TCP socket programming(server)

Introduction

This assignment introduces you to writing simple TCP servers using the socket interface of the operating system. We will be working with the socket interface provided by the Linux OS. Be sure to read the submission guidelines for the assignment and remember the rules applicable to all assignments.

Question 1: Writing a TCP server

In this assignment, you will implement a multi-forking server that can handle at most 3 clients at a time. A client will connect to the server and provide the game information of a snakes and ladders game. Using this game information, the server will respond with the following information:

- **Game state**: Whether the game has finished(i.e. has anyone reached the final square) or it is still in progress.
- **Winner of game**: The first player to reach the final square.
- **Final positions**: The final position of every player on the game board.
- **Squares traversed**: The sequence of squares traversed by each player.

Description of snakes and ladder game

A snakes and ladders game is played on a square board of dimension N. Every square is assigned a unique value between 1 and N^2(both inclusive) in sequence. The first square of the board is assigned the value 1 and all succeeding squares are assigned values in increasing order. The board also consists consists of several snakes and ladders. The objective of the game is to reach the last square as fast as possible, starting from the first square on the board. Ladders enable a player to move closer to the final destination while snakes move the player away from the final destination. Players take turns to roll a die. The value of die determines how many squares the player who rolled the die moves. Players take turns to roll the die until someone reaches the final square.

Game rules

Here are the rules followed by the grader. We recommend that you also obey these rules so that grader will correctly grade your solution.

1. If a player lands on a square that is start of a ladder, player moves up the ladder to the end of ladder. **Both start and end of ladder are included in squares traversed**.
2. If a player lands on a square that is start of a snake, player moves down to end of the snake. **Both start and end of snake are included in squares traversed.**
3. If a player rolls a number but there are not enough squares to move, the player remains on same square.
4. A game is considered finished if at least 1 player has reached the destination square i.e. the square numbered N^2 in an N-dimensional board.

Input format

The input is a JSON string consisting of the following keys:

- **board_dimension**: Dimension of game board.
- **player_count**: Number of players.
- **snakes**: A dictionary with start and end of all snakes on the board as keys and values respectively.
- **ladders**: A dictionary with start and end of all ladders on the board as keys and values respectively.
- **die_tosses**: The value of die when rolled by players in each round of the game.

Below is a sample JSON string sent by the server. This string is displayed across multiple lines for readability: the actual string will be sent in 1 single line terminated by newline character('\n').

```
{
  "player_count": 3,
  "board_dimension": 6,
  "ladders": {"19": 24, "27": 34, "28": 31, "32": 33},
  "snakes":  {"6": 5, "11": 8, "30": 29, "35": 17},
  "die_tosses": {
    "1":  {"1": 5, "2": 1, "3": 5},
    "2":  {"1": 5, "2": 1, "3": 6},
    "3":  {"1": 5, "2": 3, "3": 3},
    "4":  {"1": 4, "2": 1, "3": 5},
    "5":  {"1": 1, "2": 3, "3": 3},
    "6":  {"1": 6, "2": 6, "3": 6},
    "7":  {"1": 3, "2": 6, "3": 3},
    "8":  {"1": 5, "2": 3, "3": 4},
    "9":  {"1": 1, "2": 4, "3": 3},
    "10": {"1": 1, "2": 2}
  }
}
```

In the above game, 3 players played the game on a 6 x 6 board(i.e. 36 squares). There were 4 ladders starting at 19, 27, 28 and 32 respectively and 4 snakes starting at 6, 11, 30 and 35 respectively.

Output format

The clients expect the server to process the input provided and return the information specified earlier. For the input described in previous section, the output is the following JSON string. This string is displayed across multiple lines for readability: the actual JSON string should be sent in 1 single line terminated by newline character('\n').

```
{
  "winner": 2,
  "game_state": "finished",
  "final_positions": {"1": 18,"2": 36,"3": 36},
  "squares_traversed": {
```

```
    "1": [5, 10, 15, 19, 24, 25, 31, 34, 35, 17, 18],
    "2": [1, 2, 5, 6, 5, 8, 14, 20, 23, 27, 34, 36],
    "3": [5, 11, 8, 11, 8, 13, 16, 22, 25, 29, 32, 33]
  }
}
```

In the above input, the game has ended and player 2 finished won. All squares traversed by each player are also included.

How we will test your program

We will pass a port number on which the server has to listen for connections on. The server must accept 3 simultaneous connections else the grader will not grade the server. Using fork() is the easiest way to handle multiple clients, although other ways are also possible. A sample invocation is shown below:

```
$ python2 server.py 8080
```

In above invocation, the server must listen for connections on TCP port 8080. The server must read input from each client and send them the corresponding response.

Clients will send a game state and wait for response from server before sending the next game state. Clients will send '0' after receiving a response to the last game state from the server. The clients will exit after this and the connection will be terminated. The server must exit cleanly after it receives '0' from all clients.

We recommend that during development, it is easier if you handle just 1 client. Once you have a working solution, you can support multiple clients using fork(). Note that fork() may not be the best method to achieve this but it certainly work and is simple enough for learning.