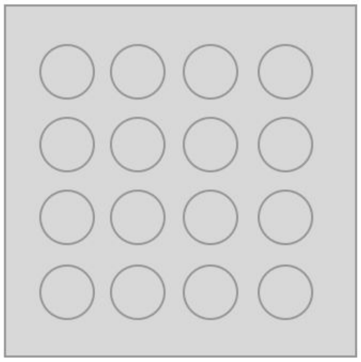
98point6 Drop-Token: At-home interview question for BE engineers

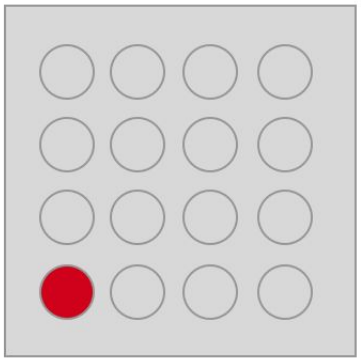
We would like you to implement a backend (REST web-service) that allows playing the game of 9dt, or 98point6 drop token. This should allow the players to create games, post moves, query moves and get state of games.

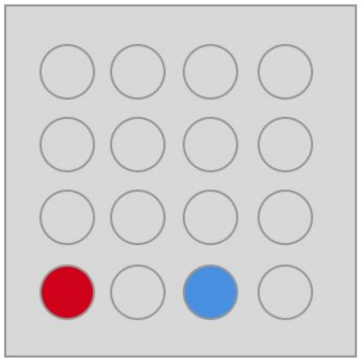
Rules of the Game

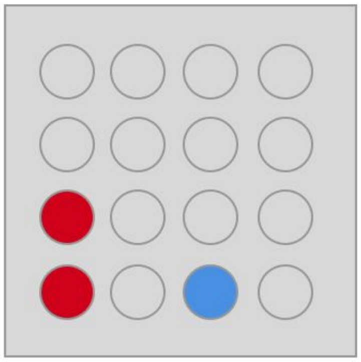
Drop Token takes place on a 4x4 grid. A token is dropped along a column and said token goes to the lowest unoccupied row of the board. A player wins when they have 4 tokens next to each other either along a row, in a column, or on a diagonal. If the board is filled, and nobody has won then the game is a draw. Each player takes a turn, starting with player 1, until the game reaches either win or draw. If a player tries to put a token in a column that is already full, it results in an error state, and the player must play again until they play a valid move.

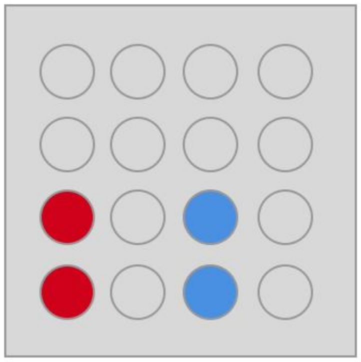
Example Game

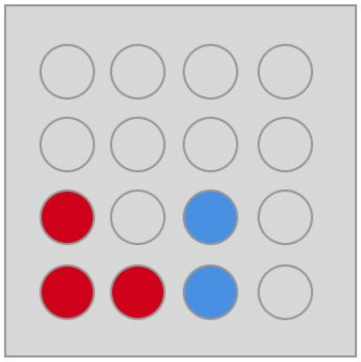

Empty Board

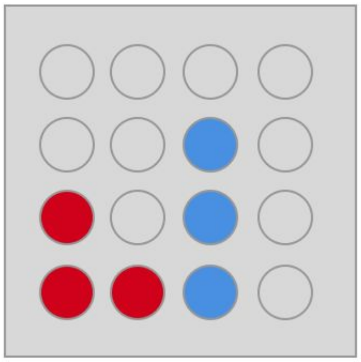

Player 1 goes in column 0

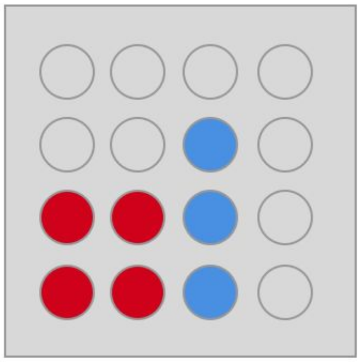

Player 2 goes in column 2

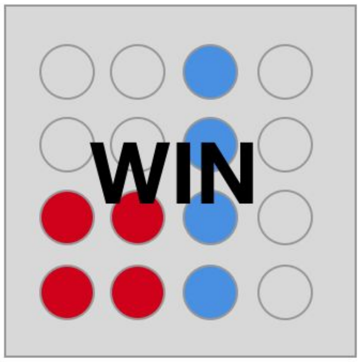

Player 1 goes in column 0


Player 2 goes in column 2


Player 1 goes in column 1


Player 2 goes in column 2


Player 1 goes in column 1



Player 2 goes in column 2

Minimal Requirements

- Each game is between k = 2 individuals, basic board size is 4x4 (number of columns x number of rows)

- A player can quit a game at every moment while the game is still in progress. The game will continue as long as there are 2 or more active players and the game is not done. In case only a single player is left, that player is considered the winner.
- The backend should validate that a move move is valid (it's the player's turn, column is not already full)
- The backend should identify a winning state.
- Multiple games may be running at the same time.

API

GET /drop_token - Return all in-progress games.

Output

```
{ "games" : ["gameid1", "gameid2"] }
```

Status codes:
- 200 - OK. On success

POST /drop_token - Create a new game.

Input:
```
{ "players": ["player1", "player2"],
  "columns": 4,
  "rows": 4
}
```

Output:
```
{ "gameId": "some_string_token"}
```

Status codes
- 200 - OK. On success
- 400 - Malformed request

GET /drop_token/{gameId} - Get the state of the game.

Output:

```
{ "players" : ["player1", "player2"], # Initial list of players.
  "state": "DONE/IN_PROGRESS",
  "winner": "player1", # in case of draw, winner will be null, state
will be DONE.
                       # in case game is still in progess, key should
not exist.
}
```

Status codes

- 200 - OK. On success
- 400 - Malformed request
- 404 - Game/moves not found.

GET /drop_token/{gameId}/moves- Get (sub) list of the moves played.

Optional Query parameters: GET /drop_token/{gameId}/moves?start=0&until=1.

Output:

```
{
  "moves": [
    {"type": "MOVE", "player": "player1", "column":1},
    {"type": "QUIT", "player": "player2"}
  ]
}
```

Status codes

- 200 - OK. On success
- 400 - Malformed request
- 404 - Game/moves not found.

POST /drop_token/{gameId}/{playerId} - Post a move.

Input:

```
{
 "column" : 2
}
```

Output:

```
{
  "move": "{gameId}/moves/{move_number}"
}
```

Status codes

200 - OK. On success
400 - Malformed input. Illegal move
404 - Game not found or player is not a part of it.
409 - Player tried to post when it's not their turn.

GET /drop_token/{gameId}/moves/{move_number} - Return the move.

Output:

```
{
  "type" : "MOVE",
  "player": "player1",
  "column": 2
}
```

Status codes

- 200 - OK. On success
- 400 - Malformed request
- 404 - Game/moves not found.

DELETE /drop_token/{gameId}/{playerId} - Player quits from game.

Status codes

- 202 - OK. On success
- 404 - Game not found or player is not a part of it.
- 410 - Game is already in DONE state.

Assessment and Interview

After we receive your submission we will conduct a code review and execute our suite of integration tests that assert the correctness of the API. Through the course of our review and testing we will assess your implementation on several different criteria:

- **Correctness**: Does your API adhere to the specification
- **Robustness**: Does your implementation handle malformed, edge case, or fuzzed input without failing and while returning meaningful messages on the cause of the failure?
- **Readability**: Can an engineer unfamiliar with your implementation read and understand what you wrote with sufficient depth to make modifications? This criteria speaks to style, naming conventions, organization, and comments.
- **Scalability**: Will your solution perform under stress of hundreds-to-thousands of games.

You will review your solution with 1-2 members of the engineering team. You should be prepared to talk about your implementation approach, design trade-offs and approach to testing and validation.

Submitting your solution

Please submit your source code and instructions for building/running it along with your test plan to the link provided in email by **12pm (noon) the day before your interview**.

If you have several files to submit, please create a zip file or compressed archive (e.g., .tar.gz) and upload it to the submission link.

An optional starting point Java shim for your code is provided in the hope it will reduce boiler-plate code and some ramp-up in unknown technologies. Access it here: https://s3-us-west-2.amazonaws.com/98point6-homework-assets/98point6BackendHomeworkJavaStarter.zip Feel free to use it or ignore it.

If you choose not to use the provided shim, you must provide thorough instructions on how to set up and run your service. We are experienced developers, but we may not be familiar with the tools or languages you used, so please draft the instructions accordingly.

For dotnet solutions, we will compile and run your code using mono.