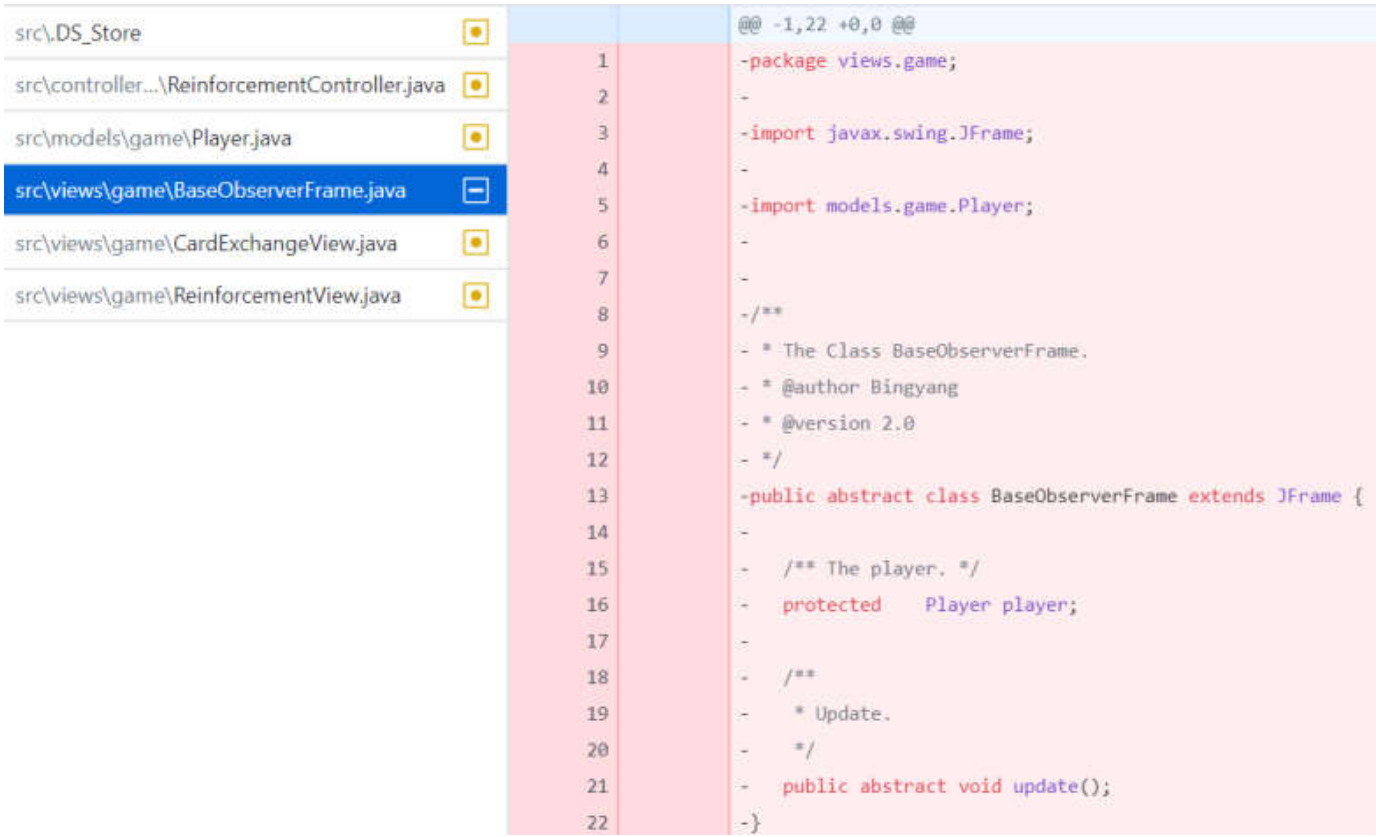
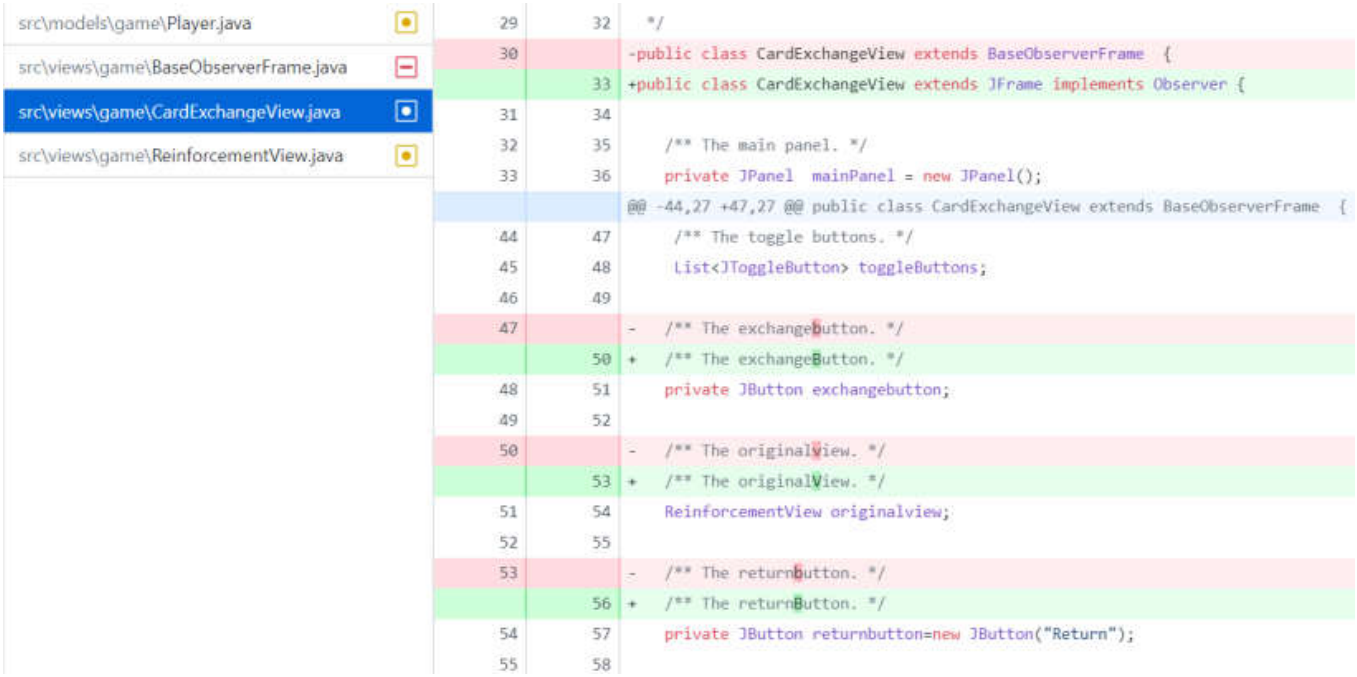
Refactoring after build #2

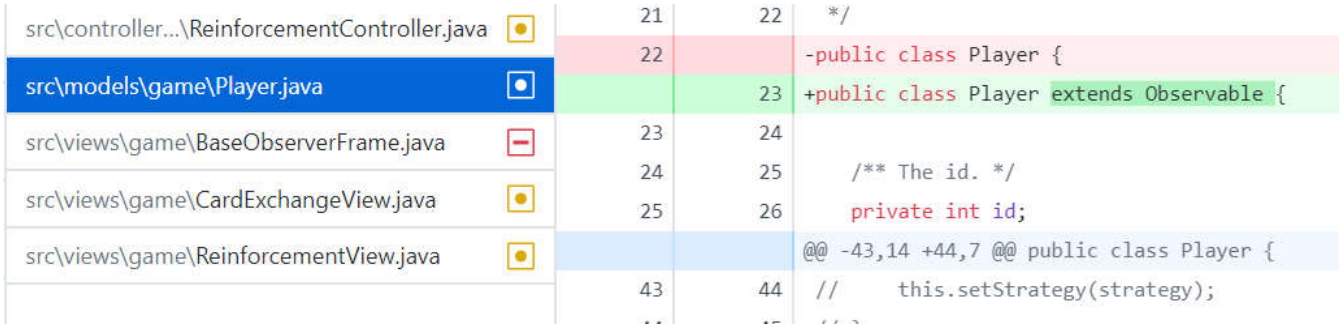
	List of potential refactoring targets	***Reason for choosing***
1	In class “CardExchangeView”, use the java build-in “observer function”, set the class to be observer, to observe class “Player” (method “removeCard()”, “getNewCard()”, to replace the observer function we created	To increase compatibility
2	In class “AttackController”, consolidate Duplicate Conditional Fragments	To improve code reusability and understandability.
3	In class “Card” of game model, extract the “cardType” to be a separate class	To improve reusability and lower coupling
4	Separate SetupView from Reinforcement View to decrease coupling	
5	Move inner class “PhaseState” out of class “GameState” to decrease coupling	
6	In Attack phase, create a new method in the player class for the attack action	

Refactoring operations applied:

1. In class “CardExchangeView”, use the java build-in “observer function”, set the class to be observer, to observe class “Player” (method “removeCard()”, “getNewCard()”, to replace the observer function we created, to increase compatibility.







Test:

```
/**
 * This test case tests calculation of number of armies in exchange with cards
 */
@Test
public void testExchangeArmyForCard() {
    int expectedResult = 15;
    Card c1=new Card(player1);
    Card c2=new Card(player1);
    Card c3=new Card(player1);
    c1.setCardType(CardType.ARTILLERY);
    c2.setCardType(CardType.CAVALRY);
    c3.setCardType(CardType.INFANTRY);

    List<Card> exchangeList=new ArrayList<Card>();
    exchangeList.add(c1);
    exchangeList.add(c2);
    exchangeList.add(c3);

    int result = player1.exchangeCardforArmy(exchangeList);
    assertEquals(expectedResult, result);

    expectedResult = 5;
    Card c6=new Card(player2);
    Card c4=new Card(player2);
    Card c5=new Card(player2);
    c6.setCardType(CardType.ARTILLERY);
    c4.setCardType(CardType.CAVALRY);
    c5.setCardType(CardType.INFANTRY);
    List<Card> exchangeList2=new ArrayList<Card>();

    exchangeList.add(c4);
    exchangeList.add(c5);
    exchangeList.add(c6);

    result = player2.exchangeCardforArmy(exchangeList2);
    assertEquals(expectedResult, result);
}
```

2. In class “AttackController”, consolidate Duplicate Conditional Fragments to improve code reusability and understandability.

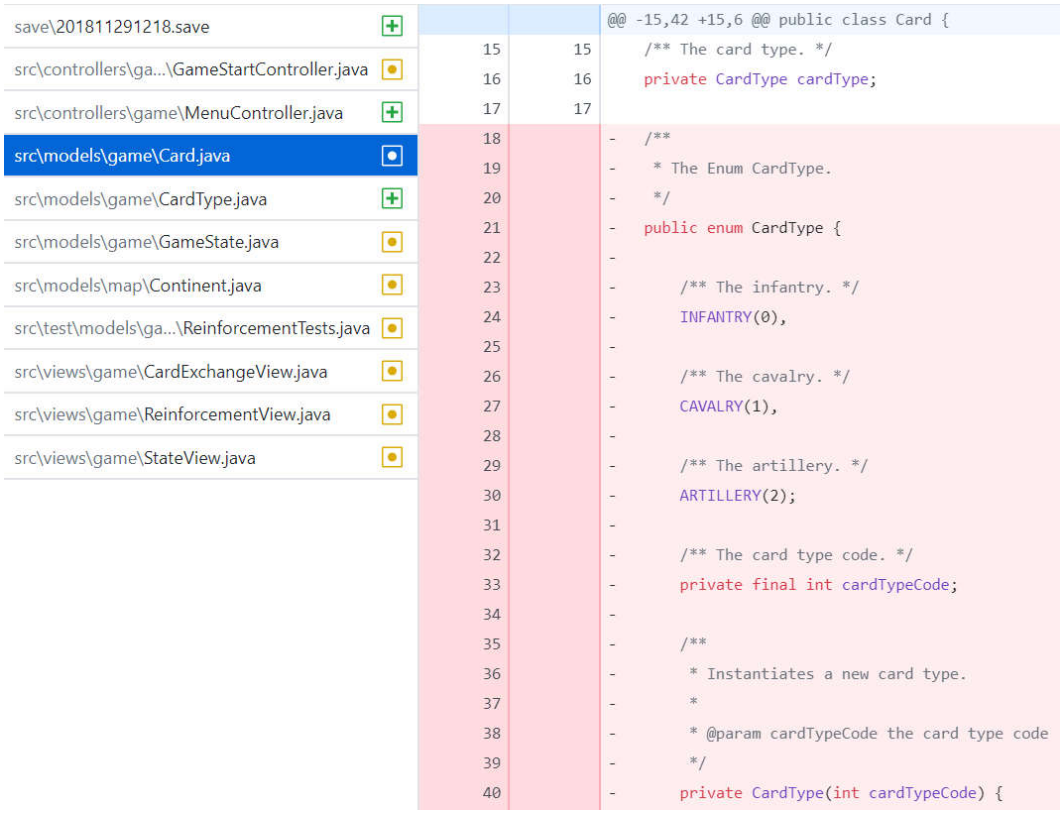
```
70  71          case AttackView.ContinueStr:
71  -               Country fromCountry = attackView.getSelecterdCountryFrom();
72  -               Country toCountry = attackView.getSelecterdCountryTo();
73  -               int diceNumber = attackView.getAttacherDiceNumber();
74  -
75  -               if (GameState.getInstance().getCurrentPlayer().conquer(toCountry)) {
76  -                   if (GameState.getInstance().getMap().mapOwner(GameState.getInstance().getCurrentPlayer()))
77  -                       StateView.getInstance().showEndGameView();
78  -                   else
79  -                       attackView.showMoveArmiesState(diceNumber);
80  -               }
81  -               else {
82  -                   if(GameState.getInstance().getCurrentPlayer().getArmyNumber() == 0) {
83  -                   // current player ended his/her turn.
84  -                   GameState.getInstance().endPlayerTurn();
85  -                   StateView.getInstance().getMapPanel().addCountryTableForMap(GameState.getInstance().getMap());
86  -
87  -                   GameState.getInstance().setPhase(Phase.REINFORCEMENT);
88  -                   StateView.getInstance().showReinforcementView();
89  -                   }
90  -                   else {
91  -                       if(GameState.getInstance().getCurrentPlayer().isAttackPossible())
92  -                           attackView.showSelectionState();
```

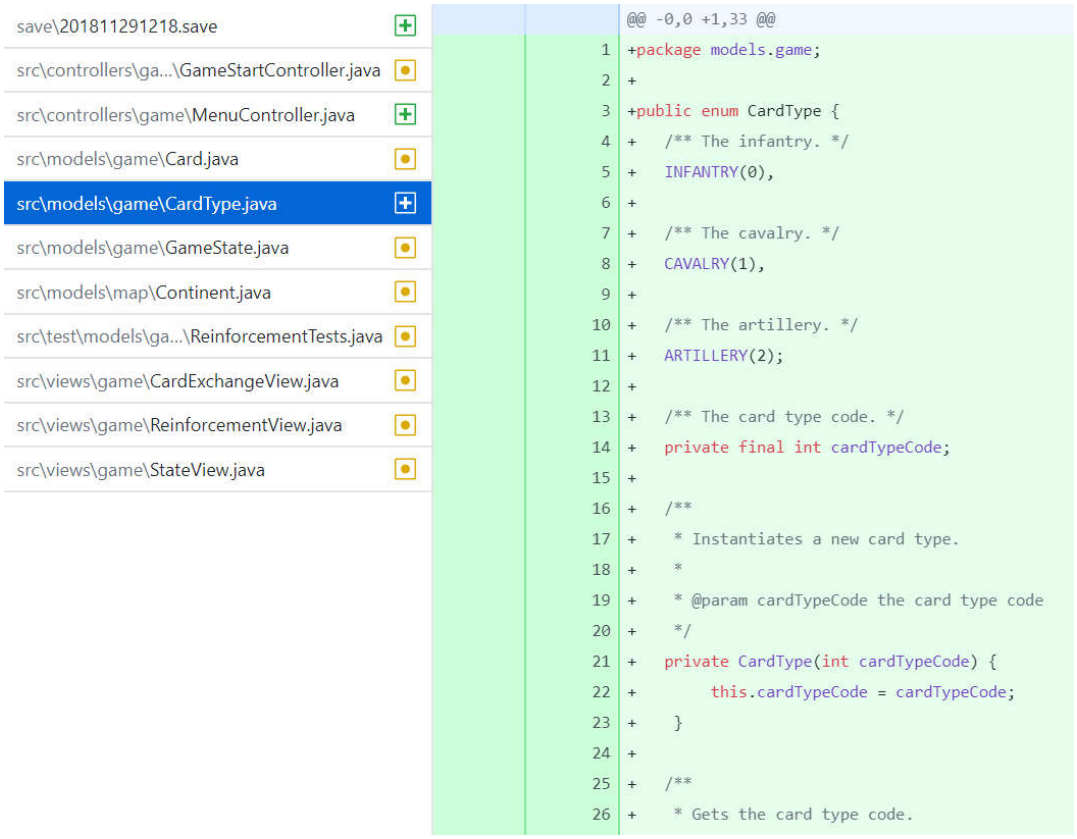
```
123 -
124 -           //if(defenderCountry.getNumOfArmies() == 0) {
125 -           if (GameState.getInstance().getCurrentPlayer().conquer(defenderCountry)) {
126 -               //defenderCountry.setOwner(GameState.getInstance().getCurrentPlayer());
127 -               if (GameState.getInstance().getMap().mapOwner(GameState.getInstance().getCurrentPlayer()))
128 -                   StateView.getInstance().showEndGameView();
129 -               else
130 -                   attackView.showMoveArmiesState(diceNo);
131 -           }
132 -           else {
133 -               if(GameState.getInstance().getCurrentPlayer().getArmyNumber() == 0) {
134 -               // current player ended his/her turn.
135 -               GameState.getInstance().endPlayerTurn();
136 -               StateView.getInstance().getMapPanel().addCountryTableForMap(GameState.getInstance().getMap());
137 -
138 -               GameState.getInstance().setPhase(Phase.REINFORCEMENT);
139 -               StateView.getInstance().showReinforcementView();
140 -               }
141 -               else {
142 -                   if(GameState.getInstance().getCurrentPlayer().isAttackPossible())
143 -                       attackView.showSelectionState();
144 -                   else {
145 -                       GameState.getInstance().setPhase(Phase.FORTIFICATION);
146 -                       StateView.getInstance().showFortificationView();
147 -                   }
148 -               }
149 -           }
```

```
125 +   private void checkNextStep() {
126 +       if (GameState.getInstance().getCurrentPlayer().conquer(defenderCountry)) {
127 +           if (GameState.getInstance().getMap().mapOwner(GameState.getInstance().getCurrentPlayer()))
128 +               StateView.getInstance().showEndGameView();
129 +           else
130 +//             attackView.showMoveArmiesState(fromCountry.getNumOfArmies()<=diceNumber ? diceNumber-1 : diceNu
    mber);
131 +               attackView.showMoveArmiesState(1);
132 +       }
133 +       else {
134 +           // in case that attacker lost the country
135 +           if (defenderCountry.getNumOfArmies() == 0) {
136 +               defenderCountry.setOwner(attackerCountry.getOwner());
137 +               attackerCountry.getOwner().moveArmies(defenderCountry, attackerCountry, 1);
138 +           }
139 +
140 +           if(GameState.getInstance().getCurrentPlayer().getArmyNumber() == 0) {
141 +           // current player ended his/her turn.
142 +           GameState.getInstance().endPlayerTurn();
143 +           GameState.getInstance().setPhase(Phase.REINFORCEMENT);
144 +           StateView.getInstance().showReinforcementView();
145 +           }
146 +           else {
147 +               if(GameState.getInstance().getCurrentPlayer().isAttackPossible())
148 +                   attackView.showSelectionState();
```

```java
/**
 * test for attack of Conquer
 */
@Test
public void testAttackConquer(){
    int ArmyNumberDefenderCountry = defenderCountry.getNumOfArmies();

    // Player0 attacks player1, until player1 has no army left in the defenderCountry
    defenderCountry.removeArmies(ArmyNumberDefenderCountry);
    player0.conquer(defenderCountry);
    assertTrue(defenderCountry.getOwner()==player0);
}


/**
 * test for attack of valid move after conquering
 */
@Test
public void testValidMoveAfterConquering(){
    int ArmyNumberDefenderCountry = defenderCountry.getNumOfArmies();

    // Player0 attacks player1, until player1 has no army left in the defenderCountry
    defenderCountry.removeArmies(ArmyNumberDefenderCountry);
    player0.conquer(defenderCountry);
    player0.moveArmies(attackerCountry, defenderCountry, 5);

    assertTrue(defenderCountry.getNumOfArmies()==5);

}
```

3. In class “Card” of game model, extract the “cardType” to be a separate class to improve reusability and lower coupling.


save\201811291218.save
src\controllers\ga...\GameStartController.java
src\controllers\game\MenuController.java
src\models\game\Card.java
src\models\game\CardType.java
src\models\game\GameState.java
src\models\map\Continent.java
src\test\models\ga...\ReinforcementTests.java
src\views\game\CardExchangeView.java
src\views\game\ReinforcementView.java
src\views\game\StateView.java
@@ -15,42 +15,6 @@ public class Card {
/** The card type. */
private CardType cardType;
/**
* The Enum CardType.
*/
public enum CardType {
/** The infantry. */
INFANTRY(0),
/** The cavalry. */
CAVALRY(1),
/** The artillery. */
ARTILLERY(2);
/** The card type code. */
private final int cardTypeCode;
/**
* Instantiates a new card type.
*
* @param cardTypeCode the card type code
*/
private CardType(int cardTypeCode) {


save\201811291218.save
src\controllers\ga...\GameStartController.java
src\controllers\game\MenuController.java
src\models\game\Card.java
src\models\game\CardType.java
src\models\game\GameState.java
src\models\map\Continent.java
src\test\models\ga...\ReinforcementTests.java
src\views\game\CardExchangeView.java
src\views\game\ReinforcementView.java
src\views\game\StateView.java
@@ -0,0 +1,33 @@
+package models.game;
+
+public enum CardType {
+ /** The infantry. */
+ INFANTRY(0),
+
+ /** The cavalry. */
+ CAVALRY(1),
+
+ /** The artillery. */
+ ARTILLERY(2);
+
+ /** The card type code. */
+ private final int cardTypeCode;
+
+ /**
+ * Instantiates a new card type.
+ *
+ * @param cardTypeCode the card type code
+ */
+ private CardType(int cardTypeCode) {
+ this.cardTypeCode = cardTypeCode;
+ }
+
+ /**
+ * Gets the card type code.

Test:

```java
/**
 * This class tests card settings of the game
 * @author Lynn
 *
 */
public class CardTest {
    String cardTypeStr = "INFANTRY";
    String CCardTypeStr;

    /**
     * Set up before test
     */
    @Before
    public void setUp(){
        GameState.reset();
        Player player0 = new Player();
        GameState.getInstance().getPlayerList().add(player0);
        Card c = new Card(player0);
        c.setCardType(CardType.INFANTRY);
        CCardTypeStr = c.getCardType().toString();
    }

    @Test
    public void test() {
        assertTrue(cardTypeStr == CCardTypeStr);
    }
}
```