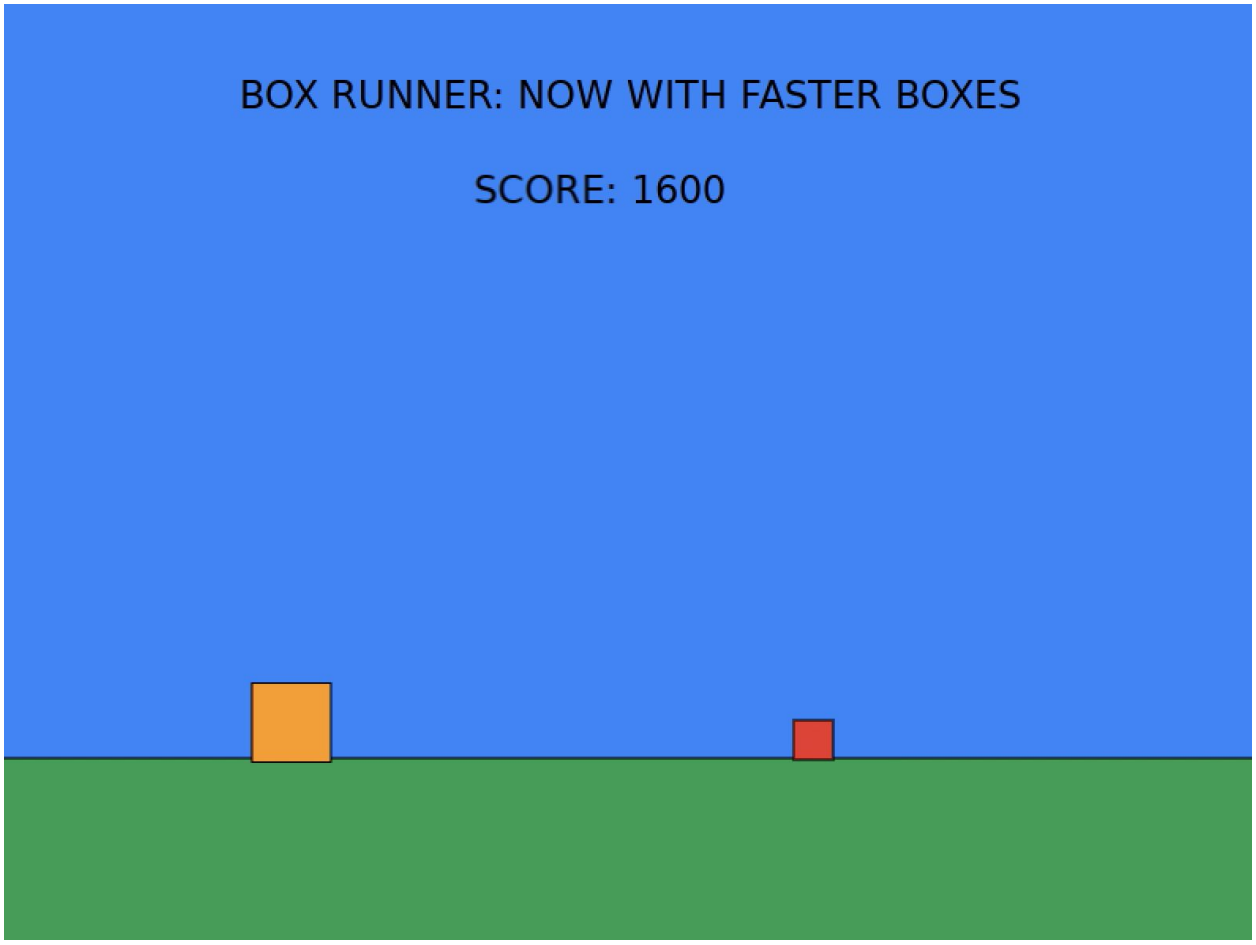
Hands-on LÖVE Quickstart

Complete the steps described in the Tutorial section to create Box Runner: a simple 2D LÖVE game in the Lua programming language.

See https://github.com/AndrewDiMola/box-runner for an animated demo, license information, and the complete source code.



LÖVE overview

LÖVE is a framework for developing 2D games with Lua on Windows, Mac OS X, Linux, Android and iOS.

Prerequisites

To run Box Runner, you'll need:

- LÖVE 11.3
- Windows or Mac OS X

Box Runner tutorial

Complete the steps described in this section to create Box Runner: a simple 2D LÖVE game in the Lua programming language.

In Box Runner, a yellow box represents the player and a red box represents the enemy. Player and enemy information is stored in the `player` and `enemy` objects, and game information, like the title of the game and the current score, is stored in a `game` object. RGB values for the rectangle objects that represent the player, enemy, and game world are stored in a `colors` object.

When the Spacebar is pressed, the yellow player box jumps. If the yellow player box collides with the red enemy box, the game is over, at least until the player presses the Spacebar to play again.

Score is based on the time it takes to collide with the enemy box, and the speed of the red enemy box increases over time.

Step 1: Initialize game data

Create a file named `main.lua` in a folder named `box runner` and copy in the following code:

```
player = {}
enemy = {}
game = {}
colors = {}
```

Next, create a new function, `love.load()`:

```
function love.load()
    game.SCREEN_WIDTH, game.SCREEN_HEIGHT = love.graphics.getDimensions()
    game.FONT = love.graphics.newFont(24)
    game.WINDOW_TITLE = "Hello Platforming World!"
```

```
    game.TITLE = "BOX RUNNER: NOW WITH FASTER BOXES"
    game.RETRY = "GAME OVER: Press Space to play again"
    game.SCOREBOARD = "SCORE:  "
    game.score = 0
    game.is_active = true

    colors.DEFAULT = {255, 255, 255}
    colors.TEXT = {0, 0, 0}
    colors.BACKGROUND = {66 / 255, 133 / 255, 244 / 255}
    colors.GROUND = {15 / 255, 157 / 255, 88 / 255}
    colors.SHAPE_OUTLINE = {0, 0, 0}
    colors.PLAYER = {244 / 255, 160 / 255, 0 / 255}
    colors.ENEMY = {219 / 255, 68 / 255, 55 / 255}

    player.x = game.SCREEN_WIDTH - (game.SCREEN_WIDTH / 1.25)
    player.y = (game.SCREEN_HEIGHT / 1.25) + 1
    player.START_Y = player.y
    player.WIDTH = 50
    player.HEIGHT = 50
    player.JUMP_HEIGHT = player.y - (2 * player.HEIGHT)
    player.GRAVITY = player.JUMP_HEIGHT / 2
    player.is_jump = false

    enemy.WIDTH = player.WIDTH / 2
    enemy.HEIGHT = player.HEIGHT / 2
    enemy.x = game.SCREEN_WIDTH + enemy.WIDTH
    enemy.y = (game.SCREEN_HEIGHT / 1.25) + 1
    enemy.START_X = enemy.x
    enemy.START_SPEED = 5
    enemy.speed = enemy.START_SPEED
    enemy.is_collide = false

    love.window.setTitle(game.WINDOW_TITLE)
    love.graphics.setFont(game.FONT)
end
```

Step 2: Draw the player, the enemy, and the game world

In a new function, `love.draw()`, copy in the following code:

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

```
    love.graphics.setColor(colors.DEFAULT)
    love.graphics.setBackgroundColor(colors.BACKGROUND)

    DrawGameText()
    DrawGround(0, game.SCREEN_HEIGHT / 1.25, game.SCREEN_WIDTH,
game.SCREEN_HEIGHT)
    DrawPlayer(player.x, player.y - player.HEIGHT, player.WIDTH,
player.HEIGHT)
    DrawEnemy(enemy.x, enemy.y - enemy.HEIGHT, enemy.WIDTH, enemy.HEIGHT)
end
```

Next, create three new functions: `DrawGround()`, `DrawPlayer()`, and `DrawEnemy()`.

```
function DrawGround(x, y, width, height)
    love.graphics.setColor(colors.GROUND)
    love.graphics.rectangle("fill", x, y, width, height)
    love.graphics.setColor(colors.SHAPE_OUTLINE)
    love.graphics.rectangle("line", x, y, width, height)
end

function DrawPlayer(x, y, width, height)
    love.graphics.setColor(colors.PLAYER)
    love.graphics.rectangle("fill", x, y, width, height)
    love.graphics.setColor(colors.SHAPE_OUTLINE)
    love.graphics.rectangle("line", x, y, width, height)
end

function DrawEnemy(x, y, width, height)
    love.graphics.setColor(colors.ENEMY)
    love.graphics.rectangle("fill", x, y, width, height)
    love.graphics.setColor(colors.SHAPE_OUTLINE)
    love.graphics.rectangle("line", x, y, width, height)
end
```

Add another function, `DrawGameText()`, which renders all of the formatted text in the game, including the active score.

```
function DrawGameText()
    love.graphics.setColor(colors.TEXT)
    if (game.is_active) then
        love.graphics.printf(
```

```
            game.TITLE,
            0,
            (game.SCREEN_HEIGHT / 10) - (game.FONT:getHeight() / 2),
            game.SCREEN_WIDTH,
            "center"
        )
    else
        love.graphics.printf(
            game.RETRY,
            0,
            (game.SCREEN_HEIGHT / 10) - (game.FONT:getHeight() / 2),
            game.SCREEN_WIDTH,
            "center"
        )
    end
    love.graphics.printf(
        game.SCOREBOARD,
        0 - (game.FONT:getWidth(game.SCOREBOARD) / 2),
        (game.SCREEN_HEIGHT / 5) - (game.FONT:getHeight() / 2),
        game.SCREEN_WIDTH,
        "center"
    )
    love.graphics.printf(
        game.score,
        (0 - (game.FONT:getWidth(game.SCOREBOARD) / 2)) +
            ((game.FONT:getWidth(game.SCOREBOARD) / 2) +
(game.FONT:getWidth(game.score) / 2)),
        (game.SCREEN_HEIGHT / 5) - (game.FONT:getHeight() / 2),
        game.SCREEN_WIDTH,
        "center"
    )
end
```

Drag the `box runner` folder into `love.exe` / `love.app` to see the game world. For now, the red enemy box is rendered off-screen.

Step 3: Move the enemy across the X-axis

Create a new function for updating game state, `love.update()`:

```
function love.update(dt)
    if game.is_active then
```

```
        if enemy.x > 0 - enemy.WIDTH and not enemy.is_collide then
            enemy.x = enemy.x - enemy.speed
        elseif not enemy.is_collide then
            enemy.x = enemy.START_X
        end
    end
end
```

To see the red enemy box travel across the X-axis, drag the `box runner` folder into `love.exe` / `love.app`.

Step 4: Capture keyboard input

Create `love.keypressed()`, a new function for capturing keyboard input, and `Reset()`, a new function for resetting game state.

```
function love.keypressed(key)
    if game.is_active then
        if key == "space" and not player.is_jump then
            player.y = player.JUMP_HEIGHT
            player.is_jump = true
        end
    elseif key == "space" then
        Reset()
    end
end

function Reset()
    game.score = 0
    player.y = player.START_Y
    enemy.x = enemy.START_X
    enemy.is_collide = false
    enemy.speed = enemy.START_SPEED
    game.is_active = true
end
```

Next, in `love.update()`, in the `game.is_active` condition, implement the jumping mechanic, as demonstrated in the following code:

```
    if player.y < player.START_Y then
        player.y = player.y + (player.GRAVITY * dt)
```

```
    else
      player.is_jump = false
    end
```

Step 5: Check for collisions

In `love.update()`, add the following code for simple collision detection:

```
    if (player.x + player.WIDTH > enemy.x) and (player.x < enemy.x +
enemy.WIDTH) and (player.y > (enemy.y - enemy.HEIGHT))
    then
        enemy.is_collide = true
        game.is_active = false
    else
        game.score = game.score + 1
    end
```

Step 6: Increase the difficulty

Add these velocity properties in `love.load()`:

```
    game.VELOCITY_COUNTER_LOCAL_MAX = 3  -- Speed up every ~3 seconds
    game.velocity_counter = 0
```

Finally, implement the velocity mechanic into an all-new `love.update()`:

```
function love.update(dt)
    if game.is_active then
        game.velocity_counter = game.velocity_counter + dt
        if
            (player.x + player.WIDTH > enemy.x) and (player.x < enemy.x +
enemy.WIDTH) and
                (player.y > (enemy.y - enemy.HEIGHT))
         then
            enemy.is_collide = true
            game.is_active = false
        else
            game.score = game.score + 1
        end

        if player.y < player.START_Y then
```

```
            player.y = player.y + (player.GRAVITY * dt)
        else
            player.is_jump = false
        end

        if enemy.x > 0 - enemy.WIDTH and not enemy.is_collide then
            enemy.x = enemy.x - enemy.speed
        elseif not enemy.is_collide then
            enemy.x = enemy.START_X
        end

        if game.velocity_counter > game.VELOCITY_COUNTER_LOCAL_MAX then
            enemy.speed = enemy.speed + 1
            game.velocity_counter = 0
        end
    end
end
```

Drag the `box runner` folder into `love.exe` / `love.app` to play the final version of the game.

Reference

Box Runner uses four LÖVE callback functions, two LÖVE modules, and five custom functions.

Callbacks

The descriptions for the callback functions in this section are excerpted from the LÖVE documentation.

love.draw

Callback function used to draw on the screen every frame. Learn more

love.keypressed

Callback function triggered when a key is pressed. Learn more

love.load

Callback function called exactly once at the beginning of the game. Learn more

love.update

Callback function used to update the state of the game every frame. Learn more

Modules

The descriptions for the modules and functions in this section are excerpted from the LÖVE documentation.

love.graphics

Drawing of shapes and images, management of screen geometry. Learn more

getDimensions

Gets the width and height of the window. Learn more

newFont

Creates a new Font. Learn more

printf

Draws formatted text, with word wrap and alignment. Learn more

rectangle

Draws a rectangle. Learn more

setBackgroundColor

Sets the background color. Learn more

setColor

Sets the color used for drawing. Learn more

setFont

Set an already-loaded Font as the current font. Learn more

love.window

Provides an interface for the program's window. Learn more

setTitle

Sets the window title. Learn more

Custom functions

Box Runner uses custom functions to reset the state of the game and to modularize code with respect to `player` and `enemy` positions that are rendered in `love.draw()`.

DrawGround

Draws a green, filled-in rectangle that represents the ground on the screen every frame.

Arguments	Returns	Example usage

`number` x,
`number` y,
`number` width,
`number` height
	Nothing	`DrawGround(`
		`ground.x,`
		`ground.y -`
		`ground.HEIGHT,`
		`ground.WIDTH,`
		`ground.HEIGHT)`

DrawGameText

Draws formatted text that displays game information, such as `game.TITLE`, `game.RETRY`, `game.SCOREBOARD`, and `game.score`.

Arguments	Returns	Example usage
None	Nothing	`DrawGameText()`

DrawPlayer

Draws a yellow, filled-in rectangle that represents the player on the screen every frame.

Arguments	Returns	Example usage

`number` x,
`number` y,
`number` width,
`number` height
	Nothing	`DrawPlayer(`
		`player.x,`
		`player.y -`
		`player.HEIGHT,`
		`player.WIDTH,`
		`player.HEIGHT)`

DrawEnemy

Draws a red, filled-in rectangle that represents an enemy on the screen every frame.

Arguments	Returns	Example usage

`number` x,
`number` y,
`number` width,
`number` height
	Nothing	`DrawEnemy(`
		`enemy.x,`
		`enemy.y -`
		`enemy.HEIGHT,`
		`enemy.WIDTH,`

		`enemy.HEIGHT)`

Reset

Resets the initial values of dynamic game settings, such as `game.score`, `player.y`, `enemy.x`, `enemy.is_collide`, `enemy.speed`, and `game.is_active`.

Arguments	Returns	Example usage
None	Nothing	`Reset()`

Further reading

- LÖVE documentation
- Lua documentation

Troubleshooting

This section details common issues that developers encounter when programming in LÖVE with Lua. If you're stuck, try the suggested solutions.

Running Lua code with the LÖVE game engine

Check that your Lua code is in a file named `main.lua` and that you are dragging a folder *containing* `main.lua` into `love.exe` / `love.app`; you should not be dragging `main.lua` into the executable.

Installing LÖVE on MAC OS X

During installation, if you see a warning message about issues with app verification, see Apple's instructions on how to Safely open apps on your Mac.