1.

```
*   0b780ab Merge branch 'bread'
|\
| * e04cb65 bread 2
| * 2b0782f bread 1
* | 6d42d64 cookie 2
* | 0e8958e cookie 1
|/
* 6bd2eb4 commit 2
* a9a1ab1 commit 1
```

2. GitHub account name - **ggomaeng**

3a. Write out a bulleted feature list for the Hare and Hounds game you were asked to implement. Aim to be complete if high-level. (Yes this is generally an easy question because the homework gives a good description, but you need to pull out the complete key properties of the game as a bulleted list from that description.)

- There are two piece types and two players, the Hare and the Hound
- There are four pieces on the board, three hound pieces and one hare hound
- The game is turn based with the hound moving first when the other player joins.
- The player with the turn can only move one of his pieces per turn. A piece can only move one step (as in one block distance) at a time and it can only move to an empty location connected to its current location.
- Hounds cannot move backwards while the hare has no such restriction.
- The turns continue until one of the following (win) conditions occur:
    - The hare is trapped such that it has no valid move. The hounds win in this case.
    - The hare manages to sneak past the hounds. i.e. it moves to a square such that there are no hounds to left of it. In this case the hare wins.
    - The same board position occurs three times over the course of the game. In this case the hounds are considered to be stalling and the hare wins.
- The player can start a new game by clicking on the start button, and can share the game via a link
- The board is specified by x and y coordinates—the size of the board is 3 by 4, excluding every corner coordinates
- The service must be able to host multiple games at once
- The client can request 4 endpoints to either start a game, join a game, make a turn, get the current board state, and get the current game state

a. 3b. Move validity checking is part of the "domain" that you need to have a firm understanding of. The assigment spec gives a high-level description; elaborate on this spec by writing a *use-case* for a hound move. All you know is the move was from `x1,y1` to `x2,y2` - you have four numerical values. Along with the lecture notes the past project examples contain sample usecases. We are not going to be picky on the exact syntax, just write something that gives a good step-by-step high-level description of the process.

1. Hound requests a move
2. Check if either “from” coordinate or “to” coordinate exist
    - a. If either doesn’t exist, throw an Error
3. Check if the requested “from” coordinate contains the hound piece
    - a. If it doesn’t exist, throw an Error
4. Check if the requested “to” coordinate is occupied by any piece
    - a. It it is occupied, throw an Error
5. Check if hound is trying to move backward
    - a. If trying to move backward, throw an Error
6. Check if the “from” coordinate and “to” coordinate are connected
    - a. If it’s not connected, throw an Error
7. Check if the hound is trying to move “1” distance or step
    - a. If it tried to move more than 1 step, throw an error
8. If all of these steps passed, complete the hound move
    - a. If the hare is trapped
        - a. End the game. Hound wins.
    - b. If the same board position occurs three times over the course of the game, hounds are considered to be stalling
        - a. End the game. Hare wins.
    - c. Else, it’s now Hare’s turn