Table of Contents

Acknowledgments

Big shout out to Ryan Heath for researching and implementing all of the awesome tech that makes all of this Lua integration into our engine possible. None of this would be possible without all of Ryan's hard work, and I would have never learned all of this without his help. Thank you, Ryan!

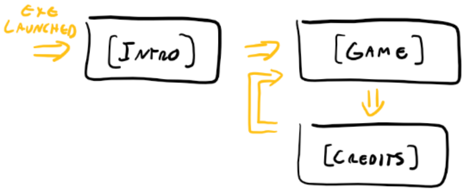
Terminology

Scene – A collection of objects saved as a .json file using the Grool Editor

State – A set of functions in a script that determines what happens from the start of the state, during the state, and at the end of the state.

Scenes Overview

Currently our game is broken up into 3 scenes. The Intro scene, the Credits scene, and the Game scene. The Intro scene is the first scene to be loaded when launching the game. At the end of the intro animation, the Game scene is loaded. From the pause menu in the Game scene, the Credits scene can be loaded. The Credits scene then goes back to the Game scene.



Intro Scene

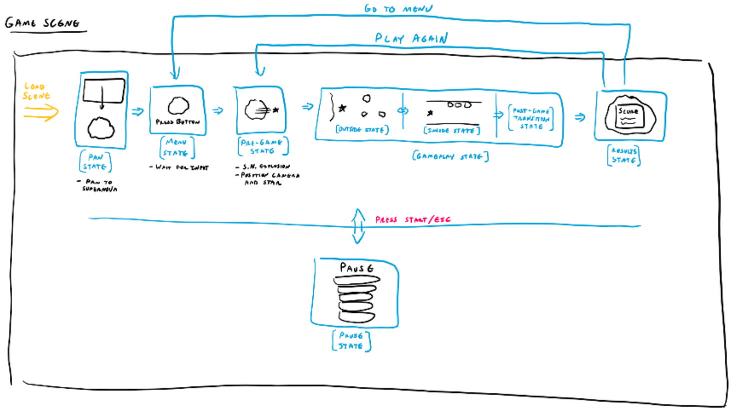
The Intro scene cycles through the logos for DigiPen, FMOD, and team Critical Orbit

Credits Scene

The Credits scene is the credits scene. That's it.

Game Scene

The Game scene contains the bulk of the actual game. It is broken up into multiple states which can be seen in the following image. This will be explained in more detail in the following section.



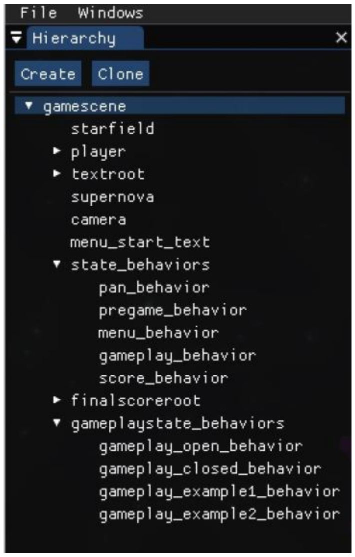
The Game Scene

A scene is just a collection of objects placed in the editor, and saved to a .json file. The Game scene contains all of the objects that will be needed to run the various states that make up the main flow of the game.

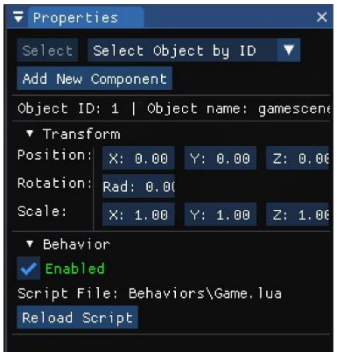
When the scenes is first loaded, the camera pans down to where the main menu state takes place. We then switch to the Menu state which awaits input from the user to start the game. When input is detected, we switch to the PreGame state that plays the animation of the supernova exploding, launching the player star into position for gameplay. We then switch to the Gameplay state, which is comprised of sub states that represent different sections of gameplay (like the open and closed sections of Infinite Supernova). When the gameplay ends, the state switches to the results screen. At this point the user can switch the state back to the menu, or back to the PreGame state to start the game over again. At any point during these states, the pause menu can be brought up.

The Current State of the Game Scene

Currently these are the objects that the Game scene contains. The root object of the scene contains a behavior component that has the file Game.lua attached to it. This file contains the state machine and essentially runs the entire game state through its Update loop.



```
▾ gamescene
    starfield
  ▸ player
  ▸ textroot
    supernova
    camera
    menu_start_text
  ▾ state_behaviors
      pan_behavior
      pregame_behavior
      menu_behavior
      gameplay_behavior
      score_behavior
  ▸ finalscoreroot
  ▾ gameplaystate_behaviors
      gameplay_open_behavior
      gameplay_closed_behavior
      gameplay_example1_behavior
      gameplay_example2_behavior
```



Object ID: 1 | Object name: gamescene

▾ Transform

Position: X: 0.00 Y: 0.00 Z: 0.0

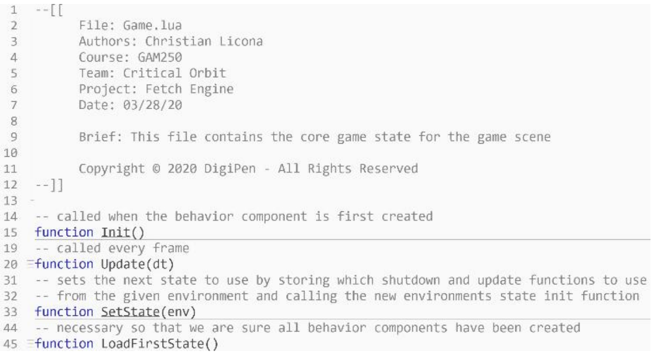
Rotation: Rad: 0.0

Scale: X: 1.00 Y: 1.00 Z: 1.0

▾ Behavior

☑ Enabled

Script File: Behaviors\Game.lua



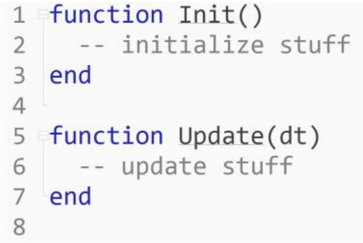
```lua
--[[
      File: Game.lua
      Authors: Christian Licona
      Course: GAM250
      Team: Critical Orbit
      Project: Fetch Engine
      Date: 03/28/20

      Brief: This file contains the core game state for the game scene

      Copyright © 2020 DigiPen - All Rights Reserved
--]]

-- called when the behavior component is first created
function Init()
-- called every frame
function Update(dt)
-- sets the next state to use by storing which shutdown and update functions to use
-- from the given environment and calling the new environments state init function
function SetState(env)
-- necessary so that we are sure all behavior components have been created
function LoadFirstState()
```

The State Machine Preview

The state machine is nothing more than a simple function, SetState(), that changes a few variables pointing to which Init, Update, and Shutdown functions to call. This will be explained in more detail in section: The State Machine.

Behavior Functions

As a refresher and reminder, these are two of the main functions that you have access to as part of a Lua script being attached to a behavior component.



```lua
function Init()
  -- initialize stuff
end

function Update(dt)
  -- update stuff
end

```

Init()

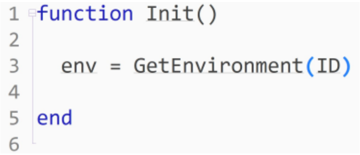
The init function gets called when the behavior component for an object is first created. This means that when the Game scene is first loaded from either the Intro scene or the Credits scene, the init function will be called for all of the objects in the Game scene containing behaviors with Lua scripts attached to them. This also means that any “scene” that is instanced in during gameplay with InstanceScene() will have the init function called for any objects containing behavior components from that instanced scene.

Update()

For any behavior component in the scene, the Update() function will be called every frame.

GetEnvironment()

This function retrieves the Lua environment from the behavior component at the given ID. This is similar to getting the Transform component from an object at a given ID.

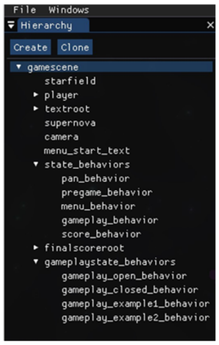


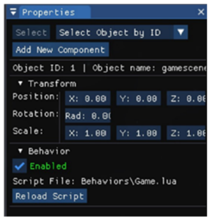
```lua
function Init()

  env = GetEnvironment(ID)

end

```

So what?

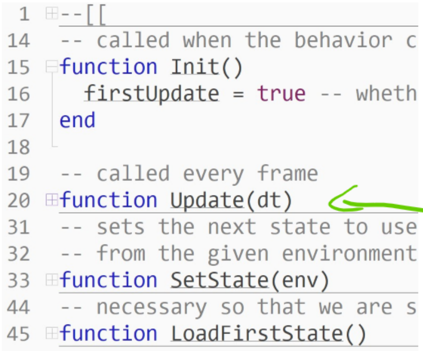
This means that you can have the Lua script attached to a certain behavior component in game, access the variables and functions from a Lua script attached to another behavior component in game.

Remember that the gamescene root object has a behavior component with the Game.lua script attached to it?





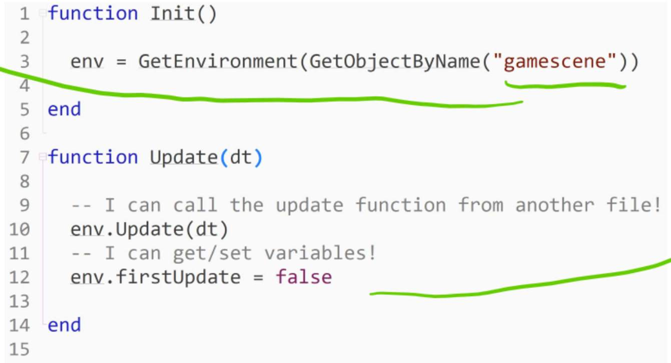
```
Object ID: 1 | Object name: gamescene
▼ Transform
Position:  X: 0.00  Y: 0.00  Z: 0.00
Rotation:  Rad: 0.00
Scale:     X: 1.00  Y: 1.00  Z: 1.00
▼ Behavior
☑ Enabled
Script File: Behaviors\Game.lua
```



```lua
--[[
-- called when the behavior c
function Init()
  firstUpdate = true -- wheth
end

-- called every frame
function Update(dt)
-- sets the next state to use
-- from the given environment
function SetState(env)
-- necessary so that we are s
function LoadFirstState()
```

Well by getting the ID of the gamescene object, you can then get the lua environment from that behavior component, which will give you access to the functions and variables you have created in the lua file script.

Imagine I have attached this script to one of the other objects in the hierarchy list above.



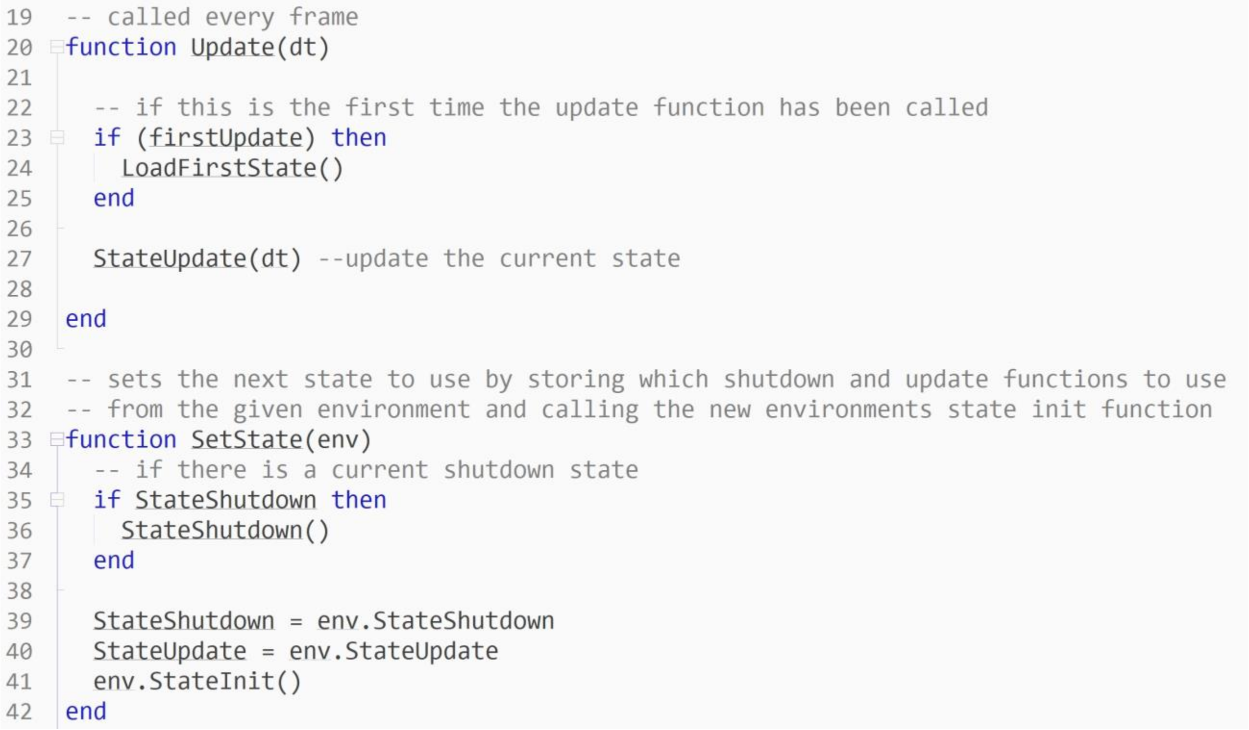
```lua
function Init()

  env = GetEnvironment(GetObjectByName("gamescene"))

end

function Update(dt)

  -- I can call the update function from another file!
  env.Update(dt)
  -- I can get/set variables!
  env.firstUpdate = false

end

```

The State Machine

The state machine handles switching the state of the Game scene between all of the different states: PanState, MenuState, PreGameState, GameplayState, and ScoreState.

Game.lua

This script is attached to the “gamescene” root object of the Game scene. It is through this script that most of the rest of the Game scene is ran. The Update() function in this file is called as being part of a behavior component. Inside the Update() function, it calls the StateUpdate() function of whatever state is the current state.

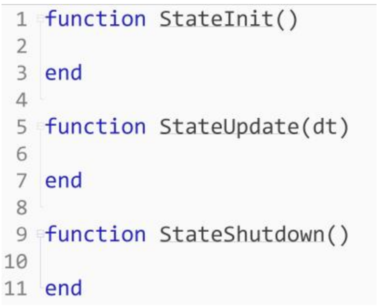


```lua
-- called every frame
function Update(dt)

 -- if this is the first time the update function has been called
 if (firstUpdate) then
  LoadFirstState()
 end

 StateUpdate(dt) --update the current state

end

-- sets the next state to use by storing which shutdown and update functions to use
-- from the given environment and calling the new environments state init function
function SetState(env)
 -- if there is a current shutdown state
 if StateShutdown then
  StateShutdown()
 end

 StateShutdown = env.StateShutdown
 StateUpdate = env.StateUpdate
 env.StateInit()
end
```

If you are new to Lua, it might not be obvious at first glance, but StateShutdown and StateUpdate are global variables declared in this file. We are using them to store which functions to call.

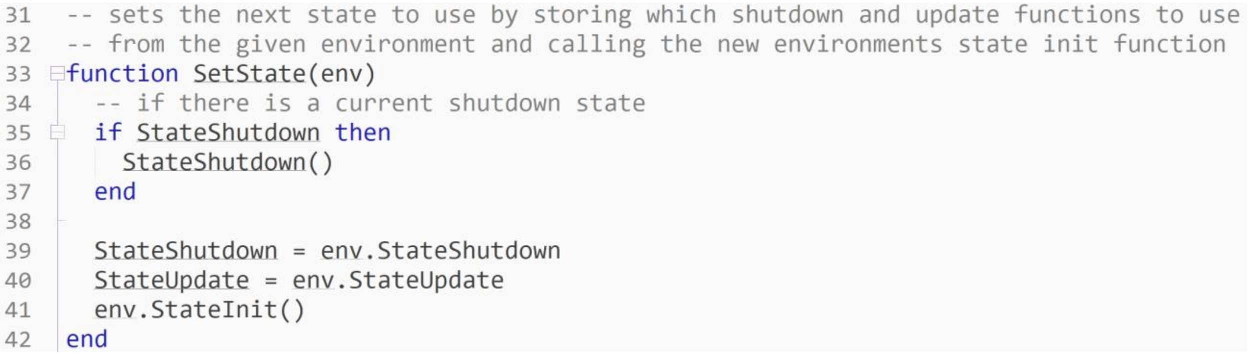
Basic State Functions

When you create a Lua file that is going to be used as a state that is part of the game scene, it must declare these three functions:



```lua
function StateInit()

end

function StateUpdate(dt)

end

function StateShutdown()

end
```

SetState()



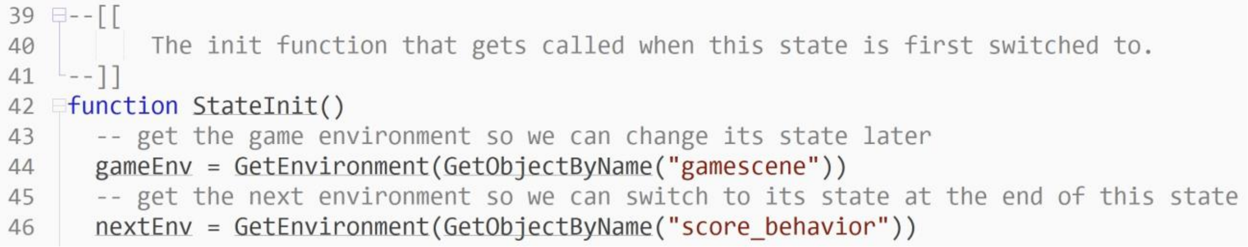
```lua
-- sets the next state to use by storing which shutdown and update functions to use
-- from the given environment and calling the new environments state init function
function SetState(env)
  -- if there is a current shutdown state
  if StateShutdown then
    StateShutdown()
  end

  StateShutdown = env.StateShutdown
  StateUpdate = env.StateUpdate
  env.StateInit()
end
```

This function is the heart of the state machine. When you call this function, you must pass in the Lua environment of a behavior that has a Lua script attached to it with the proper State functions declared. If any of the state functions, StateInit(), StateUpdate(), and StateShutdown(), are missing, you will certainly get an error.

The function first checks if the StateShutdown variable has been set and given a shutdown function to call in another state. If it has, it calls that shutdown function. The shutdown and update function from the passed in environment are then stored in the variables: StateShutdown and StateUpdate. We don't need to store the init function, so we just call it directly from here.

Calling SetState()

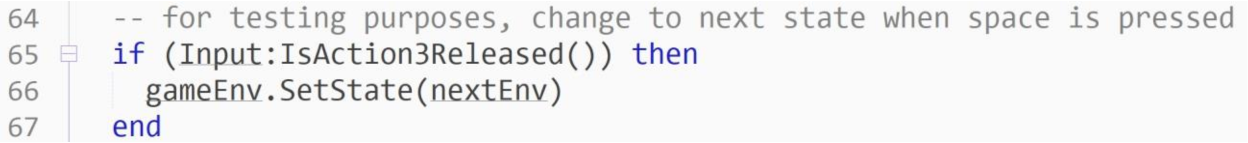
We want to be able to call the SetState() function from within our states so that we can tell the state machine to switch to the next state. To do this we will make use of GetEnviroment(). We will use the Gameplay state as an example.



```lua
--[[
    The init function that gets called when this state is first switched to.
--]]
function StateInit()
  -- get the game environment so we can change its state later
  gameEnv = GetEnvironment(GetObjectByName("gamescene"))
  -- get the next environment so we can switch to its state at the end of this state
  nextEnv = GetEnvironment(GetObjectByName("score_behavior"))
```

One of the first things you should do when your state initializes is to store a reference to the environment of the Game scene root which contains the state machine, and the environment of the next state you want to switch to after this state. (you don't HAVE to determine which scene to switch to now, you can do that later. In this case we know we want to switch to the scoreboard state after the gameplay state).

When its time to switch states, we'll call SetState() on gameEnv and pass in the environment of the next state.

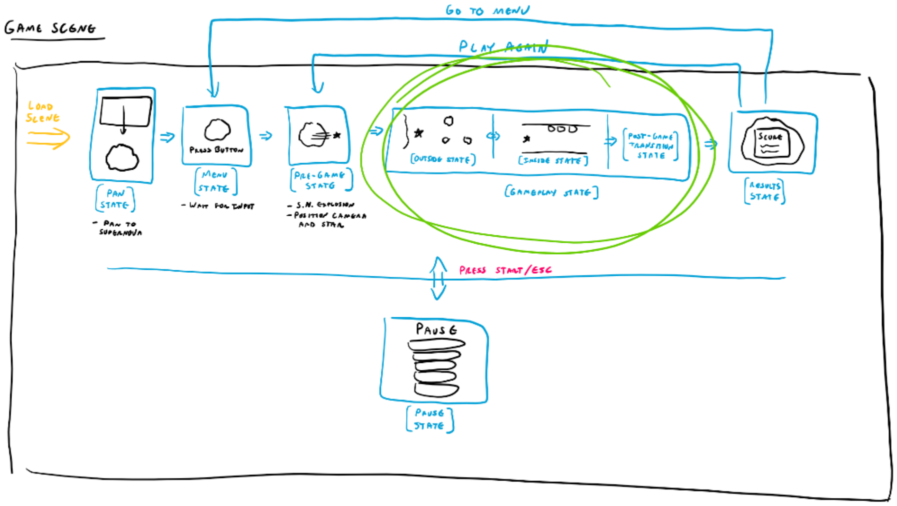


```lua
  -- for testing purposes, change to next state when space is pressed
  if (Input:IsAction3Released()) then
    gameEnv.SetState(nextEnv)
  end
```

For example, here I am switching states when I press the space bar.

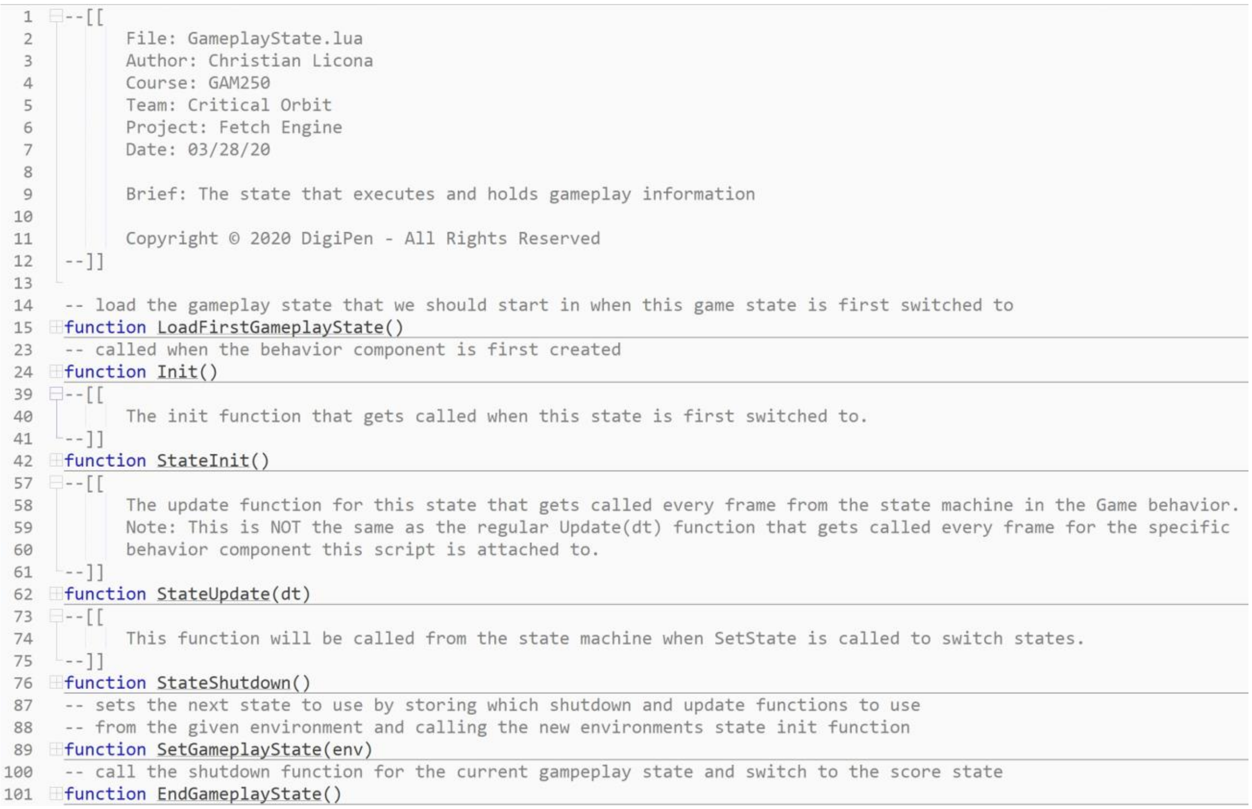
The Gameplay State

As we have seen, the Gameplay state is one of the states as part of the Game scene. However, we plan on having different sections of gameplay as part of the Gameplay state. So the Gameplay state itself has been broken up into separate smaller gameplay states, and thus has its own state machine to handle switching between these sections.



GameplayState.lua

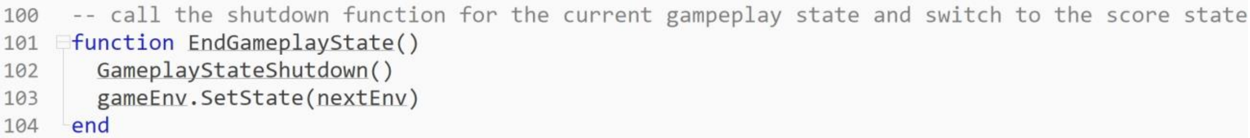
Remember that the Gameplay state is part of the Game state machine. Thus this file has the functions StateInit(), StateUpdate(), and StateShutdown(), which will be called by the state machine in Game.lua.



```
--[[
        File: GameplayState.lua
        Author: Christian Licona
        Course: GAM250
        Team: Critical Orbit
        Project: Fetch Engine
        Date: 03/28/20

        Brief: The state that executes and holds gameplay information

        Copyright © 2020 DigiPen - All Rights Reserved
--]]

-- load the gameplay state that we should start in when this game state is first switched to
function LoadFirstGameplayState()
-- called when the behavior component is first created
function Init()
--[[
        The init function that gets called when this state is first switched to.
--]]
function StateInit()
--[[
        The update function for this state that gets called every frame from the state machine in the Game behavior.
        Note: This is NOT the same as the regular Update(dt) function that gets called every frame for the specific
        behavior component this script is attached to.
--]]
function StateUpdate(dt)
--[[
        This function will be called from the state machine when SetState is called to switch states.
--]]
function StateShutdown()
-- sets the next state to use by storing which shutdown and update functions to use
-- from the given environment and calling the new environments state init function
function SetGameplayState(env)
-- call the shutdown function for the current gampeplay state and switch to the score state
function EndGameplayState()
```

Functions

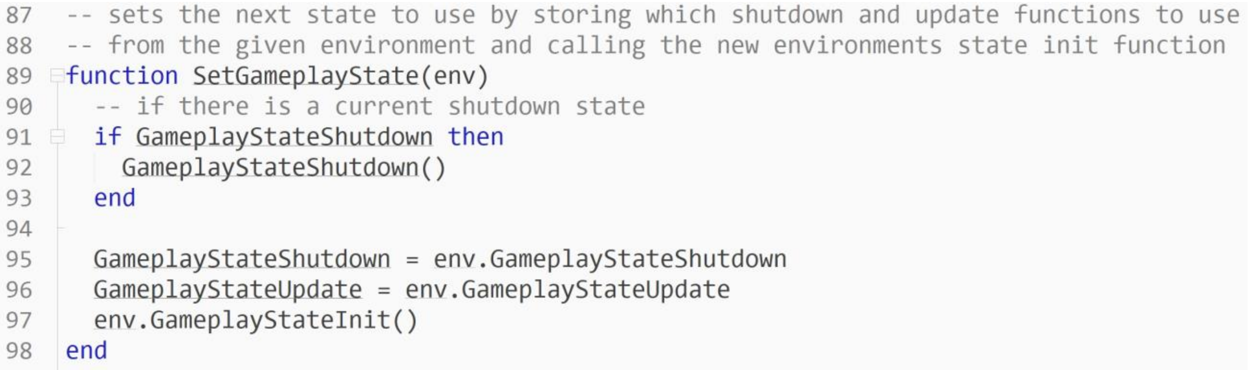
The Gameplay state has one notable function other than SetGameplayState() that may be useful to call from within a gameplay section.



```lua
-- call the shutdown function for the current gampeplay state and switch to the score state
function EndGameplayState()
  GameplayStateShutdown()
  gameEnv.SetState(nextEnv)
end
```

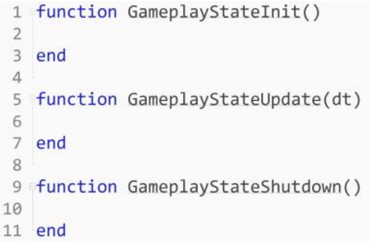
EndGameplayState() will call the shutdown function for the current gameplay section. It will then tell the Game state machine to move on to whatever state "nextEnv" is.

SetGameplayState()



```lua
-- sets the next state to use by storing which shutdown and update functions to use
-- from the given environment and calling the new environments state init function
function SetGameplayState(env)
  -- if there is a current shutdown state
  if GameplayStateShutdown then
    GameplayStateShutdown()
  end

  GameplayStateShutdown = env.GameplayStateShutdown
  GameplayStateUpdate = env.GameplayStateUpdate
  env.GameplayStateInit()
end
```

As you can see, the Gameplay state has a state machine nearly identical to the Game state machine. The only difference being that the functions it calls are prepended with GameplayState instead of just State. This of course means that any gameplay states that should be considered a part of the Gameplay state machine should have these functions:

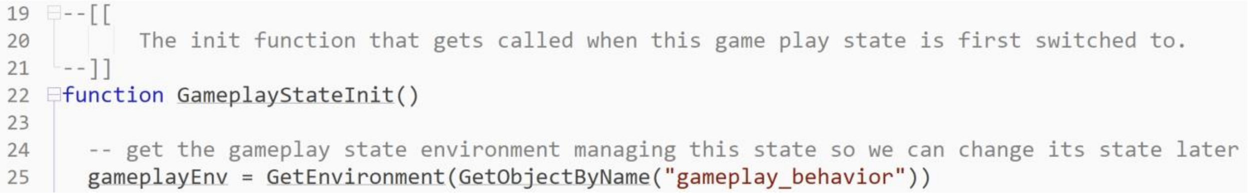


```lua
function GameplayStateInit()

end

function GameplayStateUpdate(dt)

end

function GameplayStateShutdown()

end
```

Calling SetGameplayState()

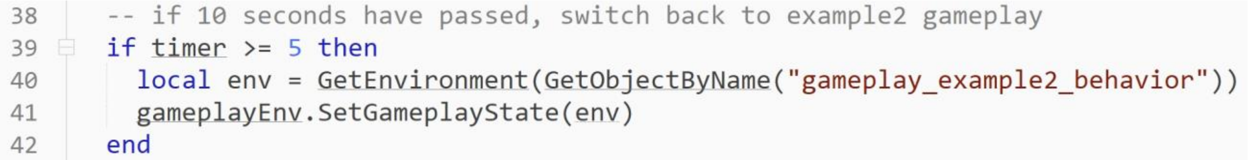
This process will be similar to calling SetState() for the Game scene.

We'll use GameplayState_Example1.lua as an example. In this example we will store a reference to the Gameplay state environment.



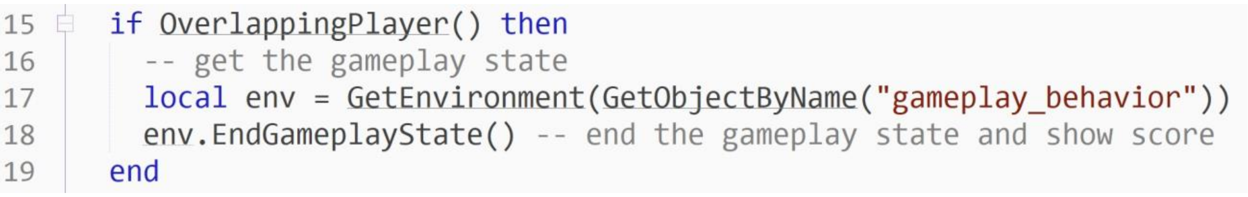
```lua
--[[
      The init function that gets called when this game play state is first switched to.
--]]
function GameplayStateInit()

  -- get the gameplay state environment managing this state so we can change its state later
  gameplayEnv = GetEnvironment(GetObjectByName("gameplay_behavior"))
```

Then, when its time to switch to another gameplay section, we'll tell the Gameplay state to switch to the section we want.



```lua
    -- if 10 seconds have passed, switch back to example2 gameplay
    if timer >= 5 then
      local env = GetEnvironment(GetObjectByName("gameplay_example2_behavior"))
      gameplayEnv.SetGameplayState(env)
    end
```

Or maybe we want to end the Gameplay state instead.



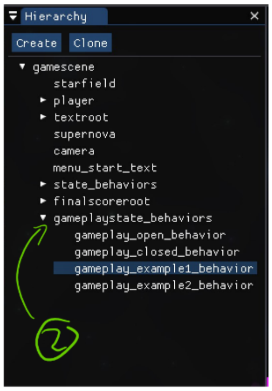
```lua
    if OverlappingPlayer() then
      -- get the gameplay state
      local env = GetEnvironment(GetObjectByName("gameplay_behavior"))
      env.EndGameplayState() -- end the gameplay state and show score
    end
```

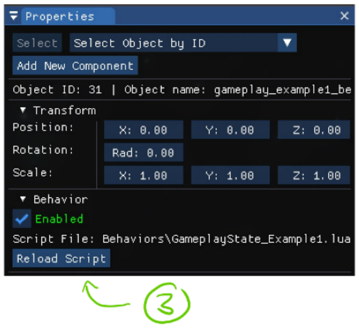
This is a snippet from Example1Enemy.lua. When the enemy overlaps the player, it tells the Gameplay state to end. The Gameplay state is handling which Game state it will switch to, which in our case is the Scoreboard. (however you could make the Game state switch to whatever state you wanted yourself. Do that at your own risk though)

GameplayState_STATENAME.lua

If you want to create a new gameplay section for the Gameplay state, you will need to follow these steps:

1. Make a copy of the GameplayState_TEMPLATE.lua file. Rename it to whatever your sections is called.
2. Create a new object in the Game scene with a behavior component. Please parent it to the gameplaystate_behaviors object for organization purposes.
3. Attach your state script to the object.





Object ID: 31 | Object name: gameplay_example1_be

Transform

Position: X: 0.00 Y: 0.00 Z: 0.00

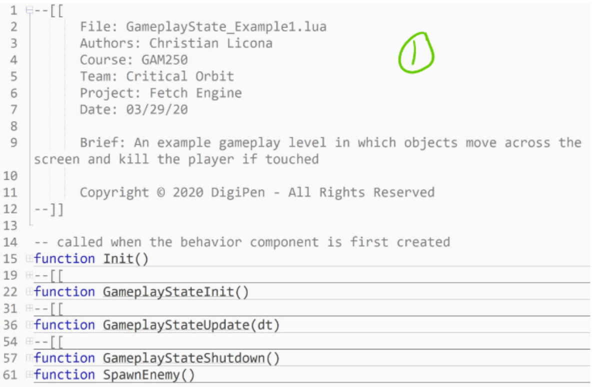
Rotation: Rad: 0.00

Scale: X: 1.00 Y: 1.00 Z: 1.00

Behavior

☑ Enabled

Script File: Behaviors\GameplayState_Example1.lua



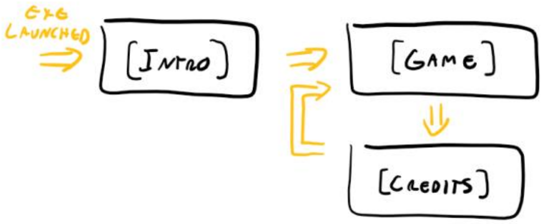
```lua
--[[
      File: GameplayState_Example1.lua
      Authors: Christian Licona
      Course: GAM250
      Team: Critical Orbit
      Project: Fetch Engine
      Date: 03/29/20

      Brief: An example gameplay level in which objects move across the screen and kill the player if touched

      Copyright © 2020 DigiPen - All Rights Reserved
--]]

-- called when the behavior component is first created
function Init()
--[[
function GameplayStateInit()
--[[
function GameplayStateUpdate(dt)
--[[
function GameplayStateShutdown()
function SpawnEnemy()
```

GameplayState_TEMPLATE.lua

This file already contains the state functions it should contain, as well as a premade header. The init function also already stores a reference to the Gameplay states environment so you can tell the gameplay state to switch sections from your section.

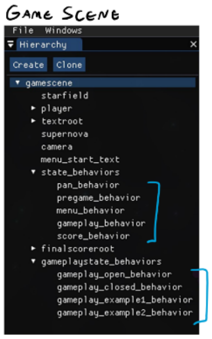
TLDR

Game Scenes Diagram

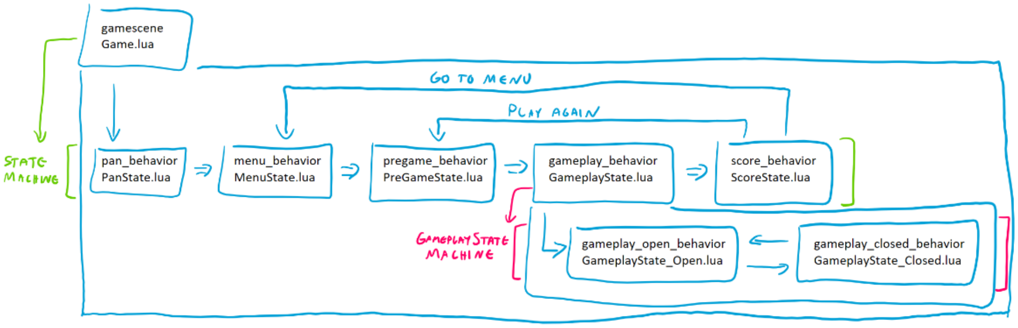


Game Scene Diagram

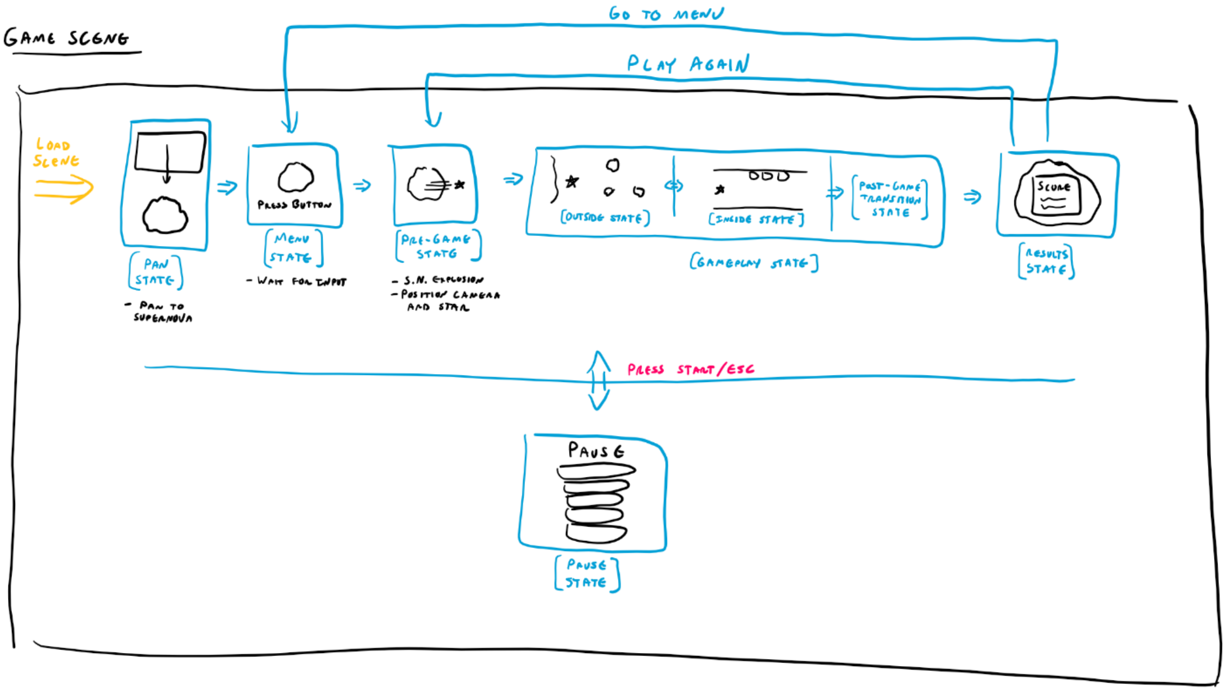
This is a view of the game scene with object names and the lua files that represent the states for each of those objects.



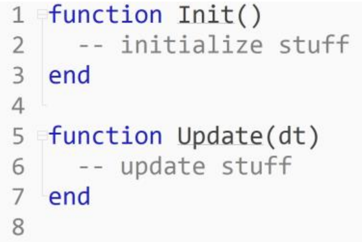
```
▼ gamescene
    starfield
  ▸ player
  ▸ textroot
    supernova
    camera
    menu_start_text
  ▼ state_behaviors
      pan_behavior
      pregame_behavior
      menu_behavior
      gameplay_behavior
      score_behavior
  ▸ finalscoreroot
  ▼ gameplaystate_behaviors
      gameplay_open_behavior
      gameplay_closed_behavior
      gameplay_example1_behavior
      gameplay_example2_behavior
```



Game State Diagram

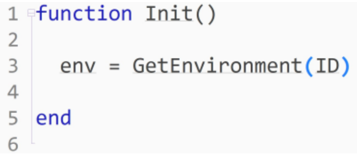


Behavior Functions

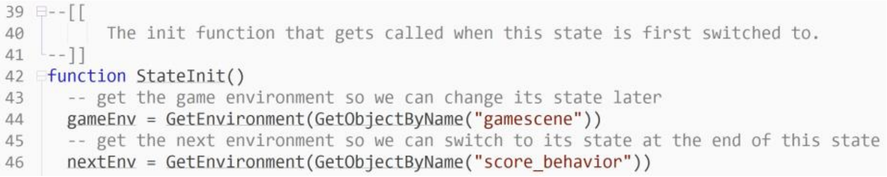


```lua
function Init()
  -- initialize stuff
end

function Update(dt)
  -- update stuff
end

```

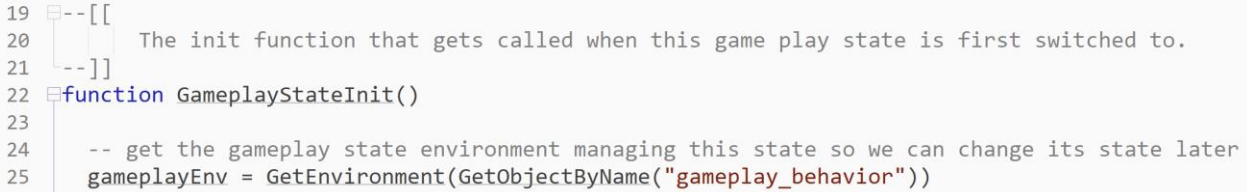
GetEnvironment()



```lua
function Init()

  env = GetEnvironment(ID)

end

```

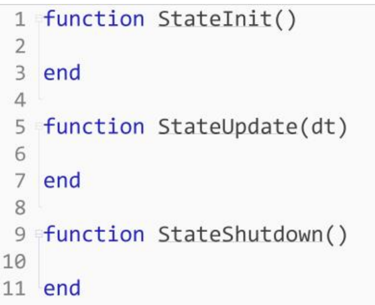


```lua
--[[
    The init function that gets called when this state is first switched to.
--]]
function StateInit()
  -- get the game environment so we can change its state later
  gameEnv = GetEnvironment(GetObjectByName("gamescene"))
  -- get the next environment so we can switch to its state at the end of this state
  nextEnv = GetEnvironment(GetObjectByName("score_behavior"))
```

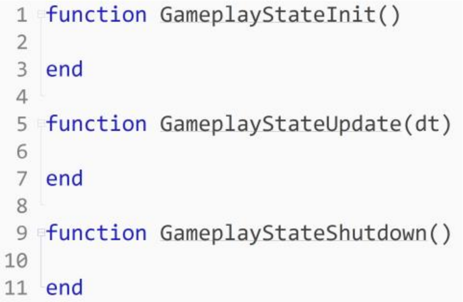


```lua
--[[
    The init function that gets called when this game play state is first switched to.
--]]
function GameplayStateInit()

  -- get the gameplay state environment managing this state so we can change its state later
  gameplayEnv = GetEnvironment(GetObjectByName("gameplay_behavior"))
```

State Functions / Gameplay State Functions



```lua
function StateInit()

end

function StateUpdate(dt)

end

function StateShutdown()

end
```



```lua
function GameplayStateInit()

end

function GameplayStateUpdate(dt)

end

function GameplayStateShutdown()

end
```