In this tutorial we'll set the camera to follow our player sprite, setup our player sprite's animations, and use physics to create obstacles.

Before beginning I suggest you make a copy of your project from the second tutorial and save it for future reference.

1) Run the Corona Terminal. This will start the Corona Simulator and also open a terminal window where error messages and print statements will display.

2) Making sure the Corona Simulator icon is selected, click File -> Open…

3) Navigate to and select your project folder. Click Open.

4) Open your IDE of choice. I will be using the free TextWrangler code editor for these tutorials.

5) Open your project in your IDE. Select the main.lua file to work with. Your main.lua should contain the following:

```
---------------------------------------------------------------------------------------
--
-- main.lua
--
---------------------------------------------------------------------------------------

-- Your code here

local mte = require("mte").createMTE()

mte.loadMap("map.tmx")

mte.setCamera({locX = 1, locY = 1})

mte.constrainCamera({})

local spriteSheet = graphics.newImageSheet("spriteSheet.png", {width = 32, height = 32,
numFrames = 96})
local sequenceData = {
	{name = "up", sheet = spriteSheet, frames = {85, 86}, time = 400, loopCount = 0},
	{name = "right", sheet = spriteSheet, frames = {73, 74}, time = 400, loopCount = 0},
	{name = "down", sheet = spriteSheet, frames = {49, 50}, time = 400, loopCount = 0},
	{name = "left", sheet = spriteSheet, frames = {61, 62}, time = 400, loopCount = 0}
}
local player = display.newSprite(spriteSheet, sequenceData)
```

```
local setup = {locX = 10, locY = 8, layer = mte.getSpriteLayer(1)}
mte.addSprite(player, setup)

local Dpad = display.newImageRect("Dpad.png", 100, 100)
Dpad.x = display.screenOriginX + 50
Dpad.y = display.contentHeight - display.screenOriginY - 50

local velX, velY = 0, 0
local move = function(event)
	if event.phase == "began" or event.phase == "moved" then
		local touchX = event.x - event.target.x
		local touchY = event.y - event.target.y

		if touchX < -15 then
			velX = -5
		elseif touchX > 15 then
			velX = 5
		else
			velX = 0
		end

		if touchY < -15 then
			velY = -5
		elseif touchY > 15 then
			velY = 5
		else
			velY = 0
		end
	elseif event.phase == "ended" then
		velX, velY = 0, 0
	end
end

Dpad:addEventListener("touch", move)

local gameLoop = function(event)
	mte.moveSprite(player, velX, velY)
	mte.update()
	mte.debug()
end

Runtime:addEventListener("enterFrame", gameLoop)
```

6) The easiest of our jobs is to set the camera to follow the sprite. Immediately after `mte.addSprite(player, setup)` add the following line of code:

`mte.setCameraFocus(player)`

7) Save your project. Relaunch the app. The camera will now follow the player sprite around the map.

Animating a sprite is accomplished through native Corona SDK API calls. I suggest checking out Corona Labs' documentation pertaining to sprites if you haven't already. We will be changing the animations of our sprite based on its movement direction, so we'll have to make a few changes to the move function we created in tutorial 2.

8) Just below `local velX, velY = 0, 0` add `local sequence = "up"`

9) Add `sequence = "left"` within `if touchX < -15 then`

10) Add `sequence = "right"` within `if touchX > 15 then`

11) Add `sequence = "up"` within `if touchY < -15 then`

12) Add `sequence = "down"` within `if touchY > 15 then`

13) Beneath those if-then-else blocks, add the following code:

```
if player.sequence ~= sequence then
	player:setSequence(sequence)
end

player:play()
```

14) Add `player:pause()` within `elseif event.phase == “ended” then`

Your move function code should now look like this:

```
local velX, velY = 0, 0
local sequence = "up"
local move = function(event)
	if event.phase == "began" or event.phase == "moved" then
```

```
                local touchX = event.x - event.target.x
                local touchY = event.y - event.target.y

                if touchX < -15 then
                        velX = -5
                        sequence = "left"
                elseif touchX > 15 then
                        velX = 5
                        sequence = "right"
                else
                        velX = 0
                end

                if touchY < -15 then
                        velY = -5
                        sequence = "up"
                elseif touchY > 15 then
                        velY = 5
                        sequence = "down"
                else
                        velY = 0
                end

                if player.sequence ~= sequence then
                        player:setSequence(sequence)
                end

                player:play()
        elseif event.phase == "ended" then
                velX, velY = 0, 0
                player:pause()
        end
end

Dpad:addEventListener("touch", move)
```

15) Save your project. Relaunch the app and move the sprite around with the Dpad to observe the animations. The above code is just one possible way to animate a sprite. How you go about doing it will depend a great deal on your gameplay mechanics and the visual style you're aiming for.

Next we'll add collision detection using Corona's native Physics API.

16) Add the following code just before `mte.loadMap("map.tmx")`:

```
mte.enableBox2DPhysics()
physics.start()
physics.setGravity(0, 0)
physics.setDrawMode("hybrid")
```

The first function call prepares MTE to load physics properties from the map. The next three lines are native Corona API calls. The first starts the physics engine, the second gets rid of gravity- our top down perspective doesn't need it. The third line is purely for demonstration and debugging purposes; it draws physics object shapes on the screen so you can see where they are and what they're doing.

17) Save your project. Relaunch the app. Nothing has change; we can't see any physics objects because we haven't added any.

18) Add the following line of code immediately after `mte.setCameraFocus(player)`:

```
physics.addBody(player, "dynamic", {bounce = 0.0})
```

19) Save your project. Relaunch the app. The player is now outlined by a yellow square. This is the physics body we just added to the player. This outline is only visible because we set physics.setDrawMode("hybrid") above.

At the moment the player has nothing to collide with. We'll add physics objects to the bottom two tiles of our map's trees in Tiled.

20) In your project folder, double click map.tmx to open the map in Tiled.

21) In the Tilesets pane, scroll to the right side of the tileset, right click the lower left tile of the large tree and select Tile Properties…

22) Double-click <new property> and enter "physics" without the quotes.

23) Double-click the empty space in the Value column next to the property you just created. Enter "true" without the quotes.

24) Click OK.

25) Repeat steps 21 through 24 for the lower right tile of the large tree.

26) Save the map and close Tiled.

27) Relaunch the app. Move the sprite into one of the trees. The effect doesn't appear quite right; the sprite pushes a fair distance into the obstacle and the sprite tends to spin around. This effect might be desireable in some games, but for this tutorial we want the player to remain upright and collisions to be nice and tight.

28) Add the following line of code immediately beneath `physics.addBody(player, "dynamic", {bounce = 0.0})`:

```
player.isFixedRotation = true
```

This will prevent the player sprite from rotating.

29) The mushy collisions are a side-effect of using mte.moveSprite() to move our sprite. MTE first moves the sprite, and then the physics object responds. This slight delay allows for the sprite to push a ways into the obstacle. Delete `mte.moveSprite(player, velX, velY)` and replace it with the following line:

```
player:setLinearVelocity(velX * 30, velY * 30)
```

Linear velocity is set in pixels per second. Previously we were moving our sprite 5 pixels per frame. There are 30 frames per second in this app as it is currently configured. Therefore we want to multiply our linear velocity by 30 to achieve the same velocity as before.

30) Save your project and relaunch the app. Move the player against a tree. You'll see that the player collisions are nice and precise, however they could use some tweaking.

31) The player currently stops a significant distance from the sides of the trees. We can adjust this in Tiled. Double click map.tmx to open it in Tiled.

32) Right click the lower left tile in the large tree near the right side of the tileset. Click Tile Properties…

33) Double click <new property> and enter "shape" without the quotes.

34) Double click in the value column next to your new property and enter "[-8, -16, 16, -16, 16, 16, -8, 16]" without the quotes. You're defining the shape of the physics body by defining its four corners (see Corona SDK documentation for details). Click OK.

35) Repeat the process for the lower right tile of the tree, but this time enter "[-16, -16, 8, -16, 8, 16, -16, 16]"

36) Save the map and exit Tiled.

37) Relaunch the app. Note that the physics bodies of the trees are now narrower, allowing more clearance.

38) Find `physics.addBody(player, "dynamic", {bounce = 0.0})` in main.lua of your project. Delete it and replace it with the following:

```
local s = {-8, -16, 8, -16, 8, 16, -8, 16}
physics.addBody(player, "dynamic", {bounce = 0.0, shape = s})
```

39) Comment out physics.setDrawMode("hybrid"), save, and relaunch the app. We now have a simple map with obstacles and a player sprite which moves and animates based on user input. In future turorials we will expand on this simple app by adding enemies and using Tiled objects to trigger events.