Exercise 2 Code review

Project Description

Overview Description of Context

Both classes are used in a java game (Snakes and Ladders). The gameboard consists of various squares, on which the players need to progress until they exactly reach the end square. The players start on the start square and walk forward as many squares as the number that the player rolled on the die (d6).
There are snake squares and ladder squares, which technically act the same if a player lands on them. If you land on a ladder square, you will get the benefit of moving forward to a predefined better square. The snake square works the same, but you will be drawn back to a predefined worse square.
If a player x lands on another player y, then player x is sent back to the start square. This rule does not apply, when x lands on y, while y is on the end or the start square.
To finish the game, a player must exactly land on the end square. If a player rolls too much on the die, such that the player would land on an undefined square index (further than end square), the player moves the remaining steps backwards instead.

player.java

The player class has two global variables, which represent the name and the position of the player on the gameboard. When creating a player, the number assigned to the position is 1 (start square). The constructor needs a name for the player.

- Move
    - In order for other classes to move a player on the gameboard, the move method is used. As input you need the gameboard and the amount of steps. Steps is an integer, which may be negative (moving backwards).

    - If the player would be moved below the start square, the player position is then set to 1. If the player would be moved further than the end square, the remaining steps will move the player backwards.

    - If a player lands on a snake/ladder square, the player is moved back/forth on the gameboard depending on the step_back/step_forward integer defined on the snake/ladder square.

    - After considering the snake/ladder squares, the player must check whether another player is already on the reached square. If yes, then the moved player's position is set to 1. Otherwise the square is set to occupied.

gameboard.java

When the gameboard is initialized with the given amount of squares, some random pairs of squares are generated. These pairs represent snakes and ladders. A pair of squares represents a ladder if the first number of the pair is lower than the second one. Otherwise it is a snake. The first square of such a pair will be a snake/ladder square, while the second

one is always a normal square. Then all remaining squares are normal squares, except the start and end square. At the end of the initialization, the gameboard is printed.

- shuffled_array
    - Here the square indices are set randomly into an array. The indices do not include the start and end square

Future improvements

- Code Clarity:
    - The gameboard constructor could be split up into several methods.

- Object Orientation:
    - Maybe the user should only interact with the gameboard. As of now, the user rolls the die and must interact with the gameboard AND the player objects
    - Class variables should be set to private or protected if it is possible.

- Code Style:
    - Class names are not capitalized
    - comments may be rewritten in a formal way
    - variable names such as “a” could be renamed with a semantically more useful name

Team

- Lara Fried, David Steiger (Author), Tim Moser and Christian Birchler

Defects found

Gameboard

1. Missing parentheses in an if-else statement.
2. There are some variable naming conflicts, like the use of a capital letter, or non meaningful names (A, B..)
3. Classnames not capitalized
4. similar names for class and variable (square and squares)
5. extract methods from constructor
6. bloated code
7. Add more comments
8. Indentations
9. override “toString” of Gameboard
10. apply design pattern to avoid “instanceOf” statements
11. useless parentheses

Player

1. Not capitalized class names
2. Didn't use camelCase convention
3. Add more comments

4. Too much functionality as a player
    a. not enough abstraction
5. Too many if else constructs

Summary of the Recommendation

Gameboard

The team recommends to rename the following Variables:

- variable name A changed to snakesAndLadders
- variable name B changed to shuffledSquares
- int a changed to start
- int b changed to end
- int i changed to assignedFields
- rename snake_square to SnakeSquare
- normal_square to NormalSquare
- rename squares to numberOfSquares

It also recommends to extract the following methods from the Constructor:

- bloated constructor
- too many low level implementations
- extract method initGameboard()
- extract method printGameboard()
- extract method calculateNumberOfSquares()

We would further recommend to replace long if else statements for example with math.min() and math.max() functions. We would also capitalize Class names (e.g. Square()).
For the comments it is recommended to add more and clearer comments. The team would also recommend adding a class description. “instanceOf” statements should be avoided as they violate the open closed principle. To do this we would recommend using design patterns.

Player

We recommend to capitalize class names and use the camelCase convention. Again we would recommend adding class descriptions and more comments.
For this class we would recommend rewriting it completely. Because it would be beneficial to restructure the Game so the Player class does not have too much functionality.

Review Time and Defects Found

It took the whole team 4 hours to find 15 **major** Defects.We did not count every single name that was refactored.