Scratch - Whack-a-Mole

You will be writing a whack-a-mole game. If you are not familiar with this game, there game is played over a set time. During that time there are a number of molehills, moles will randomly pop up and your job is to whack them with a hammer. Each wack earns you a point.

Does it sound cruel? Yep well that's video games for you.

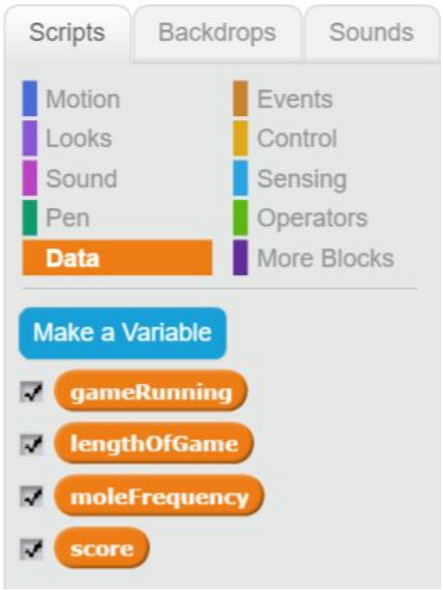
My completed version can be found here
https://scratch.mit.edu/projects/138091877/

Build the Stage

We will be attaching code to the stage to kick the game off in response to the player using the <spacebar>. We will use a variable called **gameRunning** to determine if the game is already underway. We will use **broadcasts** to initiate the game code in the sprites.

Create Variables

- Under **Data** create the following variables
    - **gameRunning** - indicates if the game is already running
    - **lengthOfGame** - the number of seconds a single game will last
    - **moleFrequency** - used to control how often the moles appear
    - **score** - used to count the moles hit within a single game

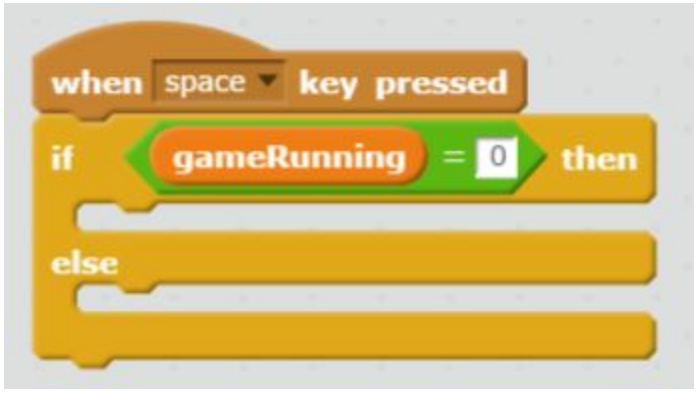


Create Event Handlers

- From **Events** drag a **When <Flag> clicked** block into the code area
- From **Data** drag a **set <> to <>** block into this new event handler, set the following values
    - gameRunning to 0
    - lengthOfGame to 30
    - moleFrequency to 10

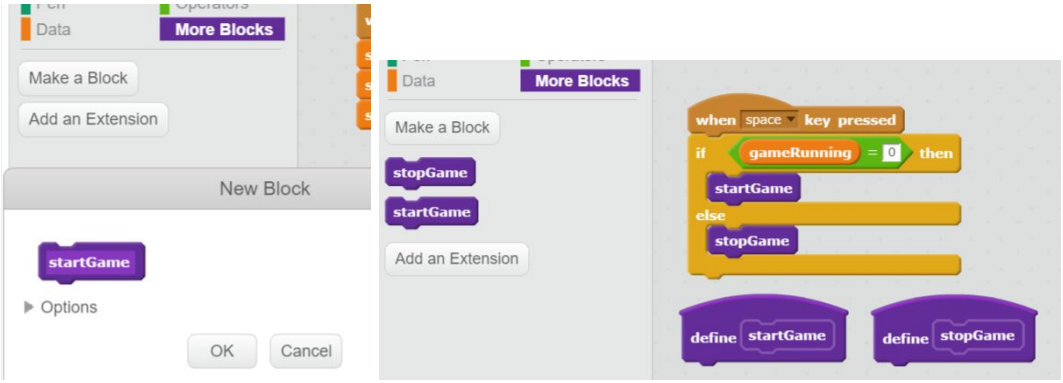
- From **Events** drag a **when <spacebar> key pressed** block into the code area.
- From **Control** drag an **if <> else <>** block into the code area.
- From **Operators** drag an **[] = [] (equals)** block into the **if** block
- The two values in the **if** block are zero and the value of **gameRunning** which can be dragged in from **Data**.

In this block we are determining if the game is already running when the player hits the spacebar.



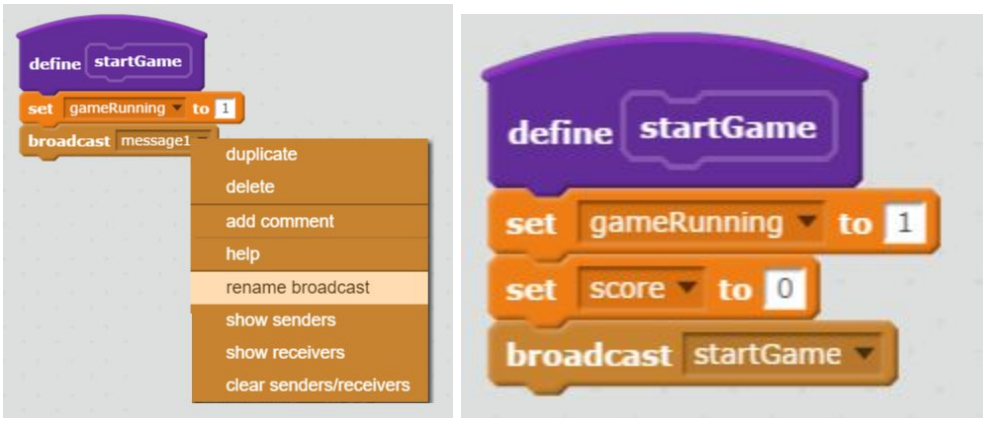
- Under **More Blocks** use **Make a Block** to create the following two functions
    - **startGame**
    - **stopGame**

- In our original **if** block, call **startGame** if **gameRunning = 0** and **stopGame** in the **else** part.



Build Start Game Block

- From **Data** drag a **set <gameRunning> to 1** block into the **startGame** function
- From **Data** drag a **set <score> to 0** block in
- From **Events** drag a **broadcast <newMessage>** into the **startGame** function.
- Rename this broadcast **startGame**.



Build End Game Block

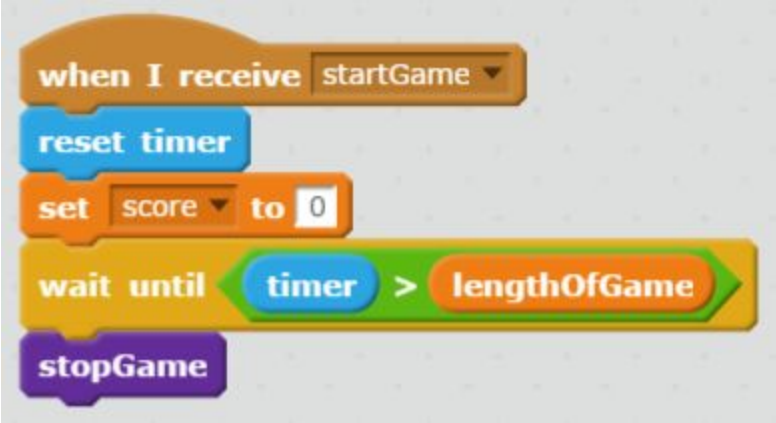
This is very similar to the **startGame** block.

- From **Data** drag a **set <gameRunning> to 0** block into **endGame**.
- From **Events** drag a **broadcast <newMessage>** into the **endGame** function.
- When you click on the **newMessage** a dialog should open and give you the chance to name the message. Call it **endGame**.

Handle the Game Timer

The last bit of code to go into the stage will handle the timing of the game. The stage itself will respond to the **startGame** broadcast, use it to reset the timer, score and then wait to stop the game after the time expires.

- From **Events** drag a **When I receive <startGame>** block into the code area.
- From **Sensing** drag a **reset timer** block into this event handler.
- From **Data** drag a **set <score> to 0** block in
- From **Control** drag a **wait until <>** block in
- From **Operators** drag a **more than ([] > [])** block into the wait handler
- From **Sensing** drag the value of **timer** into the left hand space
- From **Data** drag **lengthOfGame** into the right hand side.
- From **More Blocks** drag a call to **stopGame** into the handler



This event handler will detect the start of the game, then wait for the timer to reach to configured length, then stop the game.

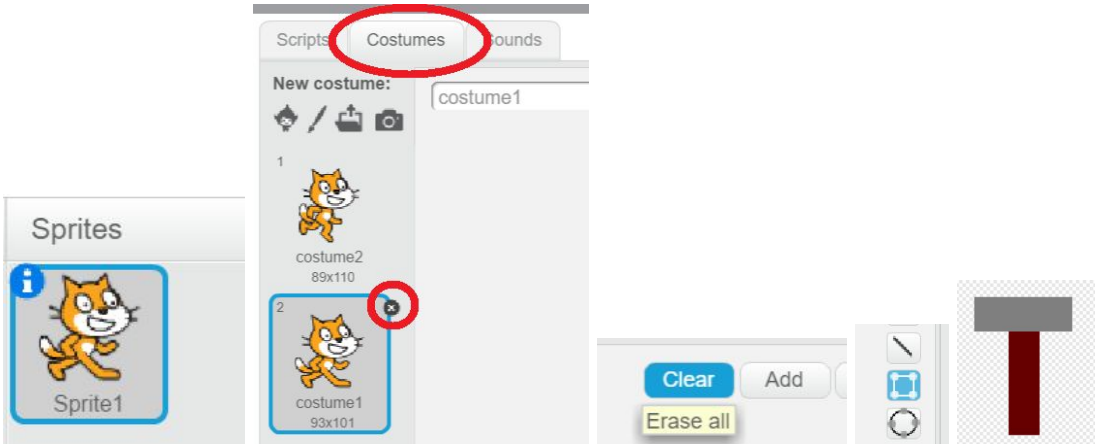
We are now ready to move onto the player.

Build the Player

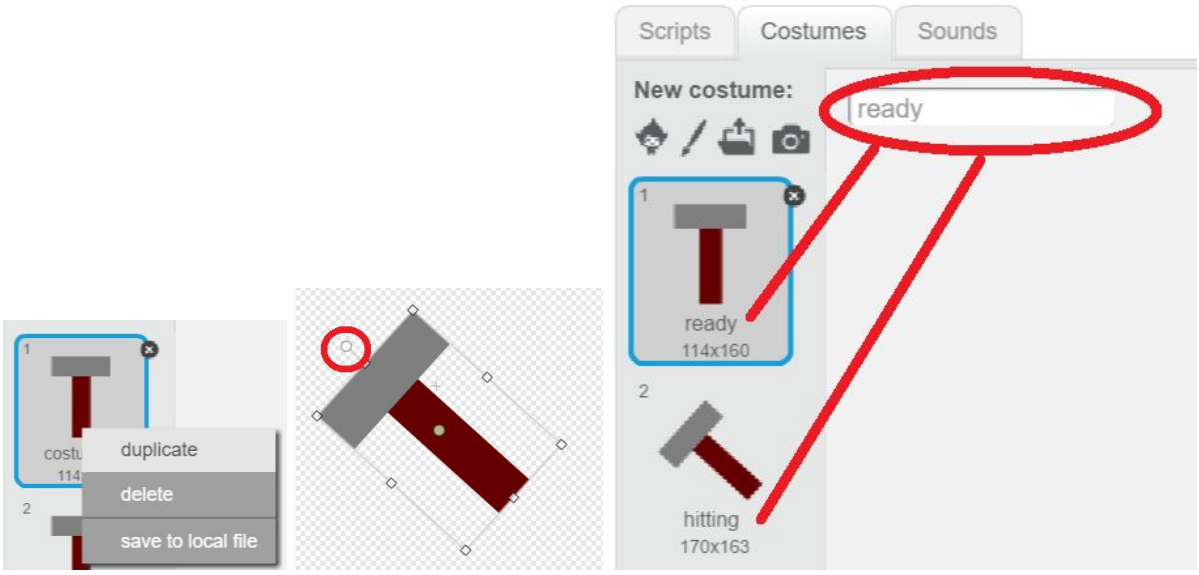
The player will act via a hammer sprite. There will be 4 possible moles to hit, each one tied to a number key on the keyboard. Pressing that number will cause the hammer to travel to that molehill and try to whack it. First we need to draw a hammer.

Draw the Hammer

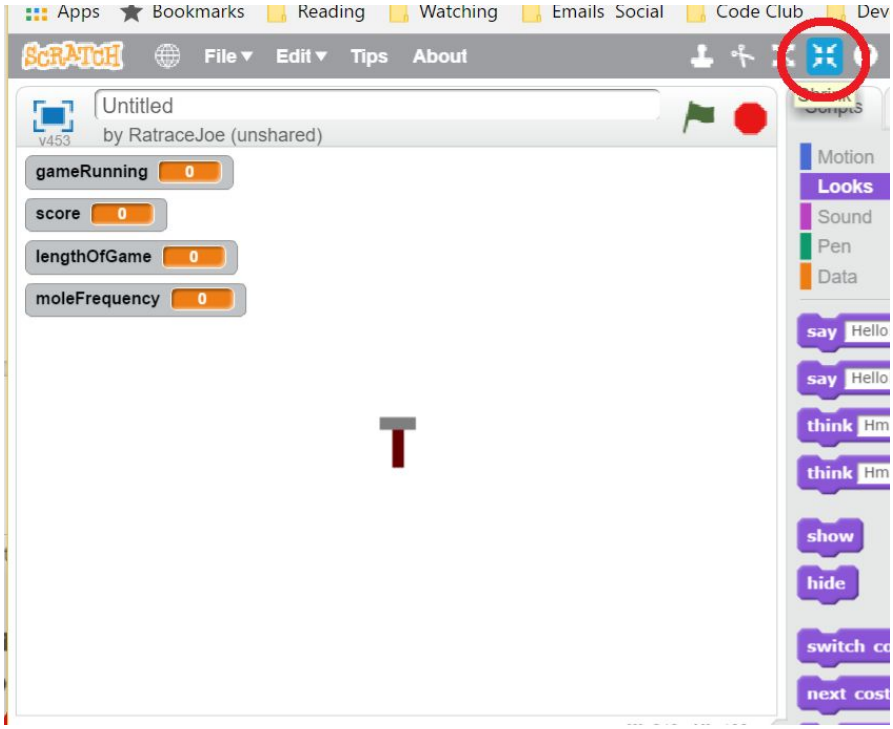
- Click on the default sprite (the good old cat)
- Click on the **Costumes** tab
- Delete the second costume using the little **x**.
- While editing the first costume, click on **Clear** to get rid of the cat.
- Draw two filled rectangles as shown as the hammer (keep it simple)



- Right click on the first sprite and click on **Duplicate**.
- Select the second costume
- Drag select all of the elements in the second costume.
- Use the rotate handle to rotate the hammer, as if it is being used to hit something
- Use the naming textbox to name the first costume **ready** and the second costume **hitting** (as shown)



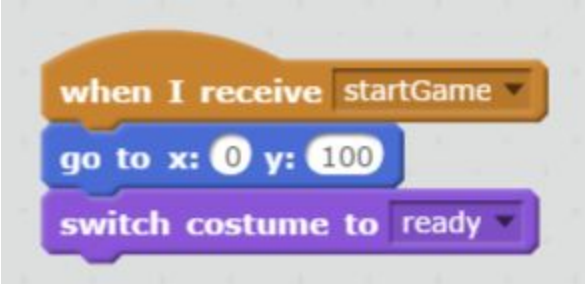
Once you have created the sprite; use the shrink tool to make the hammer fairly small within the game area.



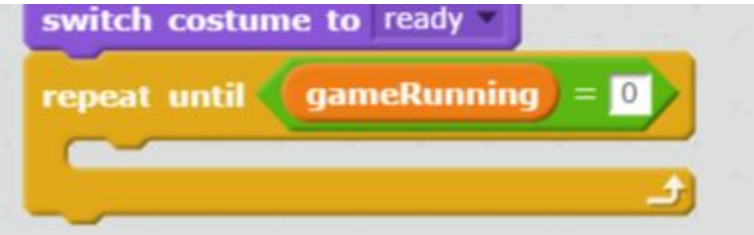
Handle the Start of the Game

We must now write the main loop of the game for the hammer sprite.

- From **Events** drag **when I receive <startGame>** into the code area
- From **Motion** drag a **go to x:0 y:100** block into the broadcast handler
- From **Looks** drag a **switch costume to <ready>** into the handler



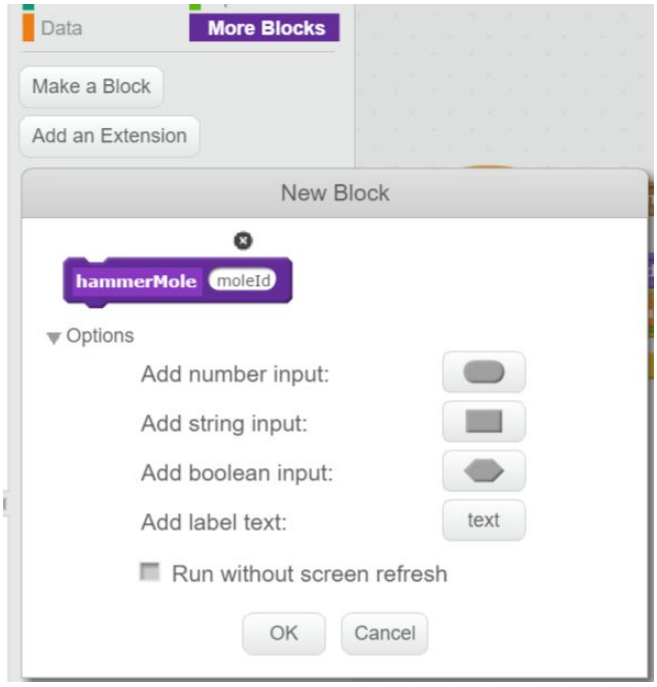
- From **Control** drag a **repeat until <>** block into our handler, this will form the main game loop for the hammer sprite
- From **Operators** and **Data** repeat the loop until **gameRunning = 0**.



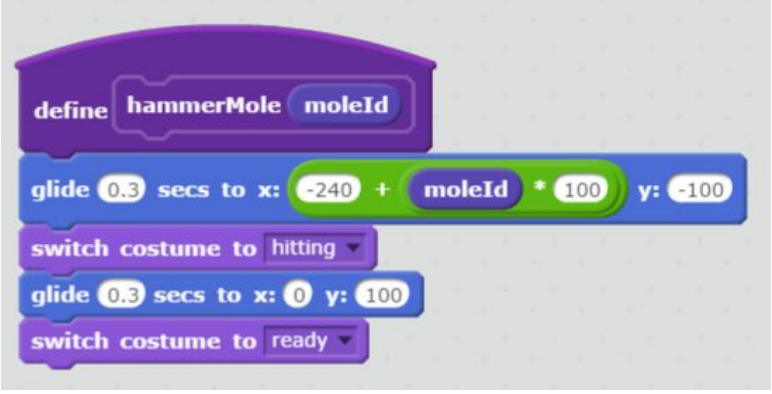
Create the Hammer Mole Block

Before we can write the controls, we must write the function to call to hammer a particular mole. The hammer is going to rest in a central position high up in the game area (as you coded with the **go to x:0, y:100** block) it will then move to a moles location and hammer it based on numerical key input.

- Under **More Blocks** click on **Make a Block**
- Give the block the name **hammerMole**
- Expand the **Options** and add a number input called **moleId**.



- From **Motion** drag a **glide <> secs to x:<> y:<>** block into the **hammerMole** function
- Set the glide block up to do the following
    - Glide for 0.3 seconds
    - Go to x = -240 + (moleId * 100) - hint: use **Operators**
    - Go to y = -100 - the moles will be located here
- After the glide, from **Looks** drag **switch costume to <hitting>** - I am assuming you named the costumes correctly
- After the costume switch, use another glide block to move back to the start location
    - 0.3 seconds
    - Go to x = 0
    - Go to y = 100
- Switch costume back to **ready**.



Handle Player Controls

Now we can write the numerical controls. This will allow us to test the **hammerMole** function.

- From **Control** drag an **if <>** block into the loop
- From **Sensing** drag a **key <1> pressed** block into the **if** block
- From **More Blocks** drag a call to **hammerMole** and type the number **1** into the argument.

Test this out by starting the game, first click the green flag to initialise the values, then use space bar until **gameRunning** is **1**.
Then try pressing the **1** number key. The hammer should move off to the left, hammer in place then glide back.

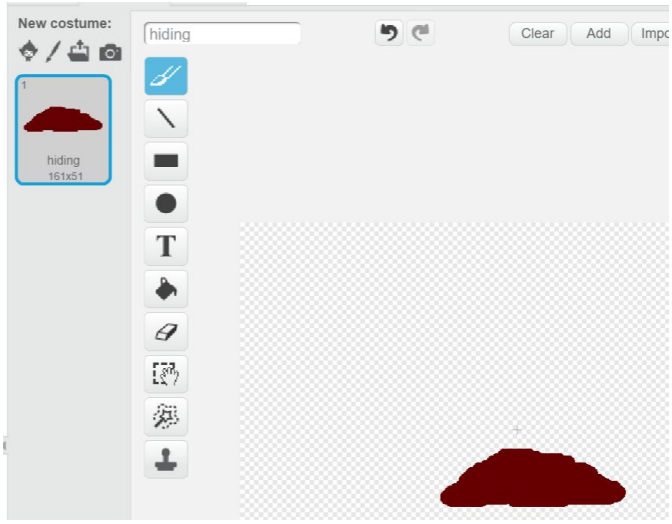
TRY YOURSELF
We will be using 4 moles, so write the key handlers for moles 2, 3 and 4.

Create the Moles

We will now create a new sprite to represent the moles. There will be a single sprite, and cloning will be used to give us multiple moles in the game.

Draw the Mole Appearing

- Under sprites click on the paintbrush to paint a new sprite
- The first costume will be the empty molehill. Just use a thick brown pen to draw some ‘dirt’
- Name this first costume **hiding**

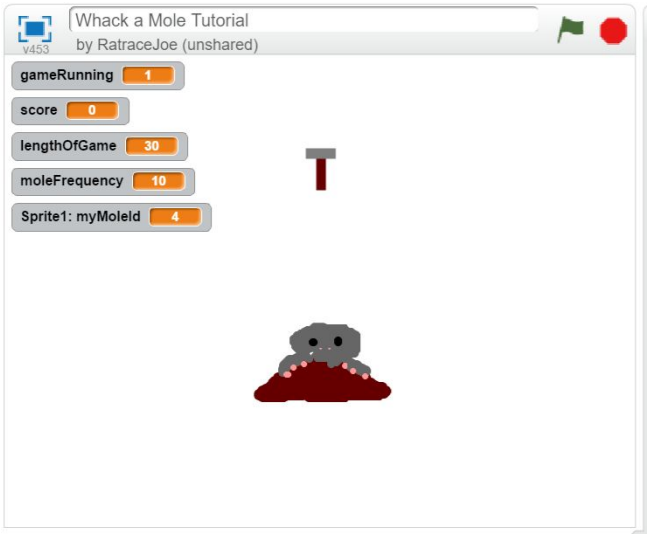


- Duplicate the costume and name the duplicate **appearing1**
- Use a thick grey pen and a thinner pink pen to draw some ‘paws’
- Duplicate the costume again and draw a bit of a nose, Scratch will probably smartly call this new duplicate **appearing2**.
- Duplicate this again and then draw a head for the mole. Rename this costume **ready**.

Hiding Appearing1 Appearing2 Ready

That should do it for drawing, don’t spend too much time here, just get something vaguely sensible.

You may wish to resize the mole using the grow/shrink tools. This screenshot shows sensible relative sizes.



Create the Clones

It is a good idea to give clones an ‘id’ number which allows them to identify themselves in the code.

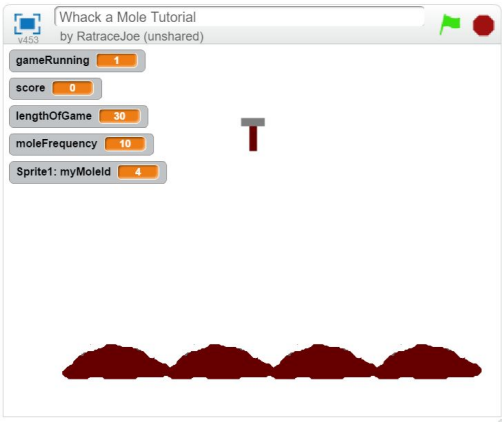
- Ensure the mole sprite is selected
- Under **Data** click on **Make a Variable**, give it the name **myMoleId** and ensure you tick **For This Sprite Only** (very important you tick this box, it gives each clone its own value of **myMoleId**).
- From **Events** drag a **when I receive <startGame>** broadcast handler into the code area.
- The default instance of the sprite should be hidden, so from **Looks** drag a **hide** block into the broadcast handler.
- From **Data** drag a **set <myMoleId> to 0** block in
- From **Control** drag a **repeat <4>** block into the code area.
- From **Data** drag a **change <myMoleId> by 1** into the **repeat** loop
- From **Control** drag a **create clone of myself** into the **repeat** loop.

Now we have to handle the creation of each clone. We hand off to a new event handler.

- From **Control** drag a **when I start as a clone** event into the code area.

- From **Looks** drag a **switch costume to <hiding>** block into the new event handler.
- Each mole needs to go to a different position. It will use the **myMoleId** to calculate this. From **Motion** drag a **go to x:<> y:<>** block into the event handler
    - x = -240 + (100 * myMoleId)
    - y = -100
- The sprite is now ready to show, from **Looks** drag a **show** block in.

It should now be possible to test this code. Press <spacebar> (you may need to press it twice to stop/start a game). You should get 4 moles across the lower part of the game area.



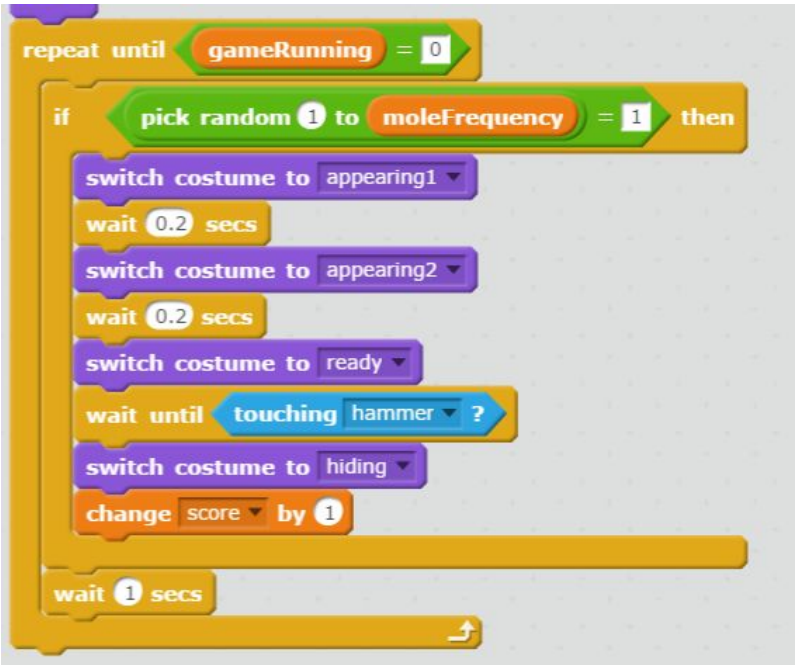
Handle Clone Behaviour

Now we need to create a game loop for this sprite. This code will all be added to our **when I start as a clone** block.

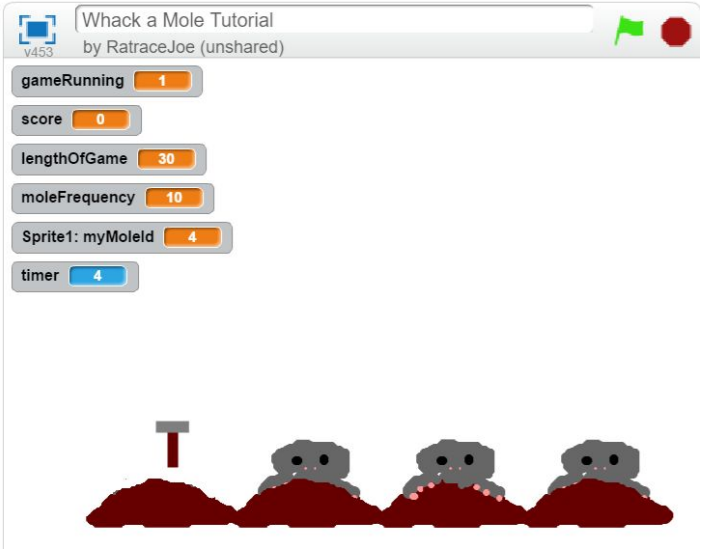
- From Control drag a **repeat until <gameRunning> = 0**. This is the same loop condition we used for the hammer sprite, so you can duplicate that one of you want.
- I want to use the **moleFrequency** variable we used earlier to randomly cause moles to appear. From **Control** drag an **if <>** block into the repeat loop.
- Use **Operators** and the value of **moleFrequency** from **Data** to construct a condition that evaluates to true if a **Operators -> pick random 1 to <moleFrequency> = 1**.
- From **Control** then drag a **wait <1> secs** block AFTER the **if <>** block.

The code we have written here will go inside the **if <>** block randomly. The frequency it does this is controlled by the **moleFrequency** variable. Inside the **if** block we are going to write the code to cause the mole to appear, wait to be whacked and then go back to **hiding**.

- From **Looks** drag a **switch costume to <appearing1>** block into the **if <>** block
- From **Control** drag a **wait <0.2> secs** block in
- From **Looks** drag **switch costume to <appearing2>** block in
- From **Control** drag a **wait <0.2> secs** block in
- From **Looks** drag a **switch costume to <ready>** block in
- From **Control** drag **wait until <>** block in
- The mole must wait until it is touching the hammer (use the block from **Sensing**). This will only happen if the player hits the right number key.
- Once the mole has been hit, it will change back to hiding, from **Looks** drag a **switch costume to <hiding>** block in
- From **Data** drag a **change <score> by <1>** block in

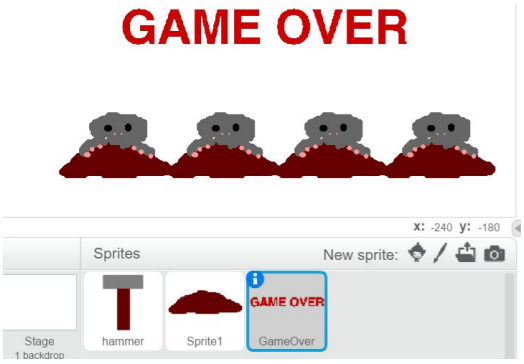


This should be sufficient to play the game. You may wish to make the timer visible using the tickbox next to **timer** in **Sensing**.



Handle Game Over Screen

Last thing we need is to display a game over message. This makes it clear when the timer has run out. For this, create a new sprite which just has the text **GAME OVER**.



The code for this sprite will be very simple. We can use the broadcasts **startGame** and **endGame** to **show** and **hide** this sprite. It would also be nice to **say** the score.

I hid a few of the variables, played a game and it should look something like this.