MINI PROJECT
(2020-21)

BIRDY

PROJECT REPORT

Institute of Engineering & Technology

Submitted by
Abhishek Singh
(181500028)

Supervised By: -
Mr. Piyush Vashisth
Technical Trainer
Department of Computer Engineering & Applications

Contents

Declaration

We hereby declare that the work which is being presented in the Mini Project “Birdy” , in partial fulfillment of the requirements for Mini project is an authentic record of our own work carried under the supervision of Mr. Piyush Vashisth, Technical Trainer

Abhisshek Singh

ACKNOWLEDGEMENT

I take this opportunity to acknowledge all the people who have helped us wholeheartedly in every stage of this project.

I also extend our sincere thanks to all other faculty members of the Computer Science & Engineering Department.

Abstract

In this project I am going to build a 2d animated game called birdy which is a type of flappy bird game developed by Dong Ngyuyen which is a background scroll based game.

Introduction

1.1 History

The history of game making begins with the development of the first video games, although which video game is the first depends on the definition of *video game*. The first games created had little entertainment value, and their development focus was separate from user experience in fact, these games required mainframe computers to play them. *OXO*, written by Alexander S. Douglas in 1952, was the first computer game to use a digital display. In 1958, a game called *Tennis for Two*, which displayed its output on an oscilloscope, was made by Willy Higinbotham, a physicist working at the Brookhaven National Laboratory. In 1961, a mainframe computer game called *Spacewar!*was developed by a group of Massachusetts Institute of Technology students led by Steve Russell.

True commercial design and development of games began in the 1970s, when arcade video games and first-generation consoles were marketed. In 1971, *Computer Space* was the first commercially sold, coin-operated video game. It used a black-and-white television for its display, and the computer system was made of 74 seriesTTLchips. In 1972, the first home console system was released called Magnavox Odyssey, developed by Ralph H. Bity. The commercial success of *Pong* led other companies to develop *Pong* clones, spawning the video game industry.

Programmers workedaer. That same year, Atari released *Pong*, an arcade game that increased video game popular within the big companies to produce games for these devices. The industry did not see huge innovation in game design and a large number of consoles had very similar games. Many of these early games were often *Pong* clones. Some games were different, however, such as *Gun Fight*, which was significant for several reasons: an early 1975 on-foot, multi-directional shooter, which depicted game characters, game violence, and human-to-human combat.Tomohiro Nishikado's original version was based on discrete logic, which Dave Nutting

adapted using the Intel 8080, making it the first video game to use a microprocessor. Console manufacturers soon started to produce consoles that were able to play independently developed games,and ran on microprocessors, marking the beginning of second-generation consoles, beginning with the release of the Fairchild Channel F in 1976.

The flood of *Pong* clones led to the video game crash of 1977, which eventually came to an end with the mainstream success of Taito's 1978 arcade shooter game*Space Invaders*, marking the beginning of the golden age of arcade video games and inspiring dozens of manufacturers to enter the market. Its creator Nishikado not only designed and programmed the game, but also did the artwork, engineered the arcade hardware, and put together a microcomputer from scratch. It was soon ported to the Atari 2600, becoming the first "killer app" and quadrupling the console's sales. At the same time, home computers appeared on the market, allowing individual programmers and hobbyists to develop games. This allowed hardware manufacturer and software manufacturers to act separately. A very large amount of games could be produced by single individuals, as games were easy to make because graphical and memory limitation did not allow for much content. Larger companies developed, who focused selected teams to work on a title. The developers of many early home video games, such as *Zork*, *Baseball*, *Air Warrior*, and *Adventure*, later transitioned their work as products of the early video game industry.

> “I wouldn't recommend [designing computer games] for someone with a weak heart or a large appetite”
>
> —Jon Freeman, 1984

The industry expanded significantly at the time, with the arcade video game sector alone (representing the largest share of the gaming industry) generating higher revenues than both pop music and Hollywood films combined.

The home video game industry, however, suffered major losses following the North American video game crash of 1983. In 1984 Jon Freeman warned in *Computer Gaming World*:

Q: Are computer games the way to fame and fortune?

A: No. Not unless your idea of fame is having your name recognized by one or two astute individuals at Origins ... I've been making a living (after a fashion) designing games for most of the last six years. I wouldn't recommend it for someone with a weak heart or a large appetite, though.

Chris Crawford and Don Daglow in 1987 similarly advised prospective designers to write games as a hobby first, and to not quit their existing jobs early. The home video game industry was revitalized soon after by the widespread success of the Nintendo Entertainment System.

By 1987 a video game required 12 months to develop and another six to plan marketing. Projects remained usually solo efforts, with single developers delivering finished games to their publishers. With the ever-increasing processing and graphical capabilities of arcade, console and computer products, along with an increase in user expectations, game design moved beyond the scope of a single developer to produce a marketable game in a reasonable time. This sparked the beginning of team-based development.In broad terms, during the 1980s, pre-production involved sketches and test routines of the only developer. In the 1990s, pre-production consisted mostly of game art previews. In the early 2000s, pre-production usually produced a playable demo.

1.2 General Introduction to the topic

Flappy Bird is a mobile game developed by Vietnamese video game artist and programmer Dong Nguyen (Vietnamese: *Nguyễn Hà Đông*), under his game development company dotGears.[1] The game is a side-scroller where the player controls a bird, attempting to fly between columns of green pipes without hitting them. Nguyen created the game over the period of several days, using a bird protagonist that he had

designed for a cancelled game in 2012.

The game was released in May 2013 but received a sudden rise in popularity in early 2014. *Flappy Bird* received poor reviews from some critics, who criticized its high level of difficulty, alleged plagiarism in graphics and game mechanics, while other reviewers found it addictive. At the end of January 2014, it was the most downloaded free game in the App Store for iOS. During this period, its developer said that *Flappy Bird* was earning $50,000 a day from in-app advertisements as well as sales.

Flappy Bird was removed from both the App Store and Google Play by its creator on February 10, 2014. He claims that he felt guilt over what he considered to be its addictive nature and overuse. The game's popularity and sudden removal caused phones with it pre-installed to be put up for sale for high prices over the Internet.[2][3][4] Games similar to *Flappy Bird* became popular on the iTunes App Store in the wake of its removal, and both Apple and Google have removed games from their app stores for being too similar to the original.

In August 2014, a revised version of *Flappy Bird*, called *Flappy Birds Family*, was released exclusively for the Amazon Fire TV. Bay Tek Games also released a licensed coin-operated *Flappy Bird* arcade game.[5]

1.3 Gameplay

Faby after passing the first pair of pipes

Flappy Bird is an arcade-style game in which the player controls the bird Faby, which moves persistently to the right. The player is tasked with navigating Faby through pairs of pipes that have equally sized gaps placed at random heights. Faby automatically descends and only ascends when the player taps the touchscreen. Each successful pass through a pair of pipes awards the player one point. Colliding with a pipe or the ground ends the gameplay. During the game over screen, the player is awarded a bronze medal if they reached ten or more points, a silver medal from twenty points, a gold medal from thirty points, and a platinum medal from forty points

1.4 Hardware Requirements

- Memory [4GB RAM (or higher)]
- Intel core i3 64-bit Processor (or higher)

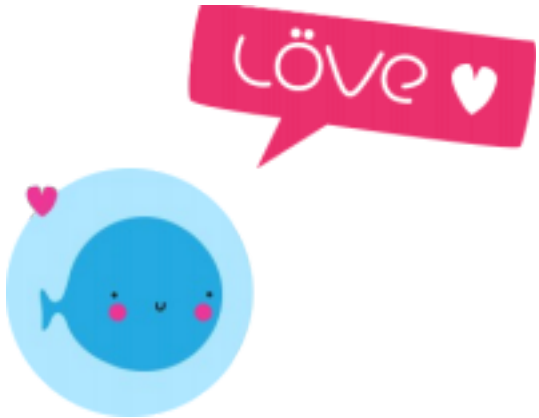
1.4 Software requirements

- Love 2D
- Sublime text3(or any text editor that support language LUA)

Objective

The purpose of the project is to design and implement a 2-dimensional game written in LUA using gaming library of love 2d. The level will include everything that should be available in an arcade adventure game like the popular Nintendo classic Super Mario game. The game will be a single-player adventure game. The goals of this project is to create an easy to use, pick up and play game that could be played by all ages as long as they have a desktop computer or a laptop pc. The reason was as stated above that they are more gamers playing video games every day meaning a larger potential market.

.



Technologies Used

Love 2D

What is love 2D?

LÖVE (or Love2D) is an open-source cross-platform engine for developing 2D video games. The engine is written in C++ and uses Lua as its scripting language. It is published under the zlib license.

The API provided by the engine gives access to the video and sound functions of the host machine through the libraries SDL and OpenGL, or since version 0.10 also OpenGL ES 2 and 3. Fonts can be rendered by the FreeType engine. A version of the engine called piLöve has been specifically ported to Raspberry Pi.

It also provides a basic "sandbox" management of the files in order to avoid giving access to all its disk to the executed games.

LÖVE is still maintained by its original developers.

This engine is frequently found in the compositions of video game development competitions, such as the international competition Ludum Dare.

In July 2018, it was the 10th game engine most used by independents on the site itch.io.

Features

These features come with the game engine:

- support of OpenGL pixel shaders GLSL,
- touchscreen support,
- support for UTF-8,
- supports image formats PNG, JPEG, GIF, TGA and BMP,[12]
- possibility to use the 2D physics engine Box2D (can be disabled, to lighten the library),

- luasocket library for network communications TCP/UDP,
- lua-enet library, another network library implementing Enet, a reliable protocol based on UDP
- native management of tiles created by the Tiled
- LOVE is a cross-platform

Additional libraries

There are various libraries to improve basic functions, such as object-oriented programming with inheritance and overloading, interpolations, camera management, network multiplayer management, game state management, configuration, etc.

The Simple Tiled Implementation library allows users to load levels as tiles, edit using Tiled and display them in games. It works in conjunction with Box2D for collision management with this decor.

The anim8 library allows users to load animations, for characters for example, from an image grid into a bitmap file (PNG or JPEG).

There is also a free platform (GPLv3) called LIKO-12, inspired by the PICO-8 *fantasy console* and using LÖVE, allowing to develop applications in a limited resolution, backup/restore in the modified PNG format, in the same way as the video game cartridges of the game consoles or some of the first microcomputers, and export them to HTML5 or to systems supported by LÖVE.

Language : LUA

Lua is a lightweight, high-level, multi-paradigm programming language designed primarily for embedded use in applications.[2] Lua is cross-platform, since the interpreter of compiled bytecode is written in ANSI C,[3] and Lua has a relatively simple C API to embed it into applications.[4]

Lua was originally designed in 1993 as a language for extending software applications to meet the increasing demand for customization at the time. It provided the basic facilities of most procedural programming languages, but more complicated or domain-specific features were not included; rather, it included mechanisms for extending the language, allowing programmers to implement such features. As Lua was intended to be a general embeddable extension language, the designers of Lua focused on improving its speed, portability, extensibility, and ease-of-use in development.

History

Lua was created in 1993 by Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes, members of the Computer Graphics Technology Group (Tecgraf) at the Pontifical Catholic University of Rio de Janeiro, in Brazil.

From 1977 until 1992, Brazil had a policy of strong trade barriers (called a market reserve) for computer hardware and software. In that atmosphere, Tecgraf's clients could not afford, either politically or financially, to buy customized software from abroad. Those reasons led Tecgraf to implement the basic tools it needed from scratch.

Lua's predecessors were the data-description/configuration languages *SOL* (Simple Object Language) and *DEL* (data-entry language). They had been independently developed at Tecgraf in 1992–1993 to add some flexibility into two different projects (both were interactive graphical programs for engineering applications at Petrobras company). There was a lack of any flow-control structures in SOL and DEL, and Petrobras felt a growing need to add full programming power to them.

In *The Evolution of Lua*, the language's authors wrote:

In 1993, the only real contender was Tcl, which had been explicitly designed to be embedded into

applications. However, Tcl had unfamiliar syntax, did not offer good support for data description, and ran only on Unix platforms. We did not consider LISP or Scheme because of their unfriendly syntax. Python was still in its infancy. In the free, do-it-yourself atmosphere that then reigned in Tecgraf, it was quite natural that we should try to develop our own scripting language ... Because many potential users of the language were not professional programmers, the language should avoid cryptic syntax and semantics. The implementation of the new language should be highly portable, because Tecgraf's clients had a very diverse collection of computer platforms. Finally, since we expected that other Tecgraf products would also need to embed a scripting language, the new language should follow the example of SOL and be provided as a library with a C API.

Lua 1.0 was designed in such a way that its object constructors, being then slightly different from the current light and flexible style, incorporated the data-description syntax of SOL (hence the name Lua: *Sol* is also the Portuguese word for "Sun", *Lua* being the word for "Moon"). Lua syntax for control structures was mostly borrowed from Modula (`if`, `while`, `repeat`/`until`), but also had taken influence from CLU (multiple assignments and multiple returns from function calls, as a simpler alternative to reference parameters or explicit pointers), C++ ("neat idea of allowing a local variable to be declared only where we need it"), SNOBOL and AWK (associative arrays). In an article published in *Dr. Dobb's Journal*, Lua's creators also state that LISP and Scheme with their single, ubiquitous data-structure mechanism (the list) were a major influence on their decision to develop the table as the primary data structure of Lua.

Lua semantics have been increasingly influenced by Scheme over time, especially with the introduction of anonymous functions and full lexical scoping. Several features were added in new Lua versions.

Versions of Lua prior to version 5.0 were released under a license similar to the BSD license. From version 5.0 onwards, Lua has been licensed under the MIT License. Both are permissive free software licences and are almost identical.

Features

Lua is commonly described as a "multi-paradigm" language, providing a small set of general features that can be extended to fit different problem types. Lua does not contain explicit support for inheritance, but allows it to be implemented with metatables. Similarly, Lua allows programmers to implement namespaces, classes, and other related features using its single table implementation; first-class functions allow the employment of many techniques from functional programming; and full lexical scoping allows fine-grained information hiding to enforce the principle of least privilege.

In general, Lua strives to provide simple, flexible meta-features that can be extended as needed, rather than supply a feature-set specific to one programming paradigm. As a result, the base language is light—the full reference interpreter is only about 247 kB compiled[3]—and easily adaptable to a broad range of applications.

Lua is a dynamically typed language intended for use as an extension or scripting language and is compact enough to fit on a variety of host platforms. It supports only a small number of atomic data structures such as boolean values, numbers (double-precision floating point and 64-bit integers by default), and strings. Typical data structures such as arrays, sets, lists, and records can be represented using Lua's single native data structure, the table, which is essentially a heterogeneous associative array.

Lua implements a small set of advanced features such as first-class functions, garbage collection, closures, proper tail calls, coercion (automatic conversion between string and number values at run time), coroutines (cooperative multitasking) and dynamic module loading.

Syntax

The classic "Hello, World!" program can be written as follows:

```
print("Hello World!")
```

or as:

```
print 'Hello World!'
```

Object Oriented Programming

Although Lua does not have a built-in concept of classes, object-oriented programming can be achieved using two language features: first-class functions and tables. By placing functions and related data into a table, an object is formed. Inheritance (both single and multiple) can be implemented using the metatable mechanism, telling the object to look up nonexistent methods and fields in parent object(s).

There is no such concept as "class" with these techniques; rather, prototypes are used, similar to Self or JavaScript. New objects are created either with a factory method (that constructs new objects from scratch) or by cloning an existing object.

Lua provides some syntactic sugar to facilitate object orientation. To declare member functions inside a prototype table, one can use `function table:func(args)`, which is equivalent to `function table.func(self, args)`. Calling class methods also makes use of the colon: `object:func(args)` is equivalent to `object.func(object, args)`.

Creating a basic vector object:

```
local Vector = {}

Vector.__index = Vector


function Vector:new(x, y, z)    -- The constructor
  return setmetatable({x = x, y = y, z = z}, Vector)
```

```
end


function Vector:magnitude()      -- Another method
  -- Reference the implicit object using self
  return math.sqrt(self.x^2 + self.y^2 + self.z^2)
end


local vec = Vector:new(0, 1, 0) -- Create a vector
print(vec:magnitude())          -- Call a method (output: 1)
print(vec.x)                    -- Access a member variable (output: 0)
```

Implementation

Lua programs are not interpreted directly from the textual Lua file, but are compiled into bytecode, which is then run on the Lua virtual machine. The compilation process is typically invisible to the user and is performed during run-time, but it can be done offline in order to increase loading performance or reduce the memory footprint of the host environment by leaving out the compiler. Lua bytecode can also be produced and executed from within Lua, using the `dump` function from the string library and the `load/loadstring/loadfile` functions. Lua version 5.3.4 is implemented in approximately 24,000 lines of C code.

Like most CPUs, and unlike most virtual machines (which are stack-based), the Lua VM is register-based, and therefore more closely resembles an actual hardware design. The register architecture both avoids excessive copying of values and reduces the total number of instructions per function. The virtual machine of Lua 5 is one of the first register-based pure VMs to have a wide use.[12] Parrot and Android's Dalvik are two other well-known register-based VMs. PCScheme's VM was also register-based.

This example is the bytecode listing of the factorial function defined above (as shown by the `luac`

5.1 compiler)

Application

In video game development, Lua is widely used as a scripting language by programmers, mainly due to its perceived easiness to embed, fast execution, and short learning curve.] It is often used for manipulating the objects and elements players see in the game.

In 2003, a poll conducted by GameDev.net showed Lua was the most popular scripting language for game programming.On 12 January 2012, Lua was announced as a winner of the Front Line Award 2011 from the magazine *Game Developer* in the category Programming Tools.

A large number of non-game applications also use Lua for extensibility, such as LuaTeX, an implementation of the TeX type-setting language, Redis, a key-value database, Neovim, a text editor, and Nginx, a web server.

Through the Scribunto extension, Lua is available as a server side scripting language in the MediaWiki software that powers Wikipedia and other wikis.Among its uses are allowing the integration of data from Wikidata into articles,] and powering the automated taxobox system.

Game Content

Background:

it is the main part of the game which shows as moving as the game remain in play state.

Bird:

It is another part of the game which is mainly control to play the game and use to move between the pipes to score.

Pipe:

it is the main part of the game which randomly changes in distance to make difficulty for the game.

Medal:

it is given in game to the player for scoring points as they move the bird between the pipe.

Implementation Details

Part1: first is to download LOVE 2D application on which our game will be test after implementing the game of code in Sublime text 3 which is essentially most important part.
Part 2: we have to create two file called main and push which will be the most important part of the game.

Part 3: now add background,pipe,bird images in the game and implement the coding of bird(by which the player will move the bird).

Part 4: we create pipepair file so that we can see the pipe moving in pair from which our bird pass to advance.

Part 5: anti gravity update in which we see our bird jump when we press space key

Part 6: collision update in which game will over when bird will strike the ground or pipe.

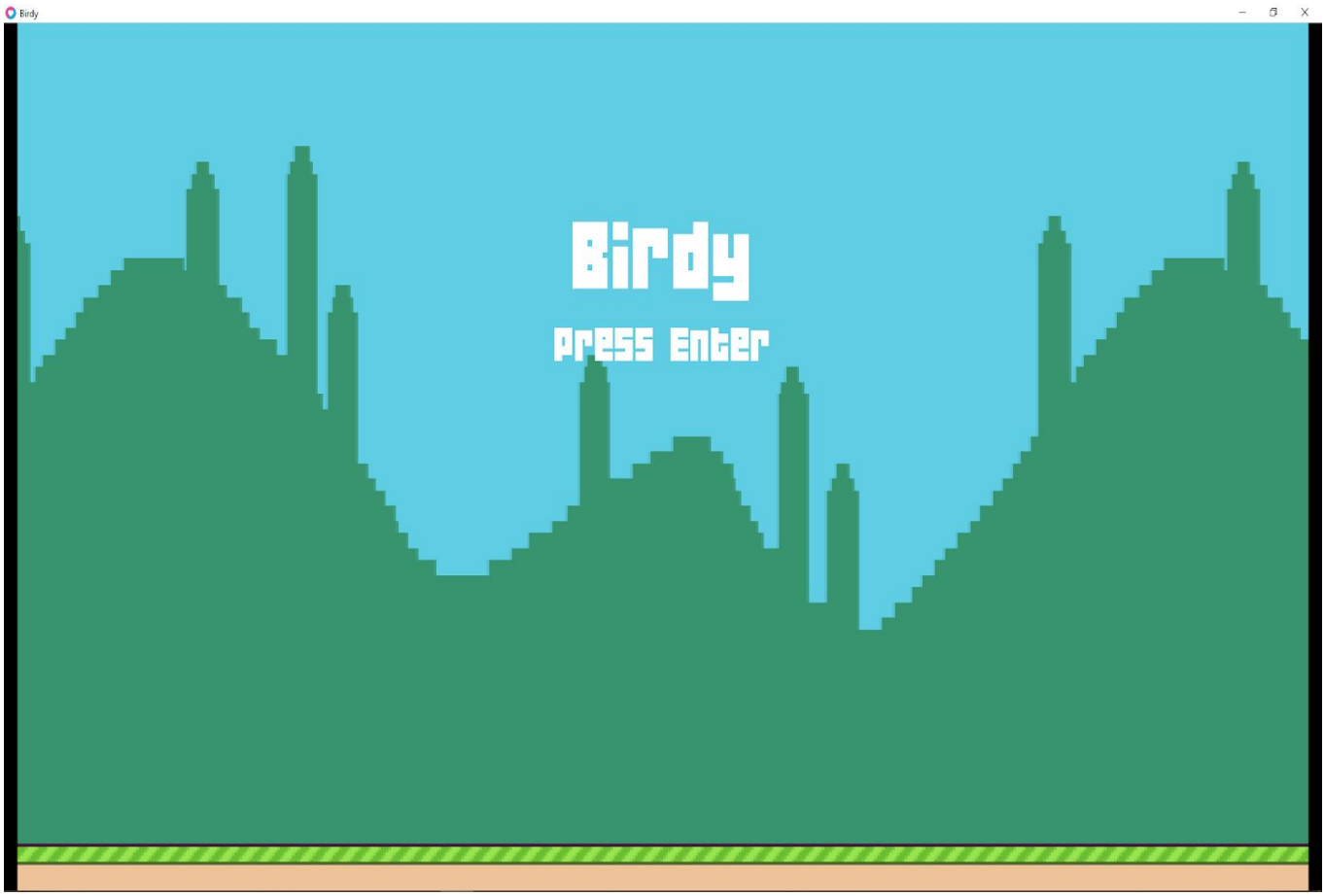
Part 7: Pause and resume update.

Part 8: Audio update.

Project Description

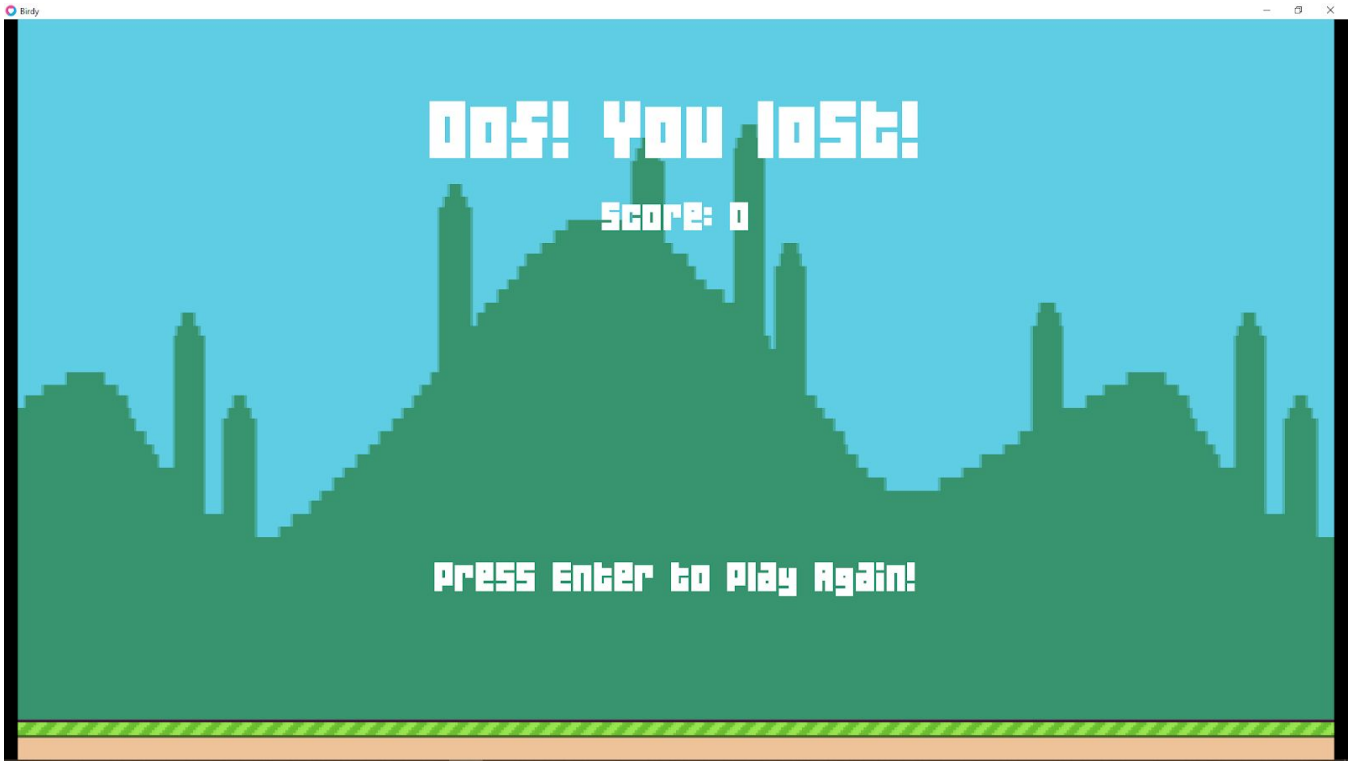
Title Screen

When game starts it shows the player a instruction of to press Enter to star the game.



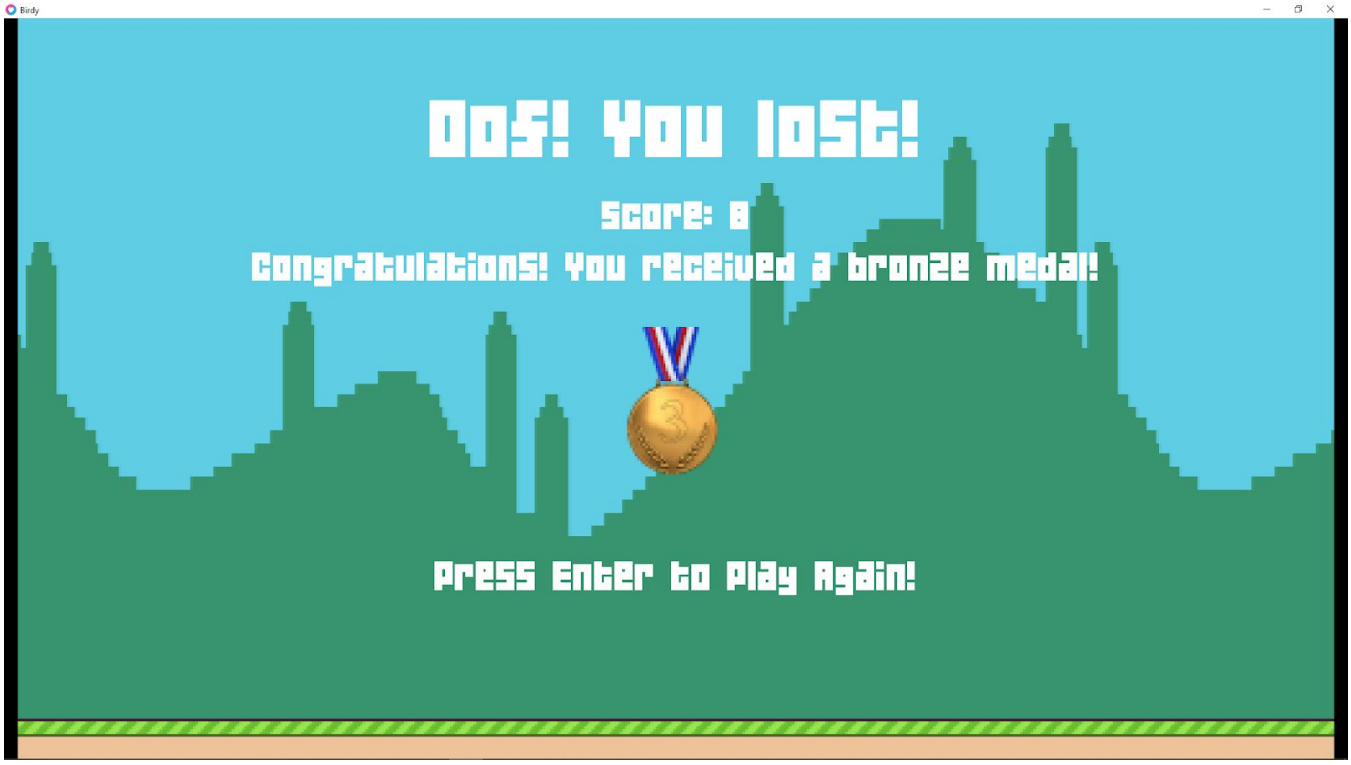
Lost Screen

When player lost this screen will show up.



Award Screen

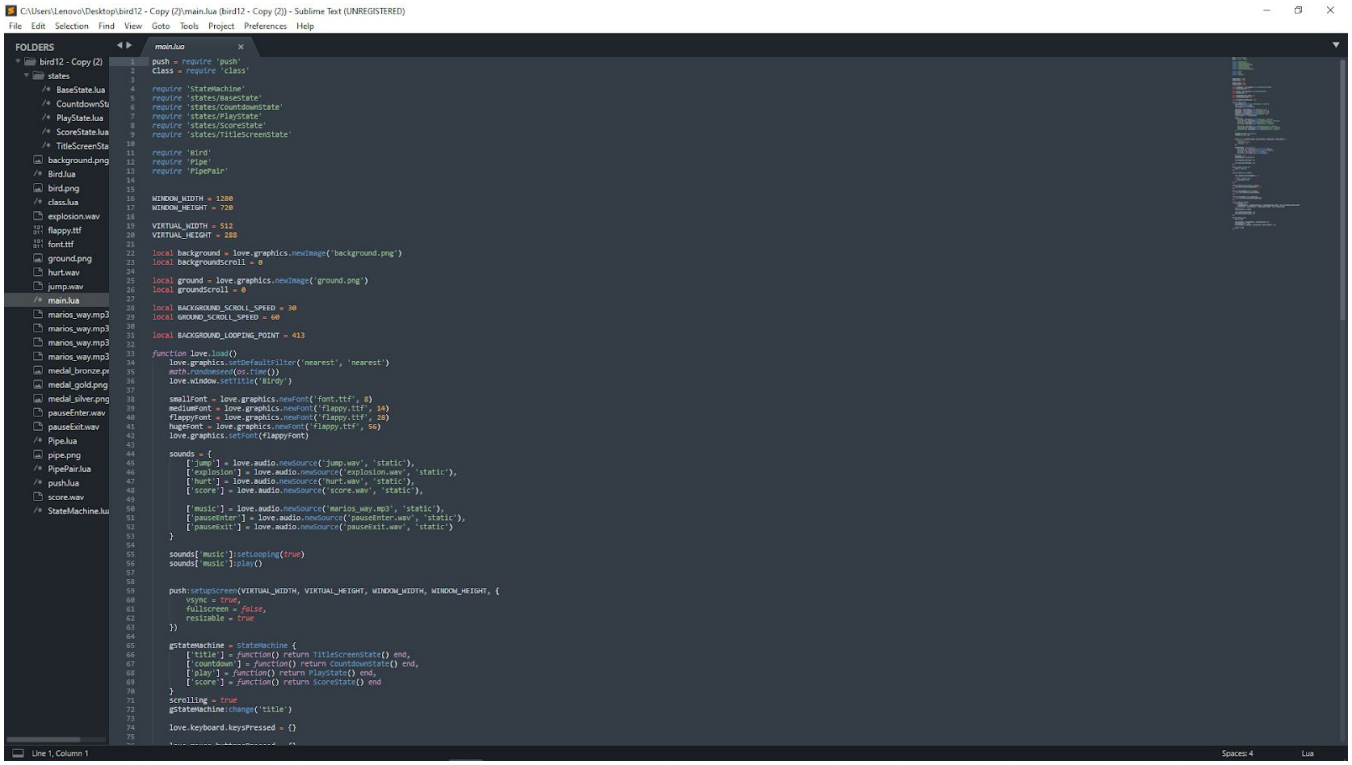
Player will receive medal for scores.

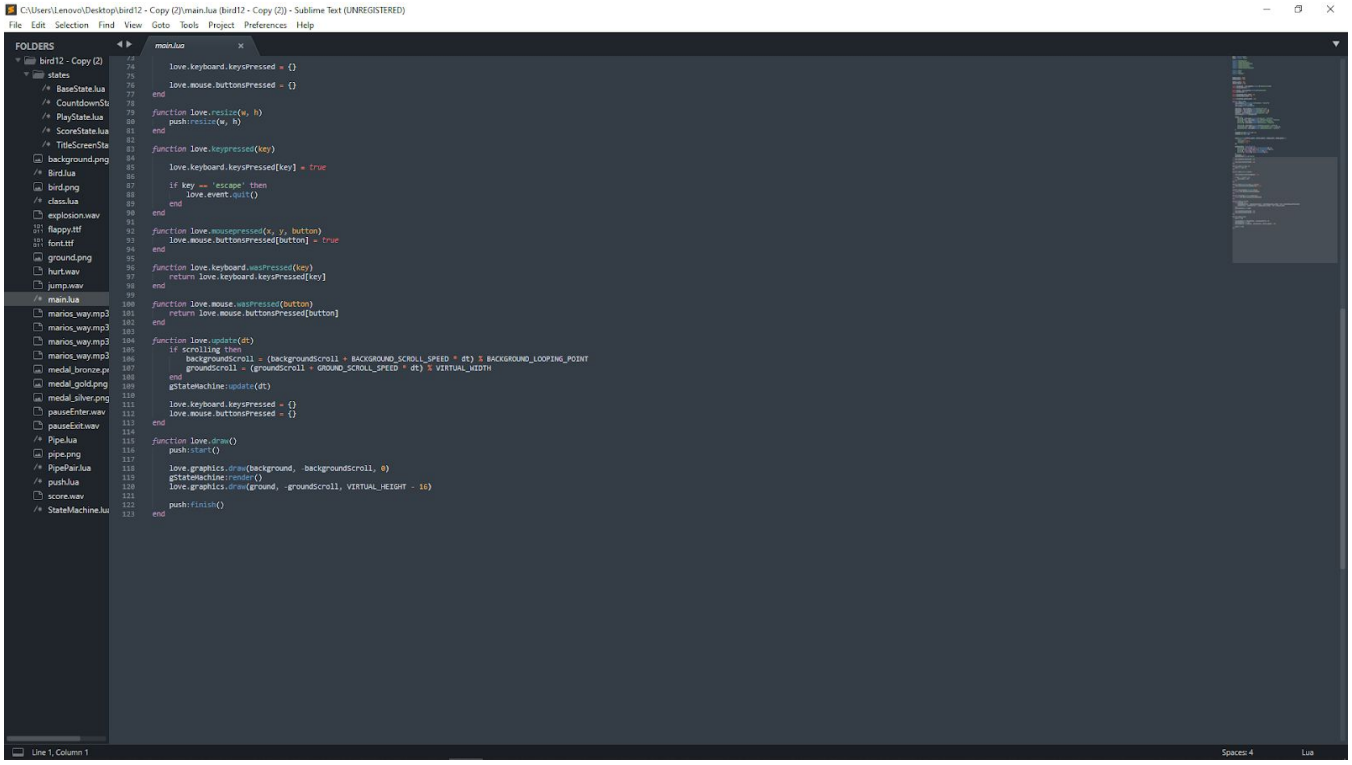


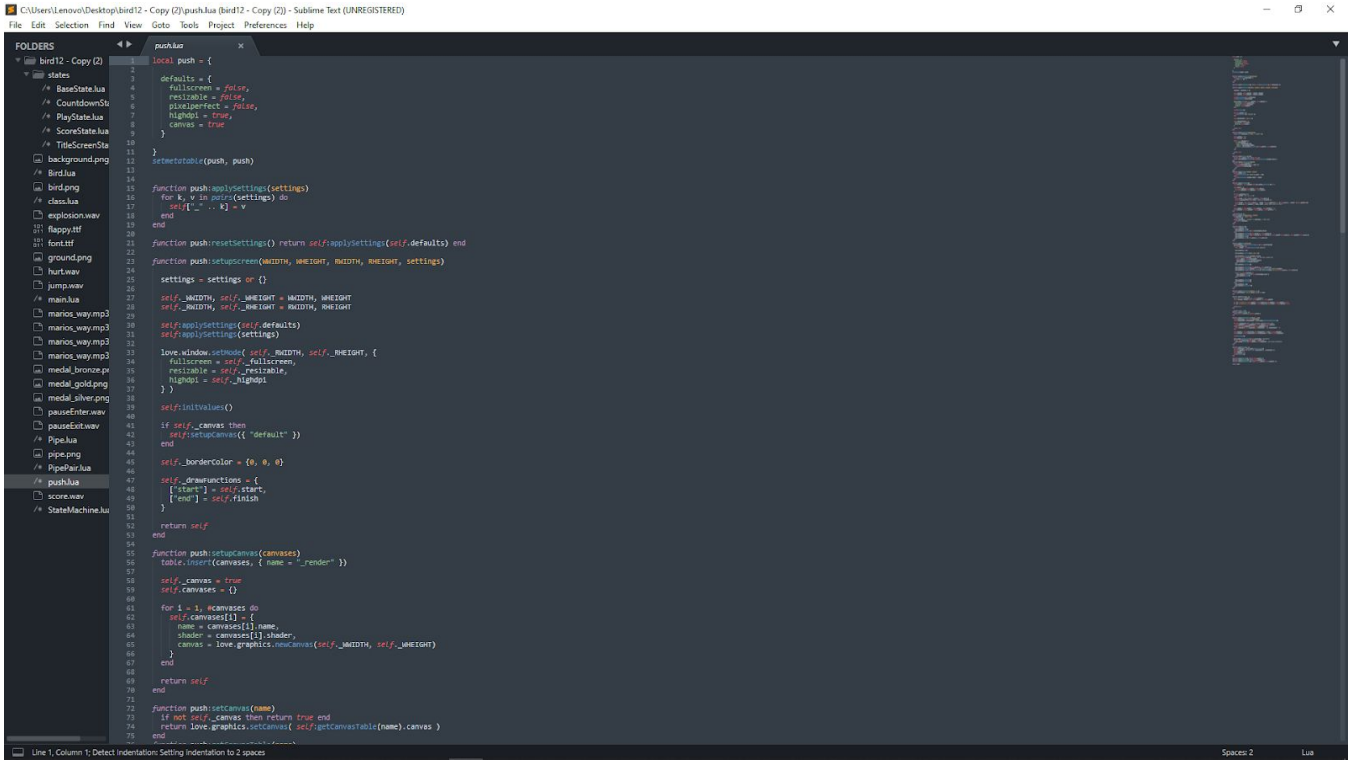
Pause & Resume

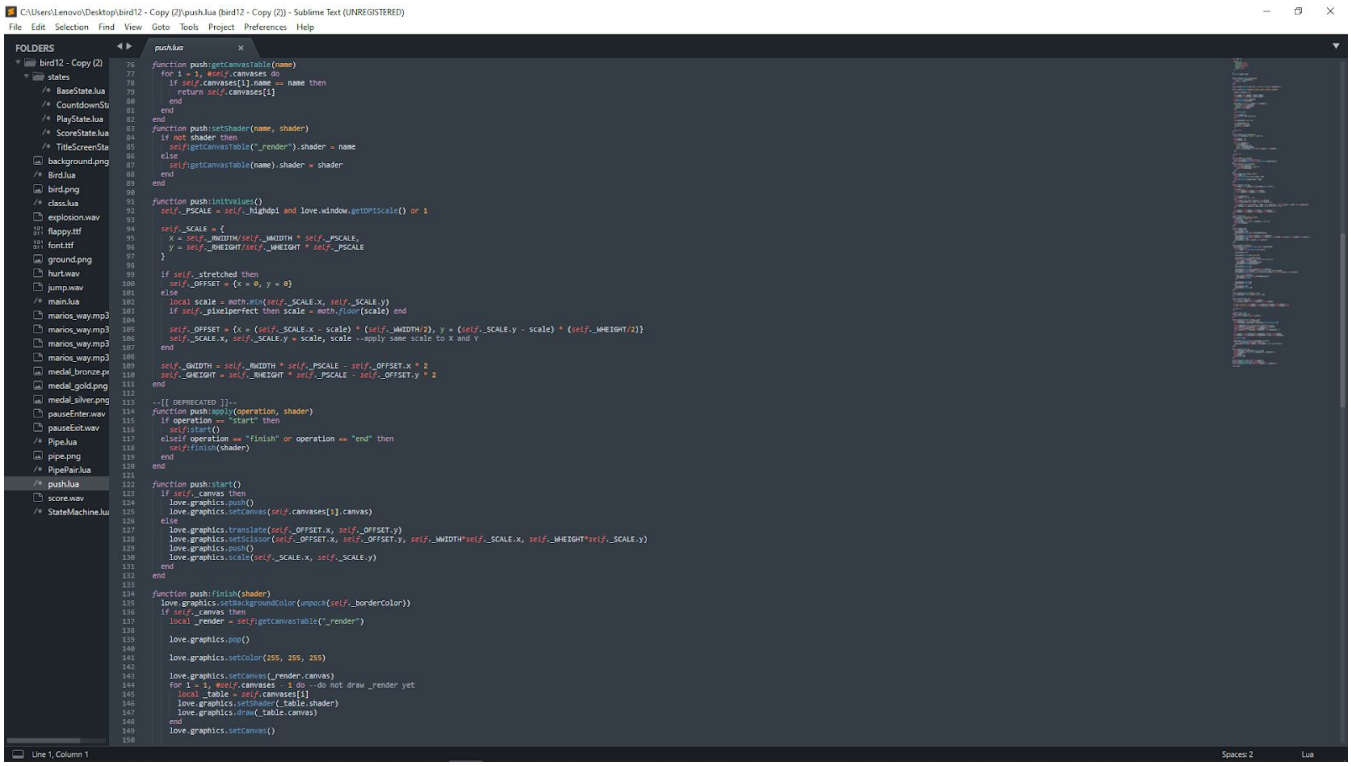
To pause and resume player has to press the P key on keyboard.

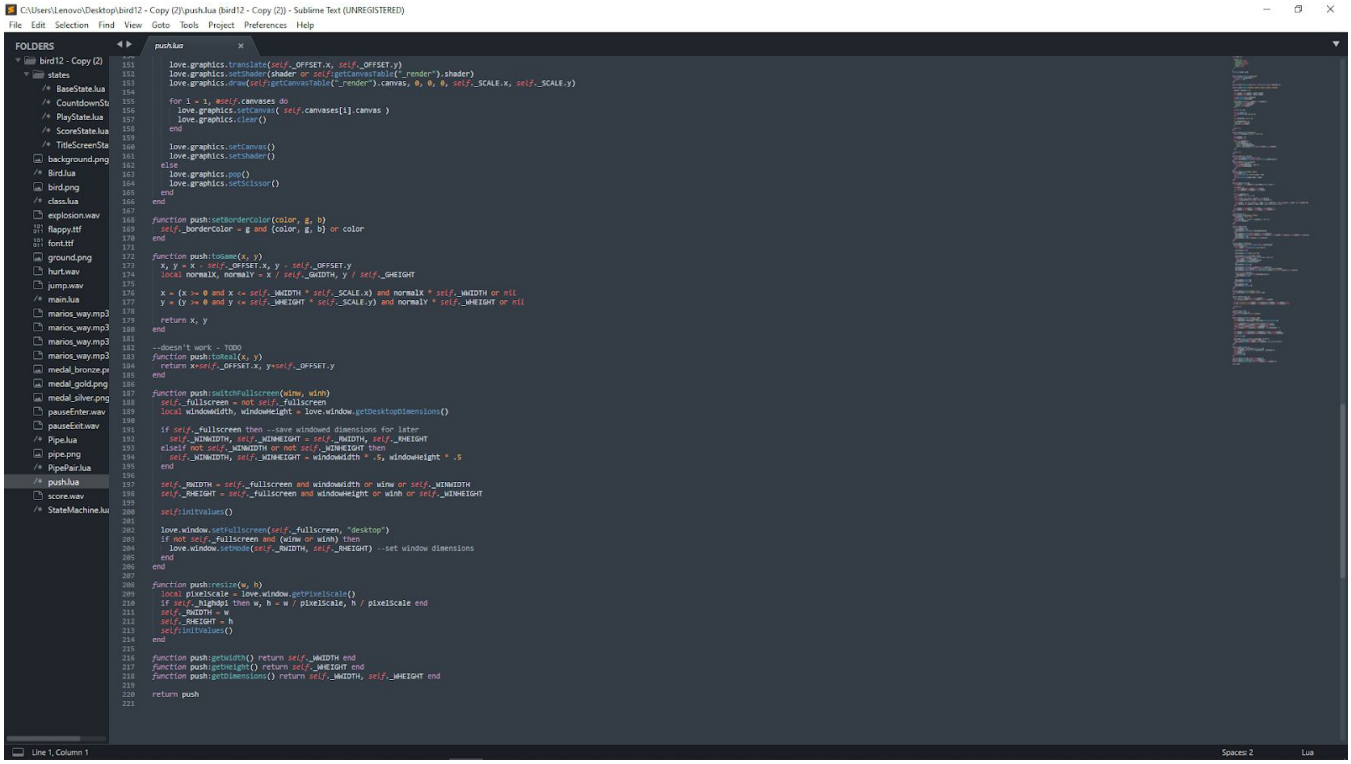
SCREENSHOTS

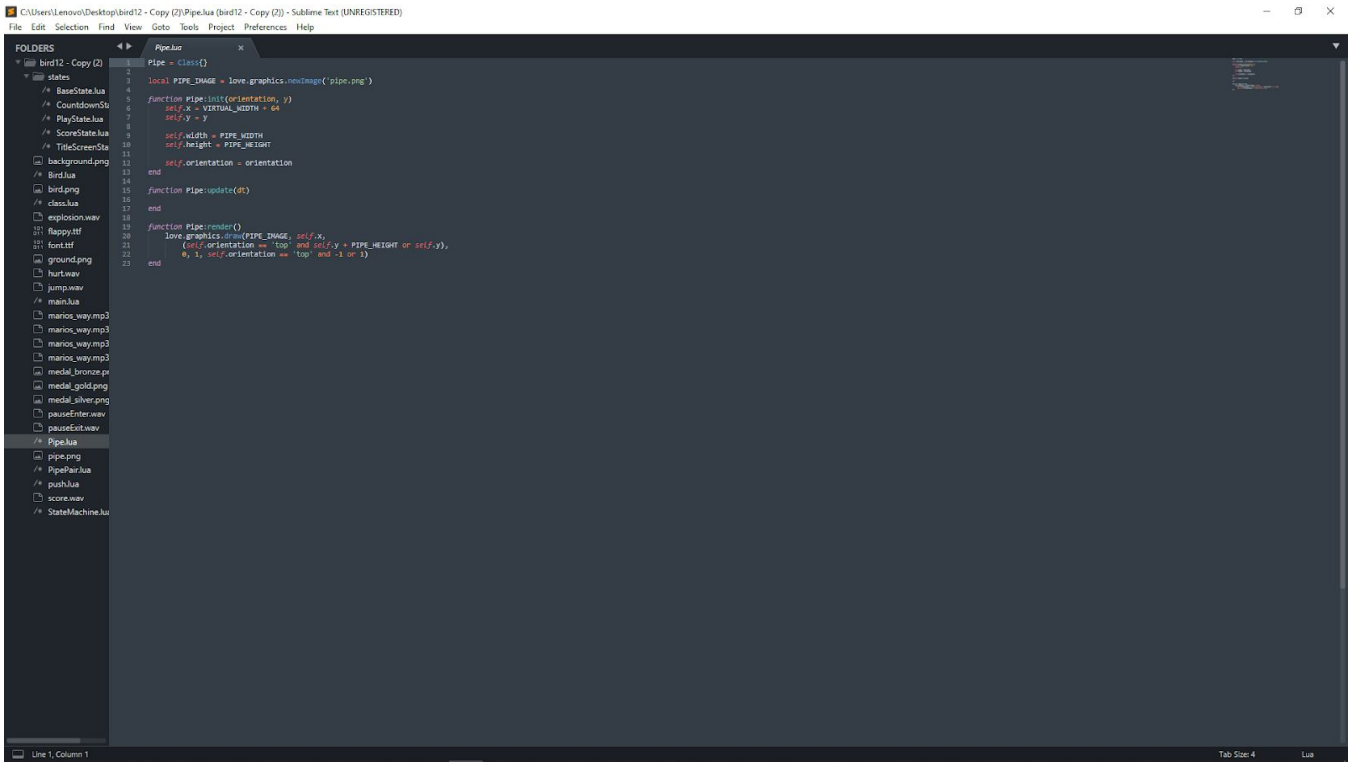





C:\Users\Lenovo\Desktop\bird12 - Copy (2)\push.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
push.lua
local push = {
defaults = {
fullscreen = false,
resizable = false,
pixelperfect = false,
highdpi = true,
canvas = true
}
}
setmetatable(push, push)
function push:applySettings(settings)
for k, v in pairs(settings) do
self["_" .. k] = v
end
end
function push:resetSettings() return self:applySettings(self.defaults) end
function push:setupScreen(WWIDTH, WHEIGHT, RWIDTH, RHEIGHT, settings)
settings = settings or {}
self._WWIDTH, self._WHEIGHT = WWIDTH, WHEIGHT
self._RWIDTH, self._RHEIGHT = RWIDTH, RHEIGHT
self:applySettings(self.defaults)
self:applySettings(settings)
love.window.setMode(self._RWIDTH, self._RHEIGHT, {
fullscreen = self._fullscreen,
resizable = self._resizable,
highdpi = self._highdpi
})
self:initValues()
if self._canvas then
self:setupCanvas({ "default" })
end
self._borderColor = {0, 0, 0}
self._drawFunctions = {
["start"] = self.start,
["end"] = self.finish
}
return self
end
function push:setupCanvas(canvases)
table.insert(canvases, { name = "_render" })
self._canvas = true
self.canvases = {}
for i = 1, #canvases do
self.canvases[i] = {
name = canvases[i].name,
shader = canvases[i].shader,
canvas = love.graphics.newCanvas(self._WWIDTH, self._WHEIGHT)
}
end
return self
end
function push:setCanvas(name)
if not self._canvas then return true end
return love.graphics.setCanvas(self:getCanvasTable(name).canvas)
end
Line 1, Column 1; Detect indentation: Setting indentation to 2 spaces
Spaces: 2
Lua


C:\Users\Lenovo\Desktop\bird12 - Copy (2)\push.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
push.lua
function push:getCanvasTable(name)
for i = 1, #self.canvases do
if self.canvases[i].name == name then
return self.canvases[i]
end
end
end
function push:setShader(name, shader)
if not shader then
self:getCanvasTable("_render").shader = name
else
self:getCanvasTable(name).shader = shader
end
end
function push:initValues()
self._PSCALE = self._highdpi and love.window.getDPIScale() or 1
self._SCALE = {
x = self._RWIDTH/self._WWIDTH * self._PSCALE,
y = self._RHEIGHT/self._WHEIGHT * self._PSCALE
}
if self._stretched then
self._OFFSET = {x = 0, y = 0}
else
local scale = math.min(self._SCALE.x, self._SCALE.y)
if self._pixelperfect then scale = math.floor(scale) end
self._OFFSET = {x = (self._SCALE.x - scale) * (self._WWIDTH/2), y = (self._SCALE.y - scale) * (self._WHEIGHT/2)}
self._SCALE.x, self._SCALE.y = scale, scale --apply same scale to X and Y
end
self._GWIDTH = self._RWIDTH * self._PSCALE - self._OFFSET.x * 2
self._GHEIGHT = self._RHEIGHT * self._PSCALE - self._OFFSET.y * 2
end
--[[DEPRECATED]]--
function push:apply(operation, shader)
if operation == "start" then
self:start()
elseif operation == "finish" or operation == "end" then
self:finish(shader)
end
end
function push:start()
if self._canvas then
love.graphics.push()
love.graphics.setCanvas(self.canvases[1].canvas)
else
love.graphics.translate(self._OFFSET.x, self._OFFSET.y)
love.graphics.setScissor(self._OFFSET.x, self._OFFSET.y, self._WWIDTH*self._SCALE.x, self._WHEIGHT*self._SCALE.y)
love.graphics.push()
love.graphics.scale(self._SCALE.x, self._SCALE.y)
end
end
function push:finish(shader)
love.graphics.setBackgroundColor(unpack(self._borderColor))
if self._canvas then
local _render = self:getCanvasTable("_render")
love.graphics.pop()
love.graphics.setColor(255, 255, 255)
love.graphics.setCanvas(_render.canvas)
for i = 1, #self.canvases - 1 do --do not draw _render yet
local _table = self.canvases[i]
love.graphics.setShader(_table.shader)
love.graphics.draw(_table.canvas)
end
love.graphics.setCanvas()
Line 1, Column 1
Spaces: 2
Lua


C:\Users\Lenovo\Desktop\bird12 - Copy (2)\push.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
push.lua
love.graphics.translate(self._OFFSET.x, self._OFFSET.y)
love.graphics.setShader(shader or self:getCanvasTable('_render').shader)
love.graphics.draw(self:getCanvasTable('_render').canvas, 0, 0, 0, self._SCALE.x, self._SCALE.y)
for i = 1, #self.canvases do
love.graphics.setCanvas(self.canvases[i].canvas)
love.graphics.clear()
end
love.graphics.setCanvas()
love.graphics.setShader()
else
love.graphics.pop()
love.graphics.setScissor()
end
end
function push:setBorderColor(color, g, b)
self._borderColor = g and {color, g, b} or color
end
function push:toGame(x, y)
x, y = x - self._OFFSET.x, y - self._OFFSET.y
local normalX, normalY = x / self._GWIDTH, y / self._GHEIGHT
x = (x >= 0 and x <= self._WIDTH * self._SCALE.x) and normalX * self._WIDTH or nil
y = (y >= 0 and y <= self._HEIGHT * self._SCALE.y) and normalY * self._HEIGHT or nil
return x, y
end
--doesn't work - TODO
function push:toReal(x, y)
return x+self._OFFSET.x, y+self._OFFSET.y
end
function push:switchFullscreen(winw, winh)
self._fullscreen = not self._fullscreen
local windowWidth, windowHeight = love.window.getDesktopDimensions()
if self._fullscreen then --save windowed dimensions for later
self._WINWIDTH, self._WINHEIGHT = self._RWIDTH, self._RHEIGHT
elseif not self._WINWIDTH or not self._WINHEIGHT then
self._WINWIDTH, self._WINHEIGHT = windowWidth * .5, windowHeight * .5
end
self._RWIDTH = self._fullscreen and windowWidth or winw or self._WINWIDTH
self._RHEIGHT = self._fullscreen and windowHeight or winh or self._WINHEIGHT
self:initValues()
love.window.setFullscreen(self._fullscreen, "desktop")
if not self._fullscreen and (winw or winh) then
love.window.setMode(self._RWIDTH, self._RHEIGHT) --set window dimensions
end
end
function push:resize(w, h)
local pixelScale = love.window.getPixelScale()
if self._highdpi then w, h = w / pixelScale, h / pixelScale end
self._RWIDTH = w
self._RHEIGHT = h
self:initValues()
end
function push:getWidth() return self._WIDTH end
function push:getHeight() return self._HEIGHT end
function push:getDimensions() return self._WIDTH, self._HEIGHT end
return push
Line 1, Column 1
Spaces: 2
Lua


C:\Users\Lenovo\Desktop\bird12 - Copy (2)\Pipe.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
Pipe.lua
Pipe = Class{}
local PIPE_IMAGE = love.graphics.newImage('pipe.png')
function Pipe:init(orientation, y)
self.x = VIRTUAL_WIDTH + 64
self.y = y
self.width = PIPE_WIDTH
self.height = PIPE_HEIGHT
self.orientation = orientation
end
function Pipe:update(dt)
end
function Pipe:render()
love.graphics.draw(PIPE_IMAGE, self.x,
(self.orientation == 'top' and self.y + PIPE_HEIGHT or self.y),
0, 1, self.orientation == 'top' and -1 or 1)
end
Line 1, Column 1
Tab Size: 4
Lua

C:\Users\Lenovo\Desktop\bird12 - Copy (2)\PipePair.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

bird12 - Copy (2)
- states
  - BaseState.lua
  - CountdownSta
  - PlayState.lua
  - ScoreState.lua
  - TitleScreenSta
- background.png
- Bird.lua
- bird.png
- class.lua
- explosion.wav
- flappy.ttf
- font.ttf
- ground.png
- hurt.wav
- jump.wav
- main.lua
- marios_way.mp3
- marios_way.mp3
- marios_way.mp3
- marios_way.mp3
- medal_bronze.pn
- medal_gold.png
- medal_silver.png
- pauseEnter.wav
- pauseExit.wav
- Pipe.lua
- pipe.png
- PipePair.lua
- push.lua
- score.wav
- StateMachine.lua

PipePair.lua

```lua
PipePair = Class{}

local GAP_HEIGHT = 90

function PipePair:init(y)

    self.scored = false
    self.x = VIRTUAL_WIDTH + 32
    self.y = y

    self.pipes = {
        ['upper'] = Pipe('top', self.y),
        ['lower'] = Pipe('bottom', self.y + PIPE_HEIGHT + math.random(GAP_HEIGHT-5, GAP_HEIGHT+30))
    }

    self.remove = false
end

function PipePair:update(dt)

    if self.x > -PIPE_WIDTH then
        self.x = self.x - PIPE_SPEED * dt
        self.pipes['lower'].x = self.x
        self.pipes['upper'].x = self.x
    else
        self.remove = true
    end
end

function PipePair:render()
    for l, pipe in pairs(self.pipes) do
        pipe:render()
    end
end
```

Line 1, Column 1; Detect indentation: Setting indentation to 4 spaces

Spaces: 4 Lua

C:\Users\Lenovo\Desktop\bird12 - Copy (2)\Bird.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

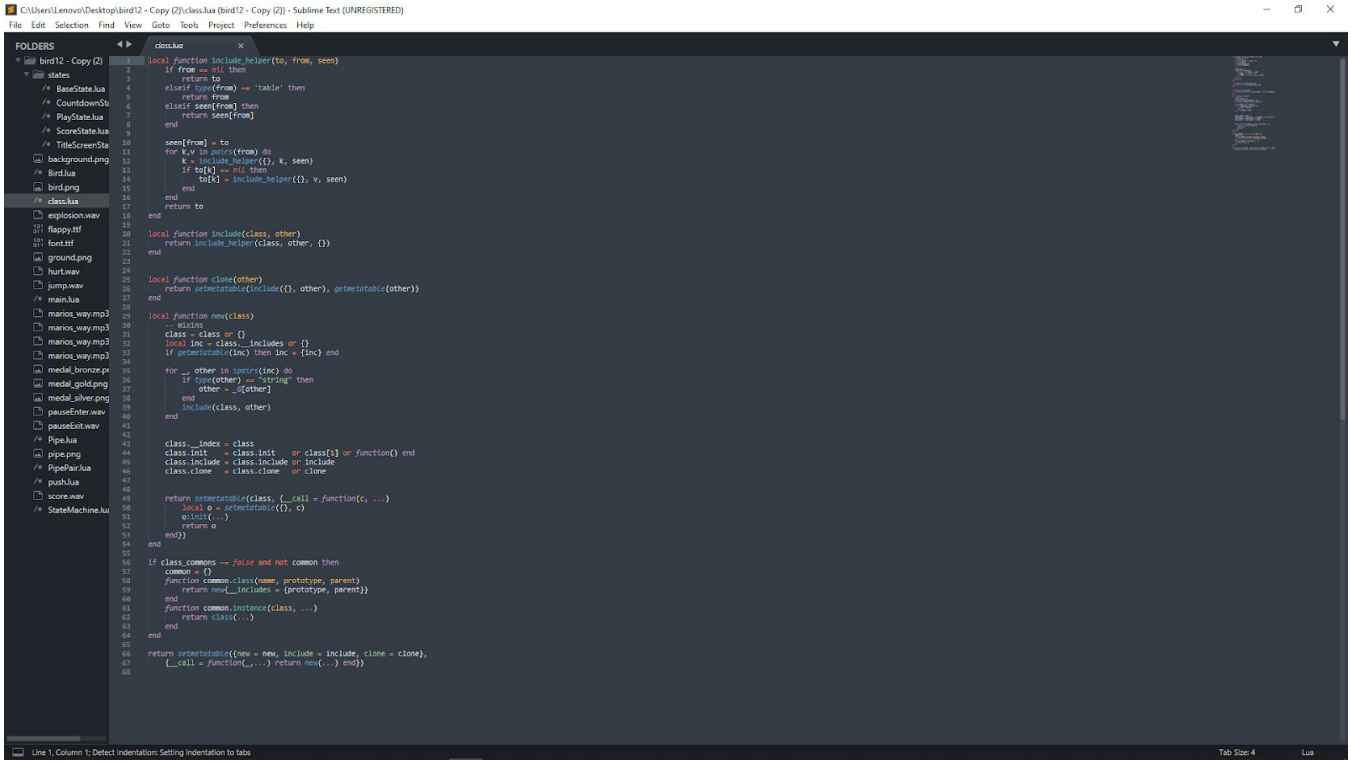
bird12 - Copy (2)
- states
  - BaseState.lua
  - CountdownSta
  - PlayState.lua
  - ScoreState.lua
  - TitleScreenSta
- background.png
- Bird.lua
- bird.png
- class.lua
- explosion.wav
- flappy.ttf
- font.ttf
- ground.png
- hurt.wav
- jump.wav
- main.lua
- marios_way.mp3
- marios_way.mp3
- marios_way.mp3
- marios_way.mp3
- medal_bronze.pn
- medal_gold.png
- medal_silver.png
- pauseEnter.wav
- pauseExit.wav
- Pipe.lua
- pipe.png
- PipePair.lua
- push.lua
- score.wav
- StateMachine.lua

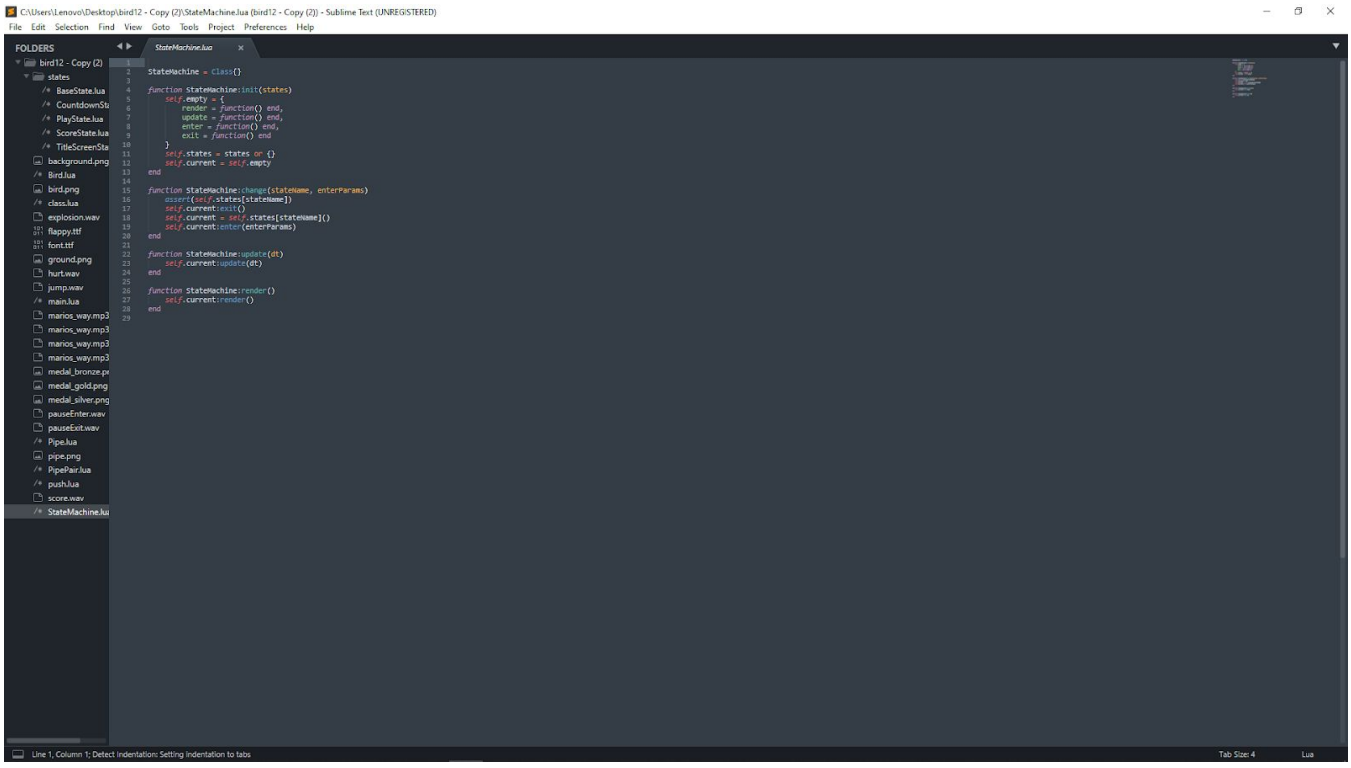
Bird.lua

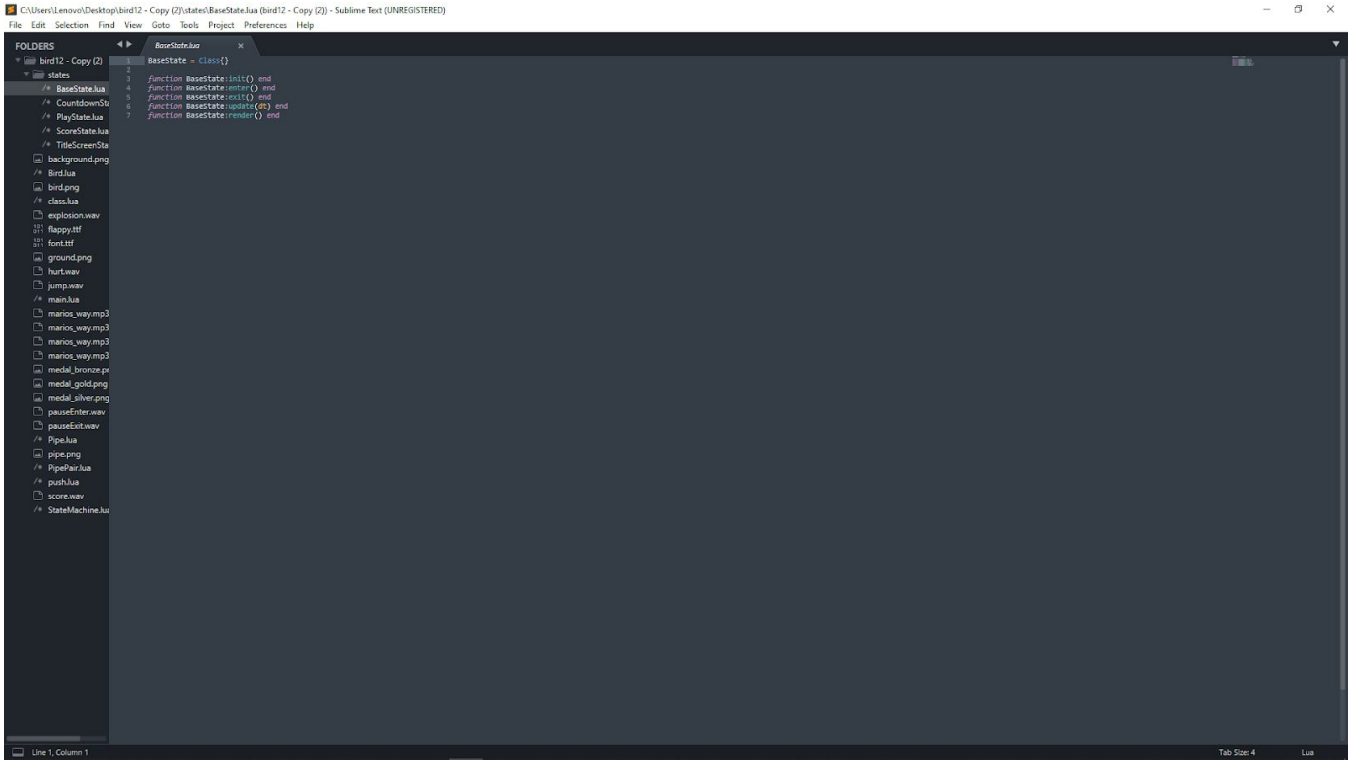
```lua
Bird = Class{}

local GRAVITY = 25

function Bird:init()
    self.image = love.graphics.newImage('bird.png')
    self.x = VIRTUAL_WIDTH / 2 - 8
    self.y = VIRTUAL_HEIGHT / 2 - 8

    self.width = self.image:getWidth()
    self.height = self.image:getHeight()

    self.dy = 0
end

function Bird:collides(pipe)

    if (self.x + 2) + (self.width - 4) >= pipe.x and self.x + 2 <= pipe.x + PIPE_WIDTH then
        if (self.y + 2) + (self.height - 4) >= pipe.y and self.y + 2 <= pipe.y + PIPE_HEIGHT then
            return true
        end
    end

    return false
end

function Bird:update(dt)
    self.dy = self.dy + GRAVITY * dt

    if love.keyboard.wasPressed('space') or love.mouse.wasPressed(1) then
        self.dy = -5
        sounds['jump']:play()
    end

    self.y = self.y + self.dy
end

function Bird:render()
    love.graphics.draw(self.image, self.x, self.y)
end
```

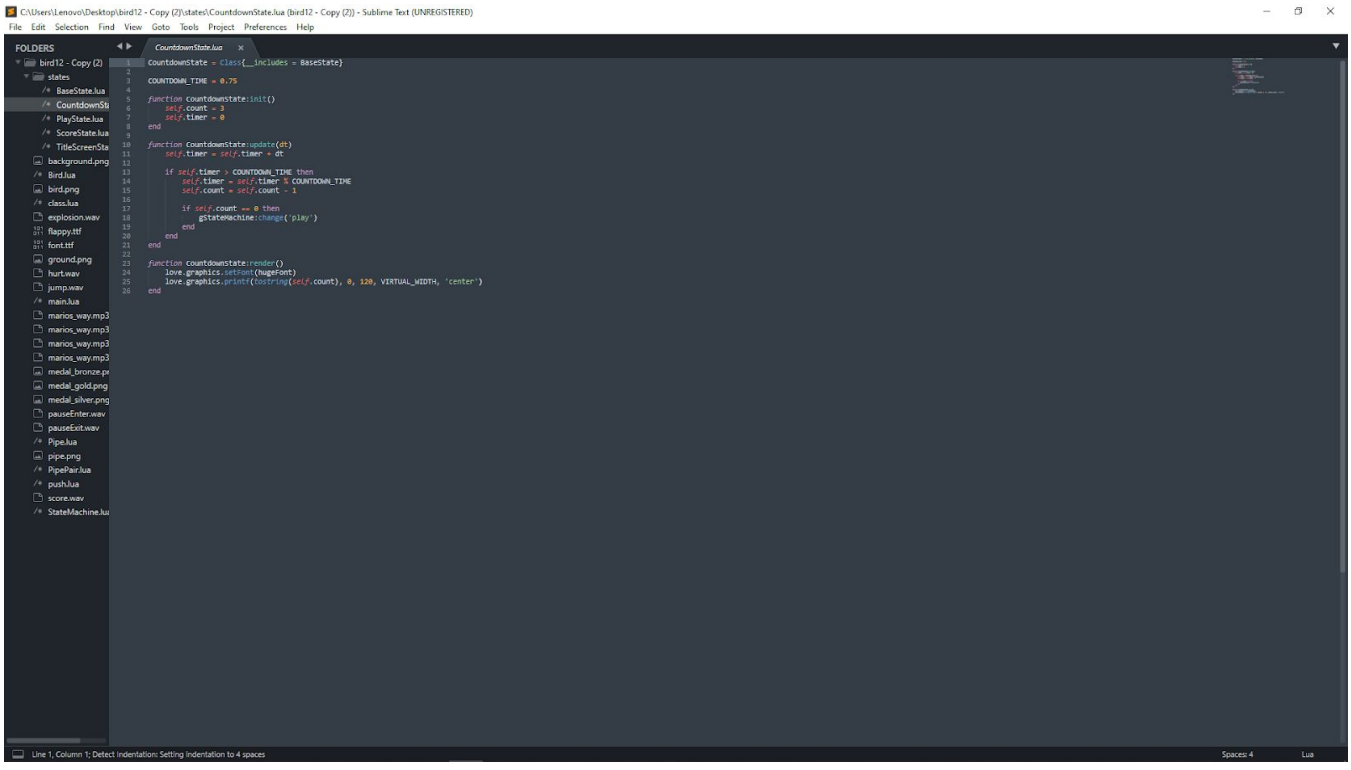
Line 1, Column 1; Detect indentation: Setting indentation to 4 spaces

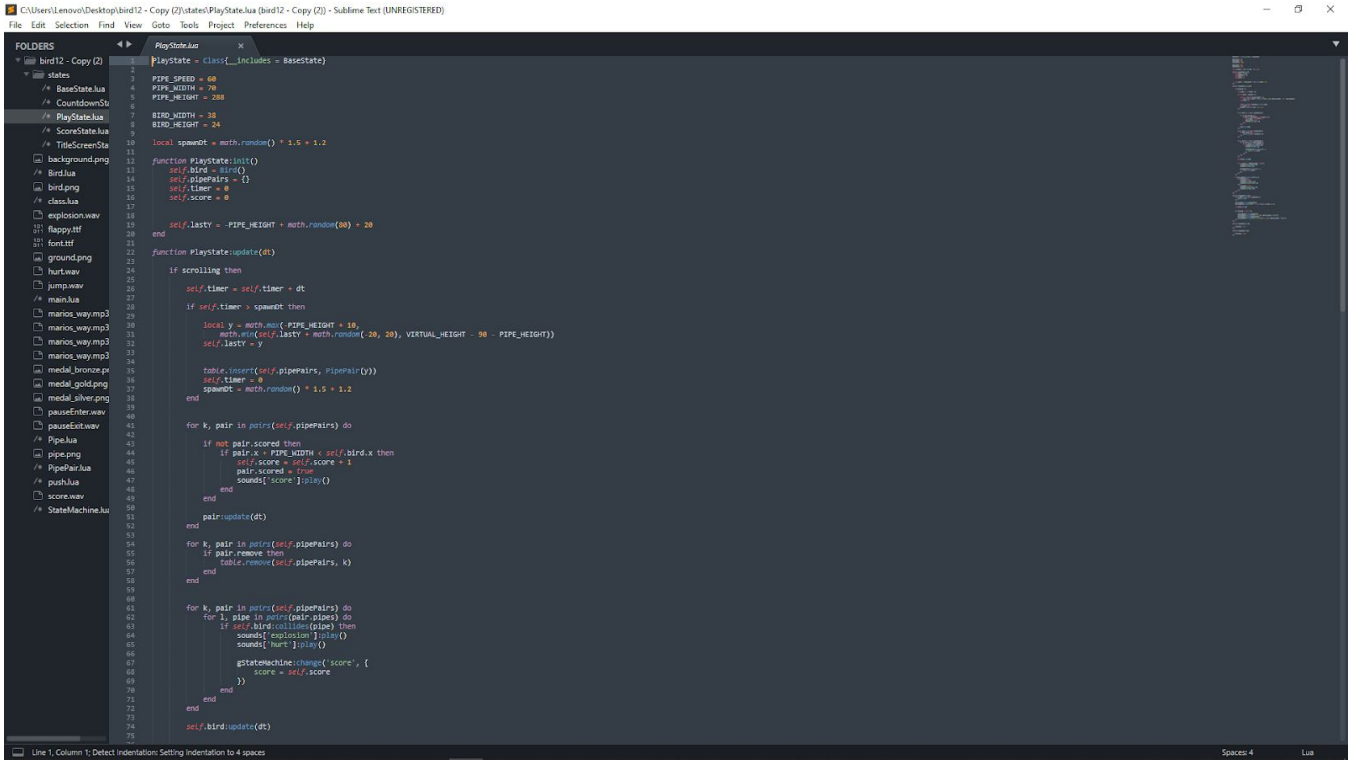
Spaces: 4 Lua

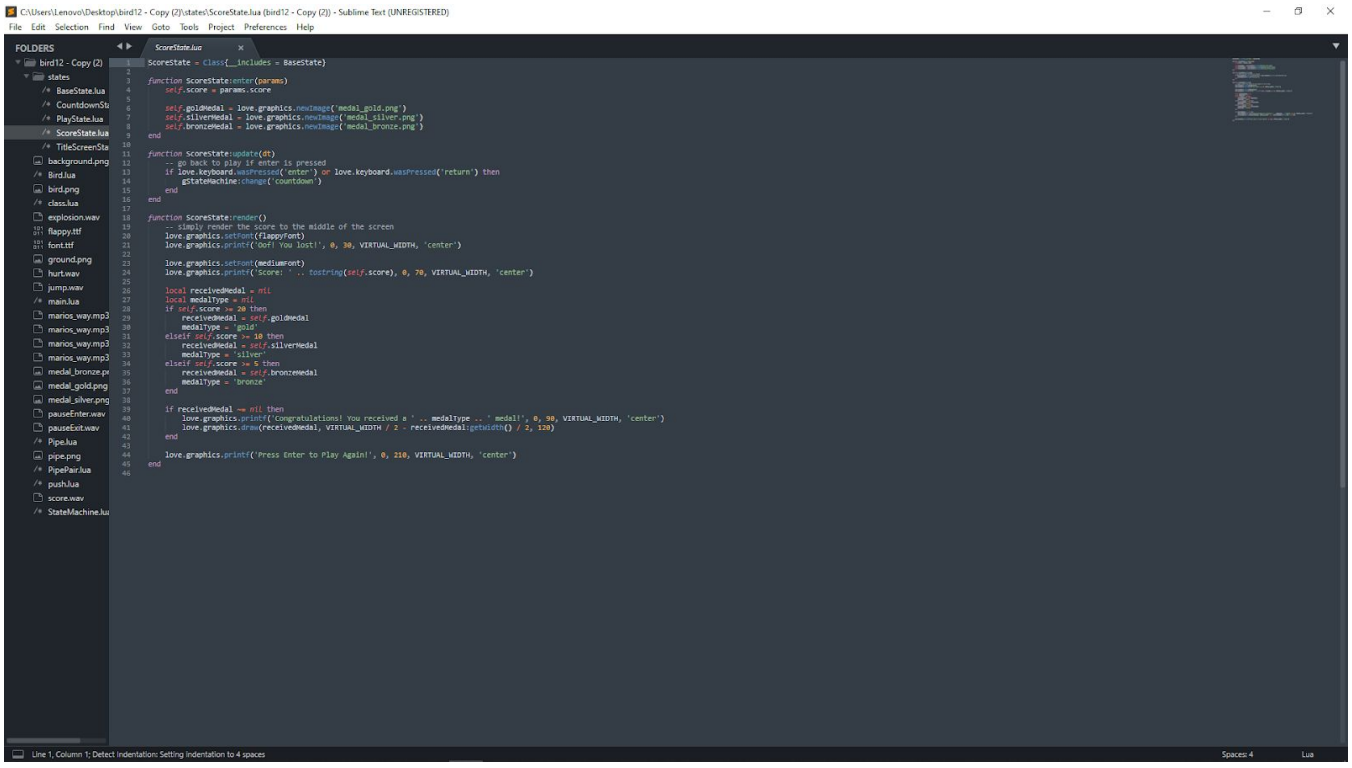

C:\Users\Lenovo\Desktop\bird12 - Copy (2)\class.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
class.lua
local function include_helper(to, from, seen)
if from == nil then
return to
elseif type(from) ~= 'table' then
return from
elseif seen[from] then
return seen[from]
end
seen[from] = to
for k,v in pairs(from) do
k = include_helper({}, k, seen)
if to[k] == nil then
to[k] = include_helper({}, v, seen)
end
end
return to
end
local function include(class, other)
return include_helper(class, other, {})
end
local function clone(other)
return setmetatable(include({}, other), getmetatable(other))
end
local function new(class)
-- mixins
class = class or {}
local inc = class.__includes or {}
if getmetatable(inc) then inc = {inc} end
for _, other in ipairs(inc) do
if type(other) == "string" then
other = _G[other]
end
include(class, other)
end
class.__index = class
class.init = class.init or class[1] or function() end
class.include = class.include or include
class.clone = class.clone or clone
return setmetatable(class, {__call = function(c, ...)
local o = setmetatable({}, c)
o:init(...)
return o
end})
end
if class_commons ~= false and not common then
common = {}
function common.class(name, prototype, parent)
return new{__includes = {prototype, parent}}
end
function common.instance(class, ...)
return class(...)
end
end
return setmetatable({new = new, include = include, clone = clone},
{__call = function(_,...) return new(...) end})
Line 1, Column 1; Detect indentation: Setting indentation to tabs
Tab Size: 4
Lua

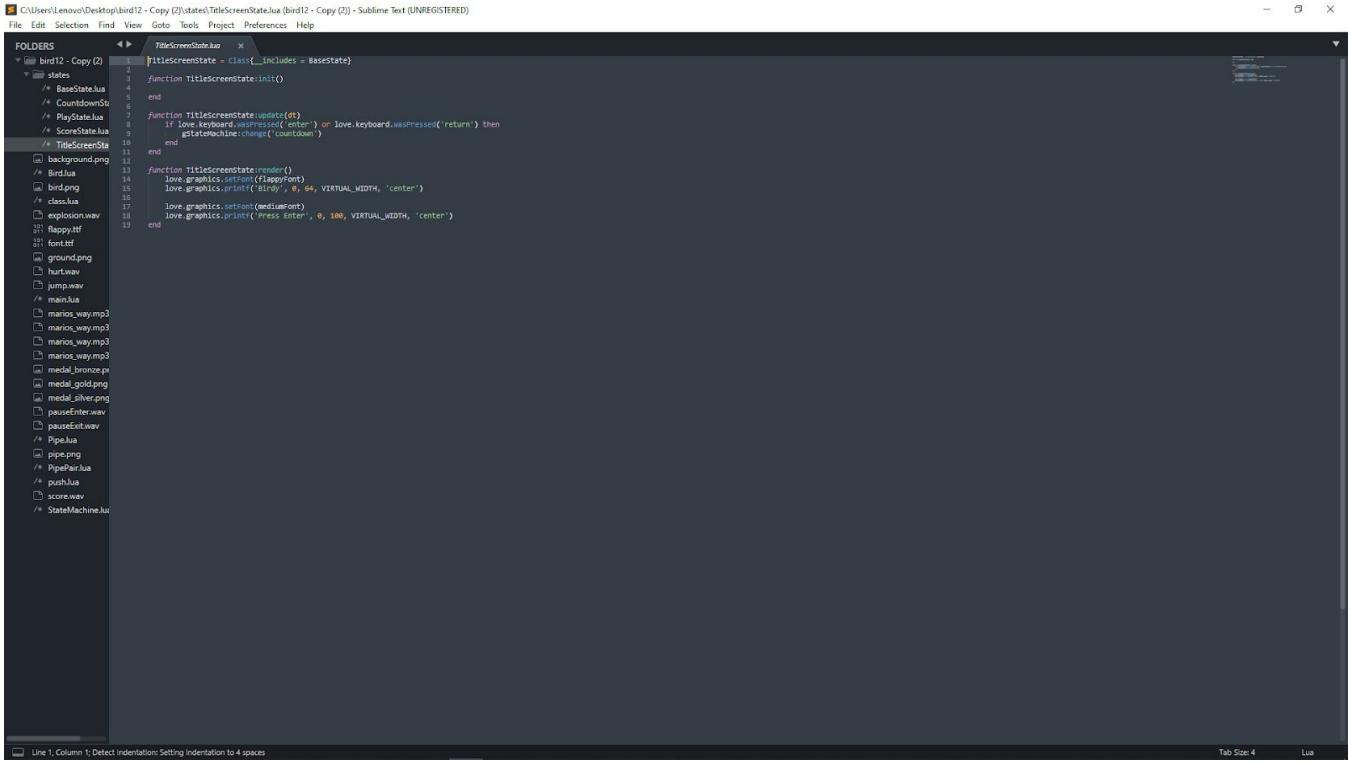

C:\Users\Lenovo\Desktop\bird12 - Copy (2)\StateMachine.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
StateMachine.lua
StateMachine = Class{}
function StateMachine:init(states)
self.empty = {
render = function() end,
update = function() end,
enter = function() end,
exit = function() end
}
self.states = states or {}
self.current = self.empty
end
function StateMachine:change(stateName, enterParams)
assert(self.states[stateName])
self.current:exit()
self.current = self.states[stateName]()
self.current:enter(enterParams)
end
function StateMachine:update(dt)
self.current:update(dt)
end
function StateMachine:render()
self.current:render()
end
Line 1, Column 1; Detect indentation: Setting indentation to tabs
Tab Size: 4
Lua


C:\Users\Lenovo\Desktop\bird12 - Copy (2)\states\BaseState.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
BaseState.lua
BaseState = Class{}
function BaseState:init() end
function BaseState:enter() end
function BaseState:exit() end
function BaseState:update(dt) end
function BaseState:render() end
Line 1, Column 1
Tab Size: 4
Lua


C:\Users\Lenovo\Desktop\bird12 - Copy (2)\states\CountdownState.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
CountdownState = Class{__includes = BaseState}
COUNTDOWN_TIME = 0.75
function countdownstate:init()
self.count = 3
self.timer = 0
end
function CountdownState:update(dt)
self.timer = self.timer + dt
if self.timer > COUNTDOWN_TIME then
self.timer = self.timer % COUNTDOWN_TIME
self.count = self.count - 1
if self.count == 0 then
gStateMachine:change('play')
end
end
end
function countdownstate:render()
love.graphics.setFont(hugeFont)
love.graphics.printf(tostring(self.count), 0, 120, VIRTUAL_WIDTH, 'center')
end
Line 1, Column 1; Detect indentation: Setting indentation to 4 spaces
Spaces: 4
Lua


C:\Users\Lenovo\Desktop\bird12 - Copy (2)\states\PlayState.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
PlayState.lua
PlayState = Class{__includes = BaseState}
PIPE_SPEED = 60
PIPE_WIDTH = 70
PIPE_HEIGHT = 288
BIRD_WIDTH = 38
BIRD_HEIGHT = 24
local spawnDt = math.random() * 1.5 + 1.2
function PlayState:init()
self.bird = Bird()
self.pipePairs = {}
self.timer = 0
self.score = 0
self.lastY = -PIPE_HEIGHT + math.random(80) + 20
end
function PlayState:update(dt)
if scrolling then
self.timer = self.timer + dt
if self.timer > spawnDt then
local y = math.max(-PIPE_HEIGHT + 10,
math.min(self.lastY + math.random(-20, 20), VIRTUAL_HEIGHT - 90 - PIPE_HEIGHT))
self.lastY = y
table.insert(self.pipePairs, PipePair(y))
self.timer = 0
spawnDt = math.random() * 1.5 + 1.2
end
for k, pair in pairs(self.pipePairs) do
if not pair.scored then
if pair.x + PIPE_WIDTH < self.bird.x then
self.score = self.score + 1
pair.scored = true
sounds['score']:play()
end
end
pair:update(dt)
end
for k, pair in pairs(self.pipePairs) do
if pair.remove then
table.remove(self.pipePairs, k)
end
end
for k, pair in pairs(self.pipePairs) do
for l, pipe in pairs(pair.pipes) do
if self.bird:collides(pipe) then
sounds['explosion']:play()
sounds['hurt']:play()
gStateMachine:change('score', {
score = self.score
})
end
end
end
self.bird:update(dt)
Line 1, Column 1; Detect indentation: Setting indentation to 4 spaces
Spaces: 4
Lua


C:\Users\Lenovo\Desktop\bird12 - Copy (2)\states\ScoreState.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
ScoreState.lua
ScoreState = Class{__includes = BaseState}
function ScoreState:enter(params)
self.score = params.score
self.goldMedal = love.graphics.newImage('medal_gold.png')
self.silverMedal = love.graphics.newImage('medal_silver.png')
self.bronzeMedal = love.graphics.newImage('medal_bronze.png')
end
function ScoreState:update(dt)
-- go back to play if enter is pressed
if love.keyboard.wasPressed('enter') or love.keyboard.wasPressed('return') then
gStateMachine:change('countdown')
end
end
function ScoreState:render()
-- simply render the score to the middle of the screen
love.graphics.setFont(flappyFont)
love.graphics.printf('Ooof! You lost!', 0, 30, VIRTUAL_WIDTH, 'center')
love.graphics.setFont(mediumFont)
love.graphics.printf('Score: ' .. tostring(self.score), 0, 70, VIRTUAL_WIDTH, 'center')
local receivedMedal = nil
local medalType = nil
if self.score >= 20 then
receivedMedal = self.goldMedal
medalType = 'gold'
elseif self.score >= 10 then
receivedMedal = self.silverMedal
medalType = 'silver'
elseif self.score >= 5 then
receivedMedal = self.bronzeMedal
medalType = 'bronze'
end
if receivedMedal ~= nil then
love.graphics.printf('Congratulations! You received a ' .. medalType .. ' medal!', 0, 90, VIRTUAL_WIDTH, 'center')
love.graphics.draw(receivedMedal, VIRTUAL_WIDTH / 2 - receivedMedal:getWidth() / 2, 120)
end
love.graphics.printf('Press Enter to Play Again!', 0, 210, VIRTUAL_WIDTH, 'center')
end
Line 1, Column 1; Detect indentation: Setting indentation to 4 spaces
Spaces: 4
Lua


C:\Users\Lenovo\Desktop\bird12 - Copy (2)\states\TitleScreenState.lua (bird12 - Copy (2)) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
bird12 - Copy (2)
states
BaseState.lua
CountdownSta
PlayState.lua
ScoreState.lua
TitleScreenSta
background.png
Bird.lua
bird.png
class.lua
explosion.wav
flappy.ttf
font.ttf
ground.png
hurt.wav
jump.wav
main.lua
marios_way.mp3
marios_way.mp3
marios_way.mp3
marios_way.mp3
medal_bronze.pn
medal_gold.png
medal_silver.png
pauseEnter.wav
pauseExit.wav
Pipe.lua
pipe.png
PipePair.lua
push.lua
score.wav
StateMachine.lua
TitleScreenState.lua
TitleScreenState = Class{__includes = BaseState}
function TitleScreenState:init()
end
function TitleScreenState:update(dt)
if love.keyboard.wasPressed('enter') or love.keyboard.wasPressed('return') then
gStateMachine:change('countdown')
end
end
function TitleScreenState:render()
love.graphics.setFont(flappyFont)
love.graphics.printf('Birdy', 0, 64, VIRTUAL_WIDTH, 'center')
love.graphics.setFont(mediumFont)
love.graphics.printf('Press Enter', 0, 100, VIRTUAL_WIDTH, 'center')
end
Line 1, Column 1; Detect indentation: Setting indentation to 4 spaces
Tab Size: 4
Lua

References

https://www.lua.org/

https://www.tutorialspoint.com/lua/index.htm

https://love2d.org/

https://www.sublimetext.com/3

https://www.wikipedia.org/