Monopoly

By: Zheyue Zhao, Emily Berger and Zejun Ma

Outline

The App 01

Some background on the SUT, short demo

Testing 02

Some information about our testing approach, demo

Results 03

Our test results so far

Conclusion 04

Wrapping it up

The App!

An Introduction

Monopoly Application

Source

~2.5k lines, 35-40 classes

Language

100% Java

Libraries

Java Swing (UI)

App Setup

Desktop application, played over the network

Users

People interested in playing Monopoly with friends online

Maintenance

Not maintained/updated recently (~7y)

Demo

We will showcase basic entry into the application here

Testing!

An outline of our approach towards testing this application

Our Testing Approach (and goals)

Whitebox

~70% branch coverage,
~80% statement

Mock

Important for testing client/server, and untangle dependencies

Blackbox

Verify basically functionalities of the game

Randomness

How do we know the game is not rigged?

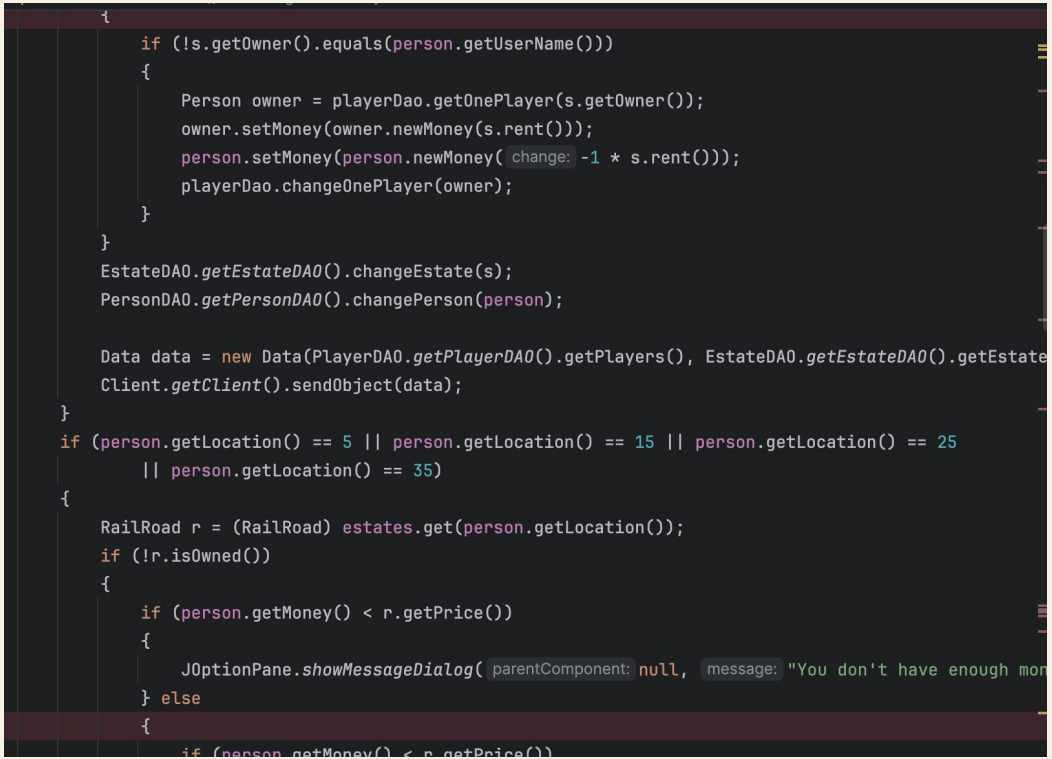
Why not 100%(or 90%) coverage?

- A lot of the front end UI code dealing with different components doesn't lend itself very well for unit testing.
- Some of them can't be meaningfully tested.

```
this.setTitle("new request");
this.setModal(true);
this.setLayout(new FlowLayout());
this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
this.setSize(new Dimension(dim.width , height: 300));
this.setMinimumSize(new Dimension(dim.width , height: 300));
this.setLocation( x: Toolkit.getDefaultToolkit().getScreenSize().width / 2 - dim.width * 13 / 32,
        y: Toolkit.getDefaultToolkit().getScreenSize().height / 2 - 75);
```

Why Mock Testing?

- Really the only thing that makes sense here.



Why random testing?

- Randomness is integral to the game of monopoly therefore its correct implementation is critical

Faults found

Yes Faults

When you have $0 and want to buy a house, if you select buy the game will put you into negative without giving you the house.

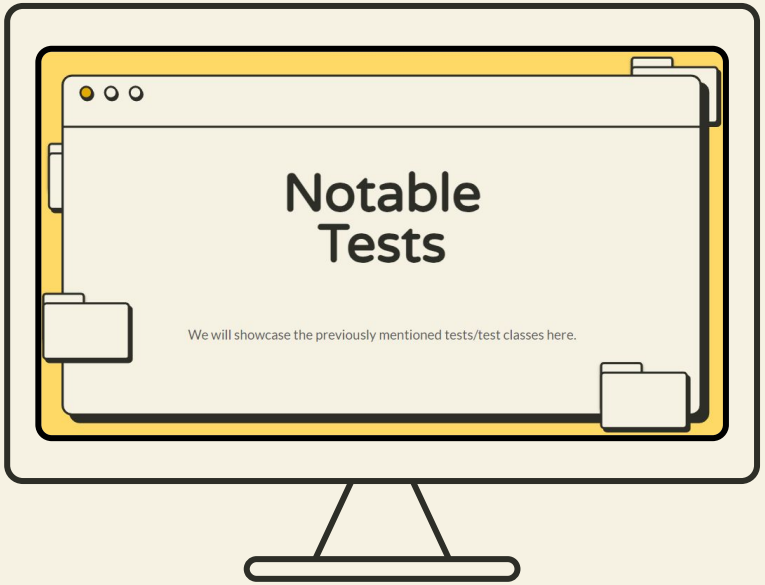
Potential Faults

Sound files included in the source folder is corrupted, which cause the entire program to crash. However; even if this isn't a bug with the code it still points to a design problem.

Maybe but not really

Money can overflow if you hit the integer max int.
Error thrown to console in Client (when typically the application catches them and sends them to JOptionPanes

Demo


Notable
Tests
We will showcase the previously mentioned tests/test classes here.

Results!

An outline of our current results

Current Coverage Report

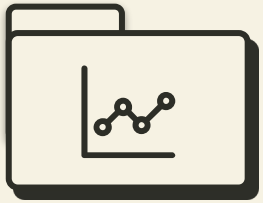
Coverage ^(?!.*IT$).*$

Element	Class, %	Method, %	Line, %	Branch, %
all	83% (47/56)	82% (275/334)	85% (2006/2357)	70% (952/1352)
resources	100% (1/1)	50% (1/2)	16% (1/6)	100% (0/0)
org.bihe	83% (46/55)	82% (274/332)	85% (2005/2351)	70% (952/1352)
Generated	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
util	100% (1/1)	100% (2/2)	50% (15/30)	40% (4/10)
ui	80% (29/36)	67% (104/154)	80% (930/1160)	77% (229/296)
playSound	0% (0/1)	0% (0/2)	0% (0/32)	0% (0/4)
network	100% (4/4)	85% (18/21)	59% (72/121)	44% (15/34)
model	100% (9/9)	98% (120/122)	99% (858/865)	69% (667/962)
main	0% (0/1)	0% (0/1)	0% (0/4)	100% (0/0)
DAO	100% (3/3)	100% (30/30)	93% (130/139)	80% (37/46)

Intended Expansion

Usability

Complete usability testing for the application



Coverage

Increase line and branch coverage - Client/Server, Streets, etc

Conclusion!

An outline of what we've achieved, and what we intend to finalize this week.

Challenges

- The source code isn’t written with testing in mind
- How useful some of the tests are is questionable
- Some tests are too complex that it is difficult for anyone other than the author of them to understand
- Lack of good ways to automate black box testing due to the random nature of the game
- Tests are fragile.

Thanks!

Do you have any questions?