Abraham Julian Santosa
Guillermo Haro
Ronald Reyes
Comp 587

Testing BuildCraft

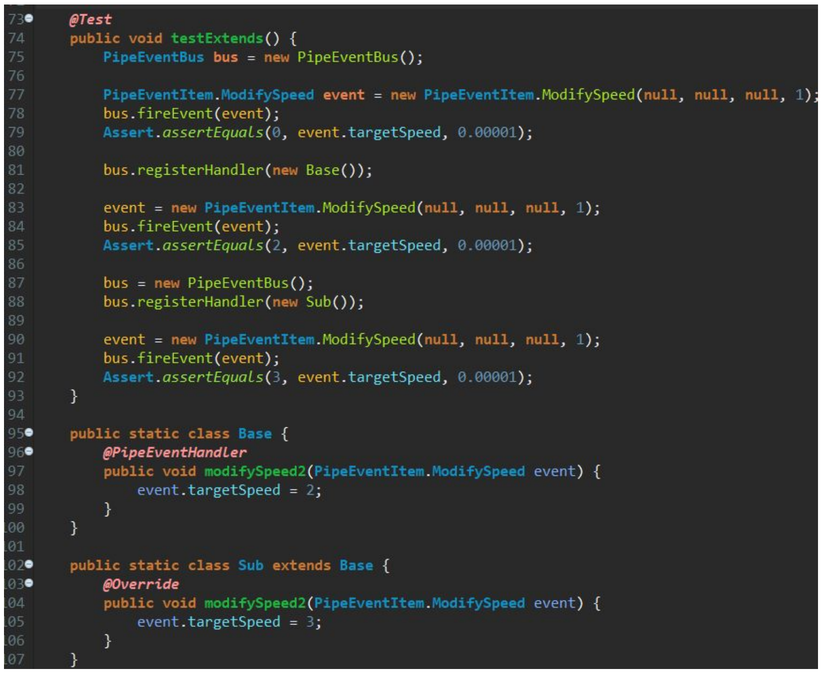
Unit Testing

BuildCraft was a bit tricky when it came to testing. We had to use gradle due to maven not being functional with BuildCraft. It was a small adaptation we had to do. We focused on testing the transport.pipe.api, ItemTransactorTester.java, GuiTester.java, BoxTester . All our tests can be found on github through the following link.
https://github.com/csun-comp587-s20/BuildCraft/commits/master

Here are some of the examples of our unit tests:

Transport pipe api: Unit testing transport.pipe.api is a bit tricky, since it needs to have a sequence of events before we can test the output. For example testExtends(), before I can check the speed I needed to create the event and the bus and then fire the event from the bus.

Testfile link: link



```java
    @Test
    public void testExtends() {
        PipeEventBus bus = new PipeEventBus();

        PipeEventItem.ModifySpeed event = new PipeEventItem.ModifySpeed(null, null, null, 1);
        bus.fireEvent(event);
        Assert.assertEquals(0, event.targetSpeed, 0.00001);

        bus.registerHandler(new Base());

        event = new PipeEventItem.ModifySpeed(null, null, null, 1);
        bus.fireEvent(event);
        Assert.assertEquals(2, event.targetSpeed, 0.00001);

        bus = new PipeEventBus();
        bus.registerHandler(new Sub());

        event = new PipeEventItem.ModifySpeed(null, null, null, 1);
        bus.fireEvent(event);
        Assert.assertEquals(3, event.targetSpeed, 0.00001);
    }

    public static class Base {
        @PipeEventHandler
        public void modifySpeed2(PipeEventItem.ModifySpeed event) {
            event.targetSpeed = 2;
        }
    }

    public static class Sub extends Base {
        @Override
        public void modifySpeed2(PipeEventItem.ModifySpeed event) {
            event.targetSpeed = 3;
        }
    }
```

Unfortunately, code coverage for transport.pipe.api is not possible. I have tried to move transport.pipe.api into the buildcraft library to see if the transport.pipe.api will come up in the jacocoTestReport index.html, but it doesn't show up. I have also tried to create a separate package for transport.pipe.api tests but it doesn't work either.

For this project, I estimate my code coverage to be around +-30%, there are multiple reasons for this:

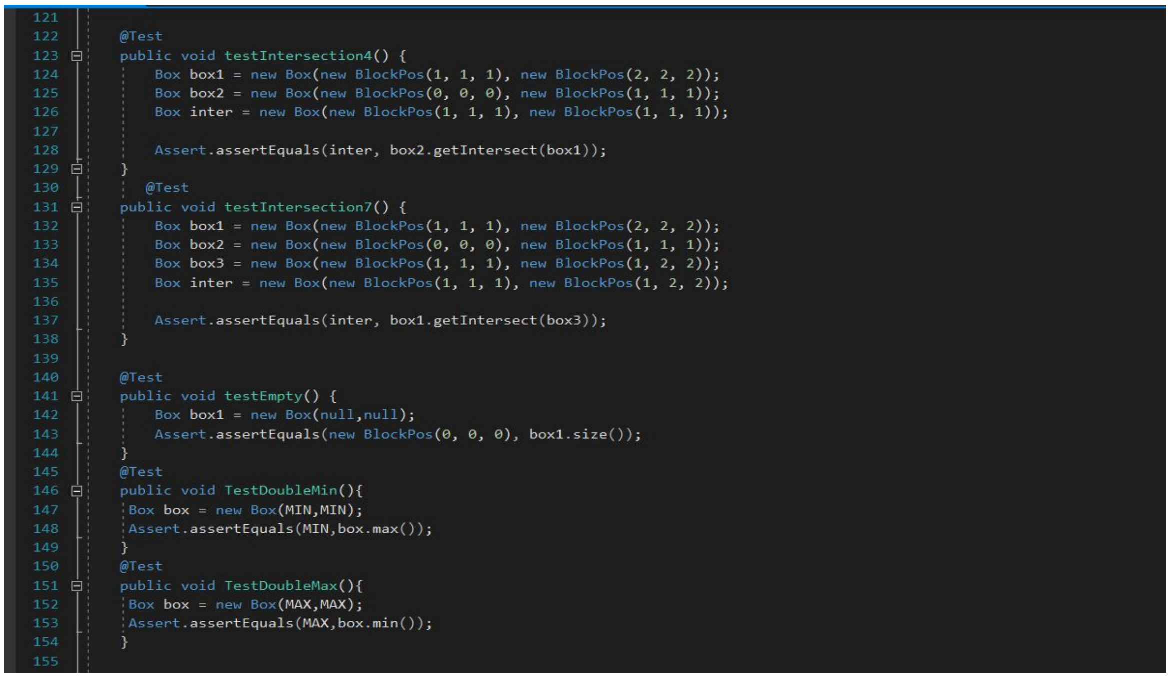
- Most classes in transport.pipe.api only contain setter and getter methods. Therefore, I don't feel like it's necessary to test them.
- It is hard to test transport.pipe.api using the library when you have never had experience working on a minecraft mod. Plus, there are only a few code examples on Buildcraft on the internet.
- I spent a significant amount of time understanding how the codes work.
- Most functional methods are concentrated in PipeEventItem.

Having code coverage is a good tool to measure one progress when testing a software, however it does not guarantee good testing. Based on all my reasonings and besides the low approximation of code coverage, I believe that I have spent a significant effort in this project.

Here is another example of our unit tests:

BoxTester: Unit testing this was a bit more simple than others due to figuring out what it meant pretty quickly. These tests were based on vector position and intersections which revolved a lot around integers making the tests a bit simple. Here I created a few tests tha i believe were needed.I started off by testing a bit of the positions. One main thing I was checking was the empty state. Once you got a couple tests going, it was easy to keep making more.

The code coverage was hard to determine due to not being able to run jacoco:report.

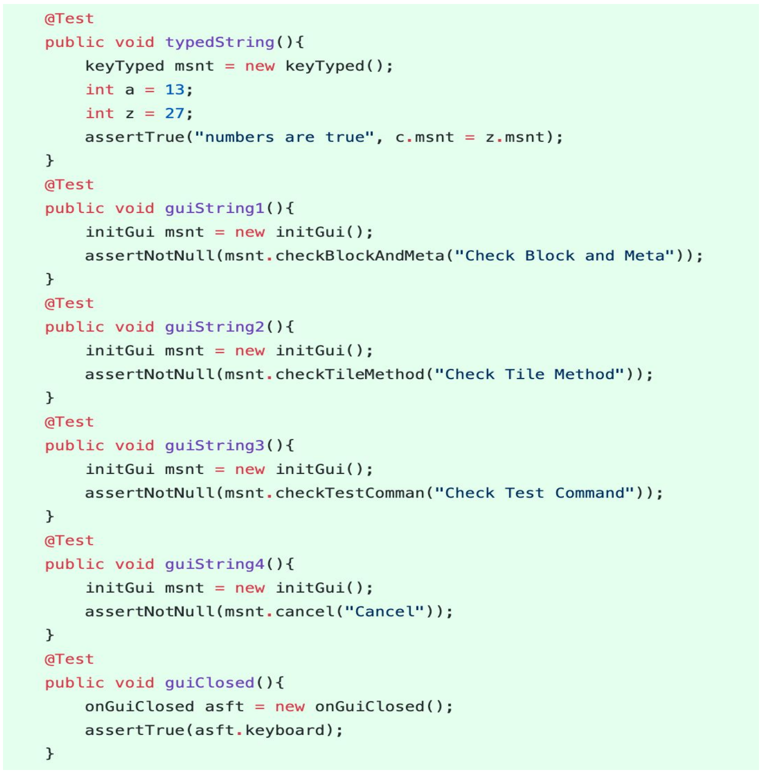


```java

    @Test
    public void testIntersection4() {
        Box box1 = new Box(new BlockPos(1, 1, 1), new BlockPos(2, 2, 2));
        Box box2 = new Box(new BlockPos(0, 0, 0), new BlockPos(1, 1, 1));
        Box inter = new Box(new BlockPos(1, 1, 1), new BlockPos(1, 1, 1));

        Assert.assertEquals(inter, box2.getIntersect(box1));
    }
      @Test
    public void testIntersection7() {
        Box box1 = new Box(new BlockPos(1, 1, 1), new BlockPos(2, 2, 2));
        Box box2 = new Box(new BlockPos(0, 0, 0), new BlockPos(1, 1, 1));
        Box box3 = new Box(new BlockPos(1, 1, 1), new BlockPos(1, 2, 2));
        Box inter = new Box(new BlockPos(1, 1, 1), new BlockPos(1, 2, 2));

        Assert.assertEquals(inter, box1.getIntersect(box3));
    }

    @Test
    public void testEmpty() {
        Box box1 = new Box(null,null);
        Assert.assertEquals(new BlockPos(0, 0, 0), box1.size());
    }
    @Test
    public void TestDoubleMin(){
     Box box = new Box(MIN,MIN);
     Assert.assertEquals(MIN,box.max());
    }
    @Test
    public void TestDoubleMax(){
     Box box = new Box(MAX,MAX);
     Assert.assertEquals(MAX,box.min());
    }

```

I estimate that this covered around 40%. I believe the reason we were getting issues obtaining the report was due to us using gradle.

GuiTester: Testing gui in minecraft was quite simple, just had to check whether the GUI worked for the user or not. To test the gui, I used assert.assertnotnull() to check if that object is null or not. If it is null then it throws an AssertionError. Had to check if a user can type strings, if strings are not null, if the user closed the gui, if buttons work on the gui, etc. These tests are basic tests for a gui, it focuses mostly all of the elements in a gui. Unable to retrieve code coverage since there is no jacoco:report file generated, however I estimated at least 30% code coverage with these tests. It seems gradle has been giving issues with this project.

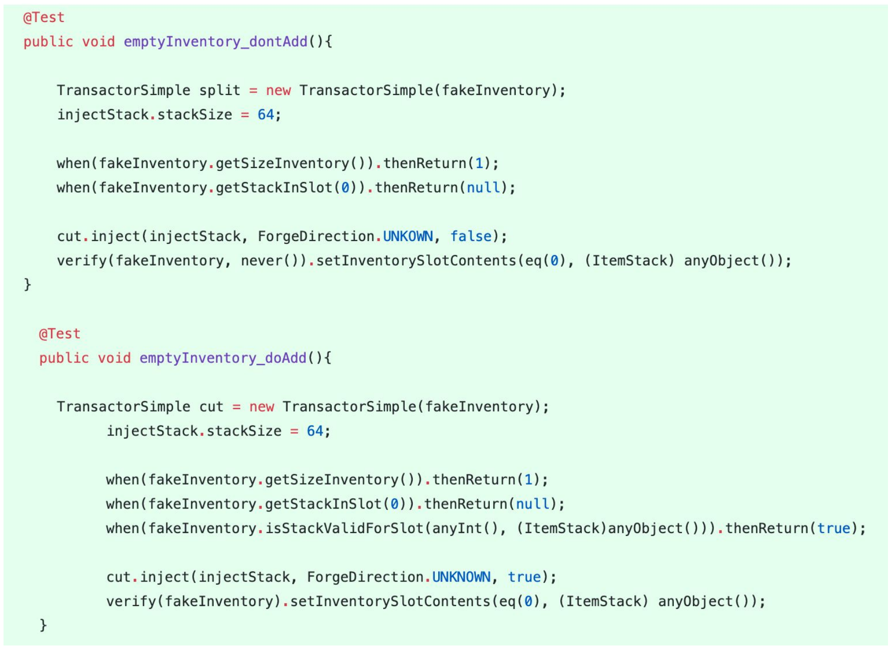
Commits for this java file:
Apr 6, 2020 “added tests, needs to be fixed later”
May 20, 2020 “added some tests”
May 14, 2020 “added gui tests”



```
    @Test
    public void typedString(){
        keyTyped msnt = new keyTyped();
        int a = 13;
        int z = 27;
        assertTrue("numbers are true", c.msnt = z.msnt);
    }
    @Test
    public void guiString1(){
        initGui msnt = new initGui();
        assertNotNull(msnt.checkBlockAndMeta("Check Block and Meta"));
    }
    @Test
    public void guiString2(){
        initGui msnt = new initGui();
        assertNotNull(msnt.checkTileMethod("Check Tile Method"));
    }
    @Test
    public void guiString3(){
        initGui msnt = new initGui();
        assertNotNull(msnt.checkTestComman("Check Test Command"));
    }
    @Test
    public void guiString4(){
        initGui msnt = new initGui();
        assertNotNull(msnt.cancel("Cancel"));
    }
    @Test
    public void guiClosed(){
        onGuiClosed asft = new onGuiClosed();
        assertTrue(asft.keyboard);
    }
```

ItemTransactorTester: Testing items in minecraft is complex, since there are a ton of items and had to test with a fake inventory to determine whether inventories with items were working. Determine if inventories were empty, full, half full, return success, errors, split items, stack items, and etc. Working with “FORGEDirection” was really confusing, forge direction is a built in function that supports BuildCraft and without it, BuildCraft cannot run in minecraft. Also unable to retrieve code coverage since there is no jacoco:report file generated, however I estimated at least 35% code coverage with these tests.

Commits for this java file:
May 4, 2020 “adding inventory tests” (missing?? So it was committed in May 14)
May 14, 2020 “fixing inventory tests”
May 14, 2020 “adding more inventory tests”

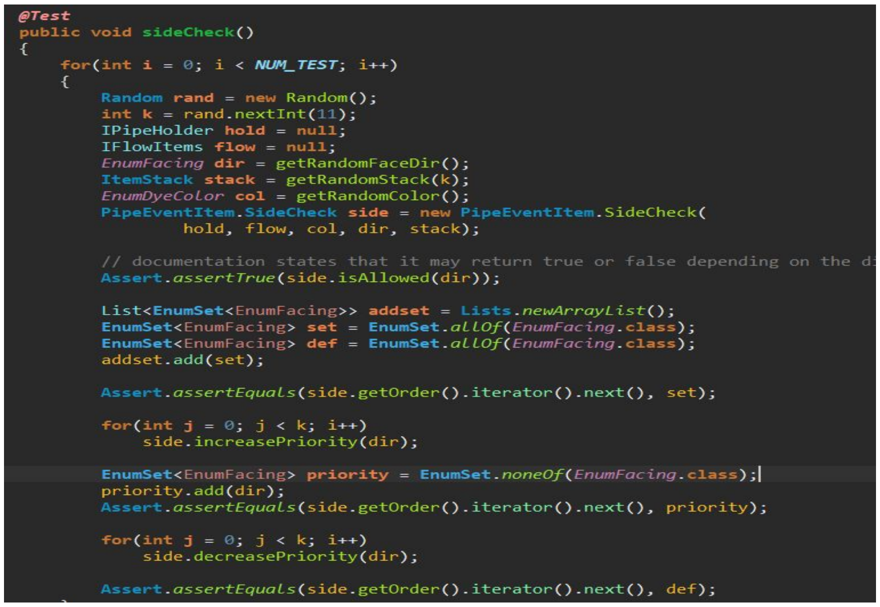


```
@Test
public void emptyInventory_dontAdd(){

    TransactorSimple split = new TransactorSimple(fakeInventory);
    injectStack.stackSize = 64;

    when(fakeInventory.getSizeInventory()).thenReturn(1);
    when(fakeInventory.getStackInSlot(0)).thenReturn(null);

    cut.inject(injectStack, ForgeDirection.UNKOWN, false);
    verify(fakeInventory, never()).setInventorySlotContents(eq(0), (ItemStack) anyObject());
}

  @Test
  public void emptyInventory_doAdd(){

    TransactorSimple cut = new TransactorSimple(fakeInventory);
          injectStack.stackSize = 64;

          when(fakeInventory.getSizeInventory()).thenReturn(1);
          when(fakeInventory.getStackInSlot(0)).thenReturn(null);
          when(fakeInventory.isStackValidForSlot(anyInt(), (ItemStack)anyObject())).thenReturn(true);

          cut.inject(injectStack, ForgeDirection.UNKNOWN, true);
          verify(fakeInventory).setInventorySlotContents(eq(0), (ItemStack) anyObject());
  }
```

Automated Testing

BuildCraft was a big project so automated testing was a must.

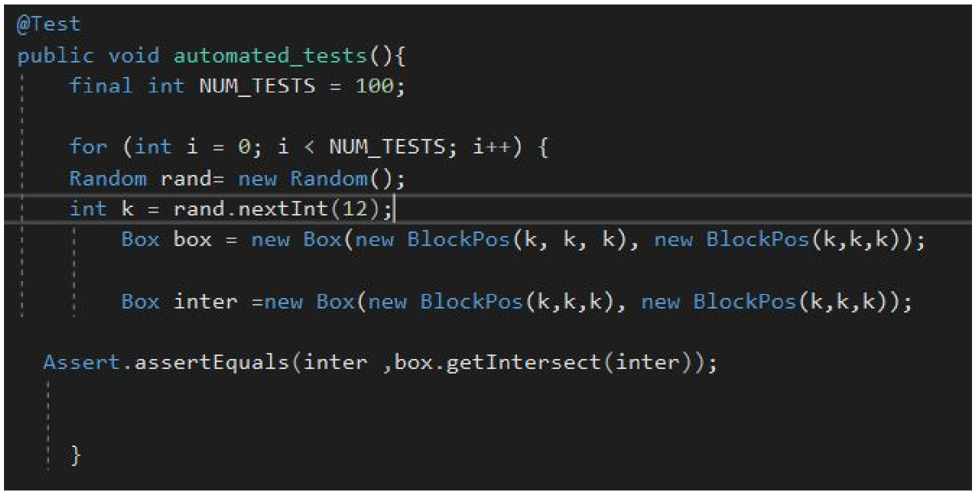
Transport pipe api: For transport pipe api, the automated test I use has a set of 40 random inputs. For tests that use a considerable amount of arguments, all arguments used are randomized to produce a broader input. For example, the method sideCheck() which checks the priorities of each EnumFacing. Variables such as dir (randomized state direction of the block), stack (stack of randomized item in minecraft), col/colour(randomized dye color), are randomized to ensure that it covers different possible inputs. All these randomized variables are achieved by utilizing java random util.



```java
@Test
public void sideCheck()
{
    for(int i = 0; i < NUM_TEST; i++)
    {
        Random rand = new Random();
        int k = rand.nextInt(11);
        IPipeHolder hold = null;
        IFlowItems flow = null;
        EnumFacing dir = getRandomFaceDir();
        ItemStack stack = getRandomStack(k);
        EnumDyeColor col = getRandomColor();
        PipeEventItem.SideCheck side = new PipeEventItem.SideCheck(
                hold, flow, col, dir, stack);

        // documentation states that it may return true or false depending on the di
        Assert.assertTrue(side.isAllowed(dir));

        List<EnumSet<EnumFacing>> addset = Lists.newArrayList();
        EnumSet<EnumFacing> set = EnumSet.allOf(EnumFacing.class);
        EnumSet<EnumFacing> def = EnumSet.allOf(EnumFacing.class);
        addset.add(set);

        Assert.assertEquals(side.getOrder().iterator().next(), set);

        for(int j = 0; j < k; i++)
            side.increasePriority(dir);

        EnumSet<EnumFacing> priority = EnumSet.noneOf(EnumFacing.class);
        priority.add(dir);
        Assert.assertEquals(side.getOrder().iterator().next(), priority);

        for(int j = 0; j < k; i++)
            side.decreasePriority(dir);

        Assert.assertEquals(side.getOrder().iterator().next(), def);
```

Another automated test we did was on the BoxTester:

For BoxTester I used a set of 100 tests. The way I did it was by setting up the vector points to be equal to a random number bounded to 12. This was quite simple to set up and I tested the functionality of it by increasing the number of tests.The numbers were set up with java.util.random.



```java
@Test
public void automated_tests(){
    final int NUM_TESTS = 100;

    for (int i = 0; i < NUM_TESTS; i++) {
    Random rand= new Random();
    int k = rand.nextInt(12);
        Box box = new Box(new BlockPos(k, k, k), new BlockPos(k,k,k));

        Box inter =new Box(new BlockPos(k,k,k), new BlockPos(k,k,k));

  Assert.assertEquals(inter ,box.getIntersect(inter));


   }
```

Lesson Learned

- Testing wand viewing coverage was hard due to not only the project being huge, but also due to not knowing the full functionality of gradle.
- Knowing how big and complex this SUT is now, I should have picked a different SUT back then. I spent a few weeks just looking at the codes and understanding how the components interact and work with each other and a few more other weeks to understand how the complex components work. Even now, there are still several components that I have yet to know how to implement.

  But it's also fun to learn new stuff, such as how the pipe works (i.e Most events in the PipeEventItem requires fireEvent to trigger the event to happen).

  In the future, I will understand this SUT better, since my soon-to-be part time job is teaching kids how to mod minecraft and roblox..
- Working with GUI is a lot easier than working with inventories in-game. GUIs are simple, testing strings, buttons, icons, and etc. On the other hand, testing inventory in minecraft is complicated, there are a lot more things to check. Stacking items in an inventory, moving items around the inventory, checking whether an item stacks exceeds the number 64 or is lower than 1, can items disappear in the inventory.

Flaws Found

- Buildcraft lacks documentation. Everytime I work on this project I had to teach myself how the classes work. One problem I encounter is defining PipeDefinition. Initializing a Pipe requires a PipeDefinition, and initializing PipeDefinition requires PipeDefinitionBuilder. When you initialize PipeDefinitionBuilder into PipeDefinition it works just fine. But when putting PipeDefinition as a parameter in the Pipe, there is a mismatch error, even though that is the correct way to initialize it, at least that's how I see it.

  So far, there are only 2 ways you can find information about the SUT
    - The code comment within the SUT. This is probably the simplest way to find what the code does.
    - Join BuildCraft discord server. I had to join their discord server to ask for guidance and advice. They also have a subreddit dedicated for BuildCraft, but it's mostly filled with threads asking about how to utilize the BuildCraft in-game.

  The chance to find information about BuildCraft on the internet other beside these two methods is very slim.
- Buildcraft does not support the use of maven. It instead uses gradle which in this case, lacked tools such as the jacoco report and pit report.