

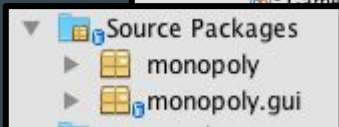
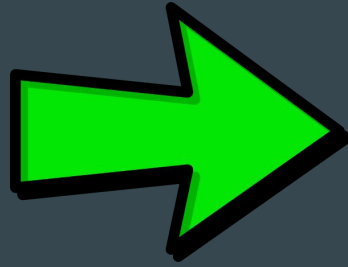
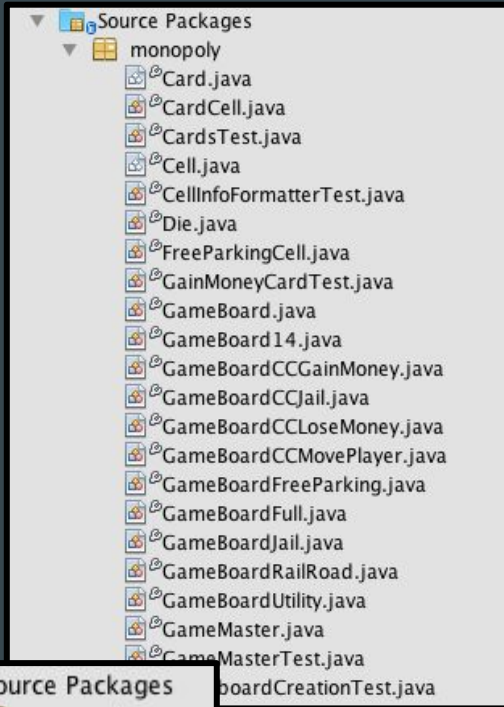
Bug Fixes

- Minimum Players
- Default Player Names (not null)
- Negative Money for Trades



Refactorings / Improvements

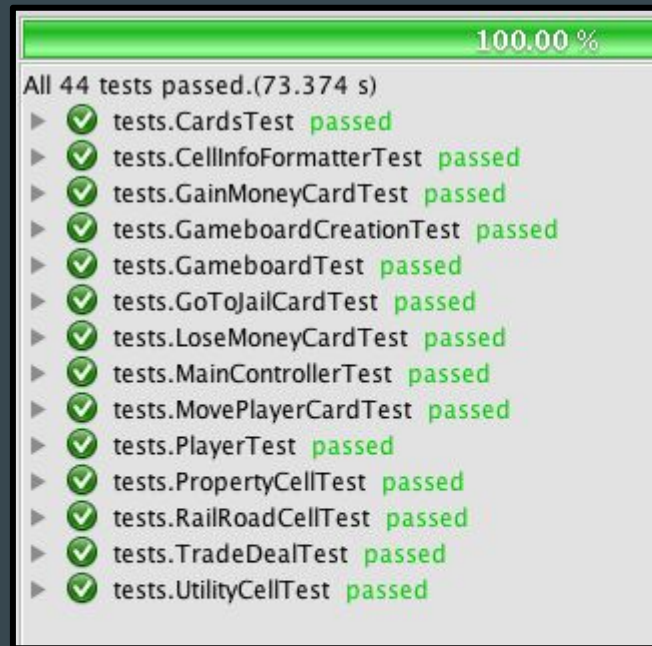
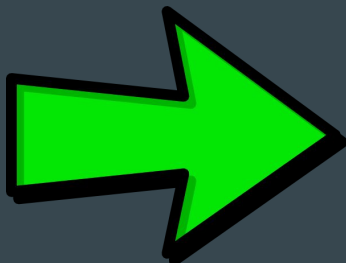
File System



Unit Tests

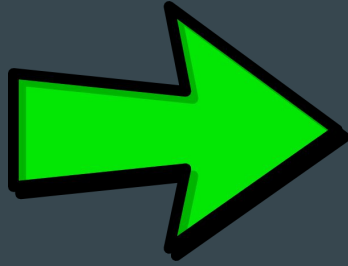


```
compile:
compile-test:
test-report:
test:
BUILD SUCCESSFUL (total time: 0 seconds)
```



Community Chest / Chance

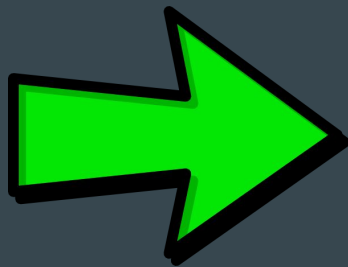
Vermont Avenue \$100 Owner: * 0	Chance 1	Oriental Avenue \$100 Owner: * 0
--	----------	---



Vermont Avenue \$100 Owner: * 0	Chance 1	Oriental Avenue \$100 Owner: * 0
--	----------	---

Warnings

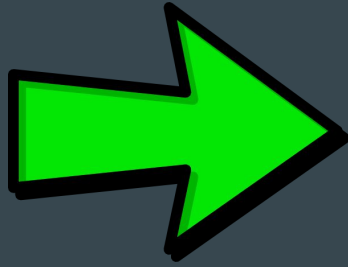
```
MockGUI.java
Source History
1 package monopoly;
2
3 public class MockGUI implements MonopolyGUI {
4     private boolean btnDrawCardState, btnEndTurnState;
5     private boolean[] btnTradeState = new boolean[2];
6
7     public void enableEndTurnBtn(int playerIndex) {
8     }
9
10    public void enablePlayerTurn(int playerIndex) {
11    }
12
13    public void enablePurchaseBtn(int playerIndex) {
14    }
15
16    public int[] getDiceRoll() {
17        int roll[] = new int[2];
18        roll[0] = 2;
19        roll[1] = 3;
20        return roll;
21    }
22
23    public boolean isDrawCardButtonEnabled() {
24        return btnDrawCardState;
25    }
26
27    public boolean isEndTurnButtonEnabled() {
28        return btnEndTurnState;
29    }
30
31    public boolean isGetOutOfJailButtonEnabled() {
32        return btnGetOutOfJailState;
33    }
34 }
```



```
MockGUI.java
Source History
1 package tests.mocks;
2
3 import monopoly.gui.MonopolyGUI;
4 import monopoly.Player;
5 import monopoly.RespondDialog;
6 import monopoly.TradeDeal;
7 import monopoly.TradeDialog;
8
9 public class MockGUI implements MonopolyGUI {
10     private boolean btnDrawCardState, btnEndTurnState, btnG
11     private final boolean[] btnTradeState = new boolean[2];
12
13     @Override
14     public void enableEndTurnBtn(int playerIndex) {}
15
16     @Override
17     public void enablePlayerTurn(int playerIndex) {}
18
19     @Override
20     public void enablePurchaseBtn(int playerIndex) {}
21
22     @Override
23     public int[] getDiceRoll() {
24         int roll[] = new int[2];
25         roll[0] = 2;
26         roll[1] = 3;
27         return roll;
28     }
29
30     @Override
31     public boolean isDrawCardButtonEnabled() {
```

Modern Libraries

```
private ArrayList cells = new ArrayList();
private ArrayList chanceCards = new ArrayList();
//the key of colorGroups is the name of the color group.
private Hashtable colorGroups = new Hashtable();
private ArrayList communityChestCards = new ArrayList();
private GameMaster gameMaster;
```



```
private final ArrayList<Cell> cells = new ArrayList<>();
private final ArrayList<Card> chanceCards = new ArrayList<>();
//the key of colorGroups is the name of the color group.
private final Map<String, Integer> colorGroups = new HashMap<>();
private final ArrayList<Card> communityChestCards = new ArrayList<>();
```

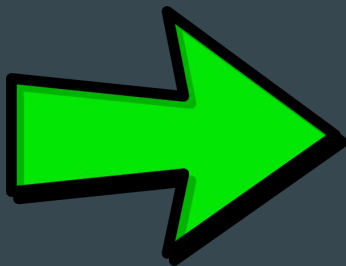

Consistent Formatting

```
public void addCard(Card card) {
    if(card.getCardType() == Card.TYPE_CC) {
        communityChestCards.add(card);
    } else {
        chanceCards.add(card);
    }
}

public void addCell(Cell cell) {
    cells.add(cell);
}

public void addCell(PropertyCell cell) {
    int propertyNumber = getPropertyNumberForColor(cell.getColorGroup());
    colorGroups.put(cell.getColorGroup(), new Integer(propertyNumber + 1));
    cells.add(cell);
}

public Card drawCCCard() {
    Card card = (Card)communityChestCards.get(0);
    communityChestCards.remove(0);
    addCard(card);
    return card;
}
```



```
public void addCard(Card card) {
    if (card.getCardType() == Card.TYPE_CC) {
        communityChestCards.add(card);
    } else {
        chanceCards.add(card);
    }
}

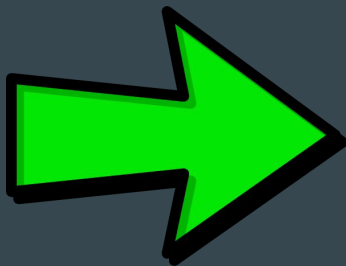
public final void addCell(Cell cell) {
    cells.add(cell);
}

public void addCell(PropertyCell cell) {
    int propertyNumber = getPropertyNumberForColor(cell.getColorGroup());
    colorGroups.put(cell.getColorGroup(), propertyNumber + 1);
    cells.add(cell);
}

public Card drawCCCard() {
    Card card = communityChestCards.remove(0);
    addCard(card);
    return card;
}
```

Loops

```
public PropertyCell[] getPropertiesInMonopoly(String color) {
    PropertyCell[] monopolyCells =
        new PropertyCell[getPropertyNumberForColor(color)];
    int counter = 0;
    for (int i = 0; i < getCellNumber(); i++) {
        Cell c = getCell(i);
        if (c instanceof PropertyCell) {
            PropertyCell pc = (PropertyCell)c;
            if (pc.getColorGroup().equals(color)) {
                monopolyCells[counter] = pc;
                counter++;
            }
        }
    }
    return monopolyCells;
}
```



```
public PropertyCell[] getPropertiesInMonopoly(String color) {
    PropertyCell[] monopolyCells =
        new PropertyCell[getPropertyNumberForColor(color)];
    int counter = 0;
    for (Object cell : cells) {
        if (cell instanceof PropertyCell) {
            PropertyCell pc = (PropertyCell)cell;
            if (pc.getColorGroup().equals(color))
                monopolyCells[counter++] = pc;
        }
    }
    return monopolyCells;
}
```