RIGA TECHNICAL UNIVERSITY
FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

Practical assignment #1
“Fundamentals of Artificial Intelligence”

https://github.com/MirFurqann/Team-18-_Game.git

Author: Team 18
Daniil Hrynyshyn 221ADB204
Furqan Ul Islam 211ADB086
Sriram Raghukumar 221ADB037
Nickson Mugerwa 211ADB046
Camille Cogno-Bourdieu 240AEB017

Checked: Alla Anahona-Naumeca

2023/ 2024study year

1. Software Demonstration or User Manual:

Link to the Github Repository

https://github.com/MirFurqann/Team-18-_Game.git

Our software is a number game where players take turns removing numbers from a string. The goal is to strategically remove numbers to maximize your score while minimizing your opponent's score. Here's how to play:

- **Start Menu:** Upon launching the software, the user is presented with a start menu. Here, they can choose whether to play as the "User" or let the "Computer" start the game. Additionally, they can select between two algorithms: "Minimax" or "Alpha-Beta".
- **Game Setup:** After making their selections, the user is prompted to enter the length of the string (15-25). This determines the number of numbers in the string for the game.
- **Game Interface:** Once the game starts, the interface displays the current scores of the human player and the computer player. It also indicates whose turn it is to play. The user can remove numbers by clicking on them, dragging them to the designated drop area, and releasing the mouse button.
- **End of Game:** The game ends when there are no numbers left in the string. The software then displays the winner and provides statistics such as the average time per move and the total number of nodes visited during the game.
- **Restart or Quit:** After the game ends, the user can choose to restart the game by pressing "R" or quit the software by pressing "Q".

Rules :

Additional software requirements
At the beginning of the game, the human player indicates the length of a numerical string to be used in the game, which can be in the range of 15 to 25 numbers. The game software randomly generates a numerical string of the specified length, including the numbers 1 to 3.

Game description
At the beginning of the game, the generated numerical string is given and each player has 80 points. Players alternate moves, taking one number from the string in each turn and subtracting it from their current score. The game ends when the string is empty. If the

players have the same number of points, the result is a draw. Otherwise, the player with the higher number of points left wins.

This is **a two-player** game **zero-sum** with **perfect information !**

ANALYSING THE PROBLEM :

State : ??
Action : removing one number from the string and subtracting it from their current score
Initial state : numerical string between 15 and 25 containing random number of “1”,”2” and “3”
Goal state : no more number in the numerical string

Parameters :

- The size of the numerical string list
- The number of “1”, “2” and “3” in the string list
- Points of player 1
- Points of player 2

$$Total\_(string\_list) = x + y + z$$

Total_(string_list) ∈ [15;25]
x = number of “1” in the string list x ∈ [0; **Total_(string_list)**]
y = number of “2” in the string list y ∈ [0; **Total_(string_list)**]
z = number of “3” in the string list z ∈ [0; **Total_(string_list)**]

Different possibilities :

To determine the different possibilities, we first used Python to give us all the different string list in root states possible. We have a list of 3 numbers like that : (**x**, **y**, **z**).

This is the program we used :

```python
def find_xyz_combinations(target_sums):
    # Initialize a dictionary to hold the combinations for each target sum
    combinations = {target_sum: [] for target_sum in target_sums}

    # Iterate through possible values of x, y, z
    for x in range(26):
        for y in range(26):
            for z in range(26):
                # Calculate the sum of x, y, z
                current_sum = x + y + z
                # If the sum is one of the target sums, add the combination
                if current_sum in target_sums:
                    combinations[current_sum].append((x, y, z))

    return combinations

# Define the target sums
target_sums = range(15, 26)  # 15 to 25 inclusive

# Get the combinations
xyz_combinations = find_xyz_combinations(target_sums)

# Display the results
for target_sum, combinations in xyz_combinations.items():
    print(f"Sum = {target_sum}: {len(combinations)} combinations")
    print(combinations[:351])

xyz_combinations
```

Results :

Sum = 15: 136 combinations

(0, 0, 15), (0, 1, 14), (0, 2, 13), (0, 3, 12), (0, 4, 11), (0, 5, 10), (0, 6, 9), (0, 7, 8), (0, 8, 7), (0, 9, 6), (0, 10, 5), (0, 11, 4), (0, 12, 3), (0, 13, 2), (0, 14, 1), (0, 15, 0)

(1, 0, 14), (1, 1, 13), (1, 2, 12), (1, 3, 11), (1, 4, 10), (1, 5, 9), (1, 6, 8), (1, 7, 7), (1, 8, 6), (1, 9, 5), (1, 10, 4), (1, 11, 3), (1, 12, 2), (1, 13, 1), (1, 14, 0)

(2, 0, 13), (2, 1, 12), (2, 2, 11), (2, 3, 10), (2, 4, 9), (2, 5, 8), (2, 6, 7), (2, 7, 6), (2, 8, 5), (2, 9, 4), (2, 10, 3), (2, 11, 2), (2, 12, 1), (2, 13, 0)

(3, 0, 12), (3, 1, 11), (3, 2, 10), (3, 3, 9), (3, 4, 8), (3, 5, 7), (3, 6, 6), (3, 7, 5), (3, 8, 4), (3, 9, 3), (3, 10, 2), (3, 11, 1), (3, 12, 0)

(4, 0, 11), (4, 1, 10), (4, 2, 9), (4, 3, 8), (4, 4, 7), (4, 5, 6), (4, 6, 5), (4, 7, 4), (4, 8, 3), (4, 9, 2), (4, 10, 1), (4, 11, 0)

(5, 0, 10), (5, 1, 9), (5, 2, 8), (5, 3, 7), (5, 4, 6), (5, 5, 5), (5, 6, 4), (5, 7, 3), (5, 8, 2), (5, 9, 1), (5, 10, 0)

(6, 0, 9), (6, 1, 8), (6, 2, 7), (6, 3, 6), (6, 4, 5), (6, 5, 4), (6, 6, 3), (6, 7, 2), (6, 8, 1), (6, 9, 0),

(7, 0, 8), (7, 1, 7), (7, 2, 6), (7, 3, 5), (7, 4, 4), (7, 5, 3), (7, 6, 2), (7, 7, 1), (7, 8, 0)

(8, 0, 7), (8, 1, 6), (8, 2, 5), (8, 3, 4), (8, 4, 3), (8, 5, 2), (8, 6, 1), (8, 7, 0)

(9, 0, 6), (9, 1, 5), (9, 2, 4), (9, 3, 3), (9, 4, 2), (9, 5, 1), (9, 6, 0)

(10, 0, 5), (10, 1, 4), (10, 2, 3), (10, 3, 2), (10, 4, 1), (10, 5, 0)

(11, 0, 4), (11, 1, 3), (11, 2, 2), (11, 3, 1), (11, 4, 0)

(12, 0, 3), (12, 1, 2), (12, 2, 1), (12, 3, 0)

(13, 0, 2), (13, 1, 1), (13, 2, 0)

(14, 0, 1), (14, 1, 0)

(15, 0, 0)

Sum = 16: 153 combinations

(0, 0, 16), (0, 1, 15), (0, 2, 14), (0, 3, 13), (0, 4, 12), (0, 5, 11), (0, 6, 10), (0, 7, 9), (0, 8, 8), (0, 9, 7), (0, 10, 6), (0, 11, 5), (0, 12, 4), (0, 13, 3), (0, 14, 2), (0, 15, 1), (0, 16, 0)

(1, 0, 15), (1, 1, 14), (1, 2, 13), (1, 3, 12), (1, 4, 11), (1, 5, 10), (1, 6, 9), (1, 7, 8), (1, 8, 7), (1, 9, 6), (1, 10, 5), (1, 11, 4), (1, 12, 3), (1, 13, 2), (1, 14, 1), (1, 15, 0)

(2, 0, 14), (2, 1, 13), (2, 2, 12), (2, 3, 11), (2, 4, 10), (2, 5, 9), (2, 6, 8), (2, 7, 7), (2, 8, 6), (2, 9, 5), (2, 10, 4), (2, 11, 3), (2, 12, 2), (2, 13, 1), (2, 14, 0)

(3, 0, 13), (3, 1, 12), (3, 2, 11), (3, 3, 10), (3, 4, 9), (3, 5, 8), (3, 6, 7), (3, 7, 6), (3, 8, 5), (3, 9, 4), (3, 10, 3), (3, 11, 2), (3, 12, 1), (3, 13, 0)

(4, 0, 12), (4, 1, 11), (4, 2, 10), (4, 3, 9), (4, 4, 8), (4, 5, 7), (4, 6, 6), (4, 7, 5), (4, 8, 4), (4, 9, 3), (4, 10, 2), (4, 11, 1), (4, 12, 0)

(5, 0, 11), (5, 1, 10), (5, 2, 9), (5, 3, 8), (5, 4, 7), (5, 5, 6), (5, 6, 5), (5, 7, 4), (5, 8, 3), (5, 9, 2), (5, 10, 1), (5, 11, 0)

(6, 0, 10), (6, 1, 9), (6, 2, 8), (6, 3, 7), (6, 4, 6), (6, 5, 5), (6, 6, 4), (6, 7, 3), (6, 8, 2), (6, 9, 1), (6, 10, 0)

(7, 0, 9), (7, 1, 8), (7, 2, 7), (7, 3, 6), (7, 4, 5), (7, 5, 4), (7, 6, 3), (7, 7, 2), (7, 8, 1), (7, 9, 0),

(8, 0, 8), (8, 1, 7), (8, 2, 6), (8, 3, 5), (8, 4, 4), (8, 5, 3), (8, 6, 2), (8, 7, 1), (8, 8, 0)

(9, 0, 7), (9, 1, 6), (9, 2, 5), (9, 3, 4), (9, 4, 3), (9, 5, 2), (9, 6, 1), (9, 7, 0)

(10, 0, 6), (10, 1, 5), (10, 2, 4), (10, 3, 3), (10, 4, 2), (10, 5, 1), (10, 6, 0)

(11, 0, 5), (11, 1, 4), (11, 2, 3), (11, 3, 2), (11, 4, 1), (11, 5, 0)

(12, 0, 4), (12, 1, 3), (12, 2, 2), (12, 3, 1), (12, 4, 0)

(13, 0, 3), (13, 1, 2), (13, 2, 1), (13, 3, 0)

(14, 0, 2), (14, 1, 1), (14, 2, 0)

(15, 0, 1), (15, 1, 0)

(16, 0, 0)

Sum = 17: 171 combinations

(0, 0, 17), (0, 1, 16), (0, 2, 15), (0, 3, 14), (0, 4, 13), (0, 5, 12), (0, 6, 11), (0, 7, 10), (0, 8, 9), (0, 9, 8), (0, 10, 7), (0, 11, 6), (0, 12, 5), (0, 13, 4), (0, 14, 3), (0, 15, 2), (0, 16, 1), (0, 17, 0),

(1, 0, 16), (1, 1, 15), (1, 2, 14), (1, 3, 13), (1, 4, 12), (1, 5, 11), (1, 6, 10), (1, 7, 9), (1, 8, 8), (1, 9, 7), (1, 10, 6), (1, 11, 5), (1, 12, 4), (1, 13, 3), (1, 14, 2), (1, 15, 1), (1, 16, 0)

(2, 0, 15), (2, 1, 14), (2, 2, 13), (2, 3, 12), (2, 4, 11), (2, 5, 10), (2, 6, 9), (2, 7, 8), (2, 8, 7), (2, 9, 6), (2, 10, 5), (2, 11, 4), (2, 12, 3), (2, 13, 2), (2, 14, 1), (2, 15, 0)

(3, 0, 14), (3, 1, 13), (3, 2, 12), (3, 3, 11), (3, 4, 10), (3, 5, 9), (3, 6, 8), (3, 7, 7), (3, 8, 6), (3, 9, 5), (3, 10, 4), (3, 11, 3), (3, 12, 2), (3, 13, 1), (3, 14, 0)

(4, 0, 13), (4, 1, 12), (4, 2, 11), (4, 3, 10), (4, 4, 9), (4, 5, 8), (4, 6, 7), (4, 7, 6), (4, 8, 5), (4, 9, 4), (4, 10, 3), (4, 11, 2), (4, 12, 1), (4, 13, 0)

(5, 0, 12), (5, 1, 11), (5, 2, 10), (5, 3, 9), (5, 4, 8), (5, 5, 7), (5, 6, 6), (5, 7, 5), (5, 8, 4), (5, 9, 3), (5, 10, 2), (5, 11, 1), (5, 12, 0)

(6, 0, 11), (6, 1, 10), (6, 2, 9), (6, 3, 8), (6, 4, 7), (6, 5, 6), (6, 6, 5), (6, 7, 4), (6, 8, 3), (6, 9, 2), (6, 10, 1), (6, 11, 0)

(7, 0, 10), (7, 1, 9), (7, 2, 8), (7, 3, 7), (7, 4, 6), (7, 5, 5), (7, 6, 4), (7, 7, 3), (7, 8, 2), (7, 9, 1), (7, 10, 0)

(8, 0, 9), (8, 1, 8), (8, 2, 7), (8, 3, 6), (8, 4, 5), (8, 5, 4), (8, 6, 3), (8, 7, 2), (8, 8, 1), (8, 9, 0),

(9, 0, 8), (9, 1, 7), (9, 2, 6), (9, 3, 5), (9, 4, 4), (9, 5, 3), (9, 6, 2), (9, 7, 1), (9, 8, 0)

(10, 0, 7), (10, 1, 6), (10, 2, 5), (10, 3, 4), (10, 4, 3), (10, 5, 2), (10, 6, 1), (10, 7, 0)

(11, 0, 6), (11, 1, 5), (11, 2, 4), (11, 3, 3), (11, 4, 2), (11, 5, 1), (11, 6, 0)

(12, 0, 5), (12, 1, 4), (12, 2, 3), (12, 3, 2), (12, 4, 1), (12, 5, 0)

(13, 0, 4), (13, 1, 3), (13, 2, 2), (13, 3, 1), (13, 4, 0)

(14, 0, 3), (14, 1, 2), (14, 2, 1), (14, 3, 0)

(15, 0, 2), (15, 1, 1), (15, 2, 0)

(16, 0, 1), (16, 1, 0)

(17, 0, 0)

Sum = 18: 190 combinations

(0, 0, 18), (0, 1, 17), (0, 2, 16), (0, 3, 15), (0, 4, 14), (0, 5, 13), (0, 6, 12), (0, 7, 11), (0, 8, 10), (0, 9, 9), (0, 10, 8), (0, 11, 7), (0, 12, 6), (0, 13, 5), (0, 14, 4), (0, 15, 3), (0, 16, 2), (0, 17, 1), (0, 18, 0)

(1, 0, 17), (1, 1, 16), (1, 2, 15), (1, 3, 14), (1, 4, 13), (1, 5, 12), (1, 6, 11), (1, 7, 10), (1, 8, 9), (1, 9, 8), (1, 10, 7), (1, 11, 6), (1, 12, 5), (1, 13, 4), (1, 14, 3), (1, 15, 2), (1, 16, 1), (1, 17, 0),

(2, 0, 16), (2, 1, 15), (2, 2, 14), (2, 3, 13), (2, 4, 12), (2, 5, 11), (2, 6, 10), (2, 7, 9), (2, 8, 8), (2, 9, 7), (2, 10, 6), (2, 11, 5), (2, 12, 4), (2, 13, 3), (2, 14, 2), (2, 15, 1), (2, 16, 0)

(3, 0, 15), (3, 1, 14), (3, 2, 13), (3, 3, 12), (3, 4, 11), (3, 5, 10), (3, 6, 9), (3, 7, 8), (3, 8, 7), (3, 9, 6), (3, 10, 5), (3, 11, 4), (3, 12, 3), (3, 13, 2), (3, 14, 1), (3, 15, 0)

(4, 0, 14), (4, 1, 13), (4, 2, 12), (4, 3, 11), (4, 4, 10), (4, 5, 9), (4, 6, 8), (4, 7, 7), (4, 8, 6), (4, 9, 5), (4, 10, 4), (4, 11, 3), (4, 12, 2), (4, 13, 1), (4, 14, 0)

(5, 0, 13), (5, 1, 12), (5, 2, 11), (5, 3, 10), (5, 4, 9), (5, 5, 8), (5, 6, 7), (5, 7, 6), (5, 8, 5), (5, 9, 4), (5, 10, 3), (5, 11, 2), (5, 12, 1), (5, 13, 0)

(6, 0, 12), (6, 1, 11), (6, 2, 10), (6, 3, 9), (6, 4, 8), (6, 5, 7), (6, 6, 6), (6, 7, 5), (6, 8, 4), (6, 9, 3), (6, 10, 2), (6, 11, 1), (6, 12, 0)

(7, 0, 11), (7, 1, 10), (7, 2, 9), (7, 3, 8), (7, 4, 7), (7, 5, 6), (7, 6, 5), (7, 7, 4), (7, 8, 3), (7, 9, 2), (7, 10, 1), (7, 11, 0)

(8, 0, 10), (8, 1, 9), (8, 2, 8), (8, 3, 7), (8, 4, 6), (8, 5, 5), (8, 6, 4), (8, 7, 3), (8, 8, 2), (8, 9, 1), (8, 10, 0)

(9, 0, 9), (9, 1, 8), (9, 2, 7), (9, 3, 6), (9, 4, 5), (9, 5, 4), (9, 6, 3), (9, 7, 2), (9, 8, 1), (9, 9, 0),

(10, 0, 8), (10, 1, 7), (10, 2, 6), (10, 3, 5), (10, 4, 4), (10, 5, 3), (10, 6, 2), (10, 7, 1), (10, 8, 0),

(11, 0, 7), (11, 1, 6), (11, 2, 5), (11, 3, 4), (11, 4, 3), (11, 5, 2), (11, 6, 1), (11, 7, 0)

(12, 0, 6), (12, 1, 5), (12, 2, 4), (12, 3, 3), (12, 4, 2), (12, 5, 1), (12, 6, 0)

(13, 0, 5), (13, 1, 4), (13, 2, 3), (13, 3, 2), (13, 4, 1), (13, 5, 0)

(14, 0, 4), (14, 1, 3), (14, 2, 2), (14, 3, 1), (14, 4, 0)

(15, 0, 3), (15, 1, 2), (15, 2, 1), (15, 3, 0)

(16, 0, 2), (16, 1, 1), (16, 2, 0)

(17, 0, 1), (17, 1, 0)

(18, 0, 0)

Sum = 19: 210 combinations

(0, 0, 19), (0, 1, 18), (0, 2, 17), (0, 3, 16), (0, 4, 15), (0, 5, 14), (0, 6, 13), (0, 7, 12), (0, 8, 11), (0, 9, 10), (0, 10, 9), (0, 11, 8), (0, 12, 7), (0, 13, 6), (0, 14, 5), (0, 15, 4), (0, 16, 3), (0, 17, 2), (0, 18, 1), (0, 19, 0)

(1, 0, 18), (1, 1, 17), (1, 2, 16), (1, 3, 15), (1, 4, 14), (1, 5, 13), (1, 6, 12), (1, 7, 11), (1, 8, 10), (1, 9, 9), (1, 10, 8), (1, 11, 7), (1, 12, 6), (1, 13, 5), (1, 14, 4), (1, 15, 3), (1, 16, 2), (1, 17, 1), (1, 18, 0)

(2, 0, 17), (2, 1, 16), (2, 2, 15), (2, 3, 14), (2, 4, 13), (2, 5, 12), (2, 6, 11), (2, 7, 10), (2, 8, 9), (2, 9, 8), (2, 10, 7), (2, 11, 6), (2, 12, 5), (2, 13, 4), (2, 14, 3), (2, 15, 2), (2, 16, 1), (2, 17, 0)

(3, 0, 16), (3, 1, 15), (3, 2, 14), (3, 3, 13), (3, 4, 12), (3, 5, 11), (3, 6, 10), (3, 7, 9), (3, 8, 8), (3, 9, 7), (3, 10, 6), (3, 11, 5), (3, 12, 4), (3, 13, 3), (3, 14, 2), (3, 15, 1), (3, 16, 0)

(4, 0, 15), (4, 1, 14), (4, 2, 13), (4, 3, 12), (4, 4, 11), (4, 5, 10), (4, 6, 9), (4, 7, 8), (4, 8, 7), (4, 9, 6), (4, 10, 5), (4, 11, 4), (4, 12, 3), (4, 13, 2), (4, 14, 1), (4, 15, 0)

(5, 0, 14), (5, 1, 13),(5, 2, 12), (5, 3, 11), (5, 4, 10), (5, 5, 9), (5, 6, 8), (5, 7, 7), (5, 8, 6), (5, 9, 5), (5, 10, 4), (5, 11, 3), (5, 12, 2), (5, 13, 1), (5, 14, 0)

(6, 0, 13), (6, 1, 12), (6, 2, 11), (6, 3, 10), (6, 4, 9), (6, 5, 8), (6, 6, 7), (6, 7, 6), (6, 8, 5), (6, 9, 4), (6, 10, 3), (6, 11, 2), (6, 12, 1), (6, 13, 0)

(7, 0, 12), (7, 1, 11), (7, 2, 10), (7, 3, 9), (7, 4, 8), (7, 5, 7), (7, 6, 6), (7, 7, 5), (7, 8, 4), (7, 9, 3), (7, 10, 2), (7, 11, 1), (7, 12, 0)

(8, 0, 11), (8, 1, 10), (8, 2, 9), (8, 3, 8), (8, 4, 7), (8, 5, 6), (8, 6, 5), (8, 7, 4), (8, 8, 3), (8, 9, 2), (8, 10, 1), (8, 11, 0)

(9, 0, 10), (9, 1, 9), (9, 2, 8), (9, 3, 7), (9, 4, 6), (9, 5, 5), (9, 6, 4), (9, 7, 3), (9, 8, 2), (9, 9, 1), (9, 10, 0)

(10, 0, 9), (10, 1, 8), (10, 2, 7), (10, 3, 6), (10, 4, 5), (10, 5, 4), (10, 6, 3), (10, 7, 2), (10, 8, 1), (10, 9, 0)

(11, 0, 8), (11, 1, 7), (11, 2, 6), (11, 3, 5), (11, 4, 4), (11, 5, 3), (11, 6, 2), (11, 7, 1), (11, 8, 0)

(12, 0, 7), (12, 1, 6), (12, 2, 5), (12, 3, 4), (12, 4, 3), (12, 5, 2), (12, 6, 1), (12, 7, 0)

(13, 0, 6), (13, 1, 5), (13, 2, 4), (13, 3, 3), (13, 4, 2), (13, 5, 1), (13, 6, 0)

(14, 0, 5), (14, 1, 4), (14, 2, 3), (14, 3, 2), (14, 4, 1), (14, 5, 0)

(15, 0, 4), (15, 1, 3), (15, 2, 2), (15, 3, 1), (15, 4, 0)

(16, 0, 3), (16, 1, 2), (16, 2, 1), (16, 3, 0)

(17, 0, 2), (17, 1, 1), (17, 2, 0), (18, 0, 1), (18, 1, 0)

(19, 0, 0)]

Sum = 20: 231 combinations

(0, 0, 20), (0, 1, 19), (0, 2, 18), (0, 3, 17), (0, 4, 16), (0, 5, 15), (0, 6, 14), (0, 7, 13), (0, 8, 12), (0, 9, 11), (0, 10, 10), (0, 11, 9), (0, 12, 8), (0, 13, 7), (0, 14, 6), (0, 15, 5), (0, 16, 4), (0, 17, 3), (0, 18, 2), (0, 19, 1), (0, 20, 0)

(1, 0, 19), (1, 1, 18), (1, 2, 17), (1, 3, 16), (1, 4, 15), (1, 5, 14), (1, 6, 13), (1, 7, 12), (1, 8, 11), (1, 9, 10), (1, 10, 9), (1, 11, 8), (1, 12, 7), (1, 13, 6), (1, 14, 5), (1, 15, 4), (1, 16, 3), (1, 17, 2), (1, 18, 1), (1, 19, 0)

(2, 0, 18), (2, 1, 17), (2, 2, 16), (2, 3, 15), (2, 4, 14), (2, 5, 13), (2, 6, 12), (2, 7, 11), (2, 8, 10), (2, 9, 9), (2, 10, 8), (2, 11, 7), (2, 12, 6), (2, 13, 5), (2, 14, 4), (2, 15, 3), (2, 16, 2), (2, 17, 1), (2, 18, 0)

(3, 0, 17), (3, 1, 16), (3, 2, 15), (3, 3, 14), (3, 4, 13), (3, 5, 12), (3, 6, 11), (3, 7, 10), (3, 8, 9), (3, 9, 8), (3, 10, 7), (3, 11, 6), (3, 12, 5), (3, 13, 4), (3, 14, 3), (3, 15, 2), (3, 16, 1), (3, 17, 0),

(4, 0, 16), (4, 1, 15), (4, 2, 14), (4, 3, 13), (4, 4, 12), (4, 5, 11), (4, 6, 10), (4, 7, 9), (4, 8, 8), (4, 9, 7), (4, 10, 6), (4, 11, 5), (4, 12, 4), (4, 13, 3), (4, 14, 2), (4, 15, 1), (4, 16, 0)

(5, 0, 15), (5, 1, 14), (5, 2, 13), (5, 3, 12), (5, 4, 11), (5, 5, 10), (5, 6, 9), (5, 7, 8), (5, 8, 7), (5, 9, 6), (5, 10, 5), (5, 11, 4), (5, 12, 3), (5, 13, 2), (5, 14, 1), (5, 15, 0)

(6, 0, 14), (6, 1, 13), (6, 2, 12), (6, 3, 11), (6, 4, 10), (6, 5, 9), (6, 6, 8), (6, 7, 7), (6, 8, 6), (6, 9, 5), (6, 10, 4), (6, 11, 3), (6, 12, 2), (6, 13, 1), (6, 14, 0)

(7, 0, 13), (7, 1, 12), (7, 2, 11), (7, 3, 10), (7, 4, 9), (7, 5, 8), (7, 6, 7), (7, 7, 6), (7, 8, 5), (7, 9, 4), (7, 10, 3), (7, 11, 2), (7, 12, 1), (7, 13, 0)

(8, 0, 12), (8, 1, 11), (8, 2, 10), (8, 3, 9), (8, 4, 8), (8, 5, 7), (8, 6, 6), (8, 7, 5), (8, 8, 4), (8, 9, 3), (8, 10, 2), (8, 11, 1), (8, 12, 0)

(9, 0, 11), (9, 1, 10), (9, 2, 9), (9, 3, 8), (9, 4, 7), (9, 5, 6), (9, 6, 5), (9, 7, 4), (9, 8, 3), (9, 9, 2), (9, 10, 1), (9, 11, 0)

(10, 0, 10), (10, 1, 9), (10, 2, 8), (10, 3, 7), (10, 4, 6), (10, 5, 5), (10, 6, 4), (10, 7, 3), (10, 8, 2), (10, 9, 1), (10, 10, 0)

(11, 0, 9), (11, 1, 8), (11, 2, 7), (11, 3, 6), (11, 4, 5), (11, 5, 4), (11, 6, 3), (11, 7, 2), (11, 8, 1), (11, 9, 0)

(12, 0, 8), (12, 1, 7), (12, 2, 6), (12, 3, 5), (12, 4, 4), (12, 5, 3), (12, 6, 2), (12, 7, 1), (12, 8, 0)

(13, 0, 7), (13, 1, 6), (13, 2, 5), (13, 3, 4), (13, 4, 3), (13, 5, 2), (13, 6, 1), (13, 7, 0)

(14, 0, 6), (14, 1, 5), (14, 2, 4), (14, 3, 3), (14, 4, 2), (14, 5, 1), (14, 6, 0)

(15, 0, 5), (15, 1, 4), (15, 2, 3), (15, 3, 2), (15, 4, 1), (15, 5, 0)

(16, 0, 4), (16, 1, 3), (16, 2, 2), (16, 3, 1), (16, 4, 0)

(17, 0, 3), (17, 1, 2), (17, 2, 1), (17, 3, 0), (18, 0, 2), (18, 1, 1), (18, 2, 0)

(19, 0, 1), (19, 1, 0)

(20, 0, 0)]

Sum = 21: 253 combinations

[(0, 0, 21), (0, 1, 20), (0, 2, 19), (0, 3, 18), (0, 4, 17), (0, 5, 16), (0, 6, 15), (0, 7, 14), (0, 8, 13), (0, 9, 12), (0, 10, 11), (0, 11, 10), (0, 12, 9), (0, 13, 8), (0, 14, 7), (0, 15, 6), (0, 16, 5), (0, 17, 4), (0, 18, 3), (0, 19, 2), (0, 20, 1), (0, 21, 0)

(1, 0, 20), (1, 1, 19), (1, 2, 18), (1, 3, 17), (1, 4, 16), (1, 5, 15), (1, 6, 14), (1, 7, 13), (1, 8, 12), (1, 9, 11), (1, 10, 10), (1, 11, 9), (1, 12, 8), (1, 13, 7), (1, 14, 6), (1, 15, 5), (1, 16, 4), (1, 17, 3), (1, 18, 2), (1, 19, 1), (1, 20, 0)

(2, 0, 19), (2, 1, 18), (2, 2, 17), (2, 3, 16), (2, 4, 15), (2, 5, 14), (2, 6, 13), (2, 7, 12), (2, 8, 11), (2, 9, 10), (2, 10, 9), (2, 11, 8), (2, 12, 7), (2, 13, 6), (2, 14, 5), (2, 15, 4), (2, 16, 3), (2, 17, 2), (2, 18, 1), (2, 19, 0)

(3, 0, 18), (3, 1, 17), (3, 2, 16), (3, 3, 15), (3, 4, 14), (3, 5, 13), (3, 6, 12), (3, 7, 11), (3, 8, 10), (3, 9, 9), (3, 10, 8), (3, 11, 7), (3, 12, 6), (3, 13, 5), (3, 14, 4), (3, 15, 3), (3, 16, 2), (3, 17, 1), (3, 18, 0)

(4, 0, 17), (4, 1, 16), (4, 2, 15), (4, 3, 14), (4, 4, 13), (4, 5, 12), (4, 6, 11), (4, 7, 10), (4, 8, 9), (4, 9, 8), (4, 10, 7), (4, 11, 6), (4, 12, 5), (4, 13, 4), (4, 14, 3), (4, 15, 2), (4, 16, 1), (4, 17, 0),

(5, 0, 16), (5, 1, 15), (5, 2, 14), (5, 3, 13), (5, 4, 12), (5, 5, 11), (5, 6, 10), (5, 7, 9), (5, 8, 8), (5, 9, 7), (5, 10, 6), (5, 11, 5), (5, 12, 4), (5, 13, 3), (5, 14, 2), (5, 15, 1), (5, 16, 0)

(6, 0, 15), (6, 1, 14), (6, 2, 13), (6, 3, 12), (6, 4, 11), (6, 5, 10), (6, 6, 9), (6, 7, 8), (6, 8, 7), (6, 9, 6), (6, 10, 5), (6, 11, 4), (6, 12, 3), (6, 13, 2), (6, 14, 1), (6, 15, 0)

(7, 0, 14), (7, 1, 13), (7, 2, 12), (7, 3, 11), (7, 4, 10), (7, 5, 9), (7, 6, 8), (7, 7, 7), (7, 8, 6), (7, 9, 5), (7, 10, 4), (7, 11, 3), (7, 12, 2), (7, 13, 1), (7, 14, 0)

(8, 0, 13), (8, 1, 12), (8, 2, 11), (8, 3, 10), (8, 4, 9), (8, 5, 8), (8, 6, 7), (8, 7, 6), (8, 8, 5), (8, 9, 4), (8, 10, 3), (8, 11, 2), (8, 12, 1), (8, 13, 0)

(9, 0, 12), (9, 1, 11), (9, 2, 10), (9, 3, 9), (9, 4, 8), (9, 5, 7), (9, 6, 6), (9, 7, 5), (9, 8, 4), (9, 9, 3), (9, 10, 2), (9, 11, 1), (9, 12, 0)

(10, 0, 11), (10, 1, 10), (10, 2, 9), (10, 3, 8), (10, 4, 7), (10, 5, 6), (10, 6, 5), (10, 7, 4), (10, 8, 3), (10, 9, 2), (10, 10, 1), (10, 11, 0)

(11, 0, 10), (11, 1, 9), (11, 2, 8), (11, 3, 7), (11, 4, 6), (11, 5, 5), (11, 6, 4), (11, 7, 3), (11, 8, 2), (11, 9, 1), (11, 10, 0)

(12, 0, 9), (12, 1, 8), (12, 2, 7), (12, 3, 6), (12, 4, 5), (12, 5, 4), (12, 6, 3), (12, 7, 2), (12, 8, 1), (12, 9, 0)

(13, 0, 8), (13, 1, 7), (13, 2, 6), (13, 3, 5), (13, 4, 4), (13, 5, 3), (13, 6, 2), (13, 7, 1), (13, 8, 0)

(14, 0, 7), (14, 1, 6), (14, 2, 5), (14, 3, 4), (14, 4, 3), (14, 5, 2), (14, 6, 1), (14, 7, 0)

(15, 0, 6), (15, 1, 5), (15, 2, 4), (15, 3, 3), (15, 4, 2), (15, 5, 1), (15, 6, 0)

(16, 0, 5), (16, 1, 4), (16, 2, 3), (16, 3, 2), (16, 4, 1), (16, 5, 0)

(17, 0, 4), (17, 1, 3), (17, 2, 2), (17, 3, 1), (17, 4, 0)

(18, 0, 3), (18, 1, 2), (18, 2, 1), (18, 3, 0)

(19, 0, 2), (19, 1, 1), (19, 2, 0)

(20, 0, 1), (20, 1, 0)

(21, 0, 0)

Sum = 22: 276 combinations

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

(1, 0, 21), (1, 1, 20), (1, 2, 19), (1, 3, 18), (1, 4, 17), (1, 5, 16), (1, 6, 15), (1, 7, 14), (1, 8, 13), (1, 9, 12), (1, 10, 11), (1, 11, 10), (1, 12, 9), (1, 13, 8), (1, 14, 7), (1, 15, 6), (1, 16, 5), (1, 17, 4), (1, 18, 3), (1, 19, 2), (1, 20, 1), (1, 21, 0)

(2, 0, 20), (2, 1, 19), (2, 2, 18), (2, 3, 17), (2, 4, 16), (2, 5, 15), (2, 6, 14), (2, 7, 13), (2, 8, 12), (2, 9, 11), (2, 10, 10), (2, 11, 9), (2, 12, 8), (2, 13, 7), (2, 14, 6), (2, 15, 5), (2, 16, 4), (2, 17, 3), (2, 18, 2), (2, 19, 1), (2, 20, 0)

(3, 0, 19), (3, 1, 18), (3, 2, 17), (3, 3, 16), (3, 4, 15), (3, 5, 14), (3, 6, 13), (3, 7, 12), (3, 8, 11), (3, 9, 10), (3, 10, 9), (3, 11, 8), (3, 12, 7), (3, 13, 6), (3, 14, 5), (3, 15, 4), (3, 16, 3), (3, 17, 2), (3, 18, 1), (3, 19, 0)

(4, 0, 18), (4, 1, 17), (4, 2, 16), (4, 3, 15), (4, 4, 14), (4, 5, 13), (4, 6, 12), (4, 7, 11), (4, 8, 10), (4, 9, 9), (4, 10, 8), (4, 11, 7), (4, 12, 6), (4, 13, 5), (4, 14, 4), (4, 15, 3), (4, 16, 2), (4, 17, 1), (4, 18, 0)

(5, 0, 17), (5, 1, 16), (5, 2, 15), (5, 3, 14), (5, 4, 13), (5, 5, 12), (5, 6, 11), (5, 7, 10), (5, 8, 9), (5, 9, 8), (5, 10, 7), (5, 11, 6), (5, 12, 5), (5, 13, 4), (5, 14, 3), (5, 15, 2), (5, 16, 1), (5, 17, 0)

(6, 0, 16), (6, 1, 15), (6, 2, 14), (6, 3, 13), (6, 4, 12), (6, 5, 11), (6, 6, 10), (6, 7, 9), (6, 8, 8), (6, 9, 7), (6, 10, 6), (6, 11, 5), (6, 12, 4), (6, 13, 3), (6, 14, 2), (6, 15, 1), (6, 16, 0)

(7, 0, 15), (7, 1, 14), (7, 2, 13), (7, 3, 12), (7, 4, 11), (7, 5, 10), (7, 6, 9), (7, 7, 8), (7, 8, 7), (7, 9, 6), (7, 10, 5), (7, 11, 4), (7, 12, 3), (7, 13, 2), (7, 14, 1), (7, 15, 0)

(8, 0, 14), (8, 1, 13), (8, 2, 12), (8, 3, 11), (8, 4, 10), (8, 5, 9), (8, 6, 8), (8, 7, 7), (8, 8, 6), (8, 9, 5), (8, 10, 4), (8, 11, 3), (8, 12, 2), (8, 13, 1), (8, 14, 0)

(9, 0, 13), (9, 1, 12), (9, 2, 11), (9, 3, 10), (9, 4, 9), (9, 5, 8), (9, 6, 7), (9, 7, 6), (9, 8, 5), (9, 9, 4), (9, 10, 3), (9, 11, 2), (9, 12, 1), (9, 13, 0)

(10, 0, 12), (10, 1, 11), (10, 2, 10), (10, 3, 9), (10, 4, 8), (10, 5, 7), (10, 6, 6), (10, 7, 5), (10, 8, 4), (10, 9, 3), (10, 10, 2), (10, 11, 1), (10, 12, 0)

(11, 0, 11), (11, 1, 10), (11, 2, 9), (11, 3, 8), (11, 4, 7), (11, 5, 6), (11, 6, 5), (11, 7, 4), (11, 8, 3), (11, 9, 2), (11, 10, 1), (11, 11, 0)

(12, 0, 10), (12, 1, 9), (12, 2, 8), (12, 3, 7), (12, 4, 6), (12, 5, 5), (12, 6, 4), (12, 7, 3), (12, 8, 2), (12, 9, 1), (12, 10, 0)

(13, 0, 9), (13, 1, 8), (13, 2, 7), (13, 3, 6), (13, 4, 5), (13, 5, 4), (13, 6, 3), (13, 7, 2), (13, 8, 1), (13, 9, 0)

(14, 0, 8), (14, 1, 7), (14, 2, 6), (14, 3, 5), (14, 4, 4), (14, 5, 3), (14, 6, 2), (14, 7, 1), (14, 8, 0)

(15, 0, 7), (15, 1, 6), (15, 2, 5), (15, 3, 4), (15, 4, 3), (15, 5, 2), (15, 6, 1), (15, 7, 0)

(16, 0, 6), (16, 1, 5), (16, 2, 4), (16, 3, 3), (16, 4, 2), (16, 5, 1), (16, 6, 0)

(17, 0, 5), (17, 1, 4), (17, 2, 3), (17, 3, 2), (17, 4, 1), (17, 5, 0)

(18, 0, 4), (18, 1, 3), (18, 2, 2), (18, 3, 1), (18, 4, 0)

(19, 0, 3), (19, 1, 2), (19, 2, 1), (19, 3, 0)

(20, 0, 2), (20, 1, 1), (20, 2, 0)

(21, 0, 1), (21, 1, 0)

(22, 0, 0)]

Sum = 23: 300 combinations

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

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

(2, 0, 21), (2, 1, 20), (2, 2, 19), (2, 3, 18), (2, 4, 17), (2, 5, 16), (2, 6, 15), (2, 7, 14), (2, 8, 13), (2, 9, 12), (2, 10, 11), (2, 11, 10), (2, 12, 9), (2, 13, 8), (2, 14, 7), (2, 15, 6), (2, 16, 5), (2, 17, 4), (2, 18, 3), (2, 19, 2), (2, 20, 1), (2, 21, 0)

(3, 0, 20), (3, 1, 19), (3, 2, 18), (3, 3, 17), (3, 4, 16), (3, 5, 15), (3, 6, 14), (3, 7, 13), (3, 8, 12), (3, 9, 11), (3, 10, 10), (3, 11, 9), (3, 12, 8), (3, 13, 7), (3, 14, 6), (3, 15, 5), (3, 16, 4), (3, 17, 3), (3, 18, 2), (3, 19, 1), (3, 20, 0)

(4, 0, 19), (4, 1, 18), (4, 2, 17), (4, 3, 16), (4, 4, 15), (4, 5, 14), (4, 6, 13), (4, 7, 12), (4, 8, 11), (4, 9, 10), (4, 10, 9), (4, 11, 8), (4, 12, 7), (4, 13, 6), (4, 14, 5), (4, 15, 4), (4, 16, 3), (4, 17, 2), (4, 18, 1), (4, 19, 0)

(5, 0, 18), (5, 1, 17), (5, 2, 16), (5, 3, 15), (5, 4, 14), (5, 5, 13), (5, 6, 12), (5, 7, 11), (5, 8, 10), (5, 9, 9), (5, 10, 8), (5, 11, 7), (5, 12, 6), (5, 13, 5), (5, 14, 4), (5, 15, 3), (5, 16, 2), (5, 17, 1), (5, 18, 0)

(6, 0, 17), (6, 1, 16), (6, 2, 15), (6, 3, 14), (6, 4, 13), (6, 5, 12), (6, 6, 11), (6, 7, 10), (6, 8, 9), (6, 9, 8), (6, 10, 7), (6, 11, 6), (6, 12, 5), (6, 13, 4), (6, 14, 3), (6, 15, 2), (6, 16, 1), (6, 17, 0)

(7, 0, 16), (7, 1, 15), (7, 2, 14), (7, 3, 13), (7, 4, 12), (7, 5, 11), (7, 6, 10), (7, 7, 9), (7, 8, 8), (7, 9, 7), (7, 10, 6), (7, 11, 5), (7, 12, 4), (7, 13, 3), (7, 14, 2), (7, 15, 1), (7, 16, 0)

(8, 0, 15), (8, 1, 14), (8, 2, 13), (8, 3, 12), (8, 4, 11), (8, 5, 10), (8, 6, 9), (8, 7, 8), (8, 8, 7), (8, 9, 6), (8, 10, 5), (8, 11, 4), (8, 12, 3), (8, 13, 2), (8, 14, 1), (8, 15, 0)

(9, 0, 14), (9, 1, 13), (9, 2, 12), (9, 3, 11), (9, 4, 10), (9, 5, 9), (9, 6, 8), (9, 7, 7), (9, 8, 6), (9, 9, 5), (9, 10, 4), (9, 11, 3), (9, 12, 2), (9, 13, 1), (9, 14, 0)

(10, 0, 13), (10, 1, 12), (10, 2, 11), (10, 3, 10), (10, 4, 9), (10, 5, 8), (10, 6, 7), (10, 7, 6), (10, 8, 5), (10, 9, 4), (10, 10, 3), (10, 11, 2), (10, 12, 1), (10, 13, 0)

(11, 0, 12), (11, 1, 11), (11, 2, 10), (11, 3, 9), (11, 4, 8), (11, 5, 7), (11, 6, 6), (11, 7, 5), (11, 8, 4), (11, 9, 3), (11, 10, 2), (11, 11, 1), (11, 12, 0)

(12, 0, 11), (12, 1, 10), (12, 2, 9), (12, 3, 8), (12, 4, 7), (12, 5, 6), (12, 6, 5), (12, 7, 4), (12, 8, 3), (12, 9, 2), (12, 10, 1), (12, 11, 0)

(13, 0, 10), (13, 1, 9), (13, 2, 8), (13, 3, 7), (13, 4, 6), (13, 5, 5), (13, 6, 4), (13, 7, 3), (13, 8, 2), (13, 9, 1), (13, 10, 0)

(14, 0, 9), (14, 1, 8), (14, 2, 7), (14, 3, 6), (14, 4, 5), (14, 5, 4), (14, 6, 3), (14, 7, 2), (14, 8, 1), (14, 9, 0)

(15, 0, 8), (15, 1, 7), (15, 2, 6), (15, 3, 5), (15, 4, 4), (15, 5, 3), (15, 6, 2), (15, 7, 1), (15, 8, 0)

(16, 0, 7), (16, 1, 6), (16, 2, 5), (16, 3, 4), (16, 4, 3), (16, 5, 2), (16, 6, 1), (16, 7, 0)

(17, 0, 6), (17, 1, 5), (17, 2, 4), (17, 3, 3), (17, 4, 2), (17, 5, 1), (17, 6, 0)

(18, 0, 5), (18, 1, 4), (18, 2, 3), (18, 3, 2), (18, 4, 1), (18, 5, 0)

(19, 0, 4), (19, 1, 3), (19, 2, 2), (19, 3, 1), (19, 4, 0)

(20, 0, 3), (20, 1, 2), (20, 2, 1), (20, 3, 0)

(21, 0, 2), (21, 1, 1), (21, 2, 0)

(22, 0, 1), (22, 1, 0)

(23, 0, 0)

Sum = 24: 325 combinations

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

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

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

(3, 0, 21), (3, 1, 20), (3, 2, 19), (3, 3, 18), (3, 4, 17), (3, 5, 16), (3, 6, 15), (3, 7, 14), (3, 8, 13), (3, 9, 12), (3, 10, 11), (3, 11, 10), (3, 12, 9), (3, 13, 8), (3, 14, 7), (3, 15, 6), (3, 16, 5), (3, 17, 4), (3, 18, 3), (3, 19, 2), (3, 20, 1), (3, 21, 0)

(4, 0, 20), (4, 1, 19), (4, 2, 18), (4, 3, 17), (4, 4, 16), (4, 5, 15), (4, 6, 14), (4, 7, 13), (4, 8, 12), (4, 9, 11), (4, 10, 10), (4, 11, 9), (4, 12, 8), (4, 13, 7), (4, 14, 6), (4, 15, 5), (4, 16, 4), (4, 17, 3), (4, 18, 2), (4, 19, 1), (4, 20, 0)

(5, 0, 19), (5, 1, 18), (5, 2, 17), (5, 3, 16), (5, 4, 15), (5, 5, 14), (5, 6, 13), (5, 7, 12), (5, 8, 11), (5, 9, 10), (5, 10, 9), (5, 11, 8), (5, 12, 7), (5, 13, 6), (5, 14, 5), (5, 15, 4), (5, 16, 3), (5, 17, 2), (5, 18, 1), (5, 19, 0)

(6, 0, 18), (6, 1, 17), (6, 2, 16), (6, 3, 15), (6, 4, 14), (6, 5, 13), (6, 6, 12), (6, 7, 11), (6, 8, 10), (6, 9, 9), (6, 10, 8), (6, 11, 7), (6, 12, 6), (6, 13, 5), (6, 14, 4), (6, 15, 3), (6, 16, 2), (6, 17, 1), (6, 18, 0)

(7, 0, 17), (7, 1, 16), (7, 2, 15), (7, 3, 14), (7, 4, 13), (7, 5, 12), (7, 6, 11), (7, 7, 10), (7, 8, 9), (7, 9, 8), (7, 10, 7), (7, 11, 6), (7, 12, 5), (7, 13, 4), (7, 14, 3), (7, 15, 2), (7, 16, 1), (7, 17, 0),

(8, 0, 16), (8, 1, 15), (8, 2, 14), (8, 3, 13), (8, 4, 12), (8, 5, 11), (8, 6, 10), (8, 7, 9), (8, 8, 8), (8, 9, 7), (8, 10, 6), (8, 11, 5), (8, 12, 4), (8, 13, 3), (8, 14, 2), (8, 15, 1), (8, 16, 0)

(9, 0, 15), (9, 1, 14), (9, 2, 13), (9, 3, 12), (9, 4, 11), (9, 5, 10), (9, 6, 9), (9, 7, 8), (9, 8, 7), (9, 9, 6), (9, 10, 5), (9, 11, 4), (9, 12, 3), (9, 13, 2), (9, 14, 1), (9, 15, 0)

(10, 0, 14), (10, 1, 13), (10, 2, 12), (10, 3, 11), (10, 4, 10), (10, 5, 9), (10, 6, 8), (10, 7, 7), (10, 8, 6), (10, 9, 5), (10, 10, 4), (10, 11, 3), (10, 12, 2), (10, 13, 1), (10, 14, 0)

(11, 0, 13), (11, 1, 12), (11, 2, 11), (11, 3, 10), (11, 4, 9), (11, 5, 8), (11, 6, 7), (11, 7, 6), (11, 8, 5), (11, 9, 4), (11, 10, 3), (11, 11, 2), (11, 12, 1), (11, 13, 0)

(12, 0, 12), (12, 1, 11), (12, 2, 10), (12, 3, 9), (12, 4, 8), (12, 5, 7), (12, 6, 6), (12, 7, 5), (12, 8, 4), (12, 9, 3), (12, 10, 2), (12, 11, 1), (12, 12, 0)

(13, 0, 11), (13, 1, 10), (13, 2, 9), (13, 3, 8), (13, 4, 7), (13, 5, 6), (13, 6, 5), (13, 7, 4), (13, 8, 3), (13, 9, 2), (13, 10, 1), (13, 11, 0)

(14, 0, 10), (14, 1, 9), (14, 2, 8), (14, 3, 7), (14, 4, 6), (14, 5, 5), (14, 6, 4), (14, 7, 3), (14, 8, 2), (14, 9, 1), (14, 10, 0)

(15, 0, 9), (15, 1, 8), (15, 2, 7), (15, 3, 6), (15, 4, 5), (15, 5, 4), (15, 6, 3), (15, 7, 2), (15, 8, 1), (15, 9, 0)

(16, 0, 8), (16, 1, 7), (16, 2, 6), (16, 3, 5), (16, 4, 4), (16, 5, 3), (16, 6, 2), (16, 7, 1), (16, 8, 0),

(17, 0, 7), (17, 1, 6), (17, 2, 5), (17, 3, 4), (17, 4, 3), (17, 5, 2), (17, 6, 1), (17, 7, 0)

(18, 0, 6), (18, 1, 5), (18, 2, 4), (18, 3, 3), (18, 4, 2), (18, 5, 1), (18, 6, 0)

(19, 0, 5), (19, 1, 4), (19, 2, 3), (19, 3, 2), (19, 4, 1), (19, 5, 0)

(20, 0, 4), (20, 1, 3), (20, 2, 2), (20, 3, 1), (20, 4, 0)

(21, 0, 3), (21, 1, 2), (21, 2, 1), (21, 3, 0)

(22, 0, 2), (22, 1, 1), (22, 2, 0)

(23, 0, 1), (23, 1, 0)

(24, 0, 0)]

Sum = 25: 351 combinations

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

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

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

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

(4, 0, 21), (4, 1, 20), (4, 2, 19), (4, 3, 18), (4, 4, 17), (4, 5, 16), (4, 6, 15), (4, 7, 14), (4, 8, 13), (4, 9, 12), (4, 10, 11), (4, 11, 10), (4, 12, 9), (4, 13, 8), (4, 14, 7), (4, 15, 6), (4, 16, 5), (4, 17, 4), (4, 18, 3), (4, 19, 2), (4, 20, 1), (4, 21, 0)

(5, 0, 20), (5, 1, 19), (5, 2, 18), (5, 3, 17), (5, 4, 16), (5, 5, 15), (5, 6, 14), (5, 7, 13), (5, 8, 12), (5, 9, 11), (5, 10, 10), (5, 11, 9), (5, 12, 8), (5, 13, 7), (5, 14, 6), (5, 15, 5), (5, 16, 4), (5, 17, 3), (5, 18, 2), (5, 19, 1), (5, 20, 0)

(6, 0, 19), (6, 1, 18), (6, 2, 17), (6, 3, 16), (6, 4, 15), (6, 5, 14), (6, 6, 13), (6, 7, 12), (6, 8, 11), (6, 9, 10), (6, 10, 9), (6, 11, 8), (6, 12, 7), (6, 13, 6), (6, 14, 5), (6, 15, 4), (6, 16, 3), (6, 17, 2), (6, 18, 1), (6, 19, 0)

(7, 0, 18), (7, 1, 17), (7, 2, 16), (7, 3, 15), (7, 4, 14), (7, 5, 13), (7, 6, 12), (7, 7, 11), (7, 8, 10), (7, 9, 9), (7, 10, 8), (7, 11, 7), (7, 12, 6), (7, 13, 5), (7, 14, 4), (7, 15, 3), (7, 16, 2), (7, 17, 1), (7, 18, 0)

(8, 0, 17), (8, 1, 16), (8, 2, 15), (8, 3, 14), (8, 4, 13), (8, 5, 12), (8, 6, 11), (8, 7, 10), (8, 8, 9), (8, 9, 8), (8, 10, 7), (8, 11, 6), (8, 12, 5), (8, 13, 4), (8, 14, 3), (8, 15, 2), (8, 16, 1), (8, 17, 0)

(9, 0, 16), (9, 1, 15), (9, 2, 14), (9, 3, 13), (9, 4, 12), (9, 5, 11), (9, 6, 10), (9, 7, 9), (9, 8, 8), (9, 9, 7), (9, 10, 6), (9, 11, 5), (9, 12, 4), (9, 13, 3), (9, 14, 2), (9, 15, 1), (9, 16, 0)

(10, 0, 15), (10, 1, 14), (10, 2, 13), (10, 3, 12), (10, 4, 11), (10, 5, 10), (10, 6, 9), (10, 7, 8), (10, 8, 7), (10, 9, 6), (10, 10, 5), (10, 11, 4), (10, 12, 3), (10, 13, 2), (10, 14, 1), (10, 15, 0)

(11, 0, 14), (11, 1, 13), (11, 2, 12), (11, 3, 11), (11, 4, 10), (11, 5, 9), (11, 6, 8), (11, 7, 7), (11, 8, 6), (11, 9, 5), (11, 10, 4), (11, 11, 3), (11, 12, 2), (11, 13, 1), (11, 14, 0)

(12, 0, 13), (12, 1, 12), (12, 2, 11), (12, 3, 10), (12, 4, 9), (12, 5, 8), (12, 6, 7), (12, 7, 6), (12, 8, 5), (12, 9, 4), (12, 10, 3), (12, 11, 2), (12, 12, 1), (12, 13, 0)

(13, 0, 12), (13, 1, 11), (13, 2, 10), (13, 3, 9), (13, 4, 8), (13, 5, 7), (13, 6, 6), (13, 7, 5), (13, 8, 4), (13, 9, 3), (13, 10, 2), (13, 11, 1), (13, 12, 0)

(14, 0, 11), (14, 1, 10), (14, 2, 9), (14, 3, 8), (14, 4, 7), (14, 5, 6), (14, 6, 5), (14, 7, 4), (14, 8, 3), (14, 9, 2), (14, 10, 1), (14, 11, 0)

(15, 0, 10), (15, 1, 9), (15, 2, 8), (15, 3, 7), (15, 4, 6), (15, 5, 5), (15, 6, 4), (15, 7, 3), (15, 8, 2), (15, 9, 1), (15, 10, 0)

(16, 0, 9), (16, 1, 8), (16, 2, 7), (16, 3, 6), (16, 4, 5), (16, 5, 4), (16, 6, 3), (16, 7, 2), (16, 8, 1), (16, 9, 0)

(17, 0, 8), (17, 1, 7), (17, 2, 6), (17, 3, 5), (17, 4, 4), (17, 5, 3), (17, 6, 2), (17, 7, 1), (17, 8, 0)

(18, 0, 7), (18, 1, 6), (18, 2, 5), (18, 3, 4), (18, 4, 3), (18, 5, 2), (18, 6, 1), (18, 7, 0)

(19, 0, 6), (19, 1, 5), (19, 2, 4), (19, 3, 3), (19, 4, 2), (19, 5, 1), (19, 6, 0)

(20, 0, 5), (20, 1, 4), (20, 2, 3), (20, 3, 2), (20, 4, 1), (20, 5, 0)

(21, 0, 4), (21, 1, 3), (21, 2, 2), (21, 3, 1), (21, 4, 0)

(22, 0, 3), (22, 1, 2), (22, 2, 1), (22, 3, 0)

(23, 0, 2), (23, 1, 1), (23, 2, 0)

(24, 0, 1), (24, 1, 0)

(25, 0, 0)]

Formula to determine the number of combination :

$$Number\_of\_combination = \frac{[\mathbf{Total\_(string\_list)} + \mathbf{1}] \times [\mathbf{Total\_(string\_list)} + \mathbf{1}] + \mathbf{1}]}{2}$$

We add +1 to **Total_(string_list)** because we need to consider “0”

Conclusion of this part :

We shouldn’t base the state space graph based on all the different numerical string list possible because the number of combinations is too high and will be expensive in costs and energy.

We need to find a way to define all the possible root states based on something simpler.

Questions :

How does the number of 1, 2 and 3 impacts the game ?

2. Data Structures for Game Tree:

String/List (self.string):

Description: This list represents the current state of the game or the game tree.

Contents: Each element in the list corresponds to a number in the game. The numbers represent the current game state.
Usage: The list is updated as moves are made during the game. Elements are removed or modified to reflect the current game state.
Child Nodes (Dynamic):

Description: Child nodes are dynamically generated during the game to represent possible future game states.
Contents: Each child node is stored as a list, similar to self.string, representing a possible move or action that can be taken from the current game state.
Generation: Child nodes are generated based on the current game state whenever a move is made.
Usage: These child nodes are used for evaluating possible future states and making decisions, such as the computer's next move.
These data structures effectively represent the game tree, allowing for manipulation and evaluation of different game states during gameplay. The primary data structure, the list (self.string), serves as the backbone of the game tree, while child nodes dynamically generated during gameplay represent possible future states.

3. The Heuristic function:

Description:

The heuristic evaluation function is responsible for evaluating the desirability or quality of a given game state from the perspective of the current player.
It assigns a numerical value to each game state based on certain criteria, providing a rough estimate of how favorable the state is for the current player.

Input Parameters:

node: The game state (represented as a list of numbers) for which the heuristic value is to be computed.
current_player_score: The current player's score in the game.
opponent_score: The opponent's score in the game.

Calculation Logic:

The heuristic function calculates the heuristic value based on the sum of absolute differences between each number in the game state and the opponent's score.
It adjusts the heuristic value based on whether the current player is ahead, behind, or equal to the opponent's score.

Justification:

The heuristic function aims to provide a simple yet effective estimation of the desirability of a game state for the current player.
By considering the difference between each number in the game state and the opponent's score, it captures the immediate impact of the current state on the game outcome.

Prioritizing reducing the difference if the current player is behind and increasing it if the current player is ahead reflects strategic considerations in the game. For example, if the current player is behind, the function might prioritize actions that decrease the opponent's advantage.
The heuristic function provides a quick evaluation of game states, which is essential for decision-making in algorithms such as minimax and alpha-beta pruning.
While the heuristic function in the our code is relatively simple, it effectively captures some aspects of game state evaluation.

4. Main algorithms implemented in the software:

1. Generating a game tree:

```
def get_children(node):
    # Placeholder function for generating child nodes
    children = []
    for i in range(len(node)):
        child = node.copy()
        child.pop(i)
        children.append(child)
    return children
```

This function generates child nodes from a given game state node. In the provided code, it simply removes one number from the current state at a time to create child nodes.

2. Assigning Heuristic Values to Graph Nodes:

```
def heuristic_value(node, current_player_score, opponent_score):
    # Calculate heuristic measures
    total_difference = sum(abs(num - opponent_score) for num in node)
    average_value = sum(node) / len(node) if node else 0
    total_score = current_player_score + opponent_score
    score_difference_ratio = abs(current_player_score - opponent_score) /
total_score if total_score != 0 else 0

    # Print heuristic measures
    print("Heuristic Measures:")
    print("Total Difference:", total_difference)
    print("Average Value:", average_value)
    print("Score Difference Ratio:", score_difference_ratio)

    # Calculate and return heuristic value
    if current_player_score < opponent_score:
        return total_difference - average_value - score_difference_ratio
    elif current_player_score > opponent_score:
        return average_value - total_difference - score_difference_ratio
    else:
        return average_value - total_difference
```

This heuristic function considers factors such as the distance of numbers from the opponent's score, the average value of numbers in the game state, and the relative difference in scores between the current player and the opponent. It aims to guide the decision-making process in selecting optimal moves by evaluating the potential outcomes of different game states.

3. Applying a Game Algorithm (Minimax):

```
def minimax(node, depth, maximizingPlayer, current_player_score,
opponent_score):
    if depth == 0 or game_over(node):
        return heuristic_value(node, current_player_score, opponent_score)
    if maximizingPlayer:
        value = -float('inf')
        for child in get_children(node):
            value = max(value, minimax(child, depth - 1, False,
current_player_score, opponent_score))
        return value
    else:
        value = float('inf')
        for child in get_children(node):
            value = min(value, minimax(child, depth - 1, True,
current_player_score, opponent_score))
        return value
```

The minimax function recursively evaluates the utility of each game state (node) in the game tree.
It takes parameters such as the current game state (node), the current depth in the game tree (depth), a boolean flag indicating whether the current player is maximizing (maximizingPlayer), and the current player's score (current_player_score) and the opponent's score (opponent_score).
If the maximum depth is reached (depth == 0) or the game is over (game_over(node)), it returns the heuristic value of the current game state calculated by the heuristic_value function.
Otherwise, it recursively evaluates the utility of each child node and selects the maximum or minimum value based on whether the current player is maximizing or minimizing, respectively.

4. **Applying a Game Algorithm (Alpha-Beta):**

```
def alphabeta(node, depth, alpha, beta, maximizingPlayer,
current_player_score, opponent_score):
    if depth == 0 or game_over(node):
        return heuristic_value(node, current_player_score, opponent_score)
    if maximizingPlayer:
        value = -float('inf')
        for child in get_children(node):
            value = max(value, alphabeta(child, depth - 1, alpha, beta, False,
current_player_score, opponent_score))
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return value
    else:
        value = float('inf')
        for child in get_children(node):
            value = min(value, alphabeta(child, depth - 1, alpha, beta, True,
current_player_score, opponent_score))
            beta = min(beta, value)
            if beta <= alpha:
```

```
                break
        return value
```

The alphabeta function is similar to the minimax function but includes alpha-beta pruning for efficiency.
It takes additional parameters alpha and beta, representing the best values found for the maximizing and minimizing players, respectively.
During the search, if the current player is maximizing and the current alphabeta value exceeds the beta value, it prunes the subtree below the current node.
Likewise, if the current player is minimizing and the current alphabeta value is less than the alpha value, it prunes the subtree.
By pruning these branches, the algorithm reduces the number of nodes explored, resulting in faster computation while still guaranteeing the optimal move.

5. Comparison of algorithms:

We did 10 Experiments with Minmax Algorithm and Alfa-Beta Pruning
I have added both the Video and Docx file Containing these in our GitHUB repository.

Comparative Analysis
Number of Wins:

- Alpha-Beta Pruning: The computer and human players each won 5 games out of the 10 experiments.
- Minimax: The computer won 8 games out of 11 experiments (note: there seems to be an extra experiment here, but for the sake of comparison, we'll focus on the first 10).

Average Time Per Move:

- Alpha-Beta Pruning demonstrated slightly increasing average times as the number of strings increased, starting from 0.4858 seconds and going up to 0.4953 seconds.
- Minimax showed a similar trend but with a generally higher time taken for each move, starting from 0.4896 seconds to 0.5732 seconds in the last experiment.

Total Nodes Visited:

- For both algorithms, the number of nodes visited increased with the number of strings. However, the Alpha-Beta Pruning algorithm consistently visited fewer nodes compared to Minimax, indicating its higher efficiency in pruning unnecessary branches and thereby reducing the search space.

Conclusions:

1. Efficiency: The Alpha-Beta Pruning algorithm is more efficient than Minimax, as evidenced by the fewer nodes visited during the experiments. This efficiency likely contributes to slightly lower average move times, particularly noticeable as the complexity (number of strings) increases.
2. Effectiveness: The Minimax algorithm led to more computer victories in the experiments. However, this could be influenced by various factors, including the game's starting conditions and inherent differences in the algorithms' decision-making processes.
3. Performance Over Time: As the complexity of the game increases (with more strings), the average time per move increases for both algorithms. This is expected, as

more possibilities need to be evaluated. However, the rate of increase in time taken is less for Alpha-Beta Pruning, showcasing its scalability and better performance under increasing complexity.

Suggested Table for Comparison

To concisely present this comparison, a table format could be useful:

Metric	Alpha-Beta Pruning	Minimax
Computer Wins	5/10	8/10
Human Wins	5/10	2/10
Average Time Per Move (Start)	0.4858s	0.4896s
Average Time Per Move (End)	0.4953s	0.5732s
Nodes Visited (Start)	56	49
Nodes Visited (End)	132	132

Conclusion Remarks

The Alpha-Beta Pruning algorithm demonstrates superior efficiency and scalability with increasing game complexity, making it preferable for games requiring deep search trees. However, the Minimax algorithm showed a higher win rate for the computer in these experiments, suggesting it might sometimes make more effective decisions within the specific context of these games. The choice between these algorithms should consider the specific requirements of efficiency, effectiveness, and performance under varying levels of complexity.

Algorithm	**Description**	**Advantages**	**Disadvantages**

Alpha-Beta Pruning	This is an optimization technique for the Minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available.	It is more efficient than Minimax as it removes all the nodes which are potentially useless.	It is more complex than the Minimax algorithm.
Minimax	This is a backtracking algorithm used for decision making in game theory and artificial intelligence. The algorithm computes the minimax decision from the current state. Minimax decisions result in the maximization of the minimum gain (maximin).	It is simpler and easier to implement.	It can be slow and inefficient as it evaluates all the nodes in the game tree.

6. The software code that corresponds to the implementation of the game excluding the graphical interface:

```
import random
import sys
import time

def heuristic_value(node, current_player_score, opponent_score):
    total_difference = sum(abs(num - opponent_score) for num in node)
    average_value = sum(node) / len(node) if node else 0
    total_score = current_player_score + opponent_score
    score_difference_ratio = abs(current_player_score - opponent_score) / total_score if
total_score != 0 else 0

    if current_player_score < opponent_score:
        return total_difference - average_value - score_difference_ratio
    elif current_player_score > opponent_score:
        return average_value - total_difference - score_difference_ratio
    else:
        return average_value - total_difference

def get_children(node):
    children = []
    for i in range(len(node)):
        child = node.copy()
        child.pop(i)
        children.append(child)
    return children

def game_over(node):
    return len(node) == 0

def minimax(node, depth, maximizingPlayer, current_player_score, opponent_score):
    if depth == 0 or game_over(node):
        return heuristic_value(node, current_player_score, opponent_score)
    if maximizingPlayer:
        value = -float('inf')
        for child in get_children(node):
            value = max(value, minimax(child, depth - 1, False, current_player_score,
opponent_score))
        return value
    else:
        value = float('inf')
        for child in get_children(node):
            value = min(value, minimax(child, depth - 1, True, current_player_score,
opponent_score))
        return value
```

```
def alphabeta(node, depth, alpha, beta, maximizingPlayer, current_player_score,
opponent_score):
    if depth == 0 or game_over(node):
        return heuristic_value(node, current_player_score, opponent_score)
    if maximizingPlayer:
        value = -float('inf')
        for child in get_children(node):
            value = max(value, alphabeta(child, depth - 1, alpha, beta, False,
current_player_score, opponent_score))
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return value
    else:
        value = float('inf')
        for child in get_children(node):
            value = min(value, alphabeta(child, depth - 1, alpha, beta, True, current_player_score,
opponent_score))
            beta = min(beta, value)
            if beta <= alpha:
                break
        return value

class Game:
    def __init__(self, string_length=20, start_player="user", algorithm="minimax"):
        self.string = [random.randint(1, 3) for _ in range(string_length)]
        self.human_score = 80
        self.computer_score = 80
        self.start_player = start_player
        self.algorithm = algorithm
        self.current_player = start_player

    def make_move(self, index):
        num = self.string.pop(index)
        if self.current_player == "user":
            self.human_score -= num
            self.current_player = "computer"
        else:
            self.computer_score -= num
            self.current_player = "user"

    def computer_move(self):
        if self.current_player == "user":
```

```
            current_player_score = self.human_score
            opponent_score = self.computer_score
        else:
            current_player_score = self.computer_score
            opponent_score = self.human_score

        if self.algorithm == "minimax":
            move, _ = self.minimax_decision(current_player_score, opponent_score)
        elif self.algorithm == "alphabeta":
            move, _ = self.alphabeta_decision(current_player_score, opponent_score)
        self.make_move(move)

    def minimax_decision(self, current_player_score, opponent_score):
        best_score = -float('inf')
        best_move = None
        for i, option in enumerate(get_children(self.string)):
            score = minimax(option, 3, False, current_player_score, opponent_score)
            if score > best_score:
                best_score = score
                best_move = i
        return best_move, best_score

    def alphabeta_decision(self, current_player_score, opponent_score):
        best_score = -float('inf')
        best_move = None
        for i, option in enumerate(get_children(self.string)):
            score = alphabeta(option, 3, -float('inf'), float('inf'), False, current_player_score,
opponent_score)
            if score > best_score:
                best_score = score
                best_move = i
        return best_move, best_score

    def run(self):
        while self.string:
            if self.current_player == "computer":
                self.computer_move()
            else:
                # Prompt user to make a move (not included in this code)
                pass

def main():
    pygame.init()
```

```
    start_player, algorithm = display_start_menu(screen, font)  # Display the start menu and get user choices
    string_length = input_number(screen, font, "Enter string length (15-25):")  # Get the string length from the user
    game = Game(string_length, start_player, algorithm)  # Initialize the game with the chosen settings
    game.run()  # Start the game loop

if __name__ == "__main__":
    main()
```

7.Conclusions:

Reflecting on our journey through this practical Assignment 1 in "Fundamentals of Artificial Intelligence," we embarked on an exploration that not only deepened our understanding of AI's role in strategic games but also reinforced the collaborative synergy between human creativity and artificial intelligence. Our efforts culminated in a comprehensive analysis of Minimax and Alpha-Beta Pruning algorithms, their application in a number game, and the crucial role of heuristic evaluation. Here are our key takeaways:

1. **Collaborative Insights and Algorithmic Strategy:** We found Alpha-Beta Pruning to outperform the Minimax algorithm in terms of computational efficiency, particularly in more complex game scenarios. This discovery underscored the importance of algorithm selection in AI applications and highlighted the strategic depth that can be achieved when human insights guide AI development.
2. **Heuristic Evaluation's Role:** Our development of a heuristic evaluation function was pivotal. It allowed us to estimate the strategic value of various game states accurately. This process underscored the heuristic evaluation's role in connecting algorithmic rigor with the nuanced strategies of the game, illustrating how AI can simulate and extend human strategic thinking.
3. **AI's Broad Applicability:** The project served as a vivid illustration of AI's capacity to model complex decision trees and its broader applicability across various fields. It opened our eyes to AI's potential in simulating complex scenarios beyond gaming, such as in economic models and strategic planning, highlighting AI's versatility in addressing multifaceted challenges.
4. **Emphasizing Human-AI Synergy:** Throughout this project, the integration of human intuition and strategic thinking with AI's computational power enriched our outcomes. This synergy was particularly evident in our approach to refining the heuristic evaluation function and selecting the most appropriate algorithmic strategy, showcasing the value of human-centric insights in AI development.

5. **Utilizing ChatGPT for Enhanced Collaboration:** A notable aspect of our project was leveraging ChatGPT for generating reports on our experiments and assisting in the development of code snippets, such as those for the graphical interface. ChatGPT became an invaluable tool in our research and documentation process, aiding in tasks ranging from clarifying complex concepts to streamlining our coding efforts. For instance, we used prompts like "Explain the difference between Minimax and Alpha-Beta Pruning algorithms" for clarifying concepts, and "Generate a Python snippet for a graphical interface for a number game" for coding assistance and also is used in drawing Conclusions and also for generating report from 2 docx experiments file which were created fully Unique (No tool) for drawing comparison . ChatGPT's contributions significantly enhanced our project's depth and breadth, showcasing its potential as a collaborative tool in academic and research settings.

In conclusion, our project journey was a testament to the powerful combination of AI and human creativity. It not only achieved its educational objectives but also provided us with valuable insights into the potential of AI in strategic decision-making and beyond. Our experience with ChatGPT, in particular, highlighted the evolving role of AI tools in enhancing academic collaboration and innovation. As we move forward, the knowledge and insights gained from this project will undoubtedly serve as a solid foundation for future explorations into the expansive field of artificial intelligence.