Lua Game Programming Assignment

Time Limit: 1 hour 30 minutes maximum
Points: 10 + 2 bonus
Focus: Core game patterns with modern Lua features

Assignment Overview

Create a simple **Snake Game** using modern Lua patterns. This focused assignment demonstrates key game programming concepts.

Part 1: Basic Game Structure

Task 1.1: Game State Manager

Create a simple state manager with two states: `Menu` and `Playing`.

```
-- gamestate.lua
local GameState = {}
GameState.__index = GameState

function GameState:new()
    local state = {
        current = "menu",
        states = {
            menu = require("menu_state"),
            playing = require("game_state")
        }
    }
    return setmetatable(state, GameState)
end

function GameState:switch(newState)
    -- TODO: Implement state switching
```

```
end

function GameState:update(dt)
    -- TODO: Update current state
end

function GameState:draw()
    -- TODO: Draw current state
end

return GameState
```

Requirements:

- Switch between menu and playing states
- Each state has `update()` and `draw()` methods
- Use proper module structure

Task 1.2: Entity Component Pattern

Create a simple entity system for the snake.

```
-- entity.lua
local Entity = {}
Entity.__index = Entity

function Entity:new()
    local entity = {
        components = {}
    }
    return setmetatable(entity, Entity)
end

function Entity:addComponent(name, component)
    -- TODO: Add component to entity
end

function Entity:getComponent(name)
    -- TODO: Get component from entity
end

function Entity:hasComponent(name)
    -- TODO: Check if entity has component
end
```

return Entity

Components to create:

- `Position` (x, y)
- `Movement` (dx, dy)
- `Renderer` (color, size)

Part 2: Modern Lua Features

Task 2.1: Resource Management with to-be-closed

Implement a simple file-based high score system using Lua 5.4's `to-be-closed` variables.

```
-- highscore.lua
local HighScore = {}

function HighScore:save(score)
    local file <close> = io.open("highscore.txt", "w")
    if file then
        file:write(tostring(score))
        -- File automatically closes even if error occurs
    end
end

function HighScore:load()
    local file <close> = io.open("highscore.txt", "r")
    if file then
        local content = file:read("*a")
        return tonumber(content) or 0
    end
    return 0
end

return HighScore
```

Task 2.2: Custom Metamethods

Create a `Vector2` class with metamethods for game math.

```
-- vector2.lua
local Vector2 = {}
Vector2.__index = Vector2

function Vector2:new(x, y)
    return setmetatable({x = x or 0, y = y or 0}, Vector2)
end

-- TODO: Implement these metamethods
function Vector2:__add(other)
    -- Vector addition
end

function Vector2:__eq(other)
    -- Vector equality
end

function Vector2:__tostring()
    -- String representation
end

return Vector2
```

Part 3: Snake Game Implementation

Task 3.1: Core Game Logic

Implement the snake game mechanics:

```
-- snake_game.lua
local SnakeGame = {}
SnakeGame.__index = SnakeGame

function SnakeGame:new()
    local game = {
        snake = {
            {x = 10, y = 10} -- Starting position
        },
        direction = {x = 1, y = 0},
        food = {x = 15, y = 15},
```

```
        score = 0,
        gameOver = false
    }
    return setmetatable(game, SnakeGame)
end

function SnakeGame:update(dt)
    -- TODO: Update snake position
    -- TODO: Check collisions
    -- TODO: Handle food eating
end

function SnakeGame:draw()
    -- TODO: Draw snake, food, and score
end

function SnakeGame:changeDirection(dx, dy)
    -- TODO: Change snake direction
end

return SnakeGame
```

Required Features:

- Snake moves continuously
- Food spawns randomly when eaten
- Score increases when food is eaten
- Game ends on wall/self collision

Task 3.2: Input Handling

Create a simple input system using the command pattern.

```
-- input.lua
local Input = {}
Input.__index = Input

function Input:new()
    local input = {
        commands = {}
    }
    return setmetatable(input, Input)
end
```

```
function Input:bind(key, command)
    self.commands[key] = command
end

function Input:handleKey(key)
    local command = self.commands[key]
    if command then
        command()
    end
end

return Input
```

Bonus Challenge: Event System

Simple Observer Pattern

Implement a basic event system for game events.

```
-- events.lua
local Events = {}

local listeners = {}

function Events:subscribe(event, callback)
    -- TODO: Add callback to event listeners
end

function Events:emit(event, data)
    -- TODO: Call all listeners for this event
end

return Events
```

Usage Example:

```
Events:subscribe("food_eaten", function(data)
    print("Score increased to:", data.score)
end)

Events:emit("food_eaten", {score = 10})
```

Project Structure (Required)

```
snake_game/
├── main.lua             -- Entry point
├── gamestate.lua        -- State manager
├── entity.lua           -- Entity system
├── vector2.lua          -- Math utilities
├── highscore.lua        -- Save/load system
├── input.lua            -- Input handling
├── events.lua           -- Event system (bonus)
├── states/
│   ├── menu_state.lua   -- Menu state
│   └── game_state.lua   -- Playing state
└── snake_game.lua       -- Core game logic
```

Starter Code Template

```
-- main.lua
local GameState = require("gamestate")

local gameState = GameState:new()

function love.update(dt)
   gameState:update(dt)
end

function love.draw()
   gameState:draw()
end

function love.keypressed(key)
   -- Handle input
   if gameState.current == "playing" then
      local game = gameState.states.playing
      if key == "up" then game:changeDirection(0, -1) end
      if key == "down" then game:changeDirection(0, 1) end
      if key == "left" then game:changeDirection(-1, 0) end
      if key == "right" then game:changeDirection(1, 0) end
   end
```

end

Bonus (2 points)

- **Event System:** Working observer pattern implementation

Submission Requirements

Must Include:

1. All source files in correct structure
2. Working game (run with `love .` or `lua main.lua`)
3. Brief comment explaining one modern Lua feature used
4. Github repo link all source files

Quick Test Checklist:

- [ ] Game starts with menu
- [ ] Snake moves and can change direction
- [ ] Food spawns and increases score
- [ ] High score saves/loads
- [ ] Vector2 math operations work
- [ ] Game over detection works
