```java
package game;

import game.Account.IllegalAmountException;

public class GameController {
    private static final int DEFAULT_START_BALANCE = 30000;
    private static final int DEFAULT_NUMBER_OF_FIELDS = 40;
    private Decorator decorator;
    private BoardController boardController;
    private PlayerController playerController;
    private CardController cardController;
    private DiceCup dicecup;

    public GameController(String language) {
        super();
        this.decorator = new Decorator(language);
        this.dicecup = DiceCup.getInstance();
        this.boardController = new BoardController(DEFAULT_NUMBER_OF_FIELDS);
        this.cardController= new CardController();
        this.playerController = new PlayerController(DEFAULT_START_BALANCE, DEFAULT_NUMBER_OF_FIELDS);
    }
    public void setupGame() {
        decorator.setupGUI(BoardController.getBoard().getFields(), null);

        playerController.playerSetup(this, decorator, DEFAULT_START_BALANCE);
        decorator.setupGUI(BoardController.getBoard().getFields(), playerController.getPlayers());
    }

    public void runGame () {
        while (playerController.getNumberOfPlayersLeft() > 1) {
            for (Player activePlayer : playerController.getPlayers()) {
                //check if player is in game
                if (!activePlayer.isBroke()) {
                    if (activePlayer.getInJail() != 0) {
                        jailTurn(activePlayer);
                    } else {
                        playerTurn(activePlayer);
                    }
                }
                if (playerController.getNumberOfPlayersLeft() <= 1) {
                    break;
                }
            }
            //Main GameLoop
        }
        for (Player winner : playerController.getPlayers()){
```

```
49              if(!winner.isBroke()){
50                  decorator.showMessage(new String[]{"Congratulations",", ",
  winner.getPlayerName(), "YouWon"});
51              }
52          }
53          System.exit(1);
54
55      }
56
57
58      private void playerTurn(Player activePlayer) {
59          activePlayer.setTwoOfAKindCount(0);
60          do {
61              //Slå med terninger og updater GUI
62              decorator.showMessage(new
  String[]{activePlayer.getPlayerName(), "- ","RollDice","!"});
63              // Ornamentation
64              for (int i = 0;i<8;i++){
65                  dicecup.rollDice();
66                  decorator.updateDice(dicecup);
67                  try {
68                      Thread.sleep(100);
69                  } catch(InterruptedException ex) {
70                      Thread.currentThread().interrupt();
71                  }
72              }
73              //endOf Ornamentation - real diceRoll...
74              int diceRoll = dicecup.rollDice();
75              decorator.updateDice(dicecup);
76              twoOfAKindCheck(activePlayer);
77              playerController.move(activePlayer, diceRoll, decorator,
78                      playerController,  cardController,  boardController);
79              decorator.updatePlayer(activePlayer);
80
81              //Delegates control to boardController...
82              //boardController.landOnField(activePlayer,
83              //                  activePlayer.getCurrentFieldNumber(),
  decorator,
84              //                  playerController, cardController);
85              if (activePlayer.getInJail()==0) {
86                  //Offer Player to buy houses and trade
87                  int UserSelection = 0;
88                  String[] msg = new String[] { "OfferBuyingTrading" };
89                  String[] options = new String[] { "NextTurn", "BuyHouses",
90                  "TradePawn" };
91                  do {
92                      UserSelection = decorator
93                              .getUserButtonPressed(msg, options);
94                      if (UserSelection == 2) {
95                          System.out.println("trading");
```

```
 96                          playerController.trade(activePlayer,decorator);
 97                      }
 98                      if (UserSelection ==1){
 99                          System.out.println("Buying Houses");
100                          playerController.buyHouse(activePlayer, decorator);
101                      }
102                  } while (UserSelection != 0);
103              }
104          } while
    (activePlayer.getTwoOfAKindCount()>0&&activePlayer.getTwoOfAKindCount()<3);
105      }
106
107      private void twoOfAKindCheck(Player activePlayer) {
108          if (dicecup.getTwoOfAKind()!=0) {
109              activePlayer.setTwoOfAKindCount(activePlayer.getTwoOfAKindCount(
    )+1);
110          }else{
111              activePlayer.setTwoOfAKindCount(0);
112          }
113          if (activePlayer.getTwoOfAKindCount() >= 3){
114              decorator.showMessage(new String[]
    {"TooManyOneOfAKind","GoToJail"});
115              playerController.moveToJail(activePlayer,
    boardController.getJailNumber());
116              activePlayer.setInJail(1);
117          }
118          if (activePlayer.getTwoOfAKindCount() > 0 &&
    activePlayer.getTwoOfAKindCount() < 3){
119              decorator.showMessage(new String[]
    {activePlayer.getPlayerName(),"TwoOfAKindExtraTurn"});
120          }
121      }
122      //TODO Refactor - extract method
123      private void afterJailTurn(Player activePlayer, int diceRoll){
124          decorator.showMessage(new String[]
    {activePlayer.getPlayerName(),"TwoOfAKindExtraTurn"});
125          playerController.move(activePlayer, diceRoll, decorator,
126                  playerController,  cardController,  boardController);
127          decorator.updatePlayer(activePlayer);
128
129          //Offer Player to buy houses and trade
130          int UserSelection = 0;
131          String[] msg = new String[] { "OfferBuyingTrading" };
132          String[] options = new String[] { "NextTurn", "BuyHouses",
133          "TradePawn" };
134          do {
135              UserSelection = decorator
136                      .getUserButtonPressed(msg, options);
137              if (UserSelection == 2) {
138                  System.out.println("trading");
```

```
139                 playerController.trade(activePlayer,decorator);
140             }
141         } while (UserSelection != 0);
142         if(activePlayer.getTwoOfAKindCount() == 2){
143             playerTurn(activePlayer);
144         }
145     }
146
147
148
149     //
150     private void jailTurn(Player activePlayer) {
151         if(activePlayer.getInJail() >= 3){
152             boolean raisedMoney = false;
153             while(!raisedMoney){
154                 try {
155                     activePlayer.getAccount().withdraw(boardController.getJ
    ail().getBail());
156                     raisedMoney = true;
157                 } catch (InsufficientFundsException e) {
158                     if(playerController.handleInsufficientFunds(activePlaye
    r, boardController.getJail().getBail(), decorator)){
159                         raisedMoney = true;
160                     }
161                     else{
162                         playerController.hostileTakeOver(null,
    activePlayer);
163                     }
164
165                 }
166                 catch (IllegalAmountException e) {
167                     System.err.println("IllegalAmount - Jailturn");
168                     e.printStackTrace();
169                 }
170             }
171             activePlayer.setInJail(0);
172             String[] msg2 = new String[] {"CongratsYouAreFree"};
173             decorator.showMessage(msg2);
174             playerTurn(activePlayer);
175         } else {
176             String[] msg = new String[] {"YouAreInJail"};
177             String[] options = new String[] {};
178             options = StringTools.add(options, new String[] {"RollDice"});
179             if(activePlayer.getAccount().getBalance() >=
    boardController.getJailNumber()){
180                 options = StringTools.add(options, new String[]{"BuyOut"});
181             }
182             if(activePlayer.getNumberOfJailCards()>0){
183                 options = StringTools.add(options, new
    String[]{"UsePardon"});
```

```
184             }
185             int jailChoice = decorator.getUserButtonPressed(msg, options);
186             if (jailChoice == 0){
187                 int diceRoll =  dicecup.rollDice();
188                 decorator.updateDice(dicecup);
189 
190                 if (dicecup.getTwoOfAKind()!=0) {
191                     activePlayer.setTwoOfAKindCount(activePlayer.getTwoOfAK
    indCount()+1);
192                     activePlayer.setInJail(0);
193                     afterJailTurn(activePlayer, diceRoll);
194                     String[] msg3 = new String[]{"CongratsYouAreFree"};
195                     decorator.showMessage(msg3);
196                 } else{
197                     activePlayer.setInJail(activePlayer.getInJail() + 1);
198                 }
199 
200             }
201             else if (jailChoice == 1){
202                 try {
203                     activePlayer.getAccount().withdraw(boardController.getJ
    ail().getBail());
204                 } catch (Exception e) {
205                     e.printStackTrace();
206                 }
207                 activePlayer.setInJail(0);
208                 String[] msg1 = new String[] {"CongratsYouAreFree"};
209                 decorator.showMessage(msg1);
210                 playerTurn(activePlayer);
211             }
212             else if (jailChoice == 2){
213                 activePlayer.setNumberOfJailCards(activePlayer.getNumberOfJ
    ailCards() - 1);
214                 activePlayer.setInJail(0);
215                 String[] msg3 = new String[]{"CongratsYouAreFree"};
216                 decorator.showMessage(msg3);
217                 cardController.useJailCard(activePlayer);
218                 playerTurn(activePlayer);
219             }
220         }
221     }
222 
223     public static void main(String[] args){
224         GameController testController = new GameController("danish");
225         Decorator testdc = testController.decorator;
226 
227         CardController testCC = testController.cardController;
228         BoardController testBC = testController.boardController;
229 
230         //testController.setupGame();
```

```
231         //TestFixture
232         //testdc.setupGUI(BoardController.getBoard().getFields(), null);
233         Player[] players = new Player[]{new Player("Jack Sparrow", 1000),
    new Player("Captain Barbossa", 20000), new Player("Bootstrap Bill",
    10000)};
234         PlayerController testPc = new PlayerController(players,
    DEFAULT_START_BALANCE, DEFAULT_NUMBER_OF_FIELDS);
235         testController.playerController = testPc;
236         Player jack = players[0];
237         Player bill = players[2];
238         testController.decorator.setupGUI(BoardController.getBoard().getFie
    lds(), testController.playerController.getPlayers());
239         ((game.fields.Ownable)BoardController.getBoard().getFields()
    [39]).setOwner(bill);
240         ((game.fields.Ownable)BoardController.getBoard().getFields()
    [37]).setOwner(bill);
241         ((game.fields.Street)BoardController.getBoard().getFields()
    [39]).addBuilding();
242         testdc.updateHouses((game.fields.Ownable)BoardController.getBoard()
    .getFields()[39]);
243         testdc.updateHouses((game.fields.Ownable)BoardController.getBoard()
    .getFields()[37]);
244         testdc.updateFieldOwner((game.fields.Ownable)BoardController.getBoa
    rd().getFields()[39]);
245         testdc.updateFieldOwner((game.fields.Ownable)BoardController.getBoa
    rd().getFields()[37]);
246         testController.playerController.move(jack, 4, testdc, testPc,
    testCC, testBC);
247     testController.playerController.move(bill, 6, testdc, testPc, testCC,
    testBC);
248 //      testController.playerController.move(bill, 33, testdc, testPc,
    testCC, testBC);
249 //      testController.playerController.move(bill, 7, testdc, testPc,
    testCC, testBC);
250 //      testController.playerController.move(bill, 0, testdc, testPc,
    testCC, testBC);
251 //      testController.playerController.move(bill, 0, testdc, testPc,
    testCC, testBC);
252
253         testController.runGame();
254     }
255 }
256
```