J. Anthony Timberlake, Jonathan Luetze
CS 401 Programming Languages
April 1st, 2018

Assignment 4 : Erlang

Work Statement

Jonathan finished the entirety of part one (solving the monte carlo implementation problem). Then, with the help of the TA in the lab, we worked on setting up the rest of the implementation for the distributed game of tictactoe together, using local IP address configurations to test our project's functionality.

Design

For part 1, the only pre-exported method was montecarlo, which takes in the N iterations over X actors as mentioned in the assignment. This method calls the estimate method to sum all of the calculations based on random points generated (as many times as necessary using recursion) and then divides the sum by N (iterations) to calculate the single mean value of all pi calculations. The hits method calculates two random numbers and determines which fall inside or outside of the curve. After montecarlo has calculated the average, it is then sent to writeNum, where it is printed for the user to see.

For part 2, several methods are pre-exported and each depend on each other in different ways. First, using the command line, one must connect to the network using the command in the form of "erl -name examplename@[IP Address] -setcookie [same as opponent's cookie]" in order to create a node that connects to the locally created server. Then, the host of the game will call t3:newgame(), which will register the player node on the server and then wait for an opponent to connect. The opponent will then connect to the existing game by using the same command prompt command followed by t3:playwith(examplename@[IP Address]) where the example name and IP address belong to the host. The host will always be x and the opponent will always be o in the game of tictactoe that is set up this way. The waiting for the opponent and the connecting to the host are handled in wait_opponent and connect_opponent respectively.

After the initial connection is set up, both members of the game have options as to what they can do. One of the players is decided at random to go first (50% chance). At that point, either player can attempt to use t3:placetoken() with an argument a1 … c3 to place an x or o in the corresponding place on the tictactoe board, but this will only work for whoever's turn it is. After one movement, the turn is shifted to the other player. Additionally, at any time in the game, players can use t3:tell() to send messages to one another instantly, regardless of whose turn it is at the time.

At the end of every turn, the board is sent to check() in order to see if a winning case has been reached, using pattern matching. For each player, there are a total of 8 winning combinations. Otherwise, the game goes until the board is filled, in which case the check returns draw to show that the game resulted in neither player winning. Whenever one player wins, he or she receives a message saying they are victorious and the other player simultaneously receives a messages stating that he or she has lost. After the completion of the game, the program will halt. A new connection will need to be established before playing again in the same manner.

Additional functions include printElement and printBoard, which together are used to print the tictactoe board in a more readable format. Also, checkCoord converts a given Coord in the form of a1 … c3 into a corresponding position in the Board tuple which contains positions 1 through 9, or -1 if the input is invalid. Play() checks if each position selected is valid or not and create_empty_board() initializes the undefined, empty board. Most of the game functions through wait_msg(), using pattern matching for differing cases of messages sent between each user.

Difficulties

The main difficulties of using this language were familiarizing ourselves with the concepts of parellelization and network functions with the use of Erlang's spawn function and registering connections. The TA was of great help in this case. Otherwise, the syntax and functionality was very similar to that of prolog. Also, given the same final version of the code, there were differing outputs between Mac/Linux and windows at times. This issue is still not solved. So, for proper output, this code was tested and shown to work between two Mac OSX computers.

Pros of this Language

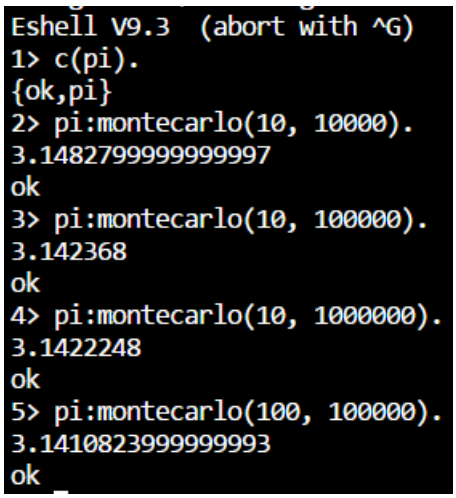
Erlang is used for functional programming in areas of distributed systems and parallel processes. It is also known for its use of immutable data and pattern matching. Applications made using Erlang are often distributed and highly available, never needing to be stopped in order for code to be changed and recompiled. This amazing functionality allows companies worldwide to make steady programs which never have to be down for maintenance time.

Cons of this Language

Having had experience primarily in imperative programming languages, the "features" of data-immutability instantly becomes problematic as updating variables gave us problems at every turn. While Erlang has its many uses and is accessible all over the world, it is much less intuitive or readable compared to other newer languages like Python which can be used to handle distributed systems as well. They both have their pros and cons, but Erlang was indeed built a longer time ago to solve more specific purposes, and more general purpose languages have developed in the years since. The syntax is difficult to pick up for new programmers needing to be trained. The amount of resources for help and community are nearly nonexistant beyond niche circles. The language itself, like prolog, relies completely on recursive calls, so there is no option for standard loops. Strings are almost impossible to deal with as well, similar to prolog. In the end, apart from particular purposes that the language was designed for, there are just too many negative aspects to this language as a general purpose language.

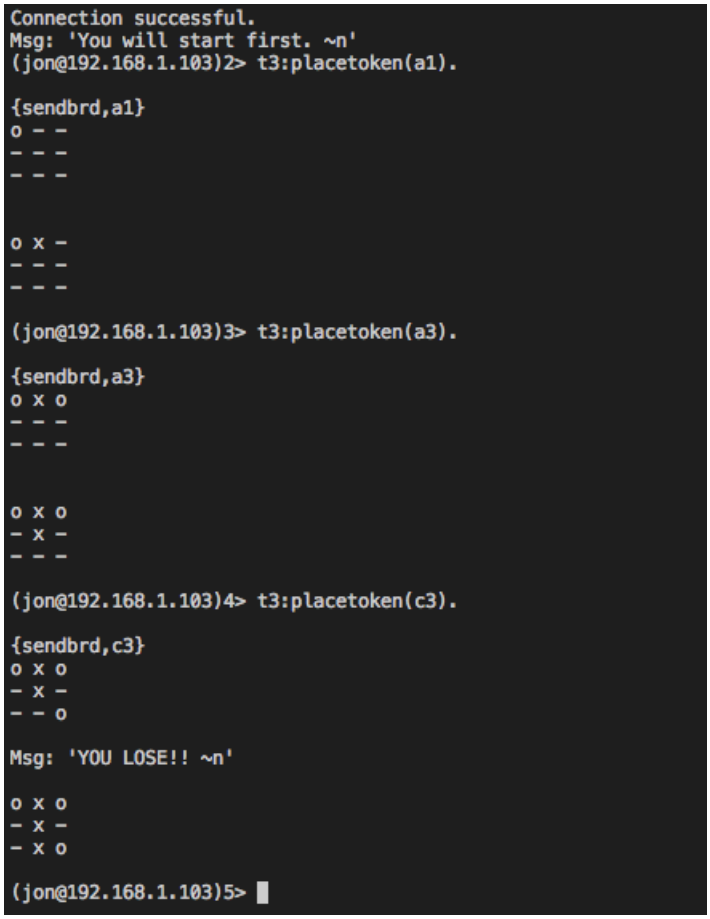
Test Cases

Part 1 :

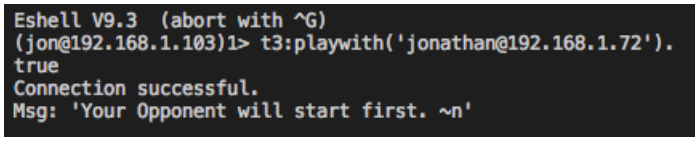


Part 2 :

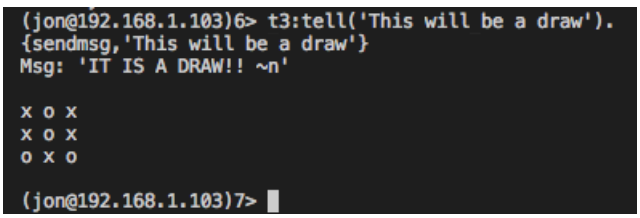
Host : start of game followed by moves and loss



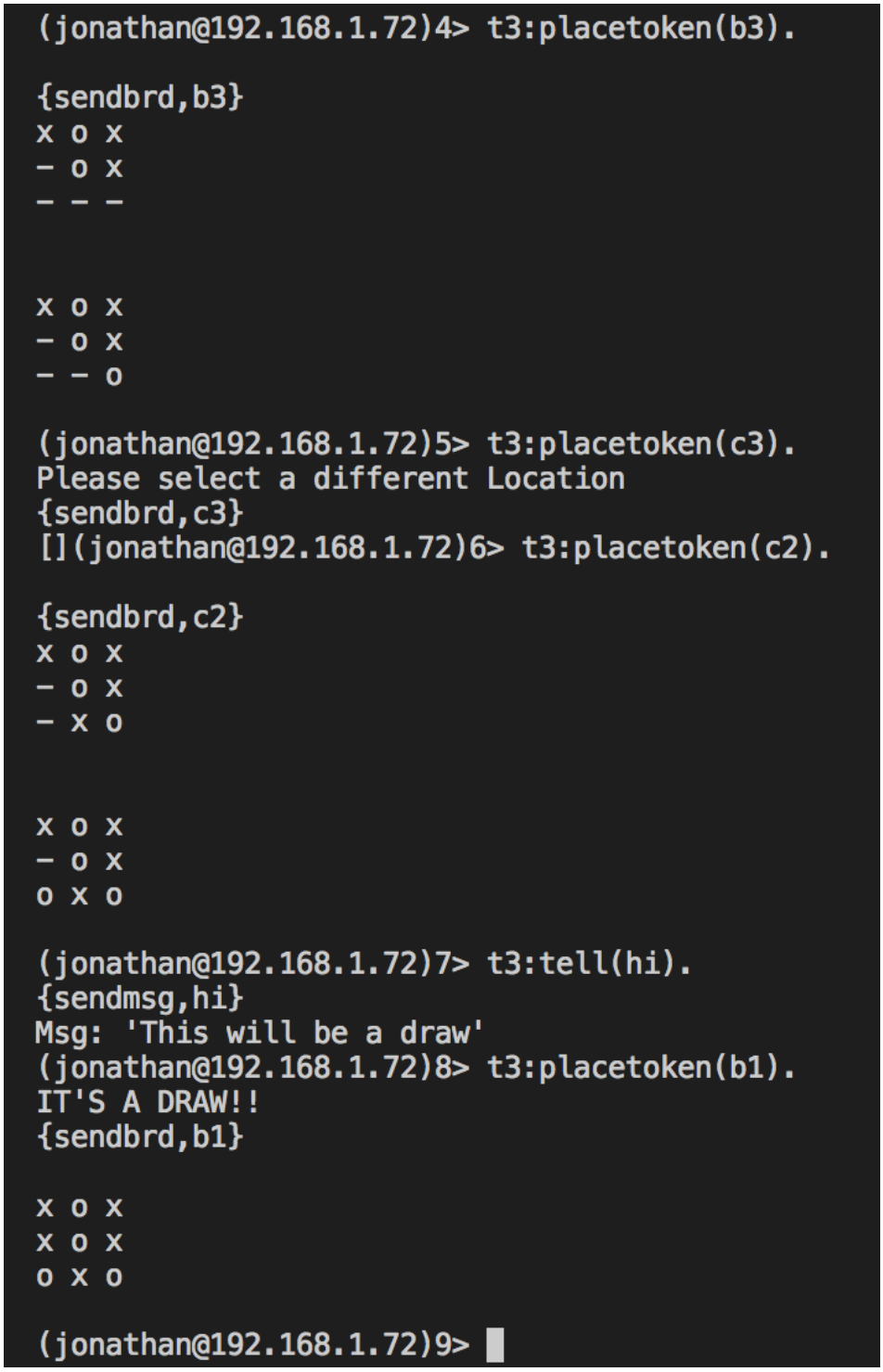
Receiver : accepting connection to host



Receiver : sending a message followed by output of board ending in draw



Host : attempting and failing to overwrite moves on occupied spaces followed by draw



Host : attempt to make a move when it is not his turn; moves followed by victory

```
(jonathan@192.168.1.72)3> t3:placetoken(a1).
It's not your turn!
{sendbrd,a1}

o - -
- - -
- - -

(jonathan@192.168.1.72)4> t3:placetoken(a2).

{sendbrd,a2}
o x -
- - -
- - -



o x o
- - -
- - -

(jonathan@192.168.1.72)5> t3:placetoken(b2).

{sendbrd,b2}
o x o
- x -
- - -



o x o
- x -
- - o

(jonathan@192.168.1.72)6> t3:placetoken(c2).
YOU ARE VICTORIOUS!!
{sendbrd,c2}

o x o
- x -
- x o
```