Tyler Jones

Bugs Report

3/19/17

CS 362: Final Project

For Part-B of this assignment I used the ONID username from GitHub "**hellwegk**" for finding bugs.

I found two large bugs in her implementation.

BUG 1

The first bug I found was very high level and easy to spot. In order to find both bugs for this assignment, I started from the highest level possible and moved down. I looked at her PlayDominion.java file to see how she actually called a game and played it fully. I noticed that she passed a number and a seed into a new Gameboard and then called play on the Gameboard. This can be seen below.

```
1  package otherDom;
2
3  public class PlayGame {
4
5      public static void main(String args[])
6      {
7          int seed = Randomness.nextRandomInt(1000);
8          DominionBoard game = new DominionBoard(3, seed);
9          game.play();
10     }
11
12 }
```

I figured I would try to edit the number of players passed into the first argument of DominionBoard to a number that isn't valid for a Dominion game. I changed it to 10, and it was then I found my first bug. When calling anywhere from 2-4 as this argument, the game plays successfully, as it should, but the game gives me a NullPointerException when I made this value too large, as the code isn't configured to handle that many players. My input and her output that revealed the bug can be seen below.

```
    @Test
    public void testPlayerNumBug(){
        int seed = Randomness.nextRandomInt(1000);
        DominionBoard game = new DominionBoard(10, seed);
        game.play();
    }

}
```

```
Exception in thread "main" java.lang.NullPointerException
        at otherDom.DominionBoard.takeCard(DominionBoard.java:112)
        at otherDom.Player.makeStartingDeck(Player.java:84)
        at otherDom.Player.<init>(Player.java:25)
        at otherDom.DominionBoard.<init>(DominionBoard.java:27)
        at otherDom.PlayGame.main(PlayGame.java:8)
```

The solution to her bug would be when she initializes her game object, to be able to handle a number that is too large. She has if statements within her code to handle numbers that are too small (0-1), but not that are too large. It is an easy fix but one that needs to be made. The area where the second if statement for values that are too large should be added can be seen below.

```
public DominionBoard(int number, int seed)
{
    Randomness.reset(seed);
    List<Card> kingdom = new ArrayList<Card>();
    if (number <= 1)
    {
        number = 2;
    }
    kingdom = kingdomCards(seed);
    setUpGame(kingdom, number);
    players = new ArrayList<Player>();
    for (int i = 0; i < number; i++)
    {
        players.add(new Player("Player" + (i + 1), i, this));
    }
}
```

By simply adding a **if(number >4){ number = 4 };** statement, that is very similar to her existing if statement for values that are too small, this bug could be fixed.

BUG 2

The second bug I found was more subtle than the first, but is still a bug. I was looking through her code as to how she handled drawing and discarding cards. I found her code that allows her to discard a card and that can be seen below.

```
//Discard a card
public void discard(Card c)
{
    System.out.println(username + " discarded " + c);
    hand.remove(c);
    discard.add(c);
}
```

Clearly, her code doesn't have any conditionals setup for the necessary state of the hand and discard piles at the time of discarding. This function just takes the current state of the hand, and removes the passed card, and takes the current state of the discard and adds the passed card. I took this as a bug. I wrote a unit test that lets me generate a player, and lets me discard a card that I have not yet added to my hand. Her code allows me to remove from my hand a card that I don't have. My unit test and the successul output of the test can be seen below.

```java
public class findBugTest {
    @Test
    public void testDrawBug(){
        DominionBoard game = new DominionBoard(4);
        Player p = new Player("Player1", 0, game);
        p.discard(Card.copper);
        List<Card> testhand = p.getHand();
        List<Card> testdiscard = p.getDiscard();
        assertEquals(testhand.size(),0);
        assertEquals(testdiscard.size(), 11);
    }

}
```

```
<terminated> findBugTest [JUnit] C:\Program Files\Java\jdk1.8.0_121\bin\javaw.exe (Mar 19, 2017, 1:25:24 PM)
Player1 discarded copper
```

In my unit test, I used Junit asserts in order to test the size of my hand and the size of my discard pile after discarding a copper card. This test passes, implying the hand doesn't empty to below 0 and doesn't catch an error, and the discard size increases from 10 to 11. The problem with this implementation is that because copper was clearly never added to player P's hand in this unit test, the discard behavior shouldn't exist.

Like in Bug 1, the fix for this bug would be to add conditional statements within her code to make sure that the card being discarded is first in the player's hand. This doesn't ever result in an output that wouldn't compile, like Bug 1 does, but rather this bug is subtle and simply undesired behavior for what would be considered a sound discard functionality.