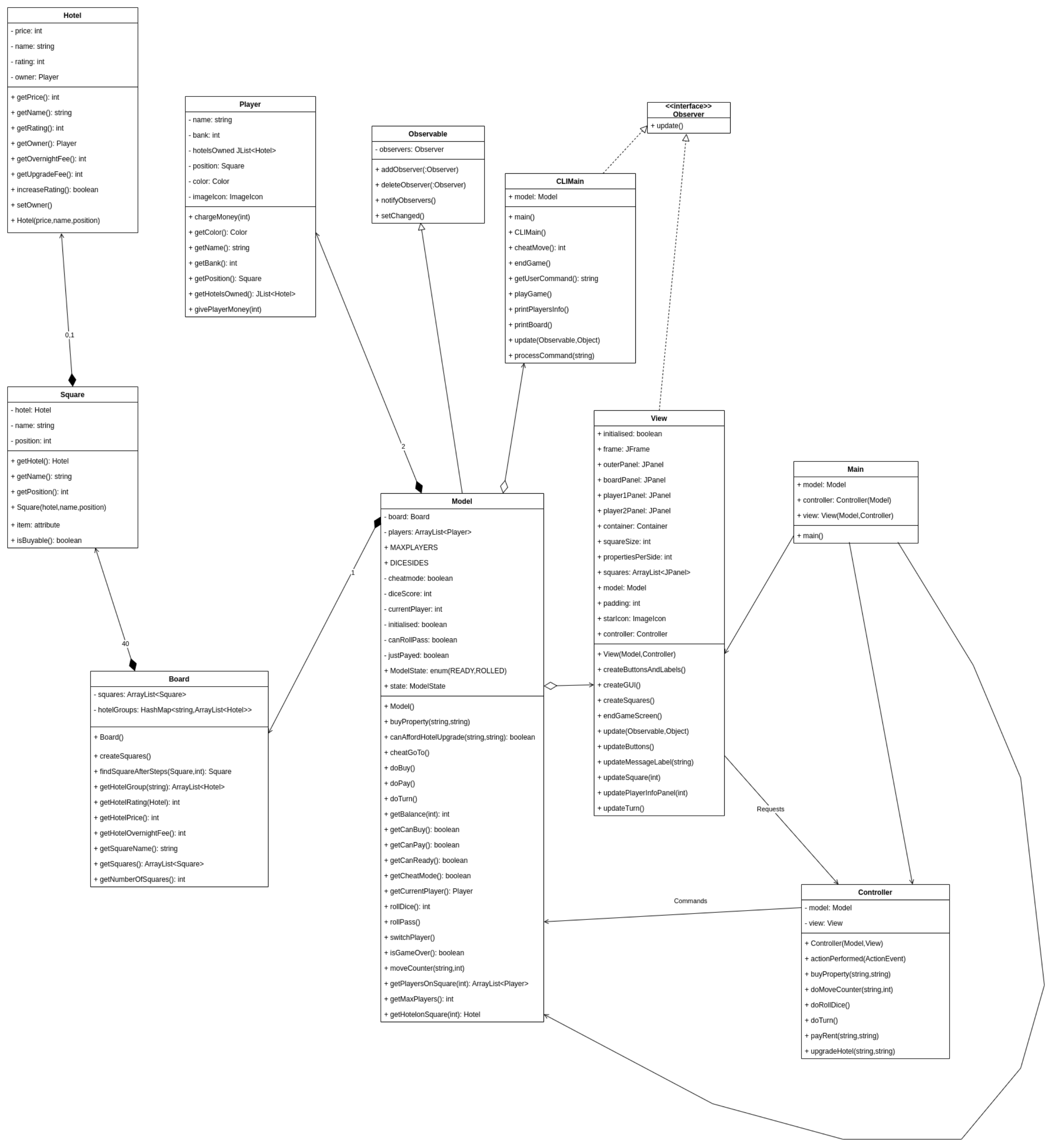
COMP6018 Coursework Deliverables

Video: https://drive.google.com/file/d/15IajR1wTb6FXkHwww8BjqbqovSoxPxL_/view?usp=share_link

Class Diagram:



Code:

Board

```
import java.util.ArrayList;
import java.util.HashMap;

public class Board {
    private ArrayList<Square> squares;
    private HashMap<String,ArrayList<Hotel>> hotelGroups;

    public Board() {
        createSquares();
    }

    public void createSquares() {
        // Data we will use to add onto the squares/JPanels as text
        this.squares = new ArrayList<Square>();
        String[] names = new String[]{"GO","A1", "", "A2", "A3", "", "B1", "", "B2", "B3", "", "C1",
"", "C2", "C3", "", "D1", "", "D2", "D3", "", "E1", "", "E2", "E3", "", "F1", "", "F2", "F3", "", "G1", "",
"G2", "G3", "", "H1", "", "H2", "H3"};
        int[] prices = new int[]{0, 50, 0, 50, 70, 0, 100, 0, 100, 120, 0, 150, 0, 150, 170, 0, 200, 0,
200, 220, 0, 250, 0, 250, 270, 0, 300, 0, 300, 320, 0, 350, 0, 350, 370, 0, 400, 0, 400, 420};

        // Property counter
        int p = 0;
        // Go through all names
        for (int i = 0; i < names.length; i++) {
            if (prices[i] > 0) {
                // Square with hotel
                this.squares.add(new Square(names[i],prices[i],i));
            } else {
                // Empty square
                this.squares.add(new Square(names[i],i));
            }

        }
        // Map first letter in a hotel group to the group of hotels, e.g: { "A" :
hotela1,hotela2,hotela3 }
        this.hotelGroups = new HashMap<String, ArrayList<Hotel>>();
        for (int i = 0; i < names.length; i++) {
            if (names[i].length() > 1 && isNumeric(names[i].substring(1,2))) {
                String groupkey = names[i].substring(0,1);
                // Check if they key already exists, if not then make the group from next positions
that are always the same
                if (!hotelGroups.containsKey(groupkey)) {
                    ArrayList<Hotel> hotelGroup = new ArrayList<Hotel>();
                    hotelGroup.add(squares.get(i).getHotel());
                    hotelGroup.add(squares.get(i+2).getHotel());
                    hotelGroup.add(squares.get(i+3).getHotel());
                    this.hotelGroups.put(groupkey,hotelGroup);
                }

            }
        }
    }

    private static boolean isNumeric(String value) {
        try {
```

```java
            Integer.parseInt(value);
            return true;
        } catch(NumberFormatException e) {
            return false;
        }

    }

    public ArrayList<Square> getSquares() {
        return this.squares;
    }

    public Square getSquareFromName(String squareName) {
        for (int i = 0; i < squares.size(); i++) {
            if (squares.get(i).getName() == squareName) {
                return squares.get(i);
            }
        }
        return null;
    }

    public Square getSquareFromIndex(int index) {
        if (index < this.squares.size()) {
            return this.squares.get(index);
        }
        return null;
    }

    public Square findSquareAfterSteps(Square startSquare, int stepsForward) {
        // Mod is to not go out of index range of 40 squares or whatever is the squares length
        int forwards = (this.squares.indexOf(startSquare)+stepsForward) % this.squares.size();
        return squares.get(forwards);
    }


    public String getSquareName(int squareIndex) {
        return squares.get(squareIndex).getName();
    }

    public int getHotelPrice(int squareIndex) {
        return squares.get(squareIndex).getHotelPrice();
    }

    public int getHotelOvernightFee(int squareIndex) {
        return squares.get(squareIndex).getHotelOvernightFee();
    }

    public int getHotelRating(int squareIndex) {
        return squares.get(squareIndex).getHotelRating();
    }

    public String getHotelOwnerName(int squareIndex) {
        if (squares.get(squareIndex).hasHotel()) {
            Player owner = squares.get(squareIndex).getHotelOwner();
            if (owner != null) {
                return owner.getName();
            }
        }
        return null;
    }
    public ArrayList<Hotel> getHotelGroup(String hotelName) {
        if (this.hotelGroups.containsKey(hotelName.substring(0,1))) {
            return this.hotelGroups.get(hotelName.substring(0,1));
        }
        return null;
    }

    public int getNumberOfSquares() {
        return this.squares.size();
    }

}
```

Square

```
import javax.swing.*;

public class Square {

    private Hotel hotel;
    private String name;
    private int position;

    Square(String name, int price, int position) {
        this.position = position;
        this.hotel = new Hotel(name,price);
    }

    Square(String name, int position) {
        this.position = position;
        this.name = name;
    }

    public int getPosition() {
        return this.position;
    }

    public boolean hasHotel() {
        return this.hotel != null;
    }

    public String getName() {
        if (hasHotel()) {
            return hotel.getName();
        } else {
            return this.name;
        }
    }

    public int getHotelPrice() {
        if (hasHotel()) {
            return hotel.getPrice();
        } else {
            return 0;
        }
```

```java
    }

    public int getHotelRating() {
        if (hasHotel()) {
            return hotel.getStarRating();
        }
        return 0;
    }

    public Player getHotelOwner() {
        if (hasHotel()) {
            return hotel.getOwner();
        }
        return null;
    }

    public int getHotelOvernightFee() {
        if (hasHotel()) {
            return hotel.getOvernightFee();
        }
        return 0;
    }

    public Player getOwner() {
        if (this.hasHotel()) {
            return hotel.getOwner();
        }
        return null;
    }

    public boolean isBuyable() {
        return this.hasHotel() && (this.getHotelOwner() == null);
    }

    public Hotel getHotel() {
        if (this.hasHotel()) {
            return this.hotel;
        }
        return null;
    }

    public void setHotel(Hotel hotel) {
        this.hotel = hotel;
    }

}
```

Hotel

```java
import javax.swing.*;

public class Hotel {
    private int price;
    private String name;
    private Player owner;
    private int rating;
    public static final int MAXRATING = 5;
    public Hotel(String name, int price) {
        this.name = name;
        this.price = price;
        this.rating = 0;
        this.owner = null;

    }

    public int getUpgradeFee() {
        return price / 2;
    }

    public int getOvernightFee() {
        if (owner == null) {
            return 0;
        } else {
            return (this.price/10)*(this.rating*this.rating);
        }
    }

    public String getName() {
        return this.name;
    }

    public int getStarRating() {
        return rating;
    }

    public boolean increaseStarRating() {
        if (rating < MAXRATING) {
            rating++;
            return true;
        }
        return false;
    }

    public int getPrice() {
        return this.price;
    }

    public boolean setOwner(Player player) {
        if (owner == null) {
            owner = player;
            return true;
        }
        return false;
    }

    public boolean hasOwner() {
        return owner != null;
    }

    public Player getOwner() {
        return owner;
    }
```

```
}
```

Player

```
import javax.lang.model.type.NullType;
import javax.swing.*;
import java.awt.*;

public class Player {
    private String name;
    private int bank;
    private JList<Hotel> hotelsOwned;
    private Square position;
    private Color color;
    ImageIcon imageIcon;

    public Player(String name, Color color,ImageIcon imageIcon) {
        this.name = name;
        this.position = null;
        this.hotelsOwned = new JList<Hotel>();
        this.bank = 2000;
        this.color = color;
        this.imageIcon = imageIcon;
    }

    public ImageIcon getImageIcon() {
        return this.imageIcon;
    }

    public Color getColor() {
        return this.color;
    }

    public int getColorComponentRed() {
        return this.color.getRed();
    }
    public int getColorComponentBlue() {
        return this.color.getBlue();
    }
    public int getColorComponentGreen() {
        return this.color.getGreen();
    }
    public String getName() {
        return name;
    }

    public void recieveMoney(int money) {
        this.bank += money;
    }

    public int getBalance() {
        return bank;
    }

    public void giveMoneyToPlayer(int amount, Player payee) {
        this.bank -= amount;
        payee.recieveMoney(amount);
    }
```

```java
    public void chargeMoney(int amount) {
        this.bank -= amount;
    }

    public void setPosition(Square position) {
        this.position = position;
    }

    public Square getPosition() {
        return this.position;
    }

    public boolean isBankrupt() {

        return this.bank <= 0;
    }
}
```

Model

```java
import javax.swing.*;
import java.awt.*;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.lang.Math;
import java.util.Observable;

// Model is given commands from controller
// it can then update the controller on data changes
// and ask it what to do
// the controller will tell it what to do, it doesn't decide to do

public class Model extends Observable {
    private Board board;
    private ArrayList<Player> players;
    public static final int MAXPLAYERS = 2;
    public static final int DICESIDES = 12;
    private boolean cheatmode;
    private int diceScore;
    private int currentPlayer;
    private boolean initialised;
    private boolean canRollPass = false;
    private boolean justPayed = false;
    public enum ModelState{
        READY_TO_ROLL,
        ROLLED
    }
    ModelState state = ModelState.READY_TO_ROLL;

    public Model(boolean cheatmode) {
        this.cheatmode = cheatmode;
        this.diceScore = 0;


        this.board = new Board();
        initialisePlayers();
        this.canRollPass = true;

    }

    public boolean getCheatMode() {
        return this.cheatmode && this.state == ModelState.READY_TO_ROLL;
    }

    public void cheatGoTo(int squareindex) {
        if (this.cheatmode && state == ModelState.READY_TO_ROLL) {
            Square square = this.board.getSquareFromIndex(squareindex);
            int currentPlayerSquare = this.getCurrentPlayerPosition();
            if (squareindex > currentPlayerSquare) {
                if (squareindex - currentPlayerSquare > 12) {
                    setChanged();
                    notifyObservers("Cheat mode more than 12 squares is illegal.");
                    return;
                }
            } else if (squareindex < currentPlayerSquare) {
                int finalIndex = squareindex + this.getMaxSquares();
                if ((finalIndex - currentPlayerSquare) > 12) {
                    setChanged();
                    notifyObservers("Cheat mode more than 12 squares is illegal.");
                    return;
                }
            } else {
                // Clicked on same square (moved 0)
                setChanged();
                notifyObservers("Cheat mode cannot move 0 squares.");
                return;
            }
            this.getCurrentPlayer().setPosition(square);
            state = ModelState.ROLLED;
            // Update all buttons
            doTurn();
            setChanged();
            notifyObservers("Cheat mode: moved " + getCurrentPlayerName() + " to square " +
square.getName());

        }
    }
    public boolean getCanBuy() {
        Square location = this.getCurrentPlayer().getPosition();
        return this.state == ModelState.ROLLED && location.isBuyable() &&
this.getCurrentPlayer().getBalance() >= location.getHotelPrice();
    }

    public boolean getCanPay() {
        Square location = this.getCurrentPlayer().getPosition();
        if (this.state == ModelState.READY_TO_ROLL) {
            return false;
        }
        else if (location.getHotel() == null) {
            return false;
        } else if (!location.getHotel().hasOwner()) {
            return false;
        } else if (location.getHotel().getOwner() == this.getCurrentPlayer() &&
location.getHotel().getUpgradeFee() <= this.getCurrentPlayer().getBalance() &&
location.getHotel().getStarRating() < Hotel.MAXRATING) {
            return true;
        } else if (location.getHotel().getOwner() != this.getCurrentPlayer() && !justPayed) {
            return true;
        }
        return false;
    }

    public boolean getCanRollPass() {
        return this.canRollPass;
    }



    /** Returns an ImageIcon, or null if the path was invalid. */
    public ImageIcon createImageIcon(String path, String description) {
        File file = new File("./");
        try {
            String pathToIcon = new String(file.getCanonicalPath()+"/"+path);
            return new ImageIcon(pathToIcon, description);

        } catch (IOException e) {
            System.err.println("Couldn't find file: " + path);
        }
        return null;
    }



    private void initialisePlayers() {
        /** @pre. this.players is null
         * @post. 2 players created, both have £2000, both start at position 0 and both players
are
         * in the players list.
         */
        assert (this.players == null) : "players must be null";



        this.players = new ArrayList<Player>();
        ImageIcon icon1 = createImageIcon("resources/car4.png","player1");
        Player player1 = new Player("player1",Color.yellow,icon1);
        player1.setPosition(this.board.getSquareFromIndex(0));



        ImageIcon icon2 = createImageIcon("resources/car2.png","player2");
        Player player2 = new Player("player2",Color.cyan,icon2);
        player2.setPosition(this.board.getSquareFromIndex(0));
        this.players.add(player1);
        this.players.add(player2);
        this.currentPlayer = 0;

        assert(null != player1) : "Error: player1 was not created correctly.";
        assert(null != player2) : "Error: player2 was not created correctly.";

        // Check both players have 2000 pounds
        assert(2000 == player1.getBalance()) : "Error: Player1 does not start with 2000.";
        assert(2000 == player2.getBalance()) : "Error: Player2 does not start with 2000.";

        // Check both players in position 0
        assert(0 == player1.getPosition().getPosition()) : "Error: player1 does not start at index 0
squares.";
        assert(0 == player2.getPosition().getPosition()) : "Error: player2 does not start at index 0
squares.";
```

```java
        assert(this.players.contains(player1)) : "Error: player1 is not in the players list.";
        assert(this.players.contains(player2)) : "Error: player2 is not in the players list.";

    }

    public String getCurrentPlayerName() {
        return this.players.get(this.currentPlayer).getName();
    }

    public int getPlayerBalance(String playerName) {
        for (int i = 0; i < this.players.size(); i++) {
            if (this.players.get(i).getName() == playerName) {
                return this.players.get(i).getBalance();
            }
        }
        return 0;
    }

    public boolean getInitialised() {
        return this.initialised;
    }

    public void setInitialised(boolean initialised) {
        this.initialised = initialised;
    }

    public void initialiseModel() {
        this.board = new Board();
        initialisePlayers();
        this.canRollPass = true;
        this.state = ModelState.READY_TO_ROLL;
        this.initialised = true;
        setChanged();
        notifyObservers("Starting new game.");
    }

    public boolean isGameOver() {
        for (int i = 0; i < this.players.size(); i++) {
            if (this.players.get(i).isBankrupt()) {
                return true;
            }
        }
        return false;
    }

    private Player getCurrentPlayer() {
        return this.players.get(this.currentPlayer);
    }

    public String getWinnerName() {
        if (isGameOver()) {
            if (getCurrentPlayer().isBankrupt()) {
                return this.players.get((currentPlayer+1)%this.players.size()).getName();
            } else {
                return getCurrentPlayerName();
            }
        }
        return null;
    }
    public String getSquareName(int squareIndex) {
        return board.getSquareName(squareIndex);
    }

    public int getHotelPrice(int squareIndex) {
        return board.getHotelPrice(squareIndex);
    }

    public int getHotelOvernightFee(int squareIndex) {
        return board.getHotelOvernightFee(squareIndex);
    }

    public int getHotelRating(int squareIndex) {
        return board.getHotelRating(squareIndex);
    }

    public String getHotelOwnerName(int squareIndex) {
        return board.getHotelOwnerName(squareIndex);
    }

    public ImageIcon getPlayerImageIcon(String playerName) {
        /** @pre. playerName exists in players
         *
         */
        assert(players.get(0).getName().equals(playerName) ||
players.get(1).getName().equals(playerName)) : "Error: precondition failed. No player with that
name.";

        Player player = this.getPlayerFromName(playerName);
        return player.getImageIcon();
    }

    public String getPlayerName(int playerIndex) {
        /** @pre. playerIndex < player.size()
         *
         */
        assert(playerIndex < players.size()) : "Error: precondition failed. Invalid player index.";
        return players.get(playerIndex).getName();
    }

    public int getBalance(int playerIndex) {
        /** @pre. playerIndex < player.size()
         * @post. returns playerBalance of players(playerIndex)
         */
        assert(playerIndex < players.size()) : "Error: precondition failed. Invalid player index.";
        return players.get(playerIndex).getBalance();
    }

    public ArrayList<String> getPlayerNamesOnSquare(int squareIndex) {
        ArrayList<String> names = new ArrayList<String>();
        Square square = this.board.getSquareFromIndex(squareIndex);
        for (int i = 0; i < this.players.size(); i++) {
            if (this.players.get(i).getPosition() == square) {
                names.add(this.players.get(i).getName());
            };
        }
        return names;
    }
    public ImageIcon getSmallImageIcon(String playerName) {
        return new
ImageIcon(this.getPlayerImageIcon(playerName).getImage().getScaledInstance(32,32,Image.S
CALE_DEFAULT));

    }

    public void switchPlayer() {
        // Increase index, and mod by players length to avoid index out of range
        int curPlayer = (this.currentPlayer + 1) % this.players.size();
        this.currentPlayer = curPlayer;
        this.justPayed = false;
        setChanged();
        notifyObservers("Switch player turn to "+this.getCurrentPlayerName());
    }

    public void doBuy() {
        Player player = this.getCurrentPlayer();
        Square square = player.getPosition();
        this.buyProperty(player.getName(),square.getName());
    }

    public void doPay() {
        Player player = this.getCurrentPlayer();
        Square square = player.getPosition();
        Player owner = square.getOwner();
        if (player == owner) {
            // Free stay and upgrade hotel available
            this.upgradeHotel(player.getName(),square.getName());
            doTurn();
        } else if (owner != null) {
            this.payRent(player.getName(),square.getName());
            if (this.isGameOver()) {
                setChanged();
                notifyObservers("Game over!");
            }
        }
    }

    public void rollPass() throws InterruptedException {
        // Decided whether to roll dice or pass to next player
        if (this.state == ModelState.READY_TO_ROLL) {
            int diceroll = this.rollDice();
            setChanged();
            notifyObservers("Dice roll is "+ diceroll);
            Thread.sleep((long)100);
            this.moveCounterForwards(this.getCurrentPlayerName(),diceroll);
            this.state = ModelState.ROLLED;
            doTurn();
            setChanged();
            notifyObservers(this.getCurrentPlayerName()+" has moved forwards by "+diceroll+"
squares, to "+this.getCurrentPlayer().getPosition().getName());



        } else if (this.state == ModelState.ROLLED) {
            this.switchPlayer();
            this.state = ModelState.READY_TO_ROLL;
        }
    }

    public int getCurrentPlayerPosition() {
        int curPlayer = this.getCurrentPlayer().getPosition().getPosition();
        return curPlayer;
    }

    public ArrayList<Square> getSquares() {
        return this.board.getSquares();
    }

    public int getMaxSquares() {
        return this.board.getSquares().size();
    }

    public int rollDice() {
        // Random number * MAXNUMBER + 1 and cast to int which truncates (cuts off the
end/any floating numbers)
        // Gives random number from 0-1 then uses dicesides
        // 0.9 * 12 = 10.8 + 1 = 11.8 > truncate to int = 11
        // 0.95 * 12 = 11.4 + 1 = 12.4 > truncate to int = 12
        this.diceScore = (int)(Math.random()*DICESIDES+1);
        setChanged();
        notifyObservers("Dice roll is "+diceScore);
        System.out.println(this.diceScore);
        return this.diceScore;
    }

    // Helper method
    protected Player getPlayerFromName(String playerName) {
        for (int i = 0; i < players.size(); i++) {
            if (players.get(i).getName() == playerName) {
                return players.get(i);
            }
        }
        return null;
    }

    public int getMaxPlayers() {
        return this.players.size();
    }

    public void moveCounterForwards(String playerName, int diceNumber) {
        Player player = getPlayerFromName(playerName);
        player.setPosition(this.board.findSquareAfterSteps(player.getPosition(),diceNumber));
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

    }

    public void buyProperty(String playerName, String squareName) {
        Player player = getPlayerFromName(playerName);
        Square location = board.getSquareFromName(squareName);
        if (location.isBuyable() && player.getBalance() >= location.getHotelPrice()) {
            player.chargeMoney(location.getHotelPrice());
            location.getHotel().setOwner(player);
            // Change
            setChanged();
            notifyObservers(playerName+" has purchased "+squareName+" for
£"+location.getHotelPrice());
        }
```

```java
        else if (player.getBalance() < location.getHotelPrice()) {
            setChanged();
            notifyObservers("Can't buy hotel, not enough money.");
        } else if (location.isBuyable() == false) {
            setChanged();
            notifyObservers("Can't buy Hotel.");
        }
    }

    public void payRent(String payerName, String squareName ) {
        Player payer = getPlayerFromName(payerName);
        Square location = board.getSquareFromName(squareName);
        Player payee = location.getHotelOwner();
        if (payee != null) {
            // Check if owner owns more than one hotel in hotel group
            ArrayList<Hotel> payeeHotelGroup = board.getHotelGroup(squareName);
            int counterHotelsOwnedPayee = 0;
            int counterHotelsOwnedPayer = 0;
            for (int i = 0; i < payeeHotelGroup.size(); i++) {
                Player owner = payeeHotelGroup.get(i).getOwner();
                if (owner == payer) {
                    counterHotelsOwnedPayer += 1;
                } else if (owner == payee) {
                    counterHotelsOwnedPayee += 1;
                }
            }
            // hotel gives standard fee
            int rent = 0;
            Hotel hotel = location.getHotel();
            rent += hotel.getOvernightFee();
            // Double fee if payee owns all hotels in group
            if (counterHotelsOwnedPayee == 3) {
                rent *= 2;
            }
            // Halve fee if guest owns one or more hotels in same group
            if (counterHotelsOwnedPayer > 0) {
                rent /= 2;
            }
            // Charge rent
            payer.giveMoneyToPlayer(rent,payee);
            this.justPayed = true;
            canRollPass = true;
            setChanged();
            notifyObservers(payerName+" has paid £"+rent+" rent to "+payee.getName());
        }
    }

    public boolean canAffordHotelUpgrade(String playerName, String squareName) {
        Player player = getPlayerFromName(playerName);
        Square location = board.getSquareFromName(squareName);
        Hotel hotel = location.getHotel();
        return player.getBalance() >= hotel.getUpgradeFee();
    }

    public boolean upgradeHotel(String playerName, String squareName) {
        /** @pre. Playername is valid, squarename is valid.
         * @post. If the player was able to upgrade the hotel
         * star rating increased by 1, player balance decreased by upgrade fee.
         * If player wasn't able to upgrade the hotel then their balance remains the same
         * and the hotel rating remains the same.
         */
        assert(this.getPlayerFromName(playerName) != null) : "Error: player could not be found";
        assert(board.getSquareFromName(squareName) != null) : "Error: square could not be
found";

        Player player = getPlayerFromName(playerName);
        Square location = board.getSquareFromName(squareName);
        Hotel hotel = location.getHotel();
        int beforeRating = hotel.getStarRating();
        int beforeBalance = player.getBalance();
        boolean upgradeSuccess = false;
        // Check player is owner of hotel
        if (hotel.getOwner() == player) {
            // Check owner has enough money
            if (player.getBalance() >= hotel.getUpgradeFee()) {
                if (hotel.increaseStarRating()) {
                    player.chargeMoney(hotel.getUpgradeFee());
                    setChanged();
                    notifyObservers(playerName+" has upgraded "+location.getName()+" which is
now "+location.getHotelRating()+" stars.");
                    upgradeSuccess = true;
                }
                else {
                    setChanged();
                    notifyObservers("Cannot upgrade hotel because it is already at
"+Hotel.MAXRATING+" stars.");
                }
            } else {
                // Don't have enough money to buy
                setChanged();
                notifyObservers("Not enough money to upgrade hotel.");
            }
        } else {
            setChanged();
            notifyObservers("Can't upgrade because you don't own the hotel");
        }
        // Check rating gone up
        assert(hotel.getStarRating() == (beforeRating+1) || !upgradeSuccess) : "Error: After
upgrade rating has not increased by 1.";
        // Check balance gone down
        assert(player.getBalance() == (beforeBalance - hotel.getUpgradeFee()) || !
upgradeSuccess) : "Error: Player balance has not deducted upgrade fee amount correctly.";

        // Check balance is the same and rating the same since upgrade has failed
        assert(hotel.getStarRating() == beforeRating || upgradeSuccess) : "Error: Star rating
should be the same as before attempted upgrade.";
        assert(player.getBalance() == beforeBalance || upgradeSuccess) : "Error: Balance should
be the same as before attempted upgrade";

        return upgradeSuccess;
    }

    public ArrayList<String> getHotelsOwnedByPlayer(String playerName) {
        ArrayList<String> hotels = new ArrayList<String>();
        Player player = getPlayerFromName(playerName);
        for (int i = 0; i < this.board.getNumberOfSquares(); i++) {
            String hotelowner = this.board.getHotelOwnerName(i);
            if (hotelowner == playerName) {
                hotels.add(this.board.getSquareName(i));
            }
        }
        return hotels;
    }

    public Color getPlayerColor(String playerName) {
        Player player = getPlayerFromName(playerName);
        return player.getColor();
    }

    public int getColorComponentRed(String playerName) {
        Player player = getPlayerFromName(playerName);
        return player.getColorComponentRed();
    }
    public int getColorComponentBlue(String playerName) {
        Player player = getPlayerFromName(playerName);
        return player.getColorComponentBlue();
    }
    public int getColorComponentGreen(String playerName) {
        Player player = getPlayerFromName(playerName);
        return player.getColorComponentGreen();
    }



    public int getDiceScore() {
        return this.diceScore;
    }

    public void doTurn() {
        Player player = this.getCurrentPlayer();
        Player owner = player.getPosition().getHotelOwner();
        if (owner == player) {
            this.canRollPass = true;
        }
        else if (owner != null) {
            this.canRollPass = false;
        }

    }


}
```

Controller

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

// connects View with the Model, gives commands
// it will store data in Model and update the View
public class Controller implements ActionListener, MouseListener {
    private Model model;
    private View view;

    public Controller(Model model) {
        // Model must be created first and then the controller and then the view
        // we can have multiple controllers and views but only one model
        this.model = model;

    }

    public void setView(View view) {
        // View needs controller to exist, call setView after creating a controller
        this.view = view;
    }

    public void doRollDice() {
        // Called by eventclickhandler from View and tell Model the dice roll
        this.model.rollDice();
    }

    public void doMoveCounter(String playerName, int diceNumber) {
        this.model.moveCounterForwards(playerName, diceNumber);
    }

    public void buyProperty(String playerName, String squareName) {
        this.model.buyProperty(playerName,squareName);
    }

    public void payRent(String payerName, String squareName ) {
        this.model.payRent(payerName, squareName);
    }

    public void upgradeHotel(String playerName, String squareName) {
        this.model.upgradeHotel(playerName, squareName);
    }

    private void doTurn(String playerName, String squareName) {
        this.model.doTurn();
    }



    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        // Gives label on button that was clicked
        String action = actionEvent.getActionCommand();
            try {
                if (action == "roll/pass") {
                this.model.rollPass();
                } else if (action == "buy") {
                    this.model.doBuy();
                } else if (action == "pay") {
                    this.model.doPay();
                } else if (action == "newgame") {
                    this.model.initialiseModel();
                }
            } catch (InterruptedException e) {
                throw new RuntimeException(e);

        }

    }



    @Override
    public void mouseClicked(MouseEvent mouseEvent) {
```

```java
        // Cheat mode clicking the JPanel squares
        int squareindex = Integer.parseInt(mouseEvent.getComponent().getName());
        model.cheatGoTo(squareindex);
    }

    @Override
    public void mousePressed(MouseEvent mouseEvent) {

    }

    @Override
    public void mouseReleased(MouseEvent mouseEvent) {

    }

    @Override
    public void mouseEntered(MouseEvent mouseEvent) {

    }

    @Override
    public void mouseExited(MouseEvent mouseEvent) {

    }
}
```

View

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseListener;
import java.io.File;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JPanel;
import javax.swing.border.LineBorder;

// View observes Model for state changes
public class View implements Observer {
    boolean initialised;
    JFrame frame;
    JPanel outerPanel;
    JPanel boardPanel;
    JPanel player1Panel;
    JPanel player2Panel;
    Container container;
    int squareSize = 150;
    int propertiesPerSide = 9;
    ArrayList<JPanel> squares;
    int padding = 10;
    ImageIcon starIcon;
    Model model;
    Controller controller;
    public View(Model model, Controller controller) throws InterruptedException,
InvocationTargetException {
        this.model = model;
        this.controller = controller;
        // View observes Model
        model.addObserver(this);
        this.squares = new ArrayList<JPanel>();

        // Use threads
        SwingUtilities.invokeAndWait(new Runnable() {
            @Override
            public void run() {
                createGUI();
                // Update all the squares so that they initially will show all their labels and icons
                for (int i = 0; i < 40; i++) {
                    updateSquare(i);
                }
            }

             });

    }

    private void updateButtons() {
        // Enable or disable buttons to match the model using variables in the Model (getCanPay()
etc. returns a boolean)
        boardPanel.getComponent(2).setEnabled(model.getCanRollPass());
        boardPanel.getComponent(3).setEnabled(model.getCanBuy());
        boardPanel.getComponent(4).setEnabled(model.getCanPay());
    }

    private void updateSquare(int squareIndex) {
        JPanel square = this.squares.get(squareIndex);
        int price = model.getHotelPrice(squareIndex);
        if (price > 0) {
            // If there is a hotel on the square

((JLabel)square.getComponent(0)).setText("£"+Integer.toString(model.getHotelPrice(squareInd
ex)));
        }
        ((JLabel)square.getComponent(1)).setText(model.getSquareName(squareIndex));

        String owner = model.getHotelOwnerName(squareIndex);
        if (owner != null) {
            square.setBackground(model.getPlayerColor(owner));

((JLabel)square.getComponent(3)).setText(Integer.toString(model.getHotelRating(squareIndex))
);
            square.getComponent(3).setVisible(owner != null);
        } else {
            square.setBackground(Color.white);
            if (square.getComponents().length > 3) {
                // Get star label
                square.getComponent(3).setVisible(owner != null);
            }
        }



        // Clear contents of previous label
        JLabel iconLabel = ((JLabel)square.getComponent(2));
        iconLabel.removeAll();
        for (String playername: this.model.getPlayerNamesOnSquare(squareIndex)) {
            ImageIcon playerCounter = this.model.getSmallImageIcon(playername);
            iconLabel.add(new JLabel(playerCounter));
        }
        square.repaint();



    }

    private void updatePlayerInfoPanel(int playerIndex) {
        JPanel playerPanel;
        if (playerIndex == 0) {
            // Player 1 panel
            playerPanel = this.player1Panel;
        } else {
            // Player 2 panel
            playerPanel = this.player2Panel;
        }

        String playerName = this.model.getPlayerName(playerIndex);
        ((JLabel)playerPanel.getComponent(0)).setText("Name: "+playerName);
        ((JLabel)playerPanel.getComponent(1)).setText("Bank:
£"+this.model.getPlayerBalance(playerName));
        // Sort hotels owned into groups and seperate with <br>
        String hotelsOwned = new String("Hotels owned: ");
        String previousGroup = new String("_");
        // Get hotels owned by player
        for (String hotelName: model.getHotelsOwnedByPlayer(playerName)) {
            if (!hotelName.contains(previousGroup)) {
                // Seperate groups with breakline
                hotelsOwned += "<br>";
                previousGroup = hotelName.substring(0,1);
            }
            hotelsOwned += hotelName;
        }
        ((JLabel)playerPanel.getComponent(2)).setText("<html>"+hotelsOwned+"</html>");

        ImageIcon icon1 = this.model.getPlayerImageIcon(playerName);
        ((JLabel)playerPanel.getComponent(3)).setIcon(icon1);

    }

    private void createPlayerInfoPanels() {
        // This sets up the player info panels initially, but we will have to update
        // the panels when information updates in the model, so we'll use an Observer/Observable
for that
        int rowHeight = 30;
        this.player1Panel.setBackground(model.getPlayerColor("player1"));
        JLabel nameLabel = new JLabel("Name: Player1");
        nameLabel.setBounds(padding,padding,400-padding,rowHeight);
        nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,20));
        this.player1Panel.add(nameLabel);

        JLabel bankLabel = new JLabel("Bank: ");
        bankLabel.setText("Bank: £"+this.model.getPlayerBalance("player1"));
        bankLabel.setBounds(padding,padding+(rowHeight+padding),400-padding,rowHeight);
        bankLabel.setFont(new Font(Font.SERIF,Font.BOLD,20));
        this.player1Panel.add(bankLabel);

        // Sort hotels owned into groups and seperate with <br>
        String hotelsOwned = new String("Hotels owned: ");
        String previousGroup = new String("_");
        // Get hotels owned by player
        for (String hotelName: model.getHotelsOwnedByPlayer("player1")) {
            if (!hotelName.contains(previousGroup)) {
                // Seperate groups with breakline
                hotelsOwned += "<br>";
                previousGroup = hotelName.substring(0,1);
            }
            hotelsOwned += hotelName;
        }

        JLabel hotelsOwnedLabel = new JLabel("<html>"+hotelsOwned+"</html>");
        hotelsOwnedLabel.setBounds(padding,padding+(rowHeight+padding)*2,400-
padding,rowHeight*8);
        hotelsOwnedLabel.setFont(new Font(Font.SERIF,Font.BOLD,20));
        player1Panel.add(hotelsOwnedLabel);

        ImageIcon icon1 = this.model.getPlayerImageIcon("player1");
        JLabel iconLabel = new JLabel(icon1);
        iconLabel.setBounds(300-padding,padding,rowHeight*2,rowHeight*2);
        player1Panel.add(iconLabel);

        ///////////////////////////////////////////////////////////////// Player 2
        this.player2Panel.setBackground(model.getPlayerColor("player2"));
        JLabel nameLabel2 = new JLabel("Name: Player2");
        nameLabel2.setBounds(padding,padding,400-padding,rowHeight);
        nameLabel2.setFont(new Font(Font.SERIF,Font.BOLD,20));
        this.player2Panel.add(nameLabel2);

        JLabel bankLabel2 = new JLabel("Bank: ");
        bankLabel2.setText("Bank: £"+this.model.getPlayerBalance("player2"));
        bankLabel2.setBounds(padding,padding+(rowHeight+padding),400-padding,rowHeight);
        bankLabel2.setFont(new Font(Font.SERIF,Font.BOLD,20));
        this.player2Panel.add(bankLabel2);

        // Sort hotels owned into groups and seperate with <br>
        String hotelsOwned2 = new String("Hotels owned: ");
        String previousGroup2 = new String("_");
        // Get hotels owned by player
        for (String hotelName: model.getHotelsOwnedByPlayer("player2")) {
            if (!hotelName.contains(previousGroup2)) {
                // Seperate groups with breakline
                hotelsOwned2 += "<br>";
                previousGroup2 = hotelName.substring(0,1);
            }
            hotelsOwned2 += hotelName;
        }

        JLabel hotelsOwnedLabel2 = new JLabel("<html>"+hotelsOwned2+"</html>");
        hotelsOwnedLabel2.setBounds(padding,padding+(rowHeight+padding)*2,400-
padding,rowHeight*8);
        hotelsOwnedLabel2.setFont(new Font(Font.SERIF,Font.BOLD,20));
        player2Panel.add(hotelsOwnedLabel2);

        ImageIcon icon2 = this.model.getPlayerImageIcon("player2");
        JLabel iconLabel2 = new JLabel(icon2);
        iconLabel2.setBounds(300-padding,padding,rowHeight*2,rowHeight*2);
        player2Panel.add(iconLabel2);
    }
    private void createButtonsAndLabels() {
        // Add label to display who's turn it is
        JLabel playerTurnLabel = new JLabel("Player 1 turn",SwingConstants.CENTER);
        playerTurnLabel.setBounds(squareSize*3/2,squareSize,squareSize*7/2,squareSize);
        playerTurnLabel.setFont(new Font(Font.SERIF,Font.BOLD,20));
        boardPanel.add(playerTurnLabel);

        // Add label to show messages from the model being updated
        JLabel userMessageLabel = new JLabel("You rolled 5",SwingConstants.CENTER);
        userMessageLabel.setBounds(squareSize*3/2,squareSize*5/3,squareSize*7/2,squareSize);
        userMessageLabel.setFont(new Font(Font.SERIF,Font.BOLD,20));
        boardPanel.add(userMessageLabel);

        // Option buttons
        JButton rollDiceButton = new JButton("Roll/pass");

rollDiceButton.setBounds(squareSize*3/2,squareSize*9/2+padding,squareSize,squareSize/2);
        rollDiceButton.setFont(new Font(Font.SERIF,Font.BOLD,20));
        rollDiceButton.setActionCommand("roll/pass");
        rollDiceButton.addActionListener(this.controller);
        boardPanel.add(rollDiceButton);

        JButton buyButton = new JButton("Buy");
        buyButton.setBounds(squareSize*11/4,squareSize*9/2+padding,squareSize,squareSize/2);
        buyButton.setFont(new Font(Font.SERIF,Font.BOLD,20));
        buyButton.setActionCommand("buy");
        buyButton.addActionListener(this.controller);
        boardPanel.add(buyButton);

        JButton payButton = new JButton("Pay");
        payButton.setBounds(squareSize*4,squareSize*9/2+padding,squareSize,squareSize/2);
        payButton.setFont(new Font(Font.SERIF,Font.BOLD,20));
        payButton.setActionCommand("pay");
        payButton.addActionListener(this.controller);
        boardPanel.add(payButton);

        this.updateButtons();

    }

    private void createSquares() {
        // Define smaller square size
        int propertyWidth = squareSize / 2;

        ///////////// All positions on board are calculated on basis of square size
        // Padding is a spacing used at the top and left hand side of board
        // GO square
        JPanel panelse = new JPanel();
        panelse.setLayout(null);

        // Set index number, the squares array changes dynamically so it increases
        // Setname sets index to be used when handling cheatmode requests
        panelse.setName(Integer.toString(this.squares.size()));
        // Configure for controller to handle mouseclicks on this panel/square
        panelse.addMouseListener((MouseListener) this.controller);

        // propertywidth is half of squareSize (it's the smaller squares)

panelse.setBounds(padding+squareSize+propertiesPerSide*propertyWidth,padding+squareSiz
e+propertiesPerSide*propertyWidth,squareSize,squareSize);
```

```
        panelse.setBackground(Color.white);
        panelse.setBorder(new LineBorder(Color.black,1));
        // Price label
        JLabel priceLabel = new JLabel("",SwingConstants.CENTER);
        priceLabel.setBounds(0,(squareSize*2)/3,squareSize,squareSize/3);
        panelse.add(priceLabel);
        // Name label
        JLabel nameLabel = new JLabel("",SwingConstants.CENTER);
        nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,70));
        nameLabel.setBounds(0,0,squareSize,squareSize);
        panelse.add(nameLabel);
        // Counter label
        JLabel counterLabel = new JLabel("",SwingConstants.CENTER);
        // Create a horizontal boxlayout to put 2 counters next to each other
        counterLabel.setLayout(new BoxLayout(counterLabel,BoxLayout.X_AXIS));
        counterLabel.setBounds(padding,0,squareSize,squareSize/3);
        panelse.add(counterLabel);
        this.squares.add(panelse);

        // This is the bottom row
        for (int i = propertiesPerSide-1; i >= 0; i--) {
            JPanel newpanel = new JPanel();

            // Set index number, the squares array changes dynamically so it increases
            newpanel.setName(Integer.toString(this.squares.size()));
            newpanel.addMouseListener((MouseListener) this.controller);

            newpanel.setLayout(null);
            // x,y,width,height

    newpanel.setBounds(padding+squareSize+i*propertyWidth,padding+squareSize+propertiesPe
rSide*propertyWidth,propertyWidth,squareSize);
            newpanel.setBorder(new LineBorder(Color.black,1));
            newpanel.setBackground(Color.white);
            this.squares.add(newpanel);
            // Price label
            priceLabel = new JLabel("",SwingConstants.CENTER);
            priceLabel.setBounds(0,(squareSize*2)/3,propertyWidth,squareSize/3);
            newpanel.add(priceLabel);
            // Name label
            nameLabel = new JLabel("",SwingConstants.CENTER);
            nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,30));
            nameLabel.setBounds(0,0,propertyWidth,propertyWidth/2);
            newpanel.add(nameLabel);
            // Counter label
            counterLabel = new JLabel("",SwingConstants.CENTER);
            // Create a horizontal boxlayout to put 2 counters next to each other
            counterLabel.setLayout(new BoxLayout(counterLabel,BoxLayout.X_AXIS));
            counterLabel.setBounds(padding/2,propertyWidth/2,propertyWidth,propertyWidth/2);
            newpanel.add(counterLabel);
            // Star rating
            JLabel starLabel = new JLabel("",this.starIcon,SwingConstants.CENTER);
            starLabel.setFont(new Font(Font.SERIF,Font.BOLD,15));
            starLabel.setText("0");
            starLabel.setBounds(0,propertyWidth,propertyWidth,propertyWidth/2);
            newpanel.add(starLabel);
            // Set starlabel to invisible and we can make it visible later
            starLabel.setVisible(false);

        }
        JPanel panelsw = new JPanel();
        panelsw.setLayout(null);

        // Set index number, the squares array changes dynamically so it increases
        panelsw.setName(Integer.toString(this.squares.size()));
        panelsw.addMouseListener((MouseListener) this.controller);


panelsw.setBounds(padding,padding+squareSize+propertiesPerSide*propertyWidth,squareSize
,squareSize);
        panelsw.setBorder(new LineBorder(Color.black,1));
        panelsw.setBackground(Color.white);
        this.squares.add(panelsw);
        priceLabel = new JLabel("",SwingConstants.CENTER);
        priceLabel.setBounds(0,(squareSize*2)/3,squareSize,squareSize/3);
        panelsw.add(priceLabel);
        nameLabel = new JLabel("",SwingConstants.CENTER);
        nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,30));
        nameLabel.setBounds(0,0,squareSize,squareSize/2);
        panelsw.add(nameLabel);
        // Counterlabel
        counterLabel = new JLabel("",SwingConstants.CENTER);
        // Create a horizontal boxlayout to put 2 counters next to each other
        counterLabel.setLayout(new BoxLayout(counterLabel,BoxLayout.X_AXIS));
        counterLabel.setBounds(padding,0,squareSize,squareSize/3);
        panelsw.add(counterLabel);

        // This is the left row
        for (int j = propertiesPerSide-1; j >= 0; j--) {
            JPanel newpanel = new JPanel();
            newpanel.setLayout(null);

            // Set index number, the squares array changes dynamically so it increases
            newpanel.setName(Integer.toString(this.squares.size()));
            newpanel.addMouseListener((MouseListener) this.controller);

            // x,y,width,height

newpanel.setBounds(padding,padding+squareSize+j*propertyWidth,squareSize,propertyWidth)
;
            newpanel.setBorder(new LineBorder(Color.black,1));
            newpanel.setBackground(Color.white);
            this.squares.add(newpanel);
            priceLabel = new JLabel("",SwingConstants.LEFT);
            priceLabel.setBounds(squareSize/9,propertyWidth/3,squareSize/2,propertyWidth/3);
            newpanel.add(priceLabel);
            nameLabel = new JLabel("",SwingConstants.RIGHT);
            nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,30));
            nameLabel.setBounds(0,propertyWidth/3,squareSize-padding,propertyWidth/3);
            newpanel.add(nameLabel);
            // Counter label
            counterLabel = new JLabel("",SwingConstants.CENTER);
            // Create a horizontal boxlayout to put 2 counters next to each other
            counterLabel.setLayout(new BoxLayout(counterLabel,BoxLayout.X_AXIS));
            counterLabel.setBounds(padding,padding/2,propertyWidth,propertyWidth/3);
            newpanel.add(counterLabel);
            // Star rating
            JLabel starLabel = new JLabel("",this.starIcon,SwingConstants.CENTER);
            starLabel.setFont(new Font(Font.SERIF,Font.BOLD,15));
            starLabel.setText("0");
            starLabel.setBounds(padding,propertyWidth*2/3,propertyWidth,propertyWidth/3);
            newpanel.add(starLabel);
            starLabel.setVisible(false);
        }
        JPanel panelnw = new JPanel();
        panelnw.setLayout(null);

        // Set index number, the squares array changes dynamically so it increases
        panelnw.setName(Integer.toString(this.squares.size()));
        panelnw.addMouseListener((MouseListener) this.controller);

        panelnw.setBounds(padding,padding,squareSize,squareSize);
        panelnw.setBorder(new LineBorder(Color.black,1));
        panelnw.setBackground(Color.white);
        this.squares.add(panelnw);
        priceLabel = new JLabel("",SwingConstants.CENTER);
        priceLabel.setBounds(0,(squareSize*2)/3,squareSize,squareSize/3);
        panelnw.add(priceLabel);
        nameLabel = new JLabel("",SwingConstants.CENTER);
        nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,30));
        nameLabel.setBounds(0,0,squareSize,squareSize/2);
        panelnw.add(nameLabel);
        // Counterlabel
        counterLabel = new JLabel("",SwingConstants.CENTER);
        // Create a horizontal boxlayout to put 2 counters next to each other
        counterLabel.setLayout(new BoxLayout(counterLabel,BoxLayout.X_AXIS));
        counterLabel.setBounds(padding,0,squareSize,squareSize/3);
        panelnw.add(counterLabel);

        // This is the top row
        for (int j = 0; j < propertiesPerSide; j++) {
            JPanel newpanel = new JPanel();
            newpanel.setLayout(null);

            // Set index number, the squares array changes dynamically so it increases
            newpanel.setName(Integer.toString(this.squares.size()));
            newpanel.addMouseListener((MouseListener) this.controller);

            // x,y,width,height

newpanel.setBounds(padding+squareSize+j*propertyWidth,padding,propertyWidth,squareSize)
;
            newpanel.setBorder(new LineBorder(Color.black,1));
            newpanel.setBackground(Color.white);
            this.squares.add(newpanel);
            priceLabel = new JLabel("",SwingConstants.CENTER);
            priceLabel.setBounds(0,(squareSize*2)/3,propertyWidth,squareSize/3);
            newpanel.add(priceLabel);
            nameLabel = new JLabel("",SwingConstants.CENTER);
            nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,30));
            nameLabel.setBounds(0,0,propertyWidth,propertyWidth/2);
            newpanel.add(nameLabel);
            // Counter label
            counterLabel = new JLabel("",SwingConstants.CENTER);
            // Create a horizontal boxlayout to put 2 counters next to each other
            counterLabel.setLayout(new BoxLayout(counterLabel,BoxLayout.X_AXIS));
            counterLabel.setBounds(padding/2,propertyWidth/2,propertyWidth,propertyWidth/2);
            newpanel.add(counterLabel);
            // Star rating
            JLabel starLabel = new JLabel("",this.starIcon,SwingConstants.CENTER);
            starLabel.setFont(new Font(Font.SERIF,Font.BOLD,15));
            starLabel.setText("0");
            starLabel.setBounds(0,propertyWidth,propertyWidth,propertyWidth/2);
            newpanel.add(starLabel);
            starLabel.setVisible(false);

        }

        JPanel panelne = new JPanel();
        panelne.setLayout(null);

        // Set index number, the squares array changes dynamically so it increases
        panelne.setName(Integer.toString(this.squares.size()));
        panelne.addMouseListener((MouseListener) this.controller);


panelne.setBounds(padding+squareSize+propertiesPerSide*propertyWidth,padding,squareSize
,squareSize);
        panelne.setBorder(new LineBorder(Color.black,1));
        panelne.setBackground(Color.white);
        this.squares.add(panelne);
        priceLabel = new JLabel("",SwingConstants.CENTER);
        priceLabel.setBounds(0,(squareSize*2)/3,squareSize,squareSize/3);
        panelne.add(priceLabel);
        nameLabel = new JLabel("",SwingConstants.CENTER);
        nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,30));
        nameLabel.setBounds(0,0,squareSize,squareSize/2);
        panelne.add(nameLabel);
        // Counterlabel
        counterLabel = new JLabel("",SwingConstants.CENTER);
        // Create a horizontal boxlayout to put 2 counters next to each other
        counterLabel.setLayout(new BoxLayout(counterLabel,BoxLayout.X_AXIS));
        counterLabel.setBounds(padding,0,squareSize,squareSize/3);
        panelne.add(counterLabel);

        // This is the right row
        for (int j = 0; j < propertiesPerSide; j++) {
            JPanel newpanel = new JPanel();
            newpanel.setLayout(null);

            // Set index number, the squares array changes dynamically so it increases
            newpanel.setName(Integer.toString(this.squares.size()));
            newpanel.addMouseListener((MouseListener) this.controller);

            // x,y,width,height

newpanel.setBounds(padding+squareSize+propertiesPerSide*propertyWidth,padding+squareSi
ze+j*propertyWidth,squareSize,propertyWidth);
            newpanel.setBorder(new LineBorder(Color.black,1));
            newpanel.setBackground(Color.white);
            this.squares.add(newpanel);
            priceLabel = new JLabel("",SwingConstants.RIGHT);
            // X is 2 thirds

priceLabel.setBounds(squareSize*2/3,propertyWidth/3,propertyWidth/2,propertyWidth/3);
            newpanel.add(priceLabel);
            nameLabel = new JLabel("",SwingConstants.LEFT);
            nameLabel.setFont(new Font(Font.SERIF,Font.BOLD,30));
            nameLabel.setBounds(padding,propertyWidth/3,squareSize,propertyWidth/3);
            newpanel.add(nameLabel);
            // Counter label
            counterLabel = new JLabel("",SwingConstants.CENTER);
            // Create a horizontal boxlayout to put 2 counters next to each other
            counterLabel.setLayout(new BoxLayout(counterLabel,BoxLayout.X_AXIS));
            counterLabel.setBounds(squareSize/2,padding/2,propertyWidth,propertyWidth/3);
            newpanel.add(counterLabel);
            // Star rating
            JLabel starLabel = new JLabel("",this.starIcon,SwingConstants.CENTER);
            starLabel.setFont(new Font(Font.SERIF,Font.BOLD,15));
            starLabel.setText("0");
            starLabel.setBounds(squareSize/2,propertyWidth*2/3,propertyWidth,propertyWidth/3);
            newpanel.add(starLabel);
            starLabel.setVisible(false);
        }

        // Add squares onto boardPanel
        for (int i = 0; i < this.squares.size(); i++) {
            this.boardPanel.add(this.squares.get(i));
        }

    }



    public void createGUI() {
        // Create frame
        this.frame = new JFrame("Hotels");
        this.frame.setSize(1400,1050);
        this.frame.setVisible(true);
        this.frame.setLayout(null);
        this.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create outer panel
        this.container = new Container();
        this.container = this.frame.getContentPane();

        this.outerPanel = new JPanel();
        this.outerPanel.setSize(new Dimension(1400,1000));
        this.frame.setContentPane(this.outerPanel);
        this.outerPanel.setLayout(null);

        this.boardPanel = new JPanel();
        this.boardPanel.setLayout(null);
        this.boardPanel.setBounds(0,0,1000,1000);
        this.boardPanel.setBackground(Color.lightGray);
        this.outerPanel.add(this.boardPanel);



        this.player1Panel = new JPanel(null);
        this.player1Panel.setBounds(1000,0,400,500);
        this.player1Panel.setBorder(new LineBorder(Color.black,1));
        this.outerPanel.add(this.player1Panel);
        this.player2Panel = new JPanel(null);
        this.player2Panel.setBounds(1000,500,400,500);
        this.player2Panel.setBorder(new LineBorder(Color.black,1));
        this.outerPanel.add(player2Panel);

        this.starIcon = new ImageIcon(createImageIcon("resources/star1.png","Star
rating").getImage().getScaledInstance(20,20,Image.SCALE_DEFAULT));

        createButtonsAndLabels();
        createSquares();
        createPlayerInfoPanels();
        updateTurn();
        this.initialised = true;
```

```java
}

/** Returns an ImageIcon, or null if the path was invalid. */
public ImageIcon createImageIcon(String path, String description) {
    File file = new File("./");
    try {
        System.out.println(file.getCanonicalPath()+"/"+path);
        String pathToIcon = new String(file.getCanonicalPath()+"/"+path);
        return new ImageIcon(pathToIcon, description);

    } catch (IOException e) {
        System.err.println("Couldn't find file: " + path);
    }
    return null;
}

private void updateTurn() {
    String playerName = model.getCurrentPlayerName();
    ((JLabel)boardPanel.getComponent(0)).setText(playerName+"'s turn.");
    ImageIcon icon = model.getPlayerImageIcon(playerName);
    ((JLabel)boardPanel.getComponent(0)).setIcon(icon);

}

private void updateMessageLabel(String message) {
    ((JLabel)boardPanel.getComponent(1)).setText(message);
}

/**
 * Implemented method from Observer interface updates GUI to reflect state of model
 * @param observable : this is the Model
 * @param o : this is a string of what change has happened
 */
@Override
public void update(Observable observable, Object o) {
    if (model.isGameOver()) {
        endgameScreen();
    } else {
        if (!initialised){
            this.frame.dispose();
            createGUI();
        }
        // Object o is instruction to player what has happened
        String message = (String) o;
        updateMessageLabel(message);
        // Update every square getting new information from Model
        for (int i = 0; i < this.squares.size(); i++) {
            updateSquare(i);
        }
        // Update player info panels each time there is a change
        updatePlayerInfoPanel(0);
        updatePlayerInfoPanel(1);
        //
        this.updateTurn();
        this.updateButtons();
    }

}
```

```
    private void endgameScreen() {
        initialised = false;
        outerPanel.removeAll();
        String winnerName = model.getWinnerName();
        Color winnerColor = model.getPlayerColor(winnerName);
        ImageIcon winnerIcon = model.getPlayerImageIcon(winnerName);
        winnerIcon = new
ImageIcon(winnerIcon.getImage().getScaledInstance(256,256,Image.SCALE_DEFAULT));
        String winnerMessage = (String) "<html>" + winnerName + " has won the game!!!!!
</html>";
        JLabel winLabel = new JLabel(winnerMessage, SwingConstants.CENTER);
        winLabel.setIcon(winnerIcon);
        winLabel.setFont(new Font(Font.SERIF, Font.BOLD, 90));
        winLabel.setBounds(0,0,outerPanel.getWidth(),outerPanel.getHeight());
        // New game button
        JButton newgameButton = new JButton("New game");
        newgameButton.setBounds(this.outerPanel.getWidth()/2,(this.outerPanel.getHeight()/2)-
newgameButton.getWidth(),this.outerPanel.getWidth()/8,this.outerPanel.getHeight()/8);
        newgameButton.setFont(new Font(Font.SERIF, Font.BOLD, 20));
        newgameButton.setActionCommand("newgame");
        newgameButton.addActionListener(this.controller);
        outerPanel.setBackground(winnerColor);
        outerPanel.add(winLabel);
        outerPanel.add(newgameButton, SwingConstants.CENTER);
    }
}
```

CLIMain

```
import java.lang.reflect.InvocationTargetException;
import java.util.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class CLIMain implements Observer {
    static Model model;
    BufferedReader reader;
    public static final String RESET = "\033[0m";
    public static final String ALERTCOLOR = "\033[38;2;255;0;255m";

    public static void main(String[] args) throws InterruptedException, InvocationTargetException
{
        System.out.println("---------------CLI Hotels---------------");
        CLIMain cli = null;
        try {
            cli = new CLIMain();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        cli.playGame();

    }

    private String getPlayerColorCode(String playername) {
        int red = model.getColorComponentRed(playername);
        int blue = model.getColorComponentBlue(playername);
        int green = model.getColorComponentGreen(playername);
        // ANSI escape sequence format - 38 is foreground-48 is background, 2 means it is
```

```
static/solid color
        String rgbformat = "\033[48;2;" + red + ";" + green + ";" + blue + "m";
        return rgbformat;
    }

    public String getUserCommand() {
        ArrayList<String> options = new ArrayList<String>();
        if (model.getCanRollPass()) {
            options.add("roll/pass");
        }
        if (model.getCanBuy()) {
            options.add("buy");
        }
        if (model.getCanPay()) {
            options.add("pay");
        }
        if (model.getCheatMode()) {
            options.add("cheat");
        }
        int optionchoice = -1;
        String playername = model.getCurrentPlayerName();
        String playerColorCode = this.getPlayerColorCode(playername);
        while (optionchoice < 1 || optionchoice > options.size()) {
            System.out.println("Please select an option " + playerColorCode + playername +
RESET + ":");
            for (int i = 0; i < options.size(); i++) {
                System.out.println("[" + (i + 1) + "] " + options.get(i));

            }
            try {
                String getline = this.reader.readLine();
                optionchoice = new Integer(getline);
            } catch (IOException e) {
                System.out.println("Invalid input, please try again.");
            }
        }
        return options.get(optionchoice - 1);

    }

    public void endGame() {
        System.out.println(this.model.getWinnerName()+" has won the game!!!!!!!!");
    }

    public void playGame() {
        while (!this.model.isGameOver()) {
            this.printBoard();
            this.printPlayersInfo();
            String command = getUserCommand();
            this.processCommand(command);
        }
        this.endGame();
    }

    public void processCommand(String command) {
        if (command == "roll/pass") {
            try {
                model.rollPass();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
        else if (command == "buy") {
            model.doBuy();
        }
        else if (command == "pay") {
            model.doPay();
        }
        else if (command == "cheat") {
            int steps = cheatMove();
            int curPlayerPosition = model.getCurrentPlayerPosition();
            int out = (curPlayerPosition + steps) % model.getMaxSquares();
            model.cheatGoTo(out);
        }
    }

    private int cheatMove() {
        int output = -1;
        while (output < 1 || output > 12) {
            try {
                System.out.println("How many squares do you want to move forwards (between 1-
12)?: ");
                String cheati = this.reader.readLine();
                output = new Integer(cheati);
            } catch (IOException e) {
            } catch (NumberFormatException e) {
                System.out.println("You must enter a number.");

            } finally {
                if (output < 1 || output > 12) {
                    System.out.println("Invalid option. Try again.");
                }

            }
        }
        return output;
    }

    public CLIMain() throws IOException {
        // Constructor
        // Only uses the model
        this.model = new Model(true);
        this.reader = new BufferedReader((new InputStreamReader(System.in)));
        this.model.addObserver(this);
    }



    public void printPlayersInfo() {
        for (int i = 0; i < model.getMaxPlayers(); i++) {
            String playername = model.getPlayerName(i);
            int playermoney = model.getBalance(i);
            ArrayList<String> hotellist = model.getHotelsOwnedByPlayer(playername);
            String playerColorString = this.getPlayerColorCode(playername);
            System.out.println(playerColorString + "Player: " + playername + "\n" + "Balance: £" +
Integer.toString(playermoney));
            for (int j = 0; j < hotellist.size(); j++) {
                System.out.print(hotellist.get(j) + " ");
                // Keeps 10 hotels on one line.
                if ((j + 1) % 10 == 0) {
                    System.out.println();
                }
            }
            // RESET color
            System.out.println(RESET + "-----------------------------");
        }
    }

    public void printBoard() {
        for (int i = 0; i < model.getMaxSquares(); i++) {
            String squarename = model.getSquareName(i);
            int price = model.getHotelPrice(i);
            String owner = model.getHotelOwnerName(i);
            int starrating = model.getHotelRating(i);
            ArrayList<String> countersOnSquare = model.getPlayerNamesOnSquare(i);
            for (int j = 0; j < countersOnSquare.size(); j++) {
                countersOnSquare.set(j, this.getPlayerColorCode(countersOnSquare.get(j)) +
countersOnSquare.get(j) + RESET);
            }

            String infostring = "Square " + i + " ";
            infostring += squarename.length() < 1 ? "BLANK" : squarename;
            if (price > 0) {
                infostring += " Hotel price: £" + price;
                if (owner != null) {
                    String ownerColor = getPlayerColorCode(owner);
                    infostring += ownerColor;
                    infostring += " Owned by: " + owner;
                    infostring += " Star rating: " + starrating;
                    infostring += RESET;
                }
            }
            infostring += " Counters on square: " + String.join(", ",countersOnSquare);
            System.out.println(infostring);
        }
    }

    @Override
    public void update(Observable observable, Object o) {
        System.out.println(ALERTCOLOR + (String)o + RESET);
    }


}
```

ModelTesting

```
public class ModelTesting extends Model {

    public ModelTesting(boolean cheatmode) {
        super(cheatmode);
    }

    public Player getPlayer(String playerName) {
        // For testing
        return getPlayerFromName(playerName);
    }

}
```

ModelTest

```
import static org.junit.jupiter.api.Assertions.*;

class ModelTest {

    @org.junit.jupiter.api.Test
    void upgradeHotel() {
        ModelTesting model = new ModelTesting(true);
        // Scenario precondition: Upgrading hotel goes ahead
        // * Player has rolled dice to move to square A3
        // * Player has purchased hotel
        // * Player has enough money to upgrade hotel
        // * The hotel is 0 stars
        // * Player upgrades hotel

        // Setup scenario to be tested
        Player player = model.getPlayer(model.getCurrentPlayerName());
        model.cheatGoTo(4);
        model.doBuy();
        int beforeBalance = player.getBalance();

        // Check preconditions hold/are valid
        // Check player location is A3
        assert(player.getPosition().getName() == "A3") : "Error: Precondition failed. Player
position is not A3";
        // Check hotel owner is player
        assert(player.getPosition().getHotel().getOwner() == player) : "Error: Precondition failed.
Player does not own this hotel.";
        // Check player has enough money to upgrade hotel
        assert(player.getBalance() >= player.getPosition().getHotel().getUpgradeFee()) : "Error:
Precondition failed. Player does not have enough money to upgrade hotel";
        // Check hotel is 0 stars
        assert(player.getPosition().getHotel().getStarRating() == 0) : "Error: Precondition failed.
Hotel is not 0 stars";

        // Upgrade hotel
        model.upgradeHotel(player.getName(), player.getPosition().getName());



        // Postcondition
        // * New rating is 1
        assert(player.getPosition().getHotel().getStarRating() == 1) : "Error: Postcondition failed.
Hotel is not 1 stars";
        // * Player balance is reduced by upgrade fee
        assert(player.getBalance() == (beforeBalance -
player.getPosition().getHotel().getUpgradeFee())) : "Error: Postcondition failed. Player balance
has not deducted upgrade fee.";

    }

    @org.junit.jupiter.api.Test
    void initialisePlayers() {
        // Intialise players is called within the constructor
        // ModelTesting has been added as a subclass of Model in order to access internal private
objects for testing purposes,
        // without interfering with Model
        ModelTesting model = new ModelTesting(true);
        /** @post. 2 players created, both have £2000, both start at position 0 and both players
are
         * in the players list.
         */

        // Check there are 2 players
        Player player1 = model.getPlayer("player1");
        Player player2 = model.getPlayer("player2");
        assertNotEquals(null,player1, "Error: player1 was not created correctly.");
        assertNotEquals(null,player2, "Error: player2 was not created correctly.");

        // Check both players have 2000 pounds
        assertEquals(2000,player1.getBalance(),"Error: Player1 does not start with 2000.");
        assertEquals(2000,player2.getBalance(),"Error: Player2 does not start with 2000.");

        // Check both players in position 0
        assertEquals(0,player1.getPosition().getPosition(), "Error: player1 does not start at index 0
squares.");
        assertEquals(0,player2.getPosition().getPosition(), "Error: player2 does not start at index 0
squares.");
    }

    @org.junit.jupiter.api.Test
    void getCanBuy() {
        // Scenario: Check canbuy is false if not enough money to buy hotel
        // * Current player's location is square A1
        // * Player's balance is 2000
        // * Square isn't owned, ie. !hasOwner()

        // Setup scenario
        ModelTesting modelTester = new ModelTesting(true);
        Player curPlayer = modelTester.getPlayer(modelTester.getCurrentPlayerName());
        modelTester.cheatGoTo(1);

        // Check preconditions hold/are valid
        // Check player location is A1
        assertTrue(curPlayer.getPosition().getName() == "A1", "Error: Precondition failed. Player
position is not A1");
        assertEquals(2000,curPlayer.getBalance(),"Error: Precondition failed. Player does not start
with 2000.");
        // Check hotel owner is player
        assertFalse(curPlayer.getPosition().getHotel().hasOwner(),"Error: Precondition failed.
Player does not own this hotel.");

        // Check canbuy is enabled
        assertTrue(modelTester.getCanBuy(), "Error: Buying property should be enabled.");

    }



    @org.junit.jupiter.api.Test
    void getCheatMode() {
        Model model = new Model(true);
        assertTrue(model.getCheatMode(),"Error: Cheat mode is not enabled correctly.");
        Model modelFalse = new Model(false);
        assertFalse(modelFalse.getCheatMode(), "Error: Cheat mode is not disabled correctly.");
    }

    @org.junit.jupiter.api.Test
    void cheatGoTo() {
        Model model = new Model(true);
        int positionBefore = model.getCurrentPlayerPosition();
        model.cheatGoTo(positionBefore + 13);
        int samePosition = model.getCurrentPlayerPosition();
        assertEquals(positionBefore,samePosition,"Error: CheatGoTo did not fail to move the
player when given a value higher than 12, " +
                "currentplayer does not stay in same place.");

        int newPosition = (positionBefore + 5) % model.getMaxSquares();
        model.cheatGoTo(newPosition);
        assertEquals(newPosition, model.getCurrentPlayerPosition(), "Error: Player new position
from cheat is not +5.");
    }

    @org.junit.jupiter.api.Test
    void getCanPay() {
        ModelTesting modelTester = new ModelTesting(true);

        // Scenario: canpay is false if square is empty
        Player player = modelTester.getPlayer(modelTester.getCurrentPlayerName());
        modelTester.cheatGoTo(2);
        assertFalse(modelTester.getCanPay(), "Error: Pay button should be disabled on empty
square.");


        // Scenario: canpay is false if nobody owns the square
        modelTester.cheatGoTo(3);
        assertEquals(null, player.getPosition().getHotelOwner(), "Error: Test square hotel should
not have an owner.");
        assertFalse(modelTester.getCanPay(), "Error: No owner on hotel, should not be able to
pay.");


        // Scenario: canpay true if player has enough money to upgrade and hotel is not 5 stars
and current player owns this hotel


        // Scenario: canpay is true if square has opposite player owner and a hotel


    }


}
```

Main

```
import java.lang.reflect.InvocationTargetException;

public class Main {
    public static void main(String[] args) throws InterruptedException, InvocationTargetException
{

        // Defines the state (define initial state immediately) and changes to state are changes to
model
        // Maintains list of observers that are prompted to update themselves if a change is made
to the model,
        // the View is one of these Observers, each view is an observer
        Model model = new Model(true);
        // Controller handles requests from View by sending commands to Model
        // Controller uses the model
        Controller controller = new Controller(model);
        // View gets data from model and sends requests to the controller
        // View uses model and controller
        View view = new View(model,controller);

    }

}
```

Github link: https://github.com/BrookesUni/19021102

Note: Please feel free to skip through the video, sorry that it is longer than 5 minutes and that there are some bugs. There are also other video versions in the 'scrap' folder though they show the same content only in different takes/speeds.