Scripting Islands of Battle, Workshop 2

Refresh

In the last workshop we covered making API requests, using the API documentation to figure out how to make requests and performing logical operators (e.g `if`) on those statements to code different behaviour based on results.

You should now have a script that tries to login and registers an account if it fails to do so. If you didn't save your work from the last one, an example is available here: IoB-Examples/task3.lua at master · Pebsie/IoB-Examples · GitHub

In this workshop we'll be covering Castle placement and world interaction.

Castle placement

The first thing you do when you do when you start playing on an Island is place your Castle. For now this'll be placed in a totally random location regardless of tactical vantage.

There are two things we need to do, once logged in:

1. Find out whether or not we already have a castle
2. Attempt to place our castle in a random location

We can get the number of buildings that are in the world with a `get` command with the parameters `scope=player`, `type=buildingsum` and `authcode`. Add this to the bottom of your script:

```
result = api.get("?a=get&scope=player&type=buildingsum&authcode="..authcode)
```

Now that we have the number of buildings we need to check if it is equal to 0, and then randomly place our castle. To build things use the `build` command with parameters `position` (corresponding to the tile id within the 100*100 world grid), `type` (the building name, in this instance Castle) and `authcode`.

The API request is `api.get("?a=build&position="..love.math.random(1,10000).."&type=Castle&authcode="..authcode")` (action is `build`, `position` is the position for the new building (see below on how world positioning works), `type` is the building type)

We're using the function `love.math.random` here, which takes two arguments: the minimum number and the maximum number. Usually in Lua this command is just `math.random` but as Islands of Battle is built using the Love2D framework we can use `love.math.random` for better randomness. We don't need to do anything other than this as the API will handle whether the random tile we get is valid to build a castle on and the script will be re-run by the client until our Castle is built.

Task 4: add in the logic to check whether or not we have 0 buildings and, if so, attempt to place the castle. (The solution can be found here: IoB-Examples/task4.lua at master · Pebsie/IoB-Examples · GitHub)

Accessing the world

Now that we're registered, logged in and have our castle placed we need to start interacting with the world.

Islands of Battle works on a 100x100 grid. We don't apply an X or Y value to each tile, instead the `id` field of each tile represents where it is in the world, indexed at 1 (because that's how Lua works!) As another example, the top left position (x=1,y=1) is `id=1` and the top right (x=1,y=100) is `id=101`. x=23,y=50 would be `id=5023`.

> This might be confusing, but it's very unlikely that you'll ever need to hard plug co-ordinates into your script. It's useful to understand, however.

The right tile of any tile is -1 and the left tile of any tile is +1. The tile above another tile is -100 and the tile below is +100.

> In theory, the world wraps around itself, but the borders imposed by the edges of the island will prevent your units from ever travelling beyond the island.

Here we'll introduce the concept of `for` loops. The idea of a for loop is that it runs a single piece of code multiple times at a limit you set, with a value `i` changing each time the code is run. This is particularly useful for accessing every tile in the world.

```
for i = 1, 10000 do
  print(i)
end
```

This would write every number from 1 to 10,000 in the console. Since there are 10,000 tiles in the world, we can get world data and then perform actions using this method. The API request for getting world data is `?a=get&scope=world`. Based on all that you've learned so far, what do you think the code below does?

```
world = api.get("?a=get&scope=world")
for i = 1, 10000 do
  print("Tile #"..i.." is "..world[i].buildingType)
end
```

We can now match this with the `build` function For example, if we wanted to try and build a House on every tile the code would be…

```
world = api.get("?a=get&scope=world")
for i = 1, 10000 do
  api.get("?a=build&position="..i.."&type=House&authcode="..authcode)
end
```

Getting information about ourselves

Moving forward we'll need to know a few things about the account we're logged in as. You can do this with the API call `?a=get&scope=player&type=data&authcode=*authcode*`. It's a good idea to get this information as soon as we login and apply it to a variable. For example, a complete script:

```
username = "demo"
password = "demo"
authcode = api.get("?a=login&username="..username.."&password="..password)
pl = api.get("?a=get&scope=player&type=data&authcode="..authcode)
```

We can now access the follow data:
`username`, `gold`, `wood`, `stone`, `pop` and `food`. We can get this data through `pl`, e.g

`print(pl.gold)` would print the amount of gold the player has to the console.

Task 5: modify the House building script to check whether or not we own the tile we're trying to build on before trying to build on it.

> The API will automatically reject a request to build something on a tile that you don't own, but Task 5 is a much more efficient way of doing things. You're limited to 100 API calls a second, so efficiency is important. Ultimately it's the person with the more efficient script that will be victorious in battle.