Chess in Lua: Final Report

Cameron Wright, Spencer Saunders, Madeline Ross Department of Computer Science,

April 11, 2018

1 Introduction

Lua is a powerful and light language developed in 1993 as a scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.(source: https://www.lua.org/about.html) Being chosen as a language for this project had to do with Lua being a language syntactically similar to Python and on the cover it says that Lua is object-oriented. The documentation was decent but after building Chess using the language it could have been better.

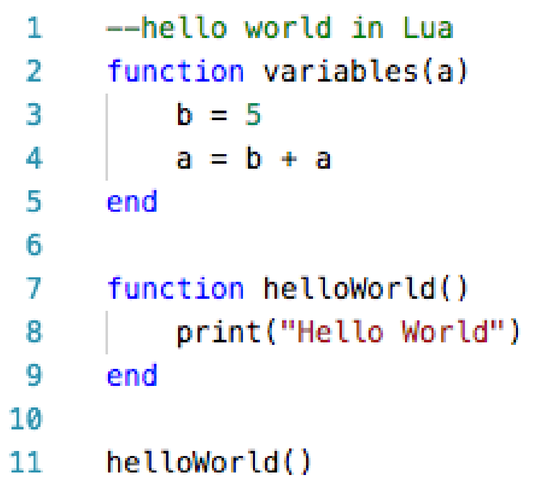
2 Lua's Syntax

2.1 Basic Syntax

As mentioned in why Lua was chosen as a language for this project, it looks similar to Python with some differences. In Figure 1 is shown what a simple script would look like, included is functions, variables, and a print statement. The most similar aspect to Python is where a new line signifies a new step in the program, unlike Java where a ; signifies the same thing. The functions are started with keyword 'function' and stop with 'end', this is different from Python which uses the keyword 'def' and knowns when to stop based on tabbing. Lua does not use tabbing to signify the scope of functions, loops, or conditions, it uses the 'end' keyword. The last noticeable similarity from 1 is with defining variable types, variable types are strong and dynamically typed. No type is needed to declare a variable and after creation it can change between types. Two different types of objects can not be added together like a String and an int, and if tried the Interpreter will throw and error showing that this language is strongly typed not weak.

2.2 Defining Class Structure

One of the reasons Lua was chosen was the mention of being able to Handle object-oriented design principles. Chess is a game where pieces behave like objects on a board and using object-oriented design patters would prove useful. Unfortunately Lua's use of classes is much more difficult than it first seems. Figure 2 shows the creation of a class, and the

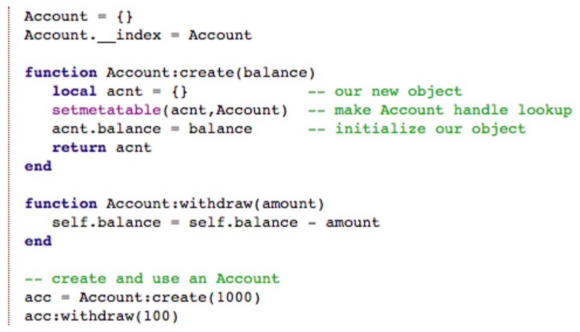


```lua
--hello world in Lua
function variables(a)
    b = 5
    a = b + a
end

function helloWorld()
    print("Hello World")
end

helloWorld()
```

Figure 1: Use of white space and variable deceleration in Lua.

amount of extra code needed compared to other languages. Lua doesn't really have classes, and while to the programmer it is a class to Lua it is an object stored in a metatable. Back in figure 2 the first line of code 'Account = ' is the creation of the metatable. Lua's class structure suffers from some major drawbacks from this, and using classes in Lua starts to show some downsides.

The first problem with classes has to do with how they are declared. The example shown in figure 2 is not the only way to create a class. The constructor can be set up multiple ways and even have different names, and adds confusion when looking up how to set up classes on the Internet. The multiple ways to write classes and just lots of boilerplate code starts showing the annoyances of classes in Lua, and there are multiple times throughout this project where a syntactical error was added or mis coding up the class. Lastly is the subject of Inheritance, and how it makes life easier when needing it for generics. Now Lua at least has inheritance and is something needed with a game like chess where all pieces can relate to a master piece class. Lua like Python can use classes though Python makes making classes easier, both though are truly scripting languages and both don't support abstract classes. Not a huge problem but shows the limitations of using classes in a language that is used for scripting.



```
Account = {}
Account.__index = Account

function Account:create(balance)
   local acnt = {}              -- our new object
   setmetatable(acnt,Account)  -- make Account handle lookup
   acnt.balance = balance      -- initialize our object
   return acnt
end

function Account:withdraw(amount)
   self.balance = self.balance - amount
end

-- create and use an Account
acc = Account:create(1000)
acc:withdraw(100)
```

Figure 2: Defining a class in Lua.

3 Naming, Binding, Scoping, Types

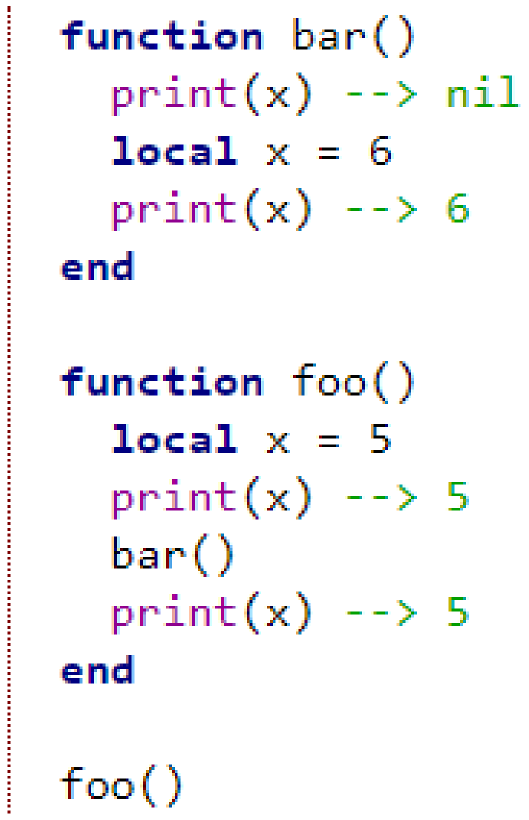
In Lua, the keyword local is almost always used to define variables and function names as it's good practice. Variables that are defined as local exist only in the block they're created in. Lua is a dynamically typed language and doesn't contain any type definitions. Thus, every value carries its own type. In total, there are eight basic types in Lua which are boolean, number, function, nil, userdata, thread, type, and string.

4 Why we chose to go with Lua

One of the reasons Lua was chosen for the group's final project is because it's syntactically similar to Python. Some examples of syntactic similarities both languages share include statements, loops, and functions all requiring indented spacing, associate indexing with arrays/tables, and dynamic variables. Another reason Lua was chosen is because the developers touted the language as object-oriented.

5 Pros of Lua

One pro to Lua is the decent documentation. There's a fair number of Wikipedia pages, books, and other various documents put out by the developers and small, but engaged community. It'd be nice to see additional notes and guides from the developers as the documentation was sufficient but a little barren. Another pro for Lua is that the language



```
function bar()
  print(x) --> nil
  local x = 6
  print(x) --> 6
end

function foo()
  local x = 5
  print(x) --> 5
  bar()
  print(x) --> 5
end

foo()
```

Figure 3: Figure 3: Scope of a local variable in Lua.

has a simplistic, clean syntax that is accessible for beginners and easy to learn. A third pro to Lua is the integrated interpreter that simply requires typing lua in the command line making it convenient to test scripts quickly. Additionally, the primary data structure in Lua, tables, are versatile and convenient because they're associate and can store values of any type aside from nil.

6 Cons of Lua

One of the major cons of Lua is the lack of built-in language techniques like object-oriented programming without importing a third-party library. A second major problem with Lua is its lack of error handling. Error handling support in Lua includes, but is limited to, calling functions pcall and xpcall. A third problem with Lua is how global scoping is the default of variables and functions. For instance, if you make a variable called next and then created a function called next in a different module without localizing either, the function will be overwritten. Finally, as mentioned previously, classes in Lua are a hassle as Lua doesn't have the concept of classes and instead has each object define its behavior and shape.

6.1 Tables, Lua's solution to Lists and Arrays

In Lua, a data structure type called table represents ordinary arrays, queues, sets, and other data structures in a simplified and efficient way. The table type is implemented and modeled off of associate arrays which allows for indexing with any valid value of Lua including numeric values and strings. All together Lua is once again like Python where having lists and arrays combine into an amalgamation of the two. Unlike many other languages that begin indexing at 0, Lua stores its first item at the indexed position of 1. An interesting aspect of tables in Lua is that you can access an indexed position using brackets or a dot. Understanding the difference between bracket and dot indexing is crucial as someone new to Lua may mistakenly use the dot to try and call a function. In our Lua Chess implementation, tables played a viable role for functionality as the tables stored the board, all the pieces, all of the valid moves, and other various features.

7 A Game Called Chess

Chess is played between two people taking turns moving pieces until someone gets into a position where the king would be checkmated. Transferring this game into the realm of computers can be easy due to the rules and pieces of the game being few. For this version of Chess, there still needs to be two players (or you play against yourself) and both will take turns inputing commands. A scanner is used to give a player multiple options about what the want to do, ranging from printing the board, showing moves for a piece, or moving a piece. When a player puts the other into checkmate the game is finished and either they play again or never again.

7.1 Using Lua to Create Chess

For the group's Lua implementation of Chess, a Board object is created in the ChessGame class. The Board object calls the function initialize which creates a 2D table, reserves positions for all the pieces, and occupies the reserved positions with piece objects. All of the individual pieces are created using OOP and inheritance from the parent class Piece. The Board object is printed out to the console and the user is prompted to enter a specific character based off of the printed options in order to advance the game. The player can choose to print the board, check if a move is valid, check available moves, make a move, or get the pieces on the board. The game will keep alternating between players until the function isKingCheckMated equates to true.

8 Conclusion

Choosing Lua to implement Chess turned out to be a harder-than-expected effort given the misleading marketing that claimed Lua supported full-fledged OOP. Without this support, the third-party library middleclass (OOP library) was the backbone of the Chess implementation. Lua certainly has its niche appeal as a scripting language for popular video games like Roblox and Gary's Mod. However, the group doesn't see themselves using Lua again in the foreseeable future. However, this project was still a valuable and rewarding experience that exposed the group to a lesser-known, under-the-radar language. Throughout the bulk of the project, the various concepts learned in CS354 including scope, types, and semantics helped became apparent and it helped the group understand the core concepts of Lua.