You have been given two full days to implement a solution to a design problem using Java, Go, or Node.js.

This exercise is designed to assess your language and low level design skills, as well as your ability to write readable and concise code.

Quality of modeling of entities and behavior, well readable & succinct code, test coverage and a clean & frequent commit history is key.

Please keep these factors in mind as you solve the problem.

Problem Statement:

Design a Magical Arena. Every Player is defined by a “health” attribute, “strength” attribute and an “attack” attribute - all positive integers. The player dies if his health attribute touches 0.

Any two player can fight a match in the arena. Players attack in turns. Attacking player rolls the attacking dice and the defending player rolls the defending dice. The “attack” value multiplied by the outcome of the attacking dice roll is the damage created by the attacker. The defender “strength” value, multiplied by the outcome of the defending dice is the damage defended by the defender. Whatever damage created by attacker which is in excess of the damage defended by the defender will reduce the “health” of the defender. Game ends when any players health reaches 0

Player with lower health attacks first at the start of a match.

Assume two players . Player A 50 health 5 strength 10 attack Player B 100 health 10 strength and 5 attack . Attacking die and Defending die are both 6 sided die with values ranging from 1- 6

Player A attacks and rolls die. Die roll : 5 . Player B defends and rolls die. Die roll 2

Attack damage is 5 \* 10 = 50 ; Defending strength = 10 \* 2 = 20 ; Player B health reduced by 30 to 70

Player B attacks and rolls die. Die roll : 4. Player A defends and rolls die. Die Roll 3

Attack damage is 4 \* 5 = 20 ; Defending strength = 5 \* 3 = 15 ; Player A health reduced by 5 to 45

And so on

Rules of the Game:

You can use Java, Go or Node.js to implement the solution, without using any third-party libraries or frameworks ( common and essential helper libs and packages are allowed ex. math.rand() is ok.

The code should NOT be hosted on GitHub or made public. Create a zip of the folder with local git history and you should provide us with the zip

You should provide a README file in the repository that explains how to run the code and any other relevant information.

You should provide unit tests for your solution.

We expect you to commit frequently with relevant commit messages. Multiple incremental commits are valued over one all-inclusive commit.

Evaluation Criteria:

We will evaluate your solution based on the following factors:

Simple design: Does the code have a clear and simple design? Is it easy to understand and modify?

Readability: Is the code well-organized and easy to read? Are the naming conventions clear and consistent?

Modelling: Are the objects and classes used in the code well-designed and appropriate for the problem at hand?

Maintainability: Is the code easy to maintain and modify? Are there any potential areas of concern or technical debt?

Testability: Are there comprehensive unit tests for the code? Does the code have a high degree of test coverage?

We look forward to seeing your solution and wish you the best of luck!