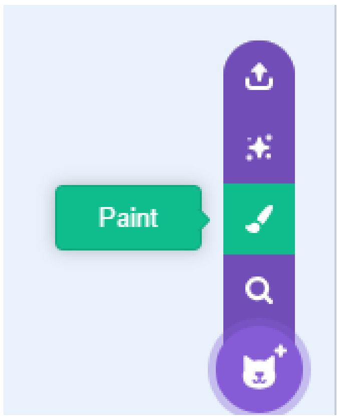
GLACIER RACE

Glacier race is a two-player game where a red car vs a blue car race up the screen, swerving around obstacles and collecting gems as you go. There's no finish line in this race – the winner is simply the person with the most gems when the time runs out.

1

Start a new project and delete the cat sprite. Use the paintbrush symbol to create a new blank sprite: -



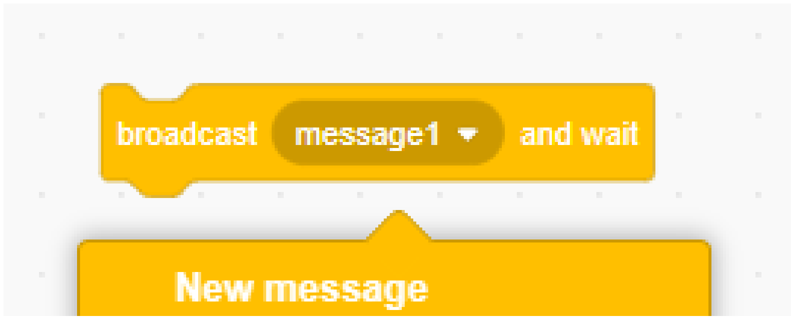
and rename it "GameLoop".

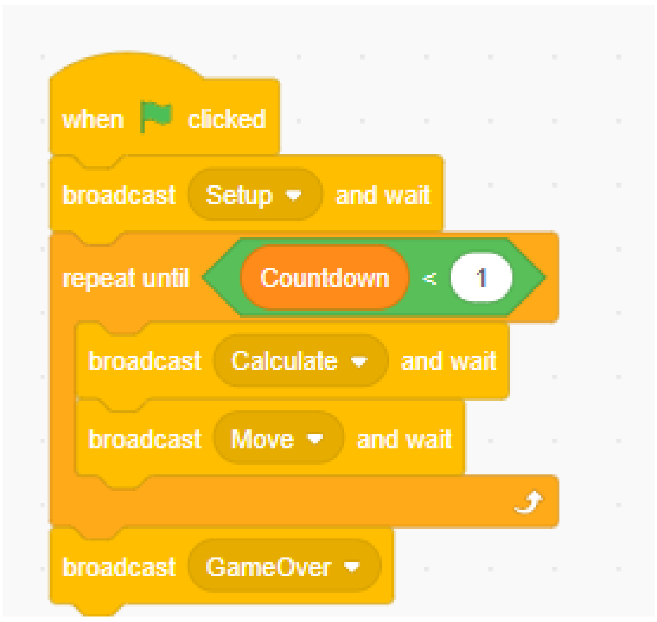
Then make a variable called "Countdown" for the game timer and show it on stage (this happens be default – the ticked box beside the variable name is what controls this).

Build the following code to make the controlling game loop for this game. You'll need to create new broadcast messages "Setup", "Calculate", "Move" and "GameOver".

A broadcast message sents a signal to your Scratch project that other sprites and other code sections can react to in response.

You can create each message by hitting the drop down arrow and selecting "new message": -





```
when 🏳 clicked
broadcast Setup and wait
repeat until Countdown < 1
    broadcast Calculate and wait
    broadcast Move and wait
broadcast GameOver
```

The idea here is the “Setup” message will signal to all sprites to get ready for the game start. It waits for them to finish, then the main loop begins. The loop sends out message tell every sprite when to run each part of its code. The loop ends only when the countdown value reaches 0, at which point the “GameOver” message is sent so all sprites can perform any final actions and the winner is announced.

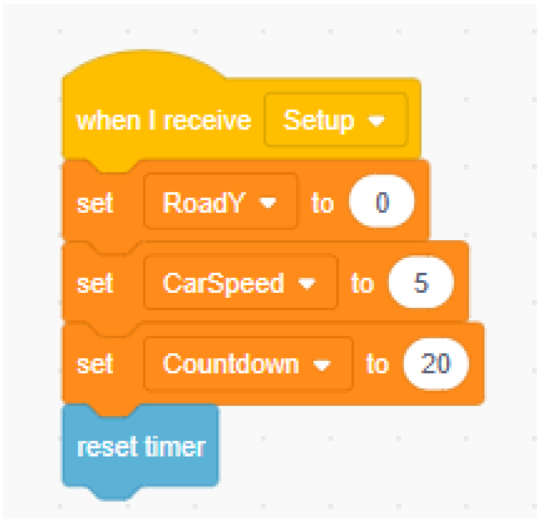
2

Create two new variables: -

“RoadY” to store the Y coordinate used to position our moving scenery and

“CarSpeed” to set how quickly the cars can move. Untick both variables so they don’t display on screen.

Add the following code in the GameLoop sprite as below to set starting values for these variables at the start of each game: -



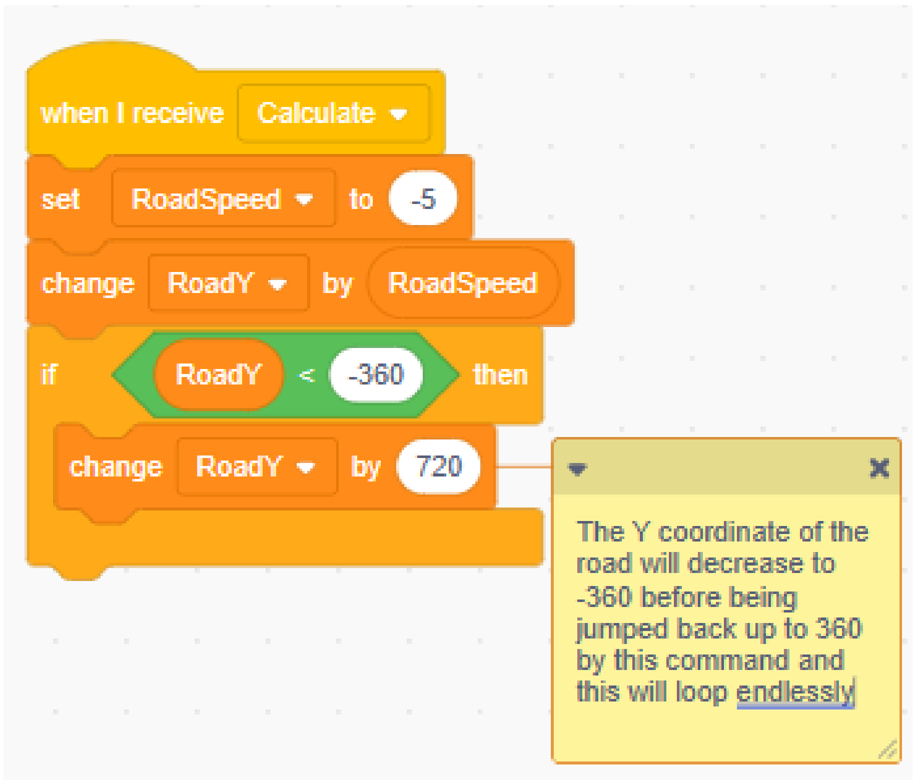
```
when I receive Setup
set RoadY to 0
set CarSpeed to 5
set Countdown to 20
reset timer
```

The "reset timer" block can be found in "Sensing" and it resets the internal timing counter built into Scratch.

3

Add another variable, for all sprites, called "RoadSpeed" – this will store the speed of the moving scenery. Again uncheck the tickbox so the variable is not displayed on screen.

Then add the code below to calculate the position of the road each time the game loop runs. It will be clearer how this works once you've made the road sprites.



```
when I receive [Calculate]
set [RoadSpeed] to (-5)
change [RoadY] by (RoadSpeed)
if <(RoadY) < (-360)> then
    change [RoadY] by (720)
```

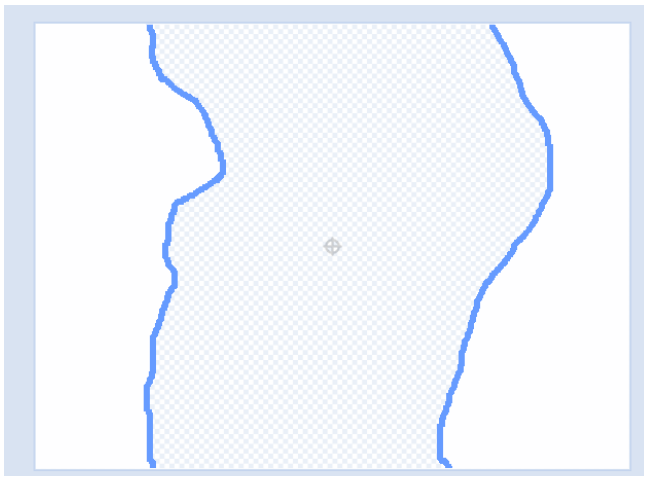
The Y coordinate of the road will decrease to -360 before being jumped back up to 360 by this command and this will loop endlessly

[note you do not have to add the yellow comment boxes, this is only for explanation to you]

The game will make seem as if the car sprites are moving quickly along the road, but in reality it is the road that is moving. The road will be made up of two sprites that fit seamlessly together – Road1 and Road2. These roads will take turns scrolling gown the stage, creating the illusion that the cars are constantly moving.

4

Create a new sprite and call it "Road1". In the paint editor, click "**convert to bitmap**" option if shown then chose the paintbrush tool, set the thickness to 10 and the colour to blue. Use this to draw the edges of the road similar to the example below. When they are drawn, being careful not to leave a gap at top or bottom, use the fill tool to fill left and right with white or a colour of your choice, to create a snowy setting – it's important to leave the middle road part transparent.

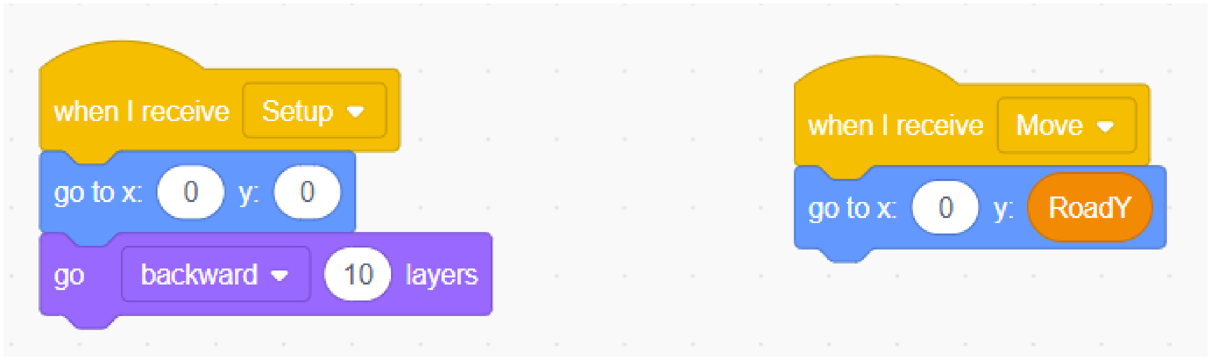


5

Now duplicate the "Road1" sprite and name the duplicate "Road2". Go to the costumes tab of Road2 and click the "Select" tool then click on the "flip vertical" option. And the Road costume will flip upside down. The edges of Road1 and Road2 will now match as they are mirror images. They'll look odd on screen for the moment, but we will fix this later in the coding.

6

Add the following code blocks to Road1 to get the road moving. If you test run after adding these blocks, half the road will scroll down the screen: -

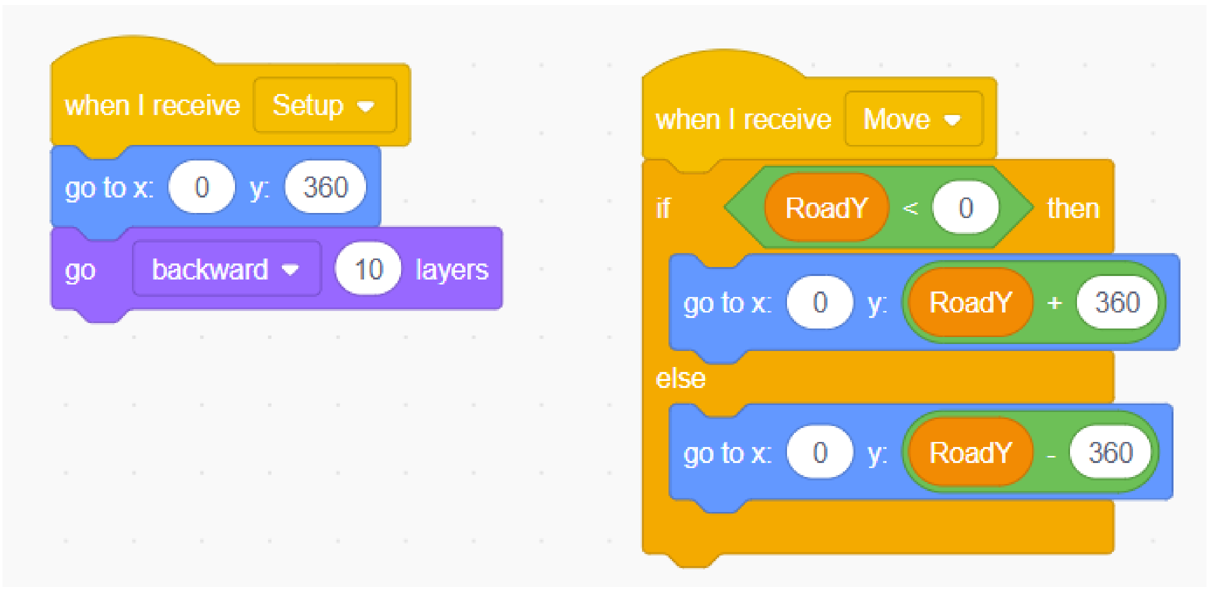


```
when I receive [Setup ▾]
go to x: (0) y: (0)
go [backward ▾] (10) layers

when I receive [Move ▾]
go to x: (0) y: (RoadY)
```

7: - Add similar code to also place and then move road2,

But note it very cleverly always keeps it 360 pixels away from Road1, which means they are always one full vertical screen with apart.



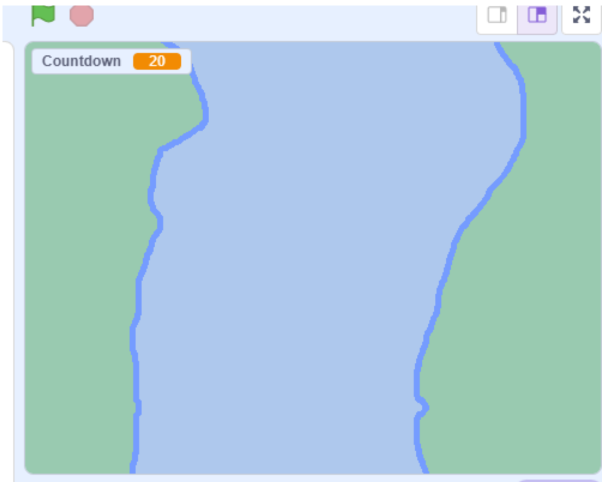
```
when I receive [Setup ▾]
go to x: (0) y: (360)
go [backward ▾] (10) layers

when I receive [Move ▾]
if <(RoadY) < (0)> then
    go to x: (0) y: ((RoadY) + (360))
else
    go to x: (0) y: ((RoadY) - (360))
```

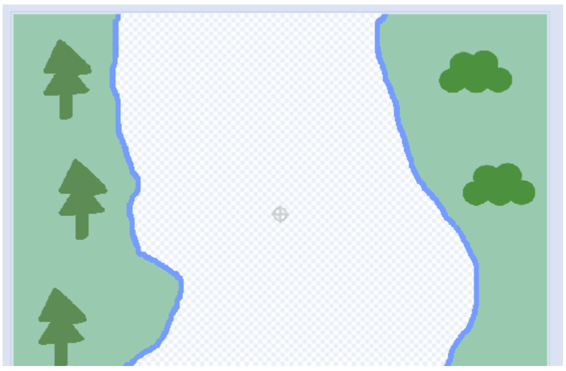
Try it and you should now have an infinitely scrolling road effect.

8

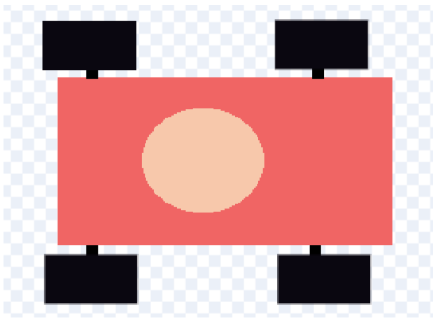
To paint the road, colour the backdrop rather than the sprite costume itself, or the cars will detect collision with the road sprites constantly. Use a light blue to suggest an icy glacier road: -



9 Add some Scenery to the side of the road, like bushes or trees: -

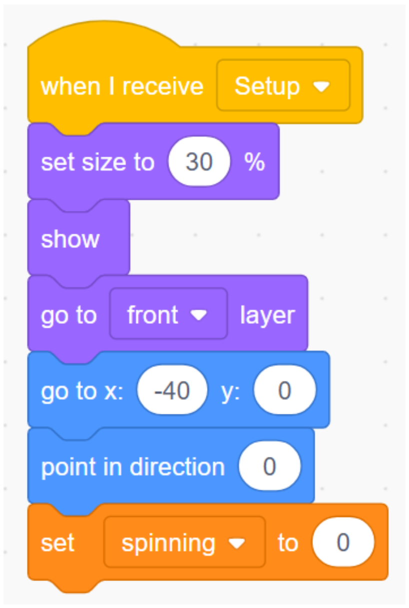


10 Time to make a red race car sprite – start with the cat sprite from the library and use this to size the car roughly the same size. It will seem to big on stage at first but we will shrink it in the code blocks to follow. Use the paint editor and click "convert to bitmap" – delete the cat, then use the paint the tools to build something like this, centered and it must be pointing to the right as below: -



11 Rename this sprite "RedCar". Next create a new variable "Spinning" and make sure to select the option "for this sprite only" in the variable creation pop-up box. This means this is a variable specific to Red Car and other sprites cannot change it.

12 Add the following code to initialise the Red Car sprite at the start of the game: -

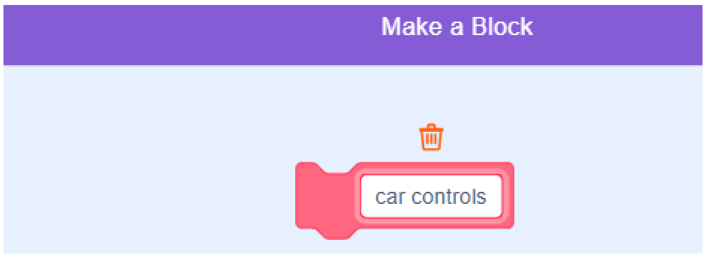


```
when I receive [Setup ▾]
set size to (30) %
show
go to [front ▾] layer
go to x: (-40) y: (0)
point in direction (0)
set [spinning ▾] to (0)
```

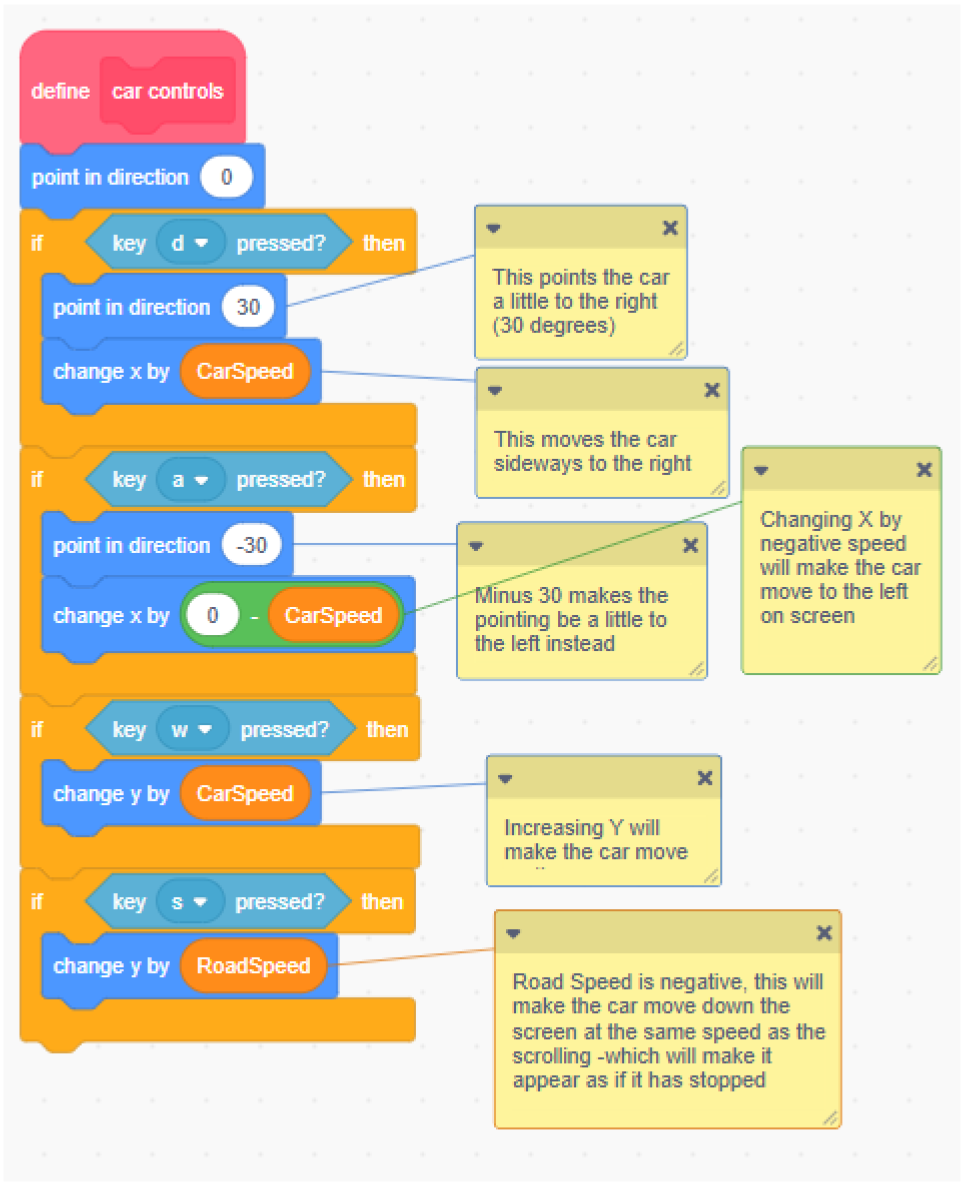
13 Let's define the controls to move the car:

1st create a custom block to hold the code for car controls neatly – Chose My Blocks, then make a block and call the new block "car controls".

You now will have a custom define block you can use to build you custom block – pull this into your code editor and add the blocks blow to build the car control logic.



Add the following code to the define block: -



```
define car controls
point in direction 0
if key d pressed? then
    point in direction 30
    change x by CarSpeed
if key a pressed? then
    point in direction -30
    change x by 0 - CarSpeed
if key w pressed? then
    change y by CarSpeed
if key s pressed? then
    change y by RoadSpeed
```

This points the car a little to the right (30 degrees)

This moves the car sideways to the right

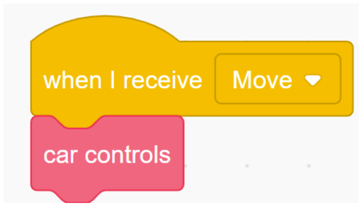
Minus 30 makes the pointing be a little to the left instead

Changing X by negative speed will make the car move to the left on screen

Increasing Y will make the car move

Road Speed is negative, this will make the car move down the screen at the same speed as the scrolling -which will make it appear as if it has stopped

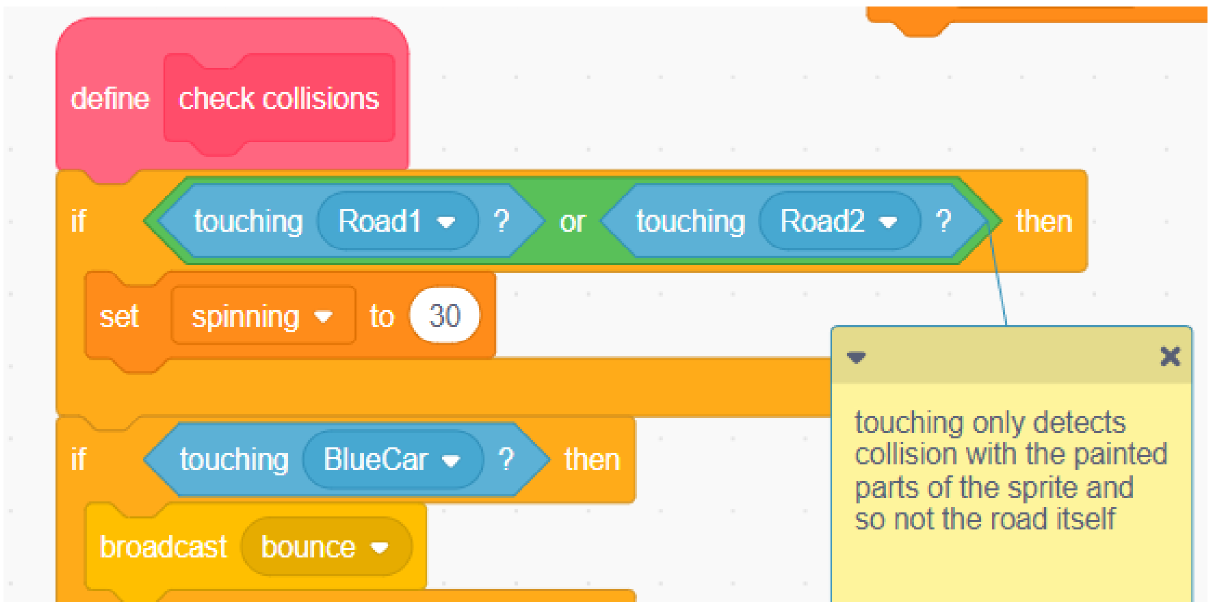
14 This is the definition part only, for the code to be run we need to add our new custom Scratch block “car controls” into the game loop code – add it like so: -



```
when I receive Move
car controls
```

Next let’s make the game more challenging, by forcing the players to avoid hitting the snow off-road.

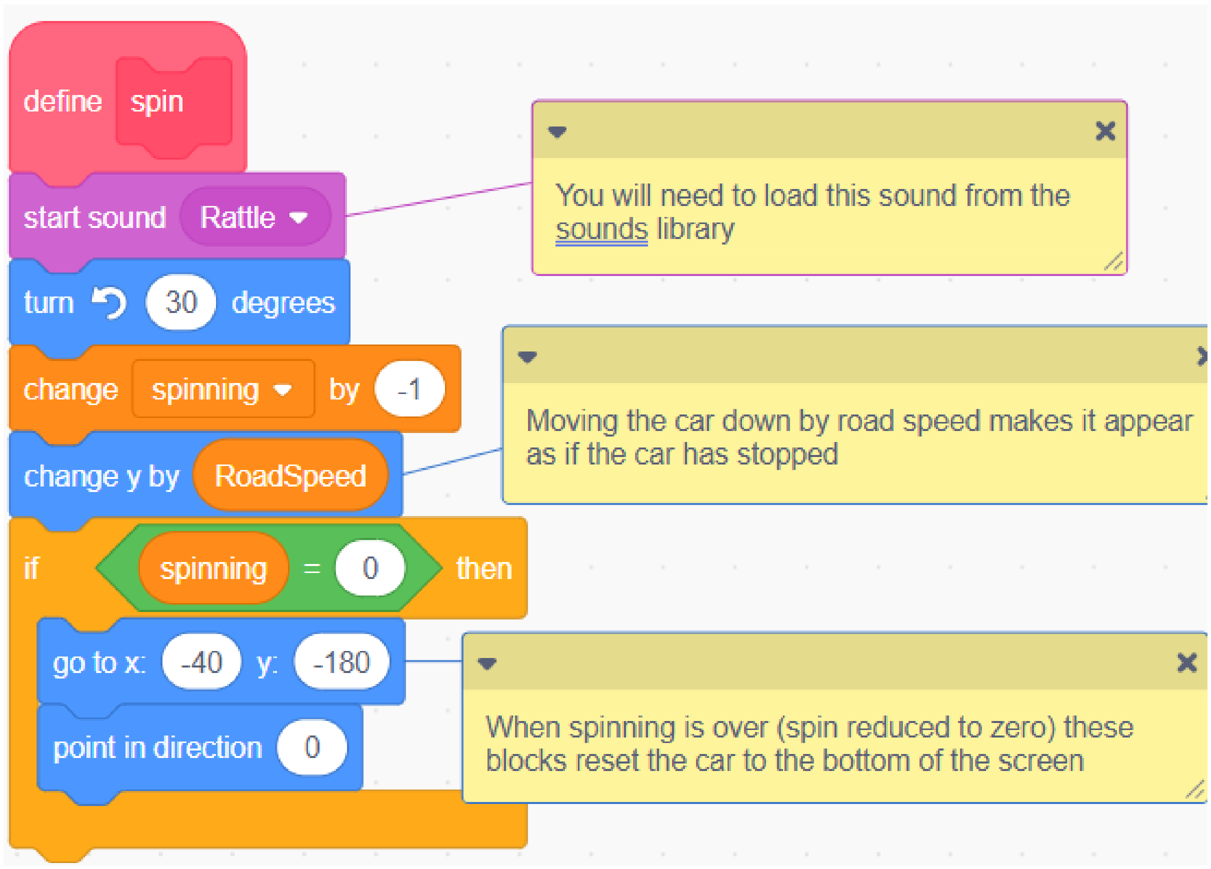
15 Firstly, with the RedCar sprite selected, let’s define another custom block as below, that will handle detecting if the car has collided with the roadside: -



```
define check collisions
if <<touching Road1 ?> or <touching Road2 ?>> then
    set spinning to 30
if <touching BlueCar ?> then
    broadcast bounce
```

touching only detects collision with the painted parts of the sprite and so not the road itself

16 Now let’s define another custom block “spin” to handle making the car spin out of control whenever a collision happens: -



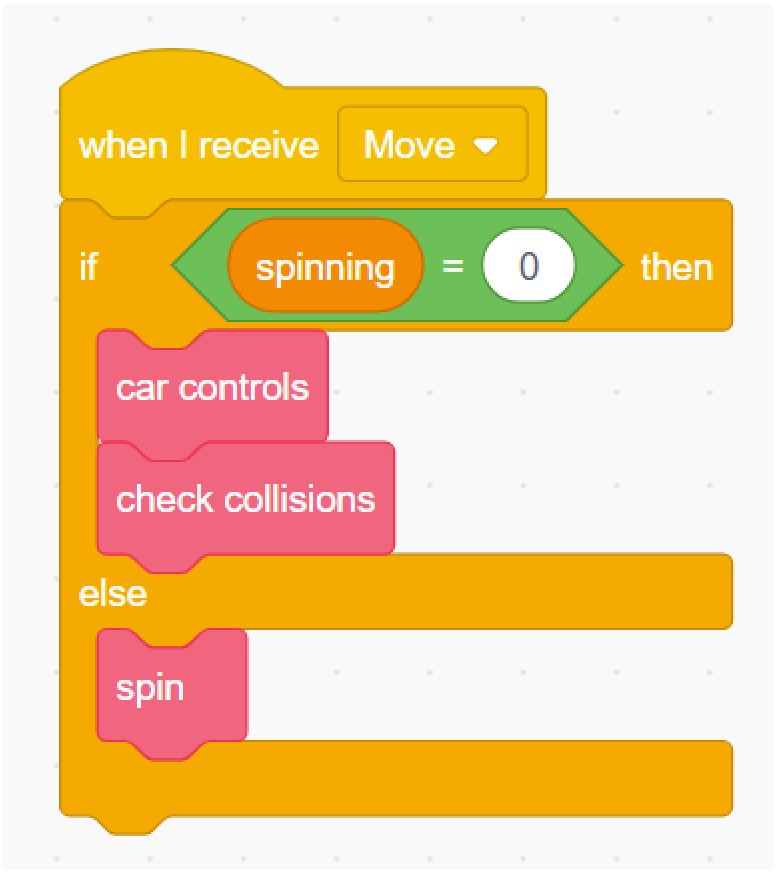
```
define spin
start sound Rattle
turn ↺ 30 degrees
change spinning by -1
change y by (RoadSpeed)
if <(spinning) = 0> then
    go to x: -40 y: -180
    point in direction 0
```

You will need to load this sound from the sounds library

Moving the car down by road speed makes it appear as if the car has stopped

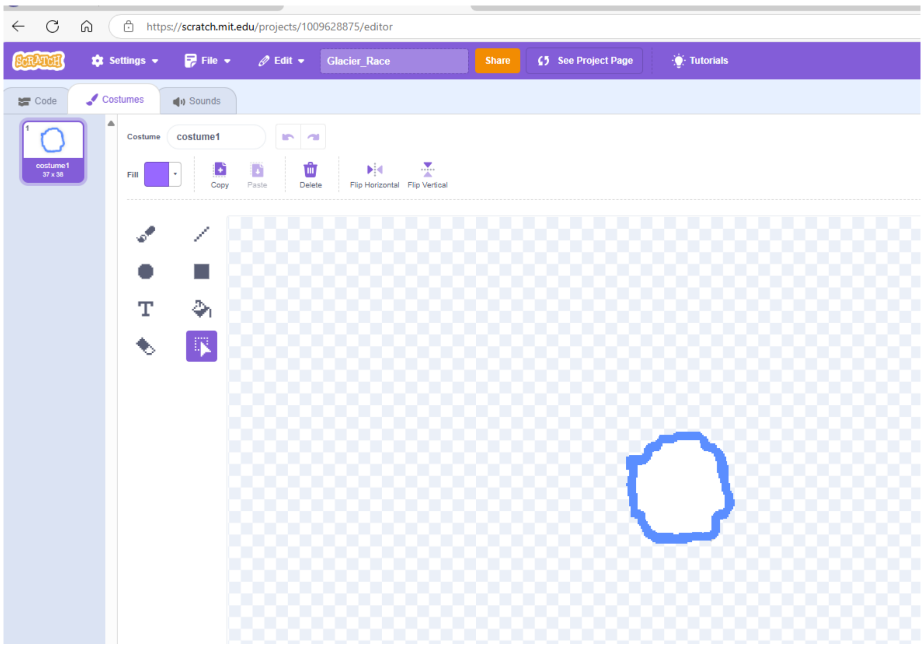
When spinning is over (spin reduced to zero) these blocks reset the car to the bottom of the screen

17 Update the "move" message handling as below – this will mean we only check for car controls and collisions when the car is NOT in a spin, also the car should now go into a spin if it hits the roadside: -



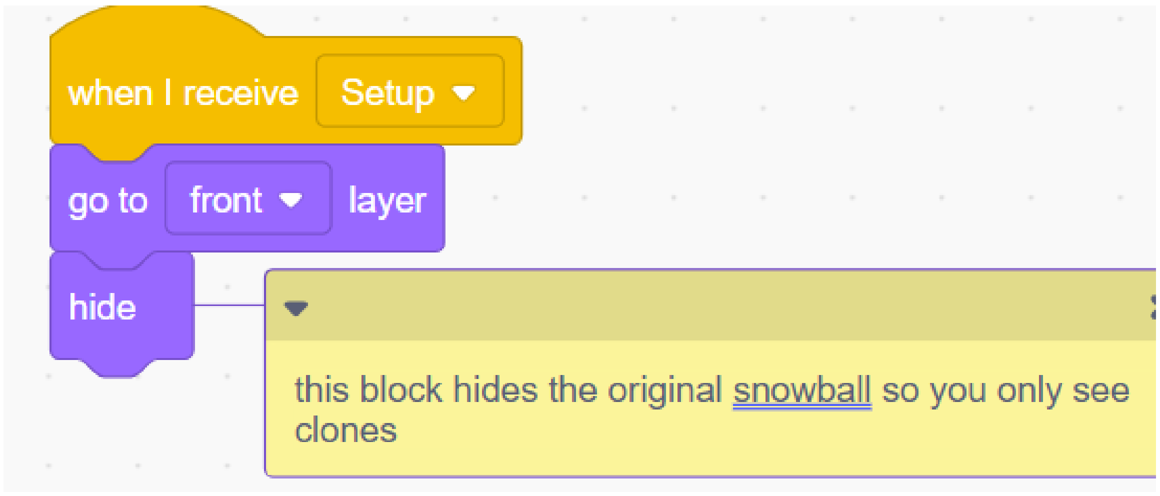
```
when I receive Move
if spinning = 0 then
    car controls
    check collisions
else
    spin
```

18 Lets add some snowball obstacles! Create a new sprite and name it "Snowball". Make it about the same size as the car on stage. You can also see the size in pixels in the costume bar in the left. Fill the snowball in with white using the "Fill" tool: -



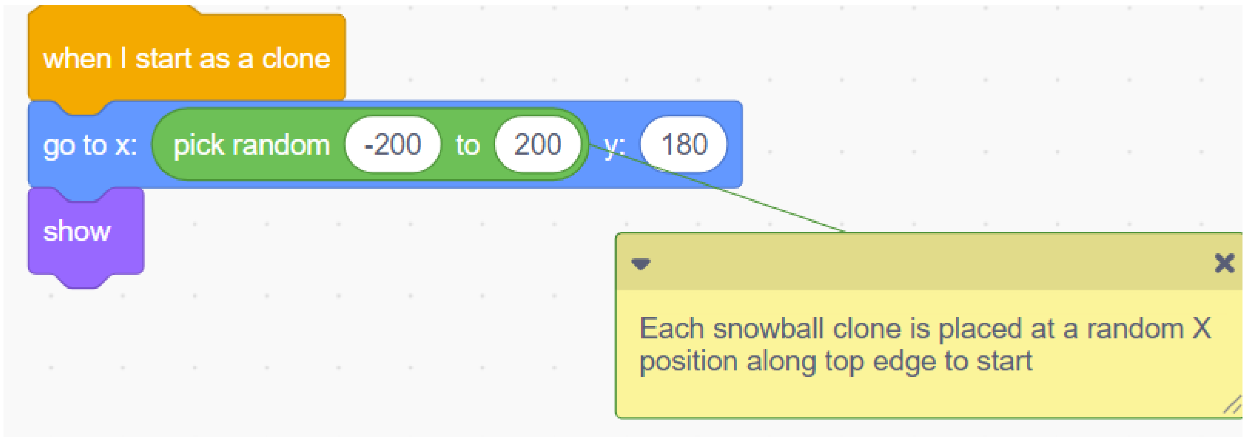
Created : 05/10/2024 08:20:00

19 Add the following 3 blocks to control how the snowball will behave every time one is cloned : -



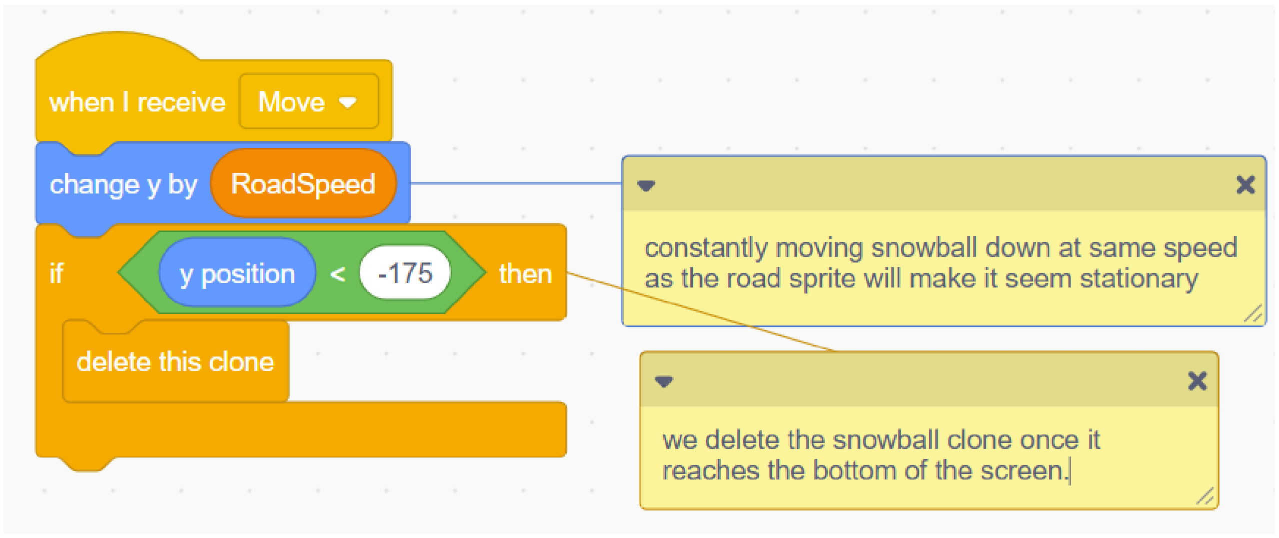
```
when I receive Setup
go to front layer
hide
```

this block hides the original snowball so you only see clones



```
when I start as a clone
go to x: (pick random -200 to 200) y: 180
show
```

Each snowball clone is placed at a random X position along top edge to start



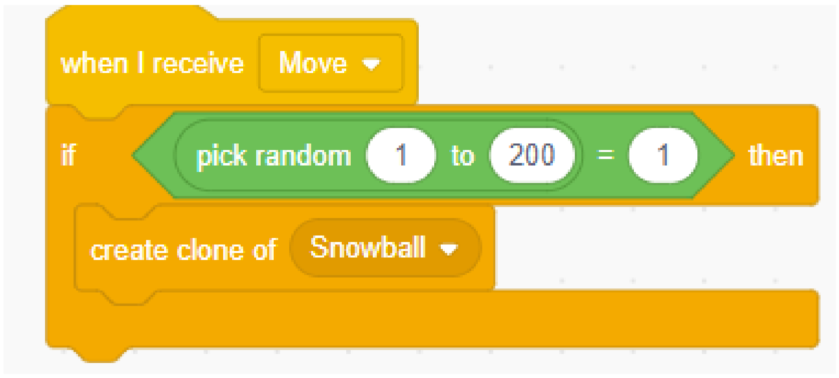
```
when I receive Move
change y by RoadSpeed
if y position < -175 then
    delete this clone
```

constantly moving snowball down at same speed as the road sprite will make it seem stationary

we delete the snowball clone once it reaches the bottom of the screen.

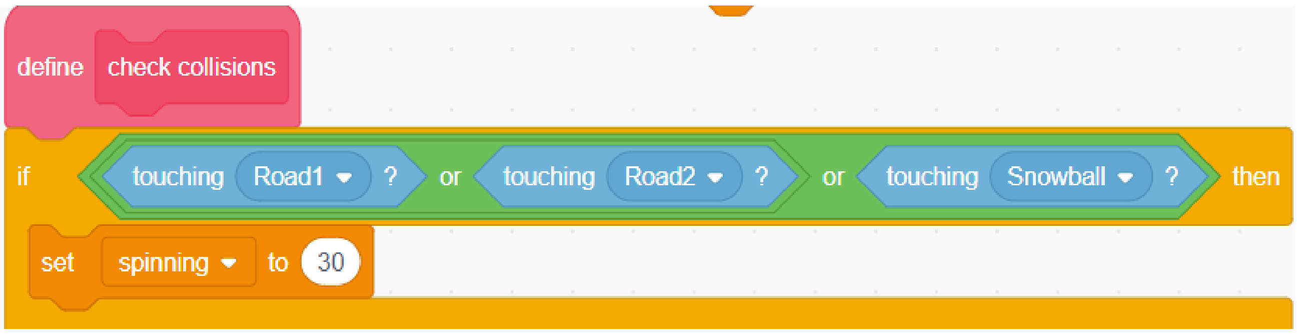
20 Now select the Game Loop sprite and add the following code to make snowballs appear at random intervals – feel free to experiment with increasing or decreasing the range to make more or less snowballs appear !

Created : 05/10/2024 08:20:00



```
when I receive Move
if pick random 1 to 200 = 1 then
    create clone of Snowball
```

21 To make the car collide with snowballs and go into a spin – just add it to the touching conditions check as below: -



```
define check collisions
if touching Road1 ? or touching Road2 ? or touching Snowball ? then
    set spinning to 30
```

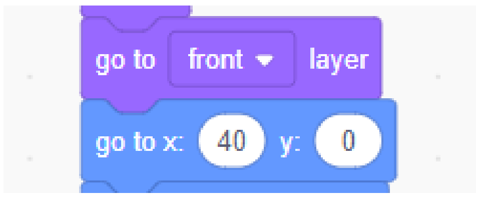
PLAYER TWO -!

Now we turn to coding for player two – easy – simply duplicate the red car sprite, colour its costume blue and tweak the code blocks as below: -

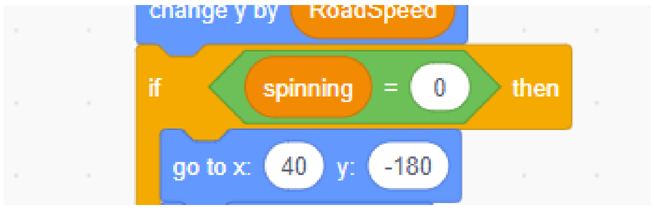
22 Duplicate the red car sprite and rename it BlueCar. Note this copies all the code blocks and BlueCar even gets its own copy of the “spinning” variable [as we defined it as “for this sprite only”.

23 Select the BlueCar sprite and click on the customes tab and use the fill tool to change the car colour to blue.

24 Change the X position in both the “when I receive setup” and “define spin” code below, this so the blue car is always placed just to the side of the red car when starting or restarting:



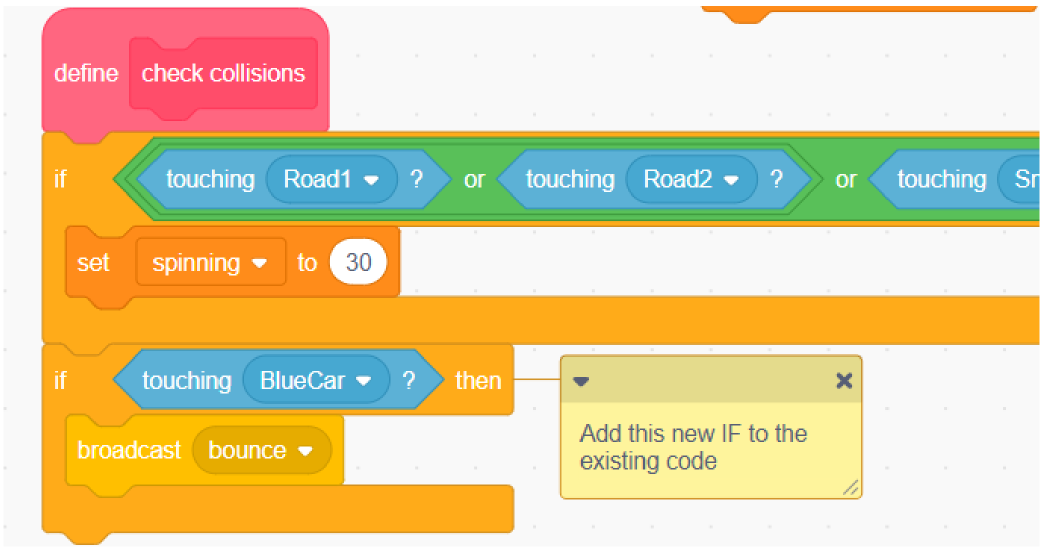
```
go to front layer
go to x: 40 y: 0
```



```
change y by RoadSpeed
if spinning = 0 then
    go to x: 40 y: -180
```

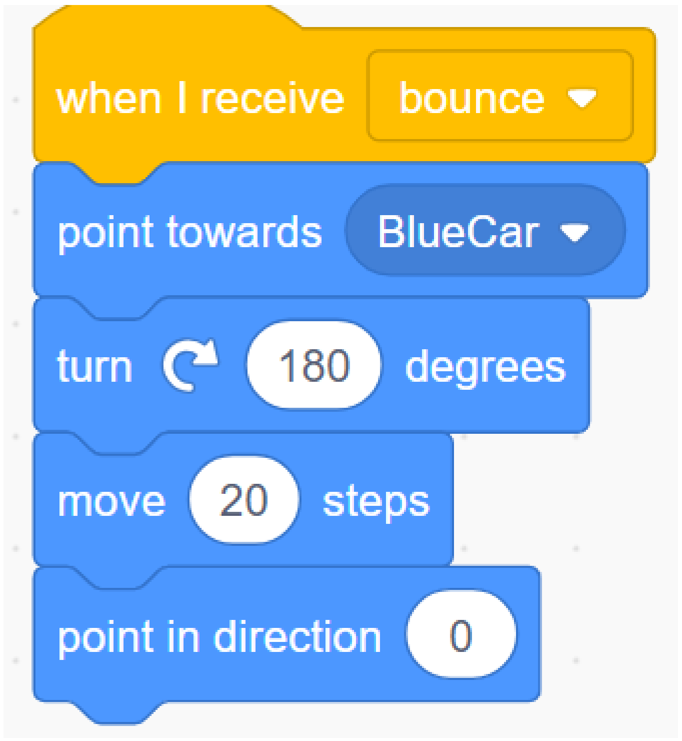
25 In the “Define car controls” code, change the keys that control the movement of the car to the arrow keys instead. If you test the game now, both cars can be controlled but for the moment they can drive through each other!

26 OK so to make this more realistic, let’s add code so that the cars can “sense” each other and when they collide they bounce apart. Return to the RedCar sprite and add the broadcast of a new bounce message if touching the Blue Car and then add an event to handle bouncing when the message is raised: -



```
define check collisions
if < <touching Road1 ?> or <touching Road2 ?> or <touching Sr
    set spinning to 30
if <touching BlueCar ?> then
    broadcast bounce
```

Add this new IF to the existing code



```
when I receive bounce
point towards BlueCar
turn ↻ 180 degrees
move 20 steps
point in direction 0
```

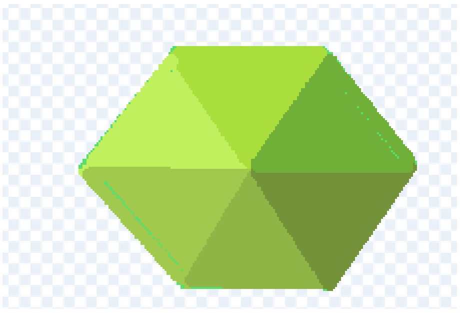
Pointing towards Bluecar but then turning a half circle (i.e. 180 degrees) has the effect of pointing directly away from the blue car..

27 Add the same code to BlueCar sprite, but change all the sprite references in the code to RedCar instead of BlueCar.

COLLECTING GEMS

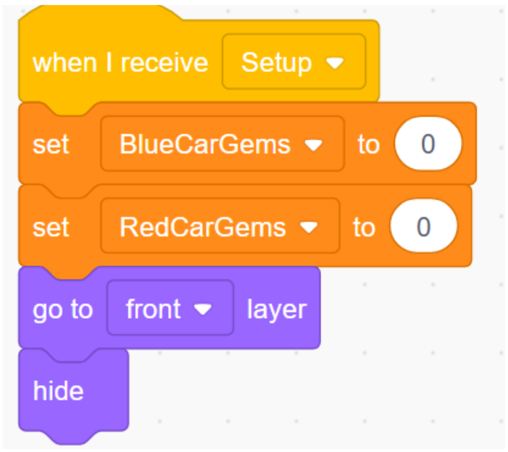
Next part of the game to add, is to add colorful gems for the 2 players to battle it out to collect. The winner will be whoever collects the most before time runs out. Again we will use the clone technique to create multiple gems at once.

28 Create a new sprite and call it simply "Gem". In the custom editor design something like below, 6 triangles arranged in a hexagon shape, colour each a slightly darker shade of green to get the shadow effect: -

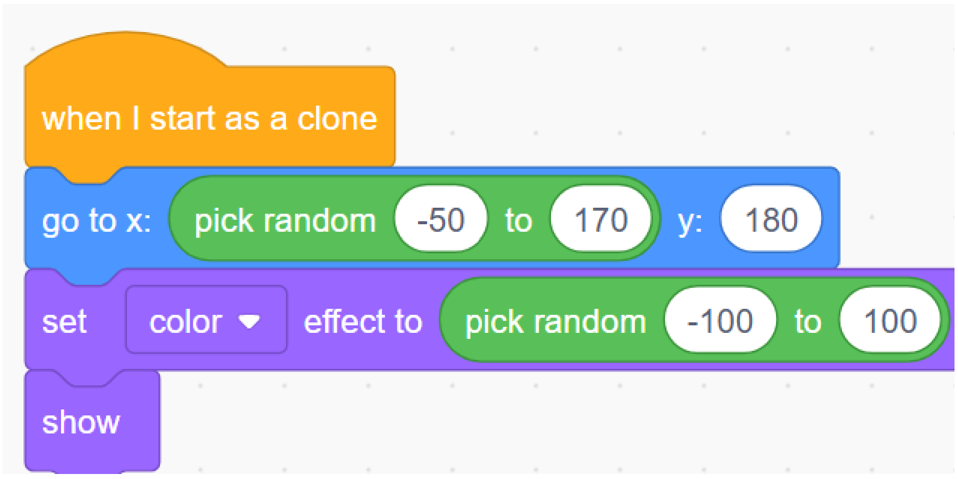


29 Create two new variables, RedCarGems and BlueCarGems. Both can be created "for all sprites". These will be used to keep track of how many gems each car has collected.

Now add the following code blocks, which are very similar to the code used to create the snowballs: -



```
when I receive [Setup ▾]
set [BlueCarGems ▾] to (0)
set [RedCarGems ▾] to (0)
go to [front ▾] layer
hide
```

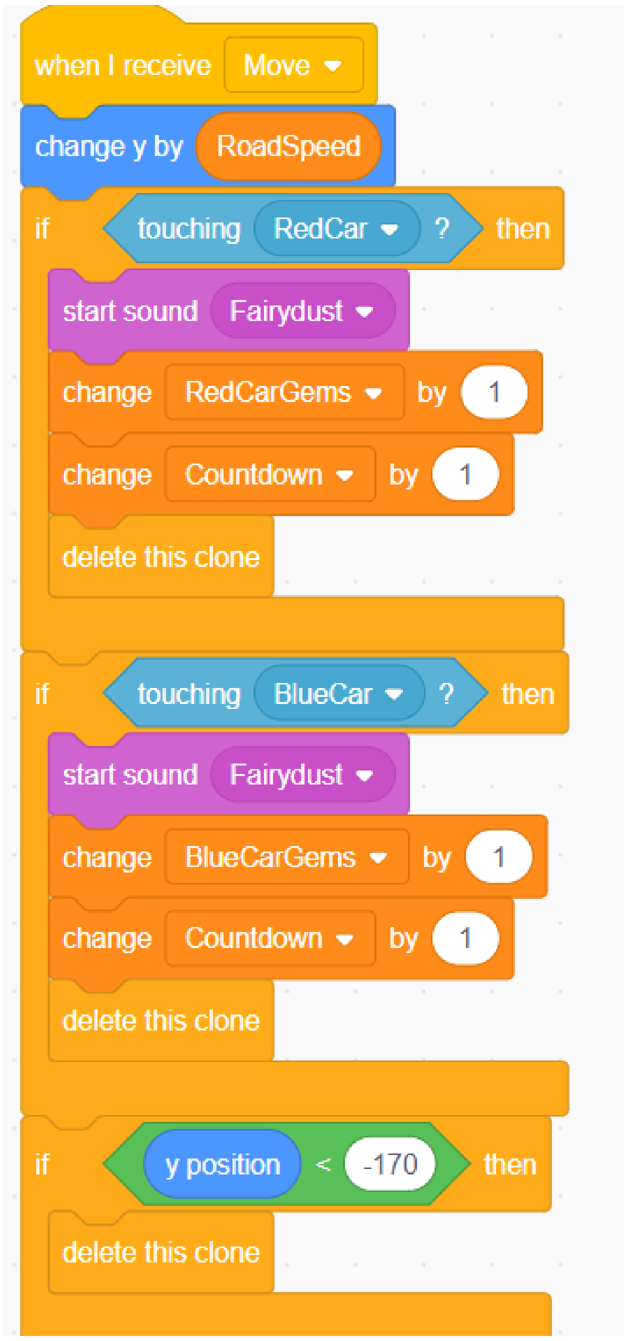


```
when I start as a clone
go to x: (pick random (-50) to (170)) y: (180)
set [color ▾] effect to (pick random (-100) to (100))
show
```

You can experiment with the width of X value in the random range in the go to – this sets essentially how far left or how far right your gems can appear and what's best will depend on how you have designed your road sprite.

30 Add the following code to move the Gems at roadspeed (so like the snowballs they appear stationary), and so the players can collect them and score for each. You will have to load the "fairydust" sound in the sounds tab for the Gem sprite.

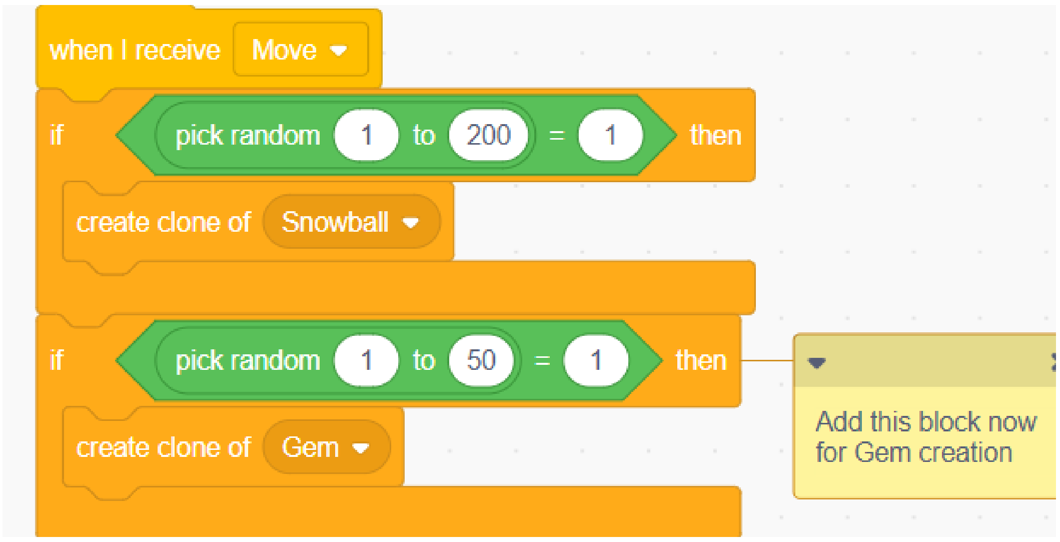
Each time a gem is collected we also add 1 second to the game over countdown.



```
when I receive [Move]
change y by (RoadSpeed)
if <touching [RedCar] ?> then
    start sound [Fairydust]
    change [RedCarGems] by (1)
    change [Countdown] by (1)
    delete this clone
if <touching [BlueCar] ?> then
    start sound [Fairydust]
    change [BlueCarGems] by (1)
    change [Countdown] by (1)
    delete this clone
if <(y position) < (-170)> then
    delete this clone
```

The last IF detects if the Gem has reached the bottom of the screen, and deletes the clone at this point.

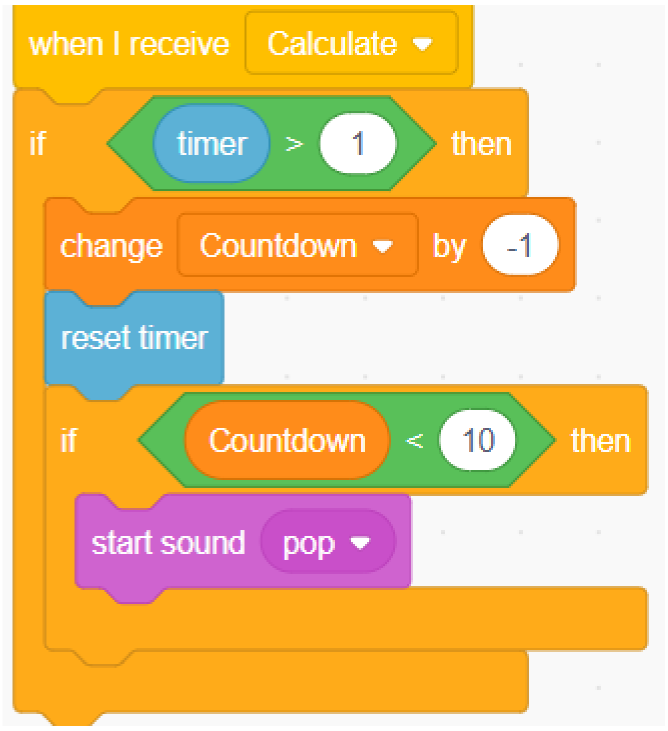
31 In the Game Loop sprite add the following code to randomly spawn Gems. You should now be able to test the game, and score by collecting Gems. The snowballs create balance in the game to discourage players from racing to the top for Gems. You can experiment with the numbers in the random range to tweak how frequently gems and snowballs appear.



```
when I receive [Move v]
if <(pick random (1) to (200)) = (1)> then
    create clone of [Snowball v]
end
if <(pick random (1) to (50)) = (1)> then
    create clone of [Gem v]
end
```

Add this block now for Gem creation

32 You may notice that the countdown isn't working and the game never ends, add the following code, to the Game Loop sprite, which uses the special "timer" variable to detect each time 1 second has passed and subtract 1 from the countdown. A "Pop" sound (will need to be loaded from the sound library) sounds for each of the last 10 seconds to warn players. And when countdown reaches zero the gameloop will exit and the game should stop:-



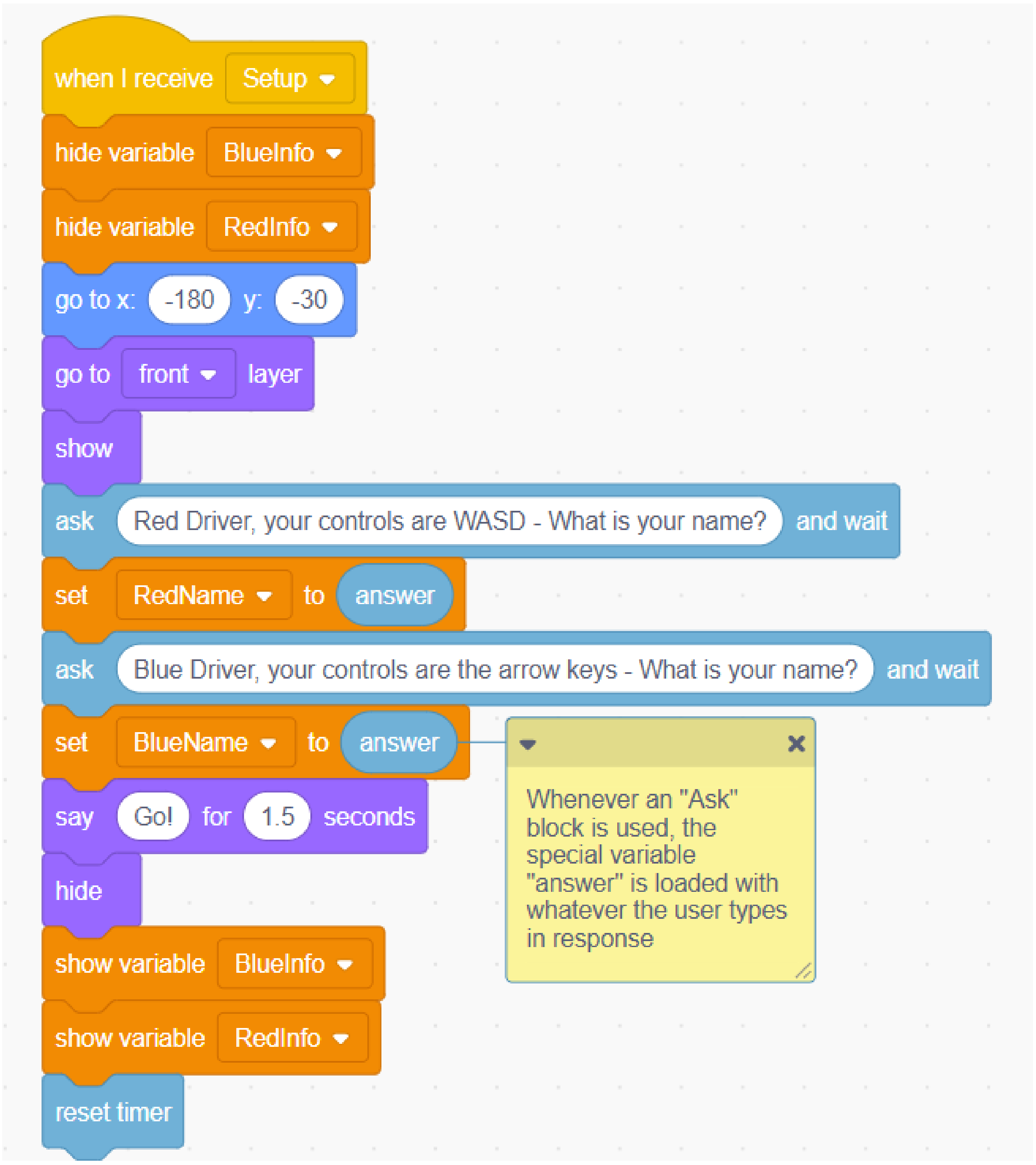
```
when I receive [Calculate v]
if <(timer) > (1)> then
    change [Countdown v] by (-1)
    reset timer
    if <(Countdown) < (10)> then
        start sound [pop v]
    end
end
```

PENGUIN IN CHARGE

Let's add a penguin race official to ask the players their names, start the race and announce the winner at the end.

33 First create 4 new variables, each defined "for all sprites". We need "RedName" and "BlueName" to store each driver's name. And "RedInfo" and "BlueInfo" to display each driver's score during the race. Also add a new sprite, the "Penguin 2" sprite from the library, and finally load the "Gong" sound for this new sprite.

34 Add the following code to the Penguin sprite. This will ask the players their names and show their names and current scores on screen. And because the game loop uses a "broadcast and wait" command – the race won't start until this code has completed: -



```
when I receive [Setup ▾]
hide variable [BlueInfo ▾]
hide variable [RedInfo ▾]
go to x: (-180) y: (-30)
go to [front ▾] layer
show
ask (Red Driver, your controls are WASD - What is your name?) and wait
set [RedName ▾] to (answer)
ask (Blue Driver, your controls are the arrow keys - What is your name?) and wait
set [BlueName ▾] to (answer)
say (Go!) for (1.5) seconds
hide
show variable [BlueInfo ▾]
show variable [RedInfo ▾]
reset timer
```

Whenever an "Ask" block is used, the special variable "answer" is loaded with whatever the user types in response