Ellis Fitzgerald
COS 420: Object Oriented Design
May 2nd, 2025
AI Lab Book #7

LLM: DeepSeek DeepThink R1
Time Spent Overall: 1 hour
Time Spent Prompting: 0

Change #1

The first change I propose is making the Dice used in the Bulldog game a singleton. The reason for this is because the game rules do not involve or require the players to have their own dice. It makes sense for the dice to be shared since there is only one of them (or at least one set). Although I know overuse of the singleton pattern is not a good habit to get into. It was an idea I had on a whim earlier in this semester, and so I decided I might as well try. This change required me to really just delete lines in Player constructors removing the dice as a parameter. Additionally, I needed a way to still support using the "FakeRandomDice" for the JUnit test we had in a previous assignment. The way I did this was to write two "getInstance" methods, with the second overloading and providing an enumeration of "FAKE_RANDOM" or "PSEUDO_RANDOM" with the default being "PSEUDO_RANDOM." It is not the cleanest solution, but it definitely works.

```java
/**
 * Retrieves the global instance of the dice
 *
 * @return Dice to be used by the players
 */
public static DiceSuper getInstance() {  1 usage  new *
    return getInstance(DiceType.PSEUDO_RANDOM);
}

/**
 * Retrieves the global instance of the dice, if uninitialized it promptly creates an object of specified type
 *
 * @return Dice to be used by the players
 */
public static DiceSuper getInstance(DiceType type) {  7 usages  new *
    if(instance == null) {
        switch(type) {
            case PSEUDO_RANDOM:
                instance = new PseudoRandomDice(SIDES);
                break;
            case FAKE_RANDOM:
                instance = new FakeRandomDice(SIDES, new Scanner(System.in));
                break;
        }
    }
    return instance;
}
```

Change #2

The second change is much more “dramatic” and perhaps it completely overwrites my previous change. Instead of having the “play” function be in the Player class: why not just put the play loop in the Referee class? This addresses four things:

1. I no longer need the Player class to “setGameEventListener” which is how DeepSeek originally had changes triggered to the view.
2. This makes the Player class more “pure” in a strategy sense. In a functional programming language I don’t think I would even write a class and would simply use function/lambdas for the strategy, but in OOP the Player class now purely wraps over the “continueTurn” method.
3. Now the relationship between a player and the dice could be interpreted differently. Perhaps Bulldog is a game of blackjack and the person rolling is the referee/dealer. In this case it doesn’t matter, but the action of rolling is completely visible to the referee (quite helpful for determining cheating. How else would they know if the player overwrote their rolling logic to make them win every time?)
4. No longer having a recursive approach to turn cycles.

The only issue with this is it *does* break backwards compatibility with the JUnit test. Something I would address given I had enough time for this assignment. Though it is my mothers birthday today (yep I am using that card). So, we cannot say that this is a true “refactor” ; it is actually more like a rewrite.