David Baugh
CS362 – W17
Assignment-1 report

JUnit Tests

CardTest.java

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Before;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;

public class CardTest {
    private GameState state;
    private Player player1;
    private Player player2;
    private List<Card> cards;

    @Before
    public void initializeGame() {
        cards = new ArrayList<Card>(Card.createCards());
        state = new GameState(cards);
        player1 = new Player(state, "PLAYER 1");
        state.addPlayer(player1);
        player2 = new Player(state, "PLAYER 2");
        state.addPlayer(player2);
        state.initializeGame();
    }

//---------------------
// TREASURE CARD TESTS
//---------------------
    @Test
    public void testValuesCopper() {
        Card Copper = Card.getCard(cards, Card.CardName.Copper);
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Copper.getCost(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Copper).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Copper).getType(),
Card.Type.TREASURE);
    }

    @Test
    public void testPlayCopper() {
        assertEquals(player1.coins, 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Copper));
        player1.playTreasureCard();
        assertEquals(player1.coins, 1);
    }
    @Test
    public void testValuesSilver() {
        assertEquals(Card.getCard(cards, Card.CardName.Silver).getCost(), 3);
        assertEquals(Card.getCard(cards, Card.CardName.Silver).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Silver).getType(),
Card.Type.TREASURE);
    }

    @Test
    public void testPlaySilver() {
        assertEquals(player1.coins, 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Silver));
        player1.playTreasureCard();
        assertEquals(player1.coins, 2);
    }

    @Test
    public void testValuesGold() {
        assertEquals(Card.getCard(cards, Card.CardName.Gold).getCost(), 6);
        assertEquals(Card.getCard(cards, Card.CardName.Gold).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Gold).getType(),
Card.Type.TREASURE);
    }

    @Test
    public void testPlayGold() {
        assertEquals(player1.coins, 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Gold));
        player1.playTreasureCard();
        assertEquals(player1.coins, 3);
    }

//-------------------
// ACTION CARD TESTS
//-------------------

    @Test
    public void testValuesAdventurer() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Adventurer).getCost(), 6);
        assertEquals(Card.getCard(cards, Card.CardName.Adventurer).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Adventurer).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayAdventurer() {
        player1.initializePlayerTurn();
        assertEquals(player1.coins, 0);
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.playedCards.size(), 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Adventurer));
        System.out.println(player1);
        player1.playKingdomCard();
        assertTrue(player1.hand.size() >= 6);
        assertTrue(player1.deck.size() >= 1);//depends if coppers are in front or not
        assertEquals(player1.playedCards.size(), 1);
        System.out.println(player1);
    }

    @Test
```

```java
    public void testValuesSmithy() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Smithy).getCost(), 4);
        assertEquals(Card.getCard(cards, Card.CardName.Smithy).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Smithy).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlaySmithy() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Smithy));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.hand.size(), 8);
        assertEquals(player1.deck.size(), 2);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 1);
        System.out.println(player1);
    }

    @Test
    public void testValuesVillage() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Village).getCost(), 3);
        assertEquals(Card.getCard(cards, Card.CardName.Village).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Village).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayVillage() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Village));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.hand.size(), 6);
        assertEquals(player1.deck.size(), 4);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 1);
        assertEquals(player1.numActions, 2);
        System.out.println(player1);
    }

    @Test
    public void testValuesAmbassador() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Ambassador).getCost(), 3);
        assertEquals(Card.getCard(cards, Card.CardName.Ambassador).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Ambassador).getType(),
Card.Type.ACTION);
    }
    @Test
    public void testPlayAmbassador() {
        player1.initializePlayerTurn();
        player2.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player2.discard.size(), 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Ambassador));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.hand.size(), 4);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 1);
        assertEquals(player2.discard.size(), 1);
        System.out.println(player1);
    }

    @Test
    public void testValuesBaron() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Baron).getCost(), 4);
        assertEquals(Card.getCard(cards, Card.CardName.Baron).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Baron).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayBaron() {
        player1.initializePlayerTurn();
        assertEquals(player1.numBuys, 1);
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player1.coins, 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Baron));
        System.out.println(player1);
        if(Card.getCard(player1.hand, Card.CardName.Estate) != null){
            player1.playKingdomCard();
            assertEquals(player1.hand.size(), 4);
            assertEquals(player1.deck.size(), 5);
            assertEquals(player1.discard.size(), 1);
            assertEquals(player1.playedCards.size(), 1);
            assertEquals(player1.numBuys, 2);
            assertEquals(player1.coins, 4);
        } else {
            player1.playKingdomCard();
            assertEquals(player1.hand.size(), 5);
            assertEquals(player1.deck.size(), 4);
            assertEquals(player1.discard.size(), 0);
            assertEquals(player1.playedCards.size(), 1);
            assertEquals(player1.numActions, 2);
            assertEquals(player1.coins, 0);
        }


        System.out.println(player1);
    }
    @Test
    public void testValuesCouncilRoom() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Council_Room).getCost(), 5);
        assertEquals(Card.getCard(cards, Card.CardName.Council_Room).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Council_Room).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayCouncilRoom() {
        player1.initializePlayerTurn();
        player2.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player1.numBuys, 1);
        player1.hand.add(Card.getCard(cards, Card.CardName.Council_Room));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.hand.size(), 9);
        assertEquals(player1.deck.size(), 1);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 1);
        assertEquals(player1.numBuys, 2);
        for(Player p : state.players){
            if(p != player1) assertEquals(p.hand.size(), 6);
        }
        System.out.println(player1);
    }

    @Test
    public void testValuesCutpurse() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Cutpurse).getCost(), 4);
        assertEquals(Card.getCard(cards, Card.CardName.Cutpurse).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Cutpurse).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayCutpurse() {
        player1.initializePlayerTurn();
        player2.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player1.coins, 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Cutpurse));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 1);
        assertEquals(player1.coins, 2);
        for(Player p : state.players){
            if(p != player1) assertEquals(p.hand.size(), 4);
        }
        System.out.println(player1);
    }

    @Test
    public void testValuesEmbargo() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Embargo).getCost(), 2);
        assertEquals(Card.getCard(cards, Card.CardName.Embargo).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Embargo).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayEmbargo() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player1.coins, 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Embargo));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player1.coins, 2);
//          assertTrue(state.embargoTokens.size() > 0);
        System.out.println(player1);
    }

    @Test
    public void testValuesFeast() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Feast).getCost(), 4);
        assertEquals(Card.getCard(cards, Card.CardName.Feast).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Feast).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayFeast() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player1.coins, 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Feast));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertTrue(player1.coins > 5);
        System.out.println(player1);
    }

    @Test
    public void testValuesGreatHall() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Great_Hall).getCost(), 3);
        assertEquals(Card.getCard(cards, Card.CardName.Great_Hall).score(), 1);
        assertEquals(Card.getCard(cards, Card.CardName.Great_Hall).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayGreatHall() {
        player1.initializePlayerTurn();
        assertEquals(player1.numActions, 1);
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Great_Hall));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.hand.size(), 6);
        assertEquals(player1.deck.size(), 4);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 1);
        assertEquals(player1.numActions, 1);
        System.out.println(player1);
    }

    @Test
    public void testValuesMine() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
        assertEquals(Card.getCard(cards, Card.CardName.Mine).getCost(), 5);
        assertEquals(Card.getCard(cards, Card.CardName.Mine).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Mine).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayMine() {
        player1.initializePlayerTurn();
        List<Card> before = Card.filter(player1.hand, Card.Type.TREASURE);
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Mine));
        System.out.println(player1);
        player1.playKingdomCard();
        List<Card> after = Card.filter(player1.hand, Card.Type.TREASURE);
        assertEquals(player1.hand.size(), 4);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 1);
        assertEquals(player1.playedCards.size(), 1);
        assertTrue(before.size() != after.size());
        assertTrue(Card.getCard(player1.discard, Card.CardName.Silver) != null);
        assertTrue(Card.getCard(player1.discard, Card.CardName.Gold) == null);
        System.out.println(player1);
    }

    @Test
    public void testValuesRemodel() {
        //assertEquals(Card.getCard(cards, Card.CardName.Copper).getCardName(),
"Copper"); //String<Copper> vs CardName<Copper>?
```

```java
        assertEquals(Card.getCard(cards, Card.CardName.Remodel).getCost(), 4);
        assertEquals(Card.getCard(cards, Card.CardName.Remodel).score(), 0);
        assertEquals(Card.getCard(cards, Card.CardName.Remodel).getType(),
Card.Type.ACTION);
    }

    @Test
    public void testPlayRemodel() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        player1.hand.add(Card.getCard(cards, Card.CardName.Remodel));
        System.out.println(player1);
        player1.playKingdomCard();
        assertEquals(player1.deck.size(), 5);
        assertTrue(player1.coins > 2);
        System.out.println(player1);
    }

}
```

GameStateTest.java

```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Before;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;

public class GameStateTest {
    private GameState state;
    private Player player1;
    private Player player2;
    private List<Card> cards;

    @Before
    public void initializeGame() {
        cards = new ArrayList<Card>(Card.createCards());
        state = new GameState(cards);
    }

    @Test
    public void addPlayer(){
        player1 = new Player(state, "PLAYER 1");
        state.addPlayer(player1);
        player2 = new Player(state, "PLAYER 2");
        state.addPlayer(player2);
        int numPlayers = 0;
        for(Player p : state.players) {
            numPlayers++;
        }
        assertEquals(numPlayers, 2);
    }

    @Test
    public void testInitializeGame(){
        player1 = new Player(state, "PLAYER 1");
        state.addPlayer(player1);
        player2 = new Player(state, "PLAYER 2");
        state.addPlayer(player2);
        assertEquals(player1.hand.size(), 0);
        assertEquals(player1.deck.size(), 0);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player2.hand.size(), 0);
        assertEquals(player2.deck.size(), 0);
        assertEquals(player2.discard.size(), 0);
        assertEquals(player2.playedCards.size(), 0);
        state.initializeGame();
        System.out.println(player1);
        assertEquals(player2.hand.size(), 0);
        assertEquals(player2.deck.size(), 0);
        assertEquals(player2.discard.size(), 10);
        assertEquals(player2.playedCards.size(), 0);
        assertEquals(player1.hand.size(), 0);
        assertEquals(player1.deck.size(), 0);
        assertEquals(player1.discard.size(), 10);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
    }

    @Test
    public void testIsGameOver(){
        player1 = new Player(state, "PLAYER 1");
        state.addPlayer(player1);
        player2 = new Player(state, "PLAYER 2");
        state.addPlayer(player2);
        state.initializeGame();
        for(int i = 0; i < 8; i++) {
            player1.deck.add(Card.getCard(state.cards, Card.CardName.Province));
            state.gameBoard.put(Card.getCard(state.cards, Card.CardName.Province),
state.gameBoard.get(Card.getCard(state.cards, Card.CardName.Province)) - 1 );
        }
        assertTrue(state.isGameOver());
/* if cards isn't in Supply then boom
        GameState State;
        State = new GameState(cards);
        Player player3 = new Player(State, "PLAYER 3");
        State.addPlayer(player3);
        Player player4 = new Player(State, "PLAYER 4");
        State.addPlayer(player4);
        State.initializeGame();
        for(int i = 0; i < 10; i++) {
            player1.deck.add(Card.getCard(State.cards, Card.CardName.Smithy));
            State.gameBoard.put(Card.getCard(State.cards, Card.CardName.Smithy),
State.gameBoard.get(Card.getCard(State.cards, Card.CardName.Smithy)) - 1 );
            player1.deck.add(Card.getCard(State.cards, Card.CardName.Mine));
            State.gameBoard.put(Card.getCard(State.cards, Card.CardName.Mine),
State.gameBoard.get(Card.getCard(State.cards, Card.CardName.Mine)) - 1 );
            player1.deck.add(Card.getCard(State.cards, Card.CardName.Feast));
            State.gameBoard.put(Card.getCard(State.cards, Card.CardName.Feast),
State.gameBoard.get(Card.getCard(State.cards, Card.CardName.Feast)) - 1 );
        }
        assertTrue(state.isGameOver());
*/
    }

    @Test
    public void testGetWinners(){
        player1 = new Player(state, "PLAYER 1");
        state.addPlayer(player1);
        player2 = new Player(state, "PLAYER 2");
        state.addPlayer(player2);
        state.initializeGame();
        for(int i = 0; i < 8; i++) {
            player1.deck.add(Card.getCard(state.cards, Card.CardName.Province));
            state.gameBoard.put(Card.getCard(state.cards, Card.CardName.Province),
state.gameBoard.get(Card.getCard(state.cards, Card.CardName.Province)) - 1 );
        }
        System.out.println(state.getWinners());
    }

    @Test
    public void testAddEmbargo(){

    }
```

PlayerTest.java

```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Before;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;

public class PlayerTest {
    private GameState state;
    private Player player1;
    private Player player2;
    private List<Card> cards;

    @Before
    public void initializeGame() {
        cards = new ArrayList<Card>(Card.createCards());
        state = new GameState(cards);
        player1 = new Player(state, "PLAYER 1");
        state.addPlayer(player1);
        player2 = new Player(state, "PLAYER 2");
        state.addPlayer(player2);
        state.initializeGame();
    }

    @Test
    public void testDrawCard() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
        player1.drawCard();
        assertEquals(player1.hand.size(), 6);
        assertEquals(player1.deck.size(), 4);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
    }
    @Test
    public void testInitializePlayerTurn() {
        assertEquals(player1.hand.size(), 0);
        assertEquals(player1.deck.size(), 0);
        assertEquals(player1.discard.size(), 10);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player1.numActions, 0);
        assertEquals(player1.coins, 0);
        assertEquals(player1.numBuys, 0);
        System.out.println(player1);
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        assertEquals(player1.numActions, 1);
        assertEquals(player1.coins, 0);
        assertEquals(player1.numBuys, 1);
        System.out.println(player1);
    }

    @Test
    public void testGain() {
        assertEquals(player1.hand.size(), 0);
        assertEquals(player1.deck.size(), 0);
        assertEquals(player1.discard.size(), 10);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
        player1.gain(Card.getCard(cards, Card.CardName.Province));
        assertEquals(player1.hand.size(), 0);
        assertEquals(player1.deck.size(), 0);
        assertEquals(player1.discard.size(), 11);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
    }

    @Test
    public void testDiscard() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
        player1.discard(Card.getCard(player1.hand, Card.CardName.Copper));
        assertEquals(player1.hand.size(), 4);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 1);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
    }

    @Test
    public void testPlayKingdomCard() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
        player1.hand.add(Card.getCard(cards, Card.CardName.Smithy));
        player1.playKingdomCard();
        assertEquals(player1.hand.size(), 8);
        assertEquals(player1.deck.size(), 2);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 1);
        System.out.println(player1);
    }

    @Test
    public void testScoreFor() {
        int score = 0;
        for(Card c : player1.discard){
            score += c.score;
        }
        assertEquals(score, player1.scoreFor());
    }

    @Test
    public void testPlayTreasureCard() {
        int money = 0;
        player1.initializePlayerTurn();
        for(Card c : player1.hand){
            money += c.getTreasureValue();
        }
        player1.playTreasureCard();
        assertEquals(money, player1.coins);
    }

    @Test
    public void testBuyCard() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
        player1.playTreasureCard();
        player1.buyCard(state);
        assertTrue(player1.hand.size() < 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 1);
        System.out.println(player1);
    }

    @Test
    public void testEndTurn() {
        player1.initializePlayerTurn();
        assertEquals(player1.hand.size(), 5);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 0);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
        player1.endTurn();
        assertEquals(player1.hand.size(), 0);
        assertEquals(player1.deck.size(), 5);
        assertEquals(player1.discard.size(), 5);
        assertEquals(player1.playedCards.size(), 0);
        System.out.println(player1);
    }
}
```

Test Output

CardTest.java – PlayerTest.java – GameStateTest.java

The actual output for all three of the three test files is dozens of player's drawing of cards and printing out of their hands. I will put a single test below so you can see the repetition

testPlaySmithy()

```
PLAYER 1's Initial Card Draw:
PLAYER 1 gains      Copper
PLAYER 1 gains      Copper
PLAYER 1 gains      Copper
PLAYER 1 gains      Copper
PLAYER 1 gains      Copper
PLAYER 1 gains      Copper
PLAYER 1 gains      Copper
PLAYER 1 gains      Estate
PLAYER 1 gains      Estate
PLAYER 1 gains      Estate

PLAYER 2's Initial Card Draw:
PLAYER 2 gains      Copper
PLAYER 2 gains      Copper
PLAYER 2 gains      Copper
PLAYER 2 gains      Copper
PLAYER 2 gains      Copper
PLAYER 2 gains      Copper
PLAYER 2 gains      Copper
PLAYER 2 gains      Estate
PLAYER 2 gains      Estate
PLAYER 2 gains      Estate

Reshuffle the deck of the player PLAYER 1 to draw FIVE cards
PLAYER 1 draws      Copper
PLAYER 1 draws      Estate
PLAYER 1 draws      Copper
PLAYER 1 draws      Copper
PLAYER 1 draws      Copper

------- PLAYER 1 -------
numActions: 1, coins: 0, numBuys: 1
```

```
Hand: [	Copper,	Copper,	Copper,	Copper,	Estate, Smithy]
Discard: []
Deck: [Copper,	Copper,	Estate, Copper,	Estate]
Played Cards: []

Player.actionPhase Card:	Smithy
+3 Cards.
PLAYER 1 draws	Copper
PLAYER 1 draws	Copper
PLAYER 1 draws	Estate

------- PLAYER 1 -------
numActions: 0, coins: 0, numBuys: 1
Hand: [	Copper,	Copper,	Copper,	Copper,	Copper,
	Copper,	Estate, Estate]
Discard: []
Deck: [Copper,	Estate]
Played Cards: [	Smithy]
```

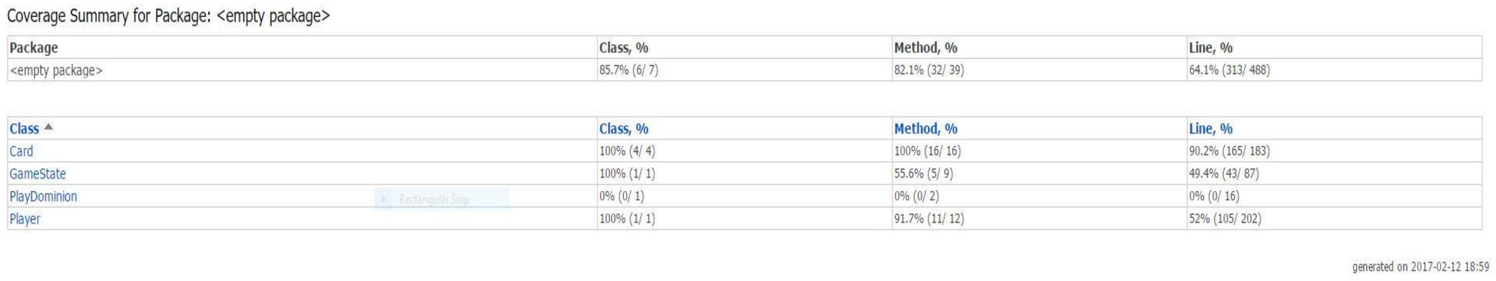
Reasons for Test Input

For each test, I decided on what the key elements of the tested function was. For instance, the smithy focuses on drawing cards so that test revolved around giving a player the standard Hand and then having that player use a Smithy. Following the use of the Smithy, I would measure the outcome and effects to the Hand, Deck, playedHand, and Discard. Each card has its own actions so each card test would generically bring in a standard hand (initializePlayerTurn()) then test the hand against the card being tested. Then for the methods in Player and GameState, I began mostly the same way by deciding what the key elements of the methods were. Then using a player's standard 5 card initialize hand, I would be able to test any of the methods by testing the effects of the card before and after the method happens. If you look at testPlayTreasureCard(), you will see that the test gathers the standard Hand then counts up the coins worth in the hand. After that, the test plays playTreasureCard() then again tests the player's coin amount.

Bugs in the Code

Bugs are easy to miss after working on the same project for about three weeks. You get so used to working with the code that simple little bugs like the fact that some of my cards that are supposed to buy card with some certain amount of coins do not in fact do that. They just give the player that many coins and add them to the overall coin amount. This is a bug that was so easy to overlook that even my tests think it is correct. Actually, most of my tests think I am doing the right thing so that is something that I will have to work on when I am not panicking to finish the project within the time allotted. Some other bugs that I have found include the fact that, while my embargo card chooses a card to put the token on, it does not follow through with it. My ambassador card only checks if there is more than one of the chosen card if that card is a curse. Otherwise it simply takes the one copy. The next bug is not really a bug but just code that is probably my worst every written code. My player.buyCard() function is absolutely horrendous. It is copy over copy over copy of the same code which only buys the most expensive card every single time so it does kind of break the rules of Dominion in that does not strategically buy cards to build its card engine correctly.

Test



Coverage Summary for Package: <empty package>

Package	Class, %	Method, %	Line, %
<empty package>	85.7% (6/ 7)	82.1% (32/ 39)	64.1% (313/ 488)

Class ▲	Class, %	Method, %	Line, %
Card	100% (4/ 4)	100% (16/ 16)	90.2% (165/ 183)
GameState	100% (1/ 1)	55.6% (5/ 9)	49.4% (43/ 87)
PlayDominion	0% (0/ 1)	0% (0/ 2)	0% (0/ 16)
Player	100% (1/ 1)	91.7% (11/ 12)	52% (105/ 202)

generated on 2017-02-12 18:59

As per the spreadsheet shows, overall there is a 82% method and 64% line coverage. The majority of the missing coverage consists of the PlayDominion.java and GameState.java files. PlayDominion.java does not have any tests because it is the driver and does not contain any new code. PlayDominion.java consists of methods from the other three files. Within GameState.java, there are a few methods that I was unable to create tests for as they require so many different random variables to match up to be able for them to be run.

Pseudo-Code

```
Get random class
Get parameters or dependencies
Cycle through all the methods and constructors
Randomly choose a constructor (if more than one)
Generate instance of the class with constructor
Randomly throw random parameters (from the parameters list) at the instance
Randomly throw inputs selected from those parameters into to an instance’s method
Run the method
If method doesn’t return true then return to randomly choosing a constructor
```