SWE265P Homework 6: Test Cases and Another Pull Request

Team pay4grade: Harry Duong, Thuc Nguyen, Deon Zhao

14 March 2020

1 Second Issue and Pull Request

Issue https://github.com/runelite/runelite/issues/10655

Pull request https://github.com/runelite/runelite/pull/10985

1.1 Report

The pull request concerns a faulty LootTracker, which is supposed to record loot received by the player from various in game activities, but isn't doing so when the player's inventory contains an open herb sack. Current implementation of LootTracker for the Herbiboar activity only detects changes in the inventory. With the introduction of the new item "Open Herb Sack," when the player harvests some herbs from Herbiboar, the herbs go straight into the herb sack, rather than appearing in the inventory, thereby not triggering an inventory change event breaking LootTracker. Furthermore, the next inventory change following the harvest event, even though unrelated to Herbiboar, will cause LootTracker to record the inventory change as loot from Herbiboar. Had the herbs entered the inventory, after which they could stay there or go straight into the sack, the LootTracker would record their presence per usual. The pull request fixes these issues by allowing LootTracker to track correct Herbiboar loot even in the presence of an open herb sack in the inventory. The pull request also includes 3 new test cases added to an existing LootTrackerPluginTest.java file, to verify that everything still functions. See Section 3 for the new test cases.

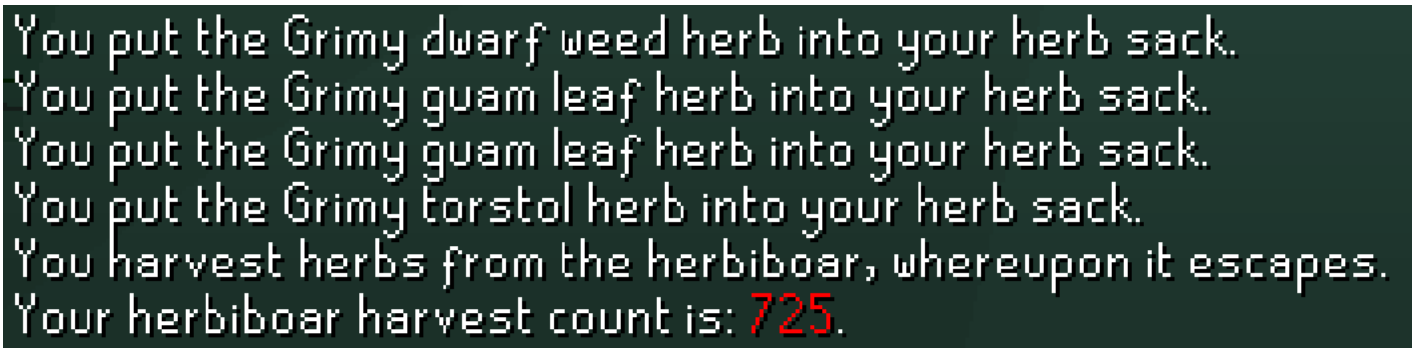


Figure 1: When players harvest Herbiboar with an open herb sack, the game will send a message for each herb that was harvested and automatically put into the herb sack, right before the original message that triggers the herbiboar event for the LootTracker.

```
2020-03-17 03:13:18 [Client] DEBUG net.runelite.client.RuneLite - Chat message type SPAM: You put the Grimy dwarf weed herb into your herb sack.
2020-03-17 03:13:18 [Client] DEBUG net.runelite.client.RuneLite - Chat message type SPAM: You put the Grimy guam leaf herb into your herb sack.
2020-03-17 03:13:18 [Client] DEBUG net.runelite.client.RuneLite - Chat message type SPAM: You put the Grimy guam leaf herb into your herb sack.
2020-03-17 03:13:18 [Client] DEBUG net.runelite.client.RuneLite - Chat message type SPAM: You put the Grimy torstol herb into your herb sack.
2020-03-17 03:13:18 [Client] DEBUG net.runelite.client.RuneLite - Chat message type GAMEMESSAGE: You harvest herbs from the herbiboar, whereupon it escapes.
2020-03-17 03:13:18 [Client] DEBUG n.r.c.plugins.examine.ExaminePlugin - Got examine without a pending examine?
2020-03-17 03:13:18 [Client] DEBUG net.runelite.client.RuneLite - Chat message type GAMEMESSAGE: Your herbiboar harvest count is: <col=ff0000>725</col>.
2020-03-17 03:13:18 [Client] DEBUG n.r.client.config.ConfigManager - Setting configuration value for killcount.deon9718.herbiboar to 725
```

Figure 2: Further investigation of these messages in the IntelliJ debug log shows that these messages are received at the same time as the trigger message.

```
private boolean processHerbiboarHerbSackLoot(int timestamp, String event, LootRecordType lootRecordType)
{
    List<ItemStack> herbs = new ArrayList<>();
    for (Object messageObject : client.getMessages())
    {
        MessageNode messageNode = (MessageNode) messageObject;
        if (messageNode.getTimestamp() != timestamp || messageNode.getType() != ChatMessageType.SPAM)
        {
            continue;
        }
        {
            Matcher matcher = HERBIBOAR_HERB_SACK_PATTERN.matcher(messageNode.getValue());
            if (matcher.matches())
            {
                herbs.add(new ItemStack(itemManager.search(matcher.group(1)).get(0).getId(),  quantity: 1, client.getLocalPlayer().getLocalLocation()));
            }
        }
    }

    if (herbs.isEmpty())
    {
        return false;
    }

    addLoot(event,  combatLevel: -1, lootRecordType, herbs);
    return true;
}
```

Figure 3: A new method called processHerbiboarHerbSackLoot() that contains the triggering message's timestamp as an argument was created to specifically process these messages from the game. We first get all messages that the client has received by calling client.getMessages(). We then loop through all messages, and filter out only the ones that have the exact timestamp and message type.

```
// Herbiboar loot handling
private static final String HERBIBOAR_LOOTED_MESSAGE = "You harvest herbs from the herbiboar, whereupon it escapes.";
private static final String HERBIBOAR_EVENT = "Herbiboar";
private static final Pattern HERBIBOAR_HERB_SACK_PATTERN = Pattern.compile(".+(Grimy .+?) herb.+");
```

Figure 4: Then the messages are matched to a regex pattern, and the herb name strings are extracted and searched via ItemManager to get their item ids.

```
if (message.equals(HERBIBOAR_LOOTED_MESSAGE))
{
    eventType = HERBIBOAR_EVENT;
    lootRecordType = LootRecordType.EVENT;
    if (processHerbiboarHerbSackLoot(event.getTimestamp(), eventType, lootRecordType))
    {
        eventType = null;
        lootRecordType = null;
    }
    else
    {
        takeInventorySnapshot();
    }
    return;
}
```

Figure 5: The item ids are then used to create ItemStacks, and all ItemStacks are put into a List. Finally the addLoot() method is called to update the LootTracker plugin, and a boolean is returned to inform the calling method onChatMessage() that if we processed any herb sack loot or not, so that it can continue on with the old way if we didn't process any loot.

1.2 Reflection

Coding up and solving the issue was at most 20% of the total work involved in this pull request. A lot more time was spent on discovering/reproducing the issue, making changes as requested by the code reviewer, and finally writing test cases. The responsiveness of the maintainers in their discord server helped tremendously in the entire process. For instance, we received a lot of input from Nightfirecat in regards to which classes we should mock and how we should parameterize our tests.

One thing we learned about the system was that a lot of things are triggered by @Subscribe. Things like onChatMessage(), or onItemContainerChanged() taught us a lot on how Runelite works in general, like the fact that onChatMessage() is listening in on every ChatMessage that is being sent from the game, even the ones that are explicitly filtered and unseen by the players. Another thing that we learned was how to implement the code that introduces the new feature in such a way that would mirror and work around the existing code base. For example, we had to know that the eventType that we were interested in was the HERBIBOAR_EVENT. Implementing it anywhere else wouldn't work. We also had to make sure to handle the false condition of processHerbiboarHerbSackLoot() by calling takeInventorySnapshot(). We would be introducing another bug if we didn't call takeInventorySnapshot(). All of the new code not only had to be woven into the existing codebase, it had to execute correctly and avoid introducing new bugs.

2 Existing Test Cases

2.1 Report

As we learned before, Plugins are an important class for Runelite and are what separate Runelite from other OSRS clients. So naturally, many, if not most, of the tests involve testing Plugins.

```
@Test
public void dumpGraph() throws Exception
{
    List<Module> modules = new ArrayList<>();
    modules.add(new GraphvizModule());
    modules.add(new RuneLiteModule(() -> null, developerMode: true));

    PluginManager pluginManager = new PluginManager( developerMode: true, eventBus: null, scheduler: null, configManager: null,
executor: null, sceneTileManager: null);
    pluginManager.loadCorePlugins();
    for (Plugin p : pluginManager.getPlugins())
    {
        modules.add(p);
    }

    File file = folder.newFile();
    try (PrintWriter out = new PrintWriter(file, csn: "UTF-8"))
    {
        Injector injector = Guice.createInjector(modules);
        GraphvizGrapher grapher = injector.getInstance(GraphvizGrapher.class);
        grapher.setOut(out);
        grapher.setRankdir("TB");
        grapher.graph(injector);
    }
}
```

Figure 6: The code for dumpGraph()

Existing Test Case 1 The first interesting test case we discovered is *PluginManagerTest.java* → **public void dumpGraph() throws Exception**. A strange thing occurs in this method - there is a try block without a catch block. In normal Java, a catch block must follow a try block. So maybe there is an additional context (presumably JUnit or the throws Exception clause) that allows for such a construct. When running PluginManagerTest.java, dumpGraph() indeed throws an exception - and the test passes. According to JUnit documentation, a (possibly custom) handler can handle exceptions during test execution. This test file indicates no such handler; maybe it inherits one such handler or the handler belongs to some other implicit context.

Existing Test Case 2 The second interesting test case is *PuzzleSolverTest.java* → **public void testSolverMM()**. When looking at this test file, the first thing that jumps out of you is arrays of hard-coded values. Runelite has a class called PuzzleState which appears to hold information about a 2D sliding puzzle. There are two test methods, testSolver() and testSolverMM(). What MM means is anybody's guess. In this file they have a similar implementation to each other. For both methods there is a corresponding array of PuzzleStates. PuzzleState's constructor takes an array of int values.

One more peculiarity is how the test methods check not only that the solver succeeded, but also that the solver didn't fail. If a puzzle solver solves the puzzle, then logically, the solver didn't fail unless the implementation allows for success and failure at the same time. In real life, of course, if a solver succeeds, then by definition it doesn't fail, at least in the same context as the success.

```
private static final PuzzleState[] START_STATES =
{
    new PuzzleState(new int[]{0, 11, 1, 3, 4, 5, 12, 2, 7, 9, 6, 20, 18, 16, 8, 15, 22, 10, 14, 13, 21, -1, 17, 23, 19}),
    new PuzzleState(new int[]{0, 2, 7, 3, 4, 10, 5, 12, 1, 9, 6, 17, 8, 14, 19, -1, 16, 21, 11, 13, 15, 20, 22, 18, 23}),
    new PuzzleState(new int[]{0, 1, 11, 3, 4, 12, 2, 7, 13, 9, 5, 21, 15, 17, 14, -1, 10, 6, 8, 19, 16, 20, 22, 18, 23}),
    new PuzzleState(new int[]{0, 1, 2, 3, 4, 10, 5, 6, 9, 14, 15, -1, 7, 13, 17, 21, 11, 20, 23, 8, 16, 22, 12, 19, 18}),
    new PuzzleState(new int[]{0, 1, 2, 3, 4, 5, 6, 8, 22, 18, 10, -1, 7, 17, 9, 20, 11, 12, 21, 14, 16, 15, 23, 13, 19}),
    new PuzzleState(new int[]{0, 1, 2, 3, 4, 5, 6, 8, 12, 9, 16, 11, 17, 7, 14, 10, 20, -1, 22, 13, 15, 18, 21, 23, 19}),
    new PuzzleState(new int[]{1, 6, 16, 8, 4, 0, 7, 11, 2, 9, 5, 21, 18, 3, 14, 10, 20, -1, 13, 22, 15, 23, 12, 17, 19}),
    new PuzzleState(new int[]{0, 1, 2, 4, 8, 6, 16, 11, 7, 3, 5, -1, 12, 14, 9, 15, 10, 17, 13, 18, 20, 21, 22, 23, 19}),
    new PuzzleState(new int[]{0, 2, 9, 14, 6, 5, 7, 11, 3, 4, 15, 10, 1, 12, 18, 16, 17, -1, 8, 13, 20, 21, 22, 23, 19}),
    new PuzzleState(new int[]{0, 1, 2, 3, 4, 11, 5, 12, 7, 8, 10, 6, 15, 13, 9, 16, 21, -1, 17, 14, 20, 22, 23, 18, 19}),
    new PuzzleState(new int[]{5, 0, 1, 2, 4, 10, 6, 3, 8, 9, 12, 13, 7, 14, 19, 15, 11, 16, 17, -1, 20, 21, 22, 18, 23}),
    new PuzzleState(new int[]{0, 6, 1, 3, 4, 5, 8, -1, 2, 9, 16, 11, 12, 7, 14, 10, 15, 17, 13, 19, 20, 21, 22, 18, 23}),
    new PuzzleState(new int[]{0, 6, 1, 2, 4, 11, 15, 8, 3, 14, 5, 7, 9, 12, 18, 16, 10, 17, 23, 13, 20, 21, 22, -1, 19}),
    new PuzzleState(new int[]{0, 1, 7, 2, 4, 5, 3, 12, 8, 9, 15, 6, 18, -1, 13, 11, 10, 22, 17, 23, 16, 21, 20, 19, 14}),
    new PuzzleState(new int[]{0, 1, 2, 7, 3, 5, 11, 6, 14, 4, 10, -1, 16, 12, 9, 15, 17, 18, 8, 19, 20, 21, 13, 22, 23}),
    new PuzzleState(new int[]{2, 10, 5, 3, 4, -1, 0, 1, 8, 9, 15, 11, 7, 13, 23, 17, 6, 20, 14, 19, 16, 12, 18, 21, 22}),
    new PuzzleState(new int[]{0, 1, 2, 8, 9, 5, 6, 7, 3, 4, 10, -1, 14, 23, 18, 21, 11, 16, 12, 19, 15, 20, 17, 13, 22}),
    new PuzzleState(new int[]{0, 6, 1, 3, 4, 11, 2, 13, 9, 12, 5, 16, 7, 18, 8, 20, 15, -1, 14, 19, 21, 10, 22, 23, 17}),
    new PuzzleState(new int[]{12, 1, 2, 3, 4, 0, 7, 6, 8, 9, 5, 10, 22, 13, 19, 15, 11, 21, 14, 17, 20, 16, 18, -1, 23}),
    new PuzzleState(new int[]{0, 2, 11, 3, 4, 5, 1, 6, 8, 9, 15, 10, 13, 14, 19, 7, 12, -1, 17, 18, 20, 21, 16, 22, 23}),
    new PuzzleState(new int[]{5, 0, 4, 2, 9, 10, 7, 3, 19, 8, 6, 1, 18, -1, 14, 15, 11, 16, 12, 13, 20, 21, 17, 22, 23}),
    new PuzzleState(new int[]{0, 3, 2, 7, 4, 6, 10, 1, 8, 9, 15, 5, 12, 18, 13, -1, 20, 11, 22, 14, 16, 21, 23, 17, 19}),
    new PuzzleState(new int[]{1, 2, 4, -1, 9, 0, 5, 7, 3, 14, 10, 6, 8, 13, 19, 15, 11, 18, 12, 22, 20, 16, 21, 23, 17}),
    new PuzzleState(new int[]{0, 1, 2, 4, 9, 5, 11, -1, 7, 14, 10, 17, 6, 13, 8, 15, 16, 20, 3, 18, 22, 21, 12, 23, 19}),
    new PuzzleState(new int[]{0, 1, 8, 2, 4, 5, 11, 17, 3, 9, 6, 16, 7, 12, 18, 15, 21, -1, 14, 13, 20, 22, 10, 23, 19}),
    new PuzzleState(new int[]{5, 0, 2, 3, 4, 1, 8, 6, 7, 9, 11, 12, 16, 13, 14, -1, 22, 20, 17, 19, 21, 10, 15, 18, 23}),
    new PuzzleState(new int[]{0, -1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 16, 11, 13, 14, 10, 15, 19, 22, 23, 20, 21, 17, 12, 18}),
    new PuzzleState(new int[]{0, 2, 7, -1, 4, 6, 1, 9, 3, 14, 5, 12, 8, 13, 19, 15, 16, 10, 22, 23, 20, 11, 18, 21, 17}),
    new PuzzleState(new int[]{7, 5, 13, 1, 12, 0, 2, 4, 3, 9, 10, 6, 8, 17, 14, 15, 11, 16, 18, -1, 20, 21, 22, 23, 19}),
    new PuzzleState(new int[]{5, 0, 3, 8, 4, 10, 6, -1, 7, 9, 11, 1, 12, 2, 19, 15, 16, 17, 14, 13, 20, 21, 22, 18, 23}),
    new PuzzleState(new int[]{5, 2, 3, 7, 4, 0, 6, 14, 9, 19, 1, 11, 22, 17, 12, 10, 15, -1, 13, 8, 20, 16, 21, 18, 23}),
    new PuzzleState(new int[]{0, 1, 3, 4, 9, 5, 6, 2, 7, 14, 10, 13, 17, -1, 8, 15, 11, 16, 12, 18, 20, 21, 22, 23, 19}),
    new PuzzleState(new int[]{0, 3, 11, 7, 4, 5, 2, 6, 12, 8, 10, 1, 17, -1, 9, 15, 16, 23, 13, 14, 20, 21, 18, 22, 19}),
    new PuzzleState(new int[]{1, 5, 8, 2, 4, -1, 0, 7, 3, 9, 11, 22, 15, 12, 14, 6, 10, 18, 16, 19, 20, 21, 17, 13, 23}),
    new PuzzleState(new int[]{7, 12, 11, 4, 9, -1, 0, 8, 10, 2, 6, 1, 16, 3, 14, 5, 15, 17, 13, 19, 20, 21, 22, 18, 23}),
    new PuzzleState(new int[]{11, 3, 2, 12, 4, 6, 0, 7, 13, 8, 1, 5, 17, 16, 9, -1, 10, 15, 18, 14, 20, 21, 22, 23, 19}),
    new PuzzleState(new int[]{0, 6, 1, 3, 4, 5, 11, 2, 10, 9, 15, 12, 8, 14, 19, 16, 21, -1, 7, 13, 20, 22, 17, 18, 23}),
    new PuzzleState(new int[]{0, 1, 2, 3, 4, 6, 10, 7, 8, 9, 5, 16, 11, 14, 17, 20, 13, 18, 12, 22, 21, 15, 23, -1, 19}),
    new PuzzleState(new int[]{0, 1, 2, 4, 8, 5, 6, 7, 3, -1, 10, 16, 18, 17, 9, 15, 12, 11, 14, 13, 20, 21, 22, 23, 19}),
    new PuzzleState(new int[]{0, 11, 6, 1, 4, 5, 21, 8, 2, 9, 10, 3, 16, -1, 14, 15, 12, 17, 13, 19, 20, 22, 7, 18, 23}),
    new PuzzleState(new int[]{0, 6, 1, 2, 4, 11, 10, 3, 13, 9, 5, 7, 8, -1, 23, 15, 16, 22, 18, 14, 20, 21, 12, 17, 19}),
    new PuzzleState(new int[]{0, 6, 1, 2, 3, 10, 11, 12, 5, 18, 15, 7, 4, -1, 14, 21, 17, 13, 8, 9, 16, 20, 22, 23, 19}),
    new PuzzleState(new int[]{0, 1, 3, 11, 4, 6, 10, 14, 2, 8, 5, -1, 12, 7, 9, 15, 16, 18, 13, 19, 20, 21, 22, 17, 23}),
    new PuzzleState(new int[]{1, 5, 2, 3, 4, -1, 0, 7, 14, 8, 11, 6, 13, 9, 23, 10, 12, 15, 19, 17, 20, 21, 16, 22, 18}),
};
```

Figure 7: Initializing a PuzzleState array

```
private static final PuzzleState[] START_STATES_MM =
{
    new PuzzleState(new int[]{0, 5, 1, 3, 4, 15, 2, 8, 10, 9, 22, 16, 11, 6, 14, 7, 21, 23, 12, 18, 20, -1, 17, 13, 19}),
    new PuzzleState(new int[]{0, 12, 8, 13, 4, 3, 16, 2, 1, 9, 21, 5, 6, 10, 14, 7, 17, 20, 18, -1, 15, 11, 22, 23, 19}),
    new PuzzleState(new int[]{1, 3, 7, 9, 8, 13, 17, 2, 4, 6, 15, 5, 22, 12, 14, 11, 0, 10, 21, -1, 20, 18, 16, 23, 19}),
    new PuzzleState(new int[]{5, 2, 16, 14, 4, 3, 0, 8, 9, 11, 15, 6, 17, 19, 7, 1, 10, -1, 23, 18, 20, 12, 21, 13, 22}),
    new PuzzleState(new int[]{0, 6, 1, 4, 13, 10, 2, 16, 7, 3, 20, -1, 8, 14, 9, 21, 5, 18, 11, 19, 17, 15, 12, 22, 23}),
    new PuzzleState(new int[]{5, 0, 1, 4, 8, 10, 6, 7, 12, 3, 17, 16, 21, 2, 9, 18, 20, 13, 14, 19, 11, -1, 23, 15, 22}),
    new PuzzleState(new int[]{1, 9, 2, 13, 17, 5, 7, 8, 3, 22, 6, -1, 16, 12, 4, 15, 18, 0, 23, 14, 10, 21, 11, 20, 19}),
    new PuzzleState(new int[]{1, 2, 11, 13, 4, 21, 7, 3, 6, 9, 0, 8, 10, 19, 14, 20, 12, 16, 23, -1, 5, 17, 15, 22, 18}),
    new PuzzleState(new int[]{2, 0, 1, 4, 13, 6, 7, 3, 8, 9, 22, 15, 10, 14, 18, 5, 12, -1, 17, 21, 20, 11, 23, 16, 19}),
    new PuzzleState(new int[]{0, 1, 2, 8, 3, 6, 12, 22, 9, 7, 11, 21, 13, 4, 14, 5, 10, -1, 18, 19, 20, 15, 16, 23, 17}),
    new PuzzleState(new int[]{1, 2, 3, 4, 8, 0, 6, 15, 14, 18, 16, 17, 20, -1, 9, 10, 12, 22, 11, 13, 21, 7, 5, 19, 23}),
    new PuzzleState(new int[]{0, 5, 2, 4, 9, 7, 15, 20, 12, 13, 6, -1, 22, 1, 8, 10, 11, 23, 14, 3, 21, 16, 17, 19, 18}),
    new PuzzleState(new int[]{0, 1, 9, 6, 13, 5, 18, -1, 4, 2, 15, 12, 3, 17, 7, 16, 10, 8, 23, 14, 20, 21, 19, 11, 22}),
    new PuzzleState(new int[]{11, 5, 12, 3, 4, 15, 8, 0, 7, 1, 6, -1, 19, 2, 9, 16, 10, 13, 17, 23, 20, 21, 22, 14, 18}),
    new PuzzleState(new int[]{10, 0, 1, 3, 4, 18, 5, 6, 12, 9, 7, 11, 8, -1, 22, 15, 23, 14, 19, 13, 20, 2, 17, 16, 21}),
    new PuzzleState(new int[]{19, -1, 6, 2, 4, 0, 21, 10, 3, 9, 1, 15, 17, 8, 14, 11, 13, 22, 7, 18, 16, 12, 5, 23, 20}),
    new PuzzleState(new int[]{11, 6, 3, 4, 9, 1, 10, 16, 2, 7, 5, 0, 13, -1, 12, 21, 8, 18, 17, 14, 15, 20, 22, 23, 19}),
    new PuzzleState(new int[]{0, 1, 5, 3, 4, -1, 6, 2, 15, 10, 7, 8, 23, 16, 13, 22, 11, 9, 12, 14, 20, 21, 18, 17, 19}),
    new PuzzleState(new int[]{10, 0, 1, -1, 2, 6, 5, 4, 13, 9, 16, 17, 12, 8, 19, 20, 15, 7, 21, 11, 22, 18, 14, 23, 3}),
    new PuzzleState(new int[]{1, 0, 5, 3, 9, 20, 15, 7, 2, 14, 6, 4, 12, -1, 8, 13, 18, 10, 23, 11, 21, 16, 17, 19, 22}),
    new PuzzleState(new int[]{0, 7, 6, 3, 4, 15, 1, 2, 8, 18, 11, 5, 13, -1, 22, 17, 16, 23, 14, 9, 20, 10, 12, 19, 21}),
    new PuzzleState(new int[]{5, 7, 0, 2, 9, 10, 1, 11, 3, 4, 16, 22, 8, 14, 17, 15, 20, 12, 13, 6, 21, 23, 19, -1, 18}),
    new PuzzleState(new int[]{3, 0, 1, 5, 4, 11, 6, 2, 16, 9, 15, 10, 7, 12, 13, 21, 19, -1, 22, 8, 20, 17, 14, 18, 23}),
    new PuzzleState(new int[]{6, 0, 3, 2, 4, 5, 1, 8, 13, 12, 15, 14, 10, 7, 9, -1, 22, 11, 19, 23, 16, 20, 17, 21, 18}),
    new PuzzleState(new int[]{11, 5, 6, 8, 9, 0, 21, 16, 4, 3, 17, 18, 2, 7, 1, 15, 10, -1, 22, 14, 20, 19, 13, 12, 23}),
    new PuzzleState(new int[]{2, 18, 3, 11, 4, -1, 5, 6, 12, 1, 10, 20, 0, 7, 9, 21, 15, 14, 23, 19, 16, 22, 13, 8, 17}),
    new PuzzleState(new int[]{0, 6, 8, 3, 1, 5, 2, 12, 9, 13, 16, 14, 19, 7, 18, 10, 11, -1, 4, 15, 20, 17, 23, 21, 22}),
    new PuzzleState(new int[]{1, 16, 10, 4, 3, 0, 15, 2, 9, -1, 8, 5, 23, 12, 6, 21, 18, 14, 13, 11, 20, 22, 7, 19, 17}),
    new PuzzleState(new int[]{1, 7, 6, 3, 4, 0, 2, 14, 5, 22, 18, 21, 16, 9, 13, 10, 20, -1, 8, 17, 15, 23, 11, 19, 12}),
    new PuzzleState(new int[]{5, 0, 1, 7, 9, 11, 8, 4, 2, 14, 15, 17, 18, -1, 3, 20, 10, 12, 22, 19, 16, 6, 13, 21, 23}),
    new PuzzleState(new int[]{5, 0, 6, 14, 7, 13, 15, 1, 3, 10, 20, 9, 17, 4, 2, 11, 12, 8, 19, -1, 21, 16, 22, 18, 23}),
    new PuzzleState(new int[]{12, 7, 8, 4, 9, 6, 11, 15, 2, 1, 5, -1, 13, 16, 3, 17, 0, 10, 18, 14, 20, 22, 21, 19, 23}),
    new PuzzleState(new int[]{15, 1, 2, 3, 14, -1, 20, 9, 4, 19, 0, 6, 7, 16, 13, 10, 5, 12, 17, 18, 22, 11, 21, 23, 8}),
    new PuzzleState(new int[]{0, 1, 17, -1, 14, 6, 4, 2, 3, 16, 10, 18, 13, 19, 9, 7, 5, 8, 21, 22, 11, 20, 15, 12, 23}),
    new PuzzleState(new int[]{5, 11, 9, 0, 3, 8, 14, -1, 6, 4, 1, 13, 7, 2, 19, 10, 21, 18, 23, 17, 15, 20, 12, 16, 22}),
    new PuzzleState(new int[]{2, 0, 14, -1, 4, 18, 1, 10, 12, 13, 5, 9, 11, 22, 7, 15, 8, 17, 19, 3, 20, 21, 6, 16, 23}),
    new PuzzleState(new int[]{0, 1, 13, 9, 2, 6, 8, 22, 3, 4, 12, 16, 10, 7, 19, -1, 5, 11, 14, 17, 15, 20, 21, 18, 23}),
    new PuzzleState(new int[]{0, 13, 17, 8, 3, 5, 1, 12, 14, 4, 10, -1, 6, 7, 9, 15, 23, 2, 16, 19, 20, 11, 21, 22, 18}),
    new PuzzleState(new int[]{5, 10, 7, 2, 9, 15, 0, -1, 1, 3, 18, 4, 17, 12, 14, 21, 11, 6, 8, 23, 20, 16, 22, 19, 13}),
    new PuzzleState(new int[]{0, 3, 1, 2, 4, 10, 5, 7, 8, 9, 11, 6, 21, 13, 12, 20, 17, -1, 14, 19, 22, 18, 15, 16, 23}),
    new PuzzleState(new int[]{0, 2, 7, 11, 13, 3, 14, 1, 4, 9, 5, -1, 12, 8, 18, 20, 10, 15, 22, 23, 17, 16, 6, 21, 19}),
    new PuzzleState(new int[]{0, 16, 3, 22, 7, 11, 6, -1, 9, 4, 2, 1, 13, 12, 18, 5, 10, 8, 19, 14, 15, 20, 17, 23, 21}),
    new PuzzleState(new int[]{0, 13, 5, 12, 3, 2, 10, 4, 6, 8, 1, 21, 19, 14, 9, 17, 23, 22, 16, 11, 15, 7, 20, -1, 18}),
    new PuzzleState(new int[]{14, 5, 6, 12, 4, 10, 20, 1, 0, 23, 2, 16, 13, 19, 3, 15, 22, -1, 9, 8, 11, 7, 18, 17, 21}),
    new PuzzleState(new int[]{0, 1, 2, 4, 7, 5, 11, -1, 18, 8, 16, 10, 12, 13, 3, 17, 6, 21, 23, 9, 15, 20, 22, 14, 19}),
    new PuzzleState(new int[]{1, 6, 7, 3, 4, 5, 17, 0, 22, 12, 10, 15, 8, -1, 14, 11, 13, 16, 18, 19, 20, 2, 21, 9, 23}),
```

Figure 8: Initializing a second PuzzleState array

```
    private static final int[] FINISHED_STATE = new int[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, -1};

    @Test
    public void testSolverMM()
    {
        for (PuzzleState state : START_STATES_MM)
        {
            PuzzleSolver solver = new PuzzleSolver(new IDAStarMM(new ManhattanDistance()), state);
            solver.run();

            assertTrue(solver.hasSolution());
            assertFalse(solver.hasFailed());
            assertTrue(solver.getStep( stepIdx: solver.getStepCount() - 1).hasPieces(FINISHED_STATE));
        }
    }

    @Test
    public void testSolver()
    {
        for (PuzzleState state : START_STATES)
        {
            PuzzleSolver solver = new PuzzleSolver(new IDAStar(new ManhattanDistance()), state);
            solver.run();

            assertTrue(solver.hasSolution());
            assertFalse(solver.hasFailed());
            assertTrue(solver.getStep( stepIdx: solver.getStepCount() - 1).hasPieces(FINISHED_STATE));
        }
    }
}
```

Figure 9: Initializing a solution (the array **FINISHED_STATE** at the top) for a PuzzleSolver; and the two tests for the corresponding PuzzleState arrays in the previous figure

Existing Test Case 3 The third interesting test case is *ClockPanelTest.java* → **incorrectTimeStringShouldThrowException()**. There is a lack of uniformity within the game; the game uses regular expressions to select an appropriate time format. There are three test methods in this test file, all which use ClockPanel.stringToSeconds(String time) throws NumberFormatException, DateTimeParseException. properColonSeparatedTimeStringShouldReturnCorrectSeconds() focuses on colon-separated time formats, while the second, properColonSeparatedTimeStringShouldReturnCorrectSeconds(), focuses on letter-separated times with the letters h, m, or s. The third is the incorrectTimeStringShouldThrowException() method. Elsewhere in the client, a ClockPanel can display different time formats, possibly for different regions or some other context. Having to display the time is presumably the reason why the game doesn't use one, unified format. If the game were to track time internally instead, it may very well stick to one format, use a built-in time library or similar, or just outright use primitive numeric types to represent seconds.

Not only does the test check for a wrong time format, it checks for the type of the exception that gets thrown for said time format. ClockPanel.stringToSeconds() throws one of two exceptions, either NumberFormatException or DateTimeParseException. This is an extra checking mechanism to make sure that Java throws the right exception because stringToSeconds() does not explicitly throw the exception; Java does.

```java
public class ClockPanelTest
{
    @Test
    public void properColonSeparatedTimeStringShouldReturnCorrectSeconds()
    {
        assertEquals( expected: 5, ClockPanel.stringToSeconds( time: "5"));
        assertEquals( expected: 50, ClockPanel.stringToSeconds( time: "50"));
        assertEquals( expected: 120, ClockPanel.stringToSeconds( time: "2:00"));
        assertEquals( expected: 120, ClockPanel.stringToSeconds( time: "0:120"));
        assertEquals( expected: 120, ClockPanel.stringToSeconds( time: "0:0:120"));
        assertEquals( expected: 1200, ClockPanel.stringToSeconds( time: "20:00"));

        assertEquals( expected: 50, ClockPanel.stringToSeconds( time: "00:00:50"));
        assertEquals( expected: 121, ClockPanel.stringToSeconds( time: "00:02:01"));
        assertEquals( expected: 3660, ClockPanel.stringToSeconds( time: "01:01:00"));
        assertEquals( expected: 9000, ClockPanel.stringToSeconds( time: "2:30:00"));
        assertEquals( expected: 9033, ClockPanel.stringToSeconds( time: "02:30:33"));
        assertEquals( expected: 82800, ClockPanel.stringToSeconds( time: "23:00:00"));
        assertEquals( expected: 400271, ClockPanel.stringToSeconds( time: "111:11:11"));
    }

    @Test
    public void properIntuitiveTimeStringShouldReturnCorrectSeconds()
    {
        assertEquals( expected: 5, ClockPanel.stringToSeconds( time: "5s"));
        assertEquals( expected: 50, ClockPanel.stringToSeconds( time: "50s"));
        assertEquals( expected: 120, ClockPanel.stringToSeconds( time: "2m"));
        assertEquals( expected: 120, ClockPanel.stringToSeconds( time: "120s"));
        assertEquals( expected: 1200, ClockPanel.stringToSeconds( time: "20m"));

        assertEquals( expected: 121, ClockPanel.stringToSeconds( time: "2m1s"));
        assertEquals( expected: 121, ClockPanel.stringToSeconds( time: "2m     1s"));
        assertEquals( expected: 3660, ClockPanel.stringToSeconds( time: "1h 1m"));
        assertEquals( expected: 3660, ClockPanel.stringToSeconds( time: "61m"));
        assertEquals( expected: 3660, ClockPanel.stringToSeconds( time: "3660s"));
        assertEquals( expected: 9000, ClockPanel.stringToSeconds( time: "2h 30m"));
        assertEquals( expected: 9033, ClockPanel.stringToSeconds( time: "2h 30m 33s"));
        assertEquals( expected: 82800, ClockPanel.stringToSeconds( time: "23h"));
        assertEquals( expected: 400271, ClockPanel.stringToSeconds( time: "111h 11m 11s"));
    }

    @Test
    public void incorrectTimeStringShouldThrowException()
    {
        Class numberEx = NumberFormatException.class;
        Class dateTimeEx = DateTimeParseException.class;

        tryFail( input: "a", numberEx);
        tryFail( input: "abc", numberEx);
        tryFail( input: "aa:bb:cc", numberEx);
        tryFail( input: "01:12=", numberEx);

        tryFail( input: "s", dateTimeEx);
        tryFail( input: "1s 1m", dateTimeEx);
        tryFail( input: "20m:10s", dateTimeEx);
        tryFail( input: "20hh10m10s", dateTimeEx);
    }

    private void tryFail(String input, Class<?> expectedException)
    {
        try
        {
            ClockPanel.stringToSeconds(input);
            fail("Should have thrown " + expectedException.getSimpleName());
        }
        catch (Exception exception)
        {
            assertEquals(expectedException, exception.getClass());
        }
    }
}
```

Figure 10: *ClockPanelText.java*. The method of interest is the third one, **incorrectTimeStringShouldThrowException()**, which in contrast to the two methods that precede it, focus on incorrect times and checking the type of exception Java throws.

```java
static long stringToSeconds(String time) throws NumberFormatException, DateTimeParseException
{
    long duration = 0;

    if (time.matches(INPUT_HMS_REGEX))
    {
        String textWithoutWhitespaces = time.replaceAll(WHITESPACE_REGEX, replacement: "");
        //parse input using ISO-8601 Duration format (e.g. 'PT1h30m10s')
        duration = Duration.parse("PT" + textWithoutWhitespaces).toMillis() / 1000;
    }
    else
    {
        String[] parts = time.split( regex: ":");
        // parse from back to front, so as to accept hour:min:sec, min:sec, and sec formats
        for (int i = parts.length - 1, multiplier = 1; i >= 0 && multiplier <= 3600; i--, multiplier *= 60)
        {
            duration += Integer.parseInt(parts[i].trim()) * multiplier;
        }
    }

    return duration;
}
```

Figure 11: The implementation for stringToSeconds(String time), a method which throws two Exceptions and is key to the tests in the previous figure.

2.2 Reflection

With a large code base, especially a game with a complex GUI, testing seems daunting. Mockito helps ease making dependencies a lot, and many of the test cases bundled with the code are actually quite simple. Maybe the developers decided that baby steps are all they need. Testing the GUI or other more complex parts of the code might be elsewhere or use more complicated testing frameworks than JUnit.

3 New Test Cases

3.1 Report

Three new test cases were written that covered the new functionality introduced by our pull request. The new functionality dealt with fixing a bug that was previously mentioned in the report about our pull request: the loot tracker is now able to display the items that were gathered from the herbiboar when the herbs are transferred to an open herb sack. We wrote test cases that covered new functionality by using assertions and by verifying execution of methods.

New Test Case 1 The first test case, **testHerbiboar()**, tested that the eventType property of the LootTrackerPlugin instance returned the expected string "Herbiboar", which represented the name of the event. This test also asserted that the lootRecordType property of the LootTrackerPlugin matched a LootRecordType of EVENT (Figure 1). Prior to the assertions, we mocked both the IteratorHashTable and the Iterator class so that when messages are received from client.getMessages(), the mocked message table would be returned rather than actual game messages. We specified that the mocked iterator returns nothing because we don't need any messages to be returned. Instead, we only want to assert that the eventType and lootRecordType are set to expected values when onChatMessage() is fired from the loot-TrackerPlugin. Basically, we just wanted to check that the herb-affiliated event was triggered when a chat

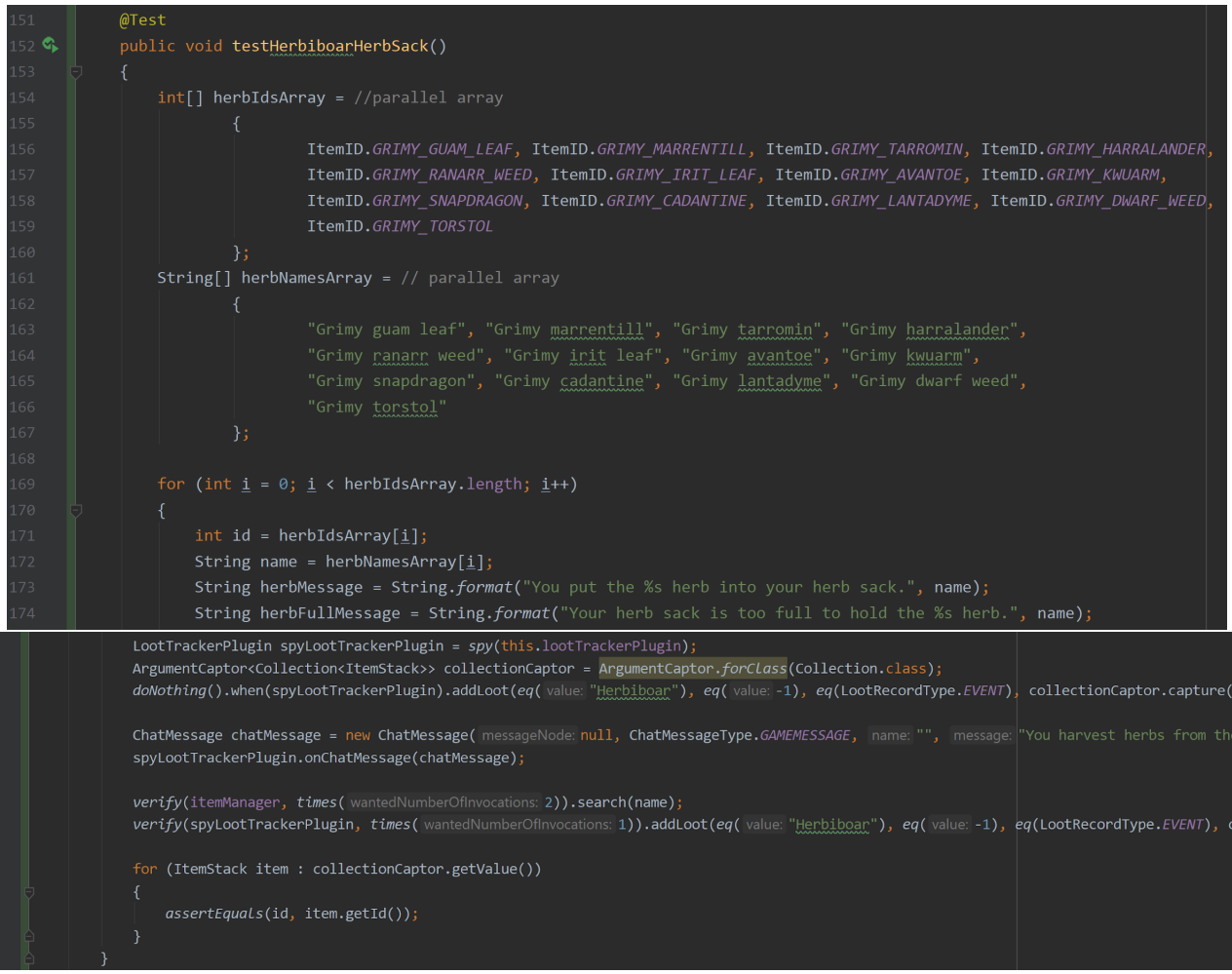
message corresponding to that herb event was sent.

```
135     @Test
136     public void testHerbiboar()
137     {
138         IterableHashTable messageTable = mock(IterableHashTable.class);
139         Iterator<Object> mockIterator = mock(Iterator.class);
140         when(mockIterator.hasNext()).thenReturn(false);
141         when(messageTable.iterator()).thenReturn(mockIterator);
142         when(client.getMessages()).thenReturn(messageTable);
143         ChatMessage chatMessage = new ChatMessage( messageNode: null, ChatMessageType.GAMEMESSAGE, name: "",
144                 message: "You harvest herbs from the herbiboar, whereupon it escapes.", sender: "", timestamp: 0);
145         lootTrackerPlugin.onChatMessage(chatMessage);
146
147         assertEquals( expected: "Herbiboar", lootTrackerPlugin.eventType);
148         assertEquals(LootRecordType.EVENT, lootTrackerPlugin.lootRecordType);
149     }
```

Figure 12: testHerbiboar() in LootTrackerPluginTest.java; displays the mocked classes, the ChatMessage with the appropriate event message, and the assertions on the eventType and lootRecordType properties.

New Test Case 2 The purpose of this test was to check that the new functionality from the code we added actually executed. The two main Runelite methods that ascertained that our functionality was used were the **search()** and **addLoot()** methods of classes ItemManager and LootTrackerPlugin, respectively. The search() method is fired when the Herbiboar event is sent and filters through chat messages using a string and a timestamp to return messages affiliated with the Herbiboar event. The addLoot() method is called after the chat messages are filtered and the associated herb items are gathered in a list. The method uses this list of herbs and adds it to the player's loot. We had to verify that the search() method was called twice (once for the herbMessage string and another for the herbFullMessage string) and that addLoot() was called once. By doing so, we know for certain that our code was executed because there wouldn't be any other interactions with the ItemManager or LootTrackerPlugin during the test run.

One thing to note was that we ended up using a spy on the actual instance of the LootTrackerPlugin that the test file was using. We had to use a spy because addLoot() would call downstream private methods that we had no control over. When addLoot() was called on the spy, we would tell addLoot() to do nothing. We only wanted to pass it the arguments the method was expecting, which included an ArgumentCaptor on a Collection of ItemStack, the last parameter of addLoot(). This last parameter allows us to capture the actual ItemStack object when addLoot() is called, so that we can ultimately extract out the item's id and assert it against the herb id.



```
151     @Test
152     public void testHerbiboarHerbSack()
153     {
154         int[] herbIdsArray = //parallel array
155                 {
156                         ItemID.GRIMY_GUAM_LEAF, ItemID.GRIMY_MARRENTILL, ItemID.GRIMY_TARROMIN, ItemID.GRIMY_HARRALANDER,
157                         ItemID.GRIMY_RANARR_WEED, ItemID.GRIMY_IRIT_LEAF, ItemID.GRIMY_AVANTOE, ItemID.GRIMY_KWUARM,
158                         ItemID.GRIMY_SNAPDRAGON, ItemID.GRIMY_CADANTINE, ItemID.GRIMY_LANTADYME, ItemID.GRIMY_DWARF_WEED,
159                         ItemID.GRIMY_TORSTOL
160                 };
161         String[] herbNamesArray = // parallel array
162                 {
163                         "Grimy guam leaf", "Grimy marrentill", "Grimy tarromin", "Grimy harralander",
164                         "Grimy ranarr weed", "Grimy irit leaf", "Grimy avantoe", "Grimy kwuarm",
165                         "Grimy snapdragon", "Grimy cadantine", "Grimy lantadyme", "Grimy dwarf weed",
166                         "Grimy torstol"
167                 };
168
169         for (int i = 0; i < herbIdsArray.length; i++)
170         {
171             int id = herbIdsArray[i];
172             String name = herbNamesArray[i];
173             String herbMessage = String.format("You put the %s herb into your herb sack.", name);
174             String herbFullMessage = String.format("Your herb sack is too full to hold the %s herb.", name);
```

```
198         LootTrackerPlugin spyLootTrackerPlugin = spy(this.lootTrackerPlugin);
199         ArgumentCaptor<Collection<ItemStack>> collectionCaptor = ArgumentCaptor.forClass(Collection.class);
200         doNothing().when(spyLootTrackerPlugin).addLoot(eq( value: "Herbiboar"), eq( value: -1), eq(LootRecordType.EVENT), collectionCaptor.capture());
201
202         ChatMessage chatMessage = new ChatMessage( messageNode: null, ChatMessageType.GAMEMESSAGE, name: "", message: "You harvest herbs from the her
203         spyLootTrackerPlugin.onChatMessage(chatMessage);
204
205         verify(itemManager, times( wantedNumberOfInvocations: 2)).search(name);
206         verify(spyLootTrackerPlugin, times( wantedNumberOfInvocations: 1)).addLoot(eq( value: "Herbiboar"), eq( value: -1), eq(LootRecordType.EVENT), colle
207
208         for (ItemStack item : collectionCaptor.getValue())
209         {
210             assertEquals(id, item.getId());
211         }
212     }
```

Figure 13: testHerbiboarHerbSack() in LootTrackerPluginTest.java; the first image shows the use of parallel arrays to iterate over each pair of herb ids and names to test. The second image shows that a spy is wrapped on lootTrackerPlugin so that addLoot does nothing (line 200), and the verifications of search() and addLoot().

New Test Case 3 The last test we wrote was called testHerbiboarPotionBug(), which asserted that the method getItems() which returned items in the inventory, was not called (Figure 3). We emulated the changing of items by initially specifying that the player has a STAMINA_POTION2 potion in the inventory and that the player's inventory has changed by calling onItemContainerChanged() and passing STAMINA_POTION1. This series of steps emulates when a player has a potion, takes a sip of it, and the potion has essentially changed form, thus triggering the onItemContainerChanged() method. Originally, there was a bug in the loot tracker that would show the potion as part of the herbs that were gathered when harvesting from Herbiboars (https://github.com/runelite/runelite/issues/10718). However since our fix, this was no longer a problem. Thus, we had to prove that this interaction with the potion being added when it wasn't supposed to no longer occurred. This is done by mocking the ItemContainer class, stubbing the getItems() call to return the expected items, and using Mockito.lenient() on the when-thenReturn statement. With the bug, the when-thenReturn statements on ItemContainer.getItems() would actually run; the stubs are used. With the bug fixed, these stubbed statements are no longer used and IntelliJ will throw an error. The use of lenient() here allows the errors to be suppressed, proving that these stubs associated with the items in the form of potions are no longer used and getItems() is no longer called.

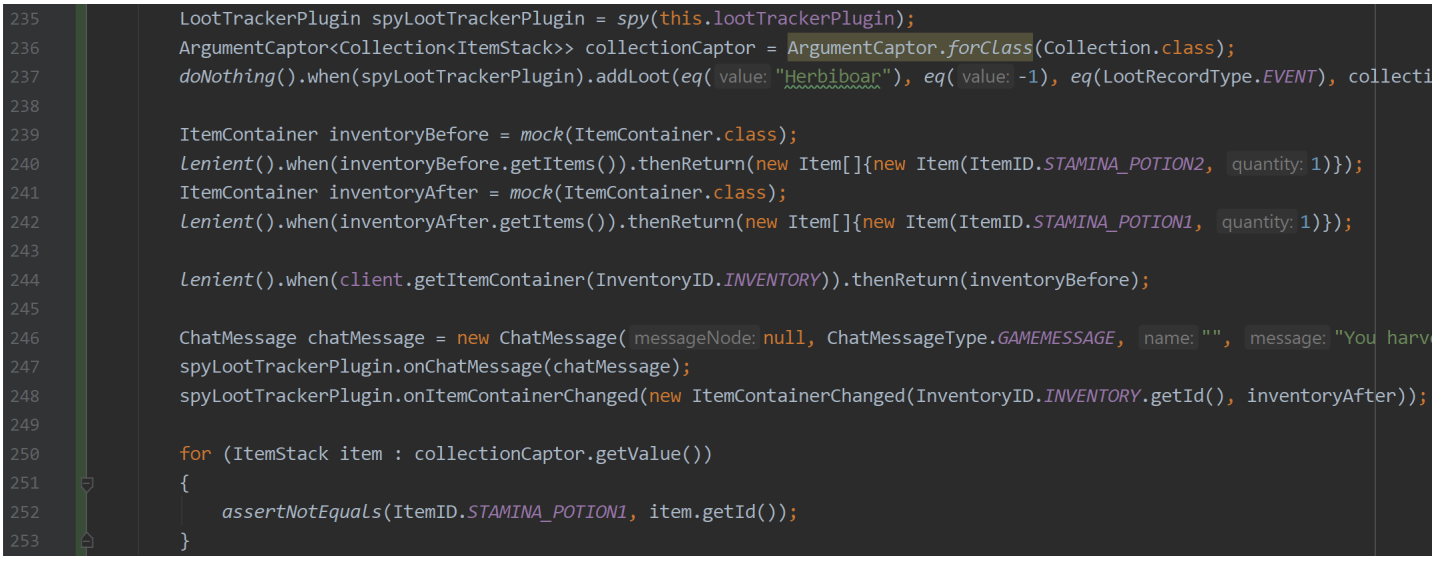


Figure 14: testHerbiboarPotionBug() in LootTrackerPlugin.java; with the presence of the bug, the stubs for getItems() within the when-thenReturn statements (lines 240 and 242) will be used. With the bug fixed, these statements will throw an unused stubs error, which is suppressed using lenient() here.

3.2 Reflection

This experience with formulating new test cases was definitely difficult. We had to first piece together what would be a representative test for the new functionality. What would we have to check and verify in order to prove that the code we wrote actually executed and returned expected values given known inputs? For the first test, we learned that asserting against the eventType and lootRecordType would prove that the appropriate Herbiboar event was actually sent. For the second test, we pinpointed the search() method of ItemManager and the addLoot() method of LootTrackerPlugin to be the methods to verify. For the third test, we confirmed that a previous interaction (getItems() of ItemContainer) in the presence of a bug no longer existed after the bug was fixed by our added functionality. By targeting specific variables and methods, mocking and spying associated classes, and asserting values and verifying interactions between components, we were able to write correct tests that solidified the validity of the new functionality.