



Target the Player: It's Fun Being Squished

Our third game will be an action game that challenges players to make quick decisions under pressure—and if they're not fast enough then they'll get squished! We'll introduce some new techniques for putting character animation into the game, and show how a *controller object* can be used to help to manage the game.

Designing the Game: Lazarus

As usual we'll need a description of our game. We've named it *Lazarus*, after the biblical character who was resurrected from the dead, because the game once had to be recovered from an old floppy disk that had become corrupted! Always remember to make backups of your data!

Lazarus has been abducted by the Blob Mob, who are intent on bringing this harmless creature to a sticky end. They've imprisoned him at the Blobfather's (sorry) factory, where they are trying to squish him under a pile of heavy boxes. However, they've not accounted for Lazarus's quick thinking, as the boxes can be used to build a stairway up to the power button that halts the machinery. Do you have the reactions needed to help Lazarus build a way up, or will the evil mob claim one more innocent victim?

Each level traps Lazarus in a pit of boxes stacked up on either side of the screen to contain him within the level. The arrow keys will move Lazarus left and right, and he will automatically jump onto boxes that are in his way. However, he can only jump the height of a single box, and stacks two or more boxes high will block his path. New boxes will periodically appear directly above Lazarus's current position and fall vertically down from the top of the screen until they come to rest. This means that the player will be able to use Lazarus's position to control where boxes fall and build a stairway up to the power button.

There will be four different types of boxes, increasing in weight and strength: cardboard, wood, metal, and stone. Falling boxes will come to rest on boxes that are stronger than them, but will crush boxes that are lighter. The type of each box is chosen at random, but the next box will be shown in the bottom-left corner of the window just before it appears. There will be a number of increasingly difficult levels, with higher stairways to build, and boxes that fall faster. When Lazarus gets squished, the level will restart to give the player another try. See Figure 4-1 for an example of how a level will look.

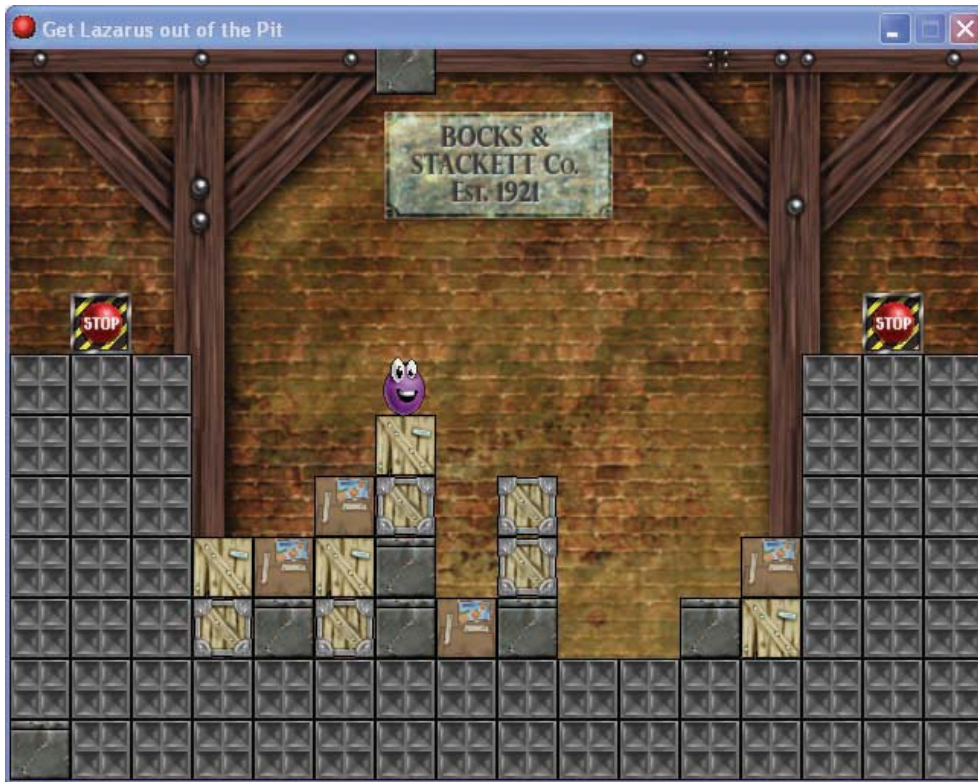


Figure 4-1. This shows how a typical level might look in the Lazarus game.

This may sound rather simple, but—as you’ll see—it actually makes for a very challenging game! All the resources can be found in the [Resources/Chapter04](#) folder on the CD.

An Animated Character

Our first task will be to create the Lazarus character. We’ll give him comical animations for when he’s moving, jumping, and being squished to add to the appeal of the game. This will require a number of different sprites and several different Lazarus objects as well. Like the different behaviors of the rocket in Galactic Mail, using several objects helps us to separate the different animations of Lazarus in a simple way.

The animations for Lazarus have been designed around the size of the boxes in the game. All the boxes are exactly 40×40 pixels in size, so the animations that show Lazarus jumping from one box to the next need to be as tall and wide as two boxes (80×80 pixels). This means we’ll be working with sprites of different sizes, and have to think carefully about where to place the origin of each sprite so that they match up correctly. Remember that Game Maker acts as if it is holding each sprite by its origin as it moves around the screen; so all the origins need to be at the same position relative to Lazarus—regardless of the size of the sprite. This should begin to make sense as you complete the steps that follow.

Tip Setting the **Smooth Edges** property in the Sprite Properties form can often make sprites appear less pixilated (blocky) in the game.

Creating the Lazarus sprite resources for the game:

1. Create a new sprite called `spr_laz_stand` using `Lazarus_stand.gif` from the `Resources/Chapter04` folder on the CD. This sprite is 40×40 pixels and shows Lazarus in his “normal” position. Remember that the origin for sprites defaults to the top-left corner (X and Y both set to 0). We'll leave this where it is and make sure that the origins of all the other sprites match up with this position. Click the **OK** button to close the form.
2. Create another sprite called `spr_laz_right` using `Lazarus_right.gif`. This sprite is 80×80 pixels and shows Lazarus jumping 40 pixels to the right (use the blue arrows to preview the animation). As usual, the origin has defaulted to the top-left corner of the sprite, but the top-left corner is further above Lazarus's head than in the last sprite. To match up with the previous sprite, we need to move the origin down by 40 pixels—so set the Y value to 40. The properties form should now look like Figure 4-2.

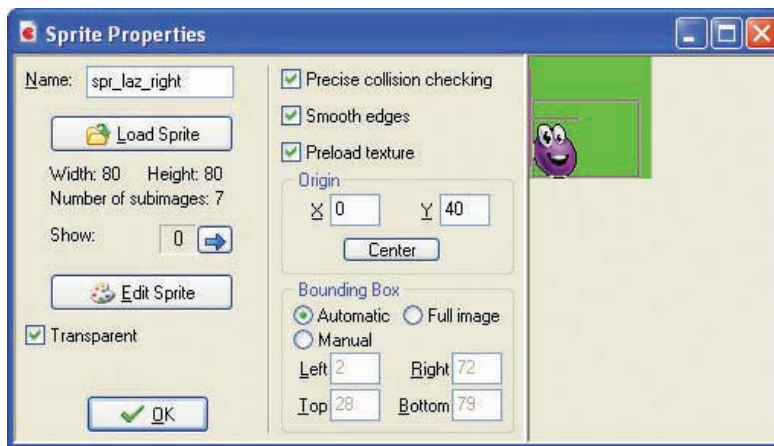


Figure 4-2. The origin for this sprite has been moved to halfway down the left-hand side.









3. Create a `spr_laz_jump_right` sprite in exactly the same way using `Lazarus_jump_right.gif` (with an X value of 0 and a Y value of 40).
4. Create a `spr_laz_left` sprite using `Lazarus_left.gif`. This sprite is also 80×80 pixels and shows Lazarus jumping 40 pixels to the left, but this time Lazarus starts on the bottom-right side of the sprite. This means we need to move the origin 40 pixels down and 40 pixels right to place it at the same relative position as before. Set both the X and Y values to 40 and close the form.
5. Create a `spr_laz_jump_left` sprite in exactly the same way using `Lazarus_jump_left.gif` (with an X value of 40 and a Y value of 40).
6. Create two more sprites called `spr_laz_afraid` and `spr_laz_squished` using `Lazarus_afraid.gif` and `Lazarus_squished.gif`. These are 40×40 pixels so there's no need to change the origin.

These are all the sprites we need for Lazarus, so our next step is to make some objects for him. The main object will be the “normal” standing Lazarus. This is the most important one, as it will react to the player's keyboard input. The others are only there to play the different

animations, after which they turn themselves back into the standing Lazarus. They will also move Lazarus to a new position that corresponds to the final frame of the animation.



We have a bit of a chicken-and-egg situation here, as we did with the two types of rockets in Galactic Mail. The “normal object” will need actions to turn it into “animating objects” (which don’t exist yet) and the “animating objects” will need actions to turn them into “normal objects” (which also don’t exist yet). So which objects do we create first? Well, the answer is that we create the “normal object” but come back to creating its events and actions after we have created the “animating objects”—crafty, eh?

Creating Lazarus object resources for the game:

1. Create a new object called `obj_laz_stand` and give it the standing Lazarus sprite.
2. Press **OK** to close the properties form (we will come back to it later).
3. Create a new object called `obj_laz_right` and give it the sprite that hops one box horizontally to the right (`spr_laz_right`).
4. Add an **Other, Animation End** event. Remember that this event happens when a sprite reaches the last subimage in its animation.
-  5. Include the **Jump to Position** action in this event (**move** tab). Set **X** to 40 and **Y** to 0, and make sure that the **Relative** option is enabled. As the boxes are all 40×40 pixels, this will move Lazarus exactly one box to the right at the end of the animation.
-  6. Also include the **Change Instance** action (**main1** tab) below this and select `obj_laz_stand` as the object to change back into.
7. Click **OK** to close the object properties form.
- 
 8. Create another object called `obj_laz_left` and give it the sprite that hops one box horizontally to the left (`spr_laz_left`). Repeat the same process as before (steps 4–7), but set **X** to -40 for the **Jump to Position** action.
- 
 9. Create another object called `obj_laz_jump_right` and give it the sprite that hops up one box diagonally to the right (`spr_laz_jump_right`). Repeat the process, setting **X** to 40 and **Y** to -40.
- 
 10. Add a final object called `obj_laz_jump_left` and give it the sprite that hops up one box diagonally to the left (`spr_laz_jump_left`). This time, set **X** to -40 and **Y** to -40.


We may as well get the squished Lazarus object out of the way now too—even though we won’t need it for a while. Once its gruesome animation finishes, this object will display a message to tell the player that they’ve been squished. This isn’t because we think they are too stupid to notice, but it provides a useful pause before starting the level again! We’re not going to add lives or high scores in this game, so we’ll simply restart the level to give the player another try.

Creating the squished Lazarus object resource:



1. Create an object called `obj_laz_squished` and give it the squished Lazarus sprite.
-  2. Add an **Other, Animation End** event and include the **Display Message** action (**main2** tab) in it.
3. Type something like “YOU'RE HISTORY!#Better luck next time” into the message properties. Note that putting the # symbol in the middle of the message will start a new line from that point.
-  4. Finally, include the **Restart Room** action (**main1** tab) after the message action and press **OK** to close the object properties form.

Okay, so now we have these animation objects in place, we can continue making the main standing Lazarus object we started on the previous page. One of its main jobs is to change into the appropriate animating object when the player presses a key. The appropriate object depends on whether Lazarus is standing next to any boxes. We'll use a conditional collision action to help Game Maker work this out for us.

Adding a right key event for the standing Lazarus object:

1. Reopen the properties form for the `obj_laz_stand` object by double-clicking it in the resource list.
-  2. We'll start by creating actions to handle moving to the right. Add a **Key Press, <Right>** event and include the **Check Collision** action in it (**control** tab).
3. This action allows us to check that there *would* be a collision if we moved this instance to a particular position on the screen. We need to make sure that Lazarus is on solid ground before allowing him to move, as he shouldn't be able to jump when he is standing on thin air! To check that this is the case, we set **X** to 0 and **Y** to 8 (slightly below his current position), and enable the **Relative** option.

Note Conditional collision actions have an **Objects** option, which allows us to choose between checking for collisions with all objects or only ones marked as solid. We're leaving this set to only solid, so we need to remember to set the **Solid** property later when we create the box objects.

-  4. All of the remaining actions in this event depend on the previous condition (they only need to be called if it is **true**). Consequently, we'll need to include them all between **Start Block** and **End Block** actions. Include the **Start Block** action now.
-  5. Now include the **Check Empty** conditional action. This conditional action is the opposite of the last one: it checks that there *wouldn't* be a collision if we moved to a particular position on the screen. So to check that the space to the right of Lazarus is free, set **X** to 40 (the width of a box), set **Y** to 0, and enable the **Relative** option.



6. Include the **Change Instance** action (**main1** tab) and select the `obj_laz_right` object. Select **yes** to **Perform Events**. This means that the **Create** event of the object we're turning into *will* get called (which is important later when we add sound effects).

Note The **Perform Events** option controls whether the **Destroy** event of the current object and the **Create** event of the new object should be called. This isn't usually necessary so it does not call them by default.



7. Next include the **Else** action from the **control** tab (more about this in a moment).



8. Include another **Check Empty** conditional action directly after this. This should verify that there are no boxes diagonally, up, and to the right of Lazarus. Set **X** to `40` and **Y** to `-40`, and enable the **Relative** option.



9. Next include the **Change Instance** action and select the `obj_laz_jump_right` object. Select **yes** to **Perform Events**.



10. Finally, include an **End Block** action to conclude the actions that should be performed if Lazarus is on solid ground. The list of actions now should look like Figure 4-3.

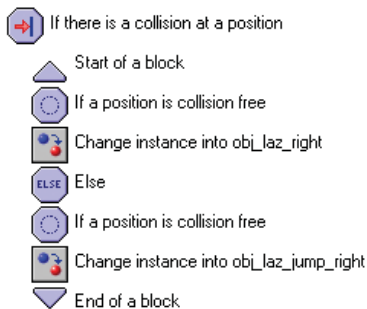


Figure 4-3. Here are the actions for moving or jumping to the right.

This is the first time we've used the **Else** action, but it is often used alongside conditional actions in this way. On its own, a conditional action only allows you to specify actions that should be performed if a condition is true. However, in combination with **Else**, you can specify different actions to be performed if that same condition is not true. This has many uses, but in this situation it allows us to ask sequences of questions like this:

Is there solid ground beneath Lazarus's feet? Yes. Well, is there a free space to the right of Lazarus? No—there's a box in the way. Okay, well, is there a free space on top of that box then? Yes—let's jump on top of it.

This is just one possible outcome, but our actions provide outcomes for four different situations: not moving when falling through the air; moving horizontally to the right when no boxes are in the way; jumping diagonally to the right when a single box is in the way; and doing nothing at all when more than one box is in the way. You can think of this action list as reading something like this:

If the position below has something solid in it, then read the next sentence. If the position to the right is collision free, then change into object `obj_laz_right`; else, if the position diagonally right is collision free, then change into object `obj_laz_jump_right`.

Before continuing, go through the actions step by step in your head and try to work out how you end up with each of these different outcomes (move right, move diagonally right, and no movement). When you're happy that this makes sense, we'll move on and do the same thing for the left arrow key.

Note Like other conditional actions, the **Else** action can be used with or without blocks. If blocks are not used, then the **Else** only affects the action that immediately follows it.

Adding a left key press event to the standing Lazarus object:



1. Add a **Key Press**, `<Left>` event and include the **Check Collision** action. Set **X** to 0 and **Y** to 8, and enable the **Relative** option (this checks below).



2. Include a **Start Block** action.



3. Include the **Check Empty** conditional action (**control** tab) with **X** set to -40, **Y** set to 0, and the **Relative** option enabled (this checks left).



4. Next, include a **Change Instance** action (**main1** tab) and select `obj_laz_left`. Choose **yes** to **Perform Events**.



5. Now include **Else** action from the **control** tab.



6. Include a **Check Empty** action with **X** set to -40, **Y** set to -40, and **Relative** enabled (this checks diagonally left).



7. Include a **Change Instance** action and select the `obj_laz_jump_left` object. Choose **yes** to **Perform Events**.





8. Finally, include an **End Block** action to finish the block of actions.

Although our keyboard events stop Lazarus from jumping in mid-air, there aren't yet any events to make him fall down to the ground when he is. We'll get Game Maker to test for this in a **Step** event so that it is continually checking to see if he should be falling. However, we need to think carefully about how far he should fall in each step. The amount of movement in each step will determine how fast he falls, but it will make our job much simpler if we also choose a number that divides exactly into 40 (the height of the boxes). Can you think why?

Let's imagine that we chose a number that doesn't divide into 40, like 12. Lazarus would have fallen 12 pixels after one step, 24 pixels after two steps, 36 pixels after three steps, and 48 pixels after four. At no stage has Lazarus fallen the exact 40 pixels needed to fall the height of one box; he is either 4 pixels too high (at 36 pixels) or 8 pixels too low (at 48 pixels). This means he would either end up floating above boxes, or jammed somehow into them! Using any number that divides into 40 will avoid this problem (1, 2, 4, 5, 8, 10, 20, or 40), so we've chosen a value of 8 because it produces a sensible-looking falling speed.

Adding a step event to the standing Lazarus object to make it fall:

1. Add the **Step, Step** event to the standing Lazarus object. We are using the “standard” **Step** event as we don't really care exactly when Lazarus falls, provided he does.
-  2. Include a **Check Empty** action in the **Step** event, setting **X** to 0 and **Y** to 8, and enabling the **Relative** option. This action checks for empty space just below Lazarus.
-  3. Include a **Jump to Position** action directly after it so that it will only be performed if the **Check Empty** condition is true. We need to give it the same relative settings as before, so that it moves into the empty space. Set **X** to 0 and **Y** to 8, and enable the **Relative** option.

A Test Environment

We've gone through quite a lot of steps so far without being able to test our work, so before going any further let's quickly create a test level for Lazarus to move around in. There are no falling boxes yet, so we'll have to create some random stacks of our own to check if the movement is working correctly. We'll create just one box type to do this: the boxes that make up the walls of the pit.

Creating the wall object resource for the game:

1. Create a new sprite called `spr_wall` using `Wall.gif`. Disable the **Transparent** option as the walls for this level need to look completely solid.
2. Create a new object called `obj_wall` and give it the wall sprite. Enable the **Solid** option so that the checks in the standing Lazarus object can detect the wall.
3. Create a new room called `room_test` and provide a caption in the **settings** tab.
4. Look in the toolbar at the top of the Room Properties form and set both **Snap X** and **Snap Y** to 40. All our boxes are 40×40 pixels, so this will help us to place them neatly on the level. The grid in the room will change accordingly.
5. Switch to the **objects** tab again and select the wall object to place. Create a level with a number of boxes that form flat areas and staircases (remember, you can hold the Shift key to add multiple instances). Also add one instance of the standing Lazarus object. Try to make it look something like Figure 4-4.

Note Sometimes when you close a room form you get a warning message saying that there are instances outside the room. This can happen when you accidentally move the mouse outside the room area while adding objects. You will be asked whether these instances should be removed—simply click the **Yes** button.

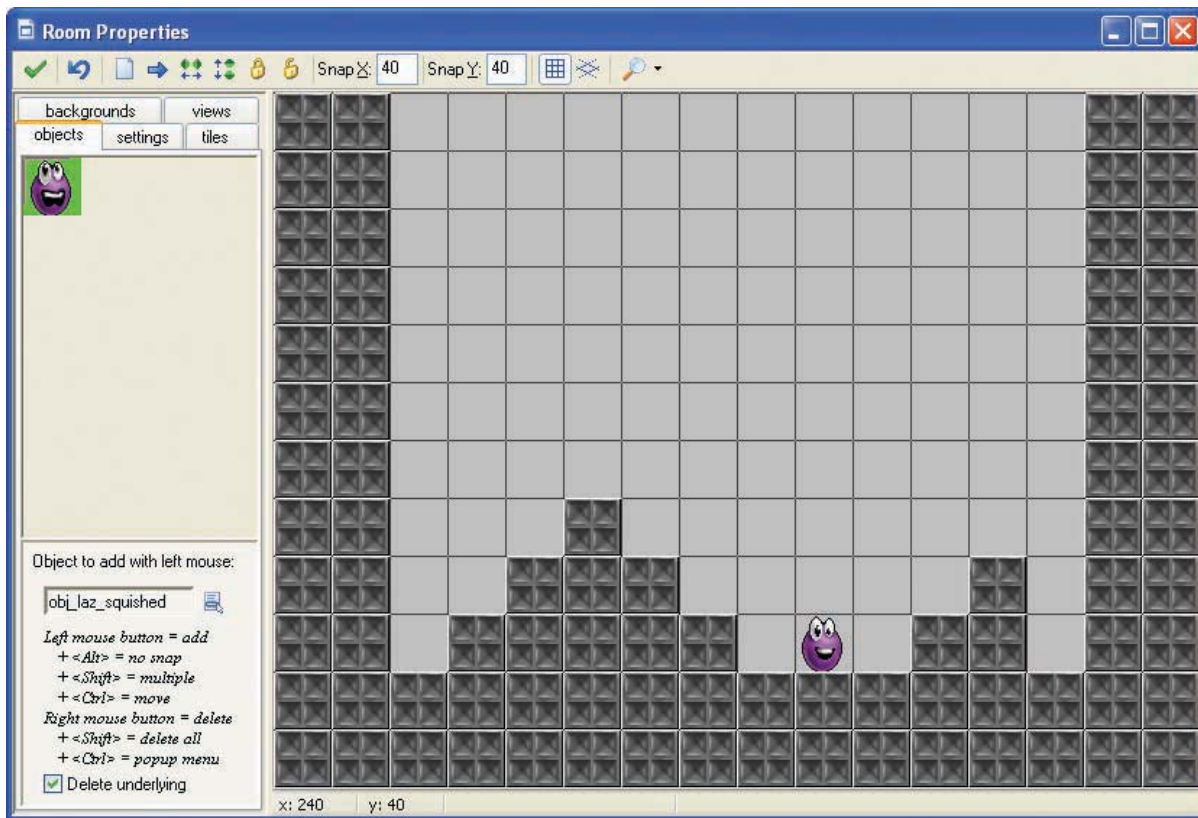


Figure 4-4. Your test level should look something like this.

At last, you can finally run the game! Test the character's movement in all the possible situations and make sure that he behaves the way you would expect. If something isn't working right, then check your steps carefully, making sure that you enabled the **Relative** option in all the actions where it was indicated. Alternatively, you can load the current version from the file [Games/Chapter04/lazarus1.gm6](#) on the CD.

Falling Boxes

Our next goal is to create the falling boxes that both threaten the player and provide the means for their escape. As indicated in the game description, there will be four types of boxes in the game: stone boxes, metal boxes, wooden boxes, and cardboard boxes. As you would expect, stone boxes are the heaviest and cardboard boxes are the lightest. Falling boxes are chosen at random and heavier boxes will crush lighter boxes as they fall—making it harder to build a stairway out of the pit. However, to give the player a chance to think ahead, the next box will be shown in the corner of the screen while the last box is still falling.








Each box will need to change its behavior three times in the game: first it appears in the corner, as the “next box”; then it falls down the screen until it lands on another box; and finally it forms a stationary obstacle for Lazarus to negotiate. As you may have guessed, we will achieve this by creating three different objects for each box: one for each behavior. We will start by creating the stationary boxes, as they are the simplest to make. First, though, we need to create some new sprites.

Creating new box sprite and object resources for the game:

1. Create sprites called `spr_box_stone` and `spr_box_card` using `StoneBox.gif` and `CardBox.gif`. Disable the **Transparent** option on both these sprites.
2. Now create sprites called `spr_box_metal` and `spr_box_wood` using `MetalBox.gif` and `WoodBox.gif`. This time leave the **Transparent** option enabled, as these two sprites have a small amount of transparency around the edges.
3. Create a new object called `obj_box_stone` and give it the sprite for the stone box. Set the **Solid** option so that it is detected in collision tests.
4. Repeat the previous step to add objects for `obj_box_metal`, `obj_box_wood` and `obj_box_card`.

Next we'll make the falling boxes. These need to start at the top of the screen, directly above Lazarus's horizontal position, so we'll make use of the `x` variable of the standing Lazarus object to tell us where that is. Once it starts falling, we'll give it a speed of 5 because that divides exactly into 40 (important for the same reasons as before) and it is slightly slower than the speed that Lazarus falls (otherwise a box might squish Lazarus in the air!). When a box collides with a heavier box, it turns into a stationary box, but when it collides with a lighter box, it destroys that box and continues to fall.

Creating falling box objects for the game:

1. Create a new object called `obj_falling_stone`, give it the sprite for the stone box, and select the **Solid** option as before.
2.  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`. This will make the box start above Lazarus, just out of view at the top of the screen.
3.  Next include the **Move Fixed** action, using a downward direction and a **Speed** of 5.
4.  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**.
5.  Add another **Collision** event, this time with `obj_wall`. This needs to stop the box moving, so include a **Move Fixed** action and select the middle square with a **Speed** of 0. Also include a **Change Instance** action, and select the stationary box `obj_box_stone`.
6.  Add a third **Collision** event with `obj_box_stone` and include the same two actions as the **Collision** event with the wall above (you could copy them).
7.  Add a fourth **Collision** event with `obj_box_metal`. The metal box is lighter than the stone box so it must be crushed. Include a **Destroy Instance** action and select the **Other** object.
8.  Add fifth and sixth **Collision** events with `obj_box_wood` and `obj_box_card`, both including identical **Destroy Instance** actions as we did in step 7 to destroy the **Other** box in the collision.

Okay, that's one of the falling boxes. The other falling boxes are similar but need to behave slightly differently when they collide with different kinds of boxes.

9. Create the remaining three falling objects for the other types (`obj_falling_metal`, `obj_falling_wood`, and `obj_falling_card`). Repeat steps 1–8 for each one, using step 7 when a box crushes another box and step 5 when a box stops moving. Refer to Table 4-1 when deciding which boxes should crush each other.

Table 4-1. *Box Materials That Should Crush Each Other*

Material	Material(s) That It Crushes
Stone	Metal, Wood, and Card
Metal	Wood and Card
Wood	Card
Card	None

Phew! That was quite a lot of work (28 events and 46 actions), made worse by the fact that we had to repeat the same steps over and over again. In Chapter 6 we will see that there is actually a quicker way to do this kind of thing using *parents*. Nonetheless, although this might have seemed like a lot of effort, it may help you to appreciate the work that goes into a commercial game. They usually take at least 18 months to program and require hundreds of thousands of lines of code to make them work!

Now let's set about creating the final set of boxes that appears in the bottom-left corner to show the player which box is coming next. This adds an important element of gameplay, allowing the player to plan ahead and adapt their strategy based on where it would be most useful for the next box to fall. It requires quick thinking and takes a bit of practice, but it helps to create a challenging and rewarding game. The “next box” objects are very simple to make, but we'll need four of them again—one for each type of box.

Creating next box object resources for the game:

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.
2. Create objects for `obj_next_metal`, `obj_next_wood`, and `obj_next_card` in the same way.

I'm sure you'll be relieved to find out that's all the boxes we need to create for this game! However, while the falling boxes have actions to turn them into stationary boxes, there are no actions yet for turning next boxes into falling boxes, or creating next boxes in the first place. That's because we are going to create a *controller object* to do this. A controller object is usually an invisible object (it doesn't have a sprite), which performs important actions on other objects. Our controller object will use a **Step** event to continually check if there is a falling box on the level. If not, then it will turn the current next box into a falling box and create a new next box. In this way, the controller object will maintain a constant cycle of new and falling boxes until the level is completed—or the player gets squished!

Creating a controller object resource for the game:

1. Create a new object called `obj_controller` and leave it without a sprite.
2. Add a **Step, Step** event and include the **Test Instance Count** conditional action (**control** tab). This counts the number of instances of a particular object on the level and tests it against a value. Choose the `obj_falling_stone` object; leave **Number** as **0** and **Operation** as **Equal to**. This creates a condition that is true if the number of falling stone box instances on the level is equal to 0 (i.e., there aren't any!).
3. Include three more **Test Instance Count** conditional actions to check if there are no instances of `obj_falling_metal`, `obj_falling_wood`, and `obj_falling_card` in the same way. When combined, these conditional actions will make sure that there are no falling boxes of any kind on the level before creating a new one.
4. Include a final **Test Instance Count** action for the `obj_laz_stand` object, but set **Number** to **1** and the **Operation** to **Equal to**. This makes sure that there is an instance of the standing Lazarus object on the level, rather than any of the animating objects.
5. Include a **Start Block** action. This will group together all the actions that need to be performed to create the new box.
6. Include a **Change Instance** action and select **Object** for **Applies to**, so that it changes *all* instances of one kind of object on the level into another. Set **Object** to `obj_next_stone`, **Change Into** to `obj_falling_stone`, and select **yes** to perform events (the **Create** event for the falling box needs to be performed to start it in the correct position). This will turn any stone next boxes into stone falling boxes. The action should now look like Figure 4-5.



Figure 4-5. Change the next box into a falling box.



7. However, because the type of box will be chosen randomly, we don't know what kind of next box object the next box will actually be. To cover all bases, 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` in the same way.



8. Next we need to randomly create one of the next box objects. Include a **Create Random** action (**main1** tab) and select the four different next box objects. Set **X** to 0 and **Y** to 440, and leave **Relative** disabled. Remember that when **Relative** is disabled, **X** and **Y** are measured from the top-left corner of the screen. These coordinates will therefore put the new next box where it should be in the bottom-left corner of the screen. The action should now look like Figure 4-6.



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

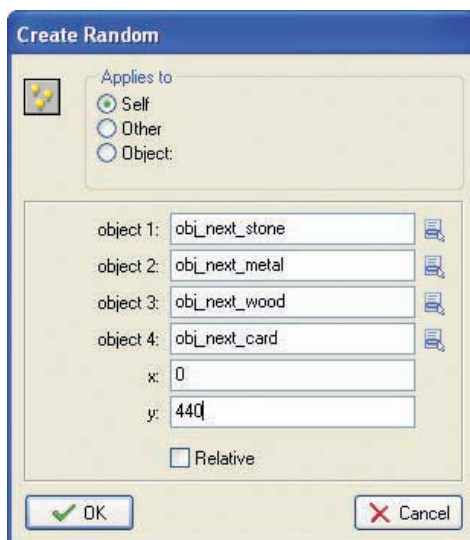


Figure 4-6. The **Create Random** action allows us to randomly create one of the four next box objects.

This long list of conditional actions means that the block of actions will only be performed if all these conditions are true. In other words, if there are no instances of `obj_falling_stone` and no instances of `obj_falling_metal` and no instances of `obj_falling_wood` and no instances of `obj_falling_card` and one instance of `obj_laz_stand`, then Game Maker will create a new box.

You might have thought it a bit odd that we need to check that there is a standing Lazarus object as part of our conditions for creating new boxes. If you look back at one of the **Jump To Position** actions in the **Create** events of the falling boxes, you will remember that we use the `obj_laz_stand.x` variable to start the object in the correct position. However, Game Maker can't provide that object's x position if it has turned into an animation object, so it will create an error in the program. So to avoid this possibility we check that there's a standing Lazarus instance on the level before creating new falling boxes.

Now it's finally time to test our new objects.

Editing the test room to add new instances:

1. Reopen the test room we created by double-clicking on it in the resource list.
2. Remove all the extra wall instances so that it leaves just a pit with walls on both sides and across the bottom.
3. Add one instance of the controller object into the room (easily forgotten!)

Note When an object has no sprite, it shows up in the Room Properties form as a blue ball with a red question mark. This will not appear in the game, but reminds us that this (invisible) object is there when we are editing the room.

Now run the game and test it carefully. Make sure that the box that appears in the bottom left is actually the box that falls down the screen next and check that heavier boxes are crushing lighter ones. As usual, if there are any problems, then carefully check the instructions or load the game from [Games/Chapter04/lazarus2.gm6](#) on the CD.





Finishing Touches

We now have all the basic ingredients of the game in place and there are just a few more things to do before we could call it a finished game. There's no way to complete a level yet, so we need to include the stop buttons that will halt the boxes and move the player onto the next level. Some sound effects would also be nice—as would a background and a title screen. We're obviously going to need a few different levels, too. However, before all that we're going to add something cool that will endear the player to Lazarus's plight a little more.

No Way Out!

You may have noticed that there's another animation we haven't used yet that shows Lazarus looking afraid. We're going to show this animation when he's in a hopeless situation and knows he is about to meet his end. However, rather than create a new object for this animation like we did with the others, we're just going to change the sprite of the standing Lazarus object when he becomes afraid. We can do this because “being afraid” does not need any actions of its own: it has exactly the same behavior as standing—it just looks different. We're going to control this animation within the **Step** event, so that the correct animation is chosen at any point in time. We will use **Check Collision** actions to detect if Lazarus is surrounded by stacks of boxes two or more high on both sides. The **Check Collision** action performs actions only when there *is* a collision at a particular point. In this way, we can detect whether Lazarus is trapped on all sides and set his animation to be afraid.

Editing the standing Lazarus object to detect for being trapped:

1. Reopen the standing Lazarus object and select its **Step** event, so that you can see the existing actions for this event.
-  2. Include the **Check Collision** conditional action (**control** tab) below the last action in the list. Set **X** to 40 and **Y** to 0, and enable the **Relative** option. This checks for a box to the right of Lazarus.
-  3. Include another **Check Collision** action with **X** set to 40, **Y** set to -40, and the **Relative** option enabled. This checks for a box diagonally to the right of Lazarus.
-  4. Include two more **Check Collision** actions: one with **X** set to -40 and **Y** set to 0, and the other with **X** set to -40 and **Y** set to -40. Both should have the **Relative** option enabled. These check for boxes to the left and diagonally to the left of Lazarus.
-  5. Finally, include a **Change Sprite** action, using the “afraid Lazarus” sprite. This will now only happen if the four conditional actions above are true and Lazarus is literally boxed in.

Hopeless as this situation may sound, it is actually possible for Lazarus to be saved from this predicament by a heavy block crushing the stack of boxes on one side of him. If this happens, then we would like Lazarus to stop being afraid. We *could* include conditional actions to check for this happening and change his sprite back to normal. However, we can achieve the same effect simply by including a **Change Sprite** action at the very beginning of the list of actions for this event. Changing into the standing Lazarus sprite by default will make the sprite revert back to normal if he stops being trapped.

Editing the standing Lazarus object to detect for being freed:








1. Select the **Step** event for the standing Lazarus object so that you can see the existing actions for this event.
2. Include a **Change Sprite** event at the very beginning of the list of actions (you can drag actions about if it falls in the wrong place). Set it to change into the standing Lazarus sprite.

You might want to play the game now and make sure that this new feature is working correctly. Features like this don't change the gameplay directly, but add to the playing experience and make the game more entertaining to play.

Adding a Goal

The player's goal is to reach one of the stop buttons, so that it halts the machinery and stops dropping the boxes. However, in practice all the buttons really need to do is move the player onto the next level when the standing Lazarus object collides with them. If there are no more rooms, then it will show a completion message and restart the game.



Creating a new button object resource for the game:

1. Create a new sprite called `spr_button` using `Button.gif`.
2. Create a new object called `obj_button` and give it the button sprite. Set **Depth** to `10` so that it appears behind other objects.
-  3. Add a **Collision** event with the standing Lazarus object and include a **Sleep** action in it (**main2** tab). Set **Milliseconds** to `1000` (1 second) and **Redraw** to true. This should give a brief pause for the player to realize they have completed the level.
-  4. Include a conditional **Check Next** action (**main1** tab).
-  5. Include a **Next Room** action (**main1** tab).
-  6. Include an **Else** action followed by a **Start Block** action.
-  7. Include a **Display Message** action (**main2** tab) and set **Message** to something like `"CONGRATULATIONS#You have completed the game!"`
-  8. Include a **Different Room** action and set **New Room** to the first room (which is the only room at the moment).
-  9. Finally, include an **End Block** action and close the object properties.
10. Edit your test room and add a stop button on either side at the top of the pit.

Starting a Level

At the moment, boxes start falling as soon as the player enters the level, leaving them with no time to gather their thoughts and prepare their strategy. We're going to help them out by creating a starter object that displays the title for a couple of seconds before changing itself into the controller object and starting to drop boxes.

Creating a new starter object resource for the game:

1. Create a new sprite called `spr_title` using `Title.gif`.
2. Create a new object called `obj_starter` and give it the title sprite.
-  3. Add a **Create** event and include a **Sleep** action in it. Set **Milliseconds** to `2000`, for a wait of two seconds.
-  4. Include the **Change Instance** action and select the controller object. Close the object properties.
5. Edit your test room, and remove the controller object using the right mouse button. Add the starter object at an appropriate place instead.

Note You may have noticed that the title doesn't appear on the first level when you run the game. This is because the starter object's **Create** event is executed before the window appears, so it has already turned into a controller object by the time we see the room. This can be remedied using an **Alarm** action to add a delay, but we won't worry about this for now, and we'll come back to alarms in Chapter 6.

Sounds, Backgrounds, and Help

It's about time we made the game feel a bit more professional by including sound effects and music in the game. This is quite simple and you can probably handle most of this on your own by now, but here are some pointers to help you on your way:



1. All the sound resources can be found in the [Resources/Chapter04](#) folder on the CD.
2. You'll need to add sounds for [Music.mp3](#), [Wall.wav](#), [Crush.wav](#), [Squished.wav](#), [Move.wav](#), and [Button.wav](#) and play them at the right times using the **Play Sound** action (**main1** tab).
3. A good place to start playing the music would be in a new **Game Start** event for the controller object. You'll find the **Game start** event in **Other** events. Don't forget to set **Loop** to true in the **Play Sound** action to make the music loop forever.
4. You'll need to add crush or wall sound effects to the existing **Collision** events between falling box objects and stationary box objects.
5. Add a new **Create** event for the squished Lazarus object, and play the squished sound effect there. This will save you the trouble of putting it in each of the four collision events between falling boxes and Lazarus.
6. Adding **Create** events to play the move sound effects would also be a good way of handling the four moving Lazarus objects.
7. Finally, you'll need to play the button sound effect in the **Collision** event between the button and Lazarus.

Test the game and make sure all the sound effects are playing in the correct place. If you don't hear a sound when moving around, check that you set **Perform Events** to **yes** in the **Change Instance** actions that change into the animating objects. If you didn't, then Game Maker won't perform the **Create** events that contain the sound effects.

A backdrop to the levels would also improve the look of the game, and we should put together some kind of help text for the player too.

Creating a background resource and Game Information:

1. Create a background using `Background.bmp` from `Resources/Chapter04` on the CD.
2. Reopen the properties form for the room and select the **backgrounds** tab. Select the new background from the menu halfway down on the left.
3. Double-click on **Game Information** in the resource list and add a help text for the game. Remember to include the name of the game and who it was created by (you), along with a short description of the aims and controls.

Levels

All that is left now is to create a variety of levels for your game. We talk about level design in much more detail in Chapter 8, but it's probably best to start with shallow pits and buttons on each side to keep things fairly easy. However, as the levels progress they can become as deep and narrow as you like! Making the floor of the pit higher will make the level harder, as the player has less time to react to the falling boxes. You could also place stationary boxes in unhelpful places or place the buttons in mid-air to vary the challenge. One sure way to make the game more challenging is to increase the **Speed** setting on the **settings** tab for each level. This controls the number of steps per second on each level. It defaults to 30 steps per second, but higher numbers will make the game faster and harder and lower numbers will make it slower and easier.

Now it's up to you to create some interesting levels for the game. Remember that duplicating rooms will save you a lot of work, so right-click on the room in the resource list and select **Duplicate** from the pop-up menu. Once you've made your levels, let someone else try to play them and see how difficult they find it. Game designers often find their games very easy because they have played them so much, but it is often much harder for everyone else. This is something you should always try to bear in mind when designing your games.

One very last thing: you may find it helpful to add a cheat in your game that allows you to skip between levels. You can do this as follows.

Editing the controller object to add cheats:

1. Open the properties form for the controller object.
2. Add a **Key Press, <N>** event and include the **Next Room** action.
3. Add a **Key Press, <P>** event and include the **Previous Room** action.



Good luck, and don't forget to remove these cheats when the game is finally finished!

Congratulations

You'll find the final version of the game in the file [Games/Chapter04/lazarus3.gm6](#) on the CD. You might want to extend the game a bit further by adding opening and closing screens, or adding a scoring system to the game so that players can compete for the highest score. If you're feeling particularly adventurous, why not try adding some bonuses that sometimes appear when boxes are crushed by each other? One of these could even transform all the stationary boxes into stone boxes—or card boxes if you're feeling mean!

By making this game, you have learned how to animate characters, both by creating different objects and by switching sprites. You have also seen how to use a controller object to manage the game, plus you've learned how to use **Else** actions to provide extra control over the outcome of conditional actions. In fact, you've learned a lot about Game Maker over these past few chapters, and it's about time we gave you a bit of a break. With this in mind, the next chapter is all about game design and you won't have to go near events and actions again until Chapter 6. In the meantime, we'll be thinking more carefully about the designs behind the games we've made so far, and we'll be exploring what makes them fun to play.



