Love2D Tutorial

To use Love2D game engine you need a text editor and the Love2D engine.
You will see tutorials using Atom, Visual Studio Code or ZeroBrane Studio. This tutorial uses ZeroBrane for the following reasons:

1. ZeroBrane is a dedicated Lua IDE and has everything you need to learn both Lua and the Love2D engine in one place.

2. ZeroBrane can run AND debug Love2D scripts, which other text editors cannot.

3. ZeroBrane will automatically find the Love2D engine as long as it is installed in it's default location (C:\Program Files)

Get ZeroBrane Studio (free) from https://studio.zerobrane.com/download?not-this-time
Get Love2D from https://love2d.org/ Choose the 64bit Installer. Current version 11.3

Install both, preferably in their default locations (C:\Program Files(x86)\ZeroBraneStudio and C:\Program Files\LOVE)

To change the appearance and behaviour of Zerobrane, there are two lua files, both called user.lua that you can edit.
Menu→Edit→Preferences→Settings:System and Menu→Edit→Preferences→Settings:User

There is a pre-configured file at https://pastebin.com/AS3E6tYw which you can copy/paste into both the user.lua files. If you do not like the colour scheme, comment out the line with - -

```
--styles = loadfile('cfg/tomorrow.lua')('SciTeLuaIDE')
```

and un-comment an alternative, eg:

```
styles = loadfile('cfg/tomorrow.lua')('Tomorrow')
```

When you have got both installed, and any configuration changes made, you are ready to begin.

Get your filesystem in order:

Every Love2D project has to run in it's own folder.
Do not mix other non-related .lua files with it

Instructions:

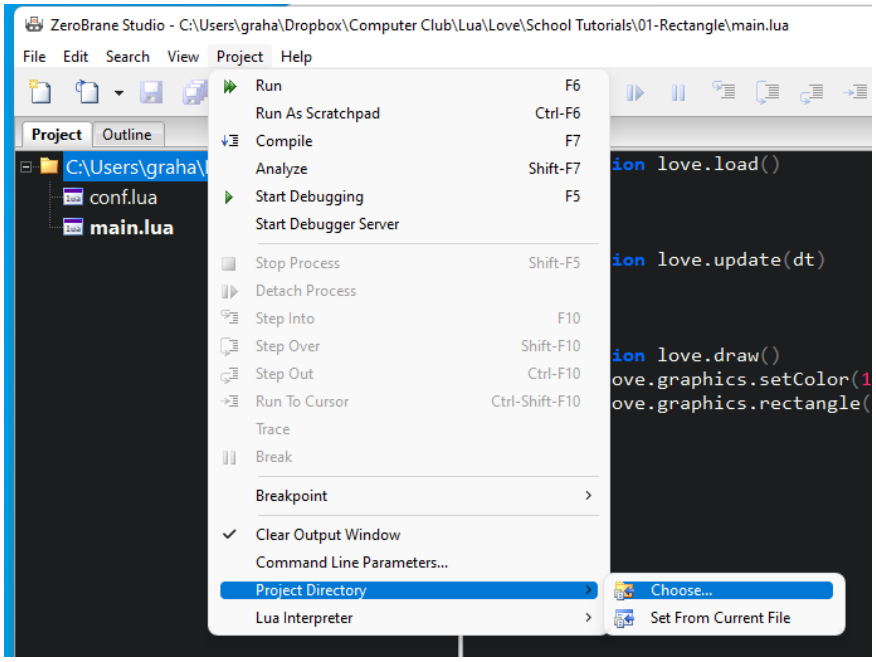
1. If you have not already done so, create a new folder in your Documents called “Love2D”

2. Inside this folder create another folder called “Template”. This will be used to create a simple template, and should not be added to after completion.

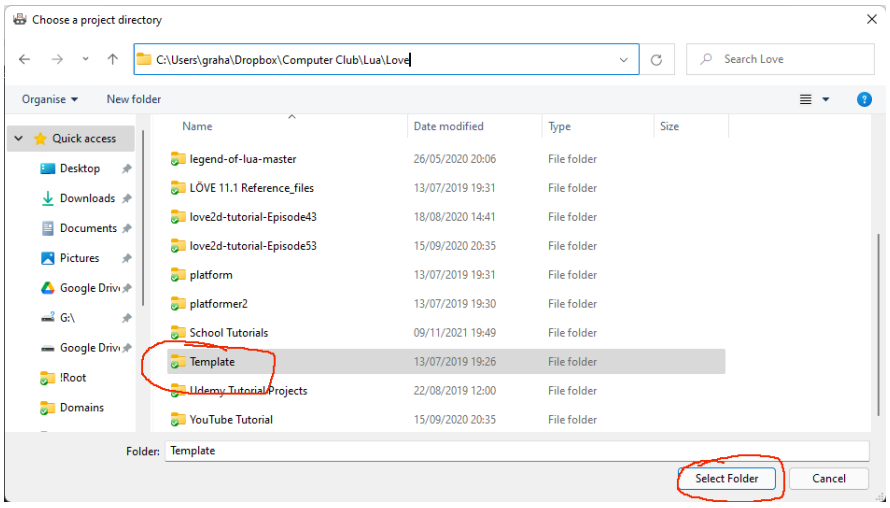
3. To create a new Love2D game copy the Template folder and rename it to reflect the game you are about to write

Setup the Template:

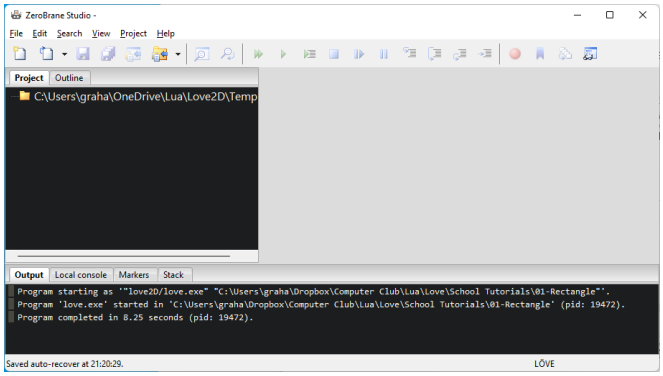
Start ZeroBrane and use the menu Project→ Project Directory→ Choose…



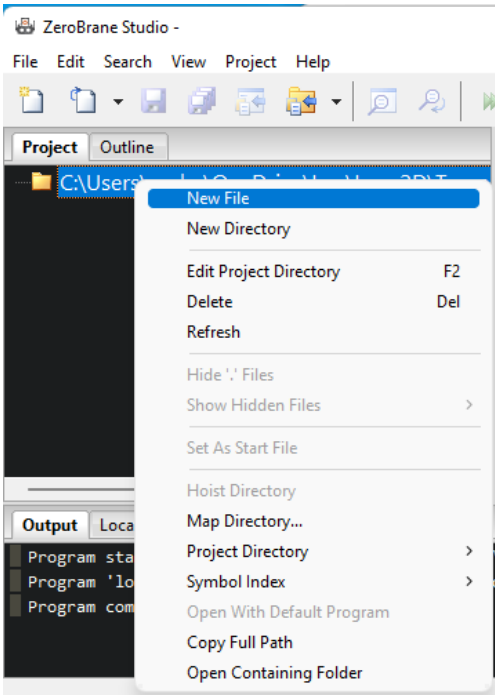
Select the Template folder:



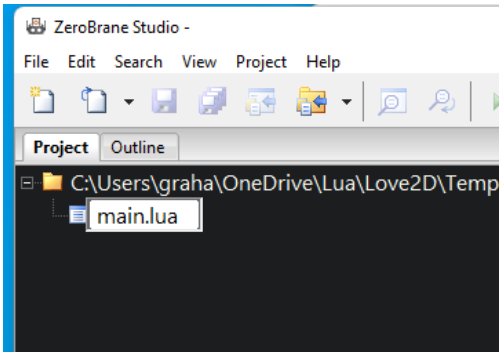
Close the default file called “Untitled”



Right-click on the project folder and select “New File”:

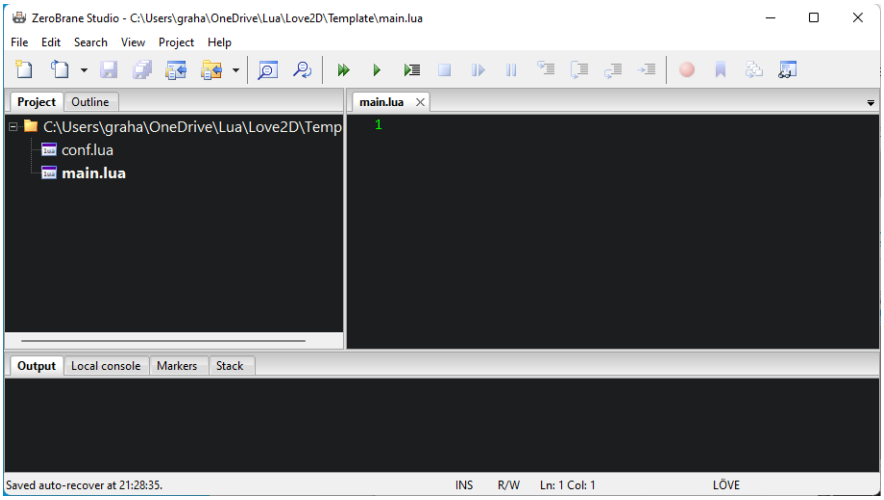


Type main.lua and press Enter



Repeat the above to create a file called conf.lua

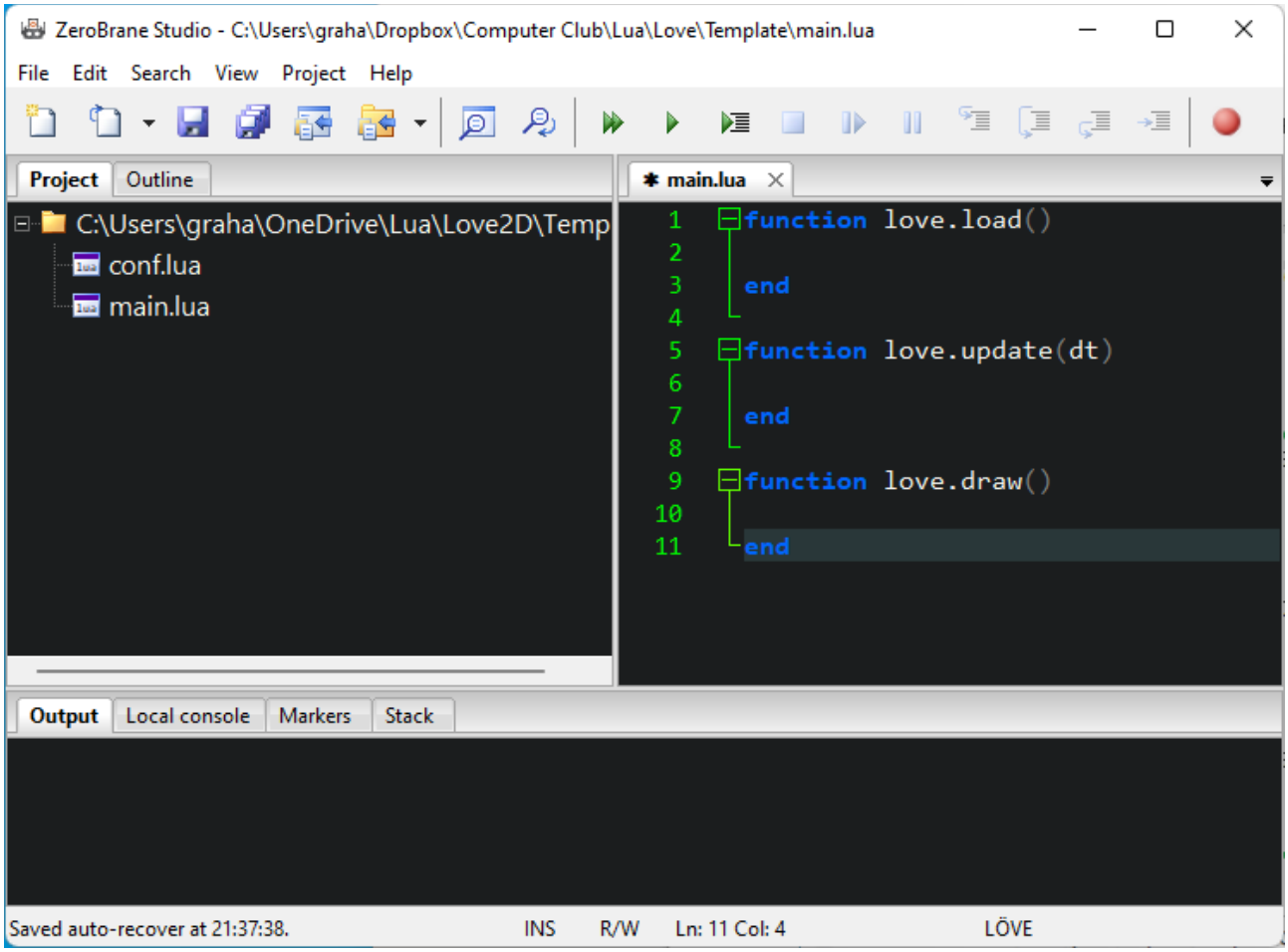
Double-click main.lua:



You are now ready to start coding

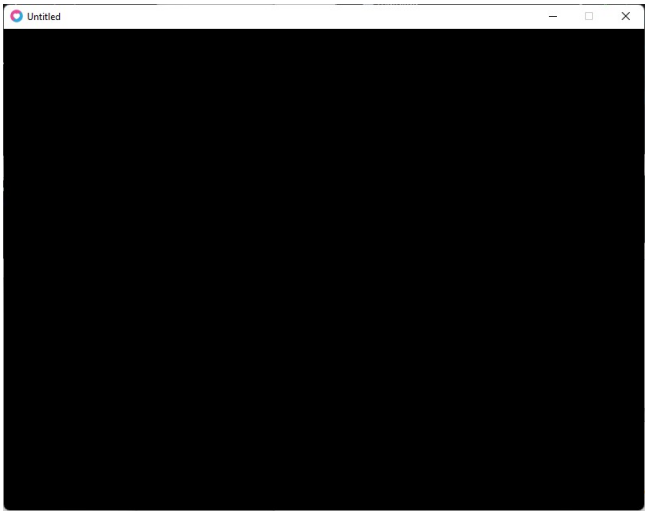
Type the following lines:

```
function love.load()

end

function love.update(dt)

end

function love.draw()

end
```



Save the project by clicking the disks icon on the toolbar or Ctrl-S or File →Save.

Click the double green triangles on the toolbar to run the project:



Not very exciting, but it works!

The 3 functions are the basis for the game engine:

```
function love.load()
```

This is run when the game starts. Use it to setup assets, variables etc.

```
function love.update(dt)
```

This runs every frame (approx 60 x per second). Use this function to make changes to the positions of game objects. The delta-time (dt) parameter allows accurate timings over all hardware by allowing for slower machines to catch up with code execution

```
function love.draw()
```

This also runs at 60 frames per second. Use this function to draw to the screen.

Pong Game version 1

No fancy graphics, just to get you started.

Make a copy of the template folder, rename it to “Pong v1” and open it in ZeroBrane Studio

Add these 2 lines at the top of the script:

```
WIDTH = love.graphics.getWidth()
HEIGHT = love.graphics.getHeight()
```

They are in CAPS because they represent Constants (like a variable, but the contents do not change). They are useful in the code for calculating the positions of game assets (rectangles, circles etc).

All comments appear in green and can be omitted from the code you type in.

Add these lines to love.load()

```
function love.load()
     -- set variables for left paddle in a table
    paddleL = {}
    paddleL.x = 10          -- start 10 pixels in from left edge
    paddleL.y = 260         -- start 260 pixels down from top
    paddleL.w = 15          -- paddle is 15 pixels wide
    paddleL.h = 80          -- paddle is 80 pixels tall
    paddleL.speed = 10

    -- set variables for right paddle in a table
    paddleR = {}
    paddleR.x = WIDTH - 25  -- start 10 pixels from the right edge (10 + width of paddle)
    paddleR.y = 260
    paddleR.w = 15
    paddleR.h = 80
    paddleR.speed = 10

    -- set variables for ball in a table
    ball = {}
    ball.x = WIDTH / 2 - 5  -- start in centre of the screen (half of WIDTH minus ball width)
    ball.y = HEIGHT / 2 - 5 -- start in centre of the screen (half of HEIGHT minus ball height)
    ball.w = 10
    ball.h = 10
    ball.speed = 5
end
```

Here, each of the 3 assests of the pong game: right paddle, left paddle and ball are defined in a table format.
Keeping variables together in this format makes them much easier to handle. Each table has an x,y coordinate width (w), height (h) and a speed.
Add these lines to the love.update() function:

```
function love.update(dt)
    -- w and s keys control left paddle
    if love.keyboard.isDown('w') then -- move left paddle up
        paddleL.y = paddleL.y - paddleL.speed
        if paddleL.y < 0 then -- is left paddle at the top of the screen?
            paddleL.y = 0
        end
    elseif love.keyboard.isDown('s') then -- move left paddle down
        paddleL.y = paddleL.y + paddleL.speed
        if paddleL.y > HEIGHT - paddleL.h then -- is left paddle at the bottom of the screen?
            paddleL.y = HEIGHT - paddleL.h
        end
    end
end
```

These lines will check if either the “w” or “s” keys are being held down:

If “w” then move the paddle up the screen by decreasing paddlL.y by the number of pixels held in paddleL.speed. As this was set to 10 in love.load(), this means that every frame, the paddles y coordinate decreases by 10. If running at 60 frames / sec, the paddle will move by 600 pixels every second.

Similarly, if the “s” key is held down, the y coordinate increases by 10 every frame, and the paddle moves down.

There are checks in place to detect if the paddle moves off the screen:

If it is moving up (towards 0) then an “if” statement checks whether it is less than 0, and if so, sets it back to 0.

If it is moving down (towards the height of the screen) then an “if” statement checks whether the top of the paddle is greater than the height of the screen. An adjustment is made by including the height of the paddle, so it remains visible. This is effectively checking the bottom of the rectangle, as the paddleL bottom coordinate is the top coordinate + paddleL height

To move the right paddle, using the “up” and “down” arrow keys, add the following lines which work in exactly the same way:

```
    -- up and down keys control right paddle
    if love.keyboard.isDown('up') then -- move right paddle up
        paddleR.y = paddleR.y - paddleR.speed
        if paddleR.y < 0 then -- is left paddle at the top of the screen?
            paddleR.y = 0
        end
    elseif love.keyboard.isDown('down') then -- move right paddle down
        paddleR.y = paddleR.y + paddleR.speed
        if paddleR.y > HEIGHT - paddleR.h then -- is left paddle at the bottom of the screen?
            paddleR.y = HEIGHT - paddleR.h
        end
    end
```

To move the ball for testing purposes, just back and forwards across the screen, add the following lines to love.update()

```
    -- just move the ball left and right for testing
    ball.x = ball.x + ball.speed
    if ball.x < 0 or ball.x > WIDTH then -- if ball at screen edge, put it back in the centre
        ball.x = WIDTH / 2 - ball.w / 2
    end
```

The way this works, is to move the ball across the screen by the number of pixels in ball.speed (5) from left to right.
If the speed happens to be a negative number, then the ball will go from right to left instead.

To get the game running, the next step is to add some code to the love.draw() function:

```
function love.draw()
    love.graphics.clear(0.18, 0.16, 0.2)    -- clear screen with a dark grey colour
    love.graphics.rectangle('fill', paddleL.x, paddleL.y, paddleL.w, paddleL.h) -- left paddle
    love.graphics.rectangle('fill', paddleR.x, paddleR.y, paddleR.w, paddleR.h) -- right paddle
    love.graphics.rectangle('fill', ball.x, ball.y, ball.w, ball.h)             -- draw ball
end
```

This will draw the paddles and ball at the current coordinates that were set in the update() function.

The game can now be started:
The paddles can be moved up and down.
The ball just moves across the screen from centre to the right, then re-appears in the centre.
Everything is in an exciting shade of white.
The paddles are completely useless and do nothing.

There needs to be some sort of collision detection, to check if the ball has hit a paddle.

The next step is to write a collision function:

This is NOT inside any of the 3 existing love. Built-in functiond, it is a new one you write yourself.

It is important it is placed above the 3 game functions, otherwise it will not work:

```
function collides(rect1, rect2)
    --[[ check whether rectangles are NOT colliding ]]

    -- if left side of rect1 is beyond rect2 right side
    -- OR left side of rect2 is beyond rect1 right side
    if rect1.x > rect2.x + rect2.w or rect2.x > rect1.x + rect1.w then
        return false
    end
    -- if top side of rect1 is beyond bottom of rect2
    -- OR top side of rect2 is beyond bottom of rect1
    if rect1.y > rect2.y + rect2.h or rect2.y > rect1.y + rect1.h then
        return false
    end

    -- code only gets this far if both if statements above fail
    return true
end
```

This works by checking whether the coordinates of the 2 rectangles passed in as parameters (rect1, rect2) are intersecting with each other in 2 separate if statements.

The first if statement checks whether the left side of rect1 is beyond the right side of rect2 or vice-versa. If so, the rectangles cannot intersect, so return false and the function exits.

The second if statement checks if the top of rect1 is beyond the bottom of rect2, and vice-versa. If so, the rectangles cannot intersect, so return false and the function exits.

If neither of these if statements are true, then the rectangles are intersecting, so return true.

To make use of this function, add these lines to the love.update() function:

```
    if collides(ball, paddleR) then         -- has the ball hit the right paddle?
        ball.speed = ball.speed * -1        -- reverse the direction right -> left
    elseif  collides(ball, paddleL) then    -- has the ball hit the right paddle?
        ball.speed = math.abs(ball.speed)   -- reverse the direction left -> right
    end
```

Run the game again, and this time the paddles can be used to bounce the ball back in the opposite direction.

It will not go up or down but at least the concept is now working.
The full code, including additional comments is listed below:

```
WIDTH = love.graphics.getWidth()
HEIGHT = love.graphics.getHeight()

function collides(rect1, rect2)
    --[[ check whether rectangles are NOT colliding ]]

    -- if left side of rect1 is beyond rect2 right side
    -- OR left side of rect2 is beyond rect1 right side
    if rect1.x > rect2.x + rect2.w or rect2.x > rect1.x + rect1.w then
        return false
    end
    -- if top side of rect1 is beyond bottom of rect2
    -- OR top side of rect2 is beyond bottom of rect1
    if rect1.y > rect2.y + rect2.h or rect2.y > rect1.y + rect1.h then
        return false
    end

    -- code only gets this far if both if statements above fail
    return true
end

function love.load()
    -- set variables for left paddle in a table
    --[[
    ---------------------------------------------------------------
    |         tables of variables for paddles and ball            |
    ---------------------------------------------------------------
    |   name  |x position | y position | width | height | speed |
    ---------------------------------------------------------------
    | paddleL |     10    |     260    |   15  |   80   |   10  |
    ---------------------------------------------------------------
    | paddleR | WIDTH - 25|     260    |   15  |   80   |   10  |
    ---------------------------------------------------------------
    |   ball  | WIDTH/2-5 | HEIGHT/2-5 |   10  |   10   |    5  |
    ---------------------------------------------------------------
    ]]
    paddleL = {}
    paddleL.x = 10          -- start 10 pixels in from left edge
    paddleL.y = 260         -- start 260 pixels down from top
    paddleL.w = 15          -- paddle is 15 pixels wide
    paddleL.h = 80          -- paddle is 80 pixels tall
    paddleL.speed = 10

    -- set variables for right paddle in a table
    paddleR = {}
    paddleR.x = WIDTH - 25  -- start 10 pixels from the right edge (10 + width of paddle)
    paddleR.y = 260
    paddleR.w = 15
    paddleR.h = 80
    paddleR.speed = 10

    -- set variables for ball in a table
    ball = {}
    ball.x = WIDTH / 2 - 5  -- start in centre of the screen (half of WIDTH minus ball width)
    ball.y = HEIGHT / 2 - 5 -- start in centre of the screen (half of HEIGHT minus ball height)
    ball.w = 10
    ball.h = 10
    ball.speed = 5
end
```

```
function love.update(dt)
    -- w and s keys control left paddle
    if love.keyboard.isDown('w') then -- move left paddle up
        paddleL.y = paddleL.y - paddleL.speed
        if paddleL.y < 0 then -- is left paddle at the top of the screen?
            paddleL.y = 0
        end
        -- paddleL.y = math.max(0, paddleL.y - paddleL.speed)
    elseif love.keyboard.isDown('s') then -- move left paddle down
        paddleL.y = paddleL.y + paddleL.speed
        if paddleL.y > HEIGHT - paddleL.h then -- is left paddle at the bottom of the screen?
            paddleL.y = HEIGHT - paddleL.h
        end
        -- paddleL.y = math.min(HEIGHT - paddleL.h, - paddleL.y + paddleL.h)
    end

    -- up and down keys control right paddle
    if love.keyboard.isDown('up') then -- move right paddle up
        paddleR.y = paddleR.y - paddleR.speed
        if paddleR.y < 0 then -- is left paddle at the top of the screen?
            paddleR.y = 0
        end
    elseif love.keyboard.isDown('down') then -- move right paddle down
        paddleR.y = paddleR.y + paddleR.speed
        if paddleR.y > HEIGHT - paddleR.h then -- is left paddle at the bottom of the screen?
            paddleR.y = HEIGHT - paddleR.h
        end
    end

    -- just move the ball left and right for testing
    ball.x = ball.x + ball.speed
    if ball.x < 0 or ball.x > WIDTH then -- if ball has hit the edge of the screen, put it back in the centre
        ball.x = WIDTH / 2 - ball.w / 2
    end

    if collides(ball, paddleR) then         -- has the ball hit the right paddle?
        ball.speed = ball.speed * -1        -- reverse the direction right -> left
    elseif  collides(ball, paddleL) then    -- has the ball hit the right paddle?
        ball.speed = math.abs(ball.speed)   -- reverse the direction left -> right
    end
end

function love.draw()
    love.graphics.clear(0.18, 0.16, 0.2)    -- clear screen with a dark grey colour
    love.graphics.rectangle('fill', paddleL.x, paddleL.y, paddleL.w, paddleL.h) -- draw left paddle
    love.graphics.rectangle('fill', paddleR.x, paddleR.y, paddleR.w, paddleR.h) -- draw right paddle
    love.graphics.rectangle('fill', ball.x, ball.y, ball.w, ball.h)             -- draw ball
end
```

Pong game version 2

Copy the folder “Pong v1” and rename the copy “Pong v2”
Open Pong v2 in ZeroBrane.
It will look exactly the same as Pong v1 (You did copy it after all)

There are some changes or additional lines as follows:

love.load()

delete :

```
ball.speed = 5
```

add:

```
ball.speedx = math.random(-5, 5)
ball.speedy = math.random(-2, 2)
```

These 2 lines will allow the ball to move vertically as well as horizontally, and will give it random values at the start. Negative numbers make it move the opposite direction

love.update(dt)

Simplify the lines checking for which key is pressed by substituting the following:

```
-- w and s keys control left paddle
if love.keyboard.isDown('w') then -- move left paddle up
    paddleL.y = math.max(0, paddleL.y - paddleL.speed)
elseif love.keyboard.isDown('s') then -- move left paddle down
    paddleL.y = math.min(HEIGHT - paddleL.h, paddleL.y + paddleL.speed)
end

-- up and down keys control right paddle
if love.keyboard.isDown('up') then -- move right paddle up
    paddleR.y = math.max(0, paddleR.y - paddleR.speed)
elseif love.keyboard.isDown('down') then -- move right paddle down
    paddleR.y = math.min(HEIGHT - paddleR.h, paddleR.y + paddleR.speed)
end
```

This uses the math.min() and math.max() functions as appropriate.
For example paddleL.y is set to either 0 or the calculated y value, math.max choosing the largest value.
These one-liners remove an if statement each time.

Next deal with the ball position by using:

```
ball.x = ball.x + ball.speedx
ball.y = ball.y + ball.speedy
if ball.x < 0 or ball.x > WIDTH then -- ball at screen edge, put it back in the centre
    ball.x = WIDTH / 2 - ball.w / 2
end
if ball.y < 0 or ball.y > HEIGHT - ball.h then
    ball.speedy = ball.speedy * -1
end
```

This adjusts the ball position both horizontally and vertically

Finally substiute the collision checking lines with:

```
if collides(ball, paddleR) then         -- has the ball hit the right paddle?
    ball.speedx = ball.speedx * -1      -- reverse the direction right -> left
    -- alter the vertical speed between 1 and 2 randomly in the same direction
    if ball.speedy < 0 then
        ball.speedy = -math.random(-2, 2)
    else
        ball.speedy = math.random(-2, 2)
    end
elseif  collides(ball, paddleL) then    -- has the ball hit the right paddle?
    ball.speedx = math.abs(ball.speedx) -- reverse the direction left -> right
    if ball.speedy < 0 then
        ball.speedy = -math.random(-2, 2)
    else
        ball.speedy = math.random(-2, 2)
    end
end
```

The game now runs more conventionally, with the ball able to move at an angle when hit with a paddle, and bounces at the top and bottom of the screen.

You could now make some improvements to the game:

1. Keep a score and display it using love.graphics.print(text, x, y)
2. Change the colour of the paddles and ball by setting the colour with love.graphics.setColor() before each rectangle is drawn

The full code for Pong v2 is shown below:

```
WIDTH = love.graphics.getWidth()
HEIGHT = love.graphics.getHeight()

function collides(rect1, rect2)
    --[[ check whether rectangles are NOT colliding ]]

    -- if left side of rect1 is beyond rect2 right side
    -- OR left side of rect2 is beyond rect1 right side
    if rect1.x > rect2.x + rect2.w or rect2.x > rect1.x + rect1.w then
        return false
    end
    -- if top side of rect1 is beyond bottom of rect2
    -- OR top side of rect2 is beyond bottom of rect1
    if rect1.y > rect2.y + rect2.h or rect2.y > rect1.y + rect1.h then
        return false
    end

    -- code only gets this far if both if statements above fail
    return true
end

function love.load()
    -- set variables for left paddle in a table
    --[[
    ------------------------------------------------------------
    |         tables of variables for paddles and ball         |
    ------------------------------------------------------------
    |   name  |x position | y position | width | height | speed |
    ------------------------------------------------------------
    | paddleL |     10    |     260    |   15  |   80   |   10  |
    ------------------------------------------------------------
    | paddleR | WIDTH - 25|     260    |   15  |   80   |   10  |
    ------------------------------------------------------------
    |   ball  | WIDTH/2-5 | HEIGHT/2-5 |   10  |   10   |    5  |
    ------------------------------------------------------------
    ]]
    paddleL = {}
    paddleL.x = 10          -- start 10 pixels in from left edge
    paddleL.y = 260         -- start 260 pixels down from top
    paddleL.w = 15          -- paddle is 15 pixels wide
    paddleL.h = 80          -- paddle is 80 pixels tall
    paddleL.speed = 10

    -- set variables for right paddle in a table
    paddleR = {}
    paddleR.x = WIDTH - 25  -- start 10 pixels from the right edge (10 + width of paddle)
    paddleR.y = 260
    paddleR.w = 15
    paddleR.h = 80
    paddleR.speed = 10

    -- set variables for ball in a table
    ball = {}
    ball.x = WIDTH / 2 - 5  -- start in centre of the screen (half of WIDTH minus ball width)
    ball.y = HEIGHT / 2 - 5 -- start in centre of the screen (half of HEIGHT minus ball height)
    ball.w = 10
    ball.h = 10
    ball.speedx = math.random(-5, 5)
    ball.speedy = math.random(-2, 2)
end
```

```
function love.update(dt)
    -- w and s keys control left paddle
    if love.keyboard.isDown('w') then -- move left paddle up
        paddleL.y = math.max(0, paddleL.y - paddleL.speed)
    elseif love.keyboard.isDown('s') then -- move left paddle down
        paddleL.y = math.min(HEIGHT - paddleL.h, paddleL.y + paddleL.speed)
    end

    -- up and down keys control right paddle
    if love.keyboard.isDown('up') then -- move right paddle up
        paddleR.y = math.max(0, paddleR.y - paddleR.speed)
    elseif love.keyboard.isDown('down') then -- move right paddle down
        paddleR.y = math.min(HEIGHT - paddleR.h, paddleR.y + paddleR.speed)
    end

    ball.x = ball.x + ball.speedx
    ball.y = ball.y + ball.speedy
    if ball.x < 0 or ball.x > WIDTH then -- if ball has hit the edge of the screen, put it back in the centre
        ball.x = WIDTH / 2 - ball.w / 2
    end
    if ball.y < 0 or ball.y > HEIGHT - ball.h then
        ball.speedy = ball.speedy * -1
    end

    if collides(ball, paddleR) then         -- has the ball hit the right paddle?
        ball.speedx = ball.speedx * -1      -- reverse the direction right -> left
        -- alter the vertical speed between 1 and 2 randomly in the same direction
        if ball.speedy < 0 then
            ball.speedy = -math.random(-2, 2)
        else
            ball.speedy = math.random(-2, 2)
        end
    elseif  collides(ball, paddleL) then    -- has the ball hit the right paddle?
        ball.speedx = math.abs(ball.speedx) -- reverse the direction left -> right
        if ball.speedy < 0 then
            ball.speedy = -math.random(-2, 2)
        else
            ball.speedy = math.random(-2, 2)
        end
    end
end

function love.draw()
    love.graphics.clear(0.18, 0.16, 0.2)    -- clear screen with a dark grey colour
    love.graphics.rectangle('fill', paddleL.x, paddleL.y, paddleL.w, paddleL.h) -- draw left paddle
    love.graphics.rectangle('fill', paddleR.x, paddleR.y, paddleR.w, paddleR.h) -- draw right paddle
    love.graphics.rectangle('fill', ball.x, ball.y, ball.w, ball.h)             -- draw ball
end
```