Alex Ruef
CS 362
Assignment 1

(a)

```
import static org.junit.Assert.*;


import java.util.Random;


import org.junit.Test;

import java.util.ArrayList;

public class unitTests {


        //tests adding cards without buying them.

        @Test

        public void addCardTest() {

                Random rand = new Random();

                int i=0;

                new dominionBoard();

                Player player = dominionBoard.player1;

                for(Card card : dominionBoard.cardList.values())

                {

                        dominionBoard.addCard(player, card);

                        assert(player.getdiscard().get(i) == card);

                        i++;

                }

        }


        //make sure you can buy when you have enough money
```

```
	@Test
	public void successfulBuy() {
		Player player;
		new dominionBoard();
		player = dominionBoard.player1;
		for(Card card : dominionBoard.cardList.values())
		{
			player.setmoney(card.cost);
			dominionBoard.buyCard(player, card);
			if(card == dominionBoard.cardList.get(Card.Name.curse))
				assert(dominionBoard.player2.getdiscard().contains(card));
			else
				assert(player.getdiscard().contains(card));
		}
	}

	//tests buying cards you have no money for(also make sure copper and curse cards get bought
since they are free)
	@Test
	public void noMoneyBuy() {
		Player player;
		new dominionBoard();
		player = dominionBoard.player1;
		for(Card card : dominionBoard.cardList.values()){
			dominionBoard.buyCard(player, card);
			if(card == dominionBoard.cardList.get(Card.Name.curse))
```

```
                assert(dominionBoard.player2.getdiscard().contains(card));
            else if(card == dominionBoard.cardList.get(Card.Name.copper))
                assert(player.getdiscard().contains(card));
            else
                assert(!player.getdiscard().contains(card));
        }
}

//tests buying cards that have are out
@Test
public void noPoolBuy() {
        new dominionBoard();
        Player player = dominionBoard.player1;
        Card card = dominionBoard.cardList.get(Card.Name.gold);
        player.setmoney(card.cost);
        dominionBoard.pool.put(card, 0);
        dominionBoard.buyCard(player, card);
        assert(player.getdiscard().size()==0);
        assert(dominionBoard.pool.get(card)==0);
}

//tests drawing cards
@Test
public void drawTest() {
        new dominionBoard();
        Player player = dominionBoard.player1;

        Card[] cards = new Card[5];

        cards[0] = dominionBoard.cardList.get(Card.Name.copper);

        cards[1] = dominionBoard.cardList.get(Card.Name.smithy);

        cards[2] = dominionBoard.cardList.get(Card.Name.gardens);

        cards[3] = dominionBoard.cardList.get(Card.Name.gold);

        cards[4] = dominionBoard.cardList.get(Card.Name.estate);

        for(int i=0;i<cards.length;i++)

        {

                player.getdiscard().add(cards[i]);

        }

        player.draw(5);

        assert(player.getdeck().size()==0);

        assert(player.getdiscard().size()==0);

        for(int i=0;i<cards.length;i++)

        {

                assert(player.gethand().contains(cards[i]));

        }

}


//test if the game is over

@Test

public void isGameOverTest() {

        new dominionBoard();

        assert(dominionBoard.isGameOver()==false);
        dominionBoard.pool.put(dominionBoard.cardList.get(Card.Name.province), 0);

        assert(dominionBoard.isGameOver()==true);

        new dominionBoard();

        assert(dominionBoard.isGameOver()==false);

        dominionBoard.pool.put(dominionBoard.cardList.get(Card.Name.gold), 0);

        dominionBoard.pool.put(dominionBoard.cardList.get(Card.Name.smithy), 0);

        dominionBoard.pool.put(dominionBoard.cardList.get(Card.Name.gardens), 0);

        assert(dominionBoard.isGameOver()==true);

}


//tests the function that plays cards

@Test

public void playCardTest() {

        new dominionBoard();

        Player player = dominionBoard.player1;

        Card card = dominionBoard.cardList.get(Card.Name.gold);

        player.gethand().add(card);

        dominionBoard.playCard(card);

        assert(player.gethand().size()==0);

        assert(player.getdiscard().get(0)==card);

}


//tests the cleanup function

@Test

public void cleanupTest() {
        new dominionBoard();
        Player player = dominionBoard.player1;
        for(Card card : dominionBoard.cardList.values())
        {
                player.getdeck().add(card);
                if(player.getdeck().size()==5)
                        break;
        }
        player.cleanup();
        assert(player.getactions()==1);
        assert(player.getbuys()==1);
        assert(player.getmoney()==0);
        assert(player.gethand().size()==5);
}

//tests the function that returns action cards in a players hand
@Test
public void actionCardsTest() {
        new dominionBoard();
        Player player = dominionBoard.player1;
        Card[] cards = new Card[3];
        cards[0] = dominionBoard.cardList.get(Card.Name.smithy);
        cards[1] = dominionBoard.cardList.get(Card.Name.adventurer);
        cards[2] = dominionBoard.cardList.get(Card.Name.salvager);
        Card vpCard = dominionBoard.cardList.get(Card.Name.estate);
```

```
		for(int i=0;i<3;i++)
		{
			player.gethand().add(cards[i]);
		}
		for(int i=0;i<3;i++)
		{
			assert(player.actionCards().contains(cards[i]));
		}
		assert(!player.actionCards().contains(vpCard));
	}

	//test that the winner function
	@Test
	public void winnerTest() {
		new dominionBoard();
		Card estate = dominionBoard.cardList.get(Card.Name.estate);
		Card duchy = dominionBoard.cardList.get(Card.Name.duchy);
		Card province = dominionBoard.cardList.get(Card.Name.province);
		Card gardens = dominionBoard.cardList.get(Card.Name.gardens);
		Card smithy = dominionBoard.cardList.get(Card.Name.smithy);
		dominionBoard.player1.getdeck().add(estate);
		dominionBoard.player1.getdeck().add(province);
		dominionBoard.player1.getdeck().add(duchy);

		dominionBoard.turn=2;
```

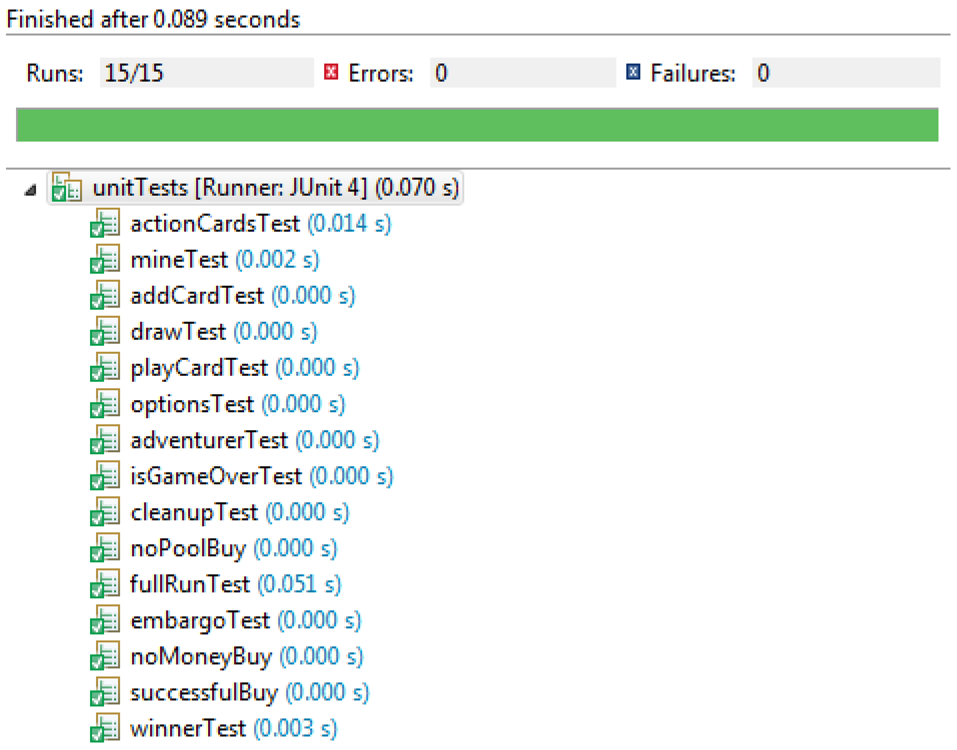
```
        for(int i=0;i<10;i++)
        {
                dominionBoard.player2.getdeck().add(smithy);
        }
        dominionBoard.player2.getdeck().add(gardens);


        assert(dominionBoard.winner()=="Player1 wins!");
        assert(dominionBoard.player1.getvp()==10);
        assert(dominionBoard.player2.getvp()==1);
}


//tests the embargo card
@Test
public void embargoTest() {
        new dominionBoard();
        Card embargo = dominionBoard.cardList.get(Card.Name.embargo);
        dominionBoard.player1.gethand().add(embargo);
        dominionBoard.playCard(embargo);
        Card card = embargo;
        for(Card i : dominionBoard.embargo.keySet())
        {
                if(dominionBoard.embargo.get(i)==1)
                        card = i;
        }
        dominionBoard.player2.setmoney(card.cost);
```

```
		dominionBoard.buyCard(dominionBoard.player2, card);

	assert(dominionBoard.player2.getdiscard().contains(dominionBoard.cardList.get(Card.Name.cur
se)));
	}


	//tests the options function
	@Test
	public void optionsTest() {
		new dominionBoard();
		Player player = dominionBoard.player1;
		Card[] cards = new Card[7];
		cards[0] = dominionBoard.cardList.get(Card.Name.ambassador);
		cards[1] = dominionBoard.cardList.get(Card.Name.embargo);
		cards[2] = dominionBoard.cardList.get(Card.Name.great_hall);
		cards[3] = dominionBoard.cardList.get(Card.Name.copper);
		cards[4] = dominionBoard.cardList.get(Card.Name.silver);
		cards[5] = dominionBoard.cardList.get(Card.Name.curse);
		cards[6] = dominionBoard.cardList.get(Card.Name.estate);
		ArrayList<Card> options = dominionBoard.cardOptions(3);
		for(int i=0;i<cards.length;i++)
		{
			assert(options.contains(cards[i]));
		}
	}
```

```
//test the adventurer card

@Test

public void adventurerTest() {

            new dominionBoard();

            Player player = dominionBoard.player1;

            Card adventurer = dominionBoard.cardList.get(Card.Name.adventurer);

            Card smithy = dominionBoard.cardList.get(Card.Name.smithy);

            Card gold = dominionBoard.cardList.get(Card.Name.gold);

            player.getdeck().add(smithy);

            player.getdeck().add(smithy);

            player.getdeck().add(smithy);

            player.getdeck().add(gold);

            player.getdeck().add(gold);

            adventurer.name.action();

            assert(player.gethand().get(0)==gold);

            assert(player.gethand().get(1)==gold);

            assert(player.getdiscard().size()==3);

}


//test the mine card

@Test

public void mineTest() {

            new dominionBoard();

            Player player = dominionBoard.player1;

            Card mine = dominionBoard.cardList.get(Card.Name.mine);
```

```
                Card copper = dominionBoard.cardList.get(Card.Name.copper);

                int cost = copper.cost+3;

                player.gethand().add(copper);

                for(int i=0;i<20;i++)

                {

                        cost = player.gethand().get(0).cost+3;

                        mine.name.action();

                        assert(player.gethand().size()==1);

                        assert(player.gethand().get(0).type==Card.Type.treasure);

                        assert(player.gethand().get(0).cost<=cost);

                }

        }


        //Run the game a few times and tests that they finished correctly

        @Test

        public void fullRunTest() {

                for(int i=0;i<20;i++)

                {

                        new dominionBoard();

                        dominionBoard.start();

                        assert(dominionBoard.isGameOver()==true);

                }

        }

}
```

(b)



```
Finished after 0.089 seconds
Runs: 15/15    Errors: 0    Failures: 0

unitTests [Runner: JUnit 4] (0.070 s)
    actionCardsTest (0.014 s)
    mineTest (0.002 s)
    addCardTest (0.000 s)
    drawTest (0.000 s)
    playCardTest (0.000 s)
    optionsTest (0.000 s)
    adventurerTest (0.000 s)
    isGameOverTest (0.000 s)
    cleanupTest (0.000 s)
    noPoolBuy (0.000 s)
    fullRunTest (0.051 s)
    embargoTest (0.000 s)
    noMoneyBuy (0.000 s)
    successfulBuy (0.000 s)
    winnerTest (0.003 s)
```

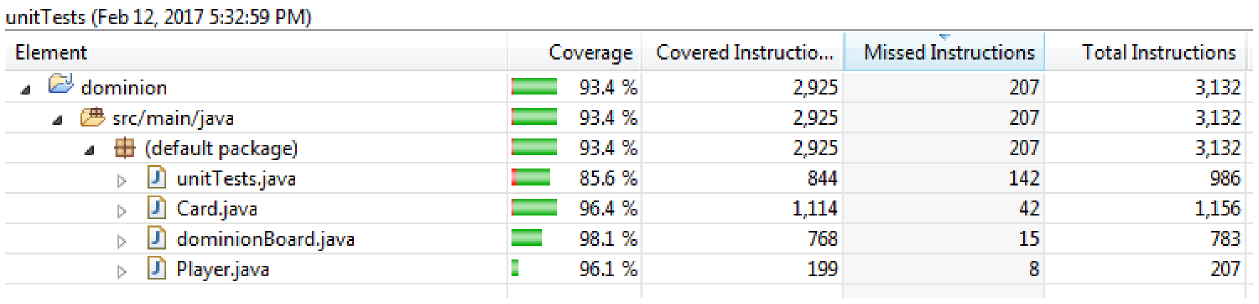
(c)

I wrote my program very modularly so I picked out the important or more complicated methods when testing. With each test I setup the player (sometimes both players) so that I can predict the outcome before running the action I was testing. Most of the asserts ended up being tests on the contents of the players decks.

(d)

I found a bug with my gardens card. It is supposed to count up all the cards the player has but I forgot to count the cards in the player's deck. I found a bug in my buy function, there was a greater/less than symbol in the wrong direction the only came up when the card's supply pile was empty. I also caught an error where buying curse cards was incorrectly putting it into the deck of the player that bought it instead of the other player's deck.

(e)



unitTests (Feb 12, 2017 5:32:59 PM)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
dominion	93.4 %	2,925	207	3,132
src/main/java	93.4 %	2,925	207	3,132
(default package)	93.4 %	2,925	207	3,132
unitTests.java	85.6 %	844	142	986
Card.java	96.4 %	1,114	42	1,156
dominionBoard.java	98.1 %	768	15	783
Player.java	96.1 %	199	8	207

More code coverage details can be found in the html output in my github folder.

I'm confident I have good tests that cover 75% of my code. The last ~18% was difficult to test well because I couldn't test parts of it without running a full dominion game. The amount of randomness that occurs during a game makes it hard to predict the final outcome. I ended up just running the game() function many times to make sure the code doesn't crash and that it ends when it's supposed to (isGameOver should return true).

(f)

The game is ran entirely by bots so It can already by ran with no input like I did in my fullRunTest. But this doesn't give much to test on since all you can accurately predict will be true at the end of the game is that the game should be over.

```
Public void fullRunTest () {
    For i=0:num
        New dominionBoard
        dominionBoard.start()
}
```

It would be easier to write automated tests for individual functions like buyCard() for example. There are 3 possible outcomes when buying a card because the player can either have enough money or not have enough money or there could be no cards of that name left.

```
Public void buyCardTest() {
    Card card = randomCard
    {
        Player.setmoney(card.cost)
```

```
                buyCard(card)
                assert(player.deck.contains(card))
        }
        OR {
                player.setmoney(0)
                buyCard(card)
                assert(player.deck.contains(card) != true)
        }
         OR {
                supplyPile.set(card) = 0
                buyCard(card)
                assert(player.deck.contains(card) != true)
        }
}
```

A method like the one above could be written for each different methods.