1. Overview

The code implements a two-player game where both players alternately choose subsets of numbers to maximize their score, with the goal of exceeding the other player's score. The game implements **Minimax** and **Alpha-Beta Pruning** algorithms to decide the optimal moves for both players. It dynamically calculates valid moves based on subsets of remaining numbers and the players' scores.

2. Components of the Code

2.1 Minimax Algorithm

The minimax function is a recursive implementation of the Minimax algorithm, designed for a turn-based game.

Function:

def minimax(remaining_numbers, p1_score, p2_score, is_p1_turn, target_score)

- **Parameters:**
    - remaining_numbers: List of numbers still available to pick.
    - p1_score: Current score of Player 1.
    - p2_score: Current score of Player 2.
    - is_p1_turn: Boolean flag to indicate if it's Player 1's turn.
    - target_score: The score the current player needs to exceed to make a valid move.
- **Description:**
    - The function generates all valid subsets of the remaining numbers that the player can choose, updates their score, and recursively calls minimax to simulate the opponent's turn. The goal is to maximize Player 1's score and minimize Player 2's score.
    - Base case: If there are no remaining numbers, it returns the difference between Player 1's and Player 2's scores.

2.2 Alpha-Beta Pruning Algorithm

The alpha_beta function is an enhanced version of the Minimax algorithm that incorporates Alpha-Beta pruning to reduce the search space.

Function:

def alpha_beta(remaining_numbers, p1_score, p2_score, is_p1_turn, target_score, alpha, beta)

- **Parameters:**
    - Same as the minimax function, but with two additional parameters:
        - alpha: The best score that the maximizer (Player 1) can guarantee.
        - beta: The best score that the minimizer (Player 2) can guarantee.
- **Description:**

- Similar to minimax, but this function prunes branches that are unnecessary to explore by checking if the current alpha (maximizer) is greater than or equal to beta (minimizer). If so, the function stops further exploration of that branch.
- Pruning happens when it's certain that a particular branch will not affect the final decision.

2.3 Valid Subsets Generation

The function get_valid_subset generates all possible valid subsets from the remaining numbers for a player, ensuring that once the player reaches or exceeds the target score, no more numbers are added to the subset.

Function:

def get_valid_subset(remaining_numbers, target_score, current_score)

- **Parameters:**
    - remaining_numbers: List of available numbers for selection.
    - target_score: The score the current player needs to exceed to make a valid move.
    - current_score: The current score of the player.
- **Description:**
    - Iterates over all possible subsets of remaining numbers and checks if they can help the player reach or exceed the target score. Only subsets that meet this condition are returned as valid moves.

2.4 Best Move Finder

The find_best_move function uses either the Minimax or Alpha-Beta Pruning algorithm to find the optimal move for the current player.

Function:

def find_best_move(remaining_numbers, player_score, opponent_score, is_p1_turn, use_alpha_beta)

- **Parameters:**
    - Same as minimax and alpha_beta, with an additional parameter:
        - use_alpha_beta: Boolean flag indicating whether to use Alpha-Beta Pruning or regular Minimax.
- **Description:**
    - For each valid subset of numbers, it computes the resulting score and evaluates the move using either Minimax or Alpha-Beta Pruning. The best move is the one that maximizes the player's score (if it's their turn) or minimizes the opponent's score (if it's the opponent's turn).

2.5 Game Execution

The main function, play_game, manages the game flow.

Function:

def play_game()

- **Description:**
    - The game starts with Player 1 making a random initial move, after which the players take turns. For each turn, it calculates the best move using either Minimax or Alpha-Beta Pruning, updates the scores, and prints the current game state.
    - The game ends when no numbers are left, and the player with the highest score wins.

2.6 Input Validation

The function get_valid_number ensures that the user provides a valid number as input, rejecting non-integer or negative inputs.

Function:

def get_valid_number(prompt)

- **Description:**
    - Repeatedly prompts the user until a valid positive integer is provided.

3. How the Game Works

- **Initialization**:
    - The game begins by asking the user how many numbers (n) to play with. The remaining numbers will be from 1 to n.
- **Player 1's First Move**:
    - Player 1 is prompted to pick the first number. This number is removed from the list of remaining numbers, and Player 1's score is updated.
- **Turns**:
    - The game alternates between Player 1 and Player 2. During each turn:
        - The best move is computed using Minimax or Alpha-Beta Pruning based on the choice made at the start.
        - The player's score is updated based on the sum of the chosen numbers.
- **Ending the Game**:
    - When no numbers are left, the game ends, and the player with the higher score wins. If the scores are equal, it is declared a tie.
- **Efficiency**:
    - Alpha-Beta Pruning reduces the number of recursive calls by pruning branches that cannot influence the final decision, making the game faster.

4. Example Flow

- Suppose the user inputs 5 numbers (1 to 5). Player 1 randomly picks the number 3 as their first move, leaving [1, 2, 4, 5].
- The game alternates between Player 1 and Player 2, each trying to select numbers to maximize their own score or minimize the opponent's score.
- The game ends when all numbers are chosen, and the final scores determine the winner.

5. Conclusion

This implementation uses advanced AI techniques like Minimax and Alpha-Beta Pruning to compute optimal game strategies, simulating the decision-making process of both players. The Alpha-Beta Pruning significantly enhances the efficiency of the game by avoiding unnecessary computations.