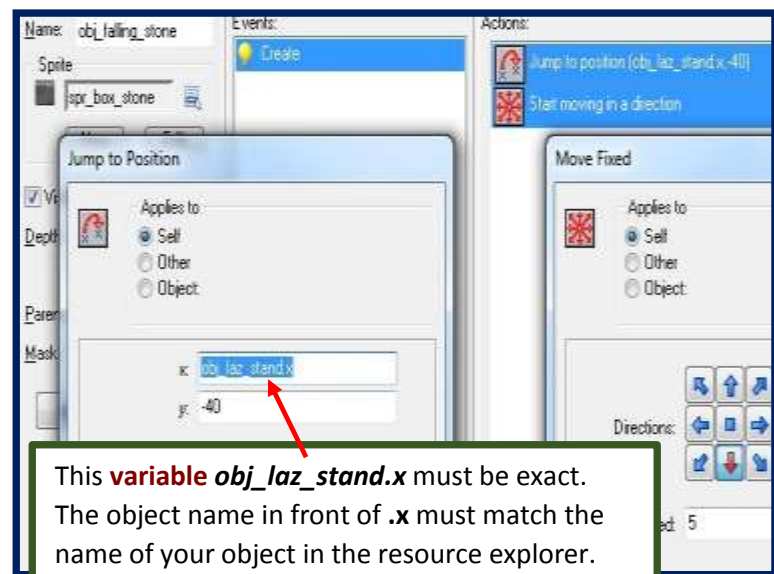
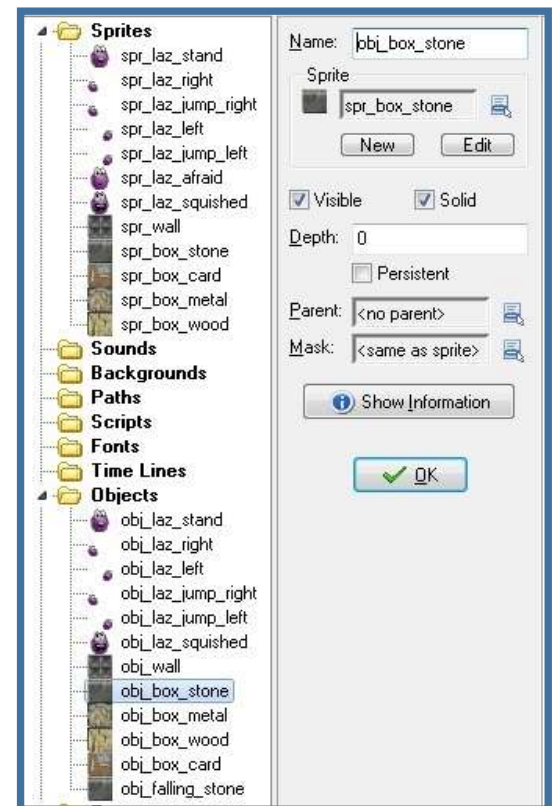
- 1- Create a sprite called **spr\_box\_stone** from the **StoneBox.gif** asset.
- 2- Create **spr\_box\_card** using **CardBox.gif**.
- 3- Now create sprites called **spr\_box\_metal** using **MetalBox.gif**.
- 4- Create **spr\_box\_wood** using **WoodBox.gif**.
- 5- Create a new object called **obj\_box\_stone** and give it the sprite for the stone box. Check the **Solid** box.
- 6- Repeat the previous step to add objects for **obj\_box\_metal**, **obj\_box\_wood** and **obj\_box\_card**.

#### Creating the falling stone objects for the game:

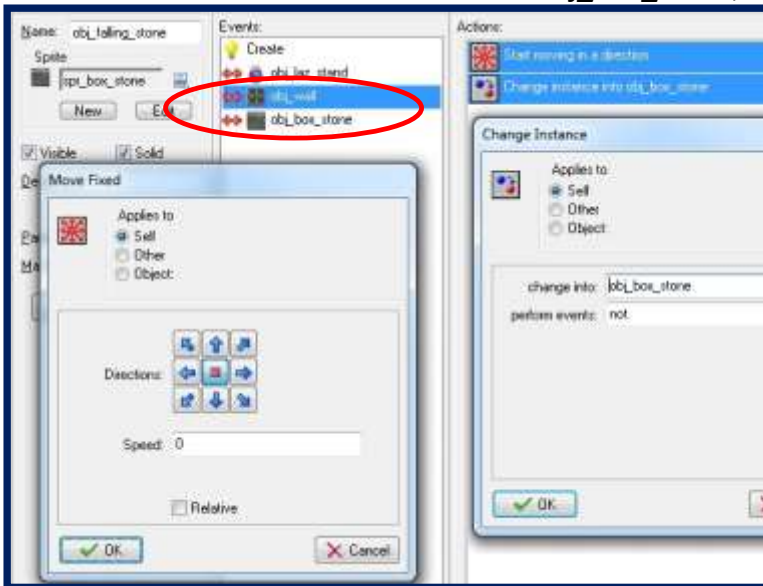
- 1- Create a new object called **obj\_falling\_stone**, using the **spr\_box\_stone**, making sure to check the **Solid** box in the form.
- 2- Add a **Create** event and include a **Jump to Position** action in the event. Type the variable **obj\_laz\_stand.x** (the horizontal position of Lazarus) into **X** and set **Y** to **-40**.
- 3- Next include the **Move Fixed** action, select the down arrow and set **Speed** to **5**.

**Variables** As you create actions, you may have to assign a value to a property without knowing what that numerical value is going to be at any time during gameplay. As the action of a game is always changing, the values of variables may also be changing. For example, as an object (like Lazarus) is moving, the *x* and *y* coordinates for its location are always changing. Of course, the numeric values for *x* and *y* will change, or *vary*. **Variables** are letters or other symbols that represent those unknown numbers or values. The **variable** for *x* in the "Jump to" action (shown on right) will always be equal to the *x* location of Lazarus in the room. As he jumps, the *y* value for the location of the stone will always be equal to -40 (hiding just above the top of the room). The *x* location of the stone will always be equal to the *x* location of Lazarus. So as Lazarus moves from left to right, the stone will always be created at an *x* value equal to the *x* location of the Lazarus *origin*. This will always put the stone directly above Lazarus! Seems kind of dangerous! The room measures to a width of 640 pixels, so the value of the **variable** **obj\_laz\_stand.x** will include any value in the *domain* from 0 to 640.

- 4- Add a **Collision** event with **obj\_laz\_stand** and include a **Change Instance** action in it. Select **Other** for the **Applies to** option, so that it changes the Lazarus object rather than the box. Select the **obj\_laz\_squished** and select **yes** to **Perform Events**.

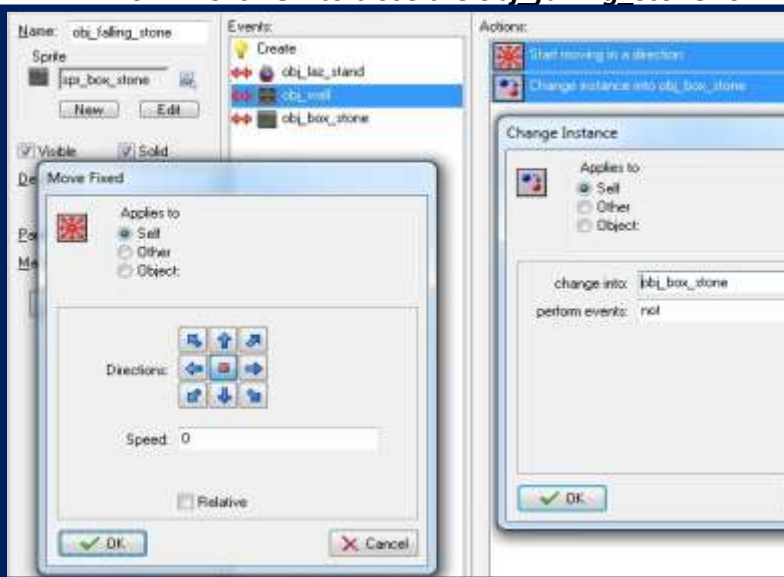


- 5- Add another **Collision** event, this time with **obj\_wall**. Include a **Move Fixed** action and **select the middle square** with a **Speed of 0**. Add a **Change Instance** action, and select the stationary box **obj\_box\_stone**.
- 6- Add a third **Collision** event with **obj\_box\_stone**, adding the same two actions as with the wall above.



**Physics in Game Design** Everything that you experience in your life is connected to **physics**. As a passenger in a car, you feel your body move forward as it comes to a stop. The **mass** in your book bag increases (gets heavier) when you have a lot of homework. If a pencil rolls off of your desk, it falls to the floor. You are comfortable with these actions because they are natural and normal. When you understand why they occur, you are understanding **physics**. In game design, you can simulate physics if you consider how things behave in nature, based on mass and the presence of **gravity**. Animation will seem natural if the action has the appearance of physics, based on what we see every day. Even if the action is a fantasy, unusual, or impossible, you can use the appearance physics to make it seem *plausible*.

- 7- Add a fourth Collision event with **obj\_box\_metal**. Include a **Destroy Instance** action and select the **Other** object.
- 8- Add the fifth **Collision** event with **obj\_box\_wood** with an identical **Destroy Instance** actions as we did in step 7 and selecting **Other**.
- 9- Finally, add a sixth **Collision** with **obj\_box\_card**, again selecting **Other** in the **Destroy Instance** property form. Click **OK** to close the **obj\_falling\_stone** form.



**The Physics in Lazarus** In the Lazarus Stages 1 & 2, animation techniques like staging, squash and stretch, exaggeration, and anticipation were used to give Lazarus life-like qualities. The behaviors that you see, although unreal, are based on real physics. The boxes that are created above Lazarus will be made of different materials, some with a **mass** (weight) that is greater than others. We haven't said much about what happens to these boxes, but with the presence of **gravity**, you can probably guess what they do! Naturally, they all fall at the same speed. But due to physics, what do you expect to see if a lighter box falls on a heavier box? What if a heavy box drops on a lighter box? And what if one falls on Lazarus? Think about this as you set more properties.

**More About Variables** In Game Maker, there are "built-in" **variables** that can be set for objects, or individual instances of the objects in the room. Built-in variables can be set for a variety of object properties besides x and y coordinates. In fact, you have already used them in other tutorials. If you built *Evil Clutches* or *Galactic Mail*, you have used variables to set coordinate, direction and speed properties. Do you recall using a variable for jump to in *Galactic Mail* with a value *object\_specialmoon.x* and *object\_specialmoon.y*? Formulas can also be used to define variables. Go back to *Galactic Mail* and look for a variable for direction that was set to *random(360)*. Another variable was then set to add control to the object, dividing the random value by five (*direction/5*). What did these variables do? Try going back through past tutorials to look for the use of variables in your programming. You can also make your own variables in Game Maker. In more advanced game design, variables are important for writing scripts. *Global* and *local* variables will be studied in future tutorials, and you can read more about variables in *Concepts Explained*.



## Other falling boxes for collisions with different kinds of boxes (no illustrations)

- 1- Create **obj\_falling\_metal**, **obj\_falling\_wood**, and **obj\_falling\_card** using the **spr\_box\_metal**, **spr\_box\_wood**, and **spr\_box\_card** sprites respectively.
- 2- Do the following for each of these objects (this a repeat of steps 2 and 3 in the previous procedure). Add a **Create** event and include a **Jump to Position** action in it. Type the variable **obj\_laz\_stand.x** (the horizontal position of Lazarus) into **X** and set **Y** to **-40**. Also add a **Move Fixed** action, select the down arrow and set **Speed** to 5. Make sure that these two actions are exact and included in the Create event for each of these objects.
- 3- Again, do the following for each of these objects (repeating step 4 in previous procedure). Add a **Collision** event with **obj\_laz\_stand** and include a **Change Instance** action in it. Change the **Applies to** option to **Other**, so that it changes Lazarus rather than the box. Select the **obj\_laz\_squished** and select **yes to Perform Events**.
- 4- **Follow the same basic steps in the last procedure, following steps 5 through 9** for each of these objects. Since the collision properties vary in the falling objects, you will have to use the property settings shown in the chart below for the falling metal, wood, and card objects. The falling stone settings are also included in the chart, although they should have been done in falling stone steps 1 through 9 above. Check the settings for each falling box object carefully.

COLLISION EVENT PROPERTIES FOR FALLING OBJECTS			
OBJECTS	ACTIONS AND ACTION PROPERTIES		
<b>obj_falling_stone</b> colloides with	MOVE FIXED PROPERTIES	CHANGE INSTANCE PROPERTIES	DESTROY INSTANCE PROPERTIES
obj_wall	self, no direction, speed 0	Self, obj_box_stone, not	
obj_box_stone	self, no direction, speed 0	Self, obj_box_stone, not	
obj_box_metal			Applies to: Other
obj_box_wood			Applies to: Other
obj_box_card			Applies to: Other
<b>obj_falling_metal</b> colloides with	MOVE FIXED PROPERTIES	CHANGE INSTANCE PROPERTIES	DESTROY INSTANCE PROPERTIES
obj_wall	self, no direction, speed 0	Self, obj_box_metal, not	
obj_box_stone	self, no direction, speed 0	Self, obj_box_metal, not	
obj_box_metal	self, no direction, speed 0	Self, obj_box_metal, not	
obj_box_wood			Applies to: Other
obj_box_card			Applies to: Other
<b>obj_falling_wood</b> colloides with	MOVE FIXED PROPERTIES	CHANGE INSTANCE PROPERTIES	DESTROY INSTANCE PROPERTIES
obj_wall	self, no direction, speed 0	Self, obj_box_wood, not	
obj_box_stone	self, no direction, speed 0	Self, obj_box_wood, not	
obj_box_metal	self, no direction, speed 0	Self, obj_box_wood, not	
obj_box_wood	self, no direction, speed 0	Self, obj_box_wood, not	
obj_box_card			Applies to: Other
<b>obj_falling_card</b> colloides with	MOVE FIXED PROPERTIES	CHANGE INSTANCE PROPERTIES	DESTROY INSTANCE PROPERTIES
obj_wall	self, no direction, speed 0	Self, obj_box_card, not	
obj_box_stone	self, no direction, speed 0	Self, obj_box_card, not	
obj_box_metal	self, no direction, speed 0	Self, obj_box_card, not	
obj_box_wood	self, no direction, speed 0	Self, obj_box_card, not	
obj_box_card	self, no direction, speed 0	Self, obj_box_card, not	

This is a tedious process, so take your time as you set the collision properties for each box.

## Creating next box object resources for the game (no illustrations)

- 1- Create a new object called **obj\_next\_stone**, give it a stone box sprite, and enable the **Solid** option. That's it, so click **OK** to close the object properties form.
- 2- Create objects for **obj\_next\_metal**, **obj\_next\_wood**, and **obj\_next\_card** using the appropriate sprites. As you did with the next stone, enable the **Solid** option. Click **OK** to close the forms.

### DO THIS ON YOUR TUTORIAL GUIDE

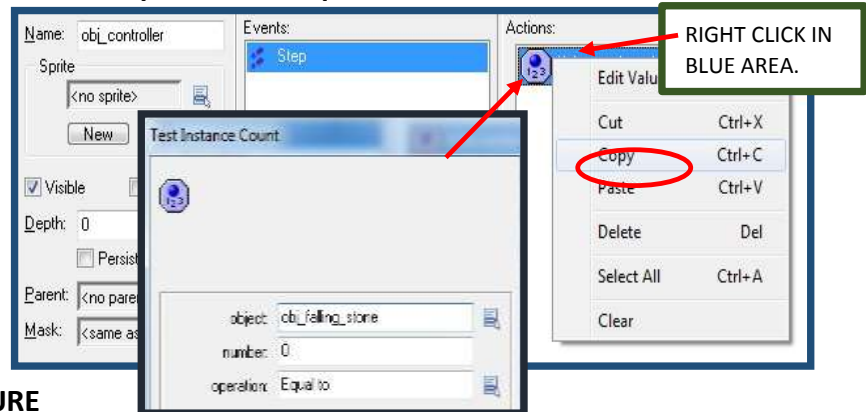
**STAGE 3 HYPOTHESIS STATEMENTS:** Write a hypothesis for the behaviors you expect to see with the Lazarus and box objects. Use appropriate new vocabulary in your hypothesis and cite events and actions. Think about where the instances of the boxes will appear and what they will do. Explain why. How is gravity and the difference in mass of each box going to be simulated? Explain why.

## STAGE 4:

### Building a controller object resource

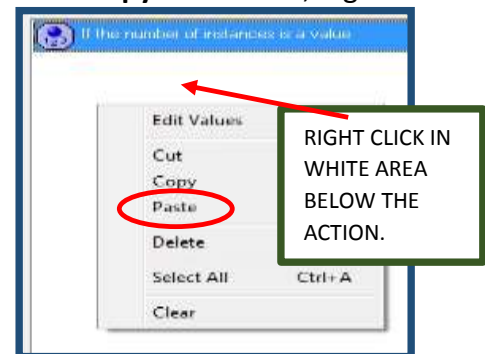
- 1- Create a new object called **obj\_controller**. There will be no sprite for this object.
- 2- Add a **Step** event, select **Step** from the menu event.
- 3- Drag and drop a **Test Instance Count** conditional action found in the **control** tab. Choose the **obj\_falling\_stone** object. Leave **Number** at **0** and **Operation** as **Equal to**. Click **OK** to close the form.

**Copying and Pasting** In the previous gold concept box, you read about *parent/child* as a setting that makes programming faster (to be learned later). But there is an easy way that for you to speed up the programming. Using a typical *copy and paste* procedure, you can copy actions, with all of their properties, and paste them for duplication. You can even open different objects and paste them there too!



### NOW LET'S TRY THE COPY AND PASTE PROCEDURE

- 4- Right click on the **Test Instance Count** action that you just created and select **Copy**. After that, Right click again in the white area below the action and select **Paste**.
- 5- Repeat this **Copy/Paste** procedure two more times so that you have a total of four **Test Instance Count** actions.
- 6- You will leave the first **Test Instance Count** action alone (this is the one that will count the instances of the falling stone object). You will however, need to open the second **Test Instance Count** action form by clicking on it, then changing it the falling stone object to **obj\_falling\_metal**. Click **OK** to close the form when you are done.
- 7- On the third action in the list, change the object to **obj\_falling\_wood**, and then **obj\_falling\_card** on the fourth action just as you did step 6.

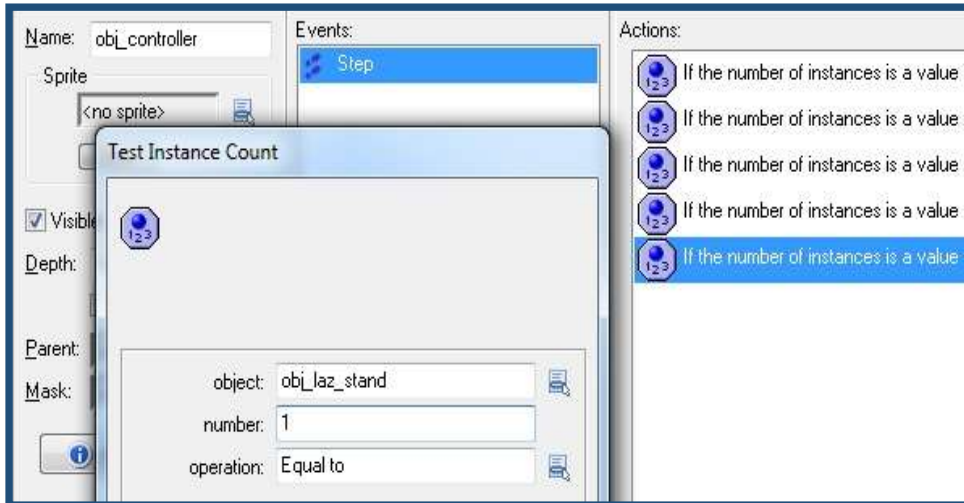


**Controller Objects** Controller objects are invisible objects with no sprite assignment. They can control various parts of gameplay, but they are not part of the game play. In Lazarus, a controller object sets the timing and appearance of box instances. They can also control parts of the game that not included in the game play. Examples might be the presence of background music, the display of point score, and other messages or information.





- 8- Add a final **Test Instance Count** action for the *obj\_laz\_stand* object, this time setting the **Number** to 1 and leaving the **Operation** as **Equal to**.

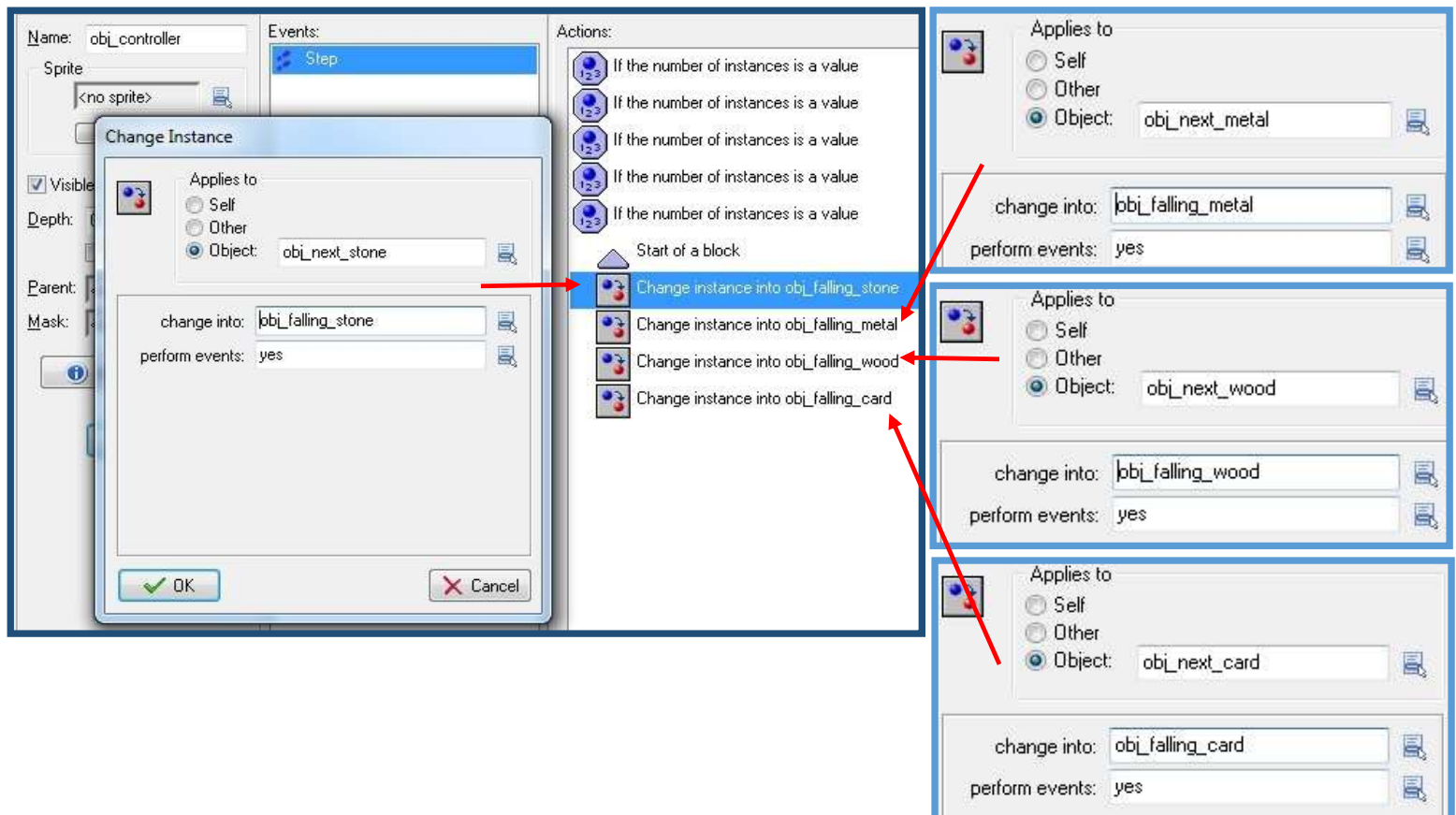


### The Box Controller

If it seems like cruel fate that hazards are always finding Lazarus. Blame it on the controller object. It is going to keep a steady supply of boxes dropping into the warehouse. At least Lazarus should have a fighting chance. These Test Instance Count actions play a role in that. Anything that happens after the conditional actions in this Step event, will only happen when there are no falling boxes (although that doesn't last long). Continue the game build from here by adding a *block*. Think about what this controller object is doing to control falling boxes.

### Adding a block sub-procedure to your controller object

- 1- If you closed the *obj\_controller* properties form, go ahead and reopen it.
- 2- Drag and drop a **Start Block** action.
- 3- Add a **Change Instance** action. Under **Applies to** select the bullet for **Object**. In the drop down menu to the right of **Object**, select *obj\_next\_stone*. In the drop down menu to the right of **Change Into**, select *obj\_falling\_stone*, and the below that select **yes** to perform events.
- 4- Add three more **Change Instance** actions to change *obj\_next\_metal* objects into *obj\_falling\_metal* objects, *obj\_next\_wood* into *obj\_falling\_wood*, and *obj\_next\_card* into *obj\_falling\_card*.



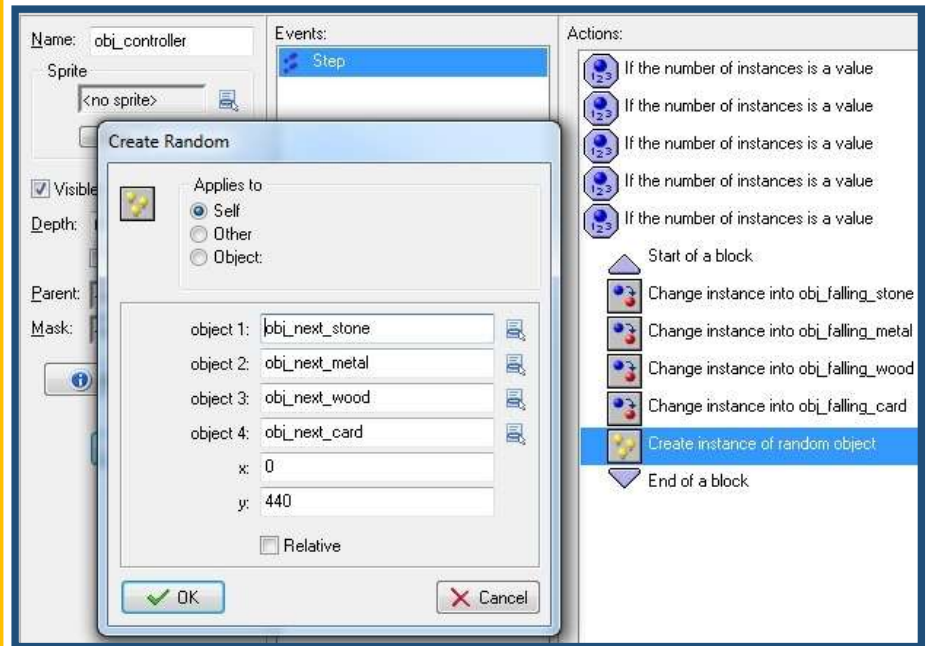
- 5- Add a **Create Random** action (**main1** tab) and select the four different next box objects as shown in graphic below. Set **X** to **0** and **Y** to **440**, and leave **Relative** disabled.

**NOTE:** These coordinates will put an instance of the next box where in the lower-left corner of the screen. The player will see this image and will be able to anticipate the next image to drop from the top of the room.

- 6- Finally, include an **End Block** action to conclude the block of actions that are dependent on all the conditions above them being true.

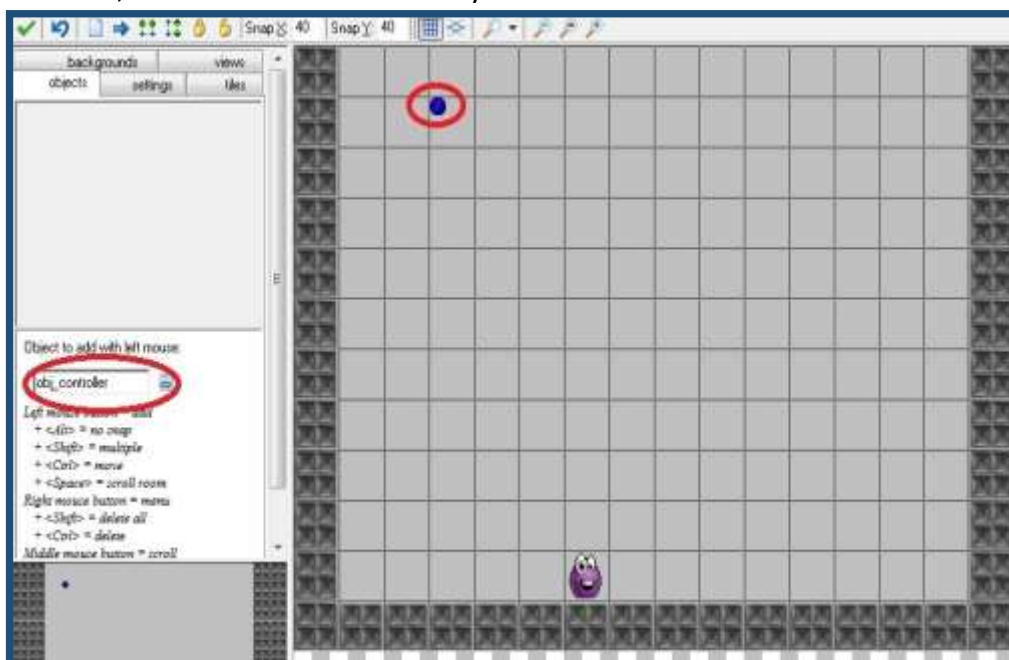
### Controller Objects and Playability

Although controller objects themselves are not part of the gameplay, they can affect the *playability* of a game. In *Galactic Mail*, you learned the importance of sharing game information by listing things like event keys, and what to do with them during gameplay. It may also be helpful to provide information about the gameplay while the game is in progress. Information, such as changes that are about to occur, might help the player make important decisions during gameplay. This controller object will provide something important that the player will want to know. What is that information? What will it look like? How and where in the room will it be delivered? These questions should be considered in your next hypothesis.



### Editing the Test Room and adding the Controller Object

- 1- Reopen the test room.
- 2- Edit the room by right clicking and deleting boxes in the middle of the room so that it leaves walls on both sides and across the bottom.
- 3- Add one instance of the controller object into the room. It will appear as a circle with a red question mark, but will be invisible when you run the test.



## DO THIS ON YOUR TUTORIAL GUIDE

**STAGE 4 HYPOTHESIS STATEMENTS:** Now it's time to predict the behaviors box objects based on the programming within the controller object.

- How many boxes will fall at any given time?
- Is there a specific order in which the boxes appear?
- What will the controller object do to inform the player of what happens next in the gameplay?
- Where in the room will that information be shown, and what will the player see?

Now it's time to test your game, so go ahead and click on the green triangle ( ▶ ) on the menu bar to run the game normally.

## DOES THE ACTION THAT YOU SEE AND THE CONTROL OF THE OBJECTS MEET YOUR HYPOTHESIS STATEMENTS?

Lazarus should be able to move and jump under the control of the left and right arrow keys as he did in the last test. Boxes should be dropping from the top of the room. Seemingly, the boxes always drop from above Lazarus, which is *plausible* if you believe in **gravity**. If Lazarus moves left or right, a box will fall to his location, forcing him to move again. You may recall creating an event with a **variable** that sets the location of these boxes. The different types of boxes do not come in any particular pattern, but instead are delivered randomly. But the player controlling Lazarus, will at least know what type of box is coming next because the **controller object** is kind enough to display an instance of that box in the lower left hand corner of the room (0,440). Don't be too appreciative, because the controller object is what keeps the random boxes coming. As the boxes fall, they either stack on top of each other, or some boxes crush other boxes due to their greater **mass** (heavier boxes crush lighter boxes). As the boxes stack, Lazarus should be able to jump on them as long as they are not stacked more than one box high. He can also climb them like stairs. The behavior of the falling, stacking, and crushing of boxes, along with the way Lazarus looks as he moves, seems to make sense because of the everyday **physics** being considered in the animations. Grapes moving, smiling, and jumping? Boxes coincidentally falling wherever the grape goes? This is impossible. Yet, in video game animation it looks *plausible*.

## DO THIS ON YOUR TUTORIAL GUIDE

**STAGE 3 & 4 TEST AND EVALUATION:** Write an evaluation of your hypothesis and programming properties from STAGE 4.

- If you state "valid", provide a detailed explanation of "why" the object behaves properly based on your hypothesis, and the events and actions that you programmed.
- If you state "invalid", make sure that you expose your errors in reasoning, or your errors in programming.

**TUTORIAL CONTINUES ON NEXT PAGE**



## DO THIS ON YOUR TUTORIAL GUIDE

**STAGES 3 & 4 CONCEPT SUMMARY:** Review the concepts found in the gold boxes throughout the tutorial. Write a full paragraph describing how **physics** concepts are used in the behaviors of the box and Lazarus objects. What are **variables** are used and what is their purpose? What is a **controller object**? Use new vocabulary correctly in your writing.

**END OF STAGES 3 & 4. REVIEW YOUR WORK ON THE TUTORIAL GUIDE. STUDY THE TUTORIAL GUIDE RIGOR SCALE. MAKE REVISIONS AND IMPROVEMENTS BASED ON THE SCALE. UPLOAD COMPLETED TUTORIAL GUIDES TO EDMODO.**