Programming Fundamentals

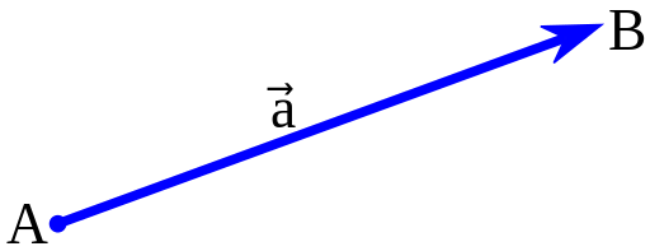
Lecture 06 – Vectors, Physics and Collision Detection

Edirlei Soares de Lima
<edirlei.lima@universidadeeuropeia.pt>

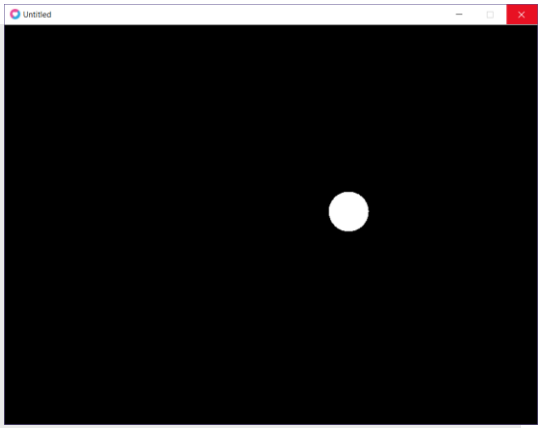
What is a Vector?

- The word vector can mean a lot of different things…
  - Biology/epidemiology: an organism that transmits infection from one host to another;
  - Computer science: a one-dimensional array;
  - **Mathematics and physics: an entity that has both magnitude and direction.**

- A vector is typically represented as a arrow, where the direction is indicated by where the arrow is pointing, and the magnitude by the length of the arrow itself.



Bouncing Ball Without Vectors



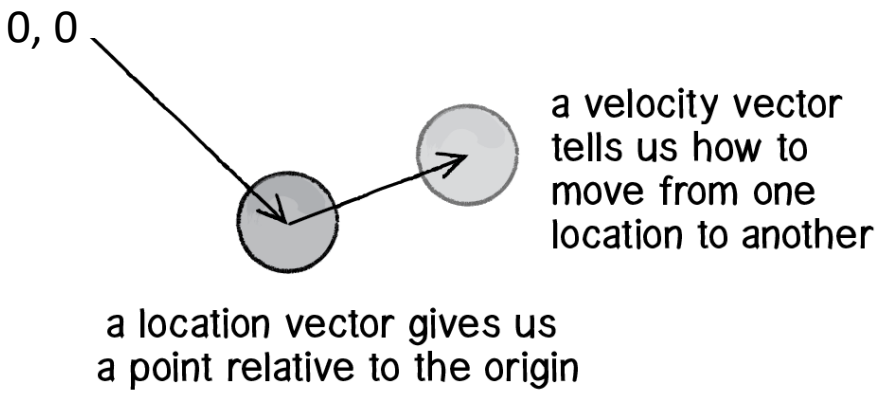

```
local x = 100
local y = 100
local xspeed = 300
local yspeed = 500
local radius = 30

function love.update(dt)
  x = x + (xspeed * dt)
  y = y + (yspeed * dt)

  if (x > love.graphics.getWidth() - radius) or (x < 0 + radius) then
    xspeed = xspeed * -1;
  end
  if (y > love.graphics.getHeight() - radius) or (y < 0 + radius) then
    yspeed = yspeed * -1;
  end
end

function love.draw()
   love.graphics.circle("fill", x, y, radius, 30)
end
```

Bouncing Ball Without Vectors

- In the code, the ball has some properties that are represented by variables:

Location	x and y
Velocity	yspeed and yspeed

- Both location and velocity can be described as vectors:



Defining a Vector

- To better represent vectors in Lua, we can create a table to store both x and y:

vector2.lua

```
vector2 = {}

function vector2.new(px, py)
  return {x = px, y = py}
end
```

- We can save it in a separated module (file) and handle it as a new data type.

main.lua | vector2.lua

```
1   require "vector2"
2
3   local position = vector2.new(100, 100)
4   local velocity = vector2.new(300, 500)
5   local radius = 30
6
```

main.lua | vector2.lua

```
1   vector2 = {}
2
3   function vector2.new(px, py)
4     return {x = px, y = py}
5   end
6
```

What is a Table?

- A Lua table is defined by a set of **pairs key-data**, where the data is referenced by the key.
  - The key (index) can be of any type of data (number, text, …);
  - The data can also be of any type of data (number, text, function, image);
- With Lua tables we can associate names to its elements and create structures to store related data:

```
player1 = {
  name   = "John",
  score  = 1000,
  lives  = 3
}
```

- We can access an element of a table by its name:

```
io.write(player1.score)
```

Vector Addition

- Now that we have two vector objects (location and velocity), we are ready to implement the algorithm for motion:
  - position = position + velocity
  - Code without vectors:

```
x = x + (xspeed * dt)
y = y + (yspeed * dt)
```

- However, we are not allowed to do that directly, we need a function to add two vectors:

$$\vec{w} = \vec{u} + \vec{v} = \begin{matrix} w_x = u_x + v_x \\ w_y = u_y + v_y \end{matrix}$$

vector2.lua

```
function vector2.add(vec, inc)
  local result = vector2.new(0, 0)
  result.x = vec.x + inc.x
  result.y = vec.y + inc.y
  return result
end
```

Bouncing Ball With Vectors

main.lua

```
require "vector2"

local position = vector2.new(100, 100)
local velocity = vector2.new(300, 500)
local radius = 30

function love.update(dt)
  position = vector2.add(position, vector2.mult(velocity, dt))

  if (position.x > love.graphics.getWidth() - radius) or
     (position.x < 0 + radius) then
    velocity.x = velocity.x * -1;
  end
  if (position.y > love.graphics.getHeight() - radius) or
     (position.y < 0 + radius) then
    velocity.y = velocity.y * -1;
  end
end

function love.draw()
   love.graphics.circle("fill", position.x, position.y, radius, 30)
end
```

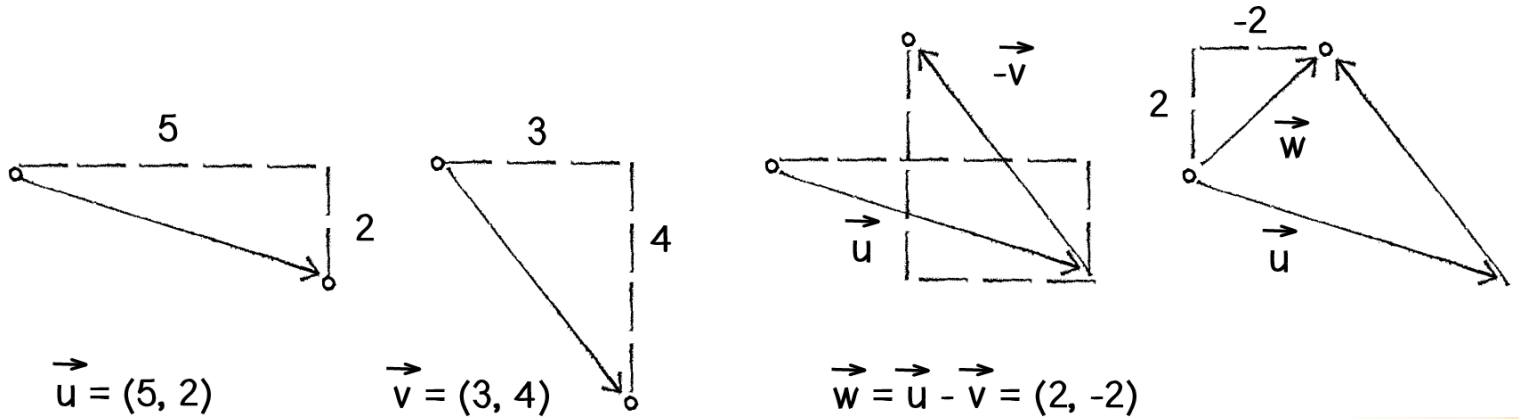
Vector Subtraction

$$\vec{w} = \vec{u} - \vec{v} = \begin{matrix} w_x = u_x - v_x \\ w_y = u_y - v_y \end{matrix}$$

- Implementation:

vector2.lua

```
function vector2.sub(vec, dec)
  local result = vector2.new(0, 0)
  result.x = vec.x - dec.x
  result.y = vec.y - dec.y
  return result
end
```



Vector Multiplication and Division

$$\overrightarrow{w} = \overrightarrow{u} * n = \begin{matrix} w_x = u_x * n \\ w_y = u_y * n \end{matrix}$$

- Multiplication implementation:

vector2.lua

```
function vector2.mult(vec, n)
  local result = vector2.new(0, 0)
  result.x = vec.x * n
  result.y = vec.y * n
  return result
end
```

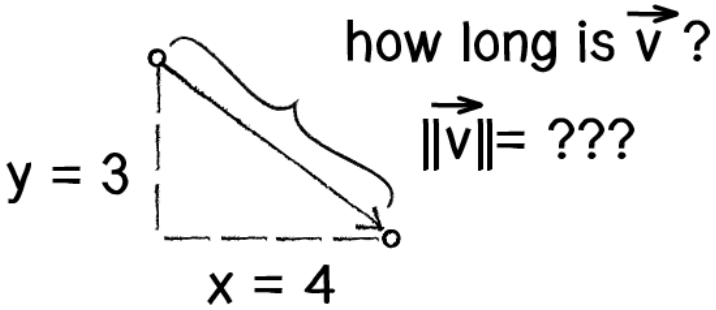
- Division implementation:

vector2.lua

```
function vector2.div(vec, n)
  local result = vector2.new(0, 0)
  result.x = vec.x / n
  result.y = vec.y / n
  return result
end
```

Vector Magnitude

$$\left\| \vec{v} \right\| = \sqrt{v_x * v_x + v_y * v_y}$$



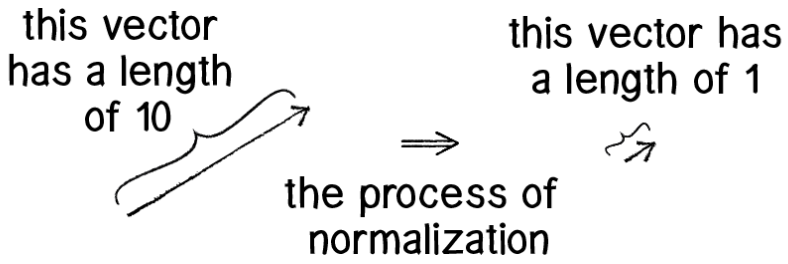
- Implementation:

vector2.lua

```
function vector2.magnitude(vec)
  return math.sqrt(vec.x * vec.x + vec.y * vec.y)
end
```

Vector Normalization

$$\hat{u} = \frac{\vec{u}}{\left\| \vec{u} \right\|}$$



- Implementation:

vector2.lua

```
function vector2.normalize(vec)
 local m = vector2.magnitude(vec)
 if m ~= 0 then
   return vector2.div(vec, m)
 end
 return vec
end
```

Vector Motion: Acceleration

- **Basic definitions:**
  - Velocity: the rate of change of location;
  - Acceleration: the rate of change of velocity;
- Acceleration affects velocity, which in turn affects location.

```
velocity = vector2.add(velocity, vector2.mult(acceleration, dt))
position = vector2.add(position, vector2.mult(velocity, dt))
```

- Acceleration values accumulate over time in the velocity, therefore, a function to limit the velocity is required:

vector2.lua

```
function vector2.limit(vec, max)
  local result = vec
  if (vector2.magnitude(vec) > max) then
    result = vector2.normalize(vec)
    result = vector2.mult(result, max)
  end
  return result
end
```

Example 1: Constant Acceleration

main.lua

```
require "vector2"

local position = vector2.new(400, 100)
local velocity = vector2.new(0, 0)
local radius = 30
local maxspeed = 500

function love.update(dt)
  local acceleration = vector2.new(-10,100)

  velocity = vector2.add(velocity, vector2.mult(acceleration, dt))
  velocity = vector2.limit(velocity, maxspeed)
  position = vector2.add(position, vector2.mult(velocity, dt))

  if (position.x > love.graphics.getWidth()) then
    position.x = 0
  elseif (position.x < 0) then
    position.x = love.graphics.getWidth()
  end

  ...
```

Example 1: Constant Acceleration

main.lua

```
  ...

  if (position.y > love.graphics.getHeight()) then
    position.y = 0
  elseif (position.y < 0) then
    position.y = love.graphics.getHeight()
  end
end

function love.draw()
   love.graphics.circle("fill", position.x, position.y, radius, 30)
end
```

Example 2: Acceleration Towards the Mouse

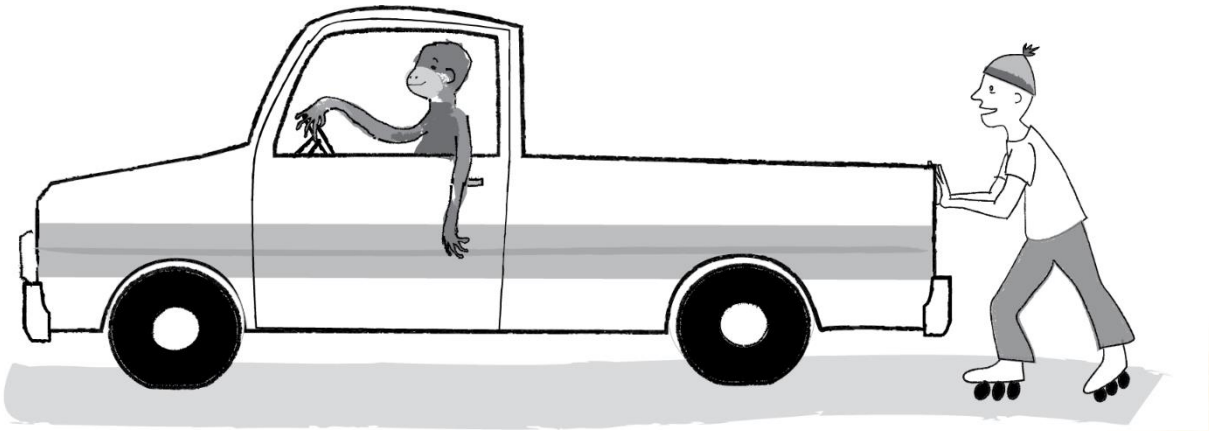
main.lua

```
require "vector2"

local position = vector2.new(400, 100)
local velocity = vector2.new(0, 0)
local radius = 30
local maxspeed = 250

function love.update(dt)
  local mouseposition = vector2.new(love.mouse.getX(),love.mouse.getY())
  local mousedirection = vector2.sub(mouseposition, position)
  mousedirection = vector2.normalize(mousedirection)

  local acceleration = vector2.mult(mousedirection, 700)

  velocity = vector2.add(velocity, vector2.mult(acceleration, dt))
  velocity = vector2.limit(velocity, maxspeed)
  position = vector2.add(position, vector2.mult(velocity, dt))
end

function love.draw()
  love.graphics.circle("fill", position.x, position.y, radius, 30)
end
```

Forces

- A force is a vector that causes an object with mass to accelerate.

- **Newton's Laws of Motion:**
  1. An object at rest stays at rest and an object in motion stays in motion.
  2. Force equals mass times acceleration.
  3. For every action there is an equal and opposite reaction.



Apply Force

- Implementation:

vector2.lua

```
function vector2.applyForce(force, mass, acceleration)
  local f = vector2.div(force, mass)
  return vector2.add(acceleration, f)
end
```

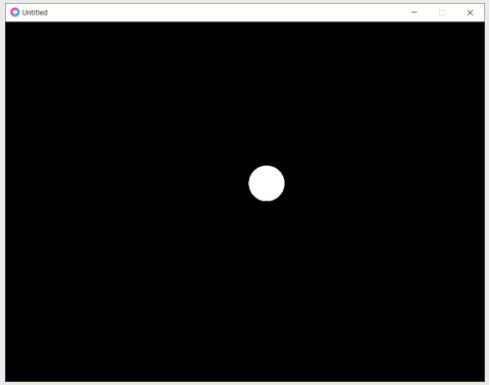
main.lua

```
...

local mass = 1

...
```

Example: Gravity and Wind Forces

main.lua

```
require "vector2"

local position = vector2.new(400, 100)
local velocity = vector2.new(0, 0)
local radius = 30
local maxspeed = 5
local mass = 1

local wind = vector2.new(100,0)
local gravity = vector2.new(0, 500)

function love.update(dt)

  local acceleration = vector2.new(0, 0)
  acceleration = vector2.applyForce(wind, mass, acceleration)
  acceleration = vector2.applyForce(gravity, mass, acceleration)

  velocity = vector2.add(velocity, vector2.mult(acceleration, dt))
  position = vector2.add(position, vector2.mult(velocity, dt))

  ...
```



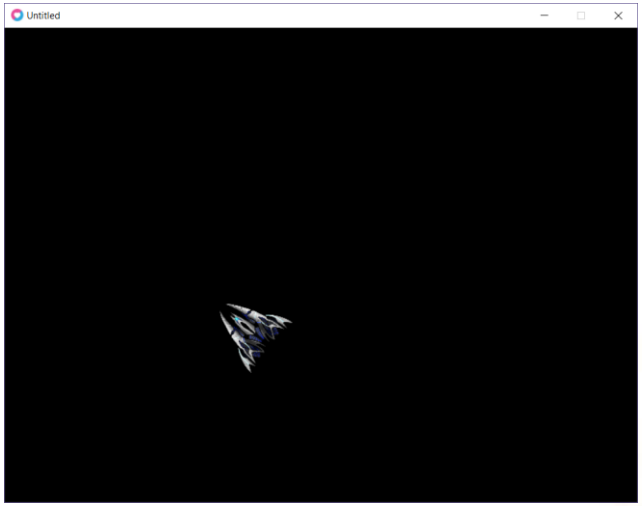
Example: Gravity and Wind Forces

main.lua

```
  ...

  if (position.x > love.graphics.getWidth() - radius) then
    velocity.x = velocity.x * -1
  elseif (position.x < radius) then
    velocity.x = velocity.x * -1
  end

  if (position.y > love.graphics.getHeight() - radius) then
    velocity.y = velocity.y * -1
  end
end

function love.draw()
   love.graphics.circle("fill", position.x, position.y, radius, 30)
end
```

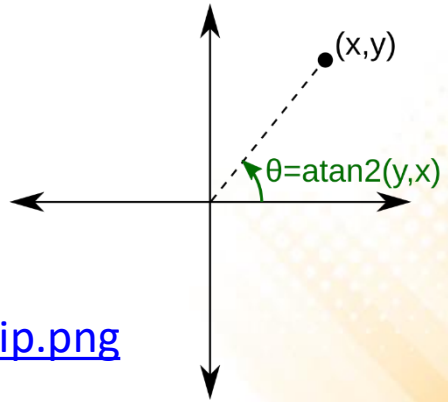
Exercise 1

1) Write the code to accelerate a spaceship towards the mouse position.



- You need to use the `applyForce` function to move the spaceship.
- Hint:
  - First, you need to calculate a vector that points from the spaceship to the mouse location (direction).

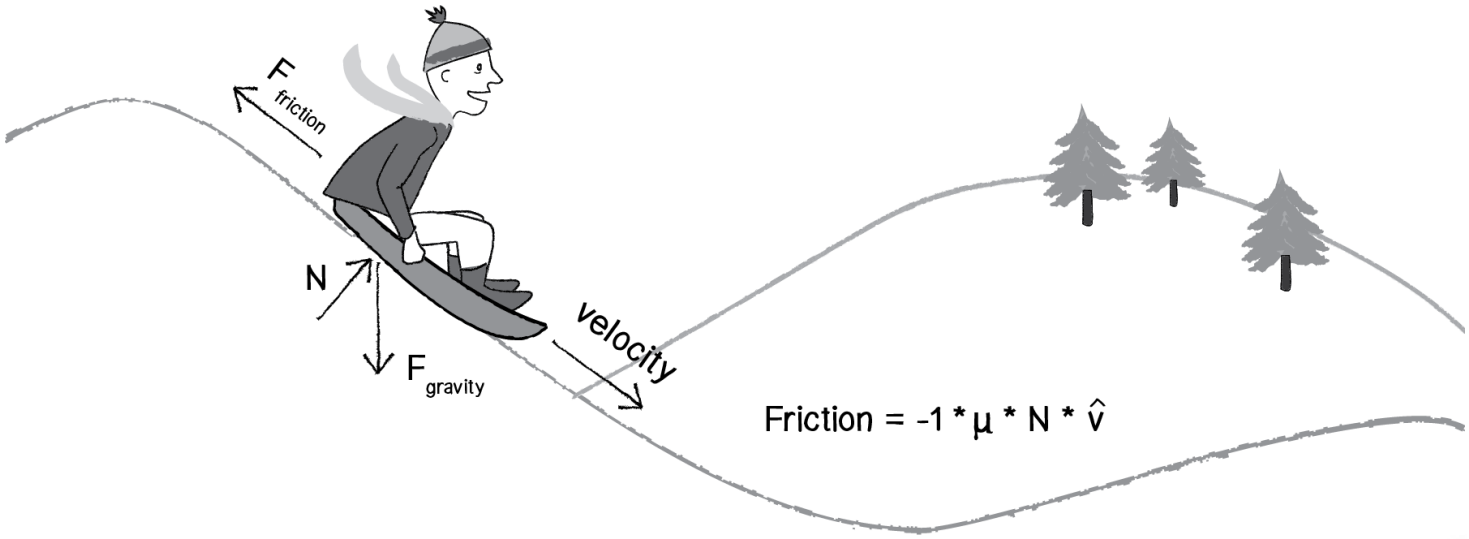
- The spaceship must rotate towards the direction of its movement.
- Hint:
  - Use the `math.atan2` function to calculate the arc tangent value from the y and x variables of the direction of the velocity.



Spaceship: http://www.inf.puc-rio.br/~elima/gameprog/spaceship.png

Friction

- Friction is a <u>dissipative force</u>, which is a type of force where the total energy of the system decreases when an object is in motion.



μ = coefficient of friction (strength of a friction force for a particular surface)
N = normal force (the force perpendicular to the object's motion along a surface)
$\hat{v}$ = velocity unit vector

Example: Friction Force

main.lua

```
require "vector2"

local position = vector2.new(400, 100)
local velocity = vector2.new(0, 0)
local radius = 30
local maxspeed = 5
local mass = 1

local frictioncoefficient = 100
local wind = vector2.new(100,0)
local gravity = vector2.new(0, 500)

function love.update(dt)
  local friction = vector2.mult(velocity, -1)
  friction = vector2.normalize(friction)
  friction = vector2.mult(friction, frictioncoefficient)

  local acceleration = vector2.new(0, 0)
  acceleration = vector2.applyForce(friction, mass, acceleration)
  acceleration = vector2.applyForce(wind, mass, acceleration)
  acceleration = vector2.applyForce(gravity, mass, acceleration)
  ...
```

Example: Friction Force

main.lua

```
  ...

  velocity = vector2.add(velocity, vector2.mult(acceleration, dt))
  position = vector2.add(position, vector2.mult(velocity, dt))

  if (position.x > love.graphics.getWidth() - radius) then
    position.x = love.graphics.getWidth() - radius
    velocity.x = velocity.x * -1
  elseif (position.x < radius) then
    position.x = radius
    velocity.x = velocity.x * -1
  end
  if (position.y > love.graphics.getHeight() - radius) then
    position.y = love.graphics.getHeight() - radius
    velocity.y = velocity.y * -1
  end
end

function love.draw()
   love.graphics.circle("fill", position.x, position.y, radius, 30)
end
```

Example: Player Movement and Jump

main.lua

```
require "vector2"

local position = vector2.new(400, 100)
local velocity = vector2.new(0, 0)
local playerwidth = 30
local playerheight = 60
local maxspeed = 400
local mass = 1
local frictioncoefficient = 300
local onGround = false

function love.update(dt)
  local acceleration = vector2.new(0, 0)
  local gravity = vector2.new(0, 500)
  acceleration = vector2.applyForce(gravity, mass, acceleration)

  if (onGround) then
    local friction = vector2.mult(velocity, -1)
    friction = vector2.normalize(friction)
    friction = vector2.mult(friction, frictioncoefficient)
    acceleration = vector2.applyForce(friction, mass, acceleration)
    ...
```

Example: Player Movement and Jump

main.lua

```
  ...

  if love.keyboard.isDown("right") then
    local move = vector2.new(500, 0)
    acceleration = vector2.applyForce(move, mass, acceleration)
  end
  if love.keyboard.isDown("left") then
    local move = vector2.new(-500, 0)
    acceleration = vector2.applyForce(move, mass, acceleration)
  end
  if love.keyboard.isDown("up") then
    local jump = vector2.new(0, -50000)
    acceleration = vector2.applyForce(jump, mass, acceleration)
    onGround = false
  end
end

velocity = vector2.add(velocity, vector2.mult(acceleration, dt))
velocity = vector2.limit(velocity, maxspeed)
position = vector2.add(position, vector2.mult(velocity, dt))

...
```

Example: Player Movement and Jump

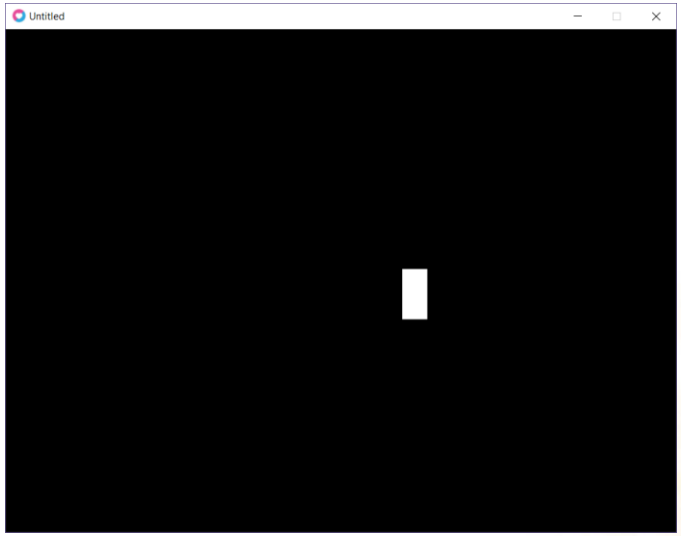
main.lua

```
  ...

  if (position.x > love.graphics.getWidth() - playerwidth) then
    position.x = love.graphics.getWidth() - playerwidth
    velocity.x = (velocity.x * -1) / 2
  elseif (position.x < 0) then
    position.x = 0
    velocity.x = (velocity.x * -1) / 2
  end

  if (position.y > love.graphics.getHeight() - playerheight) then
    position.y = love.graphics.getHeight() - playerheight
    velocity.y = 0
    onGround = true
  end

end

function love.draw()
   love.graphics.rectangle("fill", position.x, position.y,
                           playerwidth, playerheight)
end
```

Exercise 2

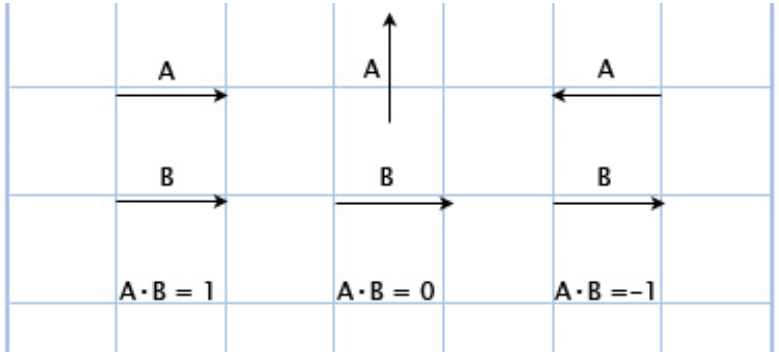
2) Modify the code of the player movement and jump example to allow the player to double jump.

- The player must be able to jump only two times.
- After double jumping, a new jump can only be performed when the player touches the ground.
- Hint:
  - Use the callback `love.keyreleased(key, scancode)` to detect when the player releases the jump key, which indicates that he/she can jump again (if it was the first jump).

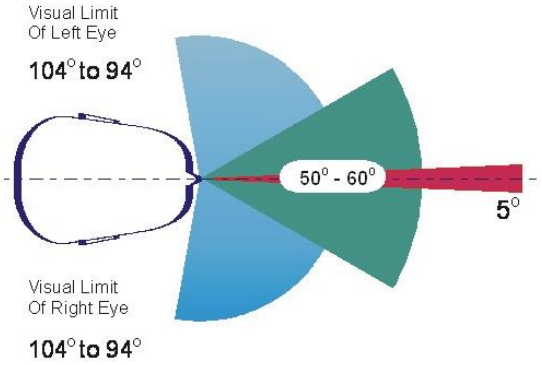


Dot Product

$$A \cdot B = (A_x, A_y) \cdot (B_x, B_y) = A_x\, B_x + A_y\, B_y$$



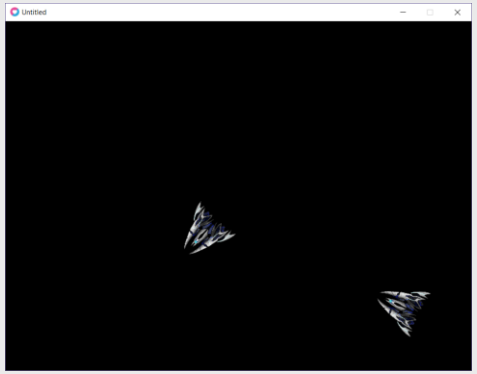
- The dot product has several applications in game programming.
  - Example: sight/view angle



Example: View Angle and Modularization

player.lua

```
require "vector2"

local playerposition = vector2.new(400, 300)
local playervelocity = vector2.new(0, 0)
local playerrotation = 0
local playermaxspeed = 100
local playermass = 1
local playerimage

function LoadPlayer()
  playerimage = love.graphics.newImage("spaceship.png")
end

function UpdatePlayer(dt)
  local acceleration = vector2.new(0, 0)
  local mouseposition = vector2.new(love.mouse.getX(),
                                    love.mouse.getY())
  local mousedirection = vector2.sub(mouseposition, playerposition)
  local velocitydirection = vector2.normalize(playervelocity)
  mousedirection = vector2.normalize(mousedirection)

  ...
```



Example: View Angle and Modularization

player.lua

```
  ...

  playerrotation = math.atan2(mousedirection.y, mousedirection.x)
  if (love.mouse.isDown(1)) then
    local engineForce = vector2.mult(mousedirection, 100)
    acceleration = vector2.applyForce(engineForce, playermass,
                                      acceleration)
  end
  playervelocity = vector2.add(playervelocity,
                               vector2.mult(acceleration, dt))
  playervelocity = vector2.limit(playervelocity, playermaxspeed)
  playerposition = vector2.add(playerposition,
                               vector2.mult(playervelocity, dt))
end
function DrawPlayer()
  love.graphics.draw(playerimage, playerposition.x, playerposition.y,
                playerrotation, 0.3, 0.3, playerimage:getWidth() / 2,
                playerimage:getHeight() / 2)
end
function GetPlayerPosition()
  return playerposition
end
```

Example: View Angle and Modularization

enemy.lua

```
require "vector2"

local enemyposition = vector2.new(700, 500)
local enemyvelocity = vector2.new(0, 0)
local enemydirection = vector2.new(-1, 0)
local enemyrotation = math.atan2(enemydirection.y, enemydirection.x)
local enemymaxspeed = 100
local enemymass = 1
local enemyviewangle = 30
local enemychasing = false
local enemyimage

function LoadEnemy()
  enemyimage = love.graphics.newImage("spaceship.png")
end

function UpdateEnemy(dt, pposition)
  local acceleration = vector2.new(0, 0)

  if (CanSee(pposition)) then
    enemychasing = true
  end
  ...
```

Example: View Angle and Modularization

player.lua

```
  ...
  if (enemychasing) then
    local pdirection = vector2.normalize(vector2.sub(pposition,
                                                     enemyposition))
    local engineForce = vector2.mult(pdirection, 100)
    acceleration = vector2.applyForce(engineForce, enemymass,
                                      acceleration)
    enemyrotation = math.atan2(enemyvelocity.y, enemyvelocity.x)
  end

  enemyvelocity = vector2.add(enemyvelocity,
                              vector2.mult(acceleration, dt))
  enemyvelocity = vector2.limit(enemyvelocity, enemymaxspeed)
  enemyposition = vector2.add(enemyposition,
                              vector2.mult(enemyvelocity, dt))
end

function DrawEnemy()
  love.graphics.draw(enemyimage, enemyposition.x, enemyposition.y,
                     enemyrotation, 0.3, 0.3, enemyimage:getWidth() / 2,
                     enemyimage:getHeight() / 2)
end
...
```

Example: View Angle and Modularization

player.lua

```
...

function CanSee(pposition)
  local pdirection = vector2.normalize(vector2.sub(pposition,
                                                   enemyposition))
  local angle = math.acos(vector2.dot(enemydirection, pdirection))
  if (math.deg(angle) < 30) then
    return true
  end
  return false
end
```

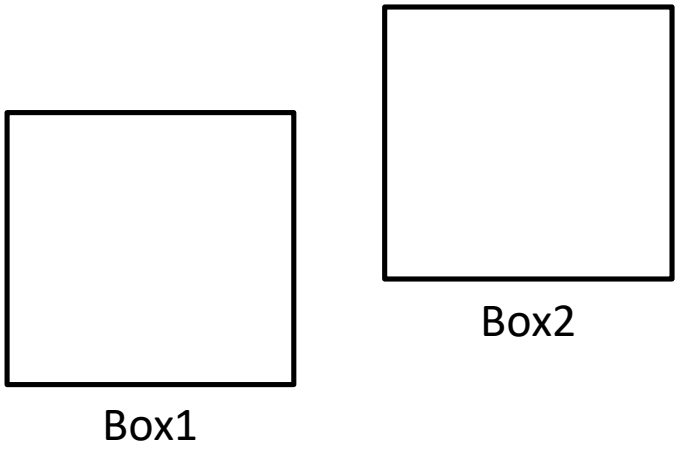
Example: View Angle and Modularization

main.lua

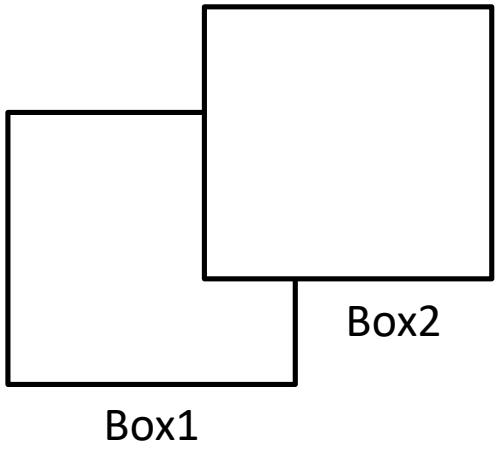
```
require "player"
require "enemy"

function love.load()
  LoadPlayer()
  LoadEnemy()
end

function love.update(dt)
  UpdatePlayer(dt)
  UpdateEnemy(dt, GetPlayerPosition())
end

function love.draw()
   DrawPlayer()
   DrawEnemy()
end
```

Collision Detection

No Collision

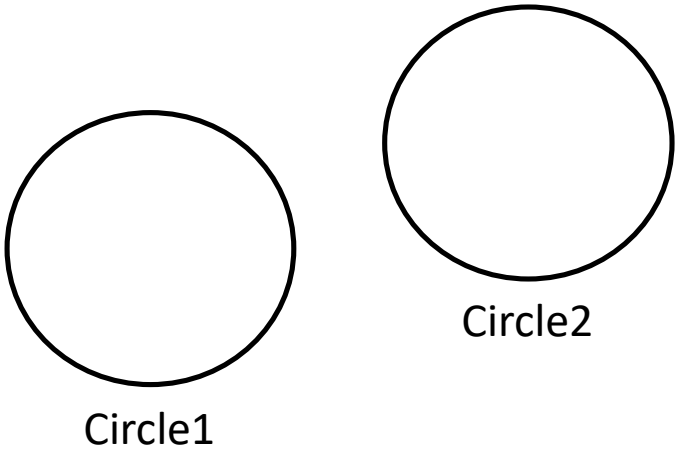


Collision

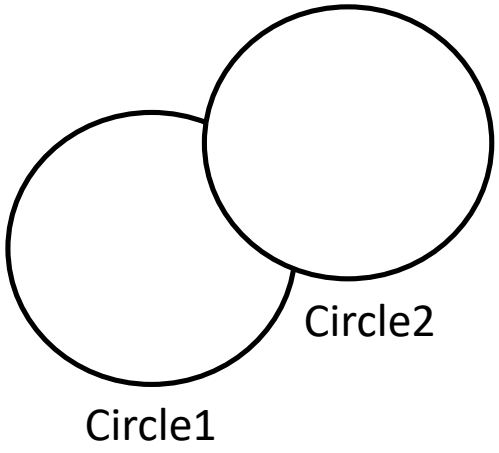


Collision Detection

No Collision



Collision

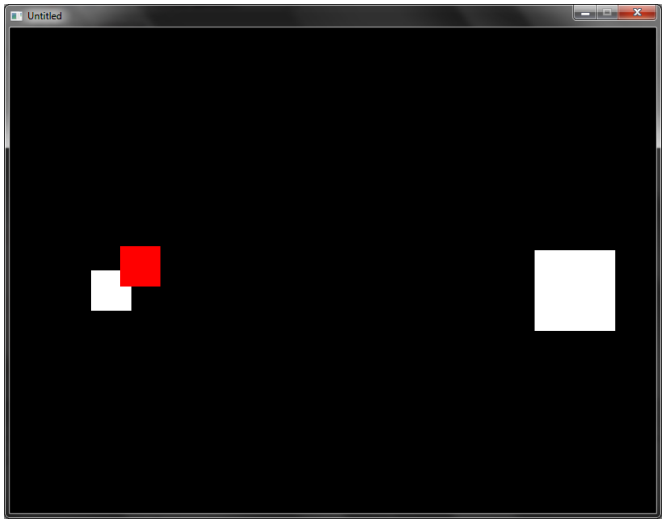


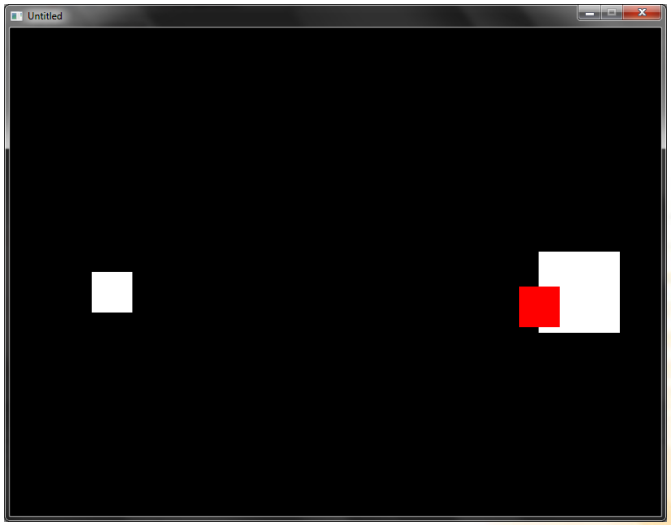
Collision Detection (Box)

```
function love.load()
  player1 = {
    x = 390,
    y = 300,
    width = 50,
    height = 50,
    collided = false
  }
  box1 = {
    x = 100,
    y = 300,
    width = 50,
    height = 50
  }
  box2 = {
    x = 650,
    y = 275,
    width = 100,
    height = 100
  }
end
```

```
function CheckBoxCollision(x1,y1,w1,h1,x2,y2,w2,h2)
  return x1 < x2+w2 and x2 < x1+w1 and y1 < y2+h2 and y2 < y1+h1
end

function love.update(dt)
  if love.keyboard.isDown("left") then
    player1.x = player1.x - (120 * dt)
  end
  if love.keyboard.isDown("right") then
    player1.x = player1.x + (120 * dt)
  end
  if love.keyboard.isDown("up") then
    player1.y = player1.y - (120 * dt)
  end
  if love.keyboard.isDown("down") then
    player1.y = player1.y + (120 * dt)
  end

  if CheckBoxCollision(player1.x, player1.y, player1.width,
           player1.height, box1.x, box1.y, box1.width, box1.height) or
     CheckBoxCollision(player1.x, player1.y, player1.width,
           player1.height, box2.x, box2.y, box2.width, box2.height) then
    player1.collided = true
  else
    player1.collided = false
  end
end
```

```
function love.draw()
  love.graphics.setColor(255,255,255)
  love.graphics.rectangle("fill", box1.x, box1.y,
                                  box1.width, box1.height)
  love.graphics.rectangle("fill", box2.x, box2.y,
                                  box2.width, box2.height)

  if player1.collided == true then
    love.graphics.setColor(255,0,0)
  end

  love.graphics.rectangle("fill", player1.x, player1.y,
                                  player1.width, player1.height)
end
```



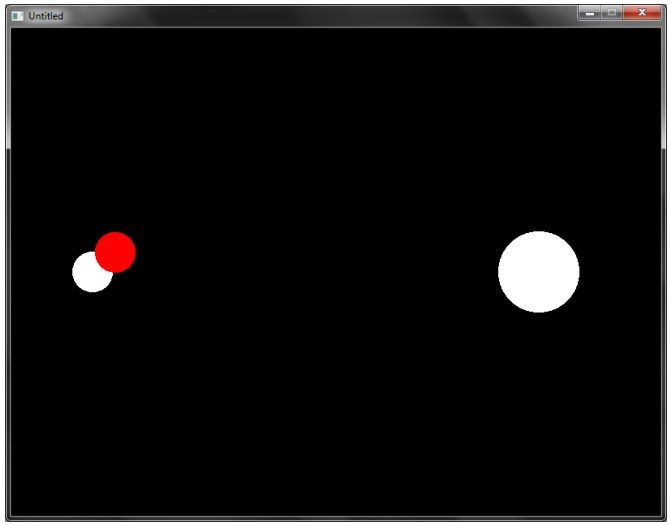


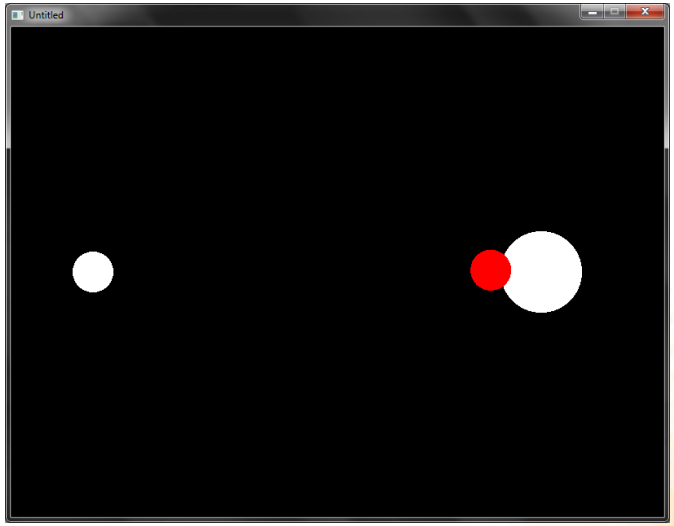
Collision Detection (Circle)

```
function love.load()
  player1 = {
    x = 390,
    y = 300,
    radius = 25,
    collided = false
  }
  circle1 = {
    x = 100,
    y = 300,
    radius = 25
  }
  circle2 = {
    x = 650,
    y = 300,
    radius = 50
  }
end
```

```
function CheckCircularCollision(ax, ay, ar, bx, by, br)
    local dx = bx - ax
    local dy = by - ay
    local dist = math.sqrt(dx * dx + dy * dy)
    return dist < ar + br
end
function love.update(dt)
  if love.keyboard.isDown("left") then
    player1.x = player1.x - (120 * dt)
  end
  if love.keyboard.isDown("right") then
    player1.x = player1.x + (120 * dt)
  end
  if love.keyboard.isDown("up") then
    player1.y = player1.y - (120 * dt)
  end
  if love.keyboard.isDown("down") then
    player1.y = player1.y + (120 * dt)
  end
  if CheckCircularCollision(player1.x, player1.y, player1.radius,
                            circle1.x, circle1.y, circle1.radius) or
     CheckCircularCollision(player1.x, player1.y, player1.radius,
                            circle2.x, circle2.y, circle2.radius) then
    player1.collided = true
  else
    player1.collided = false
  end
end
```

```
function love.draw()
  love.graphics.setColor(255,255,255)
  love.graphics.circle("fill",circle1.x, circle1.y, circle1.radius, 100)
  love.graphics.circle("fill",circle2.x, circle2.y, circle2.radius, 100)

  if player1.collided == true then
    love.graphics.setColor(255,0,0)
  end

  love.graphics.circle("fill",player1.x, player1.y, player1.radius, 100)
end
```





Example 1: Collision Detection with Physics

Example 1: Collision Detection with Physics

main.lua

```
require "vector2"

local position = vector2.new(400, 100)
local velocity = vector2.new(0, 0)
local playersize = vector2.new(30, 60)
local maxspeed = 400
local mass = 1
local frictioncoefficient = 300
local onGround = false

local groundposition = vector2.new(50, 550)
local groundsize = vector2.new(700, 50)

local box1position = vector2.new(500, 450)
local box1size = vector2.new(50, 50)

function love.update(dt)
  local acceleration = vector2.new(0, 0)
  local gravity = vector2.new(0, 500)
  acceleration = vector2.applyForce(gravity, mass, acceleration)

  ...
```

Example 1: Collision Detection with Physics

main.lua

```
local friction = vector2.mult(velocity, -1)
friction = vector2.normalize(friction)
friction = vector2.mult(friction, frictioncoefficient)
acceleration = vector2.applyForce(friction, mass, acceleration)

local movedirection = vector2.new(0, -1)
if love.keyboard.isDown("right") then
  local move = vector2.new(500, 0)
  acceleration = vector2.applyForce(move, mass, acceleration)
  movedirection.x = 1
end
if love.keyboard.isDown("left") then
  local move = vector2.new(-500, 0)
  acceleration = vector2.applyForce(move, mass, acceleration)
  movedirection.x = -1
end
if love.keyboard.isDown("up") and (onGround) then
  local jump = vector2.new(0, -50000)
  acceleration = vector2.applyForce(jump, mass, acceleration)
  movedirection.y = 1
  onGround = false
end
```

Example 1: Collision Detection with Physics

```
local futurevelocity = vector2.add(velocity,
                                   vector2.mult(acceleration, dt))
futurevelocity = vector2.limit(futurevelocity, maxspeed)
local futureposition = vector2.add(position,
                                   vector2.mult(futurevelocity, dt))
local collisiondir1 = GetBoxCollisionDirection(futureposition.x,
              futureposition.y, playersize.x, playersize.y,
              groundposition.x, groundposition.y, groundsize.x,
              groundsize.y)
if not (collisiondir1.x == 0 and collisiondir1.y == 0) then
  if collisiondir1.y == movedirection.y then --down collision
    velocity.y = 0
    acceleration.y = 0
    onGround = true
  elseif collisiondir1.y == 1 then --up collision
    velocity.y = 0
    acceleration.y = 0
  elseif movedirection.x ~= collisiondir1.x then --side collision
    velocity.x = 0
    acceleration.x = 0
  end
end
```

Example 1: Collision Detection with Physics

```
  local collisiondir2 = GetBoxCollisionDirection(futureposition.x,
                        futureposition.y, playersize.x, playersize.y,
                        box1position.x, box1position.y, box1size.x,
                        box1size.y)
  if not (collisiondir2.x == 0 and collisiondir2.y == 0) then
    if collisiondir2.y == movedirection.y then --ground collision
      velocity.y = 0
      acceleration.y = 0
      onGround = true
    elseif collisiondir2.y == 1 then --up collision
      velocity.y = 0
      acceleration.y = 0
    elseif movedirection.x ~= collisiondir2.x then --side collision
      velocity.x = 0
      acceleration.x = 0
    end
  end

  velocity = vector2.add(velocity, vector2.mult(acceleration, dt))
  velocity = vector2.limit(velocity, maxspeed)
  position = vector2.add(position, vector2.mult(velocity, dt))
end
```

Example 1: Collision Detection with Physics

```
function GetBoxCollisionDirection(x1,y1,w1,h1,x2,y2,w2,h2)
  local xdist = math.abs((x1 + (w1 / 2)) - (x2 + (w2 / 2)))
  local ydist = math.abs((y1 + (h1 / 2)) - (y2 + (h2 / 2)))
  local combinedwidth = (w1 / 2) + (w2 / 2)
  local combinedheight = (h1 / 2) + (h2 / 2)
  if(xdist > combinedwidth) then
    return vector2.new(0, 0)
  end
  if(ydist > combinedheight) then
    return vector2.new(0, 0)
  end
  local overlapx = math.abs(xdist - combinedwidth)
  local overlapy = math.abs(ydist - combinedheight)
  local playerdir = vector2.normalize(vector2.sub(vector2.new(x1,y1),
                                                  vector2.new(x2,y2)))
  local collisiondir
  if overlapx > overlapy then
    collisiondir = vector2.normalize(vector2.new(0, playerdir.y *
                                                    overlapy))
  elseif overlapx < overlapy then
    collisiondir = vector2.normalize(vector2.new(playerdir.x *
                                                    overlapx, 0))
```

Example 1: Collision Detection with Physics

```
  else
    collisiondir = vector2.normalize(vector2.new(playerdir.x *
                                    overlapx, playerdir.y * overlapy))
  end
  return collisiondir
end

function love.draw()
  love.graphics.setColor(0.2, 0.2, 0.9)
  love.graphics.rectangle("fill", position.x, position.y,
                          playersize.x, playersize.y)
  love.graphics.setColor(0.2, 0.8, 0.2)
  love.graphics.rectangle("fill", groundposition.x, groundposition.y,
                          groundsize.x, groundsize.y)
  love.graphics.rectangle("fill", box1position.x, box1position.y,
                          box1size.x, box1size.y)
end
```

Exercise 3

3) Using the code of the collision detection with physics example, implement a scene with at least four platforms (like the one illustrated bellow).

- Don’t worry about code repetitions or optimizations in this exercise.
  - In the next lecture we will see how to store the level geometry in an array and optimize the collision detection process.