GAME 3110

Assignment 3

Matchmaking Algorithm

Ricardo Shimoda Nakasako 101128885

Git Repository: https://github.com/ricardoshimoda/gbc_game3110.git

Objective

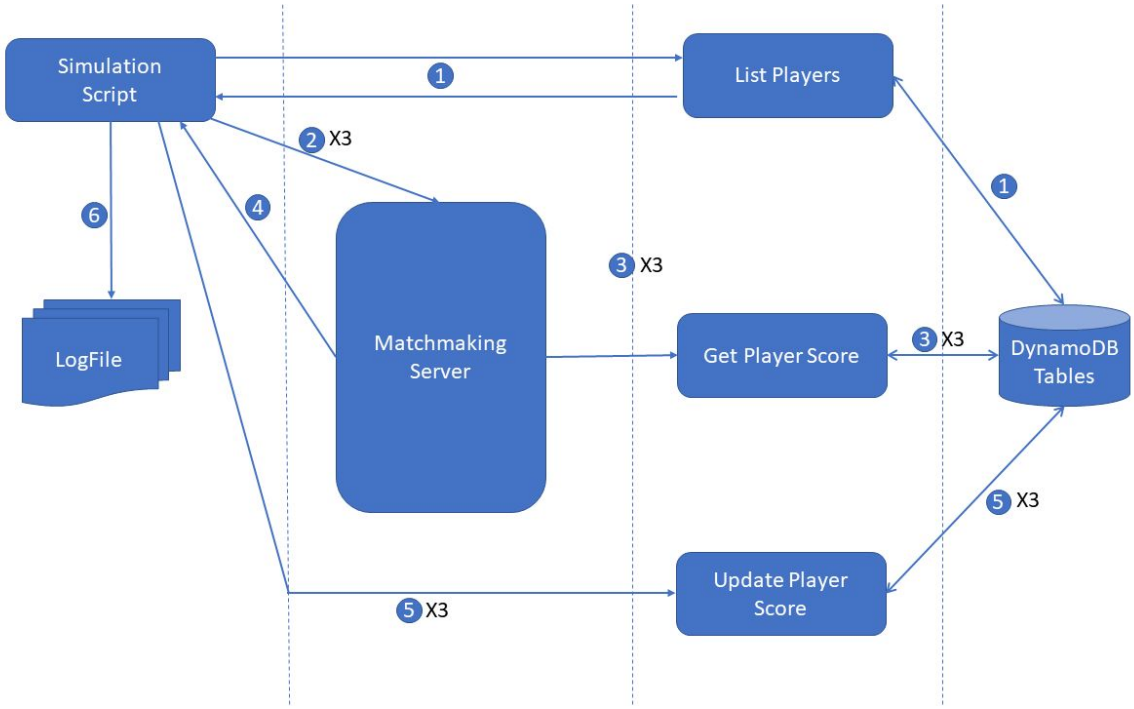
The goal behind this architecture is to build a matchmaking algorithm for a 3-way battleship game - where participants have to sink each other's boats and, in the end, only one player is considered the winner.

The score for this game is calculated based on the amount of "sunk boat parts" that you could achieve plus the amount of boat parts one has at the end of the game (0 if you lost) - regardless of whether you were able to sunk an entire ship or not. Skill is, therefore, measured in terms of the average pieces of boat sunk/defended per game/rounds * 100.

It is possible to create balance between players of different skill levels by giving the player with lower skill level more boat pieces, giving him more rounds to try to find and sink other player's boats - using a certain notion of "Handicap", which is calculated as matching happens and before a game starts.

Architecture

For every game simulation we have the following steps:



1. Three players are selected randomly from a table of existing players in the database.
2. One by one, after a random amount of time, connects to the Matchmaking Server and enters a matchmaking queue.
3. One by one, the matchmaking server requests player details.
4. When it gets to three players in the queue, it calculates the relative handicap (ie.: the final number of ship pieces that each player has access to), creates a Game ID and sends that to the players/simulation script
5. The match results are generated by the simulation script, considering a random variable and the handicap of each player - this is then sent to the Score update server - which updates the average score for each player
6. Finally, we repeat 1-5 until the required number of games is achieved and we dump the log file with all the required information

Algorithm

ELO ratings works by comparing two opponents who use the same pieces, same rules to obtain a single result (win / lose), and, therefore, is centered wholly in distributing points based upon the binary outcome of a game and the level of whoever was playing. Its calculation also incorporates the existence of different "levels" of skill - by manipulating a constant.

Formula

Battleship x3, on the other hand, has a score at the end of the game, consisting of how many pieces you were able to destroy, how many pieces you were able to keep (if you're the winner) and in how many rounds you were able to do so (all rounds if you are the winner or second place, your final round if you're third place) - thus, the final score has to be a part of calculating how skilled a player really is.

In the database we'll keep the following numbers:

1. Number of pieces destroyed (Pd)
2. Number of pieces kept (Pk)
3. Number of misses (M)
4. Number of rounds played (R)

The ranking of each player will be, thus calculated

$$RNK = (Pd + 1.5 * Pk - 0.8*M)* 100 / R + 100$$

Where 100 should be the "initial ranking" for a player. If a player miss more than hits the opponent's ship, he/she will receive a ranking lower than 100. If a player hits more than misses but loses the game, he/she might still be at the start of his/her ranking. If a player wins a game, he/she goes up in the rankings.

Fairness

In order to get players from different rankings and still make it "fair", we introduce a Handicap score, which is translated in extra pieces on the board for the weaker players. Since we have a 10x10 board (100 spaces), the limit is 60 pieces for the weakest player and 40 pieces for the strongest one and the middle one will have a directly proportional number of pieces (according to his/her ranking).

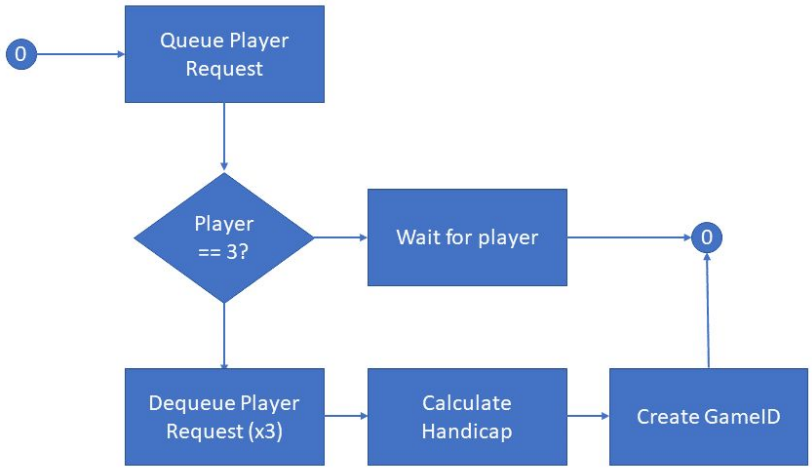
Even though missing is more probable for the weakest player than actually hitting something,I believe that the weights (1.5 for piece kept and -0.8 for misses) will balance it out.and this

arrangement makes it possible for the weakest player to have more rounds to try and win the game - as well as it keeps the good players motivated to hit "harder" and improve their own status.

Finally, as it happens with any kind of online game, if one or more players get disconnected during the game, it is considered (internally) as a No Contest and the other two players "win" but do not get their score updated in the database.

Matchmaking

Given the handicap algorithm, we will start by matching any 3 players that come into the game server requesting a game and calculate the handicap, create the boards and then wait for the next three players to request a game - it's that easy.



Lambda Functions

ListPlayers

This function simply gets the data from all the players after a GET function.

URL: https://zfum6anmbl.execute-api.us-east-1.amazonaws.com/default/ListPlayers

Result expected:


```
{
    "Items": [
        {
            "password": "password",
            "user_id": "player01",
            "session_tk": "-",
            "email": "player01@test.com",
            "login_date": "-"
        },(...),
        {
            "password": "password",
            "user_id": "player11",
            "session_tk": "-",
            "email": "player11@test.com",
            "login_date": "-"
        }
    ],
    "Count": 11,
    "ScannedCount": 11,
    "ResponseMetadata": {
        "RequestId": "K2J15HIO26GNAAGLD5TT95IN9FVV4KQNSO5AEMVJF66Q9ASUAAJG",
        "HTTPStatusCode": 200,
        "HTTPHeaders": {
            "server": "Server",
            "date": "Mon, 06 Apr 2020 17:18:22 GMT",
            "content-type": "application/x-amz-json-1.0",
            "content-length": "1547",
            "connection": "keep-alive",
            "x-amzn-requestid": "K2J15HIO26GNAAGLD5TT95IN9FVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "x-amz-crc32": "3799372878"
        },
        "RetryAttempts": 0
    }
}
```

GetPlayerScore

This function simply returns scoring data from the player, given a certain PlayerId (passed through querystring):

1. Number of pieces destroyed (Pd)
2. Number of pieces kept (Pk)
3. Number of misses (M)
4. Number of rounds played (R)

URL: https://zfum6anmbl.execute-api.us-east-1.amazonaws.com/default/GetPlayerScore

Typical Call:

```
https://zfum6anmbl.execute-api.us-east-1.amazonaws.com/default/GetPlayerScore?user_id=player01
```

Result Expected:

```
{
    "user_id": "player01",
    "destroyed": 129.0,
    "kept": 43.0,
    "rounds": 199.0,
    "misses": 70.0
}
```

UpdatePlayerScore

This function receives the player id and the counters for pieces destroyed, kept, misses and rounds played for each of the players and updates the database. All data is passed through querystring.

URL: https://zfum6anmbl.execute-api.us-east-1.amazonaws.com/default/UpdatePlayerScore

Typical Call:

```
https://zfum6anmbl.execute-api.us-east-1.amazonaws.com/default/UpdatePlayerScore?user_id=player11&destroye
d=64&kept=5&misses=412&rounds=476
```

Result Expected:

```
{
    "result": "Update successful"
}
```

Database

Players table

The fields for this table are:

- user_id: primary key - string
- email: user's e-mail
- password: user's password
- login_date: the date and time (in format yyyymmddHHMMSS) in which the user entered the queue (default value "-" if the user is not in the queue)
- session_tk: the game id (default value "-" if the user is not playing a game)

BattleScore

The fields for this table are:

- user_id: primary key - string
- destroyed: pieces destroyed in player's total career
- kept: pieces kept at the end of the game in player's total career
- misses: number of times the player has missed
- rounds: number of rounds the player has played

POSTMAN LINK:

https://www.getpostman.com/collections/9a242d0d904990d2d62f