```java
package Chancecards;

import game.Player;
import game.Account;

public class ChanceCardBuildingTaxController extends ChanceCardController{
	private ChanceCardBuildingTaxData chanceCardData;
	private Account parkinglotAccount;

	public ChanceCardBuildingTaxController(ChanceCardBuildingTaxData chanceCardData, Account parkinglotAccount) {
		super(chanceCardData);
		this.chanceCardData = chanceCardData;
		this.parkinglotAccount = parkinglotAccount;
	}

	@Override
	public boolean onDrawn(Player player) {
		int hu = player.getProperty().getTotalHouseCount();
		int ho = player.getProperty().getTotalHotelCount();
		int housecount = chanceCardData.getHouseTax();
		int hotelcount = chanceCardData.getHotelTax();

		if(hu+ho > 0) {
			player.getAccount().transferTo(parkinglotAccount, housecount*hu);
			player.getAccount().transferTo(parkinglotAccount, hotelcount*ho);
			return false;
		} else {
			return true;
		}
	}

	public String toString(){
		return chanceCardData.toString() + ", parkinglotAccount=" + parkinglotAccount;
	}

}
```

```
 1: package Chancecards;
 2:
 3: public class ChanceCardBuildingTaxData extends ChanceCardData{
 4:         private int houseTax;
 5:         private int hotelTax;
 6:
 7:         public ChanceCardBuildingTaxData(int translateID, int houseTax, int hotelTax) {
 8:                 super(translateID);
 9:                 this.houseTax = houseTax;
10:                 this.hotelTax = hotelTax;
11:         }
12:
13:         public int getHouseTax() {
14:                 return houseTax;
15:         }
16:
17:         public int getHotelTax() {
18:                 return hotelTax;
19:         }
20:
21:         public String toString(){
22:                 return "getHouseTax()=" + getHouseTax() + ", getHotelTax()=" + getHotelTax();
23:         }
24:
25: }
```

```java
package Chancecards;

import game.Player;

public class ChanceCardCashBonusController extends ChanceCardController{
	private ChanceCardCashData chanceCardCashData;

	public ChanceCardCashBonusController(ChanceCardCashData chanceCardCashBonusData) {
		super(chanceCardCashBonusData);
		this.chanceCardCashData = chanceCardCashBonusData;
	}

	@Override
	public boolean onDrawn(Player player) {
		player.getAccount().addGold(chanceCardCashData.getMoney());
		return false;
	}

	public String toString(){
		return chanceCardCashData.toString();
	}

}
```

```
 1: package Chancecards;
 2:
 3: public class ChanceCardCashData extends ChanceCardData{
 4:         private int money;
 5:
 6:         public ChanceCardCashData(int translateID, int money) {
 7:                 super(translateID);
 8:                 this.money = money;
 9:         }
10:
11:         public int getMoney() {
12:                 return money;
13:         }
14:
15:         public String toString(){
16:                 return "getMoney()=" + getMoney();
17:         }
18: }
```

```java
package Chancecards;


import game.Player;

public class ChanceCardCashTransferController extends ChanceCardController{
        private ChanceCardCashData chanceCardData;
        private  Player[]  players;

        public ChanceCardCashTransferController(ChanceCardCashData chanceCardData, Player[] players) {
                super(chanceCardData);
                this.chanceCardData = chanceCardData;
                this.players = players;
        }

        @Override
        public boolean onDrawn(Player player) {
                for(Player p : players)
                {
                        if(p != player && p != null) {
                                p.getAccount().transferTo(player.getAccount(), chanceCardData.getMoney());
                        }
                }
                return false;
        }

        public String toString(){
                return chanceCardData.toString();
        }


}
```

```
 1: package Chancecards;
 2:
 3: import game.Player;
 4: import game.Translator;
 5:
 6: public abstract class ChanceCardController {
 7:         private ChanceCardData dat;
 8:
 9:         ChanceCardController(ChanceCardData dat)
10:         {
11:                 this.dat = dat;
12:         }
13:
14:         public abstract boolean onDrawn(Player player);
15:
16:         public final String getDescription()
17:         {
18:                 return Translator.getString("CHANCECARDDSC"+dat.getTranslateID());
19:         }
20:
21:         public String toString(){
22:                 return dat.toString();
23:         }
24: }
```

```
 1: package Chancecards;
 2:
 3: public class ChanceCardData {
 4:         protected int translateID;
 5:
 6:         public int getTranslateID() {
 7:                 return translateID;
 8:         }
 9:
10:         public ChanceCardData(int translateID) {
11:                 this.translateID = translateID;
12:         }
13:
14:         public String toString(){
15:                 return "getTranslateID()=" + getTranslateID();
16:         }
17: }
```

```java
package Chancecards;

import game.Player;
import game.Prison;

public class ChanceCardGoToPrisonController extends ChanceCardController{
	private Prison prison;
	private ChanceCardGoToPrisonData chanceCardData;

	public ChanceCardGoToPrisonController(ChanceCardGoToPrisonData chanceCardData, Prison prison) {
		super(chanceCardData);
		this.chanceCardData = chanceCardData;
		this.prison = prison;
	}

	@Override
	public boolean onDrawn(Player player) {
		prison.addInmate(player);
		player.setNextPosition(chanceCardData.getPrisonLocation(), false);
		return false;
	}

	public String toString(){
		return chanceCardData.toString();
	}

}
```

```
 1: package Chancecards;
 2:
 3: public class ChanceCardGoToPrisonData extends ChanceCardData{
 4:         private int prisonLocation;
 5:
 6:         public ChanceCardGoToPrisonData(int translateID, int prisonLocation) {
 7:                 super(translateID);
 8:                 this.prisonLocation = prisonLocation;
 9:         }
10:
11:         public int getPrisonLocation() {
12:                 return prisonLocation;
13:         }
14:
15:         public String toString(){
16:                 return "getPrisonLocation()=" + getPrisonLocation();
17:         }
18: }
```

```
 1: package Chancecards;
 2:
 3: import game.Player;
 4:
 5: public class ChanceCardMatadorLegatController extends ChanceCardController{
 6: private ChanceCardCashData chanceCardData;
 7:
 8:         public ChanceCardMatadorLegatController(ChanceCardCashData chanceCardData) {
 9:                 super(chanceCardData);
10:                 this.chanceCardData = chanceCardData;
11:         }
12:
13:
14:         @Override
15:         public boolean onDrawn(Player player) {
16:                 final int MAXALLOWEDVALUE = 15000;
17:                 int g = player.getAccount().getGold();
18:                 int n = player.getProperty().getPropertyWorth();
19:                 if(g + n < MAXALLOWEDVALUE){
20:                         player.getAccount().addGold(chanceCardData.getMoney());
21:                         return false;
22:                 }
23:                 else {
24:                         return true;
25:                 }
26:         }
27:
28:         public String toString(){
29:                 return chanceCardData.toString();
30:         }
31:
32: }
```

```java
package Chancecards;

import game.Player;

public class ChanceCardMovePlayerController extends ChanceCardController{
	private ChanceCardMovePlayerData chanceCardData;

	public ChanceCardMovePlayerController(ChanceCardMovePlayerData chanceCardData) {
		super(chanceCardData);
		this.chanceCardData = chanceCardData;
	}

	@Override
	public boolean onDrawn(Player player) {
		player.setNextPosition(chanceCardData.getFieldPosition(), chanceCardData.getCashInAtStart());
		return false;
	}

	@Override
	public String toString() {
		return chanceCardData.toString();
	}


}
```

```
    1: package Chancecards;
    2:
    3: public class ChanceCardMovePlayerData extends ChanceCardData{
    4:         private int fieldPosition;
    5:         private boolean cashInAtStart;
    6:
    7:         public ChanceCardMovePlayerData(int translateID, int fieldPosition, boolean cashInAtStart) {
    8:                 super(translateID);
    9:                 this.fieldPosition = fieldPosition;
   10:                 this.cashInAtStart = cashInAtStart;
   11:         }
   12:
   13:         public int getFieldPosition() {
   14:                 return fieldPosition;
   15:         }
   16:
   17:         public boolean getCashInAtStart() {
   18:                 return cashInAtStart;
   19:         }
   20:
   21:         @Override
   22:         public String toString() {
   23:                 return "ChanceCardMovePlayerData [fieldPosition=" + fieldPosition + ", cashInAtStart=" + cashInAtStart + "]
";
   24:         }
   25: }
```

```
 1: package Chancecards;
 2:
 3: import game.Player;
 4:
 5: public class ChanceCardMovePlayerRelativeController extends ChanceCardController{
 6:         private ChanceCardMovePlayerRelativeData chanceCardData;
 7:
 8:         public ChanceCardMovePlayerRelativeController(ChanceCardMovePlayerRelativeData chanceCardData) {
 9:                 super(chanceCardData);
10:                 this.chanceCardData = chanceCardData;
11:         }
12:
13:         @Override
14:         public boolean onDrawn(Player player) {
15:                 player.move(chanceCardData.getDistance(), chanceCardData.getCashAtStart());
16:                 return false;
17:         }
18:
19:         @Override
20:         public String toString() {
21:                 return chanceCardData.toString();
22:         }
23:
24:
25:
26: }
```

```java
package Chancecards;

public class ChanceCardMovePlayerRelativeData extends ChanceCardData{
	private int distance;
	private boolean cashAtStart;

	public ChanceCardMovePlayerRelativeData(int translateID, int distance, boolean cashAtStart) {
		super(translateID);
		this.distance = distance;
		this.cashAtStart = cashAtStart;
	}

	public int getDistance() {
		return distance;
	}

	public boolean getCashAtStart() {
		return cashAtStart;
	}

	@Override
	public String toString() {
		return "ChanceCardMovePlayerRelativeData [distance=" + distance + ", cashAtStart=" + cashAtStart + "]";
	}
}
```

```java
package Chancecards;

import game.Player;

public class ChanceCardMoveToNextFleetController extends ChanceCardController{
	private ChanceCardMoveToNextFleetData chanceCardData;

	public ChanceCardMoveToNextFleetController(ChanceCardMoveToNextFleetData chanceCardData) {
		super(chanceCardData);
		this.chanceCardData = chanceCardData;
	}


	@Override
	public boolean onDrawn(Player player) {
		int plrPos = player.getPosition();
		int closestIndex = -1;
		int shortestDistance = Integer.MAX_VALUE;
		int[] fp = chanceCardData.getFleetPositions();
		int fl = chanceCardData.getFleetPositions().length;

		for(int i = 0; i < fl; i++) {
			int distance = fp[i] - plrPos;
			if(distance > 0 && distance < shortestDistance) {
				shortestDistance = distance;
				closestIndex = i;
			}
		}
		if(shortestDistance == Integer.MAX_VALUE){
			for(int j : chanceCardData.getFleetPositions()) {
				if(j < shortestDistance){
					shortestDistance = j;
				}
			}
			player.setNextPosition(shortestDistance, true);
		}
		else
		{
			player.setNextPosition(fp[closestIndex], true);
		}
		return false;
	}


	@Override
	public String toString() {
		return chanceCardData.toString();
	}
```

```
49:
50:
51: }
```

```
    1: package Chancecards;
    2:
    3: import java.util.Arrays;
    4:
    5: public class ChanceCardMoveToNextFleetData extends ChanceCardData{
    6:         private int[] fleetPositions;
    7:         private boolean cashAtStart;
    8:
    9:         public ChanceCardMoveToNextFleetData(int translateID, int[] fleetPositions, boolean cashAtStart) {
   10:                 super(translateID);
   11:                 this.fleetPositions = fleetPositions;
   12:                 this.cashAtStart = cashAtStart;
   13:         }
   14:
   15:         public int[] getFleetPositions() {
   16:                 return fleetPositions;
   17:         }
   18:
   19:         @Override
   20:         public String toString() {
   21:                 return "ChanceCardMoveToNextFleetData [fleetPositions=" + Arrays.toString(fleetPositions) + ", cashAtStart=
"
   22:                                 + cashAtStart + "]";
   23:         }
   24: }
```

```java
package Chancecards;

import game.Player;

public class ChanceCardOutOfPrisonController extends ChanceCardController{
	private ChanceCardData chanceCardData;

	public ChanceCardOutOfPrisonController(ChanceCardData chanceCardData){
		super(chanceCardData);
		this.chanceCardData = chanceCardData;
	}

	@Override
	public boolean onDrawn(Player player) {
		if(player.hasGetOutOfPrisonCard() == false)
		{
			player.setHasGetOutOfPrisonCard(true);
			return false;
		}
		else
		{
			return true;
		}
	}

	@Override
	public String toString() {
		return chanceCardData.toString();
	}


}
```

```
    1: package Chancecards;
    2:
    3: import game.Account;
    4: import game.Player;
    5:
    6: public class ChanceCardTaxController extends ChanceCardController{
    7:         private ChanceCardCashData chanceCardData;
    8:         private Account parkinglotAccount;
    9:
   10:         public ChanceCardTaxController(ChanceCardCashData chanceCardData, Account parkinlotAccount) {
   11:                 super(chanceCardData);
   12:                 this.chanceCardData = chanceCardData;
   13:                 this.parkinglotAccount = parkinlotAccount;
   14:         }
   15:
   16:
   17:         @Override
   18:         public boolean onDrawn(Player player) {
   19:                 player.getAccount().transferTo(parkinglotAccount, chanceCardData.getMoney());
   20:                 return false;
   21:         }
   22:
   23:
   24:         @Override
   25:         public String toString() {
   26:                 return "ChanceCardTaxController [chanceCardData=" + chanceCardData + ", parkinglotAccount=" + parkinglotAcc
ount
   27:                                 + "]";
   28:         }
   29:
   30:
   31:
   32: }
```

```
1: package game;
2:
3: import desktop_resources.GUI;
4:
5: public class Account {
6:         private int gold = 0;
7:         private String ownerName;
8:         public Account(int balance, String name)
9:         {
10:                ownerName = name;
11:                gold = balance;
12:        }
13:        public int getGold() {
14:                return gold;
15:        }
16:        /**
17:         * Attempts to withdraw gold from the player’s account.
18:         * @param gold
19:         * The amount of gold that needs to be withdrawn
20:         * @return
21:         * true if the withdrawal was sucessful, false if not.
22:         */
23:        public boolean withdraw(int gold)
24:        {
25:                if(this.gold<gold)
26:                {
27:                        return false;
28:                }
29:                removeGold(gold);
30:                return true;
31:        }
32:        /**
33:         * Sets the balance in the account, also updates the GUI to the new amount
34:         * @param gold How much gold the account should be set to
35:         */
36:        public void setGold(int gold) {
37:                this.gold = gold;
38:                if(getGold()<0)
39:                {
40:                        setGold(0);
41:                }
42:                if(ownerName!=null)
43:                        GUI.setBalance(ownerName, getGold());
44:        }
45:        /**
46:         * Transfers as much money as possible to the other account. If the balance is too low
47:         * then the reminder of the money is transfered and the account goes to 0.
48:         * @param other
```

```
49:          * The other account to transfer money to
50:          * @param amount
51:          * how much money should be transfered
52:          */
53:         public void transferTo(Account other, int amount){
54:                 if((getGold()-amount)<=0){
55:                         other.addGold(getGold());
56:                         setGold(0);
57:                 }else{
58:                         removeGold(amount);
59:                         other.addGold(amount);
60:                 }
61:         }
62:
63:         public void addGold(int gold){
64:                 setGold(gold+getGold());
65:         }
66:
67:         public void removeGold(int gold){
68:                 setGold(getGold()-gold);
69:         }
70:
71:         @Override
72:         public String toString() {
73:                 return ownerName + "'s account currently contains: " + gold + " gold";
74:         }
75: }
```

```
 1: package game;
 2:
 3: import desktop_resources.GUI;
 4: import slots.OwnableController;
 5: import slots.TerritoryController;
 6:
 7: public class Board {
 8:         private GameBoard slots = new GameBoard();
 9:         //private List<Player> players = new ArrayList<Player>();
10:         private Player[] players;
11:         //private Player currentPlayer;
12:         private int currentPlayerIndex;
13:
14:         private Prison prison;
15:         private DiceCup dice = new DiceCup(2);
16:
17:         public Board(DiceCup dice)
18:         {
19:                 this.dice = dice;
20:                 currentPlayerIndex = 0;
21:         }
22:         public DiceCup getDice()
23:         {
24:                 return dice;
25:         }
26:
27:         public Player getCurrentPlayer()
28:         {
29:                 return players[currentPlayerIndex];
30:         }
31:         /**
32:          * Advances to the next player.
33:          */
34:         private void swapPlayers()
35:         {
36:
37:                 do
38:                 {
39:                         if(++currentPlayerIndex==players.length)
40:                         {
41:                                 prison.advanceDay();
42:                                 currentPlayerIndex = 0;
43:                         }
44:                 }while(players[currentPlayerIndex]==null);
45:
46:         }
47:         /**
48:          * Returns the number of players left, so we can tell who the winner is, when count is returned as 1.
```

```
49:          */
50:         private int getPlayersLeft()
51:         {
52:                 int count = 0;
53:                 for (Player player : players) {
54:                         if(player!=null)
55:                         {
56:                                 ++count;
57:                         }
58:                 }
59:                 return count;
60:         }
61:
62:         /**
63:          * This operation handles the situation where a player is in prison. First it checks if the player
64:          * has a chancecard that can set him/her free. If he/she has one get the possibility of using the
65:          * card. If not it takes the player to the option of paying 1000 DKK or roll the dice.
66:          * If the option of rolling the way out is chosen, the days left in prison for the player
67:          * is checked. If the days left is bigger than zero, the dice has to be rolled. Here we have an
68:          * automatic dice, that rolls three times automatic. If one of the rolls gives to equal die, the
69:          * player is released, if not null is returned to say, the player is still in prison.
70:          */
71:         private DiceResult tryGetOutOfPrison(Inmate inmate)
72:         {
73:                 if(getCurrentPlayer().hasGetOutOfPrisonCard() && GUI.getUserLeftButtonPressed(
74:                                 Translator.getString("YOUAREINPRISONWITHCARD", getCurrentPlayer().getName()), Translator.ge
tString("YES"), Translator.getString("NO")))
75:                 {
76:                         getCurrentPlayer().setHasGetOutOfPrisonCard(false);
77:                         inmate.release();
78:                 }
79:                 else
80:                 {
81:                         if(GUI.getUserLeftButtonPressed(Translator.getString("YOUAREINPRISON", getCurrentPlayer().getName()
, inmate.getDaysLeft()), Translator.getString("PAY1KKR"), Translator.getString("ROLL")))
82:                         {
83:                                 if(getCurrentPlayer().getAccount().withdraw(1000))
84:                                 {
85:                                         inmate.release();
86:                                 }
87:                                 else
88:                                 {
89:                                         GUI.showMessage(Translator.getString("NOMONEYNOFUNNY"));
90:                                 }
91:                         }
92:                 }
93:                 if(inmate.getDaysLeft()>0)
94:                 {
```

```
                        DiceResult res = null;
                        for(int i = 0; i != 3; i++){
                                res = dice.rollDice();
                                GUI.setDice(res.getDice(0), 3, 7, res.getDice(1), 4,8);
                                try
                                {
                                        Thread.sleep(400);
                                }
                                catch(Exception e)
                                {
                                        System.out.println("Something interrupted the dice roll");
                                }
                                if(res.areDiceEqual())
                                {
                                        inmate.release();
                                        return res;
                                }
                        }
//                      //If you failed to roll two equal dices, you skip your turn.
//                      if(!res.areDiceEqual())
//                      {
//                              return null;
//                      }
                }

        return null;

        }

        /**
         * This operation is made to make a list of the pawned properties a player owns. Of course, if a
         * player has not pawned any properties, the GUI will show a message that says, that it is not
         * possible to get a list of pawned properties. If the player has a pawned property, the GUI
         * will make it possible for the player to choose the property to unpawn or cancel the operation.
         * If anything but cancel is chosen, the selected property is moved back to the array of the players
         * fields, so it functions as a "normal" field again.
         */
        private OwnableController getPawnedPropertySelection(Player owner)
        {
                String[] selections = owner.getProperty().getPawnedPropertyList();
                if(selections.length<1)
                {
                        GUI.showMessage(Translator.getString("CANNOTUNPAWN"));
                }
                else
                {
                        String fieldResponse = GUI.getUserSelection(Translator.getString("MAKESELECTION"), appendCancelOpti
on(selections));
```

```
  142:                         if(!fieldResponse.equals(Translator.getString("CANCEL")))
  143:                         {
  144:                                 System.out.println(fieldResponse);
  145:                                 OwnableController selectedField = getCurrentPlayer().getProperty().findOwnableByName(fieldR
esponse);
  146:
  147:                                 return selectedField;
  148:                         }
  149:
  150:                 }
  151:                 return null;
  152:         }
  153:
  154:         /**
  155:          * This operation makes it possible for the player to select a property to pawn.
  156:          */
  157:         private OwnableController getUnPawnedPropertySelection(Player owner)
  158:         {
  159:                 String[] selections = owner.getProperty().getPawnablePropertyList();
  160:                 if(selections.length<1)
  161:                 {
  162:                         GUI.showMessage(Translator.getString("CANNOTUNPAWN"));
  163:                 }
  164:                 else
  165:                 {
  166:                         String fieldResponse = GUI.getUserSelection(Translator.getString("MAKESELECTION"), appendCancelOpti
on(selections));
  167:                         if(!fieldResponse.equals(Translator.getString("CANCEL")))
  168:                         {
  169:                                 System.out.println(fieldResponse);
  170:                                 OwnableController selectedField = owner.getProperty().findOwnableByName(fieldResponse);
  171:                                 return selectedField;
  172:                         }
  173:
  174:                 }
  175:                 return null;
  176:         }
  177:         private void advanceGame()
  178:         {
  179:                 while(getPlayersLeft() > 1) {
  180:                         DiceResult res = null;
  181:                         int rollsLeft = 3;
  182:                         Inmate inmate = prison.getInmate(getCurrentPlayer());
  183:                         if (inmate != null){
  184:                                 res = tryGetOutOfPrison(inmate);
  185:                                 if(inmate.getDaysLeft()==0)
  186:                                 {
  187:                                         GUI.showMessage(Translator.getString("NOWOUTOFPRISON"));
```

```
188:                                    }
189:                            }
190:
191:                            if(inmate==null || inmate.getDaysLeft()==0)
192:                            {
193:                                    String buyHouse = Translator.getString("BUYHOUSE", getCurrentPlayer().getName());
194:                                    String pawnField = Translator.getString("PAWNFIELD", getCurrentPlayer().getName());
195:                                    String releasePawn = Translator.getString("RELEASEFIELD", getCurrentPlayer().getName());
196:                                    String rollTurn;
197:                                            if(res!=null)
198:                                            {
199:                                                    rollTurn = Translator.getString("MOVEOUTOFPRISON");
200:                                            }
201:                                            else
202:                                            {
203:                                                    rollTurn = Translator.getString("ROLLTURN");
204:                                            }
205:                                    String buyAnothersField = Translator.getString("BUYPLAYERPROPERTY");
206:                                    while(true)
207:                                    {
208:                                            String response = GUI.getUserSelection(Translator.getString("ASKUSER", getCurrentPl
ayer().getName()), rollTurn, buyHouse, pawnField, releasePawn,buyAnothersField);
209:
210:
211:                                            if(rollTurn.equals(response))
212:                                            {
213:                                                    //In case the player already rolled the dice to get out of prison
214:                                                    if(res==null)
215:                                                            res = dice.rollDice();
216:                                                    break;
217:                                            }
218:                                            else if(buyHouse.equals(response))
219:                                            {
220:                                                    String[] selections = getCurrentPlayer().getProperty().getTerritoryNames();
221:                                                    String fieldResponse = GUI.getUserSelection(Translator.getString("UNPAWNFIE
LD"),  appendCancelOption(selections));
222:                                                    if(!fieldResponse.equals(Translator.getString("CANCEL")))
223:                                                    {
224:                                                            TerritoryController selectedField = getCurrentPlayer().getProperty(
).findTerritoryByName(fieldResponse);
225:                                                            if(getCurrentPlayer().getProperty().ownsEntireGroup(selectedField.g
etFieldGroup())){
226:                                                            selectedField.buyHouse(getCurrentPlayer());
227:                                                            }
228:                                                            else{
229:                                                                    GUI.showMessage(Translator.getString("YOUDONTOWNGROUP"));
230:                                                            }
231:                                            }
```

```
232:                                    }
233:                                    else if(pawnField.equals(response))
234:                                    {
235:                                            OwnableController slot = getUnPawnedPropertySelection(getCurrentPlayer());
236:                                            if(slot!=null)
237:                                                    pawnField(slot);
238:
239:                                    }
240:                                    else if(releasePawn.equals(response))
241:                                    {
242:                                            OwnableController slot = getPawnedPropertySelection(getCurrentPlayer());
243:                                            if(slot!=null)
244:                                                    releaseField(slot);
245:
246:                                    }
247:                                    else if(buyAnothersField.equals(response))
248:                                    {
249:                                            String[] playerNames = new String[getPlayersLeft()-1];
250:                                            int index = 0;
251:                                            for (Player player : players) {
252:                                                    if(player!=getCurrentPlayer() && player!=null)
253:                                                    {
254:                                                            playerNames[index++] = player.getName();
255:                                                    }
256:                                            }
257:                                            String playerSelections = GUI.getUserSelection(Translator.getString("WHOOWN
SPROPERTY"), appendCancelOption(playerNames));
258:                                            if(!playerSelections.equals(Translator.getString("CANCEL")))
259:                                            {
260:                                                    Player selectedPlayer = getPlayerByName(playerSelections);
261:                                                    //Cannot buy a pawned field, so we are getting those which are able
 to be pawned(ie. not pawned already)
262:                                                            OwnableController selectedField = getUnPawnedProper
tySelection(selectedPlayer);
263:                                                            if(selectedField!=null)
264:                                                                    buyPlayerField(selectedField);
265:                                            }
266:
267:                                    }
268:
269:                            }
270:
271:                    }
272:                    //2nd check is necessary is send to prison during his turn
273:                    while(rollsLeft>0 && (prison.getInmate(getCurrentPlayer())==null || prison.getInmate(getCurrentPlay
er()).getDaysLeft()==0))
274:                    {
275:                            //Since the player has already rolled when selecting to move, we decrease this here
```

```
276:                                --rollsLeft;
277:                                GUI.setDice(res.getDice(0), 3, 7, res.getDice(1), 4,8);
278:                                if(rollsLeft==0 && res.areDiceEqual())
279:                                {
280:                                        GUI.showMessage(Translator.getString("TOOMANYDOUBLES"));
281:                                        prison.addInmate(getCurrentPlayer());
282:                                        getCurrentPlayer().setNextPosition(10, false);
283:                                        updateCurrentPlayerPosition();
284:                                        continue;
285:                                }
286:
287:
288:
289:                                getCurrentPlayer().move(res.getSum(), true);
290:                                while(getCurrentPlayer().getNextPosition()!=getCurrentPlayer().getPosition())
291:                                {
292:                                        updateCurrentPlayerPosition();
293:                                }
294:
295:
296:                                if (getCurrentPlayer().getAccount().getGold() <= 0) {
297:                                        getCurrentPlayer().getProperty().resetPlayerProperties();
298:                                                GUI.showMessage(Translator.getString("LOSINGPLAYER", getCurrentPlayer().get
Name()));
299:                                                GUI.removeAllCars(getCurrentPlayer().getName());
300:                                                players[currentPlayerIndex] = null;
301:                                                break;
302:                                }
303:                                if(res.areDiceEqual())
304:                                {
305:                                        GUI.showMessage(Translator.getString("EXTRATURN"));
306:                                        res = dice.rollDice();
307:                                }
308:                                else
309:                                {
310:                                        break;
311:                                }
312:
313:
314:                        } //what do?
315:                        swapPlayers();
316:                }
317:        }
318:
319:        /**
320:         * This operation makes it possible to get a player by the name.
321:         */
322:        private Player getPlayerByName(String name)
```

```
  323:         {
  324:                 for (Player player : players) {
  325:                         if(player!=null && player.getName().equals(name))
  326:                                 return player;
  327:                 }
  328:                 return null;
  329:         }
  330:
  331:         /**
  332:          *
  333:          */
  334:         private String[] appendCancelOption(String[] source)
  335:         {
  336:                 String[] extendedSelections = new String[source.length+1];
  337:                 //This could be implemented by an array loop as well
  338:                 System.arraycopy(source, 0, extendedSelections, 0, source.length);
  339:                 extendedSelections[extendedSelections.length-1] = Translator.getString("CANCEL");
  340:                 return extendedSelections;
  341:         }
  342:
  343:         /**
  344:          * This handles the situation where a player wants to buy a field from another player. It is the
  345:          * correspondence between the owner and the possible buyer. It works so the owner of the field
  346:          * comes with an offer, and the buyer (currentplayer) can choose to accept or not.
  347:          */
  348:         private void buyPlayerField(OwnableController selectedField) {
  349:                 while(true)
  350:                 {
  351:                         int cost = GUI.getUserInteger(Translator.getString("PLAYERFIELDCOST", selectedField.getOwner().getN
ame()));
  352:                         if(GUI.getUserLeftButtonPressed(Translator.getString("PLAYERFIELDACCEPT", getCurrentPlayer().getNam
e(), selectedField.getName()), Translator.getString("YES"), Translator.getString("NO")))
  353:                         {
  354:                                 if(getCurrentPlayer().getAccount().withdraw(cost))
  355:                                 {
  356:                                         selectedField.getOwner().getAccount().addGold(cost);
  357:                                         selectedField.removeOwner();
  358:                                         selectedField.setOwner(getCurrentPlayer());
  359:                                         GUI.showMessage(Translator.getString("BOUGHTFIELD", getCurrentPlayer().getName(), c
ost));
  360:                                         return;
  361:                                 }
  362:                                 else
  363:                                 {
  364:                                         GUI.showMessage(Translator.getString("NOMONEYNOFUNNY"));
  365:                                 }
  366:
  367:                         }
```

```
368:                        if(!GUI.getUserLeftButtonPressed(Translator.getString("PLAYERNEWOFFER", selectedField.getOwner().ge
tName()), Translator.getString("YES"), Translator.getString("NO")))
369:                        {
370:                                break;
371:                        }
372:                }
373:
374:        }
375:
376:        /**
377:         * Everytime a player rolls a dice, the position has to be updated. First the car needs to be
378:         * moved from the board, then the players position is set, and at last, the car is put on the
379:         * new field.
380:         */
381:        public void updateCurrentPlayerPosition()
382:        {
383:                GUI.removeAllCars(getCurrentPlayer().getName());
384:
385:                getCurrentPlayer().moveToNextPosition();
386:                System.out.println(getCurrentPlayer().getName());
387:                GUI.setCar(getCurrentPlayer().getPosition()+1, getCurrentPlayer().getName());
388:                slots.getField(getCurrentPlayer().getPosition()).landOnField(getCurrentPlayer());
389:        }
390:        //Pawns a field, if the field aren't pawned already, and add the pawn gold to the owner
391:        public void pawnField(OwnableController data){
392:                if(!data.pawned()){
393:
394:                        if(GUI.getUserLeftButtonPressed(Translator.getString("TOPAWN", data.getPawnValue()),
395:                                        Translator.getString("YES"),
396:                                        Translator.getString("NO")))
397:                        {
398:                        data.getOwner().getAccount().addGold(data.getPawnValue());
399:                        data.setPawned(true);
400:                        }
401:
402:                }
403:                else{
404:                        GUI.showMessage(Translator.getString("CANNOTPAWN"));
405:                }
406:        }
407:
408:        /*Releases a field from it's pawn,
409:        but only if the field are pawned
410:        and the owner have enough gold to pay the pawn gold back*/
411:        public void releaseField(OwnableController data){
412:                if(data.pawned()){
413:                        if(GUI.getUserLeftButtonPressed(Translator.getString("TOUNPAWN", data.getPawnValue()),
414:                                        Translator.getString("YES"),
```

```
415:                                        Translator.getString("NO")))
416:                        {
417:
418:                        if(data.getOwner().getAccount().withdraw(data.getPawnValue()))
419:                        {
420:                                data.setPawned(false);
421:                        }
422:                        else
423:                        {
424:                                GUI.showMessage(Translator.getString("NOMONEYNOFUNNY"));
425:                        }
426:
427:                }
428:                else{
429:                        GUI.showMessage(Translator.getString("CANNOTUNPAWN"));
430:                }
431:                }
432: }
433:
434:         public void startGame(){
435:                 System.out.println("Starting game..");
436:                 prison = new Prison(6);
437:                 PlayerCreator playerFactory = new PlayerCreator();
438:                 Player[] chanceCardPlayers = new Player[6];
439:                 slots.initializeBoard(prison,  chanceCardPlayers);
440:                 players = playerFactory.setupPlayers();
441:                 //Workaround for players needed before the board is created, but player names can only be gotten after.
442:                 for(int i=0;i<players.length;++i)
443:                 {
444:                         chanceCardPlayers[i] = players[i];
445:                 }
446:
447:                 advanceGame();
448:
449:                 GUI.showMessage(Translator.getString("WINNINGPLAYERNAME", getCurrentPlayer().getName(), getCurrentPlayer().
getProperty().getPropertyWorth()));
450:                 GUI.close();
451:         }
452:         public static void main(String[] args) {
453:                 Board board = new Board(new DiceCup(2));
454:                 board.startGame();
455:         }
456:         @Override
457:         public String toString() {
458:                 return "Board []";
459:         }
460:
461:
```

```
462: }
```

```
 1: package game;
 2:
 3: import java.util.ArrayList;
 4: import java.util.List;
 5:
 6: import org.w3c.dom.Document;
 7: import org.w3c.dom.Element;
 8: import org.w3c.dom.Node;
 9: import org.w3c.dom.NodeList;
10:
11: import Chancecards.ChanceCardBuildingTaxController;
12: import Chancecards.ChanceCardBuildingTaxData;
13: import Chancecards.ChanceCardCashBonusController;
14: import Chancecards.ChanceCardCashData;
15: import Chancecards.ChanceCardCashTransferController;
16: import Chancecards.ChanceCardController;
17: import Chancecards.ChanceCardData;
18: import Chancecards.ChanceCardGoToPrisonController;
19: import Chancecards.ChanceCardGoToPrisonData;
20: import Chancecards.ChanceCardMatadorLegatController;
21: import Chancecards.ChanceCardMovePlayerController;
22: import Chancecards.ChanceCardMovePlayerData;
23: import Chancecards.ChanceCardMovePlayerRelativeController;
24: import Chancecards.ChanceCardMovePlayerRelativeData;
25: import Chancecards.ChanceCardMoveToNextFleetController;
26: import Chancecards.ChanceCardMoveToNextFleetData;
27: import Chancecards.ChanceCardOutOfPrisonController;
28: import Chancecards.ChanceCardTaxController;
29:
30: public class ChanceCardLoader extends XMLParser{
31:
32:         private static ChanceCardBuildingTaxData parseBuildingTax(Element e) throws Exception
33:         {
34:                 System.out.println("Parsing building tax chancecard...");
35:                 try
36:                 {
37:                         Node translateNode = getUnique(e, "translateID");
38:                         Node houseTaxNode = getUnique(e, "prhouse");
39:                         Node hotelTaxNode = getUnique(e, "prhotel");
40:                         int translateID = parseInteger(translateNode);
41:                         int houseTax = parseInteger(houseTaxNode);
42:                         int hotelTax = parseInteger(hotelTaxNode);
43:                         ChanceCardBuildingTaxData newData = new ChanceCardBuildingTaxData(translateID, houseTax, hotelTax);
44:                         return newData;
45:                 }
46:                 catch(Exception exc)
47:                 {
48:                         throw new Exception("Failed to parse ChancecardBuildingTax", exc);
```

```
49:                     }
50:             }
51:
52:             private static ChanceCardCashData parseCash(Element e) throws Exception
53:             {
54:                     System.out.println("Parsing cash chancecard...");
55:                     try
56:                     {
57:                             Node translateNode = getUnique(e, "translateID");
58:                             Node cashNode = getUnique(e, "cash");
59:                             int translateID = parseInteger(translateNode);
60:                             int cash = parseInteger(cashNode);
61:                             ChanceCardCashData newData = new ChanceCardCashData(translateID, cash);
62:                             return newData;
63:                     }
64:                     catch(Exception exc)
65:                     {
66:                             throw new Exception("Failed to parse cash chancecard", exc);
67:                     }
68:             }
69:
70:             private static ChanceCardMovePlayerData parseMovePlayer(Element e) throws Exception
71:             {
72:                     System.out.println("Parsing move player chancecard...");
73:                     try
74:                     {
75:                             Node translateNode = getUnique(e, "translateID");
76:                             Node fieldnumberNode = getUnique(e, "fieldnumber");
77:                             Node cashatstartNode = getUnique(e, "cashatstart");
78:                             int translateID = parseInteger(translateNode);
79:                             int fieldnumber = parseInteger(fieldnumberNode);
80:                             int cashatstart = parseInteger(cashatstartNode);
81:                             ChanceCardMovePlayerData newData = new ChanceCardMovePlayerData(translateID, fieldnumber, cashatsta
rt==1);
82:                             return newData;
83:                     }
84:                     catch(Exception exc)
85:                     {
86:                             throw new Exception("Failed to parse move player chancecard", exc);
87:                     }
88:             }
89:
90:             private static ChanceCardMovePlayerRelativeData parseMovePlayerRelative(Element e) throws Exception
91:             {
92:                     System.out.println("Parsing move player relative chancecard...");
93:                     try
94:                     {
95:                             Node translateNode = getUnique(e, "translateID");
```

```
   96:                         Node fieldsNode = getUnique(e, "fields");
   97:                         Node cashatstartNode = getUnique(e, "cashatstart");
   98:                         int translateID = parseInteger(translateNode);
   99:                         int fields = parseInteger(fieldsNode);
  100:                         int cashatstart = parseInteger(cashatstartNode);
  101:                         ChanceCardMovePlayerRelativeData newData = new ChanceCardMovePlayerRelativeData(translateID, fields
, cashatstart==1);
  102:                         return newData;
  103:                 }
  104:                 catch(Exception exc)
  105:                 {
  106:                         throw new Exception("Failed to parse move player relative chancecard", exc);
  107:                 }
  108:         }
  109:
  110:         private static ChanceCardMoveToNextFleetData parseMoveToNextFleet(Element e) throws Exception
  111:         {
  112:                 System.out.println("Parsing move to next fleet chancecard...");
  113:                 try
  114:                 {
  115:                         Node translateNode = getUnique(e, "translateID");
  116:                         Node fleetPosition1Node = getUnique(e, "fleetPosition1");
  117:                         Node fleetPosition2Node = getUnique(e, "fleetPosition2");
  118:                         Node fleetPosition3Node = getUnique(e, "fleetPosition3");
  119:                         Node fleetPosition4Node = getUnique(e, "fleetPosition4");
  120:                         Node cashatstartNode = getUnique(e, "cashatstart");
  121:                         int translateID = parseInteger(translateNode);
  122:                         int fleetPosition1 = parseInteger(fleetPosition1Node);
  123:                         int fleetPosition2 = parseInteger(fleetPosition2Node);
  124:                         int fleetPosition3 = parseInteger(fleetPosition3Node);
  125:                         int fleetPosition4 = parseInteger(fleetPosition4Node);
  126:                         int cashatstart = parseInteger(cashatstartNode);
  127:                         ChanceCardMoveToNextFleetData newData = new ChanceCardMoveToNextFleetData(translateID, new int[] {f
leetPosition1, fleetPosition2, fleetPosition3, fleetPosition4}, cashatstart==1);
  128:                         return newData;
  129:                 }
  130:                 catch(Exception exc)
  131:                 {
  132:                         throw new Exception("Failed to parse move to next fleet chancecard", exc);
  133:                 }
  134:         }
  135:
  136:         private static ChanceCardData parseChanceCard(Element e) throws Exception
  137:         {
  138:                 System.out.println("Parsing empty chancecard...");
  139:                 try
  140:                 {
  141:                         Node translateNode = getUnique(e, "translateID");
```

```
142:                         int translateID = parseInteger(translateNode);
143:                         ChanceCardData newData = new ChanceCardData(translateID);
144:                         return newData;
145:                 }
146:                 catch(Exception exc)
147:                 {
148:                         throw new Exception("Failed to parse empty chancecard", exc);
149:                 }
150:         }
151:
152:         private static ChanceCardGoToPrisonData parseGoToPrison(Element e) throws Exception
153:         {
154:                 System.out.println("Parsing go to prison chancecard...");
155:                 try
156:                 {
157:                         Node translateNode = getUnique(e, "translateID");
158:                         Node fieldnumberNode = getUnique(e, "fieldnumber");
159:                         int translateID = parseInteger(translateNode);
160:                         int fieldnumber = parseInteger(fieldnumberNode);
161:                         ChanceCardGoToPrisonData newData = new ChanceCardGoToPrisonData(translateID, fieldnumber);
162:                         return newData;
163:                 }
164:                 catch(Exception exc)
165:                 {
166:                         throw new Exception("Failed to parse go to prison chancecard", exc);
167:                 }
168:         }
169:
170:         static public ChanceCardController[] parseChanceCards(String path, Account parkinglotAcc, Prison prison, Player[] p
layers)
171:         {
172:
173:
174:                         try
175:                         {
176:
177:                                 Document cards = getXMLDocument(path);
178:
179:                                 /***
180:                                  * Parses over the chancecards in the XML document, seperated by types.
181:                                  */
182:                                 NodeList cardNodes = cards.getElementsByTagName("card");
183:                                 List<ChanceCardController> cardList = new ArrayList<ChanceCardController>();
184:                                 for(int index=0;index < cardNodes.getLength();++index)
185:                                 {
186:                                         Node node = cardNodes.item(index);
187:                                         //Saveguard to check if the node actually is an element and not a comment, etc.
188:                                         if(node.getNodeType()==Node.ELEMENT_NODE)
```

```
  189:                                                    {
  190:                                                    ChanceCardController newController = null;
  191:                                                    Element element = (Element)node;
  192:                                                    switch(element.getAttribute("type"))
  193:                                                    {
  194:                                                            case "bonus":
  195:                                                            {
  196:                                                                    ChanceCardCashData newCard = parseCash(element);
  197:                                                                    newController = new ChanceCardCashBonusController(newCard);
  198:                                                                    break;
  199:                                                            }
  200:                                                            case "tax":
  201:                                                            {
  202:                                                                    ChanceCardCashData newCard = parseCash(element);
  203:                                                                    newController = new ChanceCardTaxController(newCard, parkin
glotAcc);
  204:                                                                    break;
  205:                                                            }
  206:                                                            case "cashtransfer":
  207:                                                            {
  208:                                                                    ChanceCardCashData newCard = parseCash(element);
  209:                                                                    newController = new ChanceCardCashTransferController(newCar
d, players);
  210:                                                                    break;
  211:                                                            }
  212:                                                            case "matadorlegat":
  213:                                                            {
  214:                                                                    ChanceCardCashData newCard = parseCash(element);
  215:                                                                    newController = new ChanceCardMatadorLegatController(newCar
d);
  216:                                                                    break;
  217:                                                            }
  218:                                                            case "moveplayer":
  219:                                                            {
  220:                                                                    ChanceCardMovePlayerData newCard = parseMovePlayer(element)
;
  221:                                                                    newController = new ChanceCardMovePlayerController(newCard)
;
  222:                                                                    break;
  223:                                                            }
  224:                                                            case "moveplayerrelative":
  225:                                                            {
  226:                                                                    ChanceCardMovePlayerRelativeData newCard =parseMovePlayerRe
lative(element);
  227:                                                                    newController = new ChanceCardMovePlayerRelativeController(
newCard);
  228:                                                                    break;
  229:                                                            }
```

```
  230:
  231:
  232:
(element);
  233:
Card);
  234:
  235:
  236:
  237:
  238:
  239:
);
  240:
  241:
  242:
  243:
  244:
);
  245:
, parkinglotAcc);
  246:
  247:
  248:
  249:
  250:
  251:
 prison);
  252:
  253:
  254:
  255:
  256:
"type") + " detected!");
  257:
  258:
  259:
  260:
  261:
  262:
  263:
  264:
  265:
  266:
  267:
  268:
  269:
  270:
```

```
        case "movetonextfleet":
        {
                ChanceCardMoveToNextFleetData newCard =parseMoveToNextFleet

                newController = new ChanceCardMoveToNextFleetController(new

                break;
        }
        case "outofprison":
        {
                ChanceCardData newCard =parseChanceCard(element);
                newController = new ChanceCardOutOfPrisonController(newCard

                break;
        }
        case "buildingtax":
        {
                ChanceCardBuildingTaxData newCard =parseBuildingTax(element

                newController = new ChanceCardBuildingTaxController(newCard

                break;
        }
        case "gotoprison":
        {
                ChanceCardGoToPrisonData newCard =parseGoToPrison(element);
                newController = new ChanceCardGoToPrisonController(newCard,

                break;
        }
        default:
        {
                System.out.println("Unknown type: " + element.getAttribute(

        }

}
if(newController!=null)
{
        int amount = 1;
        if(element.hasAttribute("amount"))
        {
                String strAmount = element.getAttribute("amount");
                amount = Integer.parseInt(strAmount);
        }

        while(amount-->0)
        {
```

```
271:                                                                    cardList.add(newController);
272:                                                            }
273:                                                    }
274:                                            }
275:                                    }
276:                                    ChanceCardController[] retCards = new ChanceCardController[cardList.size()];
277:                                    retCards = cardList.toArray(retCards);
278:                                    return retCards;
279:                    }
280:                    catch(Exception e)
281:                    {
282:                            e.printStackTrace();
283:                            return null;
284:                    }
285:
286:            }
287:
288: }
```

```
    1: package game;
    2:
    3: import java.util.Arrays;
    4: import java.util.Random;
    5:
    6: public class DiceCup {
    7:
    8:         private int[] diceArray;
    9:         private Random rGen = new Random(System.currentTimeMillis());
   10:
   11:         public DiceCup(int numberOfDice){
   12:
   13:                 diceArray = new int[numberOfDice];
   14:
   15:         }
   16:                 public DiceResult rollDice(){
   17:
   18: /*
   19:  * Inserts Ints into the array, depending on the amount of dice chosen
   20:  *      for the game. The array extends depending on the amount of dice
   21:  *      chosen. And spits out a random generated number for each array slot.
   22:  */
   23:
   24:
   25:                         for(int i=0; i < diceArray.length; i++){
   26:                                 diceArray[i] = rGen.nextInt(6)+1;
   27:                         }
   28:                         return new DiceResult(diceArray);
   29:                 }
   30:                 @Override
   31:                 public String toString() {
   32:                         return "DiceCup [diceArray=" + Arrays.toString(diceArray) + ", random Generator=" + rGen + ", rollD
ice()=" + rollDice()
   33:                                         + "]";
   34:                 }
   35:
   36: }
```

```
    1: package game;
    2:
    3: import java.util.Arrays;
    4:
    5: /*
    6:  * A compact OO-way of storing the eyes of the two dices. Is mainly a storage class, but also has some utility functions.
    7:  */
    8:
    9: public class DiceResult
   10: {
   11:         private int[] dice;
   12:
   13:         public DiceResult(int[] result){
   14:                 dice = result;
   15:         }
   16:         /**
   17:          * @param n
   18:          * a die
   19:          * @return
   20:          * instance we wanted from the die
   21:          */
   22:         public int getDice(int n){
   23:
   24:                 try
   25:                 {
   26:                         return dice[n];
   27:                 }
   28:                 catch(IndexOutOfBoundsException e)
   29:                 {
   30:                         throw new IndexOutOfBoundsException("Could not reach the correct element(dice array):\n" + e.getMes
sage());
   31:                 }
   32:
   33:
   34:         }
   35:         public int getSum(){
   36:
   37:                 int sumOfDice = 0;
   38:
   39:                 for(int i=0; i < dice.length; i++){
   40:                         sumOfDice += dice[i];
   41:
   42:                 }
   43:                 /**
   44:                  * Decides amount of dice depended on the length of the array
   45:                  */
   46:                 return sumOfDice;
   47:         }
```

```
48:
49:         public int getDiceAmount(){
50:                 return dice.length;
51:         }
52:
53:         /*
54:          * Checks if the dice are two of a kind, in order to incoorperate rule of extra roles
55:          * in case of two of a kind.
56:          */
57:
58:         public boolean areDiceEqual(){
59:                 if(getDiceAmount() < 2)
60:                         return true;
61:                 else{
62:                 for(int i = 1; i < getDiceAmount(); i++){
63:                         if(dice[i] != dice[i-1])
64:                                 return false;
65:                         }
66:                 }
67:                 return true;
68:         }
69:
70:         /*
71:          * Checks if rolls are the same, in order to incoorperate rule, that if this happens
72:          * 3 times in a row, player is thrown in prison.
73:          */
74:
75:         public boolean areRollsEqual(DiceResult res){
76:                 for(int i = 0; i < getDiceAmount(); i++){
77:                         if(dice[i] != res.getDice(i))
78:                                 return false;
79:                         }
80:                 return true;
81:         }
82:
83:         @Override
84:         public String toString() {
85:                 return "DiceResult [dice=" + Arrays.toString(dice) + ", getSum()=" + getSum() + ", getDiceAmount()="
86:                                 + getDiceAmount() +  "]";
87:         }
88: }
89:
```

```
    1: package game;
    2: import slots.*;
    3: import utilities.ShuffleBag;
    4: import java.util.ArrayList;
    5: import java.util.List;
    6: import org.w3c.dom.Document;
    7: import org.w3c.dom.Element;
    8: import org.w3c.dom.Node;
    9: import org.w3c.dom.NodeList;
   10:
   11: import Chancecards.ChanceCardController;
   12:
   13:
   14: public class FieldLoader extends XMLParser {
   15:
   16:         private static TerritoryData parseTerritory(Element e) throws Exception
   17:         {
   18:                 System.out.println("Parsing territory...");
   19:                 try
   20:                 {
   21:                         int[] rentPrices = new int[6];
   22:
   23:                         Node translateNode = getUnique(e, "translateID");
   24:                         Node groupNode = getUnique(e, "groupID");
   25:                         Node priceNode = getUnique(e, "price");
   26:                         Node pawnvalueNode = getUnique(e, "pawnvalue");
   27:                         Node housepriceNode = getUnique(e, "houseprice");
   28:                         for (int i = 0; i < 6; i++) {
   29:                                 Node houserentNode = getUnique(e, "houserent"+i);
   30:                                 rentPrices[i] = parseInteger(houserentNode);
   31:                         }
   32:                         int translateID = parseInteger(translateNode);
   33:                         int groupID = parseInteger(groupNode);
   34:                         int price = parseInteger(priceNode);
   35:                         int pawnvalue = parseInteger(pawnvalueNode);
   36:                         int houseprice = parseInteger(housepriceNode);
   37:                         TerritoryData newData = new TerritoryData(translateID, groupID, price, houseprice, pawnvalue, rentP
rices);
   38:                         return newData;
   39:                 }
   40:                 catch(Exception exc)
   41:                 {
   42:                         throw new Exception("Failed to parse Territory", exc);
   43:                 }
   44:
   45:         }
   46:
   47:         private static FieldData parseEmptyField(Element e) throws Exception
```

```
48:         {
49:                 System.out.println("Parsing empty field...");
50:                 try
51:                 {
52:                         Node translateNode = getUnique(e, "translateID");
53:                         int translateID = parseInteger(translateNode);
54:                         FieldData newData = new FieldData(translateID);
55:                         return newData;
56:                 }
57:                 catch(Exception exc)
58:                 {
59:                         throw new Exception("Failed to parse EmptyField", exc);
60:                 }
61:
62:         }
63:
64:         private static ParkinglotData parseParkinglot(Element e, Account parkingAcc) throws Exception
65:         {
66:                 System.out.println("Parsing parkinglot...");
67:                 try {
68:
69:                         Node translateNode = getUnique(e, "translateID");
70:                         int translateID = parseInteger(translateNode);
71:                         ParkinglotData newData = new ParkinglotData(translateID, parkingAcc);
72:                         return newData;
73:
74:                 } catch (Exception exc) {
75:
76:                         throw new Exception("Failed to parse Refuge", exc);
77:
78:                 }
79:
80:         }
81:         private static BreweryData parseBrewery(Element e) throws Exception
82:         {
83:                 System.out.println("Parsing laborCamp...");
84:                 try {
85:                         Node translateNode = getUnique(e, "translateID");
86:                         Node rentNode = getUnique(e, "rent");
87:                         Node priceNode = getUnique(e, "price");
88:                         Node pawnvalueNode = getUnique(e, "pawnvalue");
89:                         int translateID = parseInteger(translateNode);
90:                         int rent = parseInteger(rentNode);
91:                         int price = parseInteger(priceNode);
92:                         int pawnvalue = parseInteger(pawnvalueNode);
93:                         BreweryData newData = new BreweryData(rent, translateID, price, pawnvalue);
94:                         return newData;
95:
```

```
96:                 } catch (Exception exc) {
97:
98:                         throw new Exception("Failed to parse LaborCamp", exc);
99:
100:                }
101:        }
102:        private static TaxData parseTax(Element e) throws Exception
103:        {
104:                System.out.println("Parsing tax...");
105:                try {
106:                        Node translateNode = getUnique(e, "translateID");
107:                        Node taxNode = getUnique(e, "tax");
108:                        Node taxPercentageNode = getUnique(e, "taxPercentage");
109:                        int translateID = parseInteger(translateNode);
110:                        int tax = parseInteger(taxNode);
111:                        int taxPercentage = parseInteger(taxPercentageNode);
112:                        TaxData newData = new TaxData(translateID, tax, taxPercentage);
113:                        return newData;
114:
115:                } catch (Exception exc) {
116:
117:                        throw new Exception("Failed to parse Tax", exc);
118:
119:                }
120:        }
121:        private static FleetData parseFleet(Element e) throws Exception
122:        {
123:                System.out.println("Parsing fleet...");
124:                try {
125:                        Node translateNode = getUnique(e, "translateID");
126:                        Node priceNode = getUnique(e, "price");
127:                        Node pawnvalueNode = getUnique(e, "pawnvalue");
128:                        int translateID = parseInteger(translateNode);
129:                        int price = parseInteger(priceNode);
130:                        int pawnvalue = parseInteger(pawnvalueNode);
131:                        FleetData newData = new FleetData(translateID, price, pawnvalue);
132:                        return newData;
133:
134:                } catch (Exception exc) {
135:
136:                        throw new Exception("Failed to parse Tax", exc);
137:
138:                }
139:        }
140:        private static GoToPrisonData parseGoToPrison(Element e, Prison p) throws Exception {
141:                System.out.println("Parsing gotoPrison...");
142:                try {
143:
```

```
144:                        Node translateNode = getUnique(e, "translateID");
145:                        Node prisonNode = getUnique(e, "prisonPosition");
146:                        int translateID = parseInteger(translateNode);
147:                        int prisonLocation = parseInteger(prisonNode);
148:                        GoToPrisonData newData = new GoToPrisonData(translateID, prisonLocation, p);
149:                        return newData;
150:
151:                } catch (Exception exc) {
152:
153:                        throw new Exception("Failed to parse gotoPrison", exc);
154:
155:                }
156:        }
157:        static public FieldController[] parseFields(String path, ShuffleBag<ChanceCardController> chanceCards, Prison priso
n, Account parkinglotAccount, int[] buildingRent)
158:        {
159:
160:
161:                        try
162:                        {
163:
164:                                Document fields = getXMLDocument(path);
165:
166:                                /***
167:                                 * Parses over the fields in the XML document, seperated by types.
168:                                 */
169:                                NodeList fieldNodes = fields.getElementsByTagName("field");
170:                                List<FieldController> fieldList = new ArrayList<FieldController>();
171:                                for(int index=0;index < fieldNodes.getLength();++index)
172:                                {
173:                                        Node node = fieldNodes.item(index);
174:                                        //Saveguard to check if the node actually is an element and not a comment, etc.
175:                                        if(node.getNodeType()==Node.ELEMENT_NODE)
176:                                        {
177:                                                FieldController newController = null;
178:                                                Element element = (Element)node;
179:                                                switch(element.getAttribute("type"))
180:                                                {
181:                                                        case "territory":
182:                                                        {
183:                                                                TerritoryData newField = parseTerritory(element);
184:                                                                newController = new TerritoryController(newField);
185:                                                                break;
186:                                                        }
187:                                                        case "empty":
188:                                                        {
189:                                                                FieldData newField = parseEmptyField(element);
190:                                                                newController = new EmptyFieldController(newField);
```

```
  191:                                                                  break;
  192:                                                          }
  193:                                                          case "brewery":
  194:                                                          {
  195:                                                                  BreweryData newField = parseBrewery(element);
  196:                                                                  newController = new BreweryController(newField);
  197:                                                                  break;
  198:                                                          }
  199:                                                          case "tax":
  200:                                                          {
  201:                                                                  TaxData newField = parseTax(element);
  202:                                                                  newController = new TaxController(newField, parkinglotAccou
nt);
  203:                                                                  break;
  204:                                                          }
  205:                                                          case "fleet":
  206:                                                          {
  207:                                                                  FleetData newField = parseFleet(element);
  208:                                                                  newController = new FleetController(newField);
  209:                                                                  break;
  210:                                                          }
  211:                                                          case "chancefield":
  212:                                                          {
  213:                                                                  FieldData newField = new FieldData(3);
  214:                                                                  newController = new ChanceFieldController(chanceCards, newF
ield);
  215:                                                                  break;
  216:                                                          }
  217:                                                          case "gotoprison":
  218:                                                          {
  219:                                                                  GoToPrisonData newField = parseGoToPrison(element, prison);
  220:                                                                  newController = new GoToPrisonController(newField);
  221:                                                                  break;
  222:                                                          }
  223:                                                          case "parkinglot":
  224:                                                          {
  225:                                                                  ParkinglotData newField = parseParkinglot(element, parkingl
otAccount);
  226:                                                                  newController = new ParkinglotController(newField);
  227:                                                                  break;
  228:                                                          }
  229:                                                          default:
  230:                                                          {
  231:                                                                  System.out.println("Unknown type: " + element.getAttribute(
"type") + " detected!");
  232:                                                          }
  233:                                                  }
  234:                                                  if(newController!=null)
```

```
235:                                                    {
236:                                                            fieldList.add(newController);
237:                                                    }
238:                                            }
239:                                    }
240:                                    FieldController[] retFields = new FieldController[fieldList.size()];
241:                                    retFields = fieldList.toArray(retFields);
242:                                    return retFields;
243:                    }
244:                    catch(Exception e)
245:                    {
246:                            e.printStackTrace();
247:                            return null;
248:                    }
249:
250:            }
251:
252: }
```

```
    1: package game;
    2:
    3: import slots.*;
    4: import utilities.ShuffleBag;
    5: import desktop_resources.GUI;
    6: import Chancecards.*;
    7:
    8: public class GameBoard {
    9:
   10:         private FieldController[] fields;
   11:         public void initializeBoard(Prison prison, Player[] players) //This method builds up the game board when the game s
tarts
   12:         {
   13:                 //desktop_fields.Brewery b = new desktop_fields.Brewery.Builder().setRent("2000").build();
   14:
   15:                 System.out.println("Loading board...");
   16:                 Account acc = new Account(0, null);
   17:                 int[] houseRent = new int[6];
   18:                 ChanceCardController[] chanceCards = ChanceCardLoader.parseChanceCards("ChanceCard.xml", acc, prison, playe
rs); //loads all the chancecards into an array
   19:                 fields = FieldLoader.parseFields("Fields.xml", new ShuffleBag<ChanceCardController>(chanceCards), prison,ac
c, houseRent); //loads all the fields into an array
   20:                 desktop_fields.Field[] guiFields = new desktop_fields.Field[fields.length];
   21:
   22:                 for (FieldController field : fields) { //Gets the name of each field on the board
   23:                         System.out.println(field.getName());
   24:                 }
   25:                 System.out.println("Loaded: " + fields.length + "fields..");
   26:
   27:                 int pos = 1;
   28:                 for(FieldController f : fields) //takes all the fields and places them on a position on the board
   29:                 {
   30:                         desktop_fields.Field guiField = f.pushToGUI(pos);
   31:                         if(guiField==null)
   32:                         {
   33:                                 System.out.println("[WARNING]" + f.getName() + " returned null!"); //Means that for whateve
r reason the loader was not able to load the field
   34:                         }
   35:                         guiFields[pos-1] = guiField;
   36:                         pos++;
   37:                 }
   38:                 GUI.create(guiFields);
   39:         }
   40:         public int getFieldCount()
   41:         {
   42:                 return fields.length;
   43:         }
   44:         public FieldController getField(int index) //Get a specific field from the array of fields. Exception: If the argum
```

```
ent is longer than the array, give error message
    {
            try
            {
                    return fields[index];
            }
            catch(IndexOutOfBoundsException e)
            {
                    throw new IndexOutOfBoundsException("Attempt to access a non-existing field");
            }
    }
}
```

```
 1: package game;
 2:
 3: public class Inmate {
 4:         private int days;
 5:         private Player player;
 6:
 7:         public Inmate(int d, Player p)
 8:         {
 9:                 days = d;
10:                 player = p;
11:         }
12:         public void release()
13:         {
14:                 days = 0;
15:         }
16:         public int getDaysLeft()
17:         {
18:                 return days;
19:         }
20:
21:         public int setDaysLeft(int daysLeft)
22:         {
23:                 return daysLeft;
24:         }
25:
26:         public void decreaseDaysLeft()
27:         {
28:                 days--;
29:         }
30:
31:         public boolean isPlayer(Player p)
32:         {
33:                 return p==player;
34:         }
35: }
```

```
1: package game;
2:
3:
4: public class Player {
5:         /**
6:          * Holds all information about the player, including a reference to the account.
7:          */
8:         private String name;
9:         private int position = 0;
10:        private int nextPosition = 0;
11:        private boolean cashAtStart = true;
12:        private boolean getOutOfPrisonCard = false;
13:        private Account account;
14:        private Property propertyOwned = new Property();
15:
16:        public int getNextPosition() {
17:                return nextPosition;
18:        }
19:
20:        public void setNextPosition(int nextPosition, boolean passStart) {
21:                this.nextPosition = nextPosition;
22:                cashAtStart = passStart;
23:        }
24:        public void moveToNextPosition()
25:        {
26:                int distance = nextPosition-position;
27:                if(distance < 0)
28:                {
29:                        moveToPosition(distance+40);
30:                }
31:                else
32:                {
33:                        moveToPosition(distance);
34:                }
35:                //Fixes an error which accours when the nextPosition has been sat past start
36:                nextPosition = position;
37:
38:        }
39:
40:
41:        /**
42:         * Each player has their own set of dice which keeps track of their rolls.
43:         */
44:
45:        public Player(String s)
46:        {
47:                name = s;
48:                account = new Account(30000, name);
```

```
49:         }
50:         public String getName()
51:         {
52:                 return name;
53:         }
54:         public Account getAccount()
55:         {
56:                 return account;
57:         }
58:         public Property getProperty()
59:         {
60:                 return propertyOwned;
61:         }
62:
63:         private void moveToPosition (int afstand){
64:                 final int ANTALSLOTS = 40;
65:                 final int STARTBONUS = 4000;
66:                 position += afstand;
67:                 //add the moved distance to the old position.
68:
69:                 /**
70:                  * Decide wether or not the new position exeeds the board.
71:                  * If it does, it take the amount of fields from the position
72:                  * to find the new position.
73:                  */
74:                 if(position >= ANTALSLOTS){ //since we are 0-index, 40 is the 0th field
75:                         position -= ANTALSLOTS;
76:                         if(cashAtStart)
77:                                 account.addGold(STARTBONUS);
78:                 }
79:         }
80:
81:         public void move(int afstand, boolean cashAtStart) {
82:                 nextPosition += afstand;
83:                 this.cashAtStart = cashAtStart;
84:         }
85:
86:         public int getPosition(){
87:                 return position;
88:         }
89:
90:         public boolean hasGetOutOfPrisonCard() {
91:                 return getOutOfPrisonCard;
92:         }
93:
94:         public void setHasGetOutOfPrisonCard(boolean b) {
95:                 getOutOfPrisonCard = b;
96:         }
```

```
   97:
   98:         @Override
   99:         public String toString() {
  100:                 return "Player [, getName()=" + getName() + ", getAccount()=" + getAccount() + ", getProperty().getProperty
Count()="
  101:                                 + getProperty().getPropertyCount() + ", getPosition()=" + getPosition() + "]";
  102:         }
  103: }
  104:
```

```
    1: package game;
    2:
    3: import java.awt.Color;
    4: import java.util.regex.Pattern;
    5:
    6: import desktop_codebehind.Car;
    7: import desktop_resources.GUI;
    8: import utilities.ShuffleBag;
    9:
   10: public class PlayerCreator {
   11:         private final int PLAYERSTARTINGCASH = 30000;
   12:         private ShuffleBag<Color> availableCarColors = new ShuffleBag<Color>(new Color[]{Color.BLUE, Color.YELLOW, new Colo
r(0, 107f/255, 15f/255), Color.PINK, Color.RED, Color.MAGENTA});
   13:         private Player[] players;
   14:
   15:
   16:         public Player createPlayer(String name)
   17:         {
   18:                 Player newPlayer = new Player(name);
   19:                 Color color = Color.white;
   20:                 try {
   21:                         color = availableCarColors.getNext();
   22:                 } catch (Exception e) {
   23:                         e.printStackTrace();
   24:                 }
   25:                 /*
   26:                  * Giving players a random colored car, as well as setting up the player on the GUI
   27:                  * with starting cash, name and their newly given car.
   28:                  */
   29:                 Car car;
   30:                 int result = (int)((Math.random())*3+1);
   31:                 if(result == 1)
   32:                 {
   33:                         car = new Car.Builder().primaryColor(color).secondaryColor(Color.black).patternZebra().build();
   34:
   35:                 }
   36:                 else if(result == 2)
   37:                 {
   38:                         car = new Car.Builder().primaryColor(color).secondaryColor(Color.black).patternDotted().build();
   39:                 }
   40:                 else
   41:                 {
   42:                         car = new Car.Builder().primaryColor(color).secondaryColor(Color.black).patternCheckered().build();
   43:                 }
   44:                 System.out.println(car + " " + name);
   45:                 GUI.addPlayer(name, PLAYERSTARTINGCASH, car);
   46:
   47:                 return newPlayer;
```

```
48:         }
49:         private boolean verifyName(String s)
50:         {
51:                 if(s.isEmpty())
52:                 {
53:                         return false;
54:                 }
55:
56:                         //Checks if the string contains a whitespace character
57:                         Pattern pattern = Pattern.compile("\\s");
58:                         java.util.regex.Matcher m = pattern.matcher(s);
59:
60:                         return (!m.find());
61:
62:         }
63:         private Player setupPlayer(String user){
64:                 if (!verifyName(user) || user.length() > 15)
65:                 {
66:                         return null;
67:                 }
68:                 for(Player i : players) {
69:                         if (i != null && i.getName().equals(user)){
70:                                 return null;
71:                         }
72:                 }
73:
74:                 return createPlayer(user);
75:         }
76:
77:         public Player[] setupPlayers() {
78:                 int amount = GUI.getUserInteger(Translator.getString("NUMBEROFPLAYERS"));
79:                 final int PLAYERAMOUNTMIN = 2;
80:                 final int PLAYERAMOUNTMAX = 6;
81:                 while(amount < PLAYERAMOUNTMIN || amount > PLAYERAMOUNTMAX){
82:                         GUI.showMessage(Translator.getString("NUMBEROFPLAYERSERROR",PLAYERAMOUNTMIN,PLAYERAMOUNTMAX));
83:
84:                         amount = GUI.getUserInteger(Translator.getString("NUMBEROFPLAYERS"));
85:                 }
86:                 /*
87:                  * adds each player to the players[] array. Used to keep track of running players in the game
88:                  * and whose turn it is. Also checks if each player are fulfilling the conditions, for making a name.
89:                  */
90:                 players = new Player[amount];
91:                 for(int j = 0; j < amount; j++) {
92:                         String user;
93:                         if(j == 0)
94:                         {
95:                                 user = GUI.getUserString(Translator.getString("ENTERNAME1"));
```

```
                        }
                        else
                        {
                                user = GUI.getUserString(Translator.getString("ENTERNAME2"));
                        }
                Player newPlayer = null;
                while((newPlayer = setupPlayer(user)) == null) {
                        user = GUI.getUserString(Translator.getString("NAMEERROR"));
                }
                players[j] = newPlayer;
                }
                desktop_board.Board.getInstance().updatePlayers();
                return players;

        }

}
```

```
1: package game;
2:
3: public class Prison {
4:
5:         private Inmate[] inmates;
6:
7:         public final int SERVINGDAYS = 4;
8:
9:         public Prison(int maxInmates){
10:                inmates = new Inmate [maxInmates];
11:        }
12:
13:        public Inmate getInmate(Player player)
14:        {
15:                for (Inmate inmate : inmates)
16:                {
17:                        if(inmate!=null && inmate.isPlayer(player))
18:                        {
19:                                return inmate;
20:                        }
21:                }
22:                return null;
23:        }
24:
25:        public void addInmate(Player player){
26:                for (int i = 0; i < inmates.length; i++)
27:                {
28:                        if(inmates[i] == null)
29:                        {
30:                                Inmate newInmate = new Inmate(SERVINGDAYS, player);
31:                                inmates[i] = newInmate;
32:                                break;
33:                        }
34:                }
35:        }
36:        public void advanceDay() {
37:
38:                for (int i=0;i<inmates.length;++i)
39:                {
40:                        if(inmates[i] != null)
41:                        {
42:                                inmates[i].decreaseDaysLeft();
43:
44:                                if(inmates[i].getDaysLeft()<=0)
45:                                {
46:                                        inmates[i] = null;
47:                                }
48:                        }
```

```
49:                 }
50:
51:
52:         }
53:
54: }
55:
```

```java
package game;

import slots.BreweryController;
import slots.FleetController;
import slots.OwnableController;
import slots.OwnableController.FIELDGROUPS;
import slots.TerritoryController;

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

public class Property {
/**
 * Keeps track of how many fleets and breweries each player has.
 * 'Expand' adds an additional fleet/breweries, when the plays buys on of them
 */

	//private List<slots.OwnableController> properties = new ArrayList<slots.OwnableController>();
	private List<slots.FleetController> fleets = new ArrayList<slots.FleetController>();
	private List<slots.BreweryController> breweries = new ArrayList<slots.BreweryController>();
	private List<slots.TerritoryController> territories = new ArrayList<slots.TerritoryController>();

	public OwnableController findOwnableByName(String name)
	{
		OwnableController[] ownables = getPropertiesOwned();
		for (OwnableController ownableController : ownables) {
			String ownableName = ownableController.getName();
			if(ownableName.equals(name))
			{
				return ownableController;
			}
		}
		return null;

	}
	public void resetPlayerProperties()
	{
		for(TerritoryController territory : territories)
		{
			territory.removeHouses();
			territory.reset();
		}
		for (FleetController fleet : fleets) {
			fleet.reset();
		}
		for (BreweryController brewery : breweries) {
			brewery.reset();
```

```
49:                 }
50:         }
51:         public TerritoryController findTerritoryByName(String name)
52:         {
53:                 for(TerritoryController territory : territories)
54:                 {
55:                         if(territory.getName().equals(name))
56:                         {
57:                                 return territory;
58:                         }
59:                 }
60:                 return null;
61:         }
62:         public String[] getPawnedPropertyList()
63:         {
64:                 OwnableController[] ownables = getPropertiesOwned();
65:                 String[] names = new String[ownables.length];
66:                 int index = 0;
67:                 for (OwnableController cont : ownables) {
68:                         if(cont.pawned())
69:                         {
70:                                 names[index++] = cont.getName();
71:                         }
72:                 }
73:                 String[] retNames= new String[index];//no need to +1 on index, since its always one larger than the current
 array contains
74:                 System.arraycopy(names, 0, retNames, 0, index);
75:                 return retNames;
76:         }
77:         public String[] getPawnablePropertyList()
78:         {
79:                 OwnableController[] ownables = getPropertiesOwned();
80:                 String[] names = new String[ownables.length];
81:                 int index = 0;
82:                 for (OwnableController cont : ownables) {
83:                         if(!cont.pawned())
84:                         {
85:                                 names[index++] = cont.getName();
86:                         }
87:                 }
88:                 String[] retNames= new String[index];
89:                 System.arraycopy(names, 0, retNames, 0, index);
90:                 return retNames;
91:         }
92:
93:         public Iterator<slots.TerritoryController> getTerritories()
94:         {
95:                 return territories.iterator();
```

```
    96:         }
    97:         public slots.OwnableController[] getPropertiesOwned()
    98:         {
    99:                 slots.OwnableController[] collection = new slots.OwnableController[fleets.size()+breweries.size()+territori
es.size()];
   100:                 int collectionIndex = 0;
   101:                 for (int i = 0; i < fleets.size(); i++) {
   102:                         collection[collectionIndex++] =  fleets.get(i);
   103:                 }
   104:                 for(int i = 0; i < breweries.size();++i)
   105:                 {
   106:                         collection[collectionIndex++] = breweries.get(i);
   107:                 }
   108:                 for(int i=0;i<territories.size();++i)
   109:                 {
   110:                         collection[collectionIndex++] = territories.get(i);
   111:                 }
   112:                 return collection;
   113:         }
   114:         public void addTerritory(slots.TerritoryController p)
   115:         {
   116:                 territories.add(p);
   117:         }
   118:         public void removeTerritory(slots.TerritoryController p)
   119:         {
   120:                 int pos = territories.indexOf(p);
   121:                 if(pos!=-1)
   122:                 {
   123:                         territories.remove(pos);
   124:                 }
   125:                 else
   126:                 {
   127:                         System.out.println("Attempt to remove non-existent territory");
   128:                 }
   129:
   130:         }
   131:         public void addFleet(slots.FleetController t)
   132:         {
   133:                 fleets.add(t);
   134:         }
   135:         public void removeFleet(slots.FleetController fleet)
   136:         {
   137:                 int pos = territories.indexOf(fleet);
   138:                 if(pos!=-1)
   139:                 {
   140:                         territories.remove(pos);
   141:                 }
   142:                 else
```

```
143:                 {
144:                         System.out.println("Attempt to remove non-existent fleet");
145:                 }
146:         }
147:         public void addBreweries(slots.BreweryController b)
148:         {
149:                 breweries.add(b);
150:         }
151:         public void removeBreweries(slots.BreweryController b)
152:         {
153:                 int pos = territories.indexOf(b);
154:                 if(pos!=-1)
155:                 {
156:                         territories.remove(pos);
157:                 }
158:                 else
159:                 {
160:                         System.out.println("Attempt to remove non-existent brewery");
161:                 }
162:         }
163:         public int getPropertyCount()
164:         {
165:                 return fleets.size() + territories.size() + breweries.size();
166:         }
167:         public int getTotalHotelCount()
168:         {
169:                 int amount = 0;
170:                 for(TerritoryController territory: territories)
171:                 {
172:                         amount += territory.getHotelAmount();
173:                 }
174:                 return amount;
175:         }
176:         public String[] getTerritoryNames()
177:         {
178:
179:                 String[] propertyNames = new String[territories.size()];
180:                 for(int i=0;i<territories.size();++i)
181:                 {
182:                         propertyNames[i] = territories.get(i).getName();
183:                 }
184:                 return propertyNames;
185:         }
186:         public int getTotalHouseCount()
187:         {
188:                 int amount = 0;
189:                 for(TerritoryController territory: territories)
190:                 {
```

```
191:                         amount += territory.getHouseAmount();
192:                 }
193:                 return amount;
194:         }
195:         public int getBreweriesOwned()
196:         {
197:                 int count = 0;
198:                 for (BreweryController breweryController : breweries) {
199:                         if(!breweryController.pawned())
200:                         {
201:                                 ++count;
202:                         }
203:                 }
204:                 return count;
205:         }
206:         public int getFleetOwned()
207:         {
208:                 int count = 0;
209:                 for (FleetController fleetController : fleets) {
210:                         if(!fleetController.pawned())
211:                         {
212:                                 ++count;
213:                         }
214:                 }
215:                 return count;
216:         }
217:
218:         public int getPropertyWorth()
219:         {
220:                 slots.OwnableController[] properties = getPropertiesOwned();
221:                 int propertyWorth = 0;
222:                 for (OwnableController property : properties)
223:                 {
224:                         propertyWorth += property.getWorth();
225:                 }
226:                 return propertyWorth;
227:         }
228:
229:         public boolean ownsEntireGroup(FIELDGROUPS Id){
230:                         int groupCount = 0;
231:                 for (TerritoryController ownableController : territories) {
232:                         FIELDGROUPS ownableId = ownableController.getFieldGroup();
233:
234:                         if(ownableId == Id){
235:                         groupCount++;
236:                         }
237:                 }
238:
```

```
239:                 if(FIELDGROUPS.BLUE == Id || FIELDGROUPS.PURPLE == Id){
240:                         return groupCount==2;
241:                 }
242:
243:                 else{
244:                         return groupCount==3;
245:                 }
246:         }
247: }
```

```java
package game;

public class TestDice extends DiceCup {
	private int[][] diceSequence;
	//Has to be -1 since it gets increased by one before its used for the first time
	private int currentIndex = -1;

	public TestDice(int[][] diceSequence)
	{
		super(2);
		this.diceSequence = diceSequence;
	}
	@Override
	public DiceResult rollDice(){
		if(currentIndex+1==diceSequence.length)
		{
			currentIndex = 0;
		}
		else
		{
			++currentIndex;
		}
		return new DiceResult(diceSequence[currentIndex]);
	}
}
```

```
    1: package game;
    2:
    3: import java.util.Locale;
    4: import java.util.ResourceBundle;
    5:
    6: public class Translator {
    7:         private static ResourceBundle strings;
    8:         public static String getString(String keyword, Object... args)
    9:         {
   10:
   11:                 //If not previous set, use default locale(da, DK)
   12:                 if(strings==null)
   13:                 {
   14:                         setLocale(new Locale("dk", "DA"));
   15:                 }
   16:                 if(args!=null)
   17:                 {
   18:                         return String.format(strings.getString(keyword), args);
   19:                 }
   20:                 else
   21:                 {
   22:                         return strings.getString(keyword);
   23:                 }
   24:         }
   25:         @Override
   26:         public String toString() {
   27:                 return "Current locale: " + strings.getLocale() + " on the following file: " + strings.getBaseBundleName()
+ " where " + strings.keySet().size() + "keys are contained";
   28:         }
   29:         public static void setLocale(Locale l)
   30:         {
   31:                 strings = ResourceBundle.getBundle("MessageBundle", l);
   32:         }
   33:         //Avoids creating objects of this class
   34:         private Translator()
   35:         {
   36:
   37:         }
   38: }
```

```
    1: package game;
    2:
    3: import java.io.File;
    4: import java.io.IOException;
    5:
    6: import javax.swing.JOptionPane;
    7: import javax.xml.parsers.DocumentBuilder;
    8: import javax.xml.parsers.DocumentBuilderFactory;
    9:
   10: import org.w3c.dom.Document;
   11: import org.w3c.dom.Element;
   12: import org.w3c.dom.Node;
   13: import org.w3c.dom.NodeList;
   14:
   15: public abstract class XMLParser {
   16:
   17:
   18:         protected static Document getXMLDocument(String path)
   19:         {
   20:                 Document fields = null;
   21:                 File fieldFile = null;
   22:                 try{
   23:                         //Does not need to be closed, as it just represents a path to the file.
   24:                         //the actual read/writing is done by the XMLparser.
   25:                          fieldFile = new File(path);
   26:                         System.out.println(fieldFile.getAbsolutePath());
   27:                         //No need to store the DocumentBuilderFactory instance as we are using default settings:
   28:                         DocumentBuilder XMLparser = DocumentBuilderFactory.newInstance().newDocumentBuilder();
   29:                          fields = XMLparser.parse(fieldFile);
   30:                         return fields;
   31:                 }
   32:                 catch(IOException fileEx)
   33:                 {
   34:                         JOptionPane.showMessageDialog(desktop_board.Board.getInstance().getComponent(0),
   35:                                         "File not found at: " + fieldFile.getAbsolutePath() + "\nPlease restore "
   36:                                                         + "the file or the board cannot be created.", "Critical error accou
red",
   37:                                                         JOptionPane.ERROR_MESSAGE);
   38:                 }
   39:                 catch(Exception e)
   40:                 {
   41:                         e.printStackTrace();
   42:                 }
   43:
   44:                 return null;
   45:         }
   46:         /**
   47:          * Checks if multiple occurrences exists within the Element node and prints to the console if true
```

```
48:          * @param The element containing the element
49:          * @param The name of the element to get
50:          * @return The first occurrence of the element
51:          * @throws An exception is thrown if no elements were found.
52:          */
53:         protected static Node getUnique(Element e, String elementNameTag) throws Exception
54:         {
55:                 NodeList element = e.getElementsByTagName(elementNameTag);
56:                 if(element.getLength()>1)
57:                 {
58:                         System.out.println("Warning! " + e.getNodeName() + " had multiple of " + elementNameTag + ". Going
with the first found...");
59:                 }
60:                 else if(element.getLength()<1)
61:                 {
62:                         throw new Exception("Failed to locate " + elementNameTag + " for " + e.getNodeName());
63:                 }
64:                 else if(element.item(0).getTextContent().length()==0)
65:                 {
66:                         throw new Exception("Failed to load " + elementNameTag + " because it was left empty!");
67:                 }
68:                 return element.item(0);
69:         }
70:         protected static int parseInteger(Node n) throws Exception
71:         {
72:                 String content = n.getTextContent();
73:                 System.out.println("Got: "+ content+ " from: " + n.getNodeName());
74:                 try
75:                 {
76:                         int translateId = Integer.parseInt(content);
77:                         return translateId;
78:                 }
79:                 catch(NumberFormatException exc)
80:                 {
81:                         throw new Exception("ERROR: Failed to parse: " + content + " integer from " + n.getNodeName(), exc)
;
82:                 }
83:         }
84:
85: }
```

```java
package slots;

import java.awt.Color;


import desktop_resources.GUI;
import slots.OwnableController;
import game.DiceCup;
import game.DiceResult;
import game.Player;
import game.Translator;

public class BreweryController extends OwnableController{
        private BreweryData breweryData;

        public BreweryController(BreweryData data) {
                super((OwnableData)data);
                breweryData = data;
        }

        public void chargeRent(Player player)
        {

                GUI.getUserButtonPressed(Translator.getString("BREWERY"), Translator.getString("ROLL"));
                DiceCup dice = new DiceCup(2);
                DiceResult res = dice.rollDice();
                int price = res.getSum() * getRent();
                GUI.setDice(res.getDice(0), 3, 7, res.getDice(1), 4,8);
                GUI.showMessage(Translator.getString("BREWERYCONCLUSION", res.getSum(), price));
                player.getAccount().transferTo(breweryData.getOwner().getAccount(), price);
        }
        public desktop_fields.Field pushToGUI(int position){

                breweryData.setPosition(position);
                guiField = new desktop_fields.Brewery.Builder().setRent(Translator.getString("BREWERYRENT",/*HARDCODED VARI
ABLES IS NEVER GOOD! TODO: REMOVE*/ 100)).setBgColor(new Color(255f/255, 165f/255, 48f/255)).build();
                guiField.setDescription(getDescription());
                guiField.setTitle(breweryData.getName());

                guiField.setSubText(Integer.toString(breweryData.getPrice()));

                return guiField;
        }

        @Override
        public int getWorth() {
                return breweryData.getPrice();
        }
```

```
48:         @Override
49:         public String getDescription() {
50:
51:                 return Translator.getString("BREWERYDSC");
52:         }
53:
54:         @Override
55:         public FIELDGROUPS getFieldGroup() {
56:                 return FIELDGROUPS.BREWERY;
57:         }
58:
59:         @Override
60:         public int getRent() {
61:                 return 100*breweryData.getOwner().getProperty().getBreweriesOwned();
62:         }
63:
64:         @Override
65:         protected void registerOwner() {
66:                 breweryData.getOwner().getProperty().addBreweries(this);
67:         }
68:
69:         @Override
70:         protected void UnRegisterOwner() {
71:                 breweryData.getOwner().getProperty().removeBreweries(this);
72:
73:         }
74:
75:         public String toString(){
76:                 return "getWorth()=" + getWorth() + ", getDescription()=" + getDescription() + ", getFieldGroup()=" + getFi
eldGroup() + ", getRent()=" + getRent() + ", BreweryData()" + breweryData.toString();
77:         }
78: }
```

```
 1: package slots;
 2:
 3: public class BreweryData extends OwnableData{
 4:
 5:         private int baserent;
 6:
 7:         public BreweryData(int rent, int translateID, int price, int pawnvalue) {
 8:                 super(translateID, price, pawnvalue);
 9:                 this.baserent = rent;
10:         }
11:
12:         public String toString(){
13:                 return "baserent=" + baserent;
14:         }
15: }
```

```java
package slots;

import java.awt.Color;

import Chancecards.ChanceCardController;
import desktop_fields.Field;
import desktop_resources.GUI;
import game.Player;
import game.Translator;
import utilities.ShuffleBag;

public class ChanceFieldController extends FieldController {
	private ShuffleBag<ChanceCardController> cards;


	public ChanceFieldController(ShuffleBag<ChanceCardController> cards, FieldData data) {
		super(data);
		this.cards = cards;
	}

	@Override
	public void landOnField(Player player) {
		try{
			//If all cards has been used, reset the pile
			if(cards.getElementsLeft()==0)
			{
				cards.reset();
			}
			ChanceCardController newCard = cards.getNext();
			//If onDrawn returns false, then the card should be put back into the pile

			GUI.displayChanceCard(newCard.getDescription());
			GUI.showMessage(Translator.getString("CHANCECARDDRAWN"));
			if(newCard.onDrawn(player))
			{
				GUI.showMessage(Translator.getString("CARDCOULDNOTBEUSED"));
				cards.pushBackLastElement();
			}

		}
		catch(Exception e)
		{
			e.printStackTrace();
		}

	}

	@Override
```

```
49:         public Field pushToGUI(int position) {
50:                 desktop_fields.Chance newField = new desktop_fields.Chance.Builder().setBgColor(Color.gray).build();
51:                 newField.setTitle(getName());
52:                 return newField;
53:         }
54:         @Override
55:         public String getDescription() {
56:
57:                 return Translator.getString("CHANCEFIELDDSC");
58:         }
59:
60:         public String toString(){
61:                 return "getDescription=" + getDescription();
62:         }
63: }
```

```java
package slots;


import java.awt.Color;
import desktop_fields.Field;
import desktop_resources.GUI;
import game.Player;
import game.Translator;

public class EmptyFieldController extends FieldController{

        private FieldData emptyFieldData;

        public EmptyFieldController(FieldData fieldData) {
                super(fieldData);
                emptyFieldData = fieldData;
        }
//Field is supposed to do nothing, so there for it is empty.
        @Override
        public void landOnField(Player player) {
                GUI.showMessage(getDescription());
        }

        @Override
        public String toString() {
                return "EmptyFieldController";
        }

        @Override
        public Field pushToGUI(int position) {
                desktop_fields.Street field = new desktop_fields.Street.Builder().setBgColor(new Color(255f/255, 165f/255,
48f/255)).build();
                field.setDescription(getDescription());
                field.setTitle(this.getName());
                field.setSubText("");

                return field;
        }
        @Override
        public String getDescription() {
                return Translator.getString("EMPTYFIELDDSC" + emptyFieldData .getTranslateID());
        }
}
```

```
    1: package slots;
    2:
    3: import game.*;
    4:
    5: public abstract class FieldController {
    6:
    7:         private FieldData data;
    8:
    9:         public FieldController(FieldData d)
   10:         {
   11:                 data = d;
   12:         }
   13:
   14:         public abstract void landOnField (Player player);
   15:
   16:         /**
   17:          * Adds the field to the GUI. Should be called before the GUI is created
   18:          * @param position
   19:          */
   20:         public abstract desktop_fields.Field pushToGUI(int position);
   21:         public abstract String getDescription();
   22:         public String getName()
   23:         {
   24:                 return data.getName();
   25:         }
   26:
   27:         public String toString(){
   28:                 return data.toString() + ", getName()" + getName();
   29:         }
   30: }
   31:
```

```
    1: package slots;
    2:
    3: import game.Translator;
    4:
    5: public class FieldData {
    6:         private int translateID;
    7:         protected int position;
    8:
    9:         public FieldData(int translateID){
   10:                 this.translateID = translateID;
   11:         }
   12:
   13:         public String getName() {
   14:                 return Translator.getString("SLOT" + translateID);
   15:         }
   16:
   17:         public void setPosition(int p)
   18:         {
   19:                 position = p;
   20:         }
   21:         public int getPosition()
   22:         {
   23:                 return position;
   24:         }
   25:
   26:         public int getTranslateID() {
   27:                 return translateID;
   28:         }
   29:
   30:         public String toString(){
   31:                 return "getName()=" + getName() + "getPosition()=" + getPosition() + ", getTranslateID()" + getTranslateID(
);
   32:         }
   33: }
```

```
    1: package slots;
    2:
    3: import java.awt.Color;
    4:
    5: import desktop_resources.GUI;
    6: import game.Player;
    7: import game.Translator;
    8:
    9: public class FleetController extends OwnableController{
   10:         private FleetData fleetData;
   11:         private final int RENT[] = {500, 1000, 2000, 4000};
   12:         public FleetController(FleetData data)
   13:         {
   14:                 super((OwnableData)data);
   15:                 fleetData = data;
   16:         }
   17:
   18:         @Override
   19:         public desktop_fields.Field pushToGUI(int position) {
   20:                 fleetData.setPosition(position);
   21:                 guiField = new desktop_fields.Shipping.Builder().setRent(String.format("%d, %d, %d, %d", RENT[0], RENT[1],
RENT[2], RENT[3])).setBgColor(new Color(144f/255,211f/255, 212f/255)).build();
   22:                 guiField.setTitle(fleetData.getName());
   23:                 guiField.setDescription(getDescription());
   24:                 guiField.setSubText("" + fleetData.price);
   25:                 return guiField;
   26:         }
   27:
   28:         @Override
   29:         public int getRent()
   30:         {
   31:                 Player owner = fleetData.getOwner();
   32:                 if(owner==null)
   33:                         return RENT[0];
   34:                 return RENT[owner.getProperty().getFleetOwned()-1];
   35:
   36:         }
   37:         @Override
   38:         public int getWorth() {
   39:                 return fleetData.getPrice();
   40:         }
   41:         @Override
   42:         public String getDescription() {
   43:
   44:                 return Translator.getString("FLEETDSC");
   45:         }
   46:         @Override
   47:         public FIELDGROUPS getFieldGroup() {
```

```
48:                 return FIELDGROUPS.FLEET;
49:         }
50:         @Override
51:         protected void chargeRent(Player player) {
52:                 GUI.showMessage(Translator.getString("PAYTHEOWNER", getRent()));
53:                 player.getAccount().transferTo(fleetData.getOwner().getAccount(), getRent());
54:
55:         }
56:         @Override
57:         protected void registerOwner() {
58:                 fleetData.getOwner().getProperty().addFleet(this);
59:         }
60:
61:         @Override
62:         protected void UnRegisterOwner() {
63:                 fleetData.getOwner().getProperty().removeFleet(this);
64:         }
65:
66:         public String toString(){
67:                 return fleetData.toString() +  ", getRent()=" + getRent();
68:         }
69:
70: }
```

```
 1: package slots;
 2:
 3: public class FleetData extends OwnableData{
 4:         private final int[] RENT = {500, 1000, 2000, 4000};
 5:
 6:         public FleetData(int i, int price, int pawnvalue) {
 7:                 super(i, price, pawnvalue);
 8:         }
 9:         public int getRent(int shipAmount)
10:         {
11:                 return RENT[shipAmount];
12:         }
13:
14:         public String toString(){
15:                 return "RENT[]=" + RENT;
16:         }
17: }
```

```java
package slots;

import java.awt.Color;

import desktop_resources.GUI;
import game.Player;
import game.Translator;

public class GoToPrisonController extends FieldController{
	private desktop_fields.Jail goToPrison;
	private GoToPrisonData goToPrisonData;
	public GoToPrisonController(GoToPrisonData data)
	{
		super(data);
		goToPrisonData = data;
	}

	@Override
	public void landOnField(Player player) {
		/**
		 * When player lands on GoToPrison he is immediately sent to prison, and will remain there until he gets a double or 3 turns pass.
		 */
		goToPrison.displayOnCenter();
		GUI.showMessage(Translator.getString("LANDONGOTOPRISON", goToPrisonData.getPrisonPosition()));
		player.setNextPosition(goToPrisonData.getPrisonPosition(),false);
		goToPrisonData.getPrison().addInmate(player);
	}

	@Override
	public desktop_fields.Field pushToGUI(int position){
		position = goToPrisonData.getPrisonPosition();
		goToPrison = new desktop_fields.Jail.Builder().setBgColor(new Color(223f/255, 255f/255, 43f/255)).build();
		goToPrison.setDescription(this.getDescription());
		goToPrison.setTitle(goToPrisonData.getName());
		goToPrison.setSubText(getDescription());
		return goToPrison;
	}
	@Override
	public String getDescription() {

		return Translator.getString("GOTOPRISONDSC");
	}

	public String toString(){
		return goToPrisonData.toString();
	}
}
```

```java
package slots;

import game.Prison;

public class GoToPrisonData extends FieldData{

	private int prisonPosition;

	private Prison prison;

	public GoToPrisonData(int i, int prisonPosition, Prison prison) {
		super(i);
		this.prisonPosition = prisonPosition;
		this.prison = prison;
	}

	public int getPrisonPosition(){
		return prisonPosition;
	}

	public void setPrisonPosition(int a){
		prisonPosition = a;
	}

	public Prison getPrison() {
		return prison;
	}

	public String toString(){
		return "getPrisonPosition()=" + getPrisonPosition() + " getPrison()=" + getPrison();
	}
}
```

```
 1: package slots;
 2:
 3: import game.*;
 4: import desktop_resources.GUI;
 5: import slots.FieldController;
 6:
 7:
 8:
 9: public abstract class OwnableController extends FieldController{
10:         public enum FIELDGROUPS
11:         {
12:                 BLUE,
13:                 PINK,
14:                 GREEN,
15:                 GRAY,
16:                 RED,
17:                 WHITE,
18:                 YELLOW,
19:                 PURPLE,
20:                 FLEET,
21:                 BREWERY
22:         }
23:         private OwnableData ownableData;
24:         private boolean pawned = false;
25:         protected desktop_fields.Ownable guiField;
26:
27:         public OwnableController(OwnableData dat)
28:         {
29:                 super(dat);
30:                 ownableData = dat;
31:         }
32:
33:         public abstract int getRent();
34:         protected abstract void chargeRent(Player player);
35:         protected abstract void registerOwner();
36:         protected abstract void UnRegisterOwner();
37:
38:         @Override
39:         final public void landOnField(Player player)
40:         {
41:                         /**
42:                          * Player lands on brewery.
43:                          * If field is owned, he pays an amount depending on a roll with
44:                          * two dice times the amount of labor camps owned by the owner.
45:                          * If field is not owned, player can choose to buy it.
46:                          */
47:                         guiField.displayOnCenter();
48:                         if(hasOwner()){
```

```
49:                                 if(ownableData.getOwner()!=player)
50:                                 {
51:                                         if(!pawned)
52:                                         {
53:                                                 chargeRent(player);
54:                                         }
55:                                         else
56:                                         {
57:                                                 GUI.showMessage(Translator.getString("PAWNEDFIELD"));
58:                                         }
59:
60:                                 }else{
61:                                         GUI.showMessage(Translator.getString("YOURFIELD"));
62:                                 }
63:                         }else{
64:                                 if(buyField(player))
65:                                 {
66:                                         GUI.showMessage(Translator.getString("BOUGHTFIELD",ownableData.getName(), ownableDa
ta.getPrice()));
67:                                 }
68:                         }
69:         }
70:
71:         public Player getOwner()
72:         {
73:                 return ownableData.getOwner();
74:         }
75:         //Should be used when a player goes to 0 cash
76:         public void reset()
77:         {
78:                 ownableData.removeOwner();
79:                 GUI.removeOwner(ownableData.getPosition());
80:                 setPawned(false);
81:
82:         }
83:         //Should never be used when looping over a player's properties
84:         public void removeOwner()
85:         {
86:                 UnRegisterOwner();
87:                 ownableData.removeOwner();
88:                 GUI.removeOwner(ownableData.getPosition());
89:
90:         }
91:         public void setOwner(Player owner) {
92:                 /**
93:                  * General way to make the buyer of a field the owner.
94:                  */
95:
```

```
   96:                 System.out.println(ownableData.getName() + " now has " + owner.getName() + " as their owner" + " at slot "
+ ownableData.getPosition());
   97:                 ownableData.setOwner(owner);
   98:                 registerOwner();
   99:                 GUI.setOwner(ownableData.getPosition(), owner.getName());
  100:         }
  101:         public boolean hasOwner()
  102:         {
  103:                 return(ownableData.getOwner()!=null);
  104:         }
  105:
  106:         public boolean buyField (Player visitor){
  107:                 /**
  108:                  * General purchase procedure, with a withdrawal of money
  109:                  * and a call to setOwner if the withdraw was completed.
  110:                  */
  111:                 if(GUI.getUserLeftButtonPressed(Translator.getString("BUYFIELD", ownableData.getPrice()), Translator.getStr
ing("YES"), Translator.getString("NO"))){
  112:                         if(visitor.getAccount().withdraw(ownableData.getPrice())){
  113:                                 setOwner(visitor);
  114:                                 return true;
  115:                         }else{
  116:                                 GUI.showMessage(Translator.getString("NOTENOUGHGOLD"));
  117:                         }
  118:                 }
  119:                 else{
  120:                         GUI.showMessage(Translator.getString("ENDTURN"));
  121:                 }
  122:                 return false;
  123:
  124:
  125:         }
  126:         public boolean pawned()
  127:         {
  128:                 return pawned;
  129:         }
  130:
  131:         public void setPawned(boolean pawned)
  132:         {
  133:                 this.pawned = pawned;
  134:         }
  135:
  136:         public int getPawnValue(){
  137:                 return ownableData.getPawnValue();
  138:         }
  139:
  140:         public abstract FIELDGROUPS getFieldGroup();
  141:         public abstract int getWorth();
```

```
142:
143:         @Override
144:         public String toString() {
145:                 return ownableData.toString();
146:         }
147: }
```

```java
package slots;

import game.Player;

public abstract class OwnableData extends FieldData {


	protected int price;
	private Player owner;
	private int pawnvalue;
	public OwnableData(int translateID, int price, int pawnvalue) {
		super(translateID);
		this.price = price;
		this.pawnvalue = pawnvalue;
	}
	public int getPrice()
	{
		return price;
	}
	public void setOwner(Player newOwner)
	{
		owner = newOwner;
	}
	public void removeOwner()
	{
		owner = null;
	}
	public Player getOwner()
	{
		return owner;
	}
	public boolean hasOwner()
	{
		return (owner!=null);
	}

	public int getPawnValue(){
		return pawnvalue;
	}

	@Override
	public String toString() {
		return "OwnableData [price=" + price + ", owner=" + owner + ", pawnvalue=" + pawnvalue + "]";
	}

}
```

```
 1: package slots;
 2:
 3: import java.awt.Color;
 4:
 5: import desktop_resources.GUI;
 6: import game.Player;
 7: import game.Translator;
 8:
 9: public class ParkinglotController extends FieldController{
10:         private desktop_fields.Street parkingLot;
11:         private ParkinglotData parkinglotData;
12:         public ParkinglotController(ParkinglotData data)
13:         {
14:                 super(data);
15:                 parkinglotData = data;
16:         }
17:
18:         @Override
19:         public void landOnField(Player player) {
20:
21:                 /*
22:                  * Player lands on the Parking lot field and is given the bonus.
23:                  * The bonus is at minimum a 1000 but will get all the penality money,
24:                  * which is continously added throughout the game.
25:                  */
26:
27:                 parkingLot.displayOnCenter();
28:                 GUI.showMessage(Translator.getString("LANDONPARKINGLOT", parkinglotData.getAccount().getGold()));
29:                 parkinglotData.TransferBonus(player.getAccount());
30:         }
31:         @Override
32:         public desktop_fields.Field pushToGUI(int position){
33:                 parkinglotData.setPosition(position);
34:                 parkingLot = new desktop_fields.Street.Builder().setBgColor(new Color(223f/255, 255f/255, 43f/255)).build()
;
35:                 parkingLot.setDescription(getDescription());
36:                 parkingLot.setTitle(parkinglotData.getName());
37:                 parkingLot.setSubText(Translator.getString("PARKINGLOTSUB", parkinglotData.getAccount().getGold()));
38:                 return parkingLot;
39:         }
40:         @Override
41:         public String getDescription() {
42:
43:                 return Translator.getString("PARKINGLOTDSC");
44:         }
45:
46:         public String toString(){
47:                 return parkinglotData.toString();
```

```
48:         }
49: }
```

```
 1: package slots;
 2:
 3: import game.Account;
 4:
 5:
 6: public class ParkinglotData extends FieldData{
 7:
 8:
 9:         private Account balance;
10:         public void TransferBonus(Account acc)
11:         {
12:                 balance.transferTo(acc, balance.getGold());
13:         }
14:
15:         public ParkinglotData(int i, Account acc) {
16:                 super(i);
17:                 this.balance = acc;
18:         }
19:
20:         public Account getAccount(){
21:                 return balance;
22:         }
23:
24:         public String toString(){
25:                 return "getAccount()=" + getAccount();
26:         }
27:
28: }
```

```
    1: package slots;
    2:
    3: import java.awt.Color;
    4:
    5: import desktop_resources.GUI;
    6: import game.Account;
    7: import game.Player;
    8: import game.Translator;
    9:
   10: public class TaxController extends FieldController {
   11:         private desktop_fields.Tax tax;
   12:         private TaxData taxData;
   13:         private Account parkinglotAccount;
   14:         public TaxController(TaxData data, Account parkinglotAccount)
   15:         {
   16:                 super(data);
   17:                 taxData = data;
   18:                 this.parkinglotAccount = parkinglotAccount;
   19:         }
   20:         @Override
   21:         public void landOnField(Player player) {
   22:                 /**
   23:                  * Player lands on Tax and has to pay, either a flat amount or
   24:                  * a percentage of his fortune.
   25:                  */
   26:                 int taxPaid = 0;
   27:                 if (taxData.getTaxRate() == 0){
   28:                         GUI.getUserButtonPressed(Translator.getString("LANDONTAX"), Integer.toString(taxData.getTaxAmount()
));
   29:                         taxPaid = taxData.getTaxAmount();
   30:                 }
   31:                 else {
   32:                         tax.displayOnCenter();
   33:                         if (GUI.getUserLeftButtonPressed(Translator.getString("LANDONTAX"), Integer.toString(taxData.getTax
Rate())+"%" , Integer.toString(taxData.getTaxAmount()))) {
   34:                                 taxPaid = (int)(((float)taxData.getTaxRate()/100f)*player.getAccount().getGold());
   35:                         }
   36:                         else {
   37:                                 taxPaid = taxData.getTaxAmount();
   38:                         }
   39:                 }
   40:                 player.getAccount().transferTo(parkinglotAccount, taxPaid);
   41:         }
   42:
   43:         @Override
   44:         public desktop_fields.Field pushToGUI(int position) {
   45:                 taxData.setPosition(position);
   46:                 tax = new desktop_fields.Tax.Builder().setBgColor(new Color(255f/255, 43f/255, 57f/255)).build();
```

```
47:                 tax.setDescription(getDescription());
48:                 tax.setTitle(taxData.getName());
49:                 tax.setSubText(Integer.toString(taxData.getTaxAmount()));
50:                 return tax;
51:         }
52:         @Override
53:         public String getDescription() {
54:
55:                 return Translator.getString("TAXDSC");
56:         }
57:
58:         public String toString(){
59:                 return taxData.toString();
60:         }
61: }
```

```
 1: package slots;
 2:
 3: public class TaxData extends FieldData{
 4:
 5:         private int taxAmount;
 6:         private int taxRate;
 7:
 8:
 9:         public int getTaxAmount() {
10:                 return taxAmount;
11:         }
12:
13:
14:         public int getTaxRate() {
15:                 return taxRate;
16:         }
17:
18:
19:         public TaxData(int i, int price, int taxPercentage) {
20:                 super(i);
21:                 this.taxAmount = price;
22:                 taxRate = taxPercentage;
23:         }
24:
25:         public String toString(){
26:                 return "getTaxAmount()=" + getTaxAmount() + ", getTaxRate" + getTaxRate();
27:         }
28:
29: }
```

```
 1: package slots;
 2:
 3: import java.awt.Color;
 4:
 5: import desktop_resources.GUI;
 6: import game.Player;
 7: import game.Translator;
 8:
 9: public class TerritoryController extends OwnableController {
10:         private TerritoryData territoryData;
11:
12:         public TerritoryController(TerritoryData data)
13:         {
14:                 super(data);
15:                 territoryData = data;
16:         }
17:         public void removeHouses()
18:         {
19:                 territoryData.resetHouses();
20:                 GUI.setHouses(territoryData.getPosition(), 0);
21:                 GUI.setHotel(territoryData.getPosition(), false);
22:         }
23:         public int getUpgradeCosts()
24:         {
25:                 return territoryData.getHouseCost();
26:         }
27:         /*
28:          * If a player owns a territory it will enable him to purchase a house.
29:          */
30:         public void buyHouse(Player player){
31:                 if(territoryData.getOwner() == player){
32:                         if(territoryData.getHouses() < 5){
33:                                 if(player.getAccount().withdraw(getUpgradeCosts())){
34:                                         territoryData.addHouse();
35:                                         int houseCount = territoryData.getHouses();
36:
37:                                         if(houseCount < 5)
38:                                         {
39:                                                 GUI.setHouses(territoryData.getPosition(), territoryData.getHouses());
40:                                         }
41:                                         else
42:                                         {
43:                                                 GUI.setHotel(territoryData.getPosition(), true);
44:                                         }
45:                                         GUI.showMessage(Translator.getString("HOUSECONFIRM"));
46:                                 }
47:                                 else{
48:                                         GUI.showMessage(Translator.getString("YOUCANNOTAFFORDTHAT"));
```

```
49:                                 }
50:                         }
51:                         else{
52:                                 GUI.showMessage(Translator.getString("FULLYUPGRADED"));
53:                         }
54:                 }
55:                 else{
56:                         GUI.showMessage(Translator.getString("YOUARENOTTHEOWNER"));
57:                 }
58:         }
59:
60:
61:         @Override
62:         public desktop_fields.Field pushToGUI(int position) {
63:
64:                 Color[] colors = {Color.blue, Color.orange, Color.green, Color.lightGray, Color.red, Color.white, Color.yel
low, /*dark purple*/ new Color(155, 67, 196)};
65:                 Color thisColor = colors[getFieldGroup().ordinal()];
66:                 territoryData.setPosition(position);
67:
68:                 guiField = new desktop_fields.Street.Builder().setRent(Integer.toString(territoryData.getRent())).setBgColo
r(thisColor).build();
69:                 guiField.setDescription(getDescription());
70:                 guiField.setTitle(territoryData.getName());
71:                 guiField.setSubText(Integer.toString(territoryData.getPrice()));
72:                 return guiField;
73:         }
74:         public int getHouseAmount()
75:         {
76:                 if(territoryData.getHouses()<5)
77:                         return territoryData.getHouses();
78:                 else
79:                         return 0;
80:         }
81:         public int getHotelAmount()
82:         {
83:                 return territoryData.getHouses()>4 ? 1 : 0;
84:         }
85:
86:         @Override
87:         public int getWorth() {
88:                 int territoryWorth = 0;
89:                 return territoryWorth + territoryData.getPrice()+(territoryData.getHouses()*territoryData.getHouseCost());
90:         }
91:         @Override
92:         public String getDescription() {
93:                 if(territoryData.getGroupID()==0 || territoryData.getGroupID()==1) {
94:                         return Translator.getString("SLOTDSC1");
```

```
   95:                 }
   96:                 else if(territoryData.getGroupID()==2 || territoryData.getGroupID()==3) {
   97:                         return Translator.getString("SLOTDSC2");
   98:                 }
   99:                 else if(territoryData.getGroupID()==4 || territoryData.getGroupID()==5) {
  100:                         return Translator.getString("SLOTDSC3");
  101:                 }
  102:                 else {
  103:                         return Translator.getString("SLOTDSC4");
  104:                 }
  105:         }
  106:         @Override
  107:         public FIELDGROUPS getFieldGroup() {
  108:                 return FIELDGROUPS.values()[territoryData.getGroupID()];
  109:         }
  110:         @Override
  111:         public int getRent() {
  112:                 return territoryData.getRent();
  113:         }
  114:
  115:
  116:         @Override
  117:         protected void chargeRent(Player player) {
  118:                 if(getOwner().getProperty().ownsEntireGroup(getFieldGroup()) == true && getHouseAmount()==0){
  119:                         GUI.showMessage(Translator.getString("PAYTHEOWNERDOUBLE", this.getRent()*2));
  120:                         player.getAccount().transferTo(territoryData.getOwner().getAccount(), this.getRent()*2);
  121:                 }
  122:                 else{
  123:                 GUI.showMessage(Translator.getString("PAYTHEOWNER", this.getRent()));
  124:                         player.getAccount().transferTo(territoryData.getOwner().getAccount(), this.getRent());
  125:                 }
  126:
  127:         }
  128:         @Override
  129:         protected void registerOwner() {
  130:                 territoryData.getOwner().getProperty().addTerritory(this);
  131:
  132:         }
  133:         @Override
  134:         protected void UnRegisterOwner() {
  135:                 territoryData.getOwner().getProperty().removeTerritory(this);
  136:
  137:         }
  138:
  139:         public String toString(){
  140:                 return "getUpgradeCosts()=" + getUpgradeCosts() + ", getHouseAmount()=" + getHouseAmount() + ", getHotelAmo
unt()=" + getHotelAmount() + ", getWorth()=" + getWorth() + territoryData.toString() + " , getFieldGroup()=" + getFieldGroup();
  141:         }
```

```
142:
143: }
```

```
    1: package slots;
    2:
    3: public class TerritoryData extends OwnableData{
    4:
    5:         private int houses;
    6:         private int houseCost;
    7:         private int groupID;
    8:         private int[] buildingRent;
    9:
   10:         public TerritoryData(int translateID, int id, int price, int houseCost, int pawnvalue, int[] buildingRent) {
   11:                 super(translateID, price, pawnvalue);
   12:                 this.houseCost = houseCost;
   13:                 houses = 0;
   14:                 groupID = id;
   15:                 this.buildingRent = buildingRent;
   16:         }
   17:
   18:
   19:         public int getRent() {
   20:                 return buildingRent[getHouses()];
   21:         }
   22:
   23:         public int getHouses(){
   24:                 return houses;
   25:         }
   26:         public void resetHouses()
   27:         {
   28:                 houses = 0;
   29:         }
   30:         public void addHouse(){
   31:                 houses++;
   32:         }
   33:         public int getHouseCost(){
   34:                 return houseCost;
   35:         }
   36:
   37:         public int getGroupID(){
   38:                 return groupID;
   39:         }
   40:
   41:         public String toString(){
   42:                 return "getRent()=" + getRent() + ", getHouses()=" + getHouses() + ", getHouseCost()=" + getHouseCost() + "
, getGroupID()=" + getGroupID();
   43:         }
   44:
   45: }
```

```
    1: package test;
    2: import game.*;
    3: import static org.junit.Assert.*;
    4:
    5: import org.junit.Test;
    6:
    7: public class AccountTest {
    8:
    9:         @Test
   10:         public void testGetGold() {
   11:                 Account account = new Account(5000,"Sheep");
   12:                 assertTrue(account.getGold()==5000);
   13:         }
   14:
   15:         @Test
   16:         public void testWithdraw() {
   17:                 Account account = new Account(5000,"Sheep");
   18:                 assertTrue("Fail, you should be able to withdraw gold",account.withdraw(500));
   19:                 assertTrue("Fail, there should be 4500left", account.getGold()==4500);
   20:                 assertFalse("Fail, you should not be able to withdraw more than your total gold",account.withdraw(Integer.M
AX_VALUE));
   21:         }
   22:
   23:         @Test
   24:         public void testTransferTo() {
   25:                 Account account1 = new Account(5000,"Sheep");
   26:                 Account account2 = new Account(5000, "Isbjoern");
   27:                 account1.transferTo(account2, 5000);
   28:                 assertTrue("Fail, it should have transfered 5000gold",account1.getGold()==0 && account2.getGold()==10000);
   29:                 account1.addGold(5000);
   30:                 account1.transferTo(account2, 10000);
   31:                 assertTrue("Fail, you should only be able to receive the amount of gold that the opponent has",account1.get
Gold()==0 && account2.getGold()==15000);
   32:         }
   33:
   34:         @Test
   35:         public void testAddGold() {
   36:                 Account account = new Account(5000,"Sheep");
   37:                 account.addGold(5000);
   38:                 assertTrue("Fail, there should be 10000gold in the account",account.getGold()==10000);
   39:         }
   40:
   41:         @Test
   42:         public void testRemoveGold() {
   43:                 Account account = new Account(5000,"Sheep");
   44:                 account.removeGold(5000);
   45:                 assertTrue("Fail, there should not be any gold in the account",account.getGold()==0);
   46:                 account.removeGold(1);
```

```
47:                 assertFalse("Fail, you can have negativ gold",account.getGold()==-1);
48:         }
49:
50: }
```

```
    1: package test;
    2: import game.*;
    3: import slots.*;
    4: //import slots.FieldController.Types;
    5:
    6: import static org.junit.Assert.*;
    7:
    8: import org.junit.Test;
    9:
   10: public class BreweryControllerTest {
   11:
   12:         @Test
   13:         public void testGetRent() {
   14:                 BreweryData data = new BreweryData(100, 1, 4000, 2000);
   15:                 BreweryController laborCamp = new BreweryController(data);
   16:                 Player player = new Player("Test");
   17:                 laborCamp.pushToGUI(1);
   18:                 laborCamp.buyField(player);
   19:                 assertTrue("Fejl, renten er forkert", laborCamp.hasOwner() == true && laborCamp.getRent() == 100 || laborCa
mp.hasOwner() == false);
   20:         }
   21:
   22:         @Test
   23:         public void testGetWorth() {
   24:                 BreweryData data = new BreweryData(1, 13, 3, 2000);
   25:                 BreweryController laborCamp = new BreweryController(data);
   26:                 assertTrue(laborCamp.getWorth() == 3);
   27:         }
   28:
   29:         @Test
   30:         public void testGetDescription() {
   31:                 BreweryData data = new BreweryData(1, 13, 3, 2000);
   32:                 BreweryController laborCamp = new BreweryController(data);
   33:                 assertTrue(!laborCamp.getDescription().isEmpty());
   34:         }
   35:
   36:         @Test
   37:         public void testLandOnField(){
   38:                 BreweryData data = new BreweryData(100, 2, 4000, 2000);
   39:                 BreweryController laborCamp = new BreweryController(data);
   40:                 Player player1 = new Player("Test1");
   41:                 Player player2 = new Player("Test2");
   42:                 laborCamp.pushToGUI(2);
   43:                 laborCamp.buyField(player2);
   44:                 laborCamp.landOnField(player1);
   45:
   46:
   47:                 boolean wasTrue = false;
```

```
   48:                 for (int i = 2; i <= 12; i++)
   49:                         if(player1.getAccount().getGold() == 30000 - i * 100 && player2.getAccount().getGold() == 26000 + i
 * 100)
   50:                                 wasTrue = true;
   51:
   52:                         assertTrue(wasTrue);
   53:                 }
   54:
   55: }
```

```
 1: package test;
 2:
 3: import static org.junit.Assert.*;
 4:
 5: import org.junit.Test;
 6:
 7: import Chancecards.ChanceCardBuildingTaxController;
 8: import Chancecards.ChanceCardBuildingTaxData;
 9: import game.Player;
10: import game.Account;
11: import slots.TerritoryController;
12: import slots.TerritoryData;
13:
14:
15: public class ChanceCardBuildingTaxControllerTest {
16:
17:         private ChanceCardBuildingTaxData data;
18:         private ChanceCardBuildingTaxController ccTax;
19:         private Player player;
20:         private TerritoryController felt, felt1;
21:         private TerritoryData territoryData, territoryData1;
22:         private Account acc;
23:         private int[] buildingtax;
24:
25:
26:
27:         @Test
28:         public void testBuildingTax() {
29:
30:                 data = new ChanceCardBuildingTaxData(1, 1000, 2000);
31:
32:                 acc = new Account(0, "ParkingLot");
33:                 player = new Player("Test");
34:
35:                 territoryData = new TerritoryData(1, 1, 2500, 500, 1000, buildingtax);
36:                 territoryData1 = new TerritoryData(2, 2, 5000, 1500, 2000, buildingtax);
37:                 felt = new TerritoryController(territoryData);
38:                 felt1 = new TerritoryController(territoryData1);
39:                 felt.pushToGUI(1);
40:                 felt.buyField(player);
41:                 felt.buyHouse(player);
42:                 felt.buyHouse(player);
43:                 felt.buyHouse(player);
44:                 felt.buyHouse(player);
45:                 felt.buyHouse(player);
46:
47:                 felt1.pushToGUI(2);
48:                 felt1.buyField(player);
```

```
49:                 felt1.buyHouse(player);
50:                 felt1.buyHouse(player);
51:                 felt1.buyHouse(player);
52:                 felt1.buyHouse(player);
53:
54:                 System.out.println(felt.getHotelAmount());
55:                 System.out.println(felt.getHouseAmount());
56:
57:                 player.getAccount().setGold(30000);
58:
59:                 ccTax = new ChanceCardBuildingTaxController(data, acc);
60:                 ccTax.onDrawn(player);
61:
62:                 System.out.println(player.getAccount().getGold());
63:                 System.out.println(acc.getGold());
64:
65:                 assertTrue(player.getAccount().getGold() == 24000);
66:                 assertTrue(acc.getGold() == 6000);
67:         }
68:
69: }
```

```java
package test;

import static org.junit.Assert.*;

import org.junit.Test;

import Chancecards.ChanceCardCashData;
import Chancecards.ChanceCardCashTransferController;
import game.Player;

public class ChanceCardCashTransferTest {

	private Player player1;
	private Player player2;
	private ChanceCardCashTransferController cf;
	private ChanceCardCashData cd;

	@Test
	public void Test() {
		player1 = new Player("p1");
		player2 = new Player("p2");
		Player[] players = new Player[2];
		players[0] = player1;
		players[1] = player2;
		cd = new ChanceCardCashData(0, 500);
		cf = new ChanceCardCashTransferController(cd, players);
		player1.getAccount().setGold(1000);
		player2.getAccount().setGold(1000);
		cf.onDrawn(player1);
		assertTrue(player1.getAccount().getGold() == 1500);
		assertTrue(player2.getAccount().getGold() == 500);
	}

}
```

```
    1: package test;
    2:
    3: import static org.junit.Assert.assertFalse;
    4:
    5:
    6: import org.junit.Test;
    7:
    8: import Chancecards.ChanceCardController;
    9: import game.Account;
   10: import game.ChanceCardLoader;
   11: import game.Player;
   12: import game.Prison;
   13:
   14: public class ChanceCardLoaderTest {
   15:         private Prison prison;
   16:         private Account parkinglotAccount;
   17:         private Player[] players;
   18:
   19:         @Test
   20:         public void testParseCards() {
   21:                 ChanceCardController[] cards = ChanceCardLoader.parseChanceCards("ChanceCard.xml", parkinglotAccount, priso
n, players);
   22:                 assertFalse("Failed to parse cards!", cards==null);
   23:                 for(ChanceCardController c : cards)
   24:                 {
   25:                         assertFalse("Parsed array contained a null reference", c==null);
   26:                         assertFalse("Parsed description contained an empty string", c.getDescription().isEmpty());
   27:                 }
   28:
   29:         }
   30: }
```

```
 1: package test;
 2:
 3: import static org.junit.Assert.*;
 4:
 5: import org.junit.Test;
 6:
 7: import Chancecards.ChanceCardCashData;
 8: import Chancecards.ChanceCardMatadorLegatController;
 9: import game.Player;
10: import slots.TerritoryController;
11: import slots.BreweryController;
12: import slots.TerritoryData;
13: import slots.BreweryData;
14:
15: public class ChanceCardMatadorLegatControllerTest {
16:
17:         private Player player;
18:         private TerritoryController tc, tc1;
19:         private TerritoryData td, td1;
20:         private BreweryController bc;
21:         private BreweryData bd;
22:         private ChanceCardMatadorLegatController ccMat;
23:         private ChanceCardCashData data;
24:         private int[] buildingtax;
25:
26:         @Test
27:         public void test() {
28:                 player = new Player("Test");
29:                 buildingtax = new int[6];
30:                 td = new TerritoryData(1, 1, 6200,1200,3000, buildingtax);
31:                 td1 = new TerritoryData(2, 2, 4200,800,2000, buildingtax);
32:                 tc = new TerritoryController(td);
33:                 tc1 = new TerritoryController(td1);
34:                 bd = new BreweryData(500,3,4000, 2000);
35:                 bc = new BreweryController(bd);
36:                 bc.pushToGUI(3);
37:                 bc.buyField(player);
38:                 tc.pushToGUI(1);
39:                 tc.buyField(player);
40:                 tc1.pushToGUI(2);
41:                 tc1.buyField(player);
42:                 tc1.buyHouse(player);
43:                 tc1.buyHouse(player);
44:                 tc1.buyHouse(player);
45:                 player.getAccount().setGold(7300);
46:
47:                 data = new ChanceCardCashData(0, 15000);
48:                 ccMat = new ChanceCardMatadorLegatController(data);
```

```
49:
50:                 ccMat.onDrawn(player);
51:
52:                 assertTrue(player.getAccount().getGold() == 7300);
53:
54:                 }
55:
56:         @Test
57:         public void test2()     {
58:                 player = new Player("Test");
59:                 buildingtax = new int[6];
60:                 td1 = new TerritoryData(2, 2, 4200,800,2000, buildingtax);
61:                 tc1 = new TerritoryController(td1);
62:                 tc1.pushToGUI(2);
63:                 tc1.buyField(player);
64:                 player.getAccount().setGold(7300);
65:
66:                 data = new ChanceCardCashData(0, 15000);
67:                 ccMat = new ChanceCardMatadorLegatController(data);
68:
69:                 ccMat.onDrawn(player);
70:
71:                 assertTrue(player.getAccount().getGold() == 22300);
72:         }
73: }
```

```
    1: package test;
    2:
    3: import static org.junit.Assert.*;
    4: import game.DiceCup;
    5: import game.DiceResult;
    6:
    7: import org.junit.Test;
    8:
    9: public class DiceCupTest {
   10:
   11:         @Test
   12:         public void testArrayLength() {
   13:                 final int TRUELENGTH = 1000;
   14:                 DiceCup dice = new DiceCup(TRUELENGTH);
   15:                 int a = dice.rollDice().getDiceAmount();
   16:
   17:                 assertTrue("Failed to create array with requested length", a == TRUELENGTH);
   18:
   19:         }
   20:
   21:         @Test
   22:         public void testProbability(){
   23:                 int antalSlag = 1000;
   24:                 DiceCup dice = new DiceCup(antalSlag);
   25:                 DiceResult a = dice.rollDice();
   26:                 double[] sider = new double[6];
   27:
   28:
   29:                 for(int i = 0; i < a.getDiceAmount(); i++){
   30:                         System.out.println(a.getDice(i));
   31:
   32:                         switch(a.getDice(i)){
   33:                         case 1: sider[0]++; break;
   34:                         case 2: sider[1]++; break;
   35:                         case 3: sider[2]++; break;
   36:                         case 4: sider[3]++; break;
   37:                         case 5: sider[4]++; break;
   38:                         case 6: sider[5]++; break;
   39:                         }
   40:                         System.out.println(sider[0] +" "+ sider[1] +" "+ sider[2] +" "+ sider[3] +" "+ sider[4] +" "+ sider
[5]);
   41:                 }
   42:
   43:                 for(int j = 0; j < 6; j++ ){
   44:                 double percent = ((sider[j] / antalSlag)*100);
   45:                 double deviation = Math.abs(percent - 1.0d/6*100);
   46:                 System.out.println("Die face: " + (j+1) + ", percent: " + percent + "%, deviation: " + deviation);
   47:                 assertTrue("The deviation was above accepted limit",2.5 > deviation);
```

```
48:                 }
49:         }
50: }
```

```
 1: package test;
 2:
 3: import static org.junit.Assert.*;
 4: import game.DiceResult;
 5:
 6: import org.junit.Test;
 7:
 8:
 9: public class DiceResultTest {
10:
11:         private int[] dice = {1,2,3,4,5,6};
12:         private DiceResult dicer = new DiceResult(dice);
13:         private int[] diceE = {3,3};
14:         private DiceResult dicerE = new DiceResult(diceE);
15:
16:         @Test
17:         public void testGetDice() {
18:                 assertTrue(dicer.getDice(3) == 4);
19:         }
20:
21:         @Test
22:         public void testGetSum() {
23:                 assertTrue(dicer.getSum() == 21);
24:         }
25:
26:         @Test
27:         public void testGetDiceAmount() {
28:                 assertTrue(dicer.getDiceAmount() == 6);
29:         }
30:
31:         @Test
32:         public void testAreDiceEqual() {
33:                 assertFalse(dicer.areDiceEqual());
34:                 assertTrue(dicerE.areDiceEqual());
35:         }
36:
37:         @Test
38:         public void testAreRollsEqual() {
39:                 assertFalse(dicer.areRollsEqual(dicerE));
40:                 assertTrue(dicer.areRollsEqual(dicer));
41:         }
42: }
```

```
    1: package test;
    2: import slots.FieldController;
    3: import utilities.ShuffleBag;
    4:
    5: import static org.junit.Assert.*;
    6:
    7: import org.junit.Test;
    8:
    9: import Chancecards.ChanceCardController;
   10: import game.Account;
   11: import game.FieldLoader;
   12: import game.Prison;
   13:
   14: public class FieldLoaderTest {
   15:         private Prison prison;
   16:         private Account parkinglotAccount;
   17:         private ShuffleBag<ChanceCardController> chanceCards;
   18:         private int[] buildingtax = new int[6];
   19:
   20:         final int EXPECTEDFIELDAMOUNT = 40;
   21:         @Test
   22:         public void testParseFields() {
   23:                 FieldController[] fields = FieldLoader.parseFields("Fields.xml", chanceCards, prison, parkinglotAccount, bu
ildingtax);
   24:                 assertFalse("Failed to parse fields!", fields==null);
   25:                 assertTrue("Failed to parse the expected amount of fields", fields.length==EXPECTEDFIELDAMOUNT);
   26:                 for(FieldController f : fields)
   27:                 {
   28:                         assertFalse("Parsed array contained a null reference", f==null);
   29:                         assertFalse("Parsed description contained an empty string", f.getDescription().isEmpty());
   30:                         assertFalse("Parsed name contained an empty string", f.getName().isEmpty());
   31:                 }
   32:
   33:         }
   34:
   35: }
```

```
 1: package test;
 2: import game.*;
 3: import slots.*;
 4: import static org.junit.Assert.*;
 5:
 6: import org.junit.Test;
 7:
 8: public class FleetControllerTest {
 9:
10:         @Test
11:         public void testGetRent() {
12:                 FleetData fleetData1 = new FleetData(1,100, 2000);
13:                 FleetController ship1 = new FleetController(fleetData1);
14:                 ship1.pushToGUI(1);
15:                 Player player1 = new Player("Sheep");
16:
17:                 ship1.setOwner(player1);
18:                 assertTrue("Fail, the rent should be 500 with 1 fleet owned",ship1.getRent()==500);
19:                 player1.getProperty().addFleet(ship1);
20:                 assertTrue("Fail, the rent should be 1000 with 2 fleet owned",ship1.getRent()==1000);
21:                 player1.getProperty().addFleet(ship1);
22:                 assertTrue("Fail, the rent should be 1500 with 3 fleet owned",ship1.getRent()==2000);
23:                 player1.getProperty().addFleet(ship1);
24:                 assertTrue("Fail, the rent should be 2000 with 4 fleet owned",ship1.getRent()==4000);
25:         }
26: }
```

```java
package test;

import static org.junit.Assert.*;

import org.junit.Test;
import game.*;
import slots.*;

public class GoToPrisonControllerTest {

	@Test
	public void testGoToPrisonController() {
		Player player1 = new Player("sheep");
		Prison prison = new Prison(1);
		GoToPrisonData prisonData = new GoToPrisonData(1,11, prison);
		GoToPrisonController pController = new GoToPrisonController(prisonData);

		pController.pushToGUI(11);
		pController.landOnField(player1);

		assertTrue("Fail,Prison should not be empty",prison.getInmate(player1)!=null);
		//Does not work
	}

}
```

```
 1: package test;
 2:
 3:
 4: import org.junit.Test;
 5:
 6: import game.Board;
 7: import game.DiceCup;
 8: import game.TestDice;
 9:
10: public class GotoPrisonTest {
11:
12:         @Test
13:         public void test() {
14:                 TestDice dice = new TestDice(new int[][]{{30,0},{30,0},{6,6},{6,6}});
15:                 Board board = new Board((DiceCup)dice);
16:                 board.startGame();
17:         }
18:
19: }
```

```
    1: package test;
    2:
    3:
    4:
    5: import org.junit.Test;
    6:
    7: import game.Board;
    8: import game.DiceCup;
    9: import game.TestDice;
   10:
   11: public class landOnChanceField {
   12:
   13:         @Test
   14:         public void test() {
   15:                 TestDice dice = new TestDice(new int[][]{{2,0},{2,0},{2,0},{2,0},{2,0},{2,0},{5,0},{5,0},{5,0},{5,0},{5,0},
{5,0},{10,0},{10,0},{10,0},{10,0},{10,0},{10,0}});
   16:                 Board board = new Board((DiceCup)dice);
   17:                 board.startGame();
   18:         }
   19:
   20: }
```

```
 1: package test;
 2:
 3: import static org.junit.Assert.*;
 4:
 5: import org.junit.Test;
 6:
 7: import game.*;
 8: import slots.*;
 9:
10: public class ParkinglotControllerTest {
11:
12:         @Test
13:         public void test() {
14:
15:                 Account parkinglotAcc = new Account(10000, "PAA");
16:                 ParkinglotData parkinglotData = new ParkinglotData (1, parkinglotAcc);
17:                 ParkinglotController parkinglot = new ParkinglotController(parkinglotData);
18:
19:                 Player player = new Player("Sheep");
20:
21:                 parkinglot.pushToGUI(1);
22:                 parkinglot.landOnField(player);
23:
24:                 assertTrue(player.getAccount().getGold()== 40000);
25:         }
26:
27: }
```

```
    1: package test;
    2: import static org.junit.Assert.assertTrue;
    3:
    4: import org.junit.Before;
    5: import org.junit.Test;
    6:
    7: import game.Player;
    8:
    9: public class PlayerTest {
   10:
   11:         private Player player;
   12:
   13:         @Before
   14:         public void initialize(){
   15:         player = new Player("Sheep");
   16:         }
   17:
   18:         @Test
   19:         public void testGetAccount() {
   20:                 assertTrue("Fail, the player should have 30000 gold to start with.", player.getAccount().getGold()==30000);
   21:         }
   22:
   23:         @Test
   24:         public void testGetName(){
   25:                 assertTrue("Fail, the player should have the name \"Sheep\"", player.getName().equals("Sheep"));
   26:         }
   27:
   28:         @Test
   29:         public void testGetProperty() {
   30:                 assertTrue("Fail, A player should not own anything at the start of the game.", player.getProperty().getProp
ertyCount() == 0);
   31:         }
   32:
   33:         @Test
   34:         public void testMove() {
   35:                 player.move(22, true);
   36:                 assertTrue("Fail, the player should be on the first",player.getPosition() == 0 && player.getNextPosition()
== 22);
   37:         }
   38:
   39:         @Test
   40:         public void testMoveToNextPosition(){
   41:                 player.move(30, true);
   42:                 player.move(15, true);
   43:                 player.moveToNextPosition();
   44:                 assertTrue("Fail, the player should be on position 22", player.getPosition() == 5 && player.getAccount().ge
tGold() == 34000);
   45:         }
```

```
   46:
   47:         @Test
   48:         public void testSetNextPosition(){
   49:                 player.setNextPosition(15, true);
   50:                 player.moveToNextPosition();
   51:                 assertTrue("Fail, the player should be on position 15", player.getPosition() == 15);
   52:         }
   53:
   54:         @Test
   55:         public void testHasGetOutOfPrisonCard(){
   56:                 assertTrue("Fail, the player should not start with the getOutOfPrisonCard", !player.hasGetOutOfPrisonCard()
);
   57:         }
   58:
   59:         @Test
   60:         public void testSetGetOutOfPrisonCard(){
   61:                 player.setHasGetOutOfPrisonCard(true);
   62:                 assertTrue("Fail, the player should have the getOutOfPrisonCard", player.hasGetOutOfPrisonCard());
   63:         }
   64:
   65: }
```

```
 1: package test;
 2:
 3: import org.junit.Test;
 4:
 5: import game.Board;
 6: import game.DiceCup;
 7: import game.TestDice;
 8:
 9: public class PrisonTest {
10:
11:         @Test
12:         public void test(){
13:                 TestDice dice = new TestDice(new int[][] {{5,5}});
14:                 Board board = new Board((DiceCup)dice);
15:                 board.startGame();
16:         }
17: }
```

```
 1: package test;
 2:
 3: import static org.junit.Assert.*;
 4:
 5:
 6: import game.*;
 7: import slots.OwnableController.FIELDGROUPS;
 8: import slots.TerritoryController;
 9: import slots.TerritoryData;
10:
11: import org.junit.Test;
12:
13: import desktop_fields.Field;
14: import desktop_resources.GUI;
15:
16: public class PropertyTest {
17:
18:         private Player player = new Player("player1");
19:
20:         @Test
21:         public void ownsEntireGroupTest() {
22:
23:                 int[] a = {1,1,1,1,1};
24:                 TerritoryData data1 = new TerritoryData(2, 0, 1, 1, 1, a);
25:                 TerritoryController ctrl1 = new TerritoryController(data1);
26:                 TerritoryData data2 = new TerritoryData(4, 0, 1, 1, 1, a);
27:                 TerritoryController ctrl2 = new TerritoryController(data2);
28:                 Field fctrl1 = ctrl1.pushToGUI(1);
29:                 Field fctrl2 = ctrl2.pushToGUI(2);
30:                 GUI.create(new Field[]{fctrl1, fctrl2});
31:                 Property prop = player.getProperty();
32:
33:                 ctrl1.buyField(player);
34:                 assertFalse(prop.ownsEntireGroup(FIELDGROUPS.BLUE));
35:                 ctrl2.buyField(player);
36:                 assertTrue(prop.ownsEntireGroup(FIELDGROUPS.BLUE));
37:         }
38:
39:         @Test
40:         public void ownsEntireGroupTestWithThreeFieldGroupAndOnePlayerTest(){
41:                 int[] a = {1,1,1,1,1};
42:                 TerritoryData data3 = new TerritoryData(7, 1, 1, 1, 1, a);
43:                 TerritoryController ctrl3 = new TerritoryController(data3);
44:                 TerritoryData data4 = new TerritoryData(9, 1, 1, 1, 1, a);
45:                 TerritoryController ctrl4 = new TerritoryController(data4);
46:                 TerritoryData data5 = new TerritoryData(10, 1, 1, 1, 1, a);
47:                 TerritoryController ctrl5 = new TerritoryController(data5);
48:                 Field fctrl3 = ctrl3.pushToGUI(1);
```

```
49:                 Field fctrl4 = ctrl4.pushToGUI(2);
50:                 Field fctrl5 = ctrl5.pushToGUI(3);
51:                 GUI.create(new Field[]{fctrl3, fctrl4, fctrl5});
52:                 Property prop = player.getProperty();
53:
54:                 ctrl3.buyField(player);
55:                 assertFalse(prop.ownsEntireGroup(FIELDGROUPS.PINK));
56:                 ctrl4.buyField(player);
57:                 assertFalse(prop.ownsEntireGroup(FIELDGROUPS.PINK));
58:                 ctrl5.buyField(player);
59:                 assertTrue(prop.ownsEntireGroup(FIELDGROUPS.PINK));
60:         }
61: }
```

```java
package test;
import utilities.ShuffleBag;

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

public class ShuffleBagTest {

	@Test
	public void testRandomOutput() {
		int amount = 0;
		Integer[] testtal1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
		ShuffleBag<Integer> test = new ShuffleBag<Integer>(testtal1);
		int j = 0;
		boolean random = false;
		for(int i = 0; i < testtal1.length; i++) {
			try {
				int k = test.getNext();
				amount = amount + k;
				if (j > k) {
				random = true;
				}
				j = k;
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
		assertTrue(random);
		assertTrue(amount == 210);


	}

	@Test
	public void testPutBackInBag() throws Exception{
		Integer[] testtal = {1,2,3,4,5};
		ShuffleBag<Integer> testbag = new ShuffleBag<Integer>(testtal);
		int firstNumber = testbag.getNext();
		testbag.pushBackLastElement();
		while(testbag.getElementsLeft()!=0)
		{
			int num = testbag.getNext();
			if(firstNumber==num)
			{
				assertTrue("The element could be found again after being put back into the bag", true);
			}
		}
	}
```

```
49:
50:         @Test
51:         public void testResetSuffleBag() {
52:                 Integer[] testtal1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
53:                 ShuffleBag<Integer> testreset = new ShuffleBag<Integer>(testtal1);
54:                 int j = 0;
55:                 boolean random = false;
56:                 int amount = 0;
57:                 for(int i = 0; i < testtal1.length; i++) {
58:                         try {
59:                                 int k = testreset.getNext();
60:                                 amount = amount + k;
61:                                 if (j > k) {
62:                                 random = true;
63:                                 }
64:                                 j = k;
65:                         } catch (Exception e) {
66:                                 e.printStackTrace();
67:                         }
68:                 }
69:                 assertTrue("Random were not random!", random);
70:                 testreset.reset();
71:                 amount = 0;
72:                 for(int i = 0; i < testtal1.length; i++) {
73:                         try {
74:                                 int k = testreset.getNext();
75:                                 amount = amount + k;
76:                                 if (j > k) {
77:                                 random = true;
78:                                 }
79:                                 j = k;
80:                         } catch (Exception e) {
81:                                 e.printStackTrace();
82:                         }
83:                 }
84:                 assertTrue(amount == 210);
85:         }
86:
87: }
```

```java
package test;

import static org.junit.Assert.*;

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

import game.Account;
import game.Player;
import slots.TaxData;
import slots.TaxController;

public class TaxTest {


	private Account acc;
	private TaxData taxD;
	private TaxController tax;
	private Player player;

	@Before
	public void preTest(){

		acc = new Account(0, "Test");
		taxD = new TaxData(1, 2000, 10);
		tax = new TaxController(taxD, acc);
		player = new Player("Test");
	}

	@Test
	public void testFlatTax() {

		tax.pushToGUI(1);
		tax.landOnField(player);


		assertTrue(player.getAccount().getGold()== 27000 || player.getAccount().getGold()== 28000);
		assertTrue(acc.getGold()==2000 || acc.getGold()==3000);


	}
//
//	@Test
//	public void testPercentageTax

}
```

```java
package test;

import static org.junit.Assert.*;

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

import game.Player;
import slots.TerritoryController;
import slots.TerritoryData;

public class TerritoryTest {

	private TerritoryData data;
	private TerritoryController territoryController;
	private Player player1, player2;
	private int[] buildingtax;

	@Before
	public void initialize(){
		data = new TerritoryData(2, 2, 2500, 500, 1000, buildingtax);
		territoryController = new TerritoryController(data);

		player1 = new Player("Test1");
		player2 = new Player("Test2");

		territoryController.pushToGUI(2);
		player1.getAccount().removeGold(2500);
		data.setOwner(player1);
	}

	@Test
	public void testLandOnField() {
		territoryController.landOnField(player2);
		assertTrue(player1.getAccount().getGold()== 28000 && player2.getAccount().getGold()== 29500);
		}

	@Test
	public void testGetHotelAmount(){
		assertTrue("Fejl, du starter med ingen hoteller", territoryController.getHotelAmount() == 0);
	}

	@Test
	public void  testGetHouseAmount(){
		assertTrue("Fejl, du starter med ingen hoteller", territoryController.getHouseAmount() == 0);
	}

	@Test
```

```
   49:         public void testGetWorth(){
   50:                 assertTrue("Fejl, vÃ¦rdien burde vÃ¦re 2500, da der ingen huse og hoteller er", territoryController.getWort
h() == 2500);
   51:         }
   52:
   53:         @Test
   54:         public void testGetDescription(){
   55:                 assertTrue("Fejl, beskrivelsen burde ikke vÃ¦re tom", !territoryController.getDescription().isEmpty());
   56:         }
   57:
   58:         @Test
   59:         public void testGetUpgradeCost(){
   60:                 assertTrue("Fejl, opgraderingen burde vÃ¦re 1000", territoryController.getUpgradeCosts() == 1000);
   61:         }
   62:
   63:         @Test
   64:         public void testBuyHouse(){
   65:                 territoryController.buyHouse(player1);
   66:                 assertTrue("Fejl, huset blev ikke kÃ¸bt", territoryController.getHouseAmount() == 1);
   67:                 assertTrue("Fejl, pengene blev ikke overfÃ¸rt korrekt", player1.getAccount().getGold() == 26500);
   68:                 assertTrue("Fejl, vÃ¦rdien skulle vÃ¦re 3500", territoryController.getWorth() == 3500);
   69:         }
   70: }
```

```
    1: package utilities;
    2:
    3: import java.util.Random;
    4:
    5: public class ShuffleBag<T> {
    6:         private T[] values;
    7:         private int currentPos;
    8:         private Random rng = new Random(System.currentTimeMillis());
    9:         public ShuffleBag(T[] values)
   10:         {
   11:                 this.values = values;
   12:                 reset();
   13:         }
   14:
   15:         public int getElementsLeft()
   16:         {
   17:                 //+1 since currentPos goes from 0, and hence if the first element(0) is left it would return 0.
   18:                 return currentPos+1;
   19:         }
   20:         public void reset()
   21:         {
   22:                 currentPos = values.length-1;
   23:         }
   24:
   25:         private void swapToEnd(int index)
   26:         {
   27:                 T tmpValue = values[index];
   28:                 values[index] = values[currentPos];
   29:                 values[currentPos] = tmpValue;
   30:                 //Removed the used index out of scope.
   31:                 --currentPos;
   32:         }
   33:         public void pushBackLastElement()
   34:         {
   35:                 if(currentPos < values.length-1)
   36:                 {
   37:                         currentPos++;
   38:                 }
   39:         }
   40:         public T getNext() throws Exception
   41:         {
   42:                 if(currentPos==-1)
   43:                 {
   44:                         throw new Exception("Shuffle bag has run out of free elements. If this is intended call reset befor
e getting the next variable.");
   45:                 }
   46:                 //+1 due to the last index being exclusive
   47:                 int index = rng.nextInt(currentPos+1);
```

```
48:                 T value = values[index];
49:                 swapToEnd(index);
50:                 return value;
51:         }
52:
53: }
```