V 0.2

SPYDER

INTRODUCTION TO GAME DEVELOPMENT
PART 3 : SPY DARE

SUMO DIGITAL

INTRODUCTION

Important Note

These worksheets complement the accompanying video tutorial, and **neither will make complete sense without the other**. The video introduces the concepts behind the workshop and this document takes you step-by-step through making the game. Once you reach the end of the worksheets you can return to the video for a summary of the key concepts and some suggestions of how to extend and improve your game. Good luck!

Video **Worksheets** **Video**

Resources

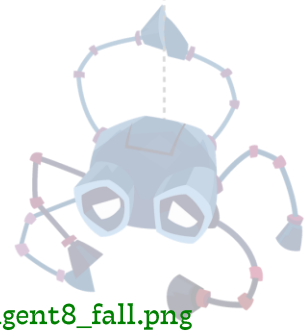
You will also need to obtain the archive of images and sound effects that are used to make the game in this tutorial. You should find a download link for this archive in the text description of the video. Download this file and unzip the archive's contents into a directory on your computer before beginning (usually this involves right- clicking on the .zip file and selecting "Extract Here"). The files and folders this creates are referred to in the tutorial, so keep a note of where you have put them for future reference.

Legal Notices

This course has been created and taught by Dr. Jacob Habgood, who would like to kindly acknowledge Dr. Mark Overmars and APress for the use of example materials derived from The Game Maker's Apprentice (Habgood and Overmars, 2006).

The tutorial assets are derived from the original Spyder™ game by Sumo Digital Ltd (© 2020 Sumo Digital). Permission is granted to use these resources for educational use only.

A SPIDER WITH A MASK



Setting up the sprite resources for a new project:

1. Create a new "Drag and Drop" project in GameMaker Studio 2.
2. Create three new sprites using agent8_dead.png, agent8_stand.png and agent8_fall.png from the Resources folder of the downloaded archive and name the Sprites accordingly.
3. Set the **Origins** of these three Sprites to 48 x 32 and set the **Collision Mask** settings to a **Manual Rectangle** with **Left** set to 22, **Right** to 74, **Top** to 40 and **Bottom** to 79. All three sprites have the same sized collision mask relative to the Origin of the sprite: this helps to make sure that when objects switch between these different sprite images they continue to collide consistently with other objects in the game.
4. Create another sprite using agent8_hop_strip11.png and name it accordingly. Set the **Origin** to 48 x 128 and set the **Collision Mask** settings to a **Manual Rectangle** with **Left** set to 118, **Right** to 170, **Top** to 136 and **Bottom** to 172. Set the **Speed** to 30.
5. Create another sprite using agent8_climb_strip10.png and name it accordingly. Set the **Origin** to 48 x 128 and set the **Collision Mask** settings to a **Manual Rectangle** with **Left** set to 118, **Right** to 170, **Top** to 40 and **Bottom** to 76. Also set the speed to 30. These last two sprites have the same sized collision mask relative to the final frame of their animations for the same reason about consistent collisions.
6. Create three more sprites using soft_box.png, wood_box.png and metal_box.png and name them accordingly. These need to have their **Origin** set to Middle Centre and **Collision Mask** settings to a **Manual Rectangle** with **Left** set to 16, **Right** to 111, **Top** to 16 and **Bottom** to 111.

Creating the remaining game sprites:

1. To create the remaining sprites for the game load the appropriate file and set the **Origins** and **Speed** as indicated below.
2. spr_pole has an **Origin** of 48 x 48 with the **Collision Mask** left as the default (Automatic Rectangle)
3. spr_stop_left has an **Origin** of 26 x 48 (to match up with the pole sprite above). The **Collision Mask** should be set to **Manual**, **Rectangle** with **Left:** 20, **Right:** 31, **Top:** 0 and **Bottom:** 95.
4. spr_stop_right has an **Origin** of 70 x 48 (to match up with the pole sprite above). The **Collision Mask** should be set to **Manual**, **Rectangle** with **Left:** 63, **Right:** 75, **Top:** 0 and **Bottom:** 95.
5. spr_solid has its **Origin** set to 48 x 48 and **Collision Mask** left as the default.
6. spr_background has its **Origin** and **Collision Mask** left as the default.

SPYDER

SOUND PLANNING

Creating the game's Sound resources:

1. Right click on **Sounds** in the **Resources** window and select **Create Sound**. Name it snd_music.
2. In the properties form that appears, click on the ellipsis and select the snd_music.wav file from the Resources folder.
3. Close the Sound Properties form.
4. Repeat the process to create sound resources for snd_soft_land, snd_wood_land, snd_metal_land, snd_reset, snd_die, and snd_crunch.

SOLID FOUNDATIONS

Creating some initial objects:

1. Right-click on the **Objects** section of the **Resources** window and choose **Create Object.** An Object Properties form will appear.
2. In the **Name** field, give the object a name. You should call this one obj_solid.
3. Click the ellipsis icon at the end of the sprite field (three dots) and a list of all the available sprites will appear. Select the spr_solid sprite.

4. Add a **Create** event to the list of events and include an **Assign Variable** action with **Name** set to strength and **Value** set to 4. This will make more sense when we create the different boxes in the game, but it will save coming back to do it later.
5. Also create a new object called obj_agent8_stand and give it the spr_agent8_stand sprite from the sprite selection menu. You can close it for now.

Editing the game room:

1. Open up the list of Rooms in the Resource window (click on the little triangle) and double click on **room0**. The Room Editor windows will appear.
2. Select the **Background** layer in the Room Editor window. This should reveal the Background Layer Properties in the pane below, with a familiar-looing Sprite selection tool. Click on the ellipsis icon and select the background sprite (see Figure 3-2).
3. You will also need to set the **Depth** of the background to 1000. This pushes the background image further back in the room, behind other objects.
4. In the bottom left, under Room Settings set the **Width** of the room to 1280 pixels wide and the **Height** to be 720 pixels.

SPYDER

5. Click on the small arrow next to the Toggle Grid button (see figure 3.1) Set both **GridX** and **GridY** to 48 (half the size of the boxes) and make sure the **snap** option is selected. This will make it much easier to place the boxes neatly in the room.

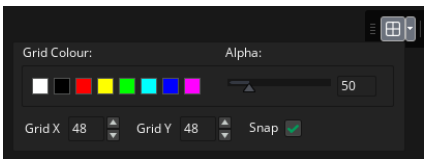


Figure 3-1. The Grid options for the room

6. Click the **Instances** layer in the top left of the Room Editor window (remember you need this selected to place instances of objects into the room).

7. Click and drag instances of obj_solid into the room grid from the Resources. Create a floor and stairway so that the room looks something like Figure 3-2. Also add a single instance of obj_agent8_stand at the foot of the stairway.

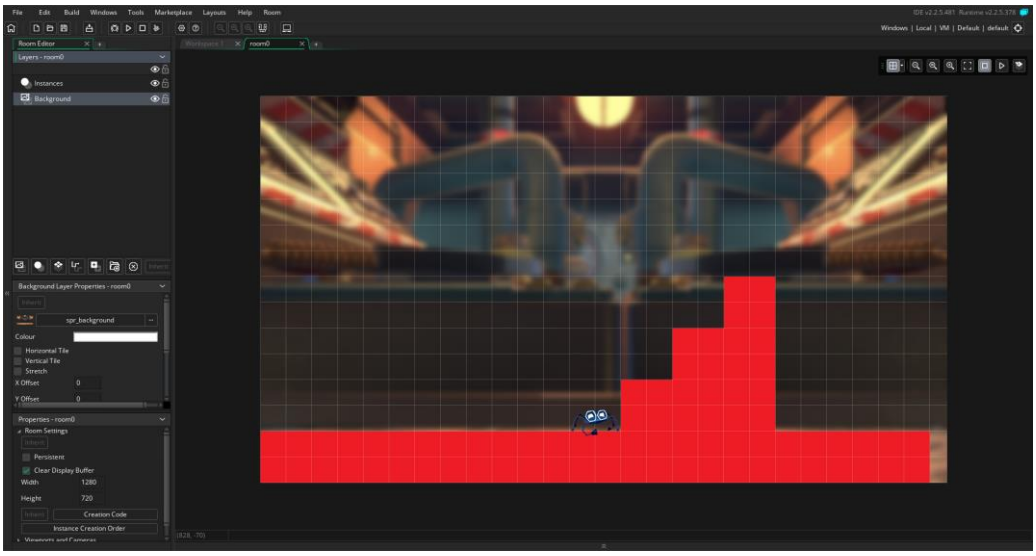


Figure 3-2. The Room Editor with the solid objects and Agent 8.

A SIX-LEGGED STATE MACHINE

Now we are going to start to implement the state-machine we talked about in the video. There are quite a lot of steps before it will work, but stick with it. It is worth understanding this approach as state-machines can help to create robust solutions that are easier to understand and modify as the behaviours of your game characters get more complex.

Programming the standing Agent 8 object:

1. Create new objects for obj_agent8_fall, obj_agent8_dead, obj_agent8_hop, obj_agent8_climb and obj_agent8, and assign them the appropriate sprites. It doesn't matter which sprite you use for the last one as it's a "parent" object used to identify all the different state objects which create Agent 8's behaviours.

2. Reopen obj_agent8_stand and set its **Parent** to be obj_agent8 by clicking on the icon with three circles, next to where it says **Parent**. A dropdown will appear where you can select another object to be that object's parent.

3. Add an **Other, Game Start** event and include a **Macro** action (Common section). Set **Macro** to FACE_LEFT, and **Value** to -1.

2. Include a second **Macro** action, with **Macro** set to FACE_RIGHT, and **Value** to 1. Macros like this help to make code more readable as it's not immediately obvious why -1 should mean left and +1 right. Can you think why we would use these values?

3. Include an **Assign Variable** action, with **Name** set to facing, and **Value** to FACE_RIGHT. This variable simply stores which direction Agent 8 is facing in.

4. Include another **Assign Variable** action, with **Name** set to depth, and **Value** to -1. The depth variable controls which order sprites are drawn in, with lower values indicating that the sprite should be drawn in front of sprites with higher depth values. This makes sure that Agent 8 is always visible in front of other objects.

5. Add a **Create** event and include a **Snap Position** action (Movement section). Set both **Horizontal** and **Vertical** to 48. This helps to make sure Agent8 always enters this state in a "valid" position, aligned with the grid.

6. Include a **Set Speed** action. Leave the **Type** set to Direction and set the **Speed** to 0. This makes sure that Agent8 is not moving when he enters this state.

7. Add a **Key Pressed, Left** event and include an **If Any Object At Place** action (Collision section). Set **X** to -96, **Y** to 0 and select both **Relative** options and the **Not** option. This now checks whether there isn't an object to the left of Agent 8.

8. Attach an **Assign Variable** action to the previous **If** action by dragging the new action onto the red word Empty. Set **Name** to facing and **Value** to FACE_LEFT.

9. Follow this with a **Change Instance** action (which makes it also dependant on the **If** action) and select obj_agent8_hop from the dropdown.

10. Now add an **Else** action (Common section) and link it to the previous **If** as shown in figure 3.3. All the actions that are attached to the **Else** will be triggered when the condition defined by the **If** action is ***not*** met (so in this case, when there ***is*** an object to the left of Agent 8).

11. Attach a second **If Any Object At Place** action to the **Else** action by dragging the new action onto the red word Empty. Set X to -96, Y to -96 and select both **Relative** options and the **Not** option. This checks whether there isn't an object up and left of Agent 8.

12. Attach an **Assign Variable** action to this second **If** action by dragging it onto the red word Empty. Set **Name** to facing and **Value** to FACE_LEFT.

13. Follow this with a **Change Instance** action and select obj_agent8_climb from the dropdown. The actions should now look like figure 3-3.

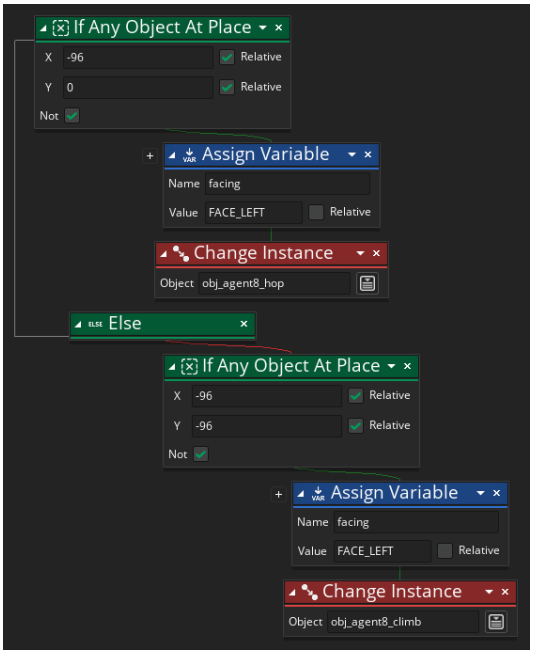


Figure 3-3. The actions for the Key Press, Left event of the standing Agent8 object.

14. Now right-click on the **Key Press, Left** event and select **Duplicate Event.** This time select a **Key Pressed, Right** event for the duplicate.

15. In the new event, change the two **X** values from -96 to 96 (negative to positive) and change FACE_LEFT to FACE_RIGHT in both places that it appears.

16. Add a **Step, End Step** event and include an **If Any Object At Place** action (Collision section). Set **X** to 0, **Y** to 96 and select both **Relative** options and the **Not** option. This now checks whether there isn't an object underneath Agent 8.

17. Follow this with a **Change Instance** action (making it also dependent on the **If** action) and select obj_agent8_fall from the dropdown.

Programming the falling Agent 8 object:

1. Reopen obj_agent8_fall and set its **Parent** to obj_agent8.
2. add a **Create** event and include a **Set Direction Fixed** action the down arrow selected.
3. Include a **Set Speed** action. Leave the **Type** set to Direction and set the **Speed** to 16. This makes Agent8 start to fall when he enters this state.
4. Add a **Collision** event with obj_solid and include a **Jump to Point** action with **X** set to xprevious and **Y** set to yprevious. These are in-built variables which hold the position of the object in the previous frame (i.e. before the collision happened).
5. Follow this with a **Change Instance** action and select obj_agent8_stand from the dropdown.

Programming the hopping Agent 8 object:

1. Reopen obj_agent8_hop and set its **Parent** to obj_agent8.

2. Add a **Create** event and include a **Set Instance Scale** action with **Horizontal** set to facing and **Vertical** set to 1. This flips the horizontal direction of the sprite depending on whether facing is set to FACE_LEFT (-1) or FACE_RIGHT (+1). You can now start to see the reason why we used these specific values to represent left and right!

3. Add an **Animation End** event and include a **Jump to Point** action with **X** set to 96*facing, **Y** set to 0 and both **Relative** options selected. This moves Agent 8 into the next horizontal grid square depending on which direction he was facing. This "jump" in Agent 8's position is completely invisible to the player as the last frame of the hopping animation lines up exactly with the next grid square.

4. Follow this with a **Change Instance** action and select obj_agent8_stand from the dropdown.

5. Add a **Step, Step** event and include an **If Variable** action (Common section). Set **Variable** to image_index, **Value** to 7 and select **Greater** as the comparison. This is another in-built variable which keeps track of the current animation frame. Frame 7 is the point in the hop animation that Agent 8 starts to return to the ground so it makes sense to check whether he should fall at this point. Any later and it looks like he lands in mid-air before falling.

6. Attach an **If Any Object At Place** action (Collision section) to the previous **If** action by dragging the new action onto the red word Empty. Set **X** to 0, **Y** to 96 and select both **Relative** options and the **Not** option. This now checks whether there isn't an object underneath Agent 8 (so he should fall).

7. Attach a **Jump to Point** action to this second **If** action. Set **X** to 96*facing, **Y** to 0 and select both **Relative** options. This moves Agent 8 into the next horizontal grid square depending on which direction he was facing. He will start falling from that point.

8. Follow this with a **Change Instance** action and select obj_agent8_fall from the dropdown. The actions should now look like figure 3-4.

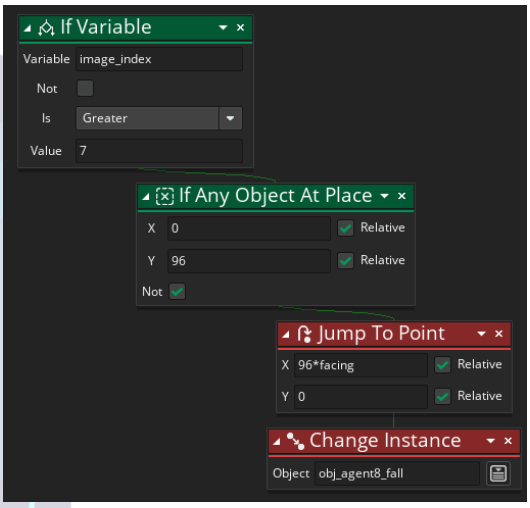


Figure 3-4. The actions for the Step, Step event of the hopping Agent8 object.

Programming the climbing Agent 8 object:

1. Reopen obj_agent8_climb and set its **Parent** to obj_agent8.
2. Add a **Create** event and include a **Set Instance Scale** action with **Horizontal** set to facing and **Vertical** set to 1. This flips the horizontal direction of the sprite like before.
3. Add an **Animation End** event and include a **Jump to Point** action with **X** set to 96*facing, **Y** set to -96 and both **Relative** options selected. This moves Agent 8 into the next diagonal grid square depending on which direction he was facing. Like for the hop, this "jump" in Agent 8's position is completely invisible to the player as the last frame of the climbing animation lines up exactly with the next diagonal grid square.
4. Include a **Change Instance** action and select obj_agent8_stand from the dropdown.

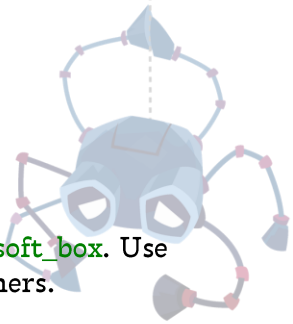
Programming the dead Agent 8 object:

1. Reopen obj_agent8_dead and set its **Parent** to obj_agent8.
2. Add a **Create** event and include a **Set Direction Variable** action with **Direction** set to random(45)+45.
3. Include a **Set Speed** action with **Type** set to **Direction** and **Speed** set to 12.
4. Include a **Set Gravity Direction** action with **Direction** set to 270.
5. Include a **Set Gravity Force** action with **Force** set to 1.
6. Add a **Step, Step** event and include a **Set Instance Rotation** action with **Angle** set to hspeed and the **Relative** option selected.
7. Add an **Other, Outside Room** event and include a **Restart Room** action (Rooms section).

Saving your work and running the game:

1. Choose **Save Project** from the **File** menu (or click the disk icon) and press F5 (or click the play icon) to run the game. After a brief pause, a game window should appear. Check that you can hop along the bottom of the screen, climb up the stairs and fall down the other side. If you experience any issues, then you can compare your version to SpyDare1.yyz in the Projects directory of the archive.

MASS PRODUCTION



Creating the box objects:

1. Create four new objects called obj_box, obj_metal_box, obj_wood_box, obj_soft_box. Use the spr_solid sprite for obj_box and assign sprites appropriately to the others.

2. Create another new object called obj_machine. It doesn't need a sprite and we'll come back to programming it later.

3. Add a **Create** event to obj_metal_box and include an **Assign Variable** action with **Name** set to strength and **Value** set to 3. Set its **Parent** to be obj_box by clicking on the icon with three circles, next to where it says **Parent**. A dropdown will appear where you can select another object to be this object's parent.

4. Add a **Create** event to obj_wood_box and include an **Assign Variable** action with **Name** set to strength and **Value** set to 2. Set its **Parent** to be obj_box in the same way.

5. Add a **Create** event to obj_soft_box and include an **Assign Variable** action with **Name** set to strength and **Value** set to 1. Set its **Parent** to be obj_box.

Now all three box types have obj_box as their parent, we can effectively program all three objects at the same time! This is because every event we add to obj_box will now be "inherited" by its "children" too. Let's try it...

6. Reopen obj_box and set its **Parent** to be obj_solid (boxes are also solid).

7. Add a **Step, Step** event and include an **Assign Variable** action with **Name** set to depth and **Value** set to y. This is a little trick for making sure that objects further up the screen appear on top of object further down the screen.

8. Include an **If Variable** action and set **Variable** to y, **Value** to 0 and select **Equal** as the comparison. Before each box falls it will "track" the player's position at the top of the screen, so that the player knows what's coming next.

9. Include an **Assign Variable** action which depends on the **If** action. Set its **Name** set to depth, **Value** to 99. This pushes the tracking block behind any other blocks which may be falling down at the same time.

10. Follow this with an **Assign Variable** action (so it also depends on the **If** action). Set its **Name** set to x, **Value** to ((obj_agent8.x-x)/2) and select **Relative**.

So what's happening here? Well all of the Agent 8 objects have obj_agent8 as their parent, so obj_agent8.x will provide his x position regardless of which object is currently being used! We then subtract the x position of the box from that value, which tells us the distance between Agent 8 and the box. The **Relative** option will just add this distance to the x position of the box, causing the box to exactly match Agent 8's position on the horizontal axis. However, dividing the distance by two as well means that the box tracks slightly behind Agent 8's current position (more like it is being followed) and also provides a more realistic type of motion which "eases in" to it's new position.

SPYDER

11. Add an **Alarm, Alarm0** event to obj_box and include an **If Variable** action with **Variable** set to y, **Value** to 0 and **Equal** as the comparison. This is the alarm that will get triggered on the box when it is ready to fall.

12. Include an **Assign Variable** action which depends on the **If** action. Set its **Name** set to x, **Value** to obj_agent8.x. This ensures that the box always falls aligned with Agent 8.

13. Follow this with a **Set Direction Fixed** action (so it also depends on the **If**) and select the down arrow.

14. Follow this with a **Set Speed** action. Leave the **Type** set to Direction and set the **Speed** between 6 and 12, depending on how difficult you want to make the game.

15. Add a **Collision** event with obj_agent8_stand and include an **If Variable** action with **Variable** set to speed, **Value** to 0 and **Greater** as the comparison. This will only count collisions with Agent 8 when the box is moving. Note that we have used the standing Agent 8 state, rather than the parent object: this is kinder to the player as they won't be squashed if they are in the middle of another action.

16. Include a **Change Instance** action which depends on this action, and select obj_agent8_dead from the dropdown. Click on the down arrow on the action name and select other from **Applies To** so that it changes Agent 8 and not the box.

Okay – great work. We're nearly there so take a deep breath...

17. Add a **Collision** event with obj_solid. This will handle collisions between boxes and any other solid object (which includes both boxes and the floor).

18. Include an **If Variable** action with **Variable** set to y, **Not** selected, **Value** set to 0 and **Equal** set as the comparison. This rules out any collisions which are with boxes which are tracking the player at the top of the screen (where their y position is fixed to 0).

19. Include a second **If Variable** action which depends on the first. Set **Variable** to other.y, **Value** to y and **Greater** as the comparison. This rules out any collisions with boxes which are above the current box on the screen.

20. Include a third **If Variable** action which depends on the second. Set **Variable** to other.strength, **Value** to strength and **Less** as the comparison. This checks to see if the block we're colliding with is made of weaker material than ours (metal crushes wood, wood crushes leather).

21. Include a **Destroy Instance** action which depends on this third **If** action. Click on the little down arrow on the action name and select other from **Applies To** so that it destroys the other (weaker) block.

22. Include an **Else** action which depends on the last **If Variable** action (the one which tested strength). Include a **Jump to Point** action with **X** set to xprevious and **Y** set to yprevious.

23. Include a **Snap Position** action, also dependant on the **Else**. Set both **Horizontal** and **Vertical** to 48. This helps to make sure boxes always rest in a "valid" position, aligned with the grid.

24. Include a **Set Speed** action (also dependant on the **Else**). Leave the **Type** set to Direction and set the **Speed** to 0.

25. Finally include a **Set Alarm Countdown** action (also dependant on the **Else**). Set **Alarm** to 0 and **Countdown** to 30 (or lower to make your game harder). Click on the down arrow on the action name and select obj_machine from **Applies To** so that it sends a message to the machine object to create the next box. The actions should now look like figure 3-5.

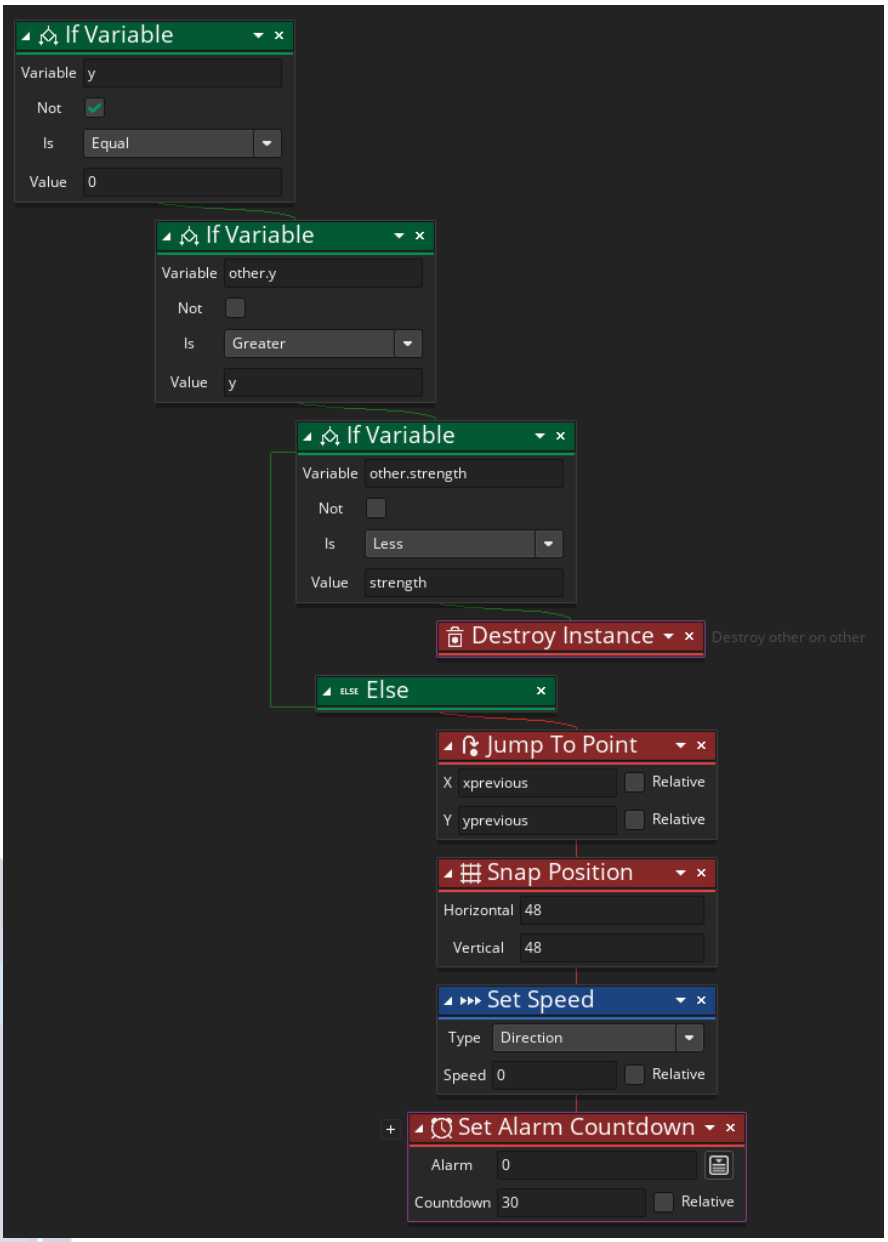
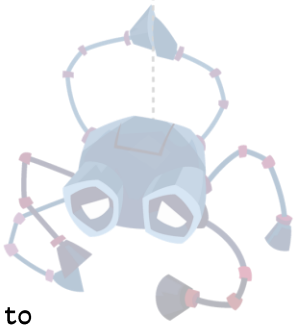


Figure 3-5. The collision event between the box parent object and other solid objects

MACHINE CODE



Programming the machine object:

1. Reopen obj_machine and add a **Create** event.

2. Include a **Create Instance** action with **Object** set to obj_metal_box, **X** set to obj_agent8.x, and **Y** set to 0. The first box in the game is always a metal one.

3. Include a **Set Alarm Countdown** action. Set **Alarm** to 0 and **Countdown** to 120. This is only the time before the first block falls, so the value doesn't make much difference.

4. Add an **Alarm, Alarm0** event and include a **Set Alarm Countdown** action. Set **Alarm** to 0 and **Countdown** to 1. Click on the down arrow on the action name and select obj_box from **Applies To** so that it sends a message to (all of) the boxes to start moving. Fortunately, we've already put a check in that object to make sure they only start moving if they are currently at the top of the screen!

5. Include a **Choose** action (Random section) and add three options by clicking on the plus (+) icon. Set the first **Option** to obj_metal_box, second **Option** to obj_wood_box, and third **Option** to obj_soft_box. Set **Target** to random_box and select the **Temp** option. This will select one of our three box types at random and put the result in a temporary variable called random_box (temporary as it only exists inside this event).

6. Include a **Create Instance** action with **Object** set to random_box (type it in rather than selecting from the dropdown menu). Set **X** to obj_agent8.x, and **Y** to 0.

There's just a couple of things you need to do now to get your game into a working order. Reopen the room, and remove the staircase that you created on the level back at the start. Only the floor is required now as the machine will create the boxes for you. Before it can do that you will need to add an instance of obj_machine into the room too: it doesn't matter where, so long as there is one there to create the boxes in the first place.

Saving your work and running the game:

1. Save your project and run the game to test out your work so far. Check that the boxes track the player, and fall correctly crushing weaker blocks underneath them as they fall. If you experience any issues, then you can compare your version to SpyDare2.yyz in the Projects directory of the archive.

SPYDER

FINAL TOUCHES

Finishing up:

1. Reopen obj_solid and deselect the **Visible** option. This will make the red floor invisible, but it will still work in the same way.

2. Create objects for obj_pole, obj_stop_left and obj_stop right and give them their sprites Set obj_pole to be the **Parent** of both obj_stop_left and obj_stop_right.

3. Open obj_pole and add a **Step, Step** event and include a **Set Instance Rotation** action with **Angle** set to speed and the **Relative** option selected.

4. Reopen obj_agent8 and add an **Other, Outside Room** event. This will trigger when the player "escapes" the room (in any state!) . Include a **Set Alarm Countdown** action with **Alarm** set to 0 and **Countdown** set to 120.

5. Include an **Apply To** action. Click on the down arrow on the action name and select obj_pole from **Applies To** (which of course includes all three pole objects). This action allows us to apply a whole group of actions to be applied to a different object.

6. Include a **Set Direction Variable** action so it depends on the **Appy To** action (works in the same way as an **If**). Set **Direction** set to random(45)+45.

7. Follow this with a **Set Speed** action (within the same group of **Apply To** actions) with **Type** set to **Direction** and **Speed** set to 12.

8. Follow this with a **Set Gravity Direction** action with **Direction** set to 270.

9. Follow this with a **Set Gravity Force** action with **Force** set to 0.5.

10. Add an **Alarm, Alarm0** event and include a **Restart Room** action.

11. Reopen the room and use the pole objects to create the boundaries of the level, so that the player has to reach a certain height in order to get out (see figure 3-6). Of course, you could easily set up a whole series of increasingly difficult rooms and use a **Go To Next Room** action instead of restarting the same room above.

12. Finally, take a look at the sound effects provided in the **Resources** folder and have a go at integrated them into the game for yourself. The fact that you've make it to this final tutorial shows that you have what it takes to start experimenting – so go and explore and have fun in the process!

SPYDER

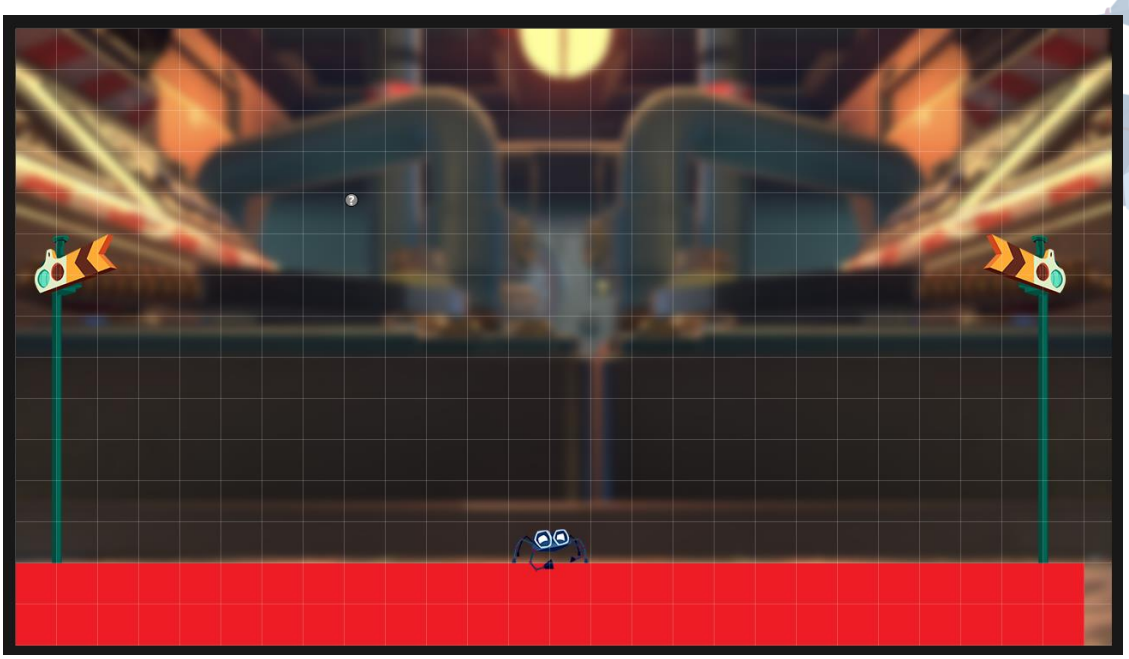


Figure 3-6. The finished layout of the room in the Room Editor.

Figure 3-7. The finished game.

CONGRATULATIONS

Fantastic Work!

You'll find a finished version of this tutorial in the project file Projects/SpyDare3.yyz in the archive. As before, you can now return to the video to celebrate what you've achieved and check out a few features which you might want to add to the game. Parenting is a really powerful feature in Game Maker and can save you a lot of time and effort if used well.

ACKNOWLEDGEMENTS

We would like to thank the whole Spyder™ team for allowing us to use assets from their amazing game in this tutorial. This tutorial would be approximately 15 years less exciting without piggy-backing on all their hard work. If you'd like to see what Spyder's real game developers did with their game then head on over to the Apple Arcade and check it out!

www.spyderthegame.com