CS300 Exam Spring 2021 Assignment 2

No discussion and no sharing of code or debugging each other's code is allowed. Ask TAs for any help.

In this assignment you are required to implement a single multiplayer game of Ludo for four players. You are not required to support multiple parallel games or to support a second game after the first one finishes or to support disconnecting players.

Part 1

Remove the hard-coded ludo board from ludo.html and generate it dynamically using a React Component called Ludo. Remember that class attribute is written className in React.

Your component state will have a board state like this one.

```
[[['blue','blue','blue','blue'],[],[],[],[],[],[],[],[],[],[],[],[]
,['red','red','red','red']],[[],[],[],[],[],[],[],[],[],[],[],[],[]
,[]],[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]],[[],[],[],[],[],[]
,[],[],[],[],[],[],[],[],[]],[[],[],[],[],[],[],[],[],[],[],[],[],[],[
],[]],[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]],[[],[],[],[],[],[
],[],[],[],[],[],[],[],[],[]],[[],[],[],[],[],[],[],[],[],[],[],[],[],
[],[]],[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]],[[],[],[],[],[],
[],[],[],[],[],[],[],[],[],[]],[[],[],[],[],[],[],[],[],[],[],[],[],[]
,[],[]],[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]],[[],[],[],[],[]
,[],[],[],[],[],[],[],[],[],[]],[[],[],[],[],[],[],[],[],[],[],[],[],[
],[],[]],[['yellow','yellow','yellow','yellow'],[],[],[],[],[],[],[],[
],[],[],[],[],['green','green','green','green']]]
```

You do not need to change or understand ludo.css

Now change your initial board state to [] in the client. Introduce websockets and have the server send the initial board state. The message format will be JSON with a key named 'type' where the only message we will be handling in this part is where type='newboard' and a second key named 'board' has the full new board. Replace the board in your state with the one coming from the server. Your server should just send the hard-coded board used in part 2 to every client that connects.

Part 2

Now you need to add the functionality that, whenever you click on a sprite, your server should be notified of that sprite's color and coordinates on the board. The server should then remove it from its current place on the board (you can use indexOf to find it in the list of sprites on a given location and use splice to remove it). The server should calculate its new coordinates using the rules of Ludo. You may write your own logic or you may use the step function below. You can hard-code the last argument to 1 for this part. The server will then add the sprite at new coordinates (you can use push function to add element to list of sprites) and send the updated board to the client.

Once finished, click any sprites will move it on its appropriate path, one step at a time.

```
const step = (color, ox, oy, steps) => {
    const transform = ([ox,oy]) => ({'blue': [+ox,+oy], 'green': [-
```

```
ox,-oy], 'red': [-oy,+ox], 'yellow': [+oy,-ox]}[color])
    const path = ['-7,-7', '-1,-6', '-1,-5', '-1,-4', '-1,-3', '-1,-
2', '-2,-1', '-3,-1', '-4,-1', '-5,-1', '-6,-1', '-7,-1', '-7,0', '-
7,1', '-6,1', '-5,1', '-4,1', '-3,1', '-2,1', '-1,2', '-1,3', '-1,4',
'-1,5', '-1,6', '-1,7', '0,7', '1,7', '1,6', '1,5', '1,4', '1,3',
'1,2', '2,1', '3,1', '4,1', '5,1', '6,1', '7,1', '7,0', '7,-1', '6,-
1', '5,-1', '4,-1', '3,-1', '2,-1', '1,-2', '1,-3', '1,-4', '1,-5',
'1,-6', '1,-7', '0,-7', '0,-6', '0,-5', '0,-4', '0,-3', '0,-2', '0,-
1']
    const [x,y] =
transform(transform(transform(path[path.indexOf(transform([ox-7, oy-
7]).join(','))+steps].split(','))))
    return [x+7,y+7]
}
```

Part 3

We now need a dice. Create a div with the class name 'dice' and inside the div, show the value of a data variable that represents the dice value.

Introduce a new server->client message (e.g. with type 'dice') to give the client its dice value. The dice value is given at connection time and then after every move (since there is a single player for now). The syntax to generate a random number from 0 to x is:

```
Math.floor(Math.random()*x)
```

Now ensure that the server takes as many steps as the dice value. The server has to remember what was the last value of dice sent to the client and use it as the last argument of step function (that was hard-coded to 1 previously).

However, if a sprite is at its starting position (you may use the iskilled function below), then it can only move if the dice has a 6 and it steps by only one place (i.e. out of the starting area). After this part every sprite moves the correct number of steps as the dice value.

```
const iskilled = (ox, oy) => (ox-7)*(ox-7)+(oy-7)*(oy-7) == 98
```

You do not need to worry about the dice being allowed to roll again if it produces a 6 the first time it is rolled. We will not be implementing this feature in our game. You also don't have to worry about what happens at the end of a sprites route. No error checks are required.

Part 4

Now modify this game into an actual 4 player game. Follow the steps below to do so:

Wait for four players to connect. Assign each player a color using a new websocket message that is sent when player connects. This color should be visible to the user in a new div with a class name of 'color red', or 'color blue', or 'color green', or 'color yellow'. Once the fourth player arrives, send all of them the newboard message so the board gets displayed.

Make sure that in the onclick handler, a message is sent to the server only if the player clicks on their own colored sprite.

Now modify your program such that all four players can play turn by turn. The turn of the player should be displayed in a text box (a new div with class="text_box"). For example, if it is blue's turn, every player's text box should have 'its blue's turn' written in their text box. All your clients should be able to see the current value on the dice.

If a player moves when it is not their turn, the server should ignore it and send a message to the client that is displayed in the text box that it is not their turn.

Part 5

Introduce killing sprites. If you reach a board location that is not a safe spot, all sprites of other colors get killed. Blue goes to (0,0), Red goes to (0,14), Green goes to (14,14), and Yellow goes to (14,0).

Create a winning condition. When a player wins, you have to inform the players of the winner in the text box. For this part, you have to ensure that no sprite can go beyond their last position so if there are 3 positions left, a 4, 5, or 6 on the dice will not allow that sprite to move.

Hint: For testing this part, you can temporarily hard-code a board state that is near winning so you can quickly test it.