Final Project

Tecnológico de Monterrey

The use of Functional Programming in Game Development

Professor: Benjamin Valdés Aguirre

June 1st, 2020

Guillermo Carsolio González A01700041

Index:

Introduction

Before I start diving into what this project entails, I will explain why I chose this project. First off, I wanted to develop a game with the game Unity. Unity is a game engine that has a collection of tools for simulations or game development, I will go deeper into what Unity is later. Now because of the situation we are living now I saw an opportunity to learn something new besides what I had been taught in school. I always have been a fan of gaming and dreamed to learn how it all works. So, when I remembered that the final project for this course was up to us to decide what we would build and then it all clicked. Then I decided that I will find anyway to develop a game using the knowledge that we learn in the course.

In the duration of the course we learned different types of programming paradigms from functional programming, logical programming up to concurrent programming. At first glanced I figured I could build the AI on my game with logical programming, but I found a lot of obstacles just to interface prolog (logical programming language) to C# which is the language used in the scripting in Unity. After a lot of desperate efforts and a back and forward conversation with my professor he suggested to use Lua a multi paradigm programming language. Now it was up to me to apply a paradigm that we learned in the course and apply it with the help of Lua. In the end I decided to use functional programming to develop the AI in my game.

One interesting thing that I found whilst developing the game was that I was also using concurrency but with C#. It will not be the focus of this report, but I will have a brief mention on the area it was utilized and information surrounding the topic.

1. Theory

1.1Functional programming

Basis

Functional programming is a paradigm that is based on the same rules that mathematical functions follow. Also, it is being around for a long time dating back to the Turing Machine. “It is a declarative type of programming style. Its main focus is on what to solve in contrast to an imperative style where the main focus is how to solve” (Vishalxviii, 2019). It bases itself by using expressions that concentrate on producing a value instead of statements that assign values. As in mathematical functions they receive several inputs and then produce an output, it is the same with functional programming. They use distinct functions, but functional programming is mainly based on lambda functions.

Lambda Functions

Lambda functions come from lambda calculus which is a formal system in mathematical logic introduced by Alan Turing's teacher Alonzo Church in the 1930s. Lambda calculous is based on expressing computations through function abstraction. Lambda functions tend to be very simple using reduction rules and in functional programming they are not declared, becoming anonymous. You can assign said lambda functions to variables and use these variables in other functions.

Pure Functions

They are based on giving the same output to the same set of inputs. They do not modify or assign anything the functions are just to produce the same output for a select set of arguments. In the case of this project this is one of the bases that we used functional programming, because as I will explain later it achieves the goal we are trying to achieve.

Programming Languages

Some of the languages that exist are the following: Haskell, JavaScript, Scala, Erlang, Lisp, ML, Clojure, OCaml, Common Lisp, Scheme and in this projects case Lua.

1.2 Concurrent programming

Basis

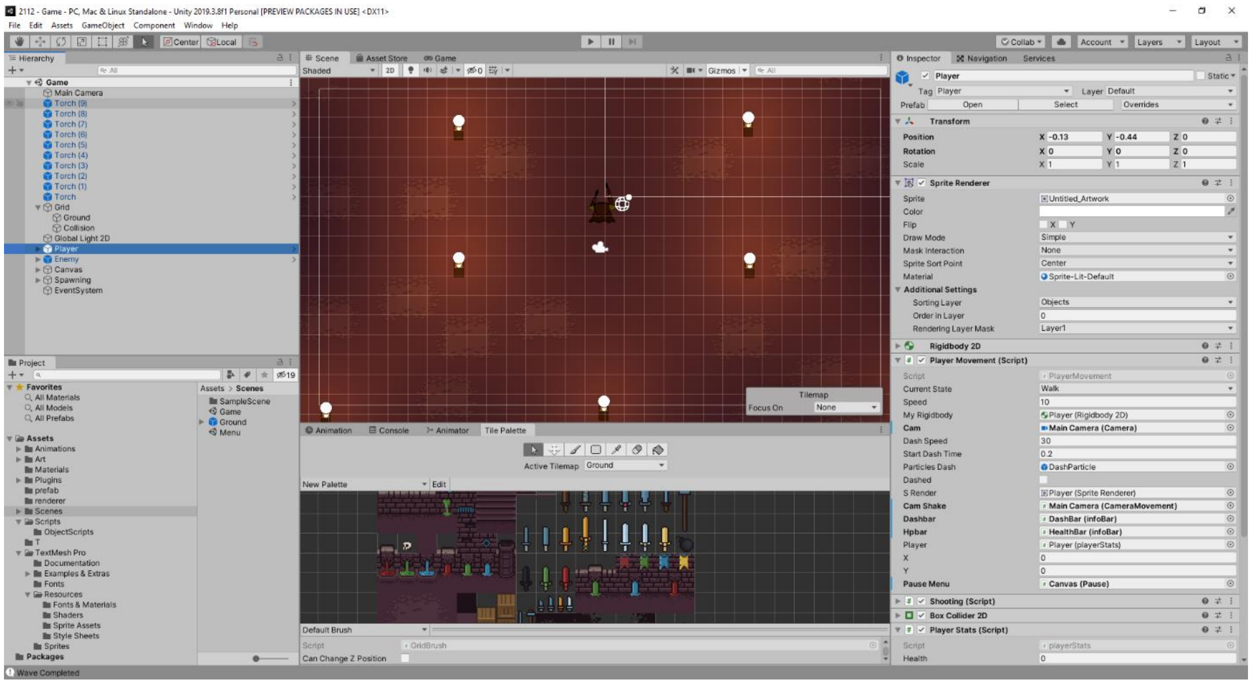
Concurrency is based on two or more circumstances happening at the same moment. In programming is a widely known technic with a variety of uses that consists of multiple processes running at the same time managing access to shared resources. One of the most popular programming languages where you can use this is Java where these processes are called threads. Now, the difference between concurrent programming and parallel programming comes from the fact that concurrent is processed on a single CPU whereas parallel uses multiple cores simultaneously.

2.Tools for development

2.1 Unity

Basis

Unity is a Game Engine developed by Unity Technologies where you can build cross platform 2d or 3d games using a variety of tools. Unity is free to use for personal use if you earn under 100k dollars a year with game development. Some of the most popular games developed with this engine are Kerbal Space Program and Hearthstone. Unity is based on a visual IDE where you can design your game



```
Player                                         Static
Tag Player                   Layer Default
Prefab   Open        Select        Overrides
Transform
Position              X -0.13     Y -0.44     Z 0
Rotation              X 0         Y 0         Z 0
Scale                 X 1         Y 1         Z 1
Sprite Renderer
Sprite                Untitled_Artwork
Color
Flip                  X  Y
Draw Mode             Simple
Mask Interaction      None
Sprite Sort Point     Center
Material              Sprite-Lit-Default
Additional Settings
  Sorting Layer       Objects
  Order in Layer      0
  Rendering Layer Mask Layer1
Rigidbody 2D
Player Movement (Script)
Script                PlayerMovement
Current State         Walk
Speed                 10
My Rigidbody          Player (Rigidbody 2D)
Cam                   Main Camera (Camera)
Dash Speed            30
Start Dash Time       0.2
Particles Dash        DashParticle
Dashed
S Render              Player (Sprite Renderer)
Cam Shake             Main Camera (CameraMovement)
Dashbar               DashBar (infoBar)
Hpbar                 HealthBar (infoBar)
Player                Player (playerStats)
X                     0
Y                     0
Pause Menu            Canvas (Pause)
Shooting (Script)
Box Collider 2D
Player Stats (Script)
Script                playerStats
Health                0
```

and a C# scripting back end to base all your logic. Some of the tools that come from Unity are, a particle system, UI building system, trail system, tile grid system, light renders, and a lot more. Almost everything you edit with the visual IDE you can edit through a script, like the design of a particle effect or the power of the lighting from a light source. Unity is very easy to use ones you understand how it works and it is a good starting point for anyone that wants to learn game development.

2.2 Lua

Basis

Lua is a multiparadigm programing language originated in Brazil in 1993. It is considered as an embeddable scripting language that “supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description” (Lua, 2020). It is very similar to JavaScript in its syntax which can also use functional programming. Lua is based on tables or Metatables which are array like data structures that assign information in pairs. It is dynamically typed so variables do not have types, but the values allocated in said variables do. This value types are nil, Boolean, string, **function,** user data, thread, and table. As said in the functional programming section lambda functions or nameless functions can be assigned to variables and this is possible in Lua. Finally, Lua has automatic memory management so as a user you should not worry on garbage collection.

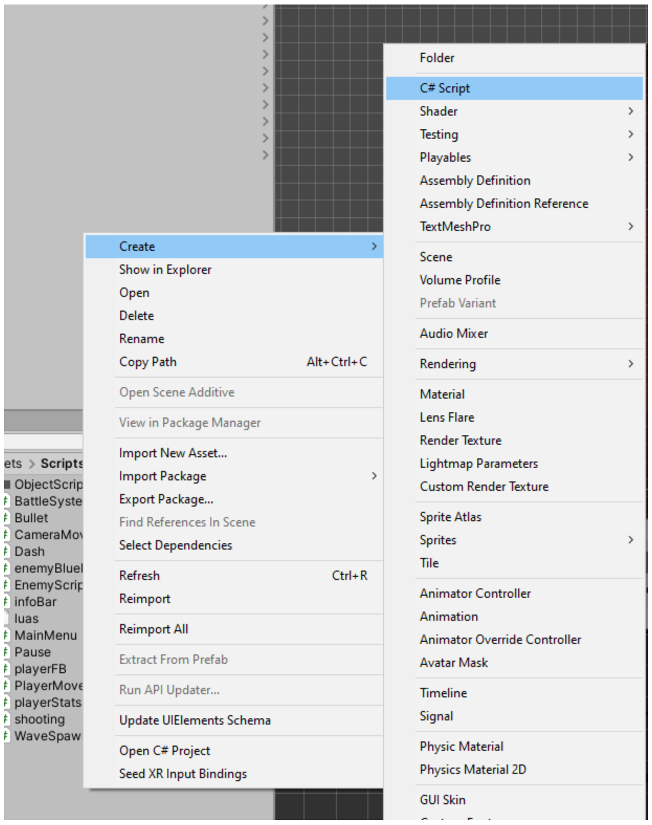
2.3 Setting up Lua with Unity

Getting the necessary tools

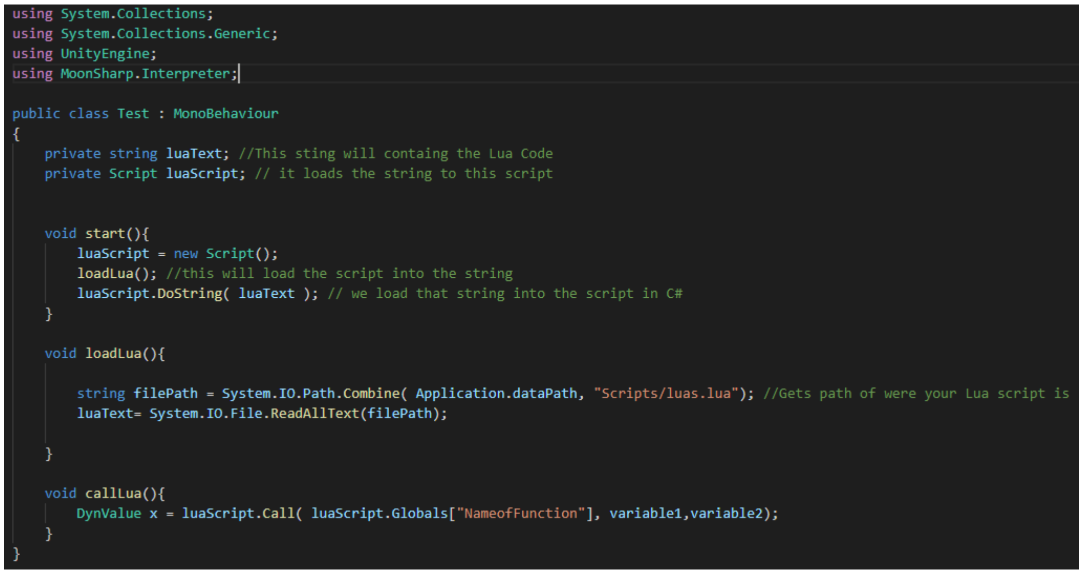
- Install Unity from: https://store.unity.com
- Either download the interpreter moonsharp from the unity asses store or to your computer, from: https://www.moonsharp.org or https://assetstore.unity.com/packages/tools/moonsharp-33776

Script interface

- First create the C# script in unity



- After creating the script, you will have to add the following
- Then you can create your script.lua file and you must allocate on a folder with the name **StreamingAssest** because if you do not, It will not be added into the final build



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using MoonSharp.Interpreter;

public class Test : MonoBehaviour
{
    private string luaText; //This sting will containg the Lua Code
    private Script luaScript; // it loads the string to this script


    void start(){
        luaScript = new Script();
        loadLua(); //this will load the script into the string
        luaScript.DoString( luaText ); // we load that string into the script in C#
    }

    void loadLua(){

        string filePath = System.IO.Path.Combine( Application.dataPath, "Scripts/luas.lua"); //Gets path of were your Lua script is
        luaText= System.IO.File.ReadAllText(filePath);

    }

    void callLua(){
        DynValue x = luaScript.Call( luaScript.Globals["NameofFunction"], variable1,variable2);
    }
}
```

- At first make sure you are **using MoonSharp.Interpreter;**
- Then declare 2 variables (a string, and a script) both private.
- After that create a function in this case **loadLua()** were you get the file path (In the end you must replace it with "StreamingAssests/scriptName.lua" and then interface it as a string in C# as showed in the image above.
- Next on you start function which is a Unity staple function when the scene loads, load the script call the load Lua function and interface the string to the script.
- Finally, you can get the values from Lua functions in dynamic values (Disvalued), later you can cast them to strings or floats.

3. Implementation

3.1 The game development process

Game's purpose

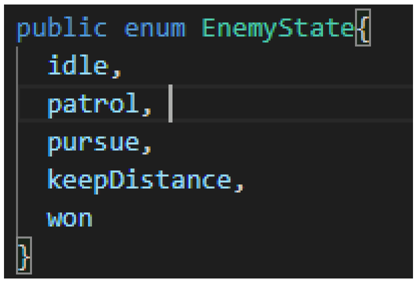
The game is called 2112 and it is set in a dungeon where you are a ninja and must deal with waves of enemies. It is a horde like game where you have three waves of enemies that have different patterns of behavior depending on how much health they have and if the main player has been localized. You have one ability and one form of attack, the ability is a dash/teleportation ability that allows you to move to a position quick, and your attack is a red fire ball.

Game components in general

On this section I will just go on a very general overview on what each of the scripts that this game consists do. The game consists of 15 scripts that all take part of a group that are the following.

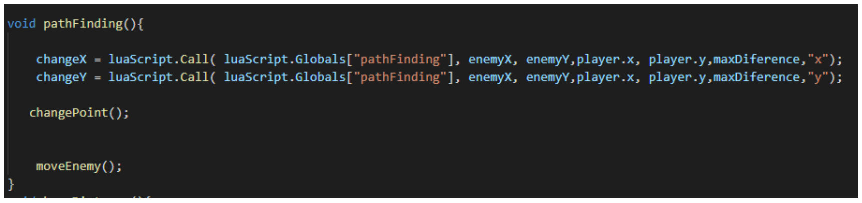
1. **Enemy scripts**: they consist of everything surrounding the enemy. In this case the Lua script would be included but, in this section, we are just going to have a brief explanation on each script. It consists of 3 scripts:
    a. *EnemyBlueFB.cs*: This script consists of managing the projectiles that are shot by the enemy (Enemy Blue Fire Balls). And it uses concurrency when the bullet collides on to any object to destroy the game object but wait till it finishes to stop its velocity.

    b. *EnemyScript.cs*: In this script we dictate the behavior of the enemy and assigns its in game values.



```
public enum EnemyState{
    idle,
    patrol,
    pursue,
    keepDistance,
    won
}
```

        i. It is based on the state machine above where depending on what is happening to his values (Health) it changes.
        ii. And then it behaves depending on what the Lua script dictates by example using the following function.



```
void pathFinding(){

    changeX = luaScript.Call( luaScript.Globals["pathFinding"], enemyX, enemyY,player.x, player.y,maxDiference,"x");
    changeY = luaScript.Call( luaScript.Globals["pathFinding"], enemyX, enemyY,player.x, player.y,maxDiference,"y");

   changePoint();


    moveEnemy();
}
```

   c. *Luas.lua:* This is the script where I based the application of functional programming and it will be addressed in a later section.

2. **General Scripts**: In this section it bases on the world behavior in a general perspective, and because it was a very small game in perspective, I only included one script

   a. *WaveSpawner.cs*: the main purpose of this script is to spawn waves of enemies, and when waves change. In this script I use concurrency, but it will be addressed in a later section.

3. **Object Scripts**: This section is also a small one like the last one because it only consists of the lighting behavior of the torches in the game. The script is *torch.cs* and it is too small to address.

4. **Player Scripts**: Now this section is one of the largest of the bunch and it consist of 5 different scripts.

   a. *playerFB.cs*: This script oversees the movement and collision of the projectile that the player shoots (Player Fire Ball). It is very simple, and it uses concurrency in the same way that the enemy projectile did.
   b. *CameraMovement.cs*: The goal of this script is to follow the player but not perfectly. The camera sort of drags on behind the player to make the movement feel smooth. This script also uses concurrency that will be addressed in a later section.
   c. *PlayerMovement.cs*: In this script it controls everything that has to do with player movement. It manages walking, mouse locating and the dash (Which is the teleportation like ability).
   d. *playerStats.cs*: It consists of just administering stats like health, dash regen, speed etc.
   e. *shooting.cs*: This is a small script that just creates and instance of a fireball when the player shoots.

5. **UI Scripts**: Finally on this section we have all the script managing the UI like the information bars or the menus (Main, Pause, Endgame) it consists of 5 different scripts but 4 are very similar so I will bundle them up in one point.

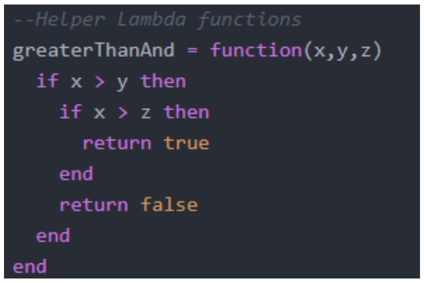
a. *Menu Scripts (Deathscreen.cs, MainMenu.cs, Pause.cs, winScreen.cs)*: This script all manages the buttons that are on the menus within the UI. And it manages scene changes so you can go back to the main menu or start a new game.

b. *InfoBar.cs*: And the last script updates the information bars in the UI that display the health and the dash regeneration.

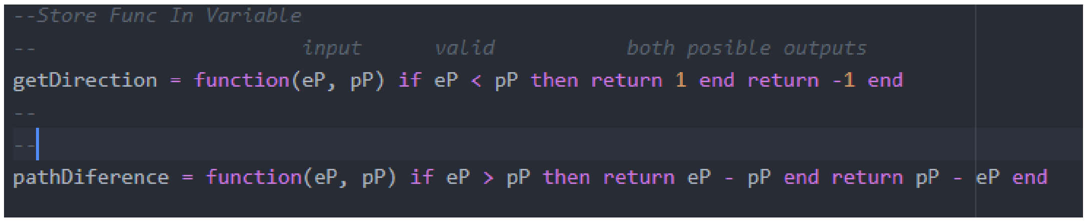
3.2 Implementation of functional programming

To implement functional programming into this project I interfaced a Lua script into the *EnemyScrip.cs* where I put it all together to be able to use in the game as shown in a previous section.

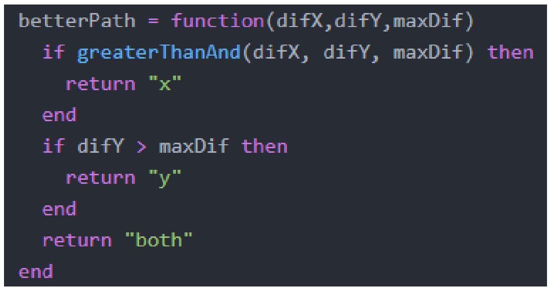
So, I started my script by typing 4 lambda functions and assigning them to variables so I can use them within other functions and because they were disposable functions.



```lua
--Helper Lambda functions
greaterThanAnd = function(x,y,z)
  if x > y then
    if x > z then
      return true
    end
    return false
  end
end
```



```lua
--Store Func In Variable
--                    input      valid           both posible outputs
getDirection = function(eP, pP) if eP < pP then return 1 end return -1 end
--
--
pathDiference = function(eP, pP) if eP > pP then return eP - pP end return pP - eP end
```



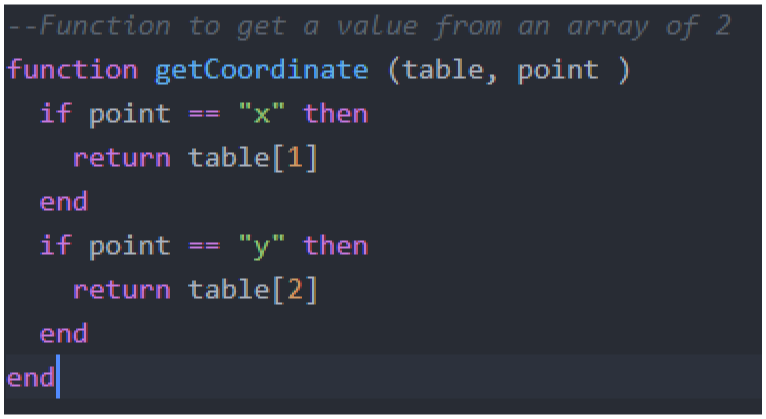
```lua
betterPath = function(difX,difY,maxDif)
  if greaterThanAnd(difX, difY, maxDif) then
    return "x"
  end
  if difY > maxDif then
    return "y"
  end
  return "both"
end
```

So this functions are:

- **greaterThanAnd**: this function will compare 1 value to 2 different values, if it the first value is then it returns true. I made this into a lambda function because it was just more organized to use in future functions instead of having to retype all the if statements.
- **getDirection**: Very simple function just tells you in what direction you should go. This function does not care what axis you are on, it just outputs a multiplier if the value is larger or smaller than the given value.
- **pathDiference**: it just gives the absolute value of the distance between to points in the same axis.
- **betterPath**: now this lambda function receives 3 distinct paths, the paths from both axis X and Y and one max path that I give the value in the enemy script. And then takes the 3 paths and uses another lambda function to get which value is the largest and output if it should go in a direction where it is the furthest from of if the enemy is equidistant from the player.

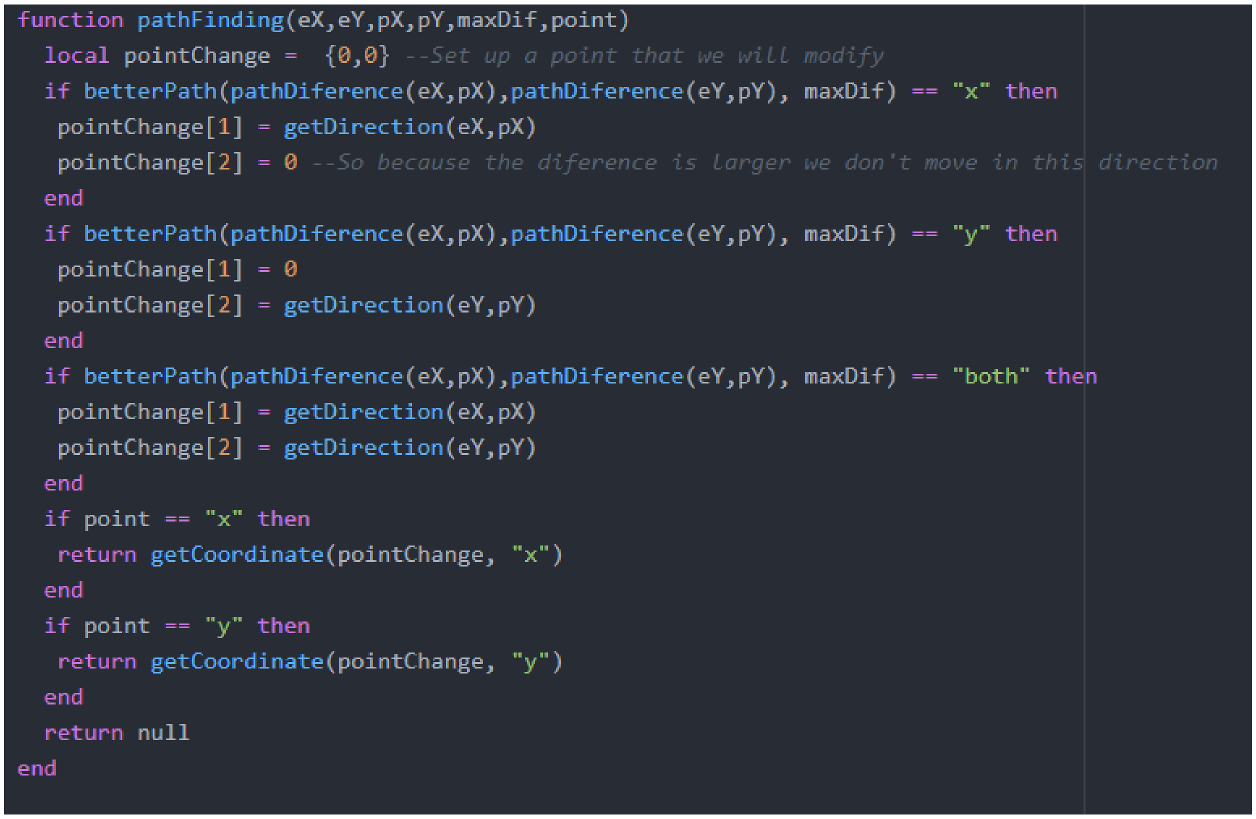
Again, all these functions will always output the same value for the same set of inputs. What this script is doing is just laying out information that can be used to make decisions or control behavior in the future.

Once I had my lambda functions, I coded one helper functions just to lay out information the way I wanted to. This function just displays the axis value (X or Y) that was pass though the Lua structure called a table

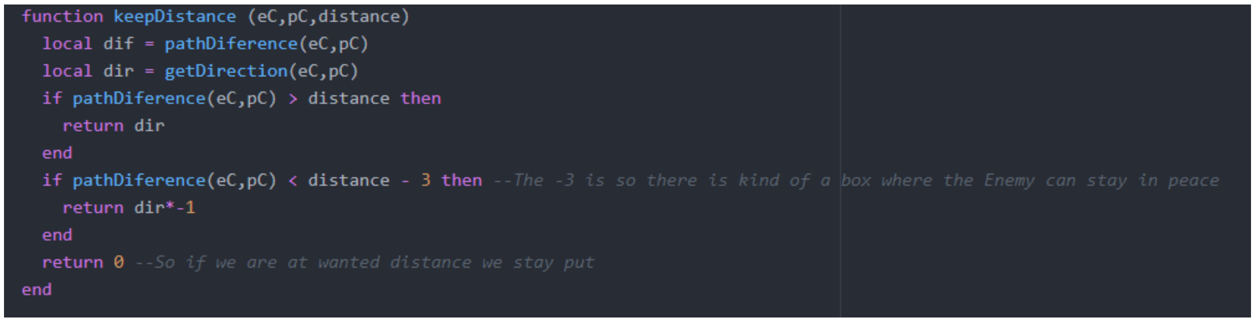


```lua
--Function to get a value from an array of 2
function getCoordinate (table, point )
  if point == "x" then
    return table[1]
  end
  if point == "y" then
    return table[2]
  end
end
```

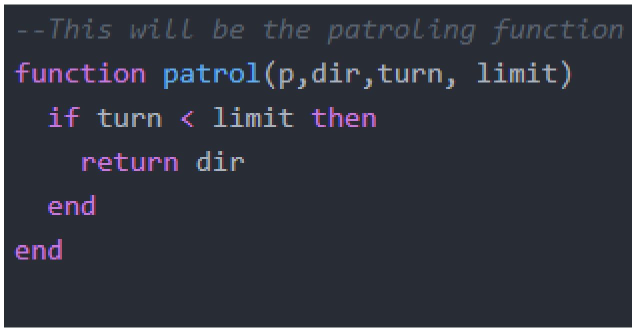
Once I was all set up it was time to code the functions that were going to dictate the behavior of the enemy. The enemy has 3 staple behaviors, patrolling state, pursuing state and, keeping distance state. The functions that are build upon these states are the following:



```lua
function pathFinding(eX,eY,pX,pY,maxDif,point)
  local pointChange =  {0,0} --Set up a point that we will modify
  if betterPath(pathDiference(eX,pX),pathDiference(eY,pY), maxDif) == "x" then
   pointChange[1] = getDirection(eX,pX)
   pointChange[2] = 0 --So because the diference is larger we don't move in this direction
  end
  if betterPath(pathDiference(eX,pX),pathDiference(eY,pY), maxDif) == "y" then
   pointChange[1] = 0
   pointChange[2] = getDirection(eY,pY)
  end
  if betterPath(pathDiference(eX,pX),pathDiference(eY,pY), maxDif) == "both" then
   pointChange[1] = getDirection(eX,pX)
   pointChange[2] = getDirection(eY,pY)
  end
  if point == "x" then
   return getCoordinate(pointChange, "x")
  end
  if point == "y" then
   return getCoordinate(pointChange, "y")
  end
  return null
end
```

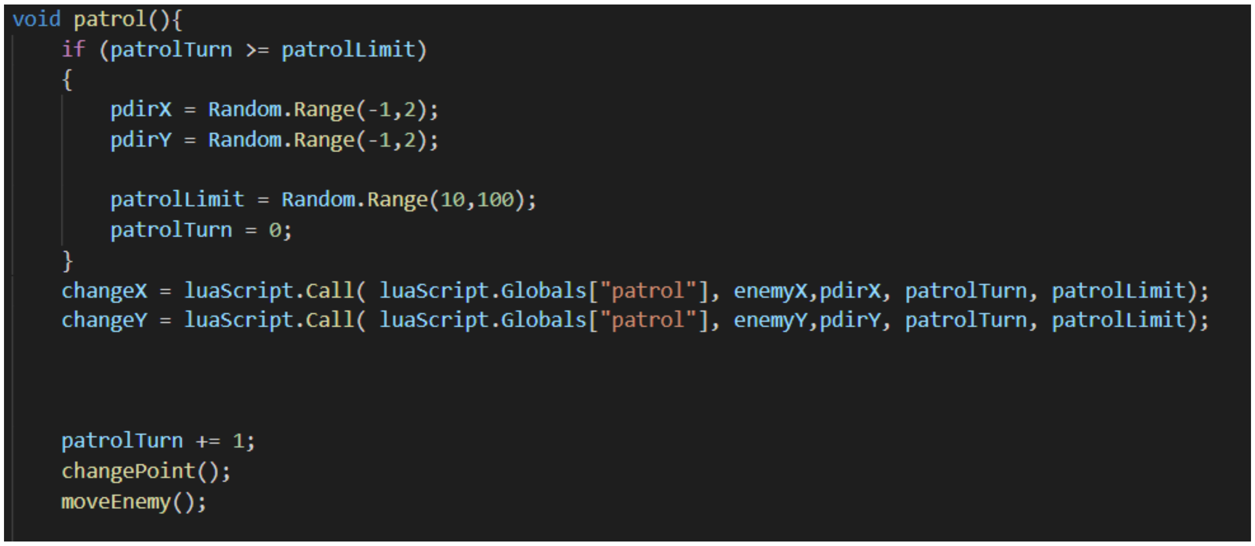


```lua
function keepDistance (eC,pC,distance)
  local dif = pathDiference(eC,pC)
  local dir = getDirection(eC,pC)
  if pathDiference(eC,pC) > distance then
    return dir
  end
  if pathDiference(eC,pC) < distance - 3 then --The -3 is so there is kind of a box where the Enemy can stay in peace
    return dir*-1
  end
  return 0 --So if we are at wanted distance we stay put
end
```



```lua
--This will be the patroling function
function patrol(p,dir,turn, limit)
  if turn < limit then
    return dir
  end
end
```

- **pathFinding**: this function takes the position of the enemy and the player as well as the maximum difference of distance from a point the enemy can be from the player to advance in the best possible scenario. Also, it has the axis of choice as an input because these functions only output one result.
- **keepDistance**: In this function the inputs are the input of the axis that we want from both the player and the enemy and the distance that they are going to keep from. It just calculates if the enemy is to close, if that is the case it retreats if not it gets as close as the condition allows it to go.
- **patrol**: Funnily enough this is one of the most complex even though it is the smallest of the bunch. Also it has one input that is not used within the function but that is because I reduced at one point and just found out. This function is more of a counter that is used with this function.



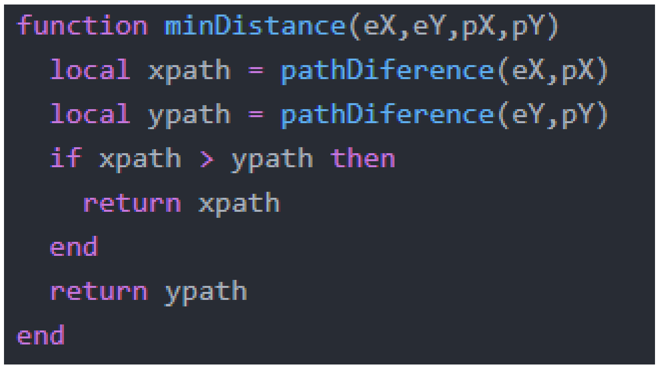
```
void patrol(){
    if (patrolTurn >= patrolLimit)
    {
        pdirX = Random.Range(-1,2);
        pdirY = Random.Range(-1,2);

        patrolLimit = Random.Range(10,100);
        patrolTurn = 0;
    }
    changeX = luaScript.Call( luaScript.Globals["patrol"], enemyX,pdirX, patrolTurn, patrolLimit);
    changeY = luaScript.Call( luaScript.Globals["patrol"], enemyY,pdirY, patrolTurn, patrolLimit);



    patrolTurn += 1;
    changePoint();
    moveEnemy();
```

Where it creates two random values for the direction that the enemy will go to patrol and a limit that is the counter for when it should change direction. Now the way the enemy stops patrolling is using the following function.



```
function minDistance(eX,eY,pX,pY)
  local xpath = pathDiference(eX,pX)
  local ypath = pathDiference(eY,pY)
  if xpath > ypath then
    return xpath
  end
  return ypath
end
```

- **minDistance**: This function just returns the smallest path from either axis. I use this function to see if the player is close enough to the enemy so the enemy changes to a pursue state.

Conclusion

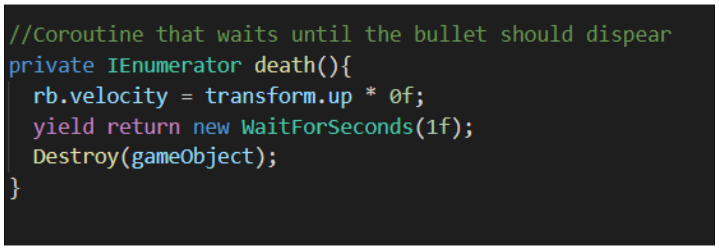
This was the way I chose to use functional programming in my game. I found that there is so much to game development that I spent so much time in everything surrounding the game to make everything work. But I was planning to implement more behaviours taking in consideration the stats or values of the player. The issue is that I had to have all those methods already set up in the game and that take time. In the end I do believe that it can be very useful in game development if used properly

3.3 Implementation of concurrent programming

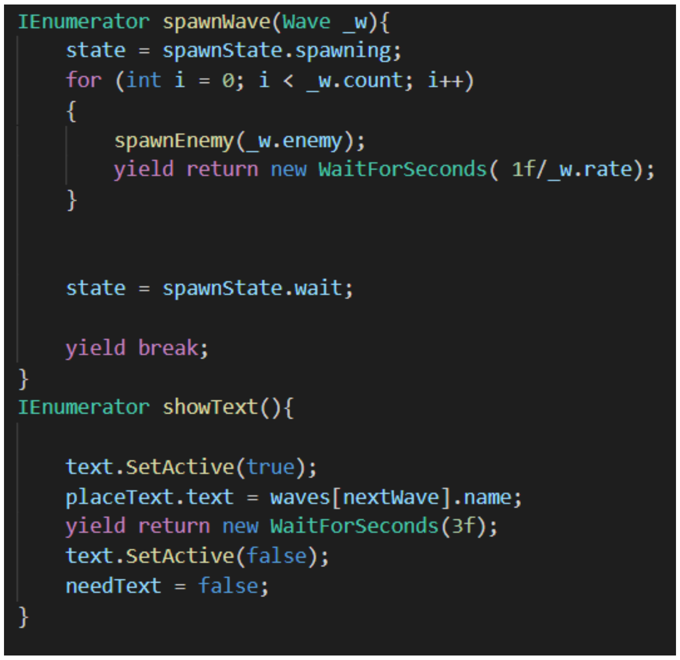
In this section I will just show where I used concurrency within my game but in a very brief matter. Because this is just an extra add on in the project, I will be brief and show how it was implemented.

First off in C# IEnumerator functions are functions that are working concurrently whilst the game is playing you use them in parts like to play a death animation, wait for an object to die or to do something external to the main game process.

I will show where I used it in my project in the following images:



```
//Coroutine that waits until the bullet should dispear
private IEnumerator death(){
  rb.velocity = transform.up * 0f;
  yield return new WaitForSeconds(1f);
  Destroy(gameObject);
}
```

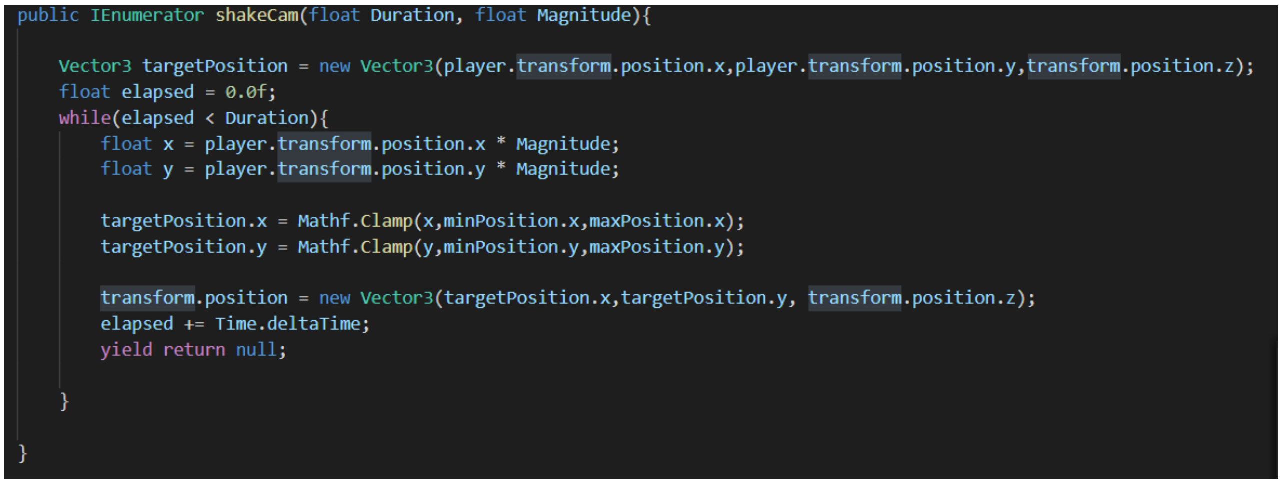


```
IEnumerator spawnWave(Wave _w){
    state = spawnState.spawning;
    for (int i = 0; i < _w.count; i++)
    {
        spawnEnemy(_w.enemy);
        yield return new WaitForSeconds( 1f/_w.rate);
    }


    state = spawnState.wait;

    yield break;
}
IEnumerator showText(){

    text.SetActive(true);
    placeText.text = waves[nextWave].name;
    yield return new WaitForSeconds(3f);
    text.SetActive(false);
    needText = false;
}
```



```
public IEnumerator shakeCam(float Duration, float Magnitude){

    Vector3 targetPosition = new Vector3(player.transform.position.x,player.transform.position.y,transform.position.z);
    float elapsed = 0.0f;
    while(elapsed < Duration){
        float x = player.transform.position.x * Magnitude;
        float y = player.transform.position.y * Magnitude;

        targetPosition.x = Mathf.Clamp(x,minPosition.x,maxPosition.x);
        targetPosition.y = Mathf.Clamp(y,minPosition.y,maxPosition.y);

        transform.position = new Vector3(targetPosition.x,targetPosition.y, transform.position.z);
        elapsed += Time.deltaTime;
        yield return null;

    }

}
```

- The first concurrent function is a death function of the fire balls when they collide with an object, I use this so the trail effect can finish its animation, so it waits a second before it destroys the object.
- The second one is spawning function that just spawns all the wanted enemies in that wave.
- The third function just displays the wave number on the screen for a moment.
- Lastly it is the shake effect that the camera does when the player does the dashing ability.

5. CONCLUSION

There are multiple uses for what we have seen in class from some of the examples shown by my fellow classmates up to building a game. Remembering when we were just taking the first classes learning about scheme and the basis of functional programming, I never thought that I could build a game with it. I was impressed with the lack of information surrounding the use of functional programming in game development considering it can be very useful. Today we are all used to premade libraries that do everything for you but understanding different tools that can make you programs more efficient can end up being more beneficial. In the end game development is hard… that is my conclusion.

References

Dechalert, A. (2019, June 15). What exactly is functional programming? Retrieved from https://medium.com/madhash/what-exactly-is-functional-programming-ea02c86753fd

Edpresso. (n.d.) What is concurrent programming? Retrieved from https://www.educative.io/edpresso/what-is-concurrent-programming

Technologies, U. (2020). Unity. Retrieved from https://unity.com/

Steinert-Threlkeld, S. (n.d.). Lambda Calculi. Retrieved from https://www.iep.utm.edu/lambda-calculi/

Vishalxviii. (2019, January 2). Functional Programming Paradigm. Retrieved from https://www.geeksforgeeks.org/functional-programming-paradigm/