LUA API Documentation

KOTRYNA VYSOCKYTĖ

422010

17TH OF APRIL 2019

TEACHER:

YVENS REBOUÇAS SERPA

Contents

Introduction

1. Assignment Description

Lua scripting course is about game engineers learning Lua scripting language and implementing it in custom C++ engine. Assignments is about creating a 2D card game that allow designers to manipulate data in Lua. Main goal was to learn how LUA api can be binned in C++. I have made my own custom 2D engine that has Lua api with which a designer can create custom scripts for cards and give custom data.

2. Card game rules

There are two types of cards in this game:

1. Spell Card
2. Attack Card

Every attack card has information of attack, health, mana, name and description. Mana is shown on top left, attack on bottom left and health on bottom right. The spell cards only have mana information.

The player is fighting only one enemy that has certain amount of health on the top left of the screen. It does damage of 1 every turn. However, enemy has a 50% to hit a card or to hit players health.

Player has health and mana that is shown on the bottom left corner of the window. After every player turn +1 mana is added. As some cards cost more money to be spelled the mana being added every turn to the player.

Player can attack or use a card only once per turn. If a card is being destroyed by an enemy the new one is being drawn from the deck.

The spell cards that are allowed:

1. Freeze (freezes enemy for n amount of turns)
2. Heal (adds health to the player)
3. Boost Damage (boost damage of all cards)
4. Reduce Mana (reduces mana it takes to use a card)

LUA API

3. Card.lua

Card
+Info{} {Name="", Damage=0, Health=0, Description="", Mana=0}
+OnEnable() +OnDisable() +OnCreate()

Every Card Lua script has these basic settings.

It can be used such as Boost.lua template did.

```lua
function OnEnable()
    Card.boost(2);
end

function OnDisable()

end

function OnCreate()
    Boost = Card.create("Boost", "Assets/Card_Boost.png")
    Card.show(Boost, true)
    Info = {Damage = 0,Health=0,Mana = 5,Name="Boost", Description = "Adds +2 Damage To every attack card"}
end
```

Info{}

Table that populates information of each card that gets passed to engine when OnCreate the card is called in C++;

Function OnEnable()

This function gets called when the card is used for attack or spell. The custom information can be added such as add boost or freeze player.

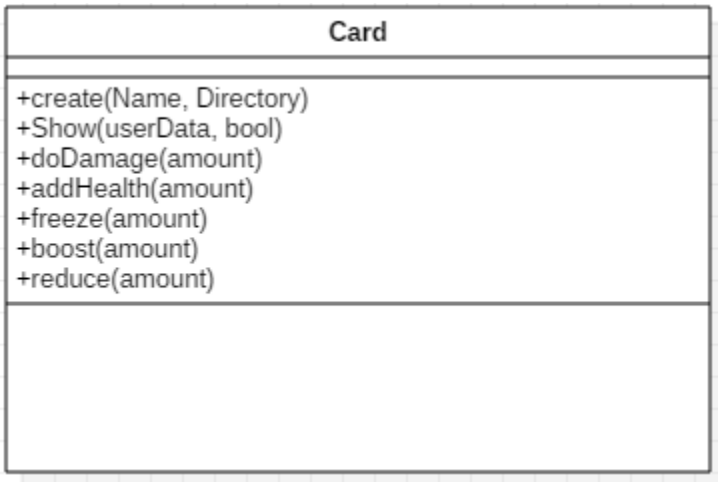
Function OnDisable()

This function gets called when the card is being destroyed. The custom information can be added such as add boost or freeze enemy and so on.

Function OnCreate()

This function is being called by the engine at the beginning of creating cards. Most of the time this card contains all information that the Gameobject stores.

4. UserData of the Card



This is userData library that can pass information to specific object.

Create(name,directory)

Creates a new gameobject.

Show(userData,bool)

Passes a value to the Card and defines if this card is going to be shown at the beginning.

doDamage(amount)

Pushes a damage amount to do to enemy.

addHealth(amount)

Pushes a variable to engine and adds health to player.

Freeze(amount)

Pushes a variable to engine that freezes enemies turns for certain amount of turns.

Boost(amount)

Pushed a variable to engine that boosts the damage by the amount of all cards.

Reduce(amount)

Pushes a variable to engine that reduces all mana used for all cards for one turn.

5. C++

Will only show lua api functions that were used. As everything else is pretty insignificant.

Void registerFunctions()

```
void State_Game::registerFunctions()
{
    lua_newtable(_state);
        lua_pushcfunction(_state, State_Game::createSprite);
            lua_setfield(_state, -2, "create");
        lua_pushcfunction(_state, State_Game::positionSprite);
            lua_setfield(_state, -2, "position");
        lua_pushcfunction(_state, State_Game::scaleSprite);
            lua_setfield(_state, -2, "scale");
    lua_setglobal(_state, "Sprite");

    lua_newtable(_state);
        lua_pushcfunction(_state, State_Game::createCard);
            lua_setfield(_state, -2, "create");
        lua_pushcfunction(_state, State_Game::showCardAtStart);
            lua_setfield(_state, -2, "show");
        lua_pushcfunction(_state, State_Game::doDamage);
            lua_setfield(_state, -2, "doDamage");
        lua_pushcfunction(_state, State_Game::addHealth);
            lua_setfield(_state, -2, "addHealth");
        lua_pushcfunction(_state, State_Game::freezeEnemy);
            lua_setfield(_state, -2, "freeze");
        lua_pushcfunction(_state, State_Game::boostDamage);
            lua_setfield(_state, -2, "boost");
        lua_pushcfunction(_state, State_Game::reduceMana);
            lua_setfield(_state, -2, "reduce");
    lua_setglobal(_state, "Card");

}
```

Registering all functions that allows to add functions to lua.

Void initialiseLua()

All Lua contexts are held in this structure. Loading lua libraries, registering functions and cards.

```
void State_Game::initialiseLua()
{
    _state = luaL_newstate();
    luaL_openlibs(_state); /* Load Lua libraries */
    registerFunctions();
    registerCards();

}
```

Void startLua()

```
void State_Game::startLua()
{
    lua_getglobal(_state, "OnCreate");
    if (lua_isfunction(_state, -1))
    {
        if (lua_pcall(_state, 0, 0, 0) != 0)
        {
            printf("Error %s\n", lua_tostring(_state, -1));
            exit(-1);
        }
    }

    lua_getglobal(_state, "Info");
    int damage = getIntField(_state, "Damage");
    int health = getIntField(_state, "Health");
    int mana = getIntField(_state, "Mana");
    std::string title = getStringField(_state, "Name");
    std::string description = getStringField(_state, "Description");


    GetGameObject(title)->SetDamage(damage);
    GetGameObject(title)->SetHealth(health);
    GetGameObject(title)->SetManaCost(mana);
    GetGameObject(title)->SetDescription(description);
}
```

Getting a function from lua called OnCreate and reading the table called info where I set that information to each gameobject.

Void updateLua()

```
void State_Game::updateLua()
{
    lua_getglobal(_state, "Update");
    if (lua_isfunction(_state, -1))
    {
        if (lua_pcall(_state, 0, 0, 0) != 0)
        {
            printf("Error %s\n", lua_tostring(_state, -1));
            exit(-1);
        }
    }
}
```

Getting a lua function called update that later is being called in State_Game update function where it's being updated every frame.

Void registerCards()

```
void State_Game::registerCards()
{
    std::string directory = "LUA";
    for (const auto & entry : fs::directory_iterator(directory))
    {
        std::cout << entry.path().string() << std::endl;
        luaL_loadfile(_state, entry.path().string().c_str());
        lua_call(_state, 0, 0);
        startLua();
        lua_close;
    }
    //DEBUG WITH ONE SCRIPT
    //luaL_loadfile(_state, "LUA/Boost.lua");
    //lua_call(_state, 0, 0);
    //startLua();
}
```

Getting all lua scipts from a LUA folder and updating all info needed by calling startLua().

Void OnEnableCard and void OnDisableCard

```
void State_Game::OnEnableCard()
{
    lua_getglobal(_state, "OnEnable");
    if (lua_isfunction(_state, -1))
    {
        if (lua_pcall(_state, 0, 0, 0) != 0)
        {
            printf("Error %s\n", lua_tostring(_state, -1));
            exit(-1);
        }
    }
}

void State_Game::OnDisableCard()
{
    lua_getglobal(_state, "OnDisable");
    if (lua_isfunction(_state, -1))
    {
        if (lua_pcall(_state, 0, 0, 0) != 0)
        {
            printf("Error %s\n", lua_tostring(_state, -1));
            exit(-1);
        }
    }
}
```

Just a method where I get lua functions that are called OnDisable and OnCreate.

int createSprite(lua_State* pState)

```
int State_Game::createSprite(lua_State* pState)
{
    if (lua_gettop(pState))
    {
        std::string name = (std::string) lua_tostring(pState, 1);
        std::string directory = (std::string) lua_tostring(pState, 2);

        sf::Texture texture;
        texture.loadFromFile(directory);
        std::cout << "Sprite " << name << " has been craeted from " << directory << std::endl;
        sf::Sprite sprite;
        sprite.setTexture(texture);
        GameObject** gameObject = (GameObject**)lua_newuserdata(pState, sizeof(GameObject*));
        *gameObject = new GameObject(sprite, texture, directory, name);

        luaL_getmetatable(pState, "Sprite");
        lua_setmetatable(pState, -2);
        _gameObjects->push_back(*gameObject);
        return 1;
    }
    return luaL_error(pState, "Sprite.create(name, directory), faulty arguments.");
}
```

Creating userdata of object. Getting name directory from the stack as strings to set them to gameobject data. Mainly just creating sprites that all information is defined in LUA.

Int scaleSprite(lua_State* pState)

```
int State_Game::scaleSprite(lua_State* pState)
{
    if (lua_gettop(pState))
    {
        sf::Vector2f scale = sf::Vector2f(lua_tonumber(pState, 2), lua_tonumber(pState, 3));
        GameObject** obj = (GameObject**)lua_touserdata(pState, 1);
        (*obj)->Scale(scale);
        return 0;
    }
    return luaL_error(pState, "Sprite.scale(name, scale), faulty arguments.");
}
```

Using metastable to get the specific gameobject and scale it.

Int createCard(lua_State* pState)

```
int State_Game::createCard(lua_State* pState)
{
    if (lua_gettop(pState))
    {
        std::string name = (std::string) lua_tostring(pState, 1);
        std::string directory = (std::string) lua_tostring(pState, 2);

        sf::Texture texture;
        texture.loadFromFile(directory);
        std::cout << "Card " << name << " has been craeted from " << directory << std::endl;
        sf::Sprite sprite;
        sprite.setTexture(texture);
        GameObject** gameObject = (GameObject**)lua_newuserdata(pState, sizeof(GameObject*));
        *gameObject = new GameObject(sprite, texture, directory, name);

        luaL_getmetatable(pState, "Card");
        lua_setmetatable(pState, -2);
        _gameObjects->push_back(*gameObject);
        return 1;
    }
    return luaL_error(pState, "Card.create(name, directory,damage,mana,health), faulty arguments.");
}
```

Similar to CreateSprite method. Just that created gameobjects and stores it in gameobject vector.

Int getIntField(lua_State* pState, const char* key)

```
int State_Game::getIntField(lua_State* L, const char* key)
{
    lua_pushstring(L, key);
    lua_gettable(L, -2);

    int result = (int)lua_tonumber(L, -1);
    lua_pop(L, 1);
    return result;
}
```

Reading table information that hold integers.

string getStringField(lua_State* pState, const char* key)

```
std::string State_Game::getStringField(lua_State* L, const char* key)
{
    lua_pushstring(L, key);
    lua_gettable(L, -2);

    std::string result = lua_tostring(L, -1);
    lua_pop(L, 1);
    return result;
}
```

Instead of int in table we get strings.

Spell methods

As all methods for spell use similar method I will show only one that has same logic of handling lua parameters.

```
int State_Game::addHealth(lua_State* pState)
{
    if (lua_gettop(pState))
    {
        int amount = lua_tonumber(pState, 1);
        //add health
        return 0;
    }
    return luaL_error(pState, "Card.addHealth(amount), faulty arguments.");
}
```

Future improvements

As my goal was to have the whole game be scripted in Lua I would like to improve on it and get it to work after assignment. One point I have notices it's way more useful to use metatables and userdata than tables itself. The reading of data is handled easier and it gives less problems on the way.