Krea Code Test: Squid Game

Ajith Ukwatta

December 7, 2022

1 Introduction

1.1 Background

This code was done as a part of the code test conducted by Krea for full stack developer position

1.2 Task

Find the board which will win last from the provided board set in a game of **Bingo** with our good Octopus friend "Peter"

1.3 Bingo

Introduction to bingo can be found in the Wikipedia articles given below
Americal Version
$https : //en.wikipedia.org/wiki/Bingo_(American_version)$
British Version
$https : //en.wikipedia.org/wiki/Bingo_(British_version)$

The "Bingo" game addressed in this document does not fully comply with any of the bingo versions given above but is much closer to the American version.

2 Win

2.1 condition

Board will be considered as won if any row or column is fully marked. Diagonal will not be counted.

2.2 Solution

Find the earliest draw which fully marked a row or a column

2.2.1 Procedure

1. Boards are numbered incrementally starting from zero.
2. Draws are stored in an array and draws will be done from zeroth element to the last element incrementally.
3. Find the index of the draws array for each element in the board.
4. Construct a new array with the same dimension as a board but filled with array indexes from the draw array instead of the actual numbers of the board.
5. Separate every column and row in the new board to arrays.
6. Find the maximum of the array which will be equivalent to the draw which filled that particular row or column.
7. Find the minimum of the above maximums. This value will be the draw which fully marked a row or a column in the board. In other words, the earliest draw to win the given board. The minimum above discussed will be referred to as the **"winning draw"** throughout the rest of the document.
8. Construct an array with winning draws. This array will be considered as **"Winning Draws"** throughout the rest of the document. The array should be constructed such a way that the array indexes represent the board number.
9. The minimum of the **"Winning Draws"** will earliest draw which produces a winner and the index of the minimum would be the board number which won.
10. Similarly, the maximum of the **"Winning Draws"** would be the draw which processes the last winner and the index of the maximum would be the board number which won the last.

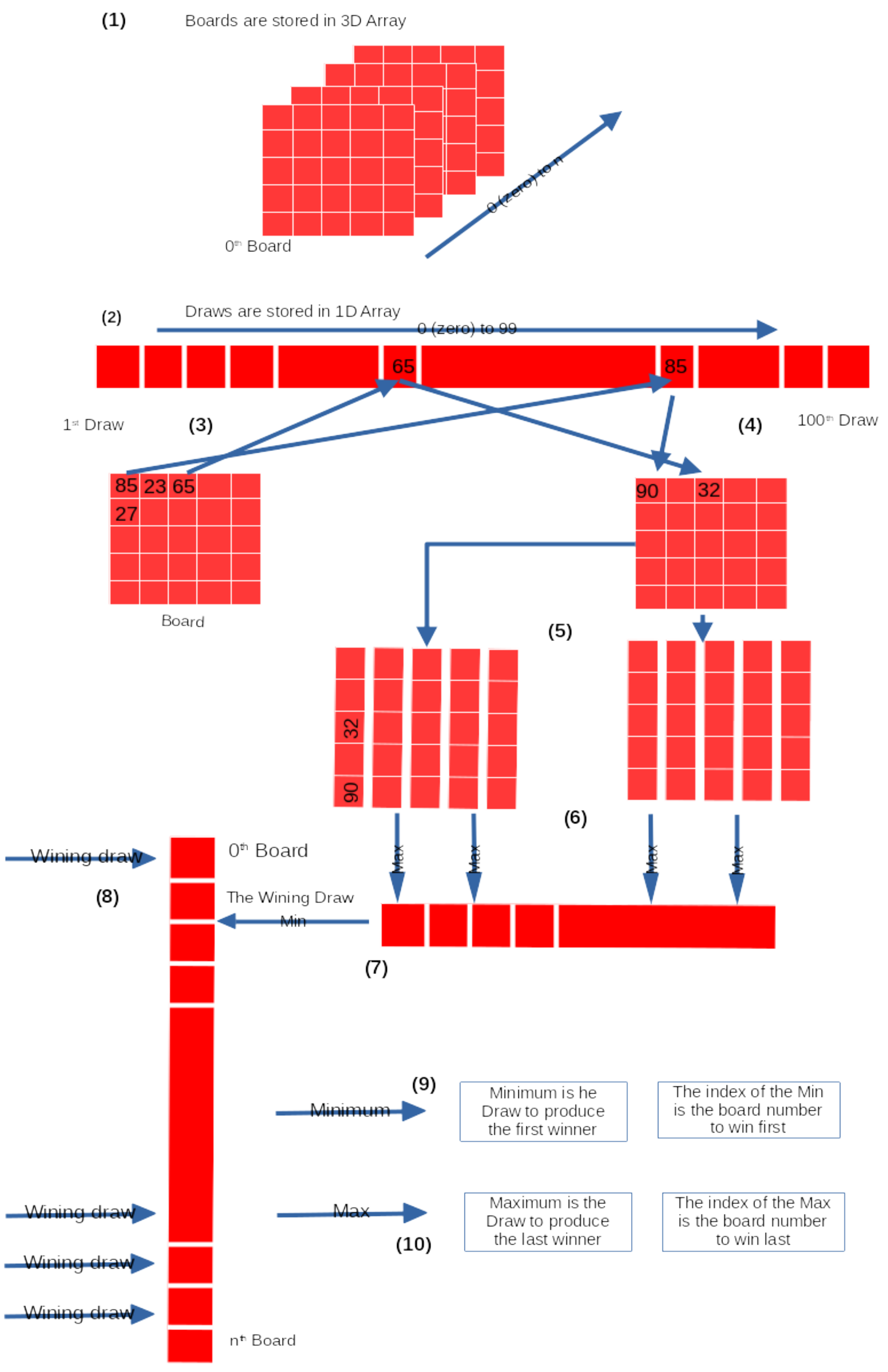

(1)
Boards are stored in 3D Array
0 (zero) to n
0th Board
(2)
Draws are stored in 1D Array
0 (zero) to 99
65
85
1st Draw
(3)
(4)
100th Draw
85 23 65
27
Board
90
32
(5)
32
90
(6)
Max
Max
Max
Max
Wining draw
0th Board
(8)
The Wining Draw
Min
(7)
(9)
Minimum
Minimum is he Draw to produce the first winner
The index of the Min is the board number to win first
Wining draw
Max
Maximum is the Draw to produce the last winner
The index of the Max is the board number to win last
(10)
Wining draw
Wining draw
nth Board

Figure 1: Solution

3 Implementation

3.1 Tools & Dependencies

- Language : Python3
- Dependencies:
  - Numpy - required
  - Matplotlib - optional (To Generate graphs)
  - termcolor - optional (To produce terminal output with color)
  - pillow - optional (To Generate cards as .png)

3.2 Directory Structure

- data
  Contains both test and actual data
  - real
    - boards.txt
    - numbers.txt
  - test
    - boards.txt
    - numbers.txt
- loader.py
  loader.py contains all the methods required to load and sanitize both test and real data
- board.py
  This file defines the Board object which contains all the methods required to search for the winning board, calculate scores and print results
- main.py main.py is the main script which should be executed to run the project. main.py will load data, search for the solution and print the results.

3.3 How to run the code (on Linux)

- Open a terminal
- Navigated to the root director
- run the command $python3main.py$

```
python3 main.py
```

3.4 The Code

3.4.1 loader.py

- imports

Listing 1: imports

```
import numpy as np
```

- Methods
  - draws
    Load draws as an array from a given text file

Listing 2: Loading draws

```
def draws(directory, file):
    """
    :param directory: directory where the file is stored <string>
    :param file:  file name with extension to opened <string>
    :return: 1-D numpy array of integers
    """

    # load file
    with open('data/'+directory+'/'+file, 'r') as f:
        numbers = f.read()

    # split string of numbers to array by comma
    numbers = numbers.split(',')
    # Sanitization
    # Remove any non numeric characters from loaded data
    # This step is not necessary for the provided data set
    # However it is a good practice to sanitize data always when import
    numbers = [x for x in numbers if x.isnumeric()]  # remove non numeric entries
    numbers = np.array([int(x) for x in numbers])  # convert to int
    return numbers
```

  - boards
    Load boards from a given text file as a collection of Board objects

Listing 3: Loading boards

```
def boards(directory, file, board_shape):
    """
    toDo: handle if more than one empty lines were present in the file
    :param directory: directory where the file is stored <string>
    :param file:  file name with extension to opened <string>
    :param board_shape:  number of rows and number cols inboard as tuple
    :return: 3-D numpy array of integers
    """
    with open('data/'+directory+'/'+file) as f:
        collection = f.read()

    rows, cols = board_shape
    # split by new line
    # It was assumed that every single line represent a row of a board
    collection = collection.split("\n")
    # print(collection)
    # detect empty lines
    # It was assumed that boards are separated by a empty line.
    # Therefore content between two empty lines can be considered as a board
    board_separators = [index for index, item in enumerate(collection)
                        if len(item.strip()) == 0]
    # the last line of the file also represents a end of the board
    board_separators.append(len(collection))
    # print(board_separators)
    start = 0
    all_boards = []
    board_id = 0
    for separator in board_separators:
        # sampling data between boar separators
        temp = collection[start:separator]
        full_board = []
        # sanitization
        # filter line by line for non numeric characters, and correct
        # number of rows and columns
        for line in temp:
            row = line.split()
            row = [x for x in row if x.isnumeric()]  # remove non numeric entries
            row = [int(x) for x in row]  # convert to int
            # skip row if does not comply column count
            if len(row) == cols:
                full_board.append(row)
            else:
                print("invalid row :", row)
        # skip board if does not comply row count
        if len(full_board) == rows:
            all_boards.append(Board(np.array(full_board), board_id, board_shape))
            board_id += 1
        else:
            print('Invalid Board. Row Count does not comply : ', len(full_board))

        start = separator+1

    return all_boards
```

3.4.2 board.py

The board object is defined here. All methods necessary to implement the proposed solution and produce the output are implemented here.

Listing 4: board.py

```
import numpy as np
from termcolor import colored  # to produce the colored text in the terminal
from PIL import Image, ImageDraw  # to generate the image of a board


class Board:
    def __init__(self, board, _id, shape):
        self.board = board
        self.id = _id
        self.shape = shape

    def draw_indexes(self, draws):
        """
        All the elements in the board will be mapped to draws array and extract
         the element index of the draws array. Extracted element indexes are filled
        to a array size of the board. This process has been graphically explained in
        the Step 3 and 4 in the figure which explains the solution
        :param draws: all the draws as 1D numpy array
        :return: 2D numpy array
        """

        board_shape = np.shape(self.board)
        # Flatten the board in to a 1D array for easy handling
        board = self.board.reshape(-1)
        # Build an array with indexes of draws which matches the numbers of the board
        # Every cell contains a array indexes and the original number from the board
        indexes = [[np.where(draws == number)[0], number] for number in board]
        output = []
        for index in indexes:
            # Check if numbers in the board exists in the draw
            # If not produces a invalid number message
            if len(index[0]) > 0:
                # It is assumed that all the draws are unique and no duplicates
                # Therefore, there will be only one index for a given board number.
                output.append(index[0][0])
            else:
                # Boards are already shaped to 5 x 5 during load
                # If any number was skipped here it will produces an error during
                # re-shaping and entire program will fail
                # This condition has not been address here.
                print("Invalid␣number␣:", index[1], "␣does␣not␣exist␣in␣the␣draws.␣This␣board␣will␣neve

        # Reshape the indexes array into the shape of the board
        return np.reshape(np.array(output), board_shape)

    def wining_draw(self, draws):
        """
        1. Get the maximum of every row and column
        2. Then get the minimum of those maximums which would be the
        wining draw for the given board
        3. Step 5, 6 and 7 in the schematic given in the solution section
        are done here
        :param draws: 1D array
        :return: Scaler ; the wining draw
        """
        # print(draws)
        rows, cols = self.shape
        # Flatten the board in to a 1D array for easy handling
        board = self.board.reshape(-1)
        # Build an array with indexes of draws which matches the numbers of the board
        # Every cell contains a array indexes and the original number from the board
        indexes = [[np.where(draws == number)[0], number] for number in board]
        output = []
        for index in indexes:
            output.append(index[0][0])

        # Reshape the indexes array into the shape of the board
        indexes = np.reshape(np.array(output), (rows, cols))
        # print(indexes)
        # get the maximum of the rows
        # find the minimum of those maximums
        row_win = min([max(indexes[index, :]) for index in range(rows)])
        # get the maximum of the columns
        # find the minimum of those maximums
        col_win = min([max(indexes[:, index]) for index in range(cols)])
        # get the minimum of above two minimums which represent the wining
        # draw of the board.
        return min([row_win, col_win])

    def diagonals(self):
        """
        this method returns both diagonals of the board.
        This method is not needed for the standard solution provided
        However, This method will be used for the extended solution.
        if assume that we have access to full board configuration.

        :return: return both diagonal of the board
        """
        ax1 = np.diagonal(self.board)
        ax2 = np.diagonal(np.fliplr(self.board))
        return ax1, ax2

    def board_score(self, draws, wining_draw):
        """

        :param draws: Full set of draws ; 1D array
        :param wining_draw: The index of the draws array for the wining
        draw of the board
        :return: Scaler; the score of the board after the wining draw
        """

        # Flatten the board for easy handling
        board = self.board.reshape(-1)
        # Actual value of the wining draws
        last_number = draws[wining_draw]
        # At all the numbers drawn until the wining draw include the wining draw
        # These numbers represent the marked numbers of the bord
        drawn_numbers = draws[:wining_draw+1]
        # Extract all unmarked numbers from the board
        remaining_numbers = board[~np.isin(board, drawn_numbers)]
        # print(remaining_numbers)
        # Add all the unmarked numbers together and multiply by the last draw
        # The score of the board
        return np.sum(remaining_numbers)*last_number

    def print(self, draws, wining_draw):
        """
        This method will print the board in the terminal with

        :param draws: Full set of draws ; 1D array
        :param wining_draw: The index of the draws array for the wining
        draw of the board
        :return: Nothing will be returned
        """

        # The last drawn number
        last_number = draws[wining_draw]
        # All the drawn numbers till the board wins including the wining draw
        drawn_numbers = draws[:wining_draw + 1]
        rows, cols = self.shape
        print("--------------------------")
        print('Board Id : ', self.id)
        for row in range(rows):
            print("|", end="")
            for col in range(cols):
                # All the numbers will be padded with 0 in the left if it is single digit
                # for visual clarity
                # If the value equal to the last draw, text will be colored green
                if self.board[row, col] == last_number:
                    print(colored(str(self.board[row, col]).zfill(2), 'green'), "|", end=" ")
            # If the value is drawn, text will be colored red
            elif self.board[row, col] in drawn_numbers:
                print(colored(str(self.board[row, col]).zfill(2), 'red'), "|", end=" ")
            else:
                print(colored(str(self.board[row, col]).zfill(2), 'white'), "|", end=" ")
        print()

    print("Legend ...")
    print("Color ", colored("green", 'green'), " represent the last drawn number")
    print("Color ", colored("red", 'red'), " represent the previously drawn numbers")
    print("Color ", colored("white", 'white'), " represent the remaining numbers")
    print("————————————")

def print_card(self,  draws, wining_draw, total_boards, title):
    """
    Generate the card as a .png file
    :param draws: 1D array of draws
    :param wining_draw:  the wining draw (count) as int
    :param total_boards: Total numbers of boards loaded ats in
    :param title:  Title of the image generated as string
    :return:
    """

    min_width = 400
    rows, cols = self.shape
    offset_left, offset_top, offset_right, offset_bottom = (20, 20, 20, 20)
    row_height = 40
    col_width = 40
    text_width = 200
    texts = [
        "Total Draws: " + str(len(draws)),
        "Total boards: " + str(total_boards),
        "Board Id (This) : " + str(self.id),
        "Board Score : " + str(self.board_score(draws, wining_draw)),
        "The Wining Draw : " + str(wining_draw) + " Draw",
        "",
       ]

    legends = [
        ['blue', 'Last Drawn Number'],
        ['orange', 'Previously Drawn Numbers'],
        ['gray', 'Remaining Numbers'],
    ]

    line_height = 20
    title_height = 20
    min_height = offset_top+line_height * (len(texts) + len(legends))+offset_bottom
    width = offset_left + col_width*rows + offset_right + text_width
    width = width if width > min_width else min_width
    height = offset_top + row_height*cols + offset_bottom + title_height
    height = height if height > min_height else min_height
    img = Image.new('RGB', (width, height), color='white')
    img1 = ImageDraw.Draw(img)

    text_xy = (50, 10)
    img1.text(text_xy, title, fill="black", font=None, anchor=None, spacing=0, align="center")

    text_offset_x = offset_left + col_width * rows + 20
    text_offset_y = offset_top + title_height

    for index, text in enumerate(texts):
        text_xy = (text_offset_x, text_offset_y + line_height * index)
        img1.text(text_xy, text, fill="black", font=None, anchor=None, spacing=0, align="left")

    for index, legend in enumerate(legends):
        x1, y1 = (text_offset_x, text_offset_y + line_height*len(texts)+line_height * index)
        x2, y2 = (x1+20, y1)
        xy = [(x1, y1), (x2, y2)]
        img1.line(xy, fill=legend[0], width=5)

        text_xy = (x2 + 10, y2)
        img1.text(text_xy, legend[1], fill="black", font=None, anchor=None, spacing=0, align="left"
        # The last drawn number
        last_number = draws[wining_draw]
        # All the drawn numbers till the board wins including the wining draw
        drawn_numbers = draws[:wining_draw + 1]

        for row in range(4, -1, -1):
            for col in range(cols):
                x1 = offset_left + col_width * row
                y1 = offset_top + row_height * col + title_height
                x2 = x1 + col_width
                y2 = y1 + row_height
                img1.rectangle([(x1, y1), (x2, y2)], fill=None, outline="black")
                text_xy = (x1 + 10, y1 + 10)
                text = str(self.board[col, row]).zfill(2)
                if self.board[col, row] == last_number:
                    text_color = "blue"
                elif self.board[col, row] in drawn_numbers:
                    text_color = "orange"
                else:
                    text_color = "gray"

                img1.text(text_xy, text, fill=text_color, font=None, anchor=None, spacing=0, align="lef
        img_name = str(self.id)+'.png'
        img.save('output/board/'+str(self.id)+'.png')
        return img_name
```

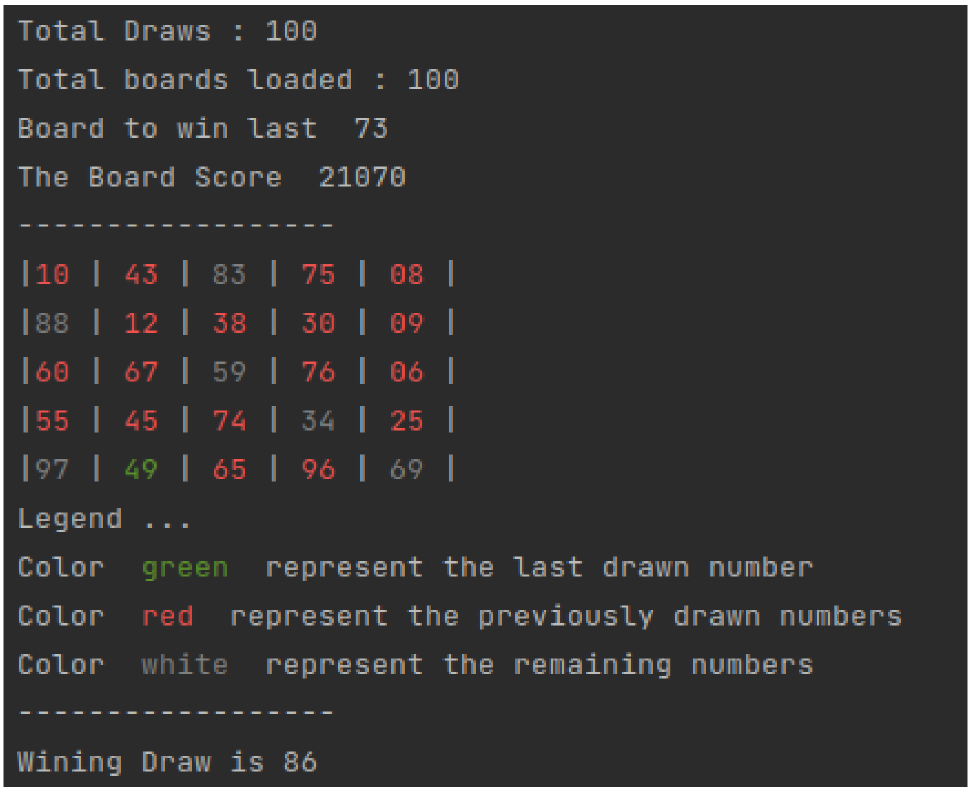
3.4.3 main.py

Listing 5: bingo.py

```
import loader as data
import utils
import numpy as np
import matplotlib.pyplot as plt
from termcolor import colored

# Load draws from file
draws = data.draws('real', 'numbers.txt')
print("Total Draws :", len(draws))

# Load boards from file
boards = data.boards('real', 'boards.txt')
print("Total boards loaded :", len(boards))
wining_draws = []

for board in boards:
    # get indexes of the board as explained in the document (step 3 and 4)
    indexes = utils.get_indexes(draws, board)
    # get the wining draw of the board
    # feed winging draw of the board to Wining Draws array
    wining_draws.append(utils.get_wining_draw(indexes))

print("Board to win last ", np.argmax(wining_draws)+1)
print("The Board Score ", utils.get_board_score(boards[np.argmax(wining_draws)],
                                                draws, np.max(wining_draws)))
utils.print_board(boards[np.argmax(wining_draws)],
                  draws, np.max(wining_draws))
print("Wining Draw is", np.max(wining_draws)+1)


# this section will generate simple plot of  wining draw against
# the board number for all the board
x = []  # X axis - Board Ids
scores = []
for board, win in enumerate(wining_draws):
    wining_score = utils.get_board_score(boards[board], draws, win)
    scores.append(wining_score)
    x.append(board)

fig, ax1 = plt.subplots()

ax2 = ax1.twinx()
ax1.plot(x, wining_draws, 'g-', label="Draws")
ax2.plot(x, scores, 'r-', label="Scores")

ax1.set_xlabel('Boards')
ax1.set_ylabel('Wining Draws', color='g')
ax2.set_ylabel('Wining Scores', color='r')

ax1.annotate('Board to win first ('+str(np.argmin(wining_draws)+1)+')',
             xy=(np.argmin(wining_draws)+1, np.min(wining_draws)),
             xytext=(np.argmin(wining_draws)+5, np.min(wining_draws)-5),
             arrowprops=dict(facecolor='black', shrink=0.01),
             )

ax1.annotate('Board to win last ('+str(np.argmax(wining_draws)+1)+')',
             xy=(np.argmax(wining_draws)+1, np.max(wining_draws)),
             xytext=(np.argmax(wining_draws)+5, np.max(wining_draws)+5),
             arrowprops=dict(facecolor='black', shrink=0.01),
             )

ax2.legend(loc='upper right')
ax1.legend(loc='upper left')

plt.title("Comparison between wining score and number of draws to win")
# plt.legend(loc='upper right')
plt.show()
```

4 Results

The board to win last amoung the provied board is the 73rd board from the top in the given data set. The board stays in the game until the 86th draw with a wining score of 21070. Program output for given data set is shown below in the Figure 2



```
Total Draws : 100
Total boards loaded : 100
Board to win last  73
The Board Score  21070
------------------
|10 | 43 | 83 | 75 | 08 |
|88 | 12 | 38 | 30 | 09 |
|60 | 67 | 59 | 76 | 06 |
|55 | 45 | 74 | 34 | 25 |
|97 | 49 | 65 | 96 | 69 |
Legend ...
Color  green  represent the last drawn number
Color  red  represent the previously drawn numbers
Color  white  represent the remaining numbers
------------------
Wining Draw is 86
```

Figure 2: The board which will win last in the game. Last until the 86th draw; The score is 21070

5 Discussion

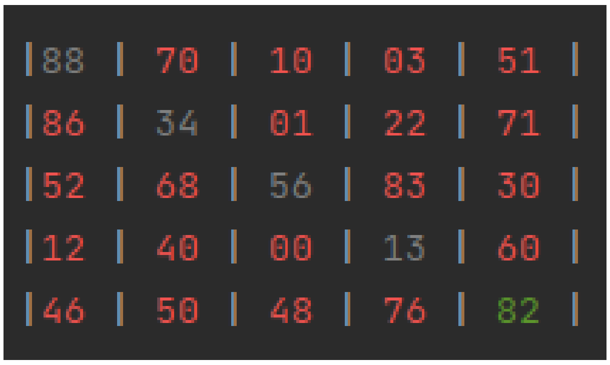
5.1 Validity of the solution

The provided solution is valid to find the last winging board as well as the first wining board provided that boards are already generated.

However, there are about 3.7E+48 possible board arrangements for bingo with 100 numbers. Searching for the last winning board in such a set would be highly computationally expensive even with massive parallelization and state of the art gpus.

5.2 The last winning board

Theoretically, the last wining board should stay in the game until the 96th draw to win in a 100-number bingo game. The last winning board of the provided data set wins at 86th draw. In other words, provided data set does not contain the board which lasts the longest in the game. Generating all the possible boards and Searching for the board to win last is impossible with the computational resources at disposal. However, such a board configuration can be constructed with a simple Theoretical approach, provided that we have access to all the draws before the game. Any board which contains the last five numbers of the draw sequence in any diagonal of the board will stay in the game until the 96th draw. A possible configuration for the provided draw configuration has been given below. The configuration in Figure 3 has been added to the end of the data set provided and placed in the "data\beyond" folder. It can be accessed by changing the directory name in the data loader to "beyond" in lines 6 and 11 the main.py file.



```
|88 | 70 | 10 | 03 | 51 |
|86 | 34 | 01 | 22 | 71 |
|52 | 68 | 56 | 83 | 30 |
|12 | 40 | 00 | 13 | 60 |
|46 | 50 | 48 | 76 | 82 |
```

Figure 3: A board configuration which will last until the 96th draw; The board score is 15662

5.3 The board score

The board score has no clear relation with the winning draw of the board. The lowest score a board can get in 100 numbers bing game is 14 for any draw configurations ending with 1,2,3,4,5. The 96th position has to be 1 and 97th to 100th possitions can be any permuations of 2,3,4,5. Also, the highest score a board can get is $495,000$ for any draw configuration starting with 1,2,3,4,100. The 5th position has to be 100 and 1st to 4th possitions can be any permuations of 1,2,3,4. A comparison of the number of draws to wind and the winning score of a board is shown in Figure 4

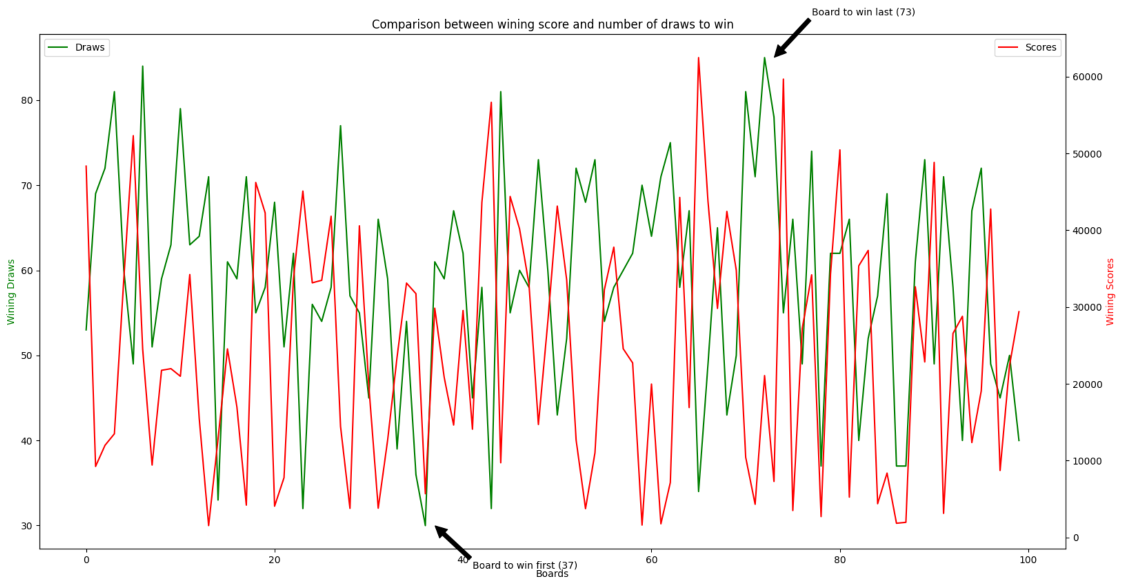


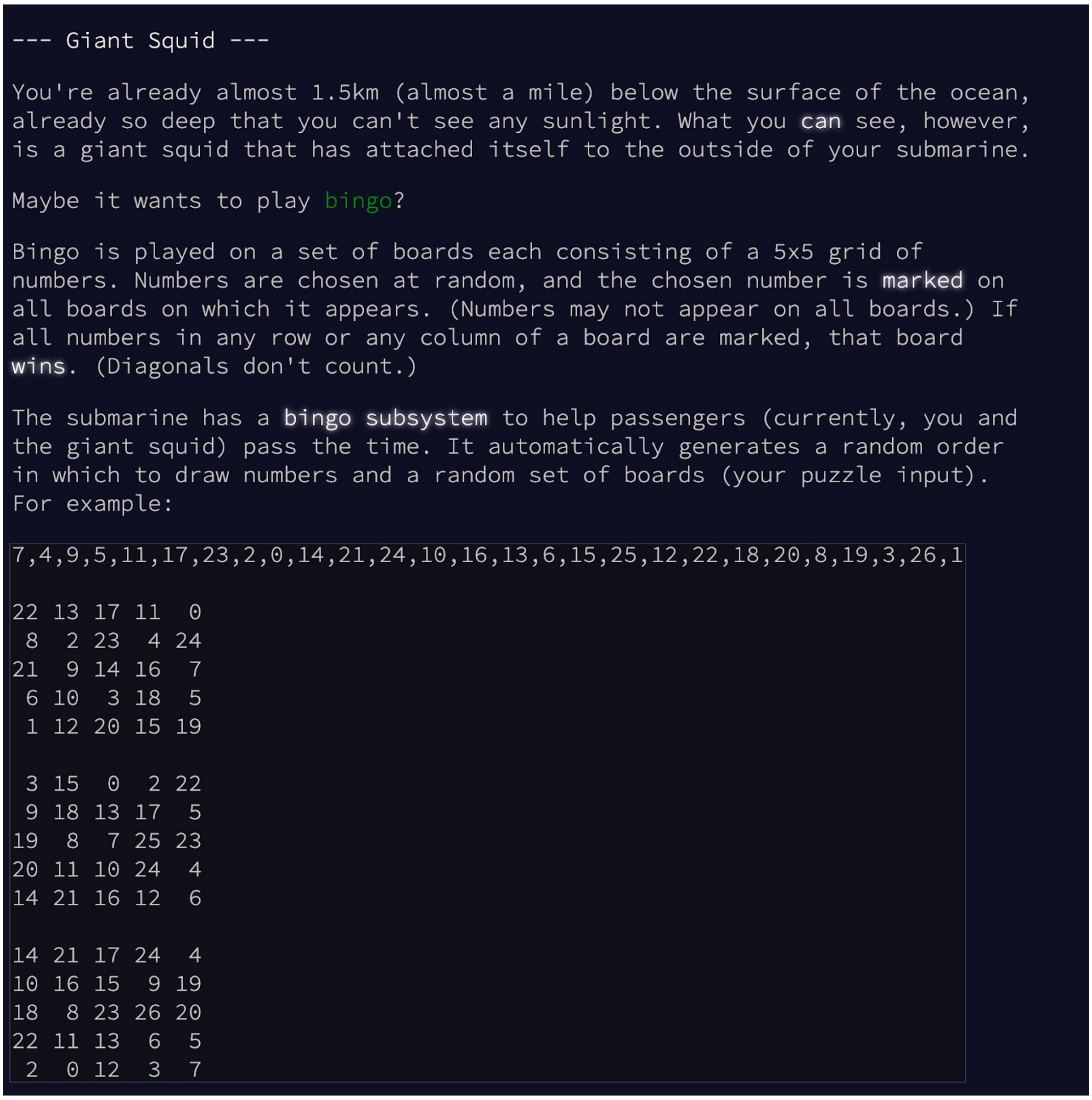
Figure 4: A Comparision between the winning draw and the winning score for the given data set

6 Appendix

The document provided by Kera is attached

Programming test

The purpose of this test is to see what data structures and implementation is used to solve a logical problem.



--- Giant Squid ---

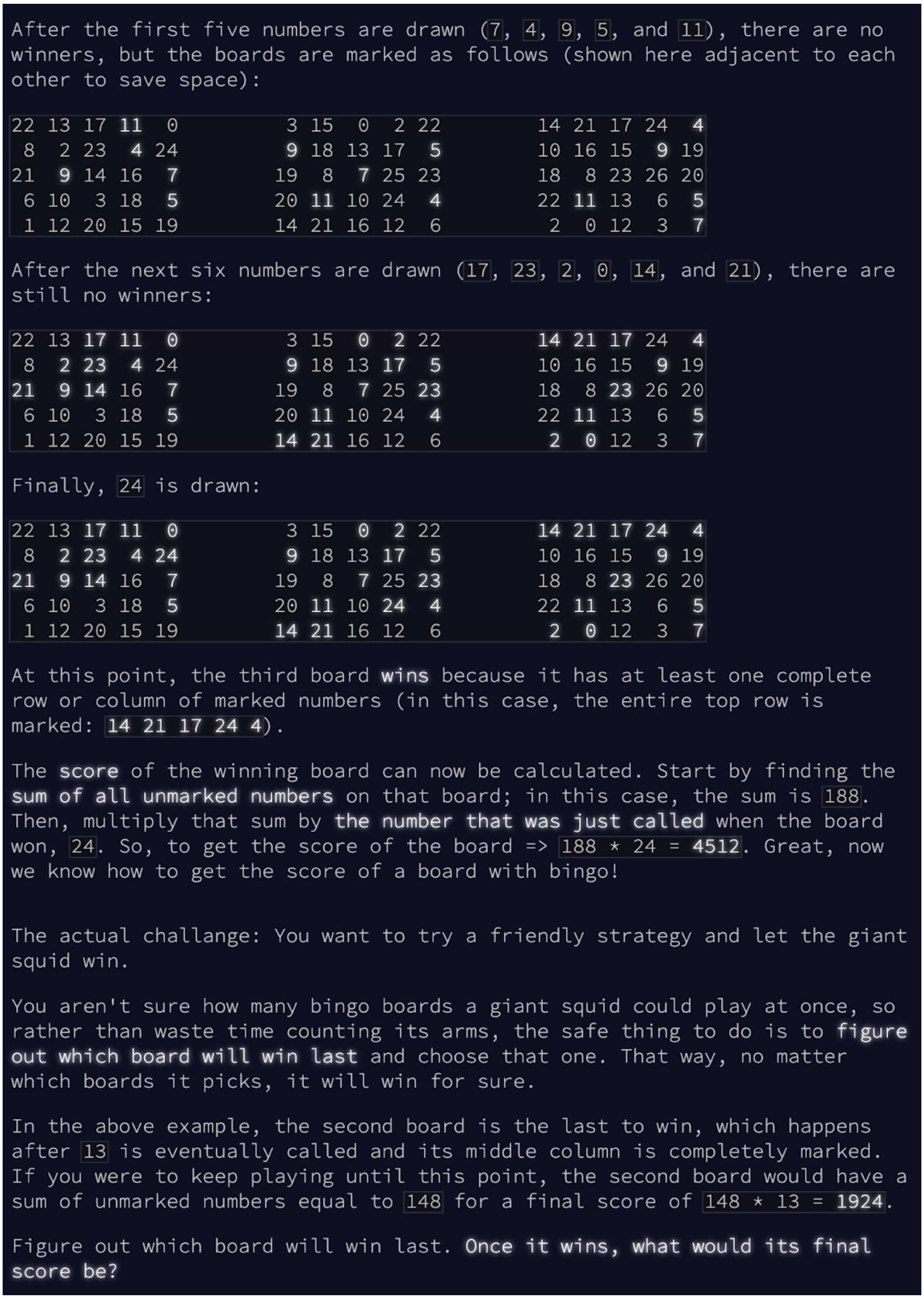
You're already almost 1.5km (almost a mile) below the surface of the ocean, already so deep that you can't see any sunlight. What you **can** see, however, is a giant squid that has attached itself to the outside of your submarine.

Maybe it wants to play bingo?

Bingo is played on a set of boards each consisting of a 5x5 grid of numbers. Numbers are chosen at random, and the chosen number is **marked** on all boards on which it appears. (Numbers may not appear on all boards.) If all numbers in any row or any column of a board are marked, that board **wins**. (Diagonals don't count.)

The submarine has a **bingo subsystem** to help passengers (currently, you and the giant squid) pass the time. It automatically generates a random order in which to draw numbers and a random set of boards (your puzzle input). For example:

```
7,4,9,5,11,17,23,2,0,14,21,24,10,16,13,6,15,25,12,22,18,20,8,19,3,26,1

22 13 17 11  0
 8  2 23  4 24
21  9 14 16  7
 6 10  3 18  5
 1 12 20 15 19

 3 15  0  2 22
 9 18 13 17  5
19  8  7 25 23
20 11 10 24  4
14 21 16 12  6

14 21 17 24  4
10 16 15  9 19
18  8 23 26 20
22 11 13  6  5
 2  0 12  3  7
```



After the first five numbers are drawn (`7`, `4`, `9`, `5`, and `11`), there are no winners, but the boards are marked as follows (shown here adjacent to each other to save space):

```
22 13 17 11  0         3 15  0  2 22        14 21 17 24  4
 8  2 23  4 24         9 18 13 17  5        10 16 15  9 19
21  9 14 16  7        19  8  7 25 23        18  8 23 26 20
 6 10  3 18  5        20 11 10 24  4        22 11 13  6  5
 1 12 20 15 19        14 21 16 12  6         2  0 12  3  7
```

After the next six numbers are drawn (`17`, `23`, `2`, `0`, `14`, and `21`), there are still no winners:

```
22 13 17 11  0         3 15  0  2 22        14 21 17 24  4
 8  2 23  4 24         9 18 13 17  5        10 16 15  9 19
21  9 14 16  7        19  8  7 25 23        18  8 23 26 20
 6 10  3 18  5        20 11 10 24  4        22 11 13  6  5
 1 12 20 15 19        14 21 16 12  6         2  0 12  3  7
```

Finally, `24` is drawn:

```
22 13 17 11  0         3 15  0  2 22        14 21 17 24  4
 8  2 23  4 24         9 18 13 17  5        10 16 15  9 19
21  9 14 16  7        19  8  7 25 23        18  8 23 26 20
 6 10  3 18  5        20 11 10 24  4        22 11 13  6  5
 1 12 20 15 19        14 21 16 12  6         2  0 12  3  7
```

At this point, the third board **wins** because it has at least one complete row or column of marked numbers (in this case, the entire top row is marked: `14 21 17 24 4`).

The **score** of the winning board can now be calculated. Start by finding the **sum of all unmarked numbers** on that board; in this case, the sum is `188`. Then, multiply that sum by **the number that was just called** when the board won, `24`. So, to get the score of the board => `188 * 24 = 4512`. Great, now we know how to get the score of a board with bingo!

The actual challange: You want to try a friendly strategy and let the giant squid win.

You aren't sure how many bingo boards a giant squid could play at once, so rather than waste time counting its arms, the safe thing to do is to **figure out which board will win last** and choose that one. That way, no matter which boards it picks, it will win for sure.

In the above example, the second board is the last to win, which happens after `13` is eventually called and its middle column is completely marked. If you were to keep playing until this point, the second board would have a sum of unmarked numbers equal to `148` for a final score of `148 * 13 = 1924`.

Figure out which board will win last. **Once it wins, what would its final score be?**

Test Input data

This is the test input data that is mentioned in the puzzle above. Just so it is easy to copy/paste input data into your code, when building your solution.

Draw number:

```
7,4,9,5,11,17,23,2,0,14,21,24,10,16,13,6,15,25,12,22,18,20,
8,19,3,26,1
```

Boards:

```
22 13 17 11  0
 8  2 23  4 24
21  9 14 16  7
 6 10  3 18  5
 1 12 20 15 19

 3 15  0  2 22
 9 18 13 17  5
19  8  7 25 23
20 11 10 24  4
14 21 16 12  6

14 21 17 24  4
10 16 15  9 19
18  8 23 26 20
22 11 13  6  5
 2  0 12  3  7
```

Final score:

```
1924
```

Real Input data:

When you have built a solution that works for the test data above you need to use this “real” input data that you need to use to produce the real “Score”.

Draw numbers:

```
1,76,38,96,62,41,27,33,4,2,94,15,89,25,66,14,30,0,71,21,48,
44,87,73,60,50,77,45,29,18,5,99,65,16,93,95,37,3,52,32,46,8
0,98,63,92,24,35,55,12,81,51,17,70,78,61,91,54,8,72,40,74,6
8,75,67,39,64,10,53,9,31,6,7,47,42,90,20,19,36,22,43,58,28,
79,86,57,49,83,84,97,11,85,26,69,23,59,82,88,34,56,13
```

Boards:

```
85 23 65 78 93
27 53 10 12 26
 5 34 83 25  6
56 40 73 29 54
33 68 41 32 82

 8 31 14 70 91
53 49 86 13 21
66 28 76 78 93
39 63 80 43 23
56 25 60 67 72

67 78 36 64 14
46 16 80 23 94
```

```
22 47 51 65 57
33 76 21 92 97
31 95 54 27 20

 1 77 86 43 30
28 88  7  5 60
66 24  3 57 33
38 23 59 84 44
74 47 17 29 85

21 50 86  2 70
85 19 22 93 25
99 38 74 30 65
81  0 47 78 63
34 11 51 88 64

45 15 29 81 30
75 21 88 91 49
39 20  4 17 78
10 12 38 11  7
98  6 65 69 86

36 20 31 44 69
30 65 55 88 64
74 85 82 61  5
57 17 90 43 54
58 83 52 23  7

42 16 82 86 76
60 26 27 59 55
 7 53 22 78  5
```

```
18 61 10 15 17
28 46 14 87 77

21 43 15 47 61
24 76 28  3 27
19 62 69 82 93
49 29 97 74 41
92 36 37 99 40

31  4  3 62 51
24 57 78 67 53
13  5 76 38 55
79  9 75 98 71
65  1 39 18 47

59  4 38 95 99
85 68 69 93 43
83 57 48 42 15
47 50 80 79 90
56 87 78 64 25

21 37 14 67 95
88 39 26 38 49
89 83 54 77 96
48 86 94 19 20
43 41  8 74 58

 1 36 12 90 91
63 21 98 82 66
39 86  7 52 77
80 81 44 33 58
```

```
78 30 11 51 28

81 74  7 33 96
75 60 87 47 91
39 73 30 50 13
 4 41  9 43 77
34 82 72 48 12

93 63 74 25 57
29 76  9 45 70
98 77 71 16 41
47 54 18 14 55
31 89 67 87 83

 8 72 45 93 68
74 26 69 94 65
28  9 20 47 41
46 54 21 56 22
84 62 18 15 48

20 51 81 40 69
71 10 13 93 75
44 86  0 95 37
99 39 76 80 66
14 64 49 62 27

75  7 51 86 79
43 30 61 39 16
85 63 90 28 96
88 78 72 31 73
98 87 23 19 58
```

```
20 95 47 97 12
92 25 68 87 91
37 10 78 23 63
74 93 58 39  5
76 51 48 72 16

37 18 32 34 85
22 31 98 42 19
29 72 48 76 25
47  1 21  7 53
79 82 86 52 78

20 16 47 78 92
88 15 71 67  2
 5 52 90 70  9
22 49 28 82 27
 6 19 61 73 48

71 26  7 11 79
52 30 47  1 31
17 75 94 91 28
81 98 23 55 21
77 15 39 24 16

 5 75 44 88 65
89 45 23 69 19
41 61 67 52 54
47 38 57 12 98
62 70 26 87 53
```

```
50  4 65 77 25
 6 21  5 27 92
39 63 97 75 79
60 34 87 26 74
99 24 44 85  2

13 64 38 78 21
74 17 83 57 94
25 39 69 53  4
54 33 81 50 76
42 75 19 77 26

63 31 70 19 39
38 87 15 90 75
61 98  6 29 86
78 62 32 11 60
55 97 13 73 82

51 63 68 84 36
12 33 37 31  8
18 41 34 74 23
72 39 85 48 60
24 19 29 88  0

46 51 17 23 13
20 93 97 99 81
57 47 33 84 44
28 96  2 43 56
68 36 62 15  5

81 99  5 30 10
```

```
38 62 57  8 37
 7 86 98  3 54
46 82 96 15 72
83  1 75 25 50

47 57 11 61 27
53 10 31 91 98
76 85 55 38 23
 6 81 67 71 70
35 29 17 50 56

24 65 15  1 89
45 60 97 23 14
84 56 58  5 54
 3 72 51 46 79
67 70 78 34 77

38 11 54 23  2
33 14 10 96 63
43  5 36 20 30
70 53 66 71  9
91 90 21  7 88

94 44  4 86 26
39 70 54 50 30
55 40 12 72 71
68  7 66 47 91
31 24 13  1 96

79 14 40 87 68
16 32 53 46 98
```

```
38 95 21 89 69
62 60 19 81 33
70 52 28 83  0

62 42 38 48 64
61 79 78 97 98
89  7  3 29 68
92 76 14 67  1
41 99 72 47 60

 5 75 18 42 33
72 61 36 31 29
19 58  1 34 94
54 84 92 99 38
76 68 79 53 37

14 91 37  5 98
68 29 34 76 43
75  0 67 33 69
81 47 58 30 93
88 92 42 77 54

64 24 28 54 53
72 68  3 73  4
83  6 59 66 94
87 80 55 20 16
13 82 74 31 70

63 92 71  0 83
98 40 50 55  2
88  5 85 30 23
```

```
10 75 81 58 68
51 31 14 89  1

67 93 94 54 53
38 71 34 40 24
31 63 30 99 75
 4 57 86 19 70
60 49 87 68 74

56 94 79 53  7
24 12 19  6 99
82 51 41 46 43
17 49 52 78 55
75 48 61 70 87

14 55 32 21 31
88 83 23 44  4
 1 77 45 90 85
46 81 51 27 62
60 24 29 18  0

95 92 91 27 26
22 43 45 64 62
83 23 25 85 94
84 53 72 28 20
75 60 52 18 73

95 41  7 21 32
58 65 16 56 97
68 25 91 83 24
66 89 15 55  6
```

```
 2 30 84 10 90

58 86 44 19 74
57 89 17  6 83
77 35 60 32 13
97 63 62 28 76
55 31 11  0 52

33 39 59 42 45
61 50 92  9 79
15  0 28  5 72
91 24 21 29 87
86 76 43 31 93

63 11 86 45 85
96 74 66 93 32
95 30 99 23 18
69 97 48 15  1
42 87 47 83 80

93  5 40 64  2
44 51 15 54 83
69 77 90 58 11
 0 48 43 30 55
25 72 38 73 52

89 58 71 68 15
23 65  9 36 74
21 29 42 79 98
55 47 33 39 28
16 75 91 69 57
```

```
13 79 12 71  2
60 94 99 43 82
84 89 29 91 87
74 80 25 32 21
70 14 68 92 11

78  1 16 51 87
58 94 59 15 43
79 41 50 47 39
53 37  9 28 72
34 63 89 35 18

31 67 70 42 43
60  2 89 49 22
56 17 81 24 74
20 65  1 96 51
68  7  0 38 25

59 14 29 53 19
 9  2 11 33 44
81  6 10 47 58
20 34 62 55 40
71 38 69 45 78

59 36 70 42 21
 3 16 49 79 98
74 25  8 84 19
61 80 47 65 64
91 62 52  9 40
```

```
 1 85 63  7  2
 0 20 61 26 77
99 37 74 42 76
25 94 19 78 60
79 72 95 22 11

51 21 79 76 32
55 23 69 19 61
71 54 94 47 92
 5 64  6 68 16
91 81  9 99 30

61 69 82 86 68
66 81 28 38 36
26 29 31 11  8
72 51 12 95 63
18 30 88 17 32

34  8 14 42 67
66 79 65 20 52
37 87 74 24  3
59 54 21 32 89
31  4 62 76 30

11 93  8 92 55
38 72 99  3 83
12 75  0 41 46
17 25  5 39 48
14 18 86 29 84

 6 20 41 51 48
 5 67 30 24 47
 3  8 92 22 39
 4 56 36 31 75
 2 45 85 81 96

47 43 72 22  3
19 87 53 12 60
29 40 56 68 18
66 97 70 33 39
85 37  0 90 98

61 35 81 84 94
11  1 58 45 77
 6 99 67 36 43
 5  7  0 87 80
44 78 39 70 20

58 34 49 29 75
17 15 28 23 84
59 25 92 48  0
20 81 47  3 71
68 60  5 22 87

90 32 41 39  6
36 78 67 24 50
55 72 52 75 44
87 15 92 31 58
83 89 68 19 43

99 44 53 68 25
71 67 16 19 36
```

```
35 58 14 86 48
88 18 61 24 23
87  9 91 37 15

37  5 63 68 28
41 50 76 99 64
34 92 78 94 71
11 96 97 42 58
33 45  0 93 48

33 68  9 12 81
60 98 28  8 99
14 17  6 82 15
57 69 43 38 29
47 84 76 22 18

79 70 92 38 47
12 82 98 46  0
76 15 53 59 97
18 52 49 29 96
44 64 68 89 24

95 14 17 27 42
55 43 57 29 25
34 73 86 50 16
69 37 75 63 39
78 79  3  4 30

27 31 15 92 46
36 23 72 40 50
51 99 55 89 21
```

```
12 70 84 63 85
78 88 77 75  0

15 67 40 39 28
 9 79 22 52 75
96 65 86 98 14
97 87 44 84 68
36 26 89 43 27

79 59 48 27 36
85 92 93 76 24
 2 25  7 42 90
23 29 74 35 86
58 60 31 75 57

10 43 83 75  8
88 12 38 30  9
60 67 59 76  6
55 45 74 34 25
97 49 65 96 69

59 86 15  3 19
89  4 74 61 23
52 98  8 79 39
95 17 22 14 51
50 18 94 30 84

19 63 58 72 67
35 93 29 91  0
39 26 43 84 21
70 42  2 53 12
```

```
59 99  8  1 86

23 86 34 22 65
71 10 16 50 91
66 89 49 81 43
40  7 26 75 61
62 59  2 46 95

24 21  0 49 25
92 42 48 12  7
81 93 59 68  3
14 23 63 39 29
35 43  6 44 89

67 74 95 34 10
39 90 59 44 51
17 16 97 24 62
20 54 76 63 88
87 66 14 78 82

96 86 67 59 79
66  3 30 77 71
 2 91 99 82 31
48 65 75 98 53
63 54 64 76  1

85 96 40 98 24
16 20 10 23 17
79 59 53 42 65
67  2  5 80 75
62 38 19 74 73
```

```
43 10 79 92  8
52 36  4  5 67
56 29 33 24 97
85 17 53 75 65
62 64  1 21 83

93 92 79 17 12
40 88  6 82 34
90 96 53 25 43
14 62 54 10 39
49 68 41 16 44

67 99 24 58 76
43 53 59 54 51
47  6 61  8  2
80 68 90 14  4
29 46 94 89 50

14 45 19 33 43
 6 55  4 31 80
51  2 69 68 61
71 70 79 91 93
66 18 54 13 87

 8 45 61 54 30
85 16 19 82 37
56 39 11 47  4
74 70 10 60 91
21 63 95 53 72
```

```
71 21 63 86 27
53 52 40 23 81
 2 47 92 68 15
46 45 31  8  1
34 80 37 11 69

96  0 15 90 66
65 43 92 83 18
 3 47 19  8 32
71 26 42 34 28
62 99 55  5 12

37 99 30 21  3
63 18 68 47 27
57  0 65 85 20
 7 58 40 92 43
15 19  5  4 53

46 16 45 95 68
 6 44 31 47 73
84 82 71 75 94
26 25 17 32 49
18 96 13 58  9

71 36 13 68 10
84  7 60 79 41
 1 83 43 81 97
90 53 80 19 38
48 25 32 42 29

37 68 86 44 78
```

```
87 67 77 70 60
45 34 27 15 47
12 21 13 55 26
81 41 63 40 74

24 50 93 94 57
99  4 56  5 28
42 31 22  6 76
90 89 16 49 59
 9  7 43 71 54

69 75 94 38 46
52 64 50 72 42
76 63 13 60 10
99 80 43 33 17
25 31  4 89 22

88 57 22 66 34
85 16 87 95 59
73  2 46  5 29
25 69 53  6 14
96 77 19 91 43

46 99 52 47 76
89 53 24 13 59
45  5  1 30 19
68 25 22 10 73
42 27 31  0 94

42 44 98 89 87
65 10 80 56 41
```

```
 3 35 95 48 43
85 97 83 12 94
50 38 93 47 17

16 73 18 81 89
 6 48 54 93 19
35 52 88 49 31
43 79 83 14 28
50 62 98 26 22

38 47  7 20 35
45 76 63 96 24
98 53  2 87 80
83 86 92 48  1
73 60 26 94  6

80 50 29 53 92
66 90 79 98 46
40 21 58 38 60
35 13 72 28  6
48 76 51 96 12

79 80 24 37 51
86 70  1 22 71
52 69 10 83 13
12 40  3  0 30
46 50 48 76  5
```

Final score:

```
<This is the score you should produce>
```

Verify you score:
When you think that you solution is complete, you can verify your Final score by sending a POST request to https://customer-api.krea.se/coding-tests/api/squid-game

With the body:

```
{
  "answer": "<your answer>",
  "name": "<your name>"
}
```

Good luck and have fun :)!!