To design a system for tracking the pairs of meeting players and determining the winner in a multiplayer game with n ≥ 1000 players, we need to carefully manage the interactions between players. The main goal is to determine when a player has met all other players at least once, and we want to track these meetings efficiently.

**Key Requirements:**

1. **Track Meetings:** Each time a player i meets another player j, we need to record that event.
2. **Track who has met whom:** For each player, we need to track which other players they have met.
3. **Determine the winner:** The first player to meet all other players should be declared the winner.

**Approach:**

1. **Data Structure to Track Meetings:**
   * For each player i, maintain a set of players they have already met. A set is suitable because it allows constant-time checking for membership and ensures that duplicates (e.g., repeated meetings between the same pair of players) are not stored.
   * A **list of sets** can be used to track the players each player has met. Specifically, the i-th set in the list will store all players that player i has met.
2. **Track the Number of Meetings:**
   * We can use a counter for each player to keep track of the number of distinct players they have met. Once this counter reaches n-1 (i.e., the player has met all other players), they are the winner.
3. **Meeting Logic (meet(i, j)):**
   * Whenever the meet(i, j) function is called, we add j to the set of players that i has met, and vice versa.
   * We also increment the counter for each of the two players involved in the meeting.
   * After each meeting, we check if either player has met all other players by comparing the size of their meeting sets to n-1.
4. **Determine the Winner:**
   * The winner is the first player whose meeting set reaches size n-1, indicating they have met all other players. If multiple players reach this condition at the same time, it's a tie.

**Detailed Solution:**

import java.util.HashSet;

public class MultiplayerGame {

// List of sets to track the players each player has met

private HashSet<Integer>[] meetings;

// Counter array to track how many distinct players each player has met

private int[] meetCount;

private int n; // Number of players

public MultiplayerGame(int n) {

this.n = n;

meetings = new HashSet[n];

meetCount = new int[n];

// Initialize the meetings sets and meet counters for each player

for (int i = 0; i < n; i++) {

meetings[i] = new HashSet<>();

meetCount[i] = 0;

}

}

// This method is called when player i meets player j

public void meet(int i, int j) {

// Add player j to the set of players player i has met

if (meetings[i].add(j)) {

meetCount[i]++;

}

// Add player i to the set of players player j has met

if (meetings[j].add(i)) {

meetCount[j]++;

}

// Check if player i has met all players

if (meetCount[i] == n - 1) {

System.out.println("Player " + (i + 1) + " has met all players. They are the winner!");

}

// Check if player j has met all players

if (meetCount[j] == n - 1) {

System.out.println("Player " + (j + 1) + " has met all players. They are the winner!");

}

}

// Main method for testing

public static void main(String[] args) {

// Create a game with 1000 players

MultiplayerGame game = new MultiplayerGame(1000);

// Simulate some meetings (this would normally be done by game events)

game.meet(0, 1);

game.meet(1, 2);

game.meet(0, 2);

game.meet(3, 4);

// Add more meet calls as needed for the simulation...

}

}

**Explanation:**

1. **meetings array**: This is an array of HashSet<Integer>, where each set stores the distinct players that a given player has met. For example, meetings[0] stores the players player 0 has met, and meetings[1] stores the players player 1 has met.
2. **meetCount array**: This array keeps track of the number of distinct players each player has met. If meetCount[i] == n-1, then player i has met all other players and is the winner.
3. **meet method**: This method updates the meeting sets for both players and checks whether either player has met all others by comparing their meetCount to n-1.
4. **Winner Declaration**: When a player meets all other players, they are declared the winner, and the program prints a message.

**Time Complexity:**

* **Meet method**: The time complexity of adding an element to a HashSet is **O(1)** on average, so each call to meet(i, j) will be **O(1)**.
* **Checking for winner**: Checking the meetCount array to see if a player has met all other players is **O(1)**.

Thus, the time complexity of each meeting is **O(1)**, making this solution highly efficient even for large values of n.

**Space Complexity:**

* The space complexity is **O(n^2)**, as we are storing the meeting records in n HashSet objects, each potentially containing up to n-1 elements (the set of players that each player has met).

This solution is efficient and scales well for large numbers of players while ensuring that the game interactions and winner determination are handled correctly.