

Invasive Species Game in Scratch	Grade 6 Understanding Life Systems
Coding Guide	

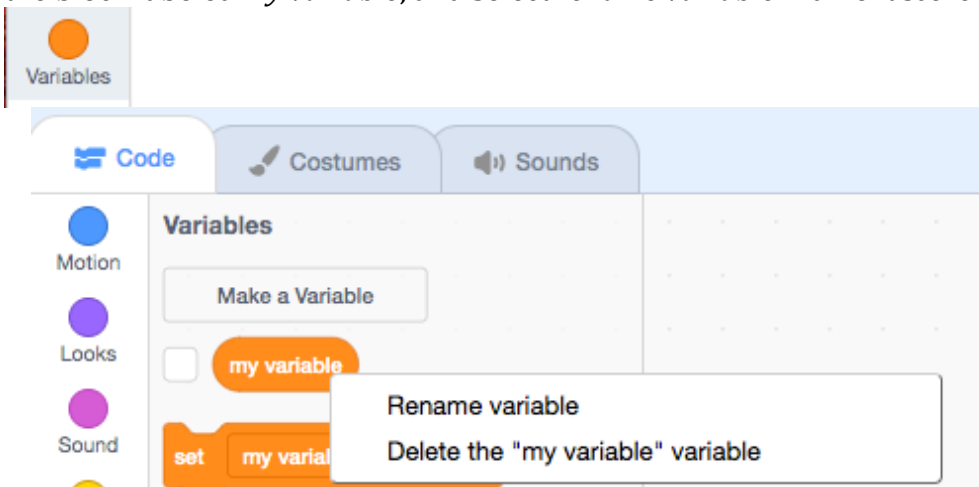
Before detailing how to program our invasive species game, let us first lay out the behaviours we expect of it. We want to illustrate the effect the invasive Emerald Ash Borer has on established ecosystems. To do this, we will allow the player to control a small, fuzzy woodland creature. The player’s goal will be to ‘eat’ (by touching) randomly appearing ash trees. This will cause the tree to disappear and will increase our score by one. In competition with the player will be one (and as the score increases, more) ash borers. We will program the game such that if the ash borers touch a tree or the player, the game is over.

The way Scratch is structured is that scripts are attached to sprites—individual actors within the game. We will thus require a player sprite, a tree sprite, and a number of ash borer sprites. (Fortunately, we only need to program one.) Before we begin to program these sprites, though, we must first (as all programmers do) define our variables.

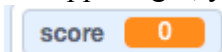
We need only one true variable for this script: the score. Every scratch script starts with one sprite (a happy cat) and one variable (called my variable). Let us rename them.

**Rename Variable:**

Click the orange circle on the left-hand side of the scratch window that says ‘variables’. Right click the block labeled *my variable*, and select *rename variable*. Name it *score*.



Click the check box next to score to set the score as visible to the player. Now in the play window in the upper right, you should see the score represented as 0. This will remain visible to the player.



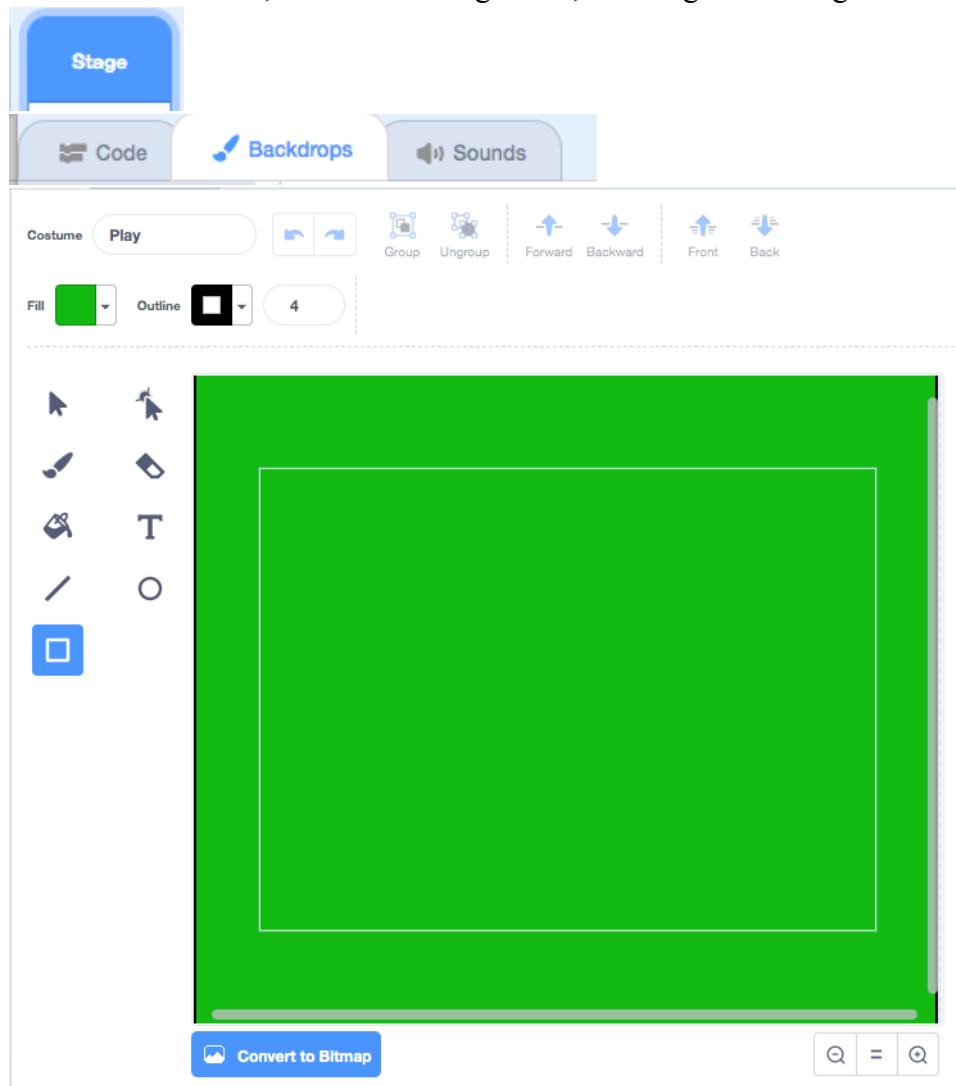
With variables defined, it is time to set the stage for our game.

## Backdrops

The ‘stage’ upon which our game plays out is called a ‘backdrop’. You can think of the backdrops as a sort of visual variable: which backdrop is in use is certainly treated by the code as a variable, so we would like to define them first.

We wish to have 2 backdrops: one for playing, and one which will serve as our game-over screen. You can choose a backdrop, upload from your computer, or use the ‘paint’ tool to create a simple one. Do this by clicking on the stage tab in the lower-left, and then selecting the backdrops tab in the upper right. (or use the button in the absolute lower-left to pick a backdrop).

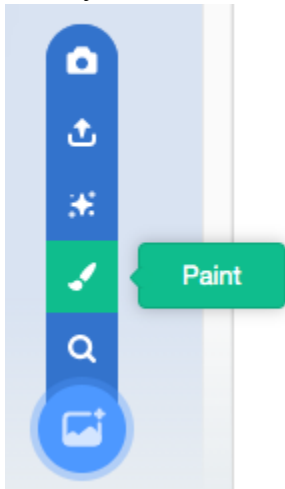
Using the rectangle tool, for this example we’ll make a simple green backdrop. Simply select green under the fill colour, click the rectangle tool, and drag the rectangle across the whole checkered area:



Notice we have also renamed the costume from ‘backdrop1’ to Play.

We will now create a second backdrop (if one has not been created automatically). Hover your mouse over the circular picture icon in the lower LEFT. Chose the paintbrush and a new backdrop will be created in paint mode.

We will use paint mode to create a game over screen. Again, fill the area with a green box. Now use the Text tool, to write an appropriate explanation. For example: Game Over! The Emerald Ash Borer destroys the habitats of other forest animals, making it difficult for them to adapt and survive.

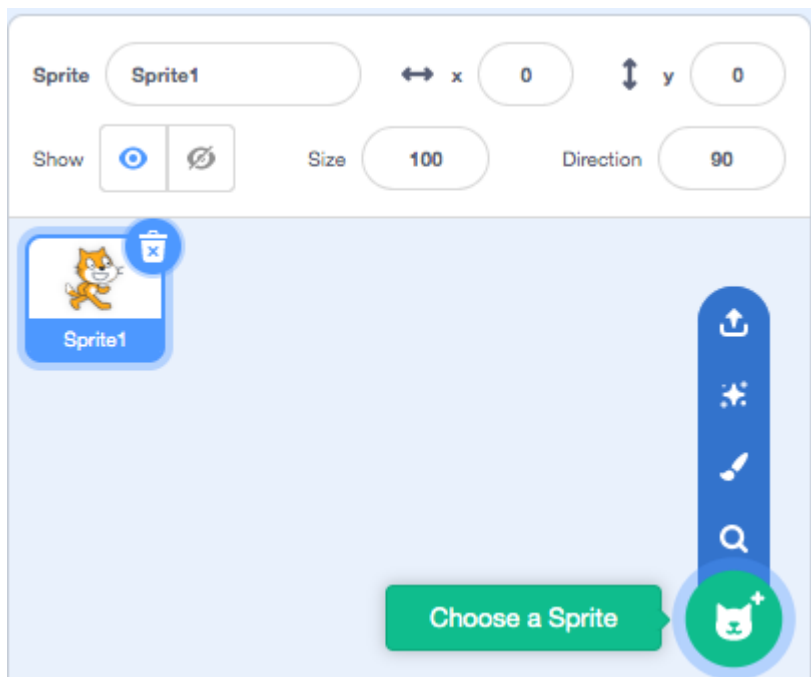


T

### Sprites

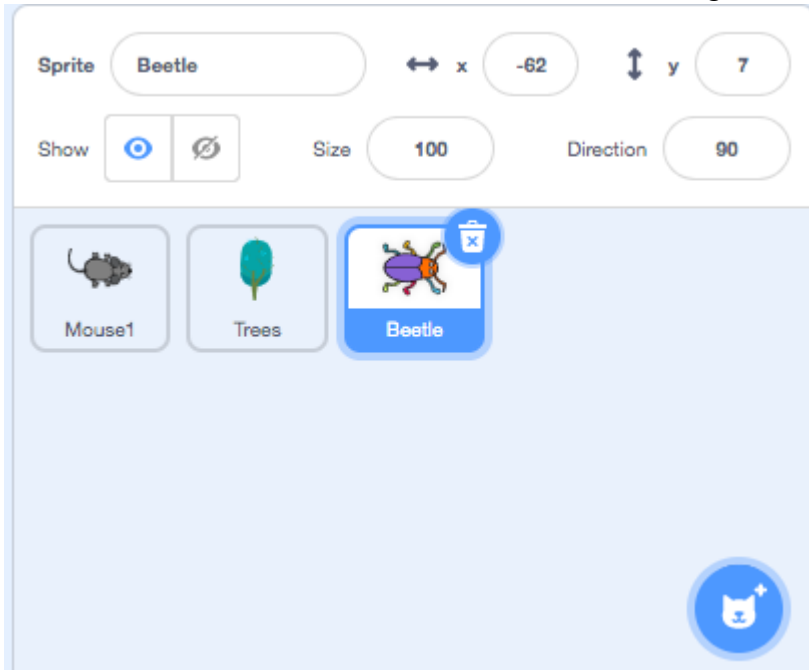
Just like backdrops, it is good practice to create sprites in advance. We will create 3: the player character, the tree, and the beetle.


We already have one sprite created, so it is just a matter of re-costuming and renaming the sprite. (In Scratch, the image associated with a sprite is called a 'costume'. This can be confusing for some, since in other programming environments what Scratch calls a costume is referred to as a sprite.)



Select the sprite, and then the ‘Choose a Sprite’ button. Pick ‘animals’ and then an animal you would expect to see in a forest habitat. For this example, we will use a mouse. (If you accidentally create a new sprite, just delete sprite 1.)

To create a new sprite, do the same thing without having already selected an existing sprite. Remember we need a tree and a beetle. You should have something like this in the lower-right sprite pane:

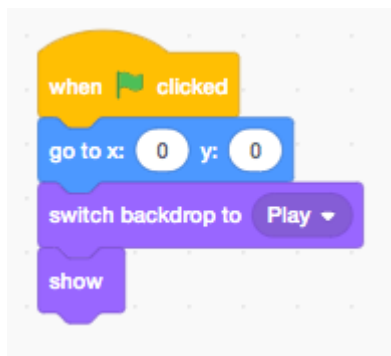


Since we do not want our sprites to all appear at the center of the screen, set them to ‘hide’ by clicking on the  icon for each sprite. The sprites are now invisible until the code asks for them.

We are ready to begin programming.

### Programming: The Player Sprite

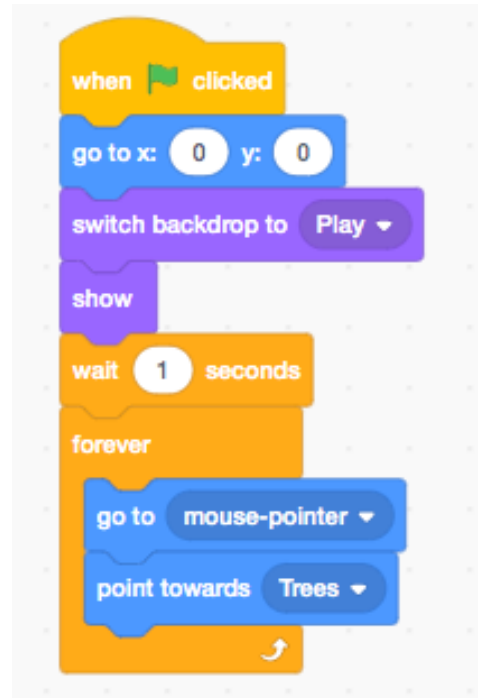
The player character must do 2 things: follow the mouse pointer (or touch), and point the mouse costume at the tree, which is the mouse’s target.



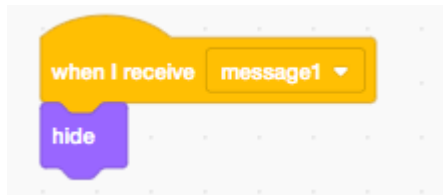
We wish to initialize the mouse when the green flag is clicked (A the mouse when the green flag is clicked (A yellow event block), have it go(a blue motion block), and make sure the backdrop is set to our Play backdrop. (A purple looks block), and then show itself (another purple looks backdrop.) Remember that blocks must click together, as seen left. Placing blocks near one another doesn’t cut it.

Now we need to put the mouse into a loop where it always (at least until game over) follows the mouse. We will use the *forever* orange control block. Inside this loop, we will tell the mouse to *go to* (blue

motion block) the mouse-pointer. (Click the drag down menu on the block and pick mouse-pointer); so that the mouse isn't just sliding around awkwardly, we will have him point his nose towards the trees, using the blue motion block *point towards*. We will also put an orange *wait 1 seconds* block before the loop, so the game does not start immediately. That tends to be jarring for players. That is nearly all the scripting we need on the mouse. We will allow scoring to be handled by the tree, and the game over conditions to be checked by the beetle. We do, however, want to have the mouse disappear when the game-over condition is set by the beetle.

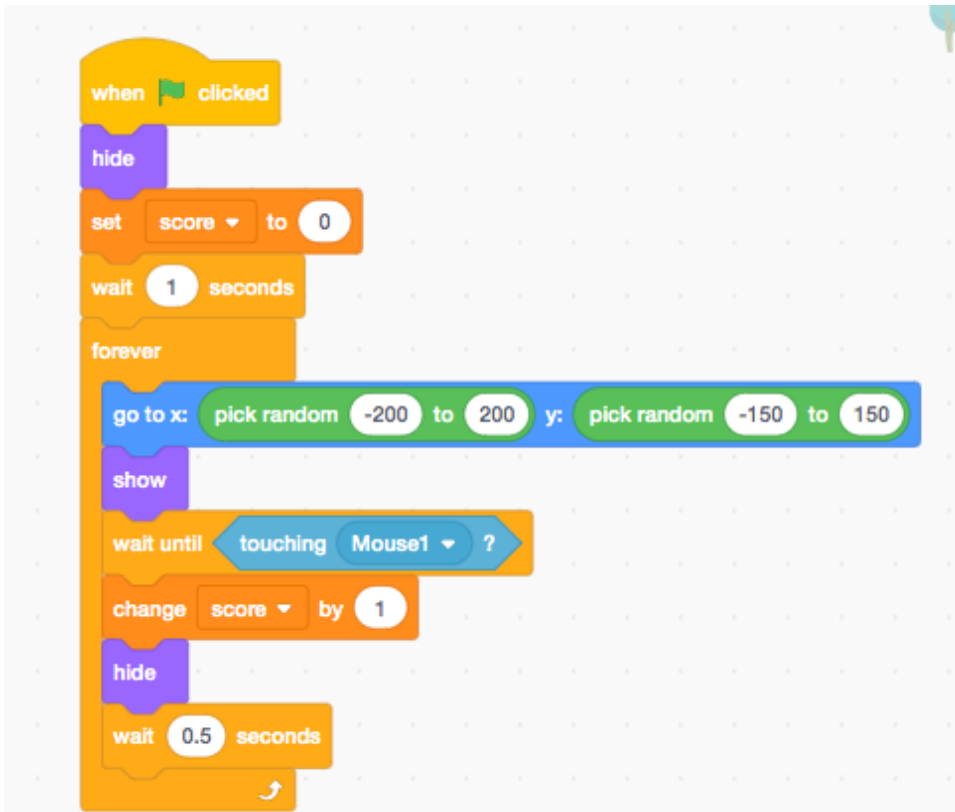


For this we will create a second script, starting with the yellow event block *when I receive message1*. Message1 will be the game over signal broadcast from the other sprite. (This message could also be renamed 'end' or 'game over' if desired. That is, in fact, better practice. To do so, create a new message from the dropdown menu in the *when I receive* block.) Whatever we call the game over signal, once we receive it, we merely want the mouse to hide, using the purple Looks block *hide*.

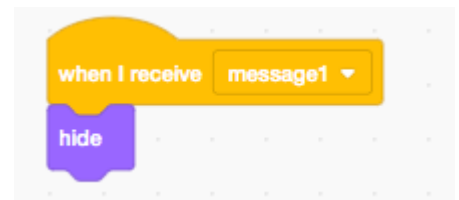
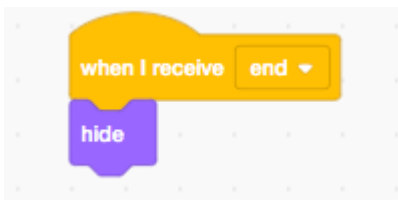


### Programming: The Tree Sprite

The tree must appear in a random location, detect if it has come in contact with the mouse, and if so, disappear and reappear in another random location. This must repeat, over, and over and over again. Begin with the yellow event block *when green flag clicked*. As with the mouse, we will wait one second, then start a 'forever' loop for the behaviour. We must also specify that the score be zero when we start the program. Use the orange variable block *set* somewhere before the *wait* block. Inside the loop, we use the blue Motion block *go to*, but rather than specifying an exact location, we drag into the block the green operator *pick random*. We wish to have the tree appear on the playing field, so the random value for X should be restricted to -200 to 200. For Y, it needs to be -150 to 150. (The playing field is wider than it is tall.) Having moved to its location, the tree can show itself, using the purple *show*. We then wish to *wait until* (using the orange control block) the tree is *touching* mouse1 (using the light blue sense block). Then we *change score* by 1, and *hide* the tree, before waiting 0.5 seconds to restart the loop. It is important to wait to before restarting the loop or the program may overcount the score.



As with the mouse, we wish to have the tree disappear on the end condition, so we use the same blocks for a second script. Depending how you set that up, it will look like one of these:

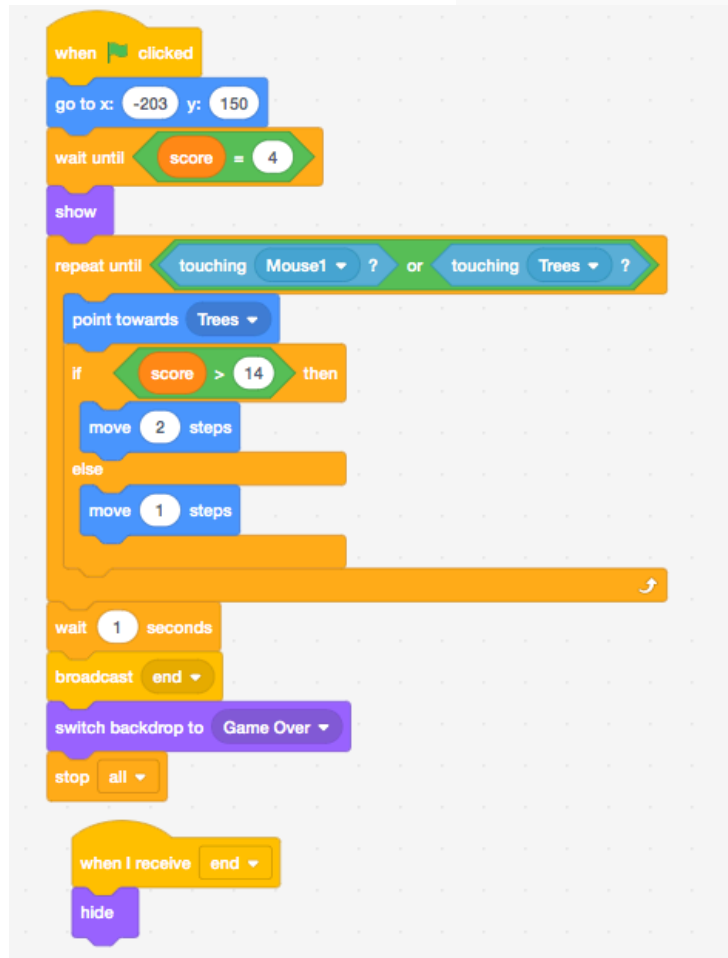
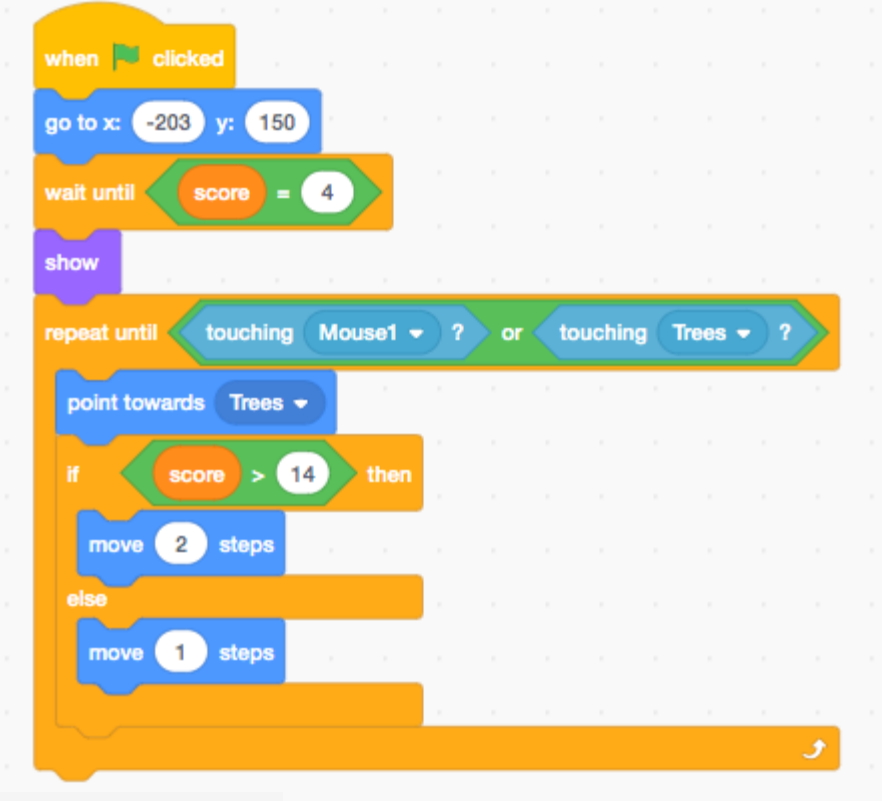


### Programming: The Ash Borer



To initialize the ash borer, we will use a similar script as before—the same event block—but we will use the *go to* block to place the beetle off screen so the tree does not accidentally appear on it. We will use a *wait until* block with the operator block = to wait for the score to reach 4 before revealing the beetle. This allows the player to start the game without interference from the ‘bad guy’.

Once shown, the beetle's behaviour should be in a loop. Rather than a forever loop, we only want to loop until the beetle touches a tree or the player. Remember, those are our game over conditions. So we will use the *repeat until* control block in conjunction with the green *or* operator block and two copies of the *touching* sensing block: one set to the tree sprite (here Trees) and one to the player sprite (here Mouse1). The repeating behaviour that will happen until game over will go inside this loop; after the loop we will have the blocks that program the end of the game.

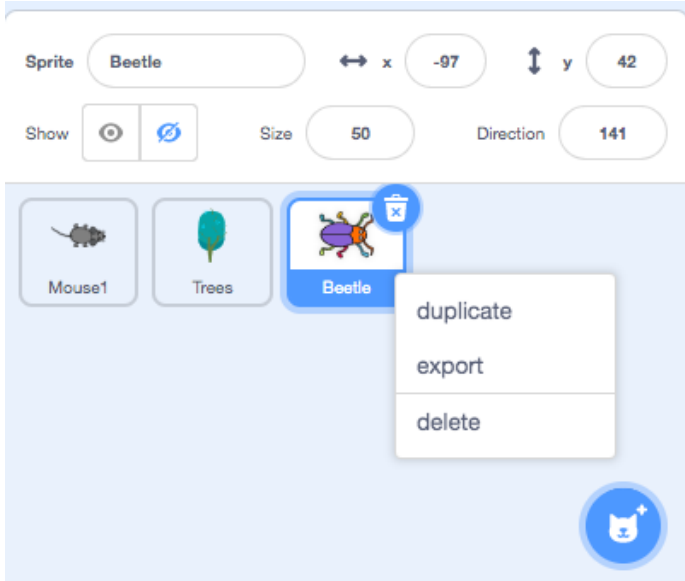


Inside the loop, we will use the blue motion block *point towards* to point the beetle towards the tree, and the *move\_steps* blocks to guide it forwards. To make the game rise in difficulty as time goes on and the player's score increases, we'll use an *if/else* control block to set the borer to move faster if the score is greater than 14. (The conditional score can be changed if desired, and more steps can be added. Alternatively, a mathematical formula linking score to speed could be adopted instead.) Finally, we wish to code what happens once the beetle is touching mouse1 or trees. We certainly wish to *broadcast* our game over message (here we'll call it end), using the yellow event block. This, remember, will cause all other objects to hide. We can then switch the backdrop (using the purple *switch backdrop* block) to our game over screen. We'll also want to stop all other scripts so we cannot accidentally get points by hitting invisible trees with our invisible mouse. A

*wait* block has been included just so that the user has a chance to see the beetle that caused game over. Otherwise it might be an unsatisfying surprise for everything to disappear too quickly. Notice we also include the second script *when I receive end, hide* on this beetle. This allows us to leave off the *hide* block in the main script, and also to include additional enemies in the next section.

### Adding Extra Beetles

Right click on the sprite Beetle, and click duplicate.



All the code associated with that sprite has been recreated in the new Beetle2 and can be altered without effecting Beetle1. Now, to make the game more balanced, it makes sense to change the conditions on which Beetle2 appears and speeds up. Say Beetle2 shows up when the score hits 6, and changes speed at score 16. An arbitrary number of beetles can be created in this way. You may wish to decrease the size of all sprites to 50 (shrinking them by half) so that all of those beetles can fit on the screen!

