APPENDIX F

Source Code

APP BUILD SETTINGS

```
settings =
 {

        orientation =
        {
                default = "portrait",--sets orientation when the app is launched it can landscapeRight ,
landscapeLeft , or portrait
                supported = { "portrait", "portraitUpsideDown" } --sets what orientation the app can
support.
        },

         androidPermissions =
         {
                "android.permission.INTERNET",-- gives the app the permission to access the internet
                "android.permission.READ_EXTERNAL_STORAGE",
                "android.permission.WRITE_EXTERNAL_STORAGE",

         },

        iphone =
        {
                        plist=
                {
                        --UIStatusBarHidden=true,
                },

        },
}
```

APP CONFIGURATIONS SETTINGS

```
application =
{
                content =
                {
                        fps = 60,                       -- Desired frame rate
                        width = 320,                    --320 Desired width of the application
                        height = 480,                   --480 Desired height of the application
                        scale = "letterbox",
                        xAlign = "center",
                        yAlign  = "center",
                imageSuffix =
                 {
                        ["@1-5x"] = 1.5, -- Various Android phones.
                         ["@2x"] = 2,    -- iPhone 4 and higher, iPod touch, iPad1, and iPad2
                         ["@3x"] = 3,    -- Various Android tablets
                         ["@4x"] = 4,    -- iPad 3+
                }

                },
}
```

CREATE SYSTEM CONFIGURATIONS

```
local json = require ("json")
local myData = require ("myData")
local makeSysConfig = require ("makeSystemConfig")

local M = {}
--checks if the system configurations exist
function doesFileExist( fname, path )

   local results = false

   local filePath = system.pathForFile( fname, path )
   --filePath will be 'nil' if file doesn't exist and the path is 'system.ResourceDirectory'
   if ( filePath ) then
      filePath = io.open( filePath, "r" )
   end
   if ( filePath ) then
      print( "SYSTEM CONFIGURATIONS FILE FOUND: " .. fname )
      --clean up file handles
      filePath:close()
      results = true
   else
      print( "SYSTEM CONFIGURATIONS FILE DOES NOT EXIST: " .. fname )

                              -- system default configurations
                              print ("CREATING SYSTEM CONFIGURATIONS...")
                              local systemConfig = {}
                              systemConfig.soundOn = true
                              systemConfig.musicOn = true
                              makeSysConfig.makeSystemConfig(systemConfig, "systemConfiguration.json")

   end

   return results
end
M.doesFileExist =doesFileExist

return M

local json = require ("json")
--make config file, if it does not exist, it will create
local M = {}
function makeSystemConfig(tbl, filename)
          local path = system.pathForFile (filename, system.DocumentsDirectory ) --path of the json file
          local file = io.open (path, "w") --open the file

                    if file then --if the file does not exist
                              local config = json.encode( tbl ) --encodes the data
                              file:write(config) --writes the data into the json file
                              io.close ( file )--closes the file
                              print ("System configurations successfully created")
                              return true
                    else
                              print ("System configurations already exist")
                              return false --file exist
                    end
end
M.makeSystemConfig = makeSystemConfig

return M
```

LOAD SYSTEM CONFIGURATIONS

```
local json = require ("json")

local gameSettings = { }

local function loadSysConfig()

        local function loadSystemConfig (filename)
                local path = system.pathForFile (filename,system.DocumentsDirectory )
                local contents = ""
                local myTable = { }
                local file = io.open ( path ,"r" )

                        if file then
                                local contents = file:read ("*a")
                                myTable = json.decode( contents )
                                io.close (file)
                                return myTable
                        end
                                return nil
        end

 gameSettings = loadSystemConfig ("systemConfiguration.json")

 end
--gets the currents system settings
loadSysConfig()
```

CREATE GAME DATABASE

```
local sqlite = require ( "sqlite3" )

local M = {}

local function createAncientDB()
local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
local db = sqlite.open(path)

local sql = [[
        CREATE TABLE IF NOT EXISTS ancientDB (
        question_ID INTEGER PRIMARY KEY,
        Sci_Name,
        inv_Name,
        img_Path,
        disc_Date,
        inv_Desc,
        choice1,
        choice2,
        choice3,
        level INTEGER
        );
        ]]
print ("SUCCESSFULLY CREATED ANCIENT ERA DATABASE")
db:exec(sql)
db:close()
end
M.createAncientDB = createAncientDB
```

```
return M



local sqlite = require ( "sqlite3" )

local M = {}

local function createMiddleDB()
local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
local db = sqlite.open(path)

local sql = [[
        CREATE TABLE IF NOT EXISTS middleDB (
        question_ID INTEGER PRIMARY KEY,
        Sci_Name,
        inv_Name,
        img_Path,
        disc_Date,
        inv_Desc,
        choice1,
        choice2,
        choice3,
        level INTEGER
        );
        ]]
print ("SUCCESSFULLY CREATED MIDDLE AGE DATABASE")
db:exec(sql)
db:close()
end
M.createMiddleDB = createMiddleDB

return M


local sqlite = require ( "sqlite3" )

local M = {}

local function createEarlyDB()
local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
local db = sqlite.open(path)

local sql = [[
        CREATE TABLE IF NOT EXISTS earlyDB (
        question_ID INTEGER PRIMARY KEY,
        Sci_Name,
        inv_Name,
        img_Path,
        disc_Date,
        inv_Desc,
        choice1,
        choice2,
        choice3,
        level INTEGER
        );
        ]]
print ("SUCCESSFULLY CREATED EARLY MODERN AGE DATABASE")
db:exec(sql)
db:close()
end
M.createEarlyDB = createEarlyDB
```

```
return M



local sqlite = require ( "sqlite3" )

local M = {}

local function createModernDB()
local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
local db = sqlite.open(path)

local sql = [[
        CREATE TABLE IF NOT EXISTS modernDB (
        question_ID INTEGER PRIMARY KEY,
        Sci_Name,
        inv_Name,
        img_Path,
        disc_Date,
        inv_Desc,
        choice1,
        choice2,
        choice3,
        level INTEGER
        );
        ]]
print ("SUCCESSFULLY CREATED MODERN AGE DATABASE")
db:exec(sql)
db:close()
end
M.createModernDB = createModernDB

return M


local sqlite = require ("sqlite3")

local M = {}

function createProfile()
        local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
        local db = sqlite.open(path)

        local sql =[[
                CREATE TABLE IF NOT EXISTS player_Data (
                player_ID INTEGER PRIMARY KEY,
                player_Name,
                current_Coins,
                coins_Acquired,
                achievement_pts,
                current_Level,
                current_Era,
                answered_Correct,
                answered_Wrong,
                num_Hints_Used,
                game_Finished,
                slots_Last_Used,
                level_Tries,
                last_LevelTry,
                save_Status
                );
                ]]
```

```
print ("SUCCESSFULLY CREATED PLAYER DATABASE")
db:exec(sql)
db:close()
end
M.createProfile = createProfile


return M


local sqlite = require ("sqlite3")

local M = {}

function createAchievement()
	local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
	local db = sqlite.open(path)

	local sql =[[
		CREATE TABLE IF NOT EXISTS player_Achievement (
		id_Num INTEGER PRIMARY KEY,
		player_ID INTEGER, --foreign key
		coll_2k INTEGER,
		coll_3k INTEGER,
		coll_5k INTEGER,
		lastColl INTEGER,
		hint1 INTEGER,
		hint2 INTEGER,
		hint3 INTEGER,
		hint4 INTEGER,
		acientFin INTEGER,
		middleFin INTEGER,
		earlyFin INTEGER,
		modernFin INTEGER,
		gameFin INTEGER,
		totalPoint INTEGER
		);
		]]
print ("SUCCESSFULLY CREATED ACHIEVEMENTS DATABASE")
db:exec(sql)
db:close()
end
M.createAchievement = createAchievement


return M
```

LOAD GAME LEVEL MECHANICS

--**Ancient Era Restrictions**

```
local M = {}

function selectRestriction(param)
	local i = tonumber(param)


	local myTable = {
```

```
--level 1
                {
                        level = {
                                timeRes = 51,
                                corAns = 5,
                                wroAns = 5,
                                level = 1,
                                }
                        },
-- level 2
                {
                        level = {
                                timeRes = 50,
                                corAns = 5,
                                wroAns = 5,
                                level = 2,
                                }
                        },
--level 3
                {
                        level = {
                                timeRes = 49,
                                corAns = 5,
                                wroAns = 5,
                                level = 3,
                                }
                        },
--level 4
                {
                        level = {
                                timeRes = 48,
                                corAns = 7,
                                wroAns = 4,
                                level = 4,
                                }
                        },
--level 5
                {
                        level = {
                                timeRes = 47,
                                corAns = 7,
                                wroAns = 4,
                                level = 5,
                                }
                        },
--level 6
                {
                        level = {
                                timeRes =46 ,
                                corAns = 7,
                                wroAns = 4,
                                level = 6,
                                }
                        },
--level 7
                {
                        level = {
                                timeRes =45 ,
                                corAns = 8,
                                wroAns = 3,
                                level = 7,
                                }
```

```
                    },
--level 8
              {
                    level = {
                          timeRes =44 ,
                          corAns = 8,
                          wroAns = 3,
                          level = 8,
                          }
                    },
--level 9
              {
                    level = {
                          timeRes =43 ,
                          corAns = 8,
                          wroAns = 3,
                          level = 9,
                          }
                    },
--level 10
              {
                    level = {
                          timeRes =42 ,
                          corAns = 9,
                          wroAns = 3,
                          level = 10,
                          }
                    },

}

return myTable[i].level

end
M.selectRestriction = selectRestriction

return M
```

--**Middle Age Restrictions**

```
local M = {}

function selectRestriction(param)
        local i = tonumber(param)


        local myTable = {
--level 1
              {
                    level = {
                          timeRes = 43,
                          corAns = 5,
                          wroAns = 5,
                          reward = 30,
                          }
                    },
-- level 2
              {
                    level = {
                          timeRes = 42,
                          corAns = 5,
                                        wroAns = 5,
                                        reward = 20,
                                        }
                                },
--level 3
                        {
                                level = {
                                        timeRes = 41,
                                        corAns = 5,
                                        wroAns = 5,
                                        reward = 30,
                                        }
                                },
--level 4
                        {
                                level = {
                                        timeRes = 40,
                                        corAns = 7,
                                        wroAns = 4,
                                        reward = 30,
                                        }
                                },
--level 5
                        {
                                level = {
                                        timeRes = 39,
                                        corAns = 7,
                                        wroAns = 4,
                                        reward = 30,
                                        }
                                },
--level 6
                        {
                                level = {
                                        timeRes = 38,
                                        corAns = 7,
                                        wroAns = 4,
                                        reward = 30,
                                        }
                                },
--level 7
                        {
                                level = {
                                        timeRes = 37,
                                        corAns = 8,
                                        wroAns = 3,
                                        reward = 30,
                                        }
                                },
--level 8
                        {
                                level = {
                                        timeRes = 36,
                                        corAns = 8,
                                        wroAns = 3,
                                        reward = 30,
                                        }
                                },
--level 9
                        {
                                level = {
                                        timeRes = 35,
                                        corAns = 8,
                                        wroAns = 3,
                                        reward = 30,
                                        }
                                },
--level 10
                        {
                                level = {
                                        timeRes = 34,
                                        corAns = 9,
                                        wroAns = 3,
                                        reward = 30,
                                        }
                                },
}

return myTable[i].level

end
M.selectRestriction = selectRestriction

return M
```

--**Early Modern Age Restrictions**

```
local M = {}

function selectRestriction(param)
        local i = tonumber(param)


        local myTable = {
--level 1
                        {
                                level = {
                                        timeRes = 35,
                                        corAns = 5,
                                        wroAns = 5,
                                        reward = 25,
                                        }
                                },
-- level 2
                        {
                                level = {
                                        timeRes = 34,
                                        corAns = 5,
                                        wroAns = 5,
                                        reward = 40,
                                        }
                                },
--level 3
                        {
                                level = {
                                        timeRes = 33,
                                        corAns = 5,
                                        wroAns = 5,
                                        reward = 35,
                                        }
                                },
--level 4
                        {
```

```
                        level = {
                                timeRes = 32,
                                corAns = 7,
                                wroAns = 4,
                                reward = 35,
                                }
                        },
--level 5
                {
                        level = {
                                timeRes = 31,
                                corAns = 7,
                                wroAns = 4,
                                reward = 30,
                                }
                        },
--level 6
                {
                        level = {
                                timeRes = 30,
                                corAns = 7,
                                wroAns = 4,
                                reward = 30,
                                }
                        },
--level 7
                {
                        level = {
                                timeRes = 29,
                                corAns = 8,
                                wroAns = 3,
                                reward = 30,
                                }
                        },
--level 8
                {
                        level = {
                                timeRes = 28,
                                corAns = 8,
                                wroAns = 3,
                                reward = 30,
                                }
                        },
--level 9
                {
                        level = {
                                timeRes = 27,
                                corAns = 8,
                                wroAns = 3,
                                reward = 30,
                                }
                        },
--level 10
                {
                        level = {
                                timeRes = 26,
                                corAns = 9,
                                wroAns = 3,
                                reward = 30,
                                }
                        },
--level 11
```

```
			{
				level = {
					timeRes = 25,
					corAns = 9,
					wroAns = 3,
					reward = 30,
					}
				},
--level 12
			{
				level = {
					timeRes = 24,
					corAns = 9,
					wroAns = 3,
					reward = 30,
					}
				},
--level 13
			{
				level = {
					timeRes = 23,
					corAns = 10,
					wroAns = 2,
					reward = 30,
					}
				},
--level 14
			{
				level = {
					timeRes = 22,
					corAns = 10,
					wroAns = 2,
					reward = 30,
					}
				},
--level 15
			{
				level = {
					timeRes = 21,
					corAns = 10,
					wroAns = 2,
					reward = 30,
					}
				},
}

return myTable[i].level

end
M.selectRestriction = selectRestriction

return M
```

--**Modern Age Restrictions**

```
local M = {}

function selectRestriction(param)
	local i = tonumber(param)
```

```
          local myTable = {
--level 1
                    {
                              level = {
                                        timeRes = 27,
                                        corAns = 5,
                                        wroAns = 5,
                                        reward = 50,
                                        }
                              },
-- level 2
                    {
                              level = {
                                        timeRes = 26,
                                        corAns = 5,
                                        wroAns = 5,
                                        reward = 40,
                                        }
                              },
--level 3
                    {
                              level = {
                                        timeRes = 25,
                                        corAns = 5,
                                        wroAns = 5,
                                        reward = 45,
                                        }
                              },
--level 4
                    {
                              level = {
                                        timeRes = 24,
                                        corAns = 7,
                                        wroAns = 4,
                                        reward = 45,
                                        }
                              },
--level 5
                    {
                              level = {
                                        timeRes = 23,
                                        corAns = 7,
                                        wroAns = 4,
                                        reward = 45,
                                        }
                              },
--level 6
                    {
                              level = {
                                        timeRes = 22,
                                        corAns = 7,
                                        wroAns = 4,
                                        reward = 45,
                                        }
                              },
--level 7
                    {
                              level = {
                                        timeRes = 21,
                                        corAns = 8,
                                        wroAns = 3,
                                        reward = 45,
					}
				},
--level 8
			{
				level = {
					timeRes = 20,
					corAns = 8,
					wroAns = 3,
					reward = 45,
					}
				},
--level 9
			{
				level = {
					timeRes = 19,
					corAns = 8,
					wroAns = 3,
					reward = 45,
					}
				},
--level 10
			{
				level = {
					timeRes = 18,
					corAns = 9,
					wroAns = 3,
					reward = 45,
					}
				},
--level 11
			{
				level = {
					timeRes = 17,
					corAns = 9,
					wroAns = 3,
					reward = 45,
					}
				},
--level 12
			{
				level = {
					timeRes = 16,
					corAns = 9,
					wroAns = 3,
					reward = 45,
					}
				},
--level 13
			{
				level = {
					timeRes = 15,
					corAns = 10,
					wroAns = 2,
					reward = 45,
					}
				},
--level 14
			{
				level = {
					timeRes = 14,
					corAns = 10,
					wroAns = 2,
                                        reward = 45,
                                        }
                                },
--level 15
                        {
                                level = {
                                        timeRes = 13,
                                        corAns = 10,
                                        wroAns = 2,
                                        reward = 45,
                                        }
                                },
}

return myTable[i].level

end
M.selectRestriction = selectRestriction

return M
```

ADDING QUESTIONS INTO DATABASE

```
local sqlite =require ( "sqlite3" )
local questionTable = require ( "questionTable" )
local M = {}

--add ancient period question
--************************************
local function addAncientQuestion()
		local path = system.pathForFile ( "playerDB.sqlite",system.DocumentsDirectory)
		local db = sqlite.open(path)
		local sql
		local ancientTable = {}
		ancientTable = questionTable.ancientQuestion
		print ("TOTAL ANCIENT QUESTION: "..#ancientTable)
		local totalRows = 0
			for row in db:nrows("SELECT * FROM ancientDB") do
				totalRows = totalRows + 1
			end

			if (totalRows == 0) then
				for z= 1, #ancientTable do
				sql = [[
					INSERT INTO ancientDB ("Sci_Name" ,"inv_Name" ,"img_Path",
"disc_Date" , "inv_Desc", "choice1", "choice2" , "choice3","level")
				VALUES ("]] .. ancientTable[z].sciName ..
				[[","]] .. ancientTable[z].invName ..
				[[","]] .. ancientTable[z].imagePath ..
				[[","]] .. ancientTable[z].discDate ..
				[[","]] .. ancientTable[z].briefDesc ..
				[[","]] .. ancientTable[z].choice1 ..
				[[","]] .. ancientTable[z].choice2 ..
				[[","]] .. ancientTable[z].choice3 ..
				[[","]] .. ancientTable[z].level ..
				[[");]]
				db:exec(sql)
					end
			end
db:close()
end
M.addAncientQuestion = addAncientQuestion


--add middle age questios
--**************************************
local function addMiddleQuestion()
		local path = system.pathForFile ( "playerDB.sqlite",system.DocumentsDirectory)
		local db = sqlite.open(path)
		local sql
		local middleTable = {}
		middleTable = questionTable.middleQuestion
		print ("TOTAL MIDDLE AGE QUESTION: ".. #middleTable)

		local totalRows = 0
			for row in db:nrows("SELECT * FROM middleDB") do
				totalRows = totalRows + 1
			end

			if (totalRows == 0) then
				for z= 1, #middleTable do
				sql = [[
					INSERT INTO middleDB ("Sci_Name" ,"inv_Name", "img_Path",
"disc_Date" , "inv_Desc", "choice1", "choice2" , "choice3","level")
				VALUES ("]] .. middleTable[z].sciName ..
                        [[","]] .. middleTable[z].invName ..
                        [[","]] .. middleTable[z].imagePath ..
                        [[","]] .. middleTable[z].discDate ..
                        [[","]] .. middleTable[z].briefDesc ..
                        [[","]] .. middleTable[z].choice1 ..
                        [[","]] .. middleTable[z].choice2 ..
                        [[","]] .. middleTable[z].choice3 ..
                        [[","]] .. middleTable[z].level ..
                        [[");]]
                        db:exec(sql)
                                end
                end
db:close()
end
M.addMiddleQuestion = addMiddleQuestion


--add early modern age questios
--*********************************************
local function addEarlyQuestion()
        local path = system.pathForFile ( "playerDB.sqlite",system.DocumentsDirectory)
        local db = sqlite.open(path)
        local sql
        local earlyTable = {}
        earlyTable = questionTable.earlyQuestion
        print ("TOTAL EARLY MODERN AGE QUESTION: ".. #earlyTable)

        local totalRows = 0
                for row in db:nrows("SELECT * FROM earlyDB") do
                        totalRows = totalRows + 1
                end

                if (totalRows == 0) then
                        for z= 1, #earlyTable do
                        sql = [[
                                INSERT INTO earlyDB ("Sci_Name" ,"inv_Name", "img_Path",
"disc_Date" , "inv_Desc", "choice1", "choice2" , "choice3","level")
                        VALUES ("]] .. earlyTable[z].sciName ..
                        [[","]] .. earlyTable[z].invName ..
                        [[","]] .. earlyTable[z].imagePath ..
                        [[","]] .. earlyTable[z].discDate ..
                        [[","]] .. earlyTable[z].briefDesc ..
                        [[","]] .. earlyTable[z].choice1 ..
                        [[","]] .. earlyTable[z].choice2 ..
                        [[","]] .. earlyTable[z].choice3 ..
                        [[","]] .. earlyTable[z].level ..
                        [[");]]
                        db:exec(sql)
                                end
                end
db:close()
end
M.addEarlyQuestion = addEarlyQuestion


--add modern age question
--*****************************************
local function addModernQuestion()
        local path = system.pathForFile ( "playerDB.sqlite",system.DocumentsDirectory)
        local db = sqlite.open(path)
        local sql
        local modernTable = {}
        modernTable = questionTable.modernQuestion
        print ("TOTAL MODERN AGE QUESTIONS ".. #modernTable)

        local totalRows = 0
                for row in db:nrows("SELECT * FROM modernDB") do
                        totalRows = totalRows + 1
                end

                if (totalRows == 0) then
                        for z= 1, #modernTable do
                        sql = [[
                                INSERT INTO modernDB ("Sci_Name" ,"inv_Name", "img_Path",
"disc_Date" , "inv_Desc", "choice1", "choice2" , "choice3","level")
                        VALUES ("]] .. modernTable[z].sciName ..
                        [[","]] .. modernTable[z].invName ..
                        [[","]] .. modernTable[z].imagePath ..
                        [[","]] .. modernTable[z].discDate ..
                        [[","]] .. modernTable[z].briefDesc ..
                        [[","]] .. modernTable[z].choice1 ..
                        [[","]] .. modernTable[z].choice2 ..
                        [[","]] .. modernTable[z].choice3 ..
                        [[","]] .. modernTable[z].level ..
                        [[");]]
                        db:exec(sql)
                                end
                end
db:close()
end
M.addModernQuestion = addModernQuestion

return M
```

ADDING PLAYER'S PROFILE DATA

```
local sqlite = require ("sqlite3")

local M = {}

function addPlayer(tbl)

        local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
        local db = sqlite.open(path)

local sql = [[
INSERT INTO player_Data
("player_Name","current_Coins","coins_Acquired","achievement_pts","current_Level","current_Era","answer
ed_Correct","answered_Wrong",
"num_Hints_Used","game_Finished","slots_Last_Used","level_Tries","last_LevelTry","save_Status")
VALUES ("]] .. tbl.player_Name ..
[[","]] .. tbl.default_coins ..
[[","]] .. tbl.acquired_coins ..
[[","]] .. tbl.achieve_points ..
[[","]] .. tbl.default_level ..
[[","]] .. tbl.default_era ..
[[","]] .. tbl.answered_correct ..
[[","]] .. tbl.answered_wrong ..
[[","]] .. tbl.num_hints_used ..
[[","]] .. tbl.game_finished  ..
[[","]] .. tbl.slots_last_used ..
[[","]] .. tbl.level_Tries ..
[[","]] .. tbl.last_LevelTry ..
[[","]] .. tbl.save_status ..
[[");]]

db:exec(sql)
db:close()
print ("SUCCESSFULLY CREATED PLAYER DATA")
end
M.addPlayer = addPlayer

return M
```

ADDING PLAYER'S DEFAULT ACHIEVEMENTS DATA

```
local sqlite = require ("sqlite3")

local M = {}

function addAchievement(playID,tbl)

        local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
        local db = sqlite.open(path)
        local playID = playID

local sql = [[
INSERT INTO player_Achievement ("player_ID","coll_2k","coll_3k","coll_5k","lastColl","hint1","hint2","hint3",
"hint4","acientFin","middleFin","earlyFin","modernFin","gameFin","totalPoint")
VALUES ("]] .. playID ..
[[","]] .. tbl.coll2k ..
[[","]] .. tbl.coll3k ..
[[","]] .. tbl.coll5k ..
[[","]] .. tbl.coll10k ..
[[","]] .. tbl.hint1 ..
[[","]] .. tbl.hint2 ..
[[","]] .. tbl.hint3 ..
[[","]] .. tbl.hint4 ..
[[","]] .. tbl.ancientFin  ..
[[","]] .. tbl.middleFin ..
[[","]] .. tbl.earlyFin ..
[[","]] .. tbl.modernFin ..
[[","]] .. tbl.gameFin ..
[[","]] .. tbl.totalPoint ..
[[");]]

db:exec(sql)
db:close()
print ("SUCCESSFULLY CREATED PLAYER ACHIEVEMENT DATA")
end
M.addAchievement = addAchievement

return M
```

CHECK GAME COIN ACHIEVEMENT

```
local function checkMe(param)
```

```
          curCoin = param --change this accroding to module coins variable handler

--stack of coin achievement
                    if tonumber(curCoin) >= tonumber(collectCoin1) and tonumber(coll2k) == 0  then
                    --show overlay here
                              print ("coin achievement 1 complete")

                              totalPoint = tonumber(totalPoint + coll1Point)
                              pMe1 = 1
                              pMe2 = 0
                              pMe3 = 0
                              pMe4 = 0

                              coll2k = 1
                              updateAchCoin.updateAchCoin(playerID,pMe1,pMe2,pMe3,pMe4,totalPoint)

                              myData.whatAch = "Stack of coins" -- used for the achievement overlay
                              timer.performWithDelay ( 500, showAch )
                              return true
--pile of coin achievement
                    elseif curCoin >= collectCoin2 and tonumber(coll3k) == 0 then
                              --show overlay here
                              print ("coin achievement 2 complete")


                              totalPoint = totalPoint + coll2Point
                              pMe1 = 1
                              pMe2 = 1
                              pMe3 = 0
                              pMe4 = 0

                              coll3k = 1
                              updateAchCoin.updateAchCoin(playerID,pMe1,pMe2,pMe3,pMe4,totalPoint)

                              myData.whatAch = "Pile of coins" -- used for the achievement overlay

                              timer.performWithDelay ( 500, showAch )
                              return true
--bag of coin achievement
                    elseif curCoin >= collectCoin3 and tonumber(coll5k) == 0 then
                              --show overlay here
                              print ("coin achievement 3 complete")

                                        print ("player current point: " .. totalPoint)
                                        totalPoint = totalPoint + coll3Point
                                        pMe1 = 1
                                        pMe2 = 1
                                        pMe3 = 1
                                        pMe4 = 0
                                        coll5k = 1

          updateAchCoin.updateAchCoin(playerID,pMe1,pMe2,pMe3,pMe4,totalPoint)

                                        myData.whatAch = "Bag of coins" -- used for the achievement overlay

                                        timer.performWithDelay ( 500, showAch )
                                        return true
--chest of coin achievement
                    elseif curCoin >= collectCoin4 and tonumber(lastColl) == 0 then
                              --show overlay here
                              print ("coin achievement 4 complete")
                                        totalPoint = totalPoint + coll4Point
                                        pMe1 = 1
                                        pMe2 = 1
                                        pMe3 = 1
                                        pMe4 = 1
                                        lastColl = 1

          updateAchCoin.updateAchCoin(playerID,pMe1,pMe2,pMe3,pMe4,totalPoint)

                                        myData.whatAch = "Chest of coins" -- used for the achievement overlay


                                        timer.performWithDelay ( 500, showAch )
                                        return true
                    else
                                        return false
                    end
end
```

CHECK IF THE PLAYER UNLOCKED NEW ERA

```
local function checkUpLevel()
    local upCurEra = tonumber(upCurEra)
    local upCurLevel = tonumber(upCurLevel)
    local ancientLevel = myData.ancientLevel
    local middleLevel = myData.middleLevel
    local earlyLevel = myData.earlyLevel
    local modernLevel = myData.modernLevel
    local newLevel


--ancient era----------------------------------------------
--*******************************************************
    if upCurEra == 1 then --ancient era

            if upCurLevel ~= 10 then --player is not in era's last level
                    newLevel = upCurLevel + 1 -- add 1 level
                    updateLevel.upLevel(playerID,newLevel,upCurEra)
                    print ("player is in new level")
                            myData.lastQ = nil
                            myData.lastQ = {}
                            myData.qCount = 0
                            storyboard.gotoScene("gameRestrictions",{ --go to the next level
                            effect = "slideLeft",
                            time = 250,
                            })
            elseif upCurLevel == ancientLevel and upCurEra == 1 then --player is in last level,
unlocked next era
                    print ("player unlocked new era")
                            storyboard.gotoScene("congratsOverlay",{ --go to the next level
                            effect = "slideLeft",
                            time = 250,
                            })
            end
--middle ages----------------------------------------------
--*******************************************************
    elseif upCurEra == 2 then -- middle ages

            if upCurLevel ~= 10 then --player is not in era's last level
				newLevel = upCurLevel + 1 -- add 1 level
				updateLevel.upLevel(playerID,newLevel,upCurEra)
				print ("player is in new level")
					myData.lastQ = nil
					myData.lastQ = {}
					myData.qCount = 0
					storyboard.gotoScene("gameRestrictions",{ --go to the next level
					effect = "slideLeft",
					time = 250,
					})
			elseif upCurLevel == middleLevel and upCurEra == 2 then --player is in last level,
unlocked next era
				print ("player unlocked new era")
					storyboard.gotoScene("congratsOverlay",{ --go to the next level
					effect = "slideLeft",
					time = 250,
					})
			end
--early modern age-----------------------------------------
--*********************************************************
		elseif upCurEra == 3 then --early age

			if upCurLevel ~= 15 then --player is not in era's last level
				newLevel = upCurLevel + 1 -- add 1 level
				updateLevel.upLevel(playerID,newLevel,upCurEra)
				print ("player is in new level")
					myData.lastQ = nil
					myData.lastQ = {}
					myData.qCount = 0
					storyboard.gotoScene("gameRestrictions",{ --go to the next level
					effect = "slideLeft",
					time = 250,
					})
			elseif upCurLevel == earlyLevel and upCurEra == 3 then --player is in last level, unlocked
next era
				print ("player unlocked new era")
					storyboard.gotoScene("congratsOverlay",{ --go to the next level
					effect = "slideLeft",
					time = 250,
					})
			end
--modern age-------------------------------------------------
--**********************************************************
		elseif upCurEra == 4 then --modern age

			if upCurLevel ~= 15 then --player is not in era's last level
				newLevel = upCurLevel + 1 -- add 1 level
				updateLevel.upLevel(playerID,newLevel,upCurEra)
				print ("player is in new level")
					myData.lastQ = nil
					myData.lastQ = {}
					myData.qCount = 0
					storyboard.gotoScene("gameRestrictions",{ --go to the next level
					effect = "slideLeft",
					time = 250,
					})
			elseif upCurLevel == modernLevel and upCurEra == 4 then --player is in last level,
unlocked next era
				print ("player unlocked new era")
					storyboard.gotoScene("gameFinishScene1",{ --go to the next level
					effect = "slideLeft",
					time = 250,
                                        })
                    end
--nothing found------------------------------------------------
--*************************************************************
            else
             print ("Invalid Era")
            end
end
```

GIVE GAME COIN REWARD

```
local coinReward
local function giveReward()

                    print (upCurLevel)
                    print (upCurEra)
                    local upCurLevel = tonumber(upCurLevel)
                    local upCurEra = tonumber(upCurEra)
                    --for era 1
                    if upCurEra == 1 and upCurLevel == 1 then
                                coinReward = 30

                    elseif upCurEra == 1 and upCurLevel == 2 then
                                coinReward = 60

                    elseif upCurEra == 1 and upCurLevel == 3 then
                                coinReward = 90

                    elseif upCurEra == 1 and upCurLevel == 4 then
                                coinReward = 120

                    elseif upCurEra == 1 and upCurLevel == 5 then
                                coinReward = 150

                    elseif upCurEra == 1 and upCurLevel == 6 then
                                coinReward = 180

                    elseif upCurEra == 1 and upCurLevel == 7 then
                                coinReward = 210

                    elseif upCurEra == 1 and upCurLevel == 8 then
                                coinReward = 240

                    elseif upCurEra == 1 and upCurLevel == 9 then
                                coinReward = 270

                    elseif upCurEra == 1 and upCurLevel == 10 then
                                coinReward = 300
--era 2

                    elseif upCurEra == 2 and upCurLevel == 1 then
                                coinReward = 30

                    elseif upCurEra == 2 and upCurLevel == 2 then
                                coinReward = 60

                    elseif upCurEra == 2 and upCurLevel == 3 then
                                coinReward = 90

                    elseif upCurEra == 2 and upCurLevel == 4 then
```

```
                        coinReward = 120

                elseif upCurEra == 2 and upCurLevel == 5 then
                        coinReward = 150

                elseif upCurEra == 2 and upCurLevel == 6 then
                        coinReward = 180

                elseif upCurEra == 2 and upCurLevel == 7 then
                        coinReward = 210

                elseif upCurEra == 2 and upCurLevel == 8 then
                        coinReward = 240

                elseif upCurEra == 2 and upCurLevel == 9 then
                        coinReward = 270

                elseif upCurEra == 2 and upCurLevel == 10 then
                        coinReward = 300

--for era 3

                elseif upCurEra == 3 and upCurLevel == 1 then
                        coinReward = 30

                elseif upCurEra == 3 and upCurLevel == 2 then
                        coinReward = 60

                elseif upCurEra == 3 and upCurLevel == 3 then
                        coinReward = 90

                elseif upCurEra == 3 and upCurLevel == 4 then
                        coinReward = 120

                elseif upCurEra == 3 and upCurLevel == 5 then
                        coinReward = 150

                elseif upCurEra == 3 and upCurLevel == 6 then
                        coinReward = 180

                elseif upCurEra == 3 and upCurLevel == 7 then
                        coinReward = 210

                elseif upCurEra == 3 and upCurLevel == 8 then
                        coinReward = 240

                elseif upCurEra == 3 and upCurLevel == 9 then
                        coinReward = 270

                elseif upCurEra == 3 and upCurLevel == 10 then
                        coinReward = 300

                elseif upCurEra == 3 and upCurLevel == 11 then
                        coinReward = 330

                elseif upCurEra == 3 and upCurLevel == 12 then
                        coinReward = 360

                elseif upCurEra == 3 and upCurLevel == 13 then
                        coinReward = 390

                elseif upCurEra == 3 and upCurLevel == 14 then
                        coinReward = 420

                elseif upCurEra == 3 and upCurLevel == 15 then
                        coinReward =450


--era 4
                elseif upCurEra == 4 and upCurLevel == 1 then
                        coinReward = 30

                elseif upCurEra == 4 and upCurLevel == 2 then
                        coinReward = 60

                elseif upCurEra == 4 and upCurLevel == 3 then
                        coinReward = 90

                elseif upCurEra == 4 and upCurLevel == 4 then
                        coinReward = 120

                elseif upCurEra == 4 and upCurLevel == 5 then
                        coinReward = 150

                elseif upCurEra == 4 and upCurLevel == 6 then
                        coinReward = 180

                elseif upCurEra == 4 and upCurLevel == 7 then
                        coinReward = 210

                elseif upCurEra == 4 and upCurLevel == 8 then
                        coinReward = 240

                elseif upCurEra == 4 and upCurLevel == 9 then
                        coinReward = 270

                elseif upCurEra == 4 and upCurLevel == 10 then
                        coinReward = 300

                elseif upCurEra == 4 and upCurLevel == 11 then
                        coinReward = 330

                elseif upCurEra == 4 and upCurLevel == 12 then
                        coinReward = 360

                elseif upCurEra == 4 and upCurLevel == 13 then
                        coinReward = 390

                elseif upCurEra == 4 and upCurLevel == 14 then
                        coinReward = 420

                elseif upCurEra == 4 and upCurLevel == 15 then
                        coinReward = 450


                end

                print ("total Reward: " .. coinReward)
                upCurCoins = upCurCoins + coinReward
                upAcqCoins = upAcqCoins + coinReward
                local curTime = upSlotUsed
                if tonumber(upCurCoins) >= 999999 then
                        upCurCoins = 999999
                end
```

```
                        updateGameCoin.upGameCoin(playerID,"playerDB.sqlite",upCurCoins,
upAcqCoins,curTime)
                        textReward.text = coinReward
                        checkMe(upCurCoins)
end
```

DROPPING CONFETTI EFFECT

```
local conTbl={}
local choice
local ranConfetti = function()
            choice = math.random(1,6)
            local confetti

            if choice == 1 then
                        confetti = display.newImage( "images/bluetri.png" )
                        confetti.width = 9
                        confetti.height = 9
                        confetti.x = 2 + math.random( 0,310 ); confetti.y = 5
                        physics.addBody( confetti, { density=0.6, friction=0.6, bounce=.6, radius=5 } )
                        confetti.angularVelocity = math.random(800) - 400
                        confetti.isSleepingAllowed = false
                        conDrop:insert(confetti)

            elseif choice == 2 then
                        confetti = display.newImage( "images/green.png" )
                                                confetti.width = 9
                        confetti.height = 9
                        confetti.x = 2 + math.random( 0,310 ); confetti.y = 5
                        physics.addBody( confetti, { density=0.6, friction=0.6, bounce=.6, radius=5 } )
                        confetti.angularVelocity = math.random(600) - 300
                        confetti.isSleepingAllowed = false
                        conDrop:insert(confetti)

            elseif choice == 3 then
                        confetti = display.newImage( "images/greentri.png" )
                                                confetti.width = 9
                        confetti.height = 9
                        confetti.x = 2 + math.random( 0,310 ); confetti.y = 5
                        physics.addBody( confetti, { density=0.6, friction=0.6, bounce=.6, radius=5 } )
                        confetti.angularVelocity = math.random(600) - 300
                        confetti.isSleepingAllowed = false
                        conDrop:insert(confetti)

            elseif choice == 4 then
                        confetti = display.newImage( "images/purpletri.png" )
                                                confetti.width = 9
                        confetti.height = 9
                        confetti.x = 2 + math.random( 0,310 ); confetti.y = 5
                        physics.addBody( confetti, { density=0.6, friction=0.6, bounce=.6, radius=5 } )
                        confetti.angularVelocity = math.random(600) - 300
                        confetti.isSleepingAllowed = false
                        conDrop:insert(confetti)

            elseif choice == 5 then
                        confetti = display.newImage( "images/red.png" )
                                                confetti.width = 9
                        confetti.height = 9
                        confetti.x = 2 + math.random( 0,310 ); confetti.y = 5
                        physics.addBody( confetti, { density=0.6, friction=0.6, bounce=.6, radius=5 } )
                        confetti.angularVelocity = math.random(600) - 300
                        confetti.isSleepingAllowed = false
                        conDrop:insert(confetti)

            elseif choice == 6 then
                        confetti = display.newImage( "images/yellow.png" )
                                confetti.width = 9
                        confetti.height = 9
                        confetti.x = 2 + math.random( 0,310 ); confetti.y = 5
                        physics.addBody( confetti, { density=0.6, friction=0.6, bounce=.6, radius=5 } )
                        confetti.angularVelocity = math.random(600) - 300
                        confetti.isSleepingAllowed = false
                        conDrop:insert(confetti)

            end

            conTbl[#conTbl + 1] = confetti
    end
```

DELETE PLAYER PROFILE

```
local sqlite = require ( "sqlite3" )
local M = {}

local function deletePlayer(playerID)
        local path = system.pathForFile ("playerDB.sqlite",system.DocumentsDirectory )
        local db = sqlite.open(path)
        local playerID = playerID

        local sql = "UPDATE player_Data SET save_Status = 0 WHERE player_ID = " .. playerID
        db:exec(sql)
        db:close()
        print ( playerID .. "HAS BEEN DELETED")
end
M.deletePlayer = deletePlayer

return M
```

DYNAMIC DELETE PROFILE BUTTON

```
--dynamic functions where corresponds to the total active profile

for i = 1, #dataTblLength do
--puts all the plater data into another table
        playerData[i] =
        {
                player_ID = dataTblLength[i].playerID,
                player_Name = dataTblLength[i].playerName,
                current_Coins = dataTblLength[i].currentCoins,
                coins_Acquired = dataTblLength[i].acquiredCoins,
                achievement_pts = dataTblLength[i].achievementPoints,
                current_Level = dataTblLength[i].currentLevel,
                current_Era = dataTblLength[i].currentEra,
                answered_Correct = dataTblLength[i].answeredCorrect,
                answered_Wrong = dataTblLength[i].answeredWrong,
                num_Hints_Used = dataTblLength[i].numHintsUsed,
                game_Finished = dataTblLength[i].gameFinished,
                slots_Last_Used = dataTblLength[i].slotsLastUsed,
                save_Status = dataTblLength[i].saveStatus,
            }

--creates a dynamic callback function which depends on how many active player the game has
--this callback functions fires if the player tapped the save slots in ui
            loadSaveGame[i] = function(event)
                        local phase = event.phase
                                    if phase == "ended" then

                                                            if enableSound == true then
                                                                        audio.play(tapSound)
                                                            end

                                                playerToDelete = playerData[i].player_ID
                                                print ("PLAYER ID TO DELETE: " .. playerToDelete)
                                    --msgbox that confirms save file deletion
                                    local alert = native.showAlert (
                                    "Delete save game",
                                    "Are you sure that you want to delete this save file? Deleted data cannot be
recovered." ,
                                     {"Yes", "No"},
                                     onAlertInteract  -- callback function
                                    )
                                    end
            end

end

local t = totalSave
local topPos = 100

--if the number of save profiles exceeds to 3,
--because the save profiles increments everytime a new user is created and the save slots depends on it
--if the save profiles exceeds to 3, the dynamic slot creater will create other save slots
if t >= 3 then
            t= 3
end
--creates a dynamic save slots
for i = 1 , t  do
              local slotName = ("saveSlot" .. i)
                        slotName = widget.newButton
                                    {
                                                left = 50,
                                                top =topPos ,
                                                width = 230,
                                                height = 100,
                                                defaultFile = "images/loadslot.png",
                                                overFile = "images/loadslotpress.png",
                                                id = ("load_slot" .. i),
                                                onEvent = loadSaveGame[i],
                                    }
button:insert(slotName)
topPos = topPos + 110
end

--displays the players info into the save slots,
local txtPlayerName = {}
local txtPlayerEra = {}
local txtPlayerLevel = {}
local txtPlayerCoins = {}
local txtPos = 0
local curEra
```

```
for i = 1, #dataTblLength do
--displays the player name
        txtPlayerName[i] = display.newText (playerData[i].player_Name, 100, 100, fontStyle,  _H * 0.04 )
        txtPlayerName[i]:setReferencePoint (display.CenterLeftReferencePoint)
        txtPlayerName[i].x = 140 ; txtPlayerName[i].y = 112 + txtPos
        txtPlayerName[i]:setTextColor ( 255, 180, 25)

--displays what era the player is
        if tonumber(playerData[i].current_Era) == 1 then
                curEra ="Ancient Period"
        elseif tonumber(playerData[i].current_Era) == 2 then
                curEra = "Middle Ages"
        elseif tonumber(playerData[i].current_Era) == 3 then
                curEra = "Early Modern Ages"
        elseif tonumber(playerData[i].current_Era) == 4 then
                curEra = "Modern Age"
        end

                txtPlayerEra[i] = display.newText(curEra, 100, 100, fontStyle, _H * 0.03 )
                txtPlayerEra[i]:setReferencePoint (display.CenterLeftReferencePoint)
                txtPlayerEra[i].x = 140 ; txtPlayerEra[i].y = 135 + txtPos
                txtPlayerEra[i]:setTextColor (255, 180, 25)

--displays what level the player is
                txtPlayerLevel[i] =display.newText( playerData[i].current_Level, 100,100,fontStyle,_H *
0.04)
                txtPlayerLevel[i]:setReferencePoint ( display.CenterLeftReferencePoint)
                txtPlayerLevel[i].x= 140 ; txtPlayerLevel[i].y= 155 + txtPos
                txtPlayerLevel[i]:setTextColor(255,180,25)
--displays      player current coins
                txtPlayerCoins[i] = display.newText( playerData[i].current_Coins,100, 100, fontStyle, _H *
0.04 )
                txtPlayerCoins[i]:setReferencePoint (display.CenterLeftReferencePoint )
                txtPlayerCoins[i].x = 140 ; txtPlayerCoins[i].y = 180 + txtPos
                txtPlayerCoins[i]:setTextColor ( 255, 180, 25 )

                txtPos= txtPos + 112

        button:insert(txtPlayerName[i])
        button:insert(txtPlayerEra[i])
        button:insert(txtPlayerLevel[i])
        button:insert(txtPlayerCoins[i])
```

LOAD PLAYER’S ACHIEVEMENT DATA

```
--gets the`player achievement data

local sqlite = require ("sqlite3")
local myData = require ("myData")


local M = {}

function getPlayerAch(filename , player_ID)
        local path = system.pathForFile (filename, system.DocumentsDirectory)
        local db = sqlite.open(path)
        local tblName = "player_Achievement"
        local colName = "player_ID"
        local playerID = player_ID--myData.currentPlayerID

        local upPlayerAch = {}
```

```
	local myTable = {}

		local sql ="SELECT * FROM " .. tblName .. " WHERE " .. colName .. " = " .. playerID

		for row in db:nrows(sql) do
			myTable = row
				return myTable
		end

	db:close()

	upPlayerAch = myTable


	return upPlayerAch

end
M.getPlayerAch = getPlayerAch
return M
```

LOAD PLAYER DATA

```
--this will get the player data everytime this module was called, in short it is updating for every changes done to the database

local sqlite = require ("sqlite3")
local myData = require ("myData")
local loadNewPlayerData = require ("loadNewPlayerData")

local M = {}

function getPlayerDat(filename , player_ID)
	local path = system.pathForFile (filename, system.DocumentsDirectory)
	local db = sqlite.open(path)
	local tblName = "player_Data"
	local colName = "player_ID"
	local playerID = player_ID--myData.currentPlayerID

	local upPlayerDat = {}
	local myTable = {}

		local sql ="SELECT * FROM " .. tblName .. " WHERE " .. colName .. " = " .. playerID

		for row in db:nrows(sql) do
			myTable = row
				return myTable
		end
	db:close()

	upPlayerDat = myTable


	return upPlayerDat

end
M.getPlayerDat = getPlayerDat
return M
```

SHUFFLE DATA TABLE

```
local function shuffle(t)

        for i = iterations, 2, -1 do

        j = math.random(i)
                t[i], t[j] = t[j], t[i]
        end
end
```

PLACE CHOICES IN RANDOM LETTER

```
local function initAns(param1,param2,param3,param4)
        local ansTable = {param1,param2,param3,param4}

        shuffle(ansTable) --shuffle the table data

        local leftPos
        local topPos
        ans = {}

                for i=1,#ansTable do --prints all the data on the table,
                                if i == 1 then
                                                leftPos = 50
                                                topPos = 365
                                                ans[i] = display.newText( ansTable[i], 100, 100,
fontStyle, _W * 0.03 )
                                                ans[i]:setReferencePoint (
display.CenterLeftReferencePoint )
                                                ans[i]:setTextColor ( 255, 255, 255 )
                                                ans[i].x = leftPos ; ans[i].y = topPos

                                                aButton.id = ansTable[i]

                                elseif i == 2 then
                                                leftPos = 200
                                                topPos = 365
                                                ans[i] = display.newText( ansTable[i], 100, 100,
fontStyle, _W * 0.03 )
                                                ans[i]:setReferencePoint (
display.CenterLeftReferencePoint )
                                                ans[i]:setTextColor ( 255, 255, 255  )
                                                ans[i].x = leftPos ; ans[i].y = topPos

                                                bButton.id = ansTable[i]

                                elseif i == 3 then
                                                leftPos = 50
                                                topPos = 430
                                                ans[i] = display.newText( ansTable[i], 100, 100,
fontStyle, _W * 0.03 )
                                                ans[i]:setReferencePoint (
display.CenterLeftReferencePoint )
                                                ans[i]:setTextColor ( 255, 255, 255  )
                                                ans[i].x = leftPos ; ans[i].y = topPos

                                                cButton.id = ansTable[i]

                                elseif i == 4 then
```

```
                                                            leftPos = 200
                                                            topPos = 430
                                                            ans[i] = display.newText( ansTable[i], 100, 100,
fontStyle, _W * 0.03 )
                                                            ans[i]:setReferencePoint (
display.CenterLeftReferencePoint )
                                                            ans[i]:setTextColor ( 255, 255, 255  )
                                                            ans[i].x = leftPos ; ans[i].y = topPos

                                                            dButton.id = ansTable[i]

                                    end
                                            button:insert(ans[i])
                        end
end
```

GET RANDOM QUESTION FROM DATA TABLE

```
local function getQuestion()
        upCurEra = tonumber(upCurEra)
        local ran

        local curQuestion = nil
        qTable = {}
        qTable = myData.lastQ

--for ancient era--------------------------------------------
--------------------------------------------------------------
                if upCurEra == 1 then
                        print ("GENERATING ANCIENT ERA QUESTION")
                        if #qTable == 0 then
                        qTable  = generateQuestion.genQuestion("ancientDB",upCurLevel)
                        print ("LEVEL " .. upCurLevel .." questions: " .. #qTable)
                        shuffle(qTable) --shuffles the table
                        myData.lastQ = qTable --save the table in myData
                        qTable = myData.lastQ
                        ran = myData.qCount + 1
                        myData.qCount = ran
                                for i = 1, #qTable do
                                        local a = qTable[i].invName
                                        print (a)
                                end
                        else
                        qTable = myData.lastQ
                                for i = 1, #qTable do
                                        local a = qTable[i].invName
                                        print (a)
                                end
                        ran = myData.qCount + 1
                        myData.qCount = ran
                        end


                        qID = qTable[ran].qID
                        sciName = qTable[ran].sciName
                        invName = qTable[ran].invName
                        imagePath = qTable[ran].imagePath
                        discDate = qTable[ran].discDate
                        briefDesc = qTable[ran].briefDesc
                        choice1 = qTable[ran].choice1
                        choice2 = qTable[ran].choice2
                        choice3 = qTable[ran].choice3


                        initAns(sciName,choice1,choice2,choice3)
                        hintName.text = invName
                        print ("INVENTION NAME: " ..invName)
                        print ("INVENTOR: " ..sciName)
                        print ("QUESTION ID: " .. qID)

                        invImg = display.newImageRect ( imagePath  , 135, 135 )
                        invImg.x = _W * 0.5 ; invImg.y = 183
                        screenGroup:insert(invImg)

--for middle age---------------------------------------------------
-----------------------------------------------------------------
                elseif upCurEra == 2 then
                        print ("GENERATING MIDDLE AGE QUESTION")
                        if #qTable == 0 then
                        qTable  = generateQuestion.genQuestion("middleDB",upCurLevel)
                        print ("LEVEL " .. upCurLevel .." questions: " .. #qTable)
                        shuffle(qTable) --shuffles the table
                        myData.lastQ = qTable --save the table in myData
                        qTable = myData.lastQ
                        ran = myData.qCount + 1
                        myData.qCount = ran
                                for i = 1, #qTable do
                                        local a = qTable[i].invName
                                        print (a)
                                end
                        else
                        qTable = myData.lastQ
                                for i = 1, #qTable do
                                        local a = qTable[i].invName
                                        print (a)
                                end
                        ran = myData.qCount + 1
                        myData.qCount = ran
                        end

                        qID = qTable[ran].qID
                        sciName = qTable[ran].sciName
                        invName = qTable[ran].invName
                        imagePath = qTable[ran].imagePath
                        discDate = qTable[ran].discDate
                        briefDesc = qTable[ran].briefDesc
                        choice1 = qTable[ran].choice1
                        choice2 = qTable[ran].choice2
                        choice3 = qTable[ran].choice3

                        initAns(sciName,choice1,choice2,choice3)
                        hintName.text = invName
                        print ("INVENTION NAME: " ..invName)
                        print ("INVENTOR: " ..sciName)
                        print ("QUESTION ID: " .. qID)

                        invImg = display.newImageRect ( imagePath  , 135, 135 )
                        invImg.x = _W * 0.5 ; invImg.y = 183
                        screenGroup:insert(invImg)
--for early modern-----------------------------------------
----------------------------------------------------------
```

```
		elseif upCurEra == 3 then
			print ("GENERATING EARLY MODERN AGE QUESTION")
			if #qTable == 0 then
			qTable  = generateQuestion.genQuestion("earlyDB",upCurLevel)
			print ("LEVEL " .. upCurLevel .." questions: " .. #qTable)
			shuffle(qTable) --shuffles the table
			myData.lastQ = qTable --save the table in myData
			qTable = myData.lastQ
			ran = myData.qCount + 1
			myData.qCount = ran
				for i = 1, #qTable do
					local a = qTable[i].invName
					print (a)
				end
			else
			qTable = myData.lastQ
				for i = 1, #qTable do
					local a = qTable[i].invName
					print (a)
				end
			ran = myData.qCount + 1
			myData.qCount = ran
			end

			qID = qTable[ran].qID
			sciName = qTable[ran].sciName
			invName = qTable[ran].invName
			imagePath = qTable[ran].imagePath
			discDate = qTable[ran].discDate
			briefDesc = qTable[ran].briefDesc
			choice1 = qTable[ran].choice1
			choice2 = qTable[ran].choice2
			choice3 = qTable[ran].choice3

			initAns(sciName,choice1,choice2,choice3)
			hintName.text = invName
			print ("INVENTION NAME: " ..invName)
			print ("INVENTOR: " ..sciName)
			print ("QUESTION ID: " .. qID)

			invImg = display.newImageRect ( imagePath  , 135, 135 )
			invImg.x = _W * 0.5 ; invImg.y = 183
			screenGroup:insert(invImg)
--for modern age-----------------------------------------------
----------------------------------------------------------------
		elseif upCurEra == 4 then
			print ("GENERATING MODERN AGE QUESTION")
			if #qTable == 0 then
			qTable  = generateQuestion.genQuestion("modernDB",upCurLevel)
			print ("LEVEL " .. upCurLevel .." questions: " .. #qTable)
			shuffle(qTable) --shuffles the table
			myData.lastQ = qTable --save the table in myData
			qTable = myData.lastQ
			ran = myData.qCount + 1
			myData.qCount = ran
				for i = 1, #qTable do
					local a = qTable[i].invName
					print (a)
				end
			else
			qTable = myData.lastQ
				for i = 1, #qTable do
```

```
                    local a = qTable[i].invName
                    print (a)
                end
            ran = myData.qCount + 1
            myData.qCount = ran
            end

            qID = qTable[ran].qID
            sciName = qTable[ran].sciName
            invName = qTable[ran].invName
            imagePath = qTable[ran].imagePath
            discDate = qTable[ran].discDate
            briefDesc = qTable[ran].briefDesc
            choice1 = qTable[ran].choice1
            choice2 = qTable[ran].choice2
            choice3 = qTable[ran].choice3

            initAns(sciName,choice1,choice2,choice3)
            hintName.text = invName
            print ("INVENTION NAME: " ..invName)
            print ("INVENTOR: " ..sciName)
            print ("QUESTION ID: " .. qID)

            invImg = display.newImageRect ( imagePath  , 135, 135 )
            invImg.x = _W * 0.5 ; invImg.y = 183
            screenGroup:insert(invImg)

        end
```

CHECKS IF THE PLAYER TAPPED THE CORRECT ANSWER

```
local upCor
local fName
local function checkAns1(event)
    local phase = event.phase
        if phase  == "ended" then
                if enableSound == true then
                    audio.play(tapSound)
                end
            timer.cancel (timerEvent)
                    if aButton.id == sciName then
                        print ("corrent answer")
                        upCor = nil
                        fName = nil
                         upCor = upAnsCor +1
                         fName = "answered_Correct"
    myData.year1 = discDate
    myData.name1 = sciName
    myData.invent1 = invName
                        updateCorWro.upCorWro(playerID,fName,upCor)
                        correctAnswer()
                    else
                        print ("wrong answer")
                        upCor = nil
                        fName = nil
                         upCor = upAnsWro +1
                         fName = "answered_Wrong"
    myData.year1 = discDate
    myData.name1 = sciName
    myData.invent1 = invName
                        updateCorWro.upCorWro(playerID,fName,upCor)
                        wrongAnswer()
                    end
        end
end

local function checkAns2(event)
    local phase = event.phase
        if phase  == "ended" then
                if enableSound == true then
                    audio.play(tapSound)
                end
            timer.cancel (timerEvent)
                    if bButton.id == sciName then
                        print ("corrent answer")
                        upCor = nil
                        fName = nil
                         upCor = upAnsCor +1
                         fName = "answered_Correct"
    myData.year1 = discDate
    myData.name1 = sciName
    myData.invent1 = invName
                        updateCorWro.upCorWro(playerID,fName,upCor)
                        correctAnswer()
                    else
                        print ("wrong answer")
                        upCor = nil
                        fName = nil
                         upCor = upAnsWro +1
                         fName = "answered_Wrong"
    myData.year1 = discDate
    myData.name1 = sciName
    myData.invent1 = invName
                        updateCorWro.upCorWro(playerID,fName,upCor)
                        wrongAnswer()
                    end
        end
end

local function checkAns3(event)
    local phase = event.phase
        if phase  == "ended" then
                if enableSound == true then
                    audio.play(tapSound)
                end
            timer.cancel (timerEvent)
                    if cButton.id == sciName then
                        print ("corrent answer")
                        upCor = nil
                        fName = nil
                         upCor = upAnsCor +1
                         fName = "answered_Correct"
    myData.year1 = discDate
    myData.name1 = sciName
    myData.invent1 = invName
                        updateCorWro.upCorWro(playerID,fName,upCor)
                        correctAnswer()
                    else
                        print ("wrong answer")
                        upCor = nil
                        fName = nil
```

```
                                                 upCor = upAnsWro +1
                                                 fName = "answered_Wrong"
          myData.year1 = discDate
          myData.name1 = sciName
          myData.invent1 = invName
                                                updateCorWro.upCorWro(playerID,fName,upCor)
                                                wrongAnswer()
                                        end
                    end
end

local function checkAns4(event)
          local phase = event.phase
                    if phase  == "ended" then
                                        if enableSound == true then
                                                  audio.play(tapSound)
                                        end
                              timer.cancel (timerEvent)
                                                  if dButton.id == sciName then
                                                            print ("corrent answer")
                                                            upCor = nil
                                                            fName = nil
                                                             upCor = upAnsCor +1
                                                             fName = "answered_Correct"
          myData.year1 = discDate
          myData.name1 = sciName
          myData.invent1 = invName
                                                            updateCorWro.upCorWro(playerID,fName,upCor)
                                                            correctAnswer()
                                                  else
                                                            print ("wrong answer")
                                                            upCor = nil
                                                            fName = nil
                                                             upCor = upAnsWro +1
                                                             fName = "answered_Wrong"
          myData.year1 = discDate
          myData.name1 = sciName
          myData.invent1 = invName
                                                            updateCorWro.upCorWro(playerID,fName,upCor)
                                                            wrongAnswer()
                                                  end
                    end
end
```

DELETE RANDOM WRONG ANSWER

```
local function deleteBogus()
                    --removes the other choices randomly
local butTbl = {aButton,bButton,cButton,dButton}
local f = 1
local lastRan = 0

          repeat
                    local ran = math.random(1,4)
                    local button = butTbl[ran]
                              if ran ~= lastRan then
                                                  if tostring(button.id) ~= tostring(sciName) then
                                                            lastRan = ran
                                                            button.alpha = 0
                                                            ans[ran].text = ""
                                                            f = f + 1
                                        end
```

```
                        end
        until f == 3
end

local function hideBogusComplete(event)
        if "clicked" == event.action then
                local i = event.index
                        if i == 1 then
                                alertS3 = 0
                                upCurCoins = upCurCoins - fifHintCost
                                playerCoin.text = upCurCoins
                                playerCoin.text = upCurCoins
                                upHintUsed = upHintUsed + 1
                                print ("number of hints used: " .. upHintUsed)
                                fifButton:setEnabled(false)
                                deleteBogus()
                                hintUsed.hintUse(playerID,upCurCoins,upHintUsed)
                                checkMe(upHintUsed)
                        end
        elseif "cancelled" == event.action then
                        alertS3 = 0
        end
end
```

GET QUESTIONS FROM DATABASE

```
--this module will generate the question for the player, according to its era
local sqlite = require ( "sqlite3" )

local M = {}

local function genQuestion(dbName,pLevel)
        local path = system.pathForFile ("playerDB.sqlite",system.DocumentsDirectory)
        local db = sqlite.open(path)
        local sql
        local dataBase = dbName
        local level = pLevel
        local eraQuestion = {}

sql = ("SELECT * FROM " .. dataBase .. " WHERE level = " .. level .. [[ ORDER BY RANDOM() LIMIT 15]])

        for row in db:nrows(sql) do
                eraQuestion[#eraQuestion + 1] =
{
                qID = row.question_ID,
                sciName = row.Sci_Name,
                invName = row.inv_Name,
                imagePath = row.img_Path,
                discDate = row.disc_Date,
                briefDesc = row.inv_Desc,
                choice1 = row.choice1,
                choice2 = row.choice2,
                choice3 = row.choice3,
}
        end
        db:close()
        return eraQuestion
end
M.genQuestion = genQuestion
return M
```

GET TOP TEN PLAYER FOR LEADERBOARDS

```
--get the total achievement point of player and sort is in descending order

local sqlite = require ( "sqlite3" )

local M = {}

local function getAchPoint()

                local path = system.pathForFile ("playerDB.sqlite" , system.DocumentsDirectory)
                local db = sqlite.open(path)

                local myTable = {}
                local result = {}

        local sql =[[SELECT player_Achievement.player_ID, player_Data.player_Name,
                player_Achievement.coll_2k,player_Achievement.coll_3k,player_Achievement.coll_5k,
                player_Achievement.lastColl,player_Achievement.hint1,player_Achievement.hint2,
                player_Achievement.hint3,player_Achievement.hint4,player_Achievement.acientFin,
                player_Achievement.middleFin,player_Achievement.earlyFin,

        player_Achievement.modernFin,player_Achievement.gameFin,player_Achievement.totalPoint
        FROM player_Data INNER JOIN player_Achievement ON player_Data.player_ID =
player_Achievement.player_ID
        WHERE totalPoint <> 0 ORDER BY totalPoint DESC LIMIT 10]]

                for row in db:nrows(sql) do
                        myTable[#myTable+1] =
                        {
                                playerID = row.player_ID,
                                playerName = row.player_Name,
                                coll2k = row.coll_2k,
                                coll3k = row.coll_3k,
                                coll5k = row.coll_5k,
                                coll10k = row.lastColl,
                                hint1 = row.hint1,
                                hint2 = row.hint2,
                                hint3 = row.hint3,
                                hint4 = row.hint4,
                                ancientFin = row.acientFin,
                                middleFin = row.middleFin,
                                earlyFin = row.earlyFin,
                                modernFin = row.modernFin,
                                gameFin = row.gameFin,
                                totalPoint = row.totalPoint,
                        }
                end

                db:close()


        result = myTable

        return result


end
M.getAchPoint =getAchPoint
```

```
return M
```

CHECK IF THE PLAYER TYPED A VALID NAME

```
function checkForSaveSlot(event)
local phase = event.phase
if "ended" == phase then
print ("next press and released")
if enableSound == true then
audio.play(tapSound)
end
if pTextField.text == "" or pTextField.text == "Enter your nickname" then
--remove the comments when testing on device---------------------
checkIfBlank() -- uncomment this when building on device
else --there is text on the textfield / uncomment this when building on device
pName = pTextField.text --uncomment this when building on device
--pName = "newPlayer" --comment this when testing on device
if (pName:match('%"')) then
print ("Invalid")
alertDisplay()
elseif (pName:match('%+')) then
print ("Invalid")
alertDisplay()
elseif (pName:match('%-')) then
print ("Invalid")
alertDisplay()
elseif (pName:match('%/')) then
print ("Invalid")
alertDisplay()
elseif (pName:match('%*')) then
print ("Invalid")
alertDisplay()
elseif (pName:match("%A+%A")) then --checks if there are special characters and numbers
print ("Invalid player name")
alertDisplay()
else --valid  name

myData.playerName = pName -- pass the value this is used for delete player
print ("player name: " .. pName)
if totalSave <= 2 then --go to the player main menu if there are save slot
t = {
player_Name = pName,

default_coins = myData.default_coins,

acquired_coins = myData.acquired_coins,

achieve_points = myData.achieve_points,

default_level = myData.default_level,

default_era = myData.default_era,

answered_correct = myData.answered_correct,

answered_wrong = myData.answered_wrong,

num_hints_used = myData.num_hints_used,

game_finished = myData.game_finished,
slots_last_used = myData.slots_last_used,

level_Tries = myData.level_Tries,

last_LevelTry = myData.last_LevelTry,

save_status = myData.save_status,

}

addPlayerData.addPlayer(t) --add new player in db
getLastPlayer.getLast() --add new player achievements data
updateSaveCounter.upSaveCounter() --updates the save counter
myData.loadPlayerData = true-- tells loadNewPlayerData module to load the newly created save game

storyboard.gotoScene ( "scene1",{
effect = "slideLeft",
time = 250,
} )
else --go to delete save player profile

local alert = native.showAlert (
"Maximum save space reached.",
"In order to create a new profile, please delete other save profile." ,
{"Yes", "No"},
onAlertInteract  -- callback function
)

end
end
end
native.setKeyboardFocus ( nil ) --dismissess the keyboard
end
end
```

CONVERT OS TIME INTO TIME FORMAT

```
local function lockedSlotMachine(event)
        local phase = event.phase
                if phase == "ended" then
                        print ("slot machine is not available")
                end
end

--slot machtime time count down
local curTime = os.time ()
        print ("os time: " .. curTime)
local slotLastUse = upSlotUsed
        print ("slot last used: " .. slotLastUse)
local coolTime = "Slot Ready"

local function timeCount(numSec)
        local nSeconds = numSec
                if nSeconds == 0 then

                        coolTime.text = "Slot Ready";
                else
                        local nHours = string.format("%02.f", math.floor(nSeconds/3600));
                        local nMins = string.format("%02.f", math.floor(nSeconds/60 - (nHours*60)));
```

```
                              local nSecs = string.format("%02.f", math.floor(nSeconds - nHours*3600 - nMins
*60));

                              coolTime.text =  nHours..":"..nMins..":"..nSecs
                    end
end

--checks if the slot machine available or not
local function checkSlotTime()

          local timeSince = (curTime - slotLastUse)
    if timeSince >= slotCoolDown then
        print ("slot is ready")
        coolTime.text = "Slot Ready"
        slotMachineButton.alpha  = 1
        slotNotMachineButton.alpha = 0
     else
          slotLastUse = slotLastUse - 1
          print (slotLastUse)
          timeCount(slotCoolDown-timeSince)
          print ("slot is NOT ready")
           slotNotMachineButton.alpha = 1
           slotMachineButton.alpha  = 0
    end
end
```

SLOT MACHINE

```
local function endRoll()
          -- Set default winnings for spin
           winnings = 0
           if enableSound == true then
           audio.pause ( slotSpinSound )
           end
          canSpin = false
          -- Stop slides from moving
          slide:pause()
          slide2:pause()
          slide3:pause()

          -- Check for matches
          if (slide.currentFrame == 7 and slide2.currentFrame == 7 and slide3.currentFrame == 7) then -- All
three are 7's
                    winnings = 500
                    print ("you get " .. winnings)
          elseif (slide.currentFrame == 2 and slide2.currentFrame == 2 and slide3.currentFrame == 2) then --
All three are Bar's
                    winnings = 200
                    print ("you get " .. winnings)
          elseif (slide.currentFrame == 3 and slide2.currentFrame == 3 and slide3.currentFrame == 3) then --
All three are Bell's
                    winnings =  100
                    print ("you get " .. winnings)
          elseif (slide.currentFrame == 6 and slide2.currentFrame == 6 and slide3.currentFrame == 6) then --
All three are Watermelon's
                    winnings =  50
                    print ("you get " .. winnings)
          elseif ( (slide.currentFrame == 1 or slide.currentFrame == 4 or slide.currentFrame == 5) and
(slide2.currentFrame == 1 or slide2.currentFrame == 4 or slide2.currentFrame == 5) and
(slide3.currentFrame == 1 or slide3.currentFrame == 4 or slide3.currentFrame == 5) ) then -- All three are
Cherry's
                    winnings =  25
                    print ("you get " .. winnings)
          end

                    --if the player has won
                    if winnings > 0 then
                              slotCongrats.alpha = 1
                              showWin.text = winnings
                              showWin.alpha = 1

                              slotTapStart.alpha = 0
                              slotTryAgain.alpha = 0
                              slotNoSpin.alpha = 0
                              if enableSound == true then
                                        audio.play(kaChingSound)
                              end
                              --textFadeIn(winnings)
                    elseif winnings == 0 then
                              slotCongrats.alpha = 0
                              showWin.alpha = 0

                              slotTapStart.alpha = 0
                              slotTryAgain.alpha = 1
                              slotNoSpin.alpha = 0
                    end

          -- Set and display new total
          totalcoins = totalcoins  + winnings
          if tonumber(totalcoins) >= 999999 then
                    totalcoins = 999999
          end

          checkMe()

          curTime =os.time()
          updateGameCoin.upGameCoin(playerID,"playerDB.sqlite",totalcoins, playerAcqCoins,curTime)

          playerAcqCoins = playerAcqCoins + winnings
          print ("player acquired coins: " .. playerAcqCoins)
          moneyTxt.text = totalcoins
          lifeTxt.text = "Spins left: ".. totalspin



          if totalspin == 0 then
                    slotCongrats.alpha = 0
                    showWin.alpha = 0

                    slotTapStart.alpha = 0
                    slotTryAgain.alpha = 0
                    slotNoSpin.alpha = 1

          end

          -- Let user spin again
          canSpin = true

end

local function rollslide()
```

```
        if totalspin ~= 0 then
                if canSpin == true  then
                        if enableSound == true then
                                slotSpinSound = audio.play(slotSound)
                        end
                        totalspin = totalspin - bet
                -- Start spinning all three slides
                        slide:play()
                        slide2:play()
                        slide3:play()

                -- Random spin time
                        randomTime = math.random(1500, 3500)
                        timer.performWithDelay(randomTime, endRoll, 1)

                else
                        canSpin = false
                end
        end

end
```

TABLE VIEW

```
module(..., package.seeall)

--properties
 local screenW, screenH = display.contentWidth, display.contentHeight
local viewableScreenW, viewableScreenH = display.viewableContentWidth, display.viewableContentHeight
local screenOffsetW, screenOffsetH = display.contentWidth -  display.viewableContentWidth,
display.contentHeight - display.viewableContentHeight

local currentTarget, detailScreen, velocity, currentDefault, currentOver, prevY
local startTime, lastTime, prevTime = 0, 0, 0

--methods

function showHighlight(event)
    local timePassed = system.getTimer() - startTime

    if timePassed > 100 then
        print("highlight")
        currentDefault.isVisible = false
        currentOver.isVisible = true
        Runtime:removeEventListener( "enterFrame", showHighlight )
    end
end

function newListItemHandler(self, event)
        local t = currentTarget --could use self.target.parent possibly
        local phase = event.phase


        local default = self.default
        local over = self.over
        local top = self.top
        local bottom = self.bottom
        local upperLimit, bottomLimit = top, screenH - currentTarget.height - bottom
```

```
                        local result = true

        if( phase == "began" ) then
            -- Subsequent touch events will target button even if they are outside the stageBounds of button
            display.getCurrentStage():setFocus( self )
            self.isFocus = true

            startPos = event.y
            prevPos = event.y
            delta, velocity = 0, 0
            if currentTarget.tween then transition.cancel(currentTarget.tween) end

            Runtime:removeEventListener("enterFrame", scrollList )
            Runtime:addEventListener("enterFrame", moveCat)

                                -- Start tracking velocity
                                Runtime:addEventListener("enterFrame", trackVelocity)

            if over then
                currentDefault = default
                currentOver = over
                startTime = system.getTimer()
                Runtime:addEventListener( "enterFrame", showHighlight )
            end

        elseif( self.isFocus ) then

            if( phase == "moved" ) then

                Runtime:removeEventListener( "enterFrame", showHighlight )
                if over then
                    default.isVisible = true
                    over.isVisible = false
                end

                delta = event.y - prevPos
                prevPos = event.y
                if ( t.y > upperLimit or t.y < bottomLimit ) then
                    t.y  = t.y + delta/2
                else
                    t.y = t.y + delta
                end

            elseif( phase == "ended" or phase == "cancelled" ) then

                                        lastTime = event.time

                local dragDistance = event.y - startPos
                --velocity = delta
                                        Runtime:removeEventListener("enterFrame", moveCat)
                                        Runtime:removeEventListener("enterFrame", trackVelocity)
                Runtime:addEventListener("enterFrame", scrollList )

                local bounds = self.stageBounds
                local x, y = event.x, event.y
                local isWithinBounds = bounds.xMin <= x and bounds.xMax >= x and bounds.yMin <= y and
bounds.yMax >= y

                -- Only consider this a "click", if the user lifts their finger inside button's stageBounds
                if isWithinBounds and (dragDistance < 10 and dragDistance > -10 ) then
                                                velocity = 0
                    result = self.onRelease(event)
            end

            -- Allow touch events to be sent normally to the objects they "hit"
            display.getCurrentStage():setFocus( nil )
            self.isFocus = false

            if over then
               default.isVisible = true
               over.isVisible = false
               Runtime:removeEventListener( "enterFrame", showHighlight )
            end
         end
      end

      return result
end

function newListItem(params)
      local data = params.data
      local default = params.default
      local over = params.over
      local onRelease = params.onRelease
      local top = params.top
      local bottom = params.bottom
      local callback = params.callback
      local id = params.id

      local thisItem = display.newGroup()

      if params.default then
            default = display.newImage( params.default )
            thisItem:insert( default )
            default.x = default.width*.5 - screenOffsetW
            thisItem.default  = default
      end

      if params.over then
            over = display.newImage( params.over )
            over.isVisible = false
            thisItem:insert( over )
            over.x = over.width*.5 - screenOffsetW
            thisItem.over = over
      end

      thisItem.id = id
      thisItem.data = data
      thisItem.onRelease = onRelease
      thisItem.top = top
      thisItem.bottom = bottom

      local t = callback(data)
      thisItem:insert( t )

      thisItem.touch = newListItemHandler
      thisItem:addEventListener( "touch", thisItem )

      return thisItem
end

function newList(params)
                  local textSize = 16
      local data = params.data
      local default = params.default
      local over = params.over
      local onRelease = params.onRelease
      local top = params.top or 20
      local bottom = params.bottom or 48
      local cat = params.cat
      local order = params.order or {}
      local categoryBackground = params.categoryBackground
      local backgroundColor = params.backgroundColor
      local callback = params.callback or function(item)
                                              local t = display.newText(item, 0, 0, native.systemFontBold, textSize)
                                              t:setTextColor(255, 255, 255)
                                              t.x = math.floor(t.width/2) + 20
                                              t.y = 24
                                              return t
                                                                              end

      --setup the list view
      local listView = display.newGroup()
      local prevY, prevH = 0, 0


      if cat then
                      local catTable = {}

          --get the implicit categories
          local prevCat = 0
          for i=1, #data do
                  if data[i][cat] ~= prevCat then
                      table.insert(catTable, data[i][cat])
                      prevCat = data[i][cat]
                  end
          end

          if order then
                  --clean up the user provided order table by removing any empty categories
                  local n = 1
                  while n < #order do
                      if not in_table(order[n], catTable) then
                            table.remove(order, n)
                      else
                            n = n + 1
                      end
                   end

                  --add any categories not specified to the user order of categories
                  for i=1, #catTable do
                      if not in_table(catTable[i], order) then
                            table.insert(order, catTable[i])
                      end
                   end
                  else
                            order = catTable
          end

      end

      local j = 1
      local c = {}
      local offset = 12
      while true do
          local h = order[j]
```

```
            if h then
                        local g = display.newGroup()
                        local b
                        if categoryBackground then
                                    b = display.newImage(categoryBackground, true)
                        else
                                    b = display.newRect(0, 0, screenW, textSize*1.5)
                                    b:setFillColor(0, 0, 0, 100)
                        end
                        g:insert( b )

                                                local labelShadow = display.newText( h, 0, 0, native.systemFontBold,
textSize )
                                                labelShadow:setTextColor( 0, 0, 0, 128 )
                                                g:insert( labelShadow, true )
                                                labelShadow.x = labelShadow.width*.5 + 1 + offset + screenOffsetW*.5
                                                labelShadow.y = textSize*.8 + 1

                        local t = display.newText(h, 0, 0, native.systemFontBold, textSize)
                      t:setTextColor(255, 255, 255)
             g:insert( t )
             t.x = t.width*.5 + offset + screenOffsetW*.5
             t.y = textSize*.8

             listView:insert( g )
             g.x = 0
             g.y = prevY + prevH
                    prevY = g.y
                    prevH = g.height
                    table.insert(c, g)
                    c[#c].yInit = g.y
                 end

                 --iterate over the data and add items to the list view
                 for i=1, #data do
                        if data[i][cat] == h then
                         local thisItem = newListItem{
                             data = data[i],
                             default = default,
                             over = over,
                             onRelease = onRelease,
                             top = top,
                             bottom = bottom,
                             callback = callback,
                             id = i
                         }

                         listView:insert( 1, thisItem )

                         thisItem.x = 0 + screenOffsetW*.5
                         thisItem.y = prevY + prevH

                         --save the Y and height
                         prevY = thisItem.y
                         prevH = thisItem.height
                    end --if
                 end --for

                        j = j + 1

                        if not order[j] then break end
```

```
     end --while

     if backgroundColor then
         local bgColor = display.newRect(0, 0, screenW, screenH)
         bgColor:setFillColor(backgroundColor[1], backgroundColor[2], backgroundColor[3])
              bgColor.width = listView.width
              bgColor.height = listView.height
              bgColor.y = bgColor.height*.5
         listView:insert(1, bgColor)
            end

     listView.y = top
     listView.top = top
     listView.bottom = bottom
     listView.c = c

     currentTarget = listView

                  function listView:cleanUp()
                           print("tableView cleanUp")
                           Runtime:removeEventListener("enterFrame", moveCat )
                           Runtime:removeEventListener("enterFrame", scrollList )
       Runtime:removeEventListener( "enterFrame", showHighlight )
                           Runtime:removeEventListener("enterFrame", trackVelocity)
                           local i
                           for i = listView.numChildren, 1, -1 do
                                    --test
                                    listView[i]:removeEventListener("touch", newListItemHandler)
                                    listView:remove(i)
                                    listView[i] = nil
                           end
                  end

                  function listView:scrollTo(yVal, timeVal)
       local timeVal = timeVal or 400
       local yVal = yVal or 0

       velocity = 0
       Runtime:removeEventListener("enterFrame", scrollList )
       Runtime:addEventListener("enterFrame", moveCat )

       self.tween = transition.to(self, { time=timeVal, y=yVal, transition=easing.outQuad})

                  end

     return listView
end

function scrollList(event)
                  local friction = 0.9
                  local timePassed = event.time - lastTime
                  lastTime = lastTime + timePassed

     --turn off scrolling if velocity is near zero
     if math.abs(velocity) < .01 then
          velocity = 0
          Runtime:removeEventListener("enterFrame", scrollList )
     end

     velocity = velocity*friction

     currentTarget.y = math.floor(currentTarget.y + velocity*timePassed)
```

```
	moveCat()

	local upperLimit = currentTarget.top
	local bottomLimit = screenH - currentTarget.height - currentTarget.bottom

	if ( currentTarget.y > upperLimit ) then
		velocity = 0
		Runtime:removeEventListener("enterFrame", scrollList )
		Runtime:addEventListener("enterFrame", moveCat )
		currentTarget.tween = transition.to(currentTarget, { time=400, y=upperLimit,
transition=easing.outQuad})
	elseif ( currentTarget.y < bottomLimit and bottomLimit < 0 ) then
		velocity = 0
		Runtime:removeEventListener("enterFrame", scrollList )
		Runtime:addEventListener("enterFrame", moveCat )
		currentTarget.tween = transition.to(currentTarget, { time=400, y=bottomLimit,
transition=easing.outQuad})
	elseif ( currentTarget.y < bottomLimit ) then
		velocity = 0
		Runtime:removeEventListener("enterFrame", scrollList )
		Runtime:addEventListener("enterFrame", moveCat )
		currentTarget.tween = transition.to(currentTarget, { time=400, y=upperLimit,
transition=easing.outQuad})
	end

	return true
end

function moveCat()
	local upperLimit = currentTarget.top

			for i=1, #currentTarget.c do
				if( currentTarget.y > upperLimit - currentTarget.c[i].yInit ) then
					currentTarget.c[i].y = currentTarget.c[i].yInit
				end

				if ( currentTarget.y < upperLimit - currentTarget.c[i].yInit ) then
					currentTarget.c[i].y = upperLimit - currentTarget.y
				end

				if( i > 1 ) then
					if ( currentTarget.c[i].y < currentTarget.c[i-1].y + currentTarget.c[i].height
) then
						currentTarget.c[i-1].y = currentTarget.c[i].y -
currentTarget.c[i].height
					end
				end
			end

			return true
end

function trackVelocity(event)
		local timePassed = event.time - prevTime
		prevTime = prevTime + timePassed

		if prevY then
			velocity = (currentTarget.y - prevY)/timePassed
		end
		prevY = currentTarget.y
end
```

```
--look for an item in a table
function in_table ( e, t )
	for _,v in pairs(t) do
		if (v==e) then return true end
	end
	return false
end
```

TABLE UI

```
module(..., package.seeall)

-----------------
-- Helper function for newButton utility function below
local function newButtonHandler( self, event )

	local result = true

	local default = self[1]
	local over = self[2]

	-- General "onEvent" function overrides onPress and onRelease, if present
	local onEvent = self._onEvent

	local onPress = self._onPress
	local onRelease = self._onRelease

	local buttonEvent = {}
	if (self._id) then
		buttonEvent.id = self._id
	end

	local phase = event.phase
	if "began" == phase then
		if over then
			default.isVisible = false
			over.isVisible = true
		end

		if onEvent then
			buttonEvent.phase = "press"
			result = onEvent( buttonEvent )
		elseif onPress then
			result = onPress( event )
		end

		-- Subsequent touch events will target button even if they are outside the stageBounds of
button
		display.getCurrentStage():setFocus( self, event.id )
		self.isFocus = true

	elseif self.isFocus then
		local bounds = self.stageBounds
		local x,y = event.x,event.y
		local isWithinBounds =
			bounds.xMin <= x and bounds.xMax >= x and bounds.yMin <= y and
bounds.yMax >= y

		if "moved" == phase then
```

```
                        if over then
                                -- The rollover image should only be visible while the finger is within
button's stageBounds
                                default.isVisible = not isWithinBounds
                                over.isVisible = isWithinBounds
                        end

                elseif "ended" == phase or "cancelled" == phase then
                        if over then
                                default.isVisible = true
                                over.isVisible = false
                        end

                        if "ended" == phase then
                                -- Only consider this a "click" if the user lifts their finger inside button's
stageBounds
                                if isWithinBounds then
                                        if onEvent then
                                                buttonEvent.phase = "release"
                                                result = onEvent( buttonEvent )
                                        elseif onRelease then
                                                result = onRelease( event )
                                        end
                                end
                        end

                        -- Allow touch events to be sent normally to the objects they "hit"
                        display.getCurrentStage():setFocus( self, nil )
                        self.isFocus = false
                end
        end

        return result
end


---------------
-- Button class

function newButton( params )
        local button, default, over, size, font, textColor, offset

        if params.default then
                button = display.newGroup()
                default = display.newImage( params.default )
                button:insert( default, true )
        end

        if params.over then
                over = display.newImage( params.over )
                over.isVisible = false
                button:insert( over, true )
        end

        -- Public methods
        function button:setText( newText )

                local labelText = self.text
                if ( labelText ) then
                        labelText:removeSelf()
                        self.text = nil
                end
                local labelShadow = self.shadow
                if ( labelShadow ) then
                        labelShadow:removeSelf()
                        self.shadow = nil
                end

                local labelHighlight = self.highlight
                if ( labelHighlight ) then
                        labelHighlight:removeSelf()
                        self.highlight = nil
                end

                if ( params.size and type(params.size) == "number" ) then size=params.size else size=20
end
                if ( params.font ) then font=params.font else font=native.systemFontBold end
                if ( params.textColor ) then textColor=params.textColor else textColor={ 255, 255, 255,
255 } end

                -- Optional vertical correction for fonts with unusual baselines (I'm looking at you, Zapfino)
                if ( params.offset and type(params.offset) == "number" ) then offset=params.offset else
offset = 0 end

                if ( params.emboss ) then
                        -- Make the label text look "embossed" (also adjusts effect for textColor
brightness)
                        local textBrightness = ( textColor[1] + textColor[2] + textColor[3] ) / 3

                        labelHighlight = display.newText( newText, 0, 0, font, size )
                        if ( textBrightness > 127) then
                                labelHighlight:setTextColor( 255, 255, 255, 20 )
                        else
                                labelHighlight:setTextColor( 255, 255, 255, 140 )
                        end
                        button:insert( labelHighlight, true )
                        labelHighlight.x = labelHighlight.x + 1.5; labelHighlight.y = labelHighlight.y + 1.5 +
offset
                        self.highlight = labelHighlight

                        labelShadow = display.newText( newText, 0, 0, font, size )
                        if ( textBrightness > 127) then
                                labelShadow:setTextColor( 0, 0, 0, 128 )
                        else
                                labelShadow:setTextColor( 0, 0, 0, 20 )
                        end
                        button:insert( labelShadow, true )
                        labelShadow.x = labelShadow.x - 1; labelShadow.y = labelShadow.y - 1 + offset
                        self.shadow = labelShadow
                end

                labelText = display.newText( newText, 0, 0, font, size )
                labelText:setTextColor( textColor[1], textColor[2], textColor[3], textColor[4] )
                button:insert( labelText, true )
                labelText.y = labelText.y + offset
                self.text = labelText
        end
        if params.text then
                button:setText( params.text )
        end

        if ( params.onPress and ( type(params.onPress) == "function" ) ) then
                button._onPress = params.onPress
```

```
	end
	if ( params.onRelease and ( type(params.onRelease) == "function" ) ) then
		button._onRelease = params.onRelease
	end

	if (params.onEvent and ( type(params.onEvent) == "function" ) ) then
		button._onEvent = params.onEvent
	end

	-- Set button as a table listener by setting a table method and adding the button as its own table
listener for "touch" events
	button.touch = newButtonHandler
	button:addEventListener( "touch", button )

	if params.x then
		button.x = params.x
	end

	if params.y then
		button.y = params.y
	end

	if params.id then
		button._id = params.id
	end

	return button
end


--------------
-- Label class

function newLabel( params )
	local labelText
	local size, font, textColor, align
	local t = display.newGroup()

	if ( params.bounds ) then
		local bounds = params.bounds
		local left = bounds[1]
		local top = bounds[2]
		local width = bounds[3]
		local height = bounds[4]

		if ( params.size and type(params.size) == "number" ) then size=params.size else size=20
end
		if ( params.font ) then font=params.font else font=native.systemFontBold end
		if ( params.textColor ) then textColor=params.textColor else textColor={ 255, 255, 255,
255 } end
		if ( params.offset and type(params.offset) == "number" ) then offset=params.offset else
offset = 0 end
		if ( params.align ) then align = params.align else align = "center" end

		if ( params.text ) then
			labelText = display.newText( params.text, 0, 0, font, size )
			labelText:setTextColor( textColor[1], textColor[2], textColor[3], textColor[4] )
			t:insert( labelText )
			-- TODO: handle no-initial-text case by creating a field with an empty string?

			if ( align == "left" ) then
				labelText.x = left + labelText.stageWidth * 0.5
				elseif ( align == "right" ) then
					labelText.x = (left + width) - labelText.stageWidth * 0.5
				else
					labelText.x = ((2 * left) + width) * 0.5
				end
			end

			labelText.y = top + labelText.stageHeight * 0.5

			-- Public methods
			function t:setText( newText )
				if ( newText ) then
					labelText.text = newText

					if ( "left" == align ) then
						labelText.x = left + labelText.stageWidth / 2
					elseif ( "right" == align ) then
						labelText.x = (left + width) - labelText.stageWidth / 2
					else
						labelText.x = ((2 * left) + width) / 2
					end
				end
			end
			function t:setTextColor( r, g, b, a )
				local newR = 255
				local newG = 255
				local newB = 255
				local newA = 255

				if ( r and type(r) == "number" ) then newR = r end
				if ( g and type(g) == "number" ) then newG = g end
				if ( b and type(b) == "number" ) then newB = b end
				if ( a and type(a) == "number" ) then newA = a end

				labelText:setTextColor( r, g, b, a )
			end
		end

		-- Return instance (as display group)
		return t
end
```

UPDATE PLAYER DATA

```
local sqlite = require ( "sqlite3" )

local M = {}

local function updateAchCoin(playerID,params1,params2,params3,params4,params5)


		local path = system.pathForFile ("playerDB.sqlite", system.DocumentsDirectory)
		local db = sqlite.open(path)

		local sql = "UPDATE player_Achievement SET coll_2k = " .. params1 .. ", coll_3k = " .. params2 .. ",
coll_5k = " .. params3 .. " , lastColl = " .. params4 .. " , totalPoint = " .. params5 .. " WHERE player_ID = " ..
playerID
print (sql)
		db:exec(sql)
		print ("successfully updated player achievement DB")
		db:close()
return true
end
M.updateAchCoin = updateAchCoin

return M


local sqlite = require ( "sqlite3" )

local M = {}

local function updateAchEra(playerID,params1,params2)


		local path = system.pathForFile ("playerDB.sqlite", system.DocumentsDirectory)
		local db = sqlite.open(path)

		local sql = "UPDATE player_Achievement SET " ..params1.. " = 1 , totalPoint = " .. params2 .. "
WHERE player_ID = " .. playerID
		print (sql)
		db:exec(sql)
		print ("successfully updated player achievement DB")
		db:close()
return true
end
M.updateAchEra = updateAchEra

return M


local sqlite = require ( "sqlite3" )

local M = {}

local function updateAchFinish(playerID,params1,params2)


		local path = system.pathForFile ("playerDB.sqlite", system.DocumentsDirectory)
		local db = sqlite.open(path)

		local sql = "UPDATE player_Achievement SET gameFin = " .. params1 .. " , totalPoint = " ..
params2 .. " WHERE player_ID = " .. playerID
		print (sql)
		db:exec(sql)
		print ("successfully updated player achievement DB")
		db:close()
return true
end
M.updateAchFinish = updateAchFinish

return M


local sqlite = require ( "sqlite3" )

local M = {}

local function updateAchHint(playerID,params1,params2,params3,params4,params5)
local path = system.pathForFile ("playerDB.sqlite", system.DocumentsDirectory)
		local db = sqlite.open(path)
```

```
            local sql = "UPDATE player_Achievement SET hint1 = " .. params1 .. ", hint2 = " .. params2 .. ",
hint3 = " .. params3 .. " , hint4 = " .. params4 .. " , totalPoint = " .. params5 .. " WHERE player_ID = " ..
playerID
            print (sql)
            db:exec(sql)
            print ("successfully updated player achievement DB")
            db:close()
return true
end
M.updateAchHint = updateAchHint

return M


local sqlite = require ( "sqlite3" )

local M = {}

local function upCorWro(playerID,fieldName,dataTo)
            local playerID = playerID
            local fieldName = fieldName
            local dataTo = dataTo
            local path = system.pathForFile ("playerDB.sqlite",system.DocumentsDirectory)
            local db = sqlite.open(path)

            local sql = "UPDATE player_Data SET " .. fieldName .. " = " .. dataTo .. " WHERE player_ID = " ..
playerID

            db:exec(sql)
            db:close()

end
M.upCorWro = upCorWro

return M


local sqlite= require("sqlite3")

local M = {}

function upGameCoin(playerID, fileName,coins, acqCoins,curTime)
            local sql
            local coins = coins
            local acqCoins = acqCoins
            local tblName = "player_Data"
            local colCoin = "current_Coins"
            local acqCoin = "coins_Acquired"
            local slotLast = "slots_Last_Used"
            local curTime = curTime

            --if tonumber(coins) >= 999999 then
                        --coins = 999999
            --end

            local path = system.pathForFile (fileName,system.DocumentsDirectory)
            local db = sqlite.open(path)

            sql = "UPDATE " .. tblName .. " SET " .. colCoin .. " = " .. coins .. "," .. acqCoin .. " = " .. acqCoins ..
",".. slotLast .. " = " .. curTime .. " WHERE player_ID = " .. playerID
            print (sql)
            db:exec(sql)
```

```
        db:close()
        print ("successfully updated")
end
M.upGameCoin = upGameCoin

return M


local sqlite = require ( "sqlite3" )

local M = {}

local function upLevel(playerID,newLevel,newEra)

        local playerID = playerID
        local newLevel = newLevel
        local newEra = newEra

        local path = system.pathForFile ("playerDB.sqlite", system.DocumentsDirectory)
        local db = sqlite.open(path)

        local sql = "UPDATE player_Data SET current_Level = " .. newLevel .. " , current_Era = " .. newEra
.. " WHERE player_ID = " .. playerID

        db:exec(sql)
        db:close()
        print ("level successfully updated")
end
M.upLevel = upLevel

return M
```