**ACTIVITY: Line Rider**

BY: Sam Germain

GRADE and CAMP: Grade 7-9, Codemakers (Love 2D)

TOPIC(s): Game Developement

|  |
| --- |
| TIME: A few hours |

OBJECTIVE: To combine the skills learned into a final project

MATERIALS:

* 😐

SCIENTIFIC BASIS (learning outcomes - teach this):

All the necessary skills have already been taught to the kids, they must use their previous code to figure out how to make this project

PROCEDURE

Get them to visit [www.linerider.com](http://www.linerider.com) and play line rider for 5 minutes or so, so they have an idea of the game that they are trying to make.

Get the kids to open the main.lua, and camera.lua files within the line\_rider folder, they DO NOT need to open the eraser file(that comes later).

**Love.load()**

The 4 variables drawn\_x, drawn\_y, oldx and oldy must be created and initialized to 0 or nil

*drawn\_x = 0 --position to be drawn on when the mouse is pressed*

*drawn\_y = 0 --position to be drawn on when the mouse is pressed*

*oldx = 0*

*oldy = 0*

The Rider must be created and put into the world. The rider is actually just a rectangle, with the toboggan rider drawn overtop of the rectangle, this was coded for the kids in the quidditch game, although the kids did not code it themselves. Friction should be set to 0, so that the rider can do loops and stuff. The kids have not learned how to do this, but it’s in the comments.

*objects.rider = {}*

*objects.rider.body = love.physics.newBody(world, 650/2, 0, "dynamic")*

*objects.rider.width = 40*

*objects.rider.height = 30*

*objects.rider.shape = love.physics.newRectangleShape(objects.rider.width, objects.rider.height)*

*objects.rider.fixture = love.physics.newFixture(objects.rider.body, objects.rider.shape) objects.rider.fixture:setFriction(0.0)*

*objects.rider.img = love.graphics.newImage("images/tobboggan.png")*

**love.update()**

The if not pause if statement must be modified to include both of the Camera functions

*if not pause then*

*world:update(dt) --this puts the world into motion*

*Camera:follow(dt,objects.rider)*

*else*

*Camera.update(dt)*

*End*

Code must be put in to draw a line if the mouse is down. This can basically just be copied from the pinball game, but Camera.x and Camera.y must be subtracted from drawn\_x and drawn\_y

*if love.mouse.isDown(1) then*

*drawn\_x = love.mouse.getX() + Camera.x --x coordinate of the mouse*

*drawn\_y = love.mouse.getY() + Camera.y --y coordinate of the mouse*

*if oldx ~= nil then*

*line = {}*

*line.x1 = oldx*

*line.x2 = drawn\_x*

*line.y1 = oldy*

*line.y2 = drawn\_y*

*line.body = love.physics.newBody(world, 0, 0, "static")*

*line.shape = love.physics.newEdgeShape(drawn\_x, drawn\_y, oldx, oldy)*

*line.fixture = love.physics.newFixture(line.body, line.shape, 5)*

*table.insert(objects.lines, line)*

*end*

*oldx = drawn\_x*

*oldy = drawn\_y*

*else*

*oldx = nil*

*oldy = nil*

*end*

**love.draw()**

The rider must be drawn. The angle variable makes it so that the rider image turns in accordance with the way the rectangle is facing, so when the top of the rectangle is facing down, the top of the rider image is facing down. Orient\_x and Orient\_y set the center of rotation for the image. When Orient\_x and Orient\_y are set to the top left corner, the image does not touch the surface it’s resting on when upside down.

*rider\_img = objects.rider.img*

*rider\_x = objects.rider.body:getX() - Camera.x*

*rider\_y = objects.rider.body:getY() - Camera.y*

*rider\_angle = objects.rider.body:getAngle()*

*orient\_x = objects.rider.img:getWidth()/2*

*orient\_y = objects.rider.img:getHeight()/2*

*love.graphics.draw(rider\_img, rider\_x, rider\_y, rider\_angle, 1, 1, orient\_x, orient\_y )*

They must draw the lines that are drawn on the screen. This is done exactly like the pinball game but Camera.x and Camera.y must be subtracted from the lines x and y coordinates

*for \_, line in pairs(objects.lines) do*

*love.graphics.line( line.x1 - Camera.x, line.y1 - Camera.y, line.x2 - Camera.x, line.y2 - Camera.y)*

*end*

**Camera.lua**

\*\*For the four day camp, they did not do the quidditch exercise, so the camera is coded for them mostly. They will have to subtract Camera.x and Camera.y from their other objects however, so explain a little bit about the camera, just so they kind of understand why they have to subtract it.

The Camera object exists as an empty object

*Camera = {}*

X and y coordinates for camera must be created and set to 0

*Camera = {*

*x = 0,*

*y = 0*

*}*

**Camera.update**

The actions for the arrow keys must be coded into camera.update. This includes adding and subtracting from Camera.x and Camera.y if an arrow is being pressed. This function only operates when the game is paused.

*if love.keyboard.isDown("right") then --RIGHT ARROW BUTTON IS DOWN then*

*Camera.x = Camera.x + 5*

*elseif love.keyboard.isDown("left") then*

*Camera.x = Camera.x - 5*

*end*

*if love.keyboard.isDown("up") then*

*Camera.y = Camera.y - 5*

*elseif love.keyboard.isDown("down") then*

*Camera.y = Camera.y + 5*

*end*

**Camera.follow**

When the game is not paused, the camera.follow function is used to set Camera.x and Camera.y to the coordinates of the rider.

*Camera.x = rider.body:getX() - love.graphics.getWidth()/2*

*Camera.y = rider.body:getY() - love.graphics.getHeight()/2*

**The eraser can be coded after this, or just given to the kids if there is no more time. There is a separate write up for the eraser, with what to do if there is more time or if there is no more time.**

**ACTIVITY: Line Rider Eraser**

BY: Sam Germain

GRADE and CAMP: Grade 7-9, Codemakers (Love 2D)

TOPIC(s): Game Developement

|  |
| --- |
| TIME: 60 mins |

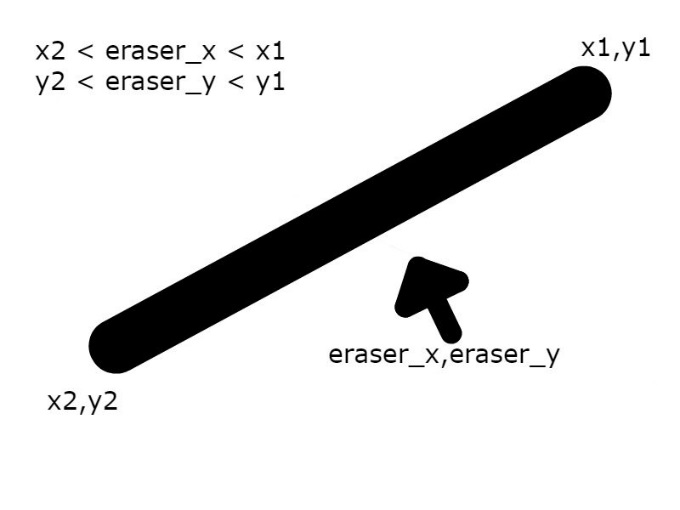
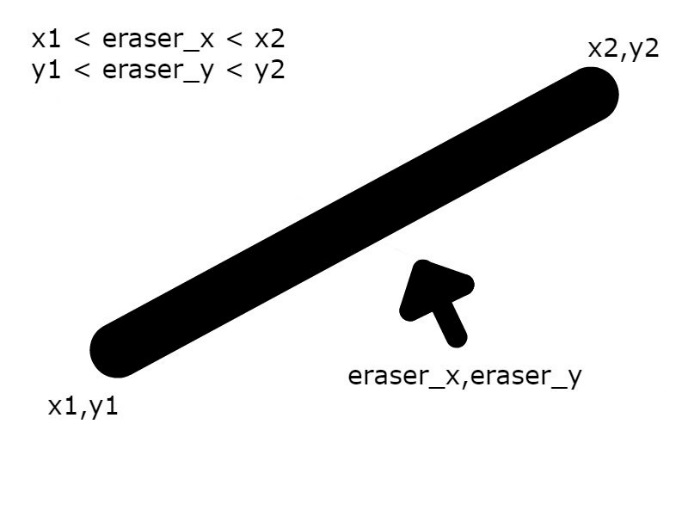
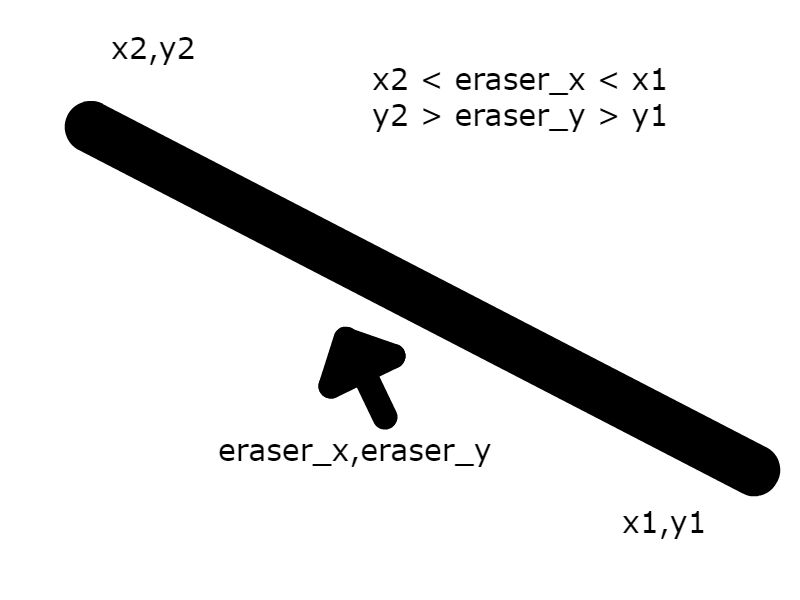
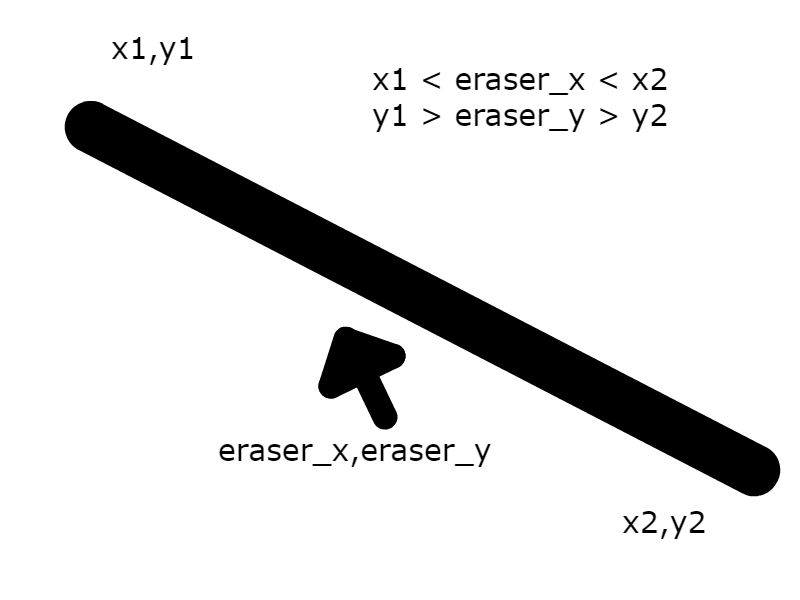
OBJECTIVE: To put the eraser into line rider

MATERIALS:

* 😐

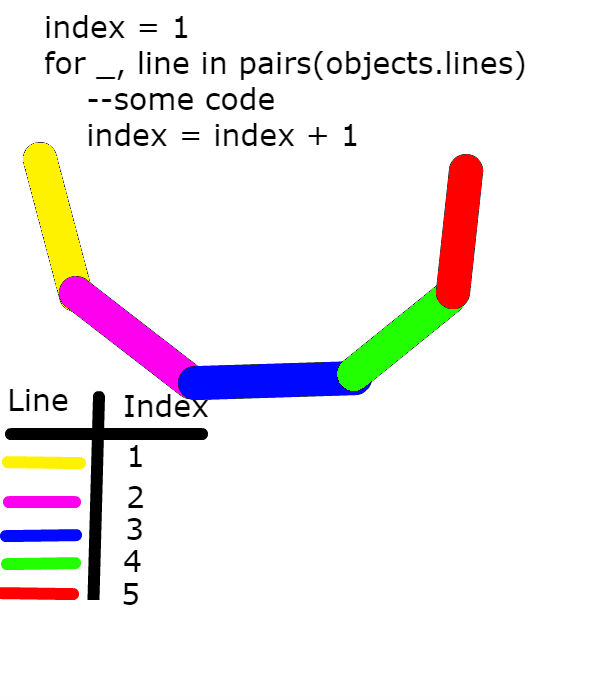
SCIENTIFIC BASIS (learning outcomes - teach this):

The eraser will be coded in the work on the right click of the mouse. A variable, which we will call erase\_x and erase\_y will be assigned to the x and y coordinates of the mouse we the mouse is held down. We need to loop through each of our lines and check if the line spans over the spot we are trying to erase. The pictures below explain the four scenario’s where a line would span over the spot we are trying to erase.



**Indices**

The code is in there for them, but here’s an explanation of whats going on.  
Each line also has an index in a table. The first line segment is the first one in the table, and the second line segement is the second line in the table. If we set a line\_counter to 0, loop through objects.lines, and increase line\_counter by 1 every time we execute the loop, then line\_counter is always the index of the line we’re looking at.



From this image, we can see that when looping through all our lines, when we are on our first line, the index is equal to 1, when we are on our second line, the index is equal to 2, so if we delete the line with index 2, we are deleting the purple line.

PROCEDURE

If there is no more time

* give the kids the eraser.lua file
* put require(“eraser”) at the top of main.lua
* put erase() into draw.update()

Otherwise

* they must edit their eraser.lua file so that it works as an actual eraser
* if the eraser is within the range of the line, I use line.body:setType("dynamic") to make the line fall, this doesn’t actual delete the line, but the line won’t interfere with the game anymore
* Using table.remove(objects.lines, Line\_index) is a comment in the file. If they uncomment this and place it in the correct nest, then the line should no longer be drawn.
* put require(“eraser”) at the top of main.lua
* put erase() into draw.update()