2019 Coding Test (English version)

Created	@Nov 05, 2019 2:11 PM
Tags	

1. Problem solving ability (Front-End)

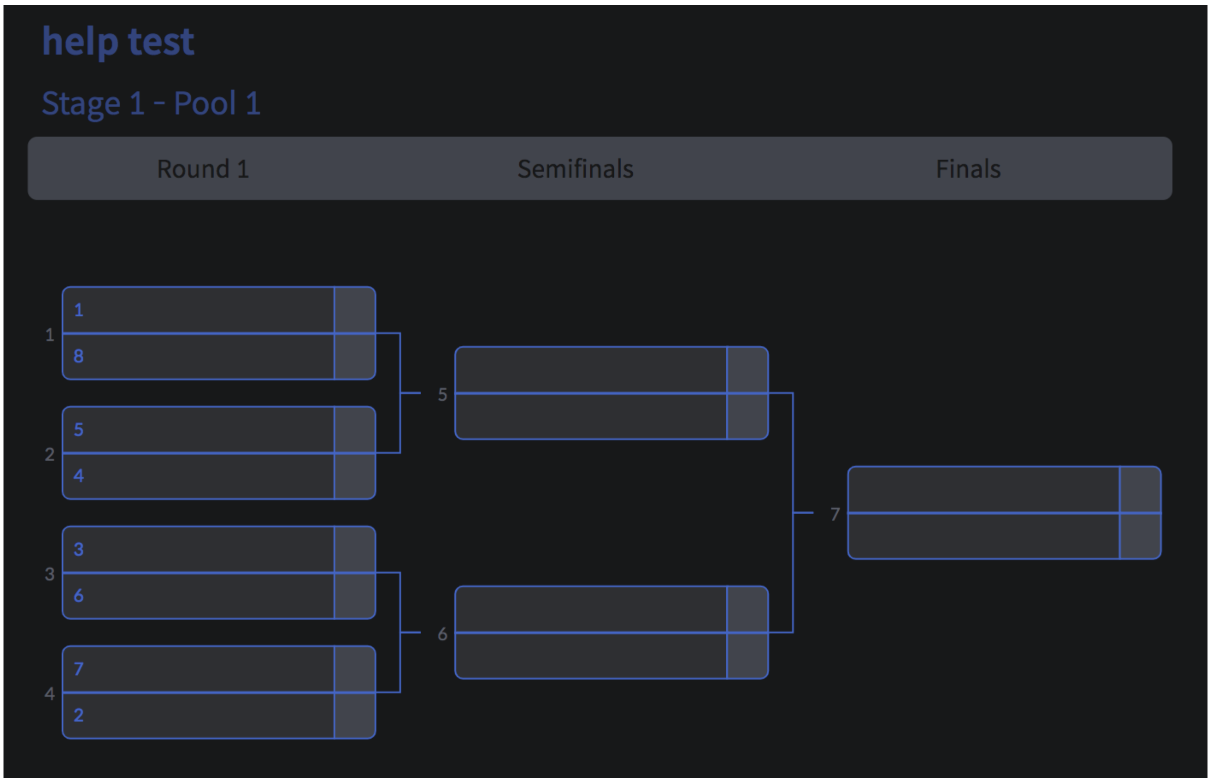
[Tournament Bracket]

Battle.dog is a platform designed to provide an accessible means for users to create tournaments.

The most popular tournament type on Battle.dog is "single elimination".

"Single elimination" is how most tournaments are organized.

Tournament participants are each pitted against another participant in a 1:1 match, where the victor moves on to the next match with another victor, while the loser is "eliminated" from the tournament.



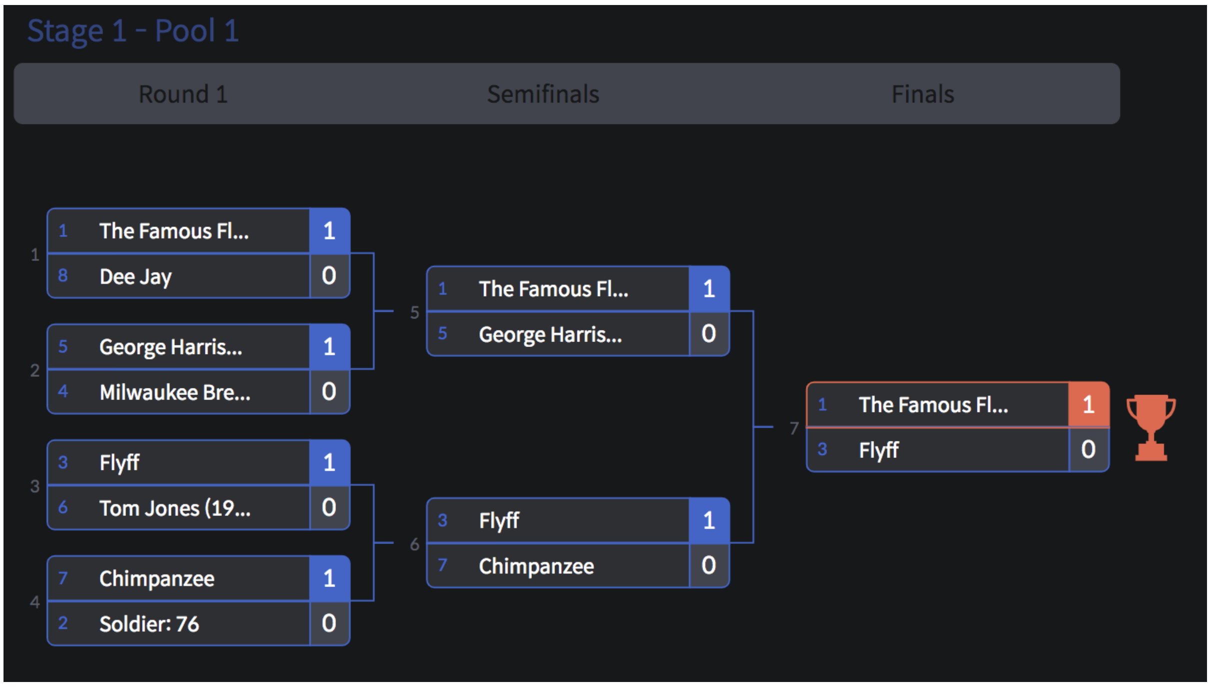


Figure 1

Configure single elimination tournaments and use React or **Framework that you feel comfortable** with.

You can use Snippet and Boilerplate, but please do not use existing tournament libraries.

Requirements

- Please keep in mind that **the number of tournament participants may change at any time.**
  - The number of tournament participants dictate the number of matches.
  - The number of matches in turn affect the number of rounds.
- Use the data sample currently used in Battle.dog and implement it as shown above (Figure 1).

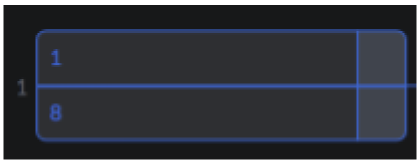


Figure 2

  - Figure above(Figure 2) is a single `match` . A single `match` is between two (`participant.`)
  - The `number` before the `match` is what the matches are referred to by.
  - The numbers in the spaces where player names go are `seed` numbers for match participants.
  - The matches above are grouped as a single `round`. Each Round is identified by the unique ID `r` of each `match` .
  - Matches and rounds should be spaced at regular intervals as shown in Figure 1 to help distinguish between the two.
- Brackets may be described freely as `canvas, div, svg` or whatever is appropriate.
  - No restrictions are set for style, and `css, scss, less, styled-component` etc. are all allowed. **If time is insufficient, you do not have to implement all styles.**

Additional requirements

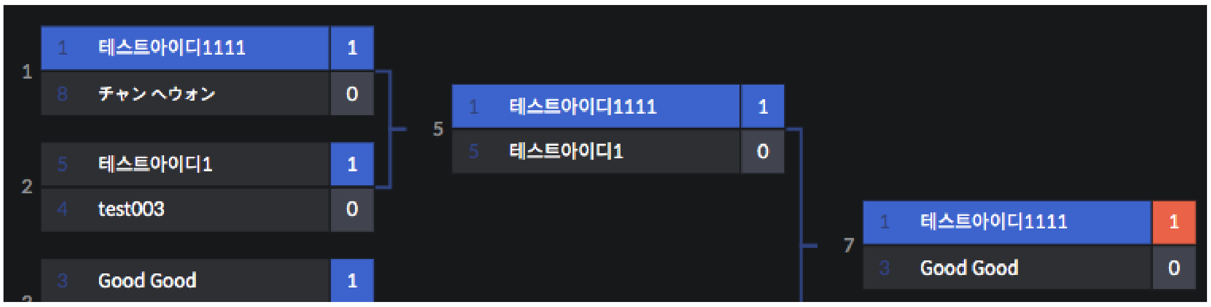
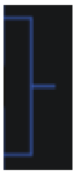
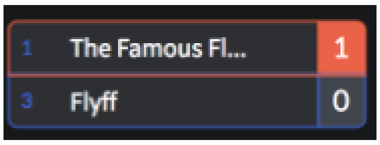


Figure 3

- When the mouse cursor is over a participant, the background color of all match participants also change, as in the example of Figure 3.



- Form a line that connects match outcomes to the next round.



- Emphasize the names of match victors as in the example below:

Match Data Sample JSON

```
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/3776d7f
7-c238-499c-88f3-c0725e45f733/InterviewData.json
```

2. Algorithms

[Pokemon GO]

One of the easiest ways to gain XP in Pokemon GO is to evolve captured Pokemons.

Every time you capture a Pokemon, you can gain a candy.

However, candy types and Pokemon types need to be compatible for use.

If you have enough candy for a particular Pokemon, you can use it to evolve your Pokemon.

You may choose to send Pikachu to Professor Oak, Pikachu will not return, but you will receive a Pikachu candy.

(For example, Pidgy candy can only be used with Pidgy.)

In the case of Pikachu, evolution requires 12 Pikachu candies. The evolution provides 500XP and also a bonus candy.

Example

Let's say that you have 2 Pikachus and 23 Pikachu candies.

You evolved one of the Pikachus using 12 candies.

This would leave you with one only 11 candies, but the evolution bonus provides 500XP and 1 extra candy. This allows enough candies for another evolution of the second Pikachu, which adds another 500XP, for a total of 1,000XP.

Assignment

Write a function with the number of Pikachus and Pikachu candies as two variables and find the maximum XP that a player may receive from a set of given Pikachus and Pikachu candies.

Sample tests

```
Test.describe("Basic Tests", function(){
  Test.assertEquals(pikachuCalc(1, 12), 500, 'Expected 500');
  Test.assertEquals(pikachuCalc(13, 144), 6500, 'Expected 6500');
  Test.assertEquals(pikachuCalc(10, 63), 3000, 'Expected 3000');
  Test.assertEquals(pikachuCalc(1, 63), 500, 'Expected 500');
  Test.assertEquals(pikachuCalc(63, 1), 2500, 'Expected 2500');
  Test.assertEquals(pikachuCalc(0, 0), 0, 'Expected 0');
  Test.assertEquals(pikachuCalc(3, 9), 0, 'Expected 0');
  Test.assertEquals(pikachuCalc(4, 9), 500, 'Expected 500');
});
```