Lab Exercise: Networking with WebSockets

Learning Outcomes

At the end of this lab you will be able to:

- Create a custom message
- Start a game session between two players (one Android client, one HTML client)
- Update a player's state locally and remotely

Note: before starting, be sure to use the Python server and HTML client inside the zip archive with this lab sheet.

Task 1 – Starting a Game Session

The first thing players need to do is to join a session. When the Android client starts up, it will automatically send a message of type **join** (with no other parameters) to the server.

The server will respond with a **gameState** message with a data value of 0 if you are the first player to join the game. The second player to join the game also gets a gameState message but with a data value of 1. Any subsequent client connections receive an error message (as opposed to state) and a data field of "no available space, two players already in the game".

Task 2 – Updating state

In a multiplayer game, you need to synchronise what is happening on both player's screens. One way of doing this is to send specific game state data in a message.

To do this, a custom messages type can be made, in this case an **updateState** message.

We are going to capture the x,y location of a scene touch event and send those coordinates, via the server, to the other connected client as an **updateState** message (we will also handle incoming updateState messages).

1. In the NetworkScene class, the method `addEvents()` captures scene touch events.

```
void NetworkScene::addEvents()
{
	//Create a "one by one" touch event listener (processes one touch at a time)
	auto listener1 = EventListenerTouchOneByOne::create();
	listener1->setSwallowTouches(true);

	listener1->onTouchBegan = [this](Touch* touch, Event* event){
		this->m_pos = touch->getLocation();
		return true;
	};

	cocos2d::Director::getInstance()->getEventDispatcher()->
				addEventListenerWithFixedPriority(listener1, 30);
}
```

Variable `m_pos` is declared as a `cocos2d::Vec2` in the private section of class NetworkScene – this variable stores the location of the touch event. The above method is called from inside method `NetworkScene::init()`.

2. There is a method in class NetworkScene, called `updateState()`. When the user touches on the scene, method `updateState()` is called from the scene touch handler.

3. Inside method `updateState()`,the x and y values of the touch event are printed to the debug window like this:

```
CCLOG("Local touch x,y pos = : %d, %d", this->m_pos.x, this->m_pos.y);
```

(CCLOG is a little archaic in that it uses conventions from the C language. %d is a format specifier in C which is short for decimal. There are two decimal values to output (x,y positions), so two format specifiers are needed).

Continuing inside method `updateState()`, we need to encode the x,y touch positions as a JSON object and send an **updateState** message to the other client.

```
std::string touchX = std::to_string(this->m_pos.x);
std::string touchY = std::to_string(this->m_pos.y);
std::string jsonMessage = "{\"type\": \"updateState\", \"data\" : {\"x\":" +
touchX + ", \"y\":" + touchY + "}}";
mSocket->send(jsonMessage);
```

So the message will look like this:

```
{ “type”: “updateState”, “data” : { “x” : 100, “y” : 100 } }
```

Task 3 – Handling updateState messages from the server

In this task, we want to display any message received from the server as text on the main client window. The server currently responds to the client with two attribute/value pairs, **type** and **data**

We want to show the attribute/value pairs as text on the main client window (e.g. “Received….type: updateState, data: x 20, y 100”)

Note that in the case of updateState messages, the data payload is no longer a string, it is in fact a JSON object. The `NetworkScene::parseJSONMessage()` checks the received message type (either gameState, error or updateState) and updates the data field accordingly:

```
void NetworkScene::parseJSONMessage(std::string message)
{
	m_document.Parse<0>(message.c_str());
	auto it1 = m_document.MemberonBegin();
	// Hold a reference to the "data" field which may be a number of a string
	 // (depending on the "type" field)
	auto & data = it1->value;
	it1++;
	// Get the "type" field which is always a string
	m_type = it1->value.GetString();
	if (m_type == "gameState")
	{
		// get a number: 0 = WAITING_FOR_PLAYERS, 1 = STARTING_GAME
		int num = data.GetUint();
		m_data = std::to_string(num);
	}
	else if (m_type == "error")
	{
		m_data = data.GetString();
	}
	else if (m_type == "updateState")
	{
		// Retrieve the touch position as two separate x,y values
		double x = data["x"].GetDouble();
		double y = data["y"].GetDouble();
		m_data = "x: " + std::to_string(x) + ", y: " + std::to_string(y);
	}
}
```

Check the solution works by testing with one Android client and one HTML5 client.