

**Assignment 4**

**Course:** Software Testing

**Instructor:** Dr. Shaarbaf

**Student:** Alireza Dastmalchi Saei

**Student ID:** 993613026

# **Introduction**

This assignment delves into control flow graphs, data-flow analysis, and the nuanced art of test case generation. Our focal point is a code snippet extracted from game logic, and our mission is two-fold. Initially, we embark on constructing a Control Flow Graph (CFG) that visually encapsulates the program's intricate control flow structure. Subsequently, we navigate the terrain of Data-Flow Analysis to unveil the Def-Use (DU) paths within the code. The final leg of this intellectual odyssey entails crafting three discerning test cases tailored to the identified DU paths. Through these endeavors, we aim to fortify our understanding of program flow dynamics, decipher the nuanced relationships between data elements, and cultivate the skill of formulating compelling test scenarios to validate the robustness of the provided code.

**Objectives**

1. **Constructing Control Flow Graph (CFG):** You will begin by creating a visual representation of the program's control flow using a Control Flow Graph. This will provide insights into the order in which statements and branches are executed.
2. **Data-Flow Analysis and DU Paths:** After building the CFG, you'll delve into Data-Flow Analysis to identify Def-Use (DU) paths within the program. Understanding these paths is crucial for uncovering how data is manipulated and used across various code sections.
3. **Test Case Generation:** The final part of the assignment involves generating three meaningful test cases based on the identified DU paths. These test cases will aim to provide adequate coverage to ensure the robustness of the code.

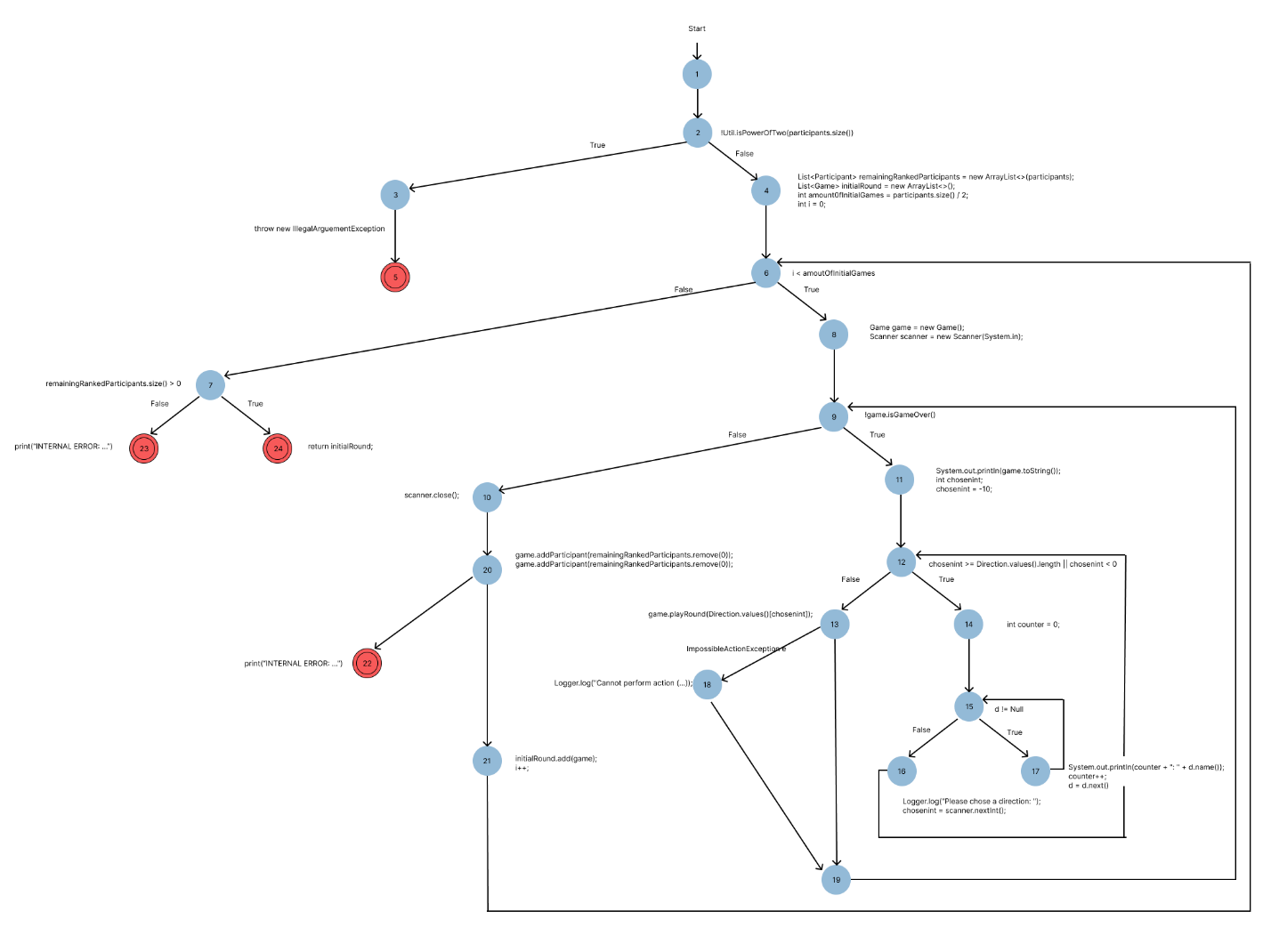
# **Code**

The code given in the assignment is represented below.



# **Control Flow Graph (CFG)**

The following graph is constructed from the code given in the previous part. It has four end nodes, three due to the Exceptions. One end node contains the “*return”* statement.



# **Definition Use Paths (DU Paths)**

In this part, we will extract all Def-Use Paths from the constructed CFG in the previous part. First, all variable definitions and their uses must be identified.

1. **participants**

* **D:** Declared as a method parameter List<Participant> participants.
* **U:** Used as an input to check whether the size is a power of two and later to build initial games.

1. **remainingRankedParticipants**

* **D:** Defined as List<Participant> remainingRankedParticipants = new ArrayList<>(participants).
* **U:** Used to add participants to the games, ensuring that the participants are removed once added to a game.

1. **initialRound**

* **D:** Defined as List<Game> initialRound = new ArrayList<>().
* **U:** Populated with newly created games.

1. **amount0fInitialGames**

* **D:** Defined as int amount0fInitialGames = participants.size() / 2.
* **U:** Used in the loop condition to determine the number of iterations for creating games.

1. **Game**

* **D:** Defined as Game game = new Game().
* **U:** Used to play rounds, check if the game is over, and add participants.

1. **scanner**

* **D:** Defined as Scanner scanner = new Scanner(System.in).
* **U:** Used to obtain user input for the game's direction.

1. **chosenint**

* **D:** Defined as int chosenint.
* **U:** Used to store the user's chosen direction during the input loop.

1. **counter**

* **D:** Defined as int counter.
* **U:** Used in the loop to display the available directions with corresponding indices.

1. **D**

* **D:** Defined in the enhanced for loop as Direction d.
* **U:** Used to iterate over the Direction.values().

1. **e**

* **D:** Defined as ImpossibleActionException e.
* **U:** Used to catch exceptions and log error messages.

1. **Logger**

* **D:** A logger is a class or method responsible for logging.
* **U:** Used to log messages for user interaction and error handling.