Building a Multiplayer Game of Battleships Using Python

Kourosh R. A. Simpkins

University of the Arts London

Table of Contents

[Abstract 3](#_Toc87948596)

[Building a Multiplayer Game of Battleships Using Python 4](#_Toc87948597)

[What is Battleship? 4](#_Toc87948598)

[Steps Required and the Order They Will Be Completed 5](#_Toc87948599)

[Basic game logic 6](#_Toc87948600)

[[Heading 3] 6](#_Toc87948601)

[References 7](#_Toc87948602)

Abstract

Over the course of the next few months, I aim to build a version of the classic boardgame battleship (or battleships, I refer to it as either battleship or battleships throughout this write up). This piece of writing aims to cover the thought process from the planning of logic and networking to potentials for deployment as well as some pseudocode.

Keywords: Python, Networking, Microsoft Azure, Battleships, Online Gaming

Building a Multiplayer Game of Battleships Using Python

The game of battleships feels like a very well-known game. Most people will have played it or heard of it at some point in their life. The aim of this project is to expand my knowledge of networking and communicating with a server via Python programming and (relatively) basic networking fundamentals. To start with, I aim to lay out a set of clear goals, and the orders in which I hope to achieve them. I also aim to explain certain technologies, as well as give a simple explanation as to what battleships is, and the rules it must follow. This paper may assume prior knowledge of certain pythonic functions, however anyone with a basic grasp of programming and computer science should have no problem understanding what I am talking about and should be able to follow along with relative ease.

# What is Battleship?

Battleship was originally a simple pen and paper-based game played between two players. It is thought to have origins in the French game L’Attaque, played during World War 1, however it is also said to have been played before the first world war by Russian Soldiers, so it is difficult to trace its exact historical origin (Hinebaugh, 2009). In 1967, Milton Bradley introduced a version of the game that was played on a plastic pegboard, using small plastic pegs and ships, which is the version that most people will have played in the modern day. Battleship was also one of the earliest games that was turned into a computer game. During 1976 a version of Battleship was released for the Z80 Compucolor, which was a series of microcomputers, the first with built in colour graphics! There have been several editions of the game since.

A picture containing icon

Description automatically generated

# Steps Required and the Order They Will Be Completed

For the sake of abstraction, I will break down the problem here, and potentially break down each sub-problem into smaller problems, that can be solved on a smaller scale with OOP. To begin with I need to figure out the basic programmed logic of the game, in other words the basic ruleset and how I can get Python to handle it. Once I have completed the logic of the game to the point that two players can play on the same machine, I should get started working on a back-end server program and translating the logic from the single program interface. Once I have a piece of working server code, I can design the code for the client following the same ruleset, and such that a client can communicate with a server properly. At this point, in theory the program should be able to run on two separate machines, if one machine has access to the server-side code, and both have the client code downloaded (it should also be able to run with one machine as a server and two machines as clients). Once this works, I would like to set about creating an instance on Azure, so that two players can play from anywhere in the world so long as they have the client program available to them, although creating an Azure instance is technically outside of the scope of this module, so I won’t bore you with the details.

## Basic game logic

The game must be able to handle two players, and each player must have 2 grids (a targeting grid and a grid of their own ships), so as such I will likely need four 2-diemsional arrays; Player1\_Ships, Player2\_Ships, Player1\_Targeting, Player2\_Targeting. The ships 2D arrays will store ‘0’ in a position where no ship is present, and ‘1’ in a position where there is a ship present. The targeting array should hold data about where the current player has fired, and whether a location they have fired on has been a hit or a miss (with each indicated by a h or an m respectively, and an ‘0’ if the location has not been fired upon). I have also considered using a dictionary to represent a ship, or possibly a brand new struct, however now the most logical approach (to me) is to have the data of the ships stored in the 2D arrays, although this isn’t set in stone. Printing the grid should be possible using a pair of for loops (i and j):

for i in range(x-axis of array):

for j in range(y-axis of array):

print([i][j])

endfor

print(“\n”)

endfor

The logic can then use strcmp within an if statement to compare the value of array[i][j] against the given characters above. If it is a value then the program should be able to handle it in the correct way (hit, miss, depending on turn/array run off etc.). There are some other thoughts I need to finalise [finish those off here as working].

## Networking objectives

Networking two or even three machines together, as long as they’re on the same network, is not necessarily that difficult. All you really need is the ip address, and potentially to expose a port on each machine. Over multiple networks, you need to expose ports in multiple locations (routers, switches, on the machines themselves etc.), so it adds a level of complexity that I have not worked with for a while. The easiest way to create a networked game would be for me to have a server-side program that runs game logic and a client-side program that searches for the server and takes the outputs of the broadcast from the server and serves it to the player. To perform client-server handshakes and communications I will probably be using the socket module (a low level networking interface built into the Python standard library) or potentially SSL (a TLS/SSL wrapper for socket objects [TLS = Transport Layer Security, SSL = Secure Sockets Layer]).

References

Hinebaugh, J., 2009. *A board game education*. Lanham, Md.: Rowman & Littlefield Education.