JAVA UNO GAME

DONOVAN WEBB 19330681

Table of Contents

1. Overview of the game

Uno is a popular card game which was based on crazy 8's and was brought to market by Mattel. The original game can be played from 2-10 players ages 7+.

The game contains all 108 original Mattel cards which are ok since I am not going to publish it. I have adapted the game to suit the computer by removing the need to call UNO.

- The game can be played against 1-8 robots.
- You can select how many starting cards (the official rules recommend 7).
- Choose whether the computer records your score.
- Fullscreen mode.
- Sounds
- Full-colour GUI with animations
- Winning screen
- Play robot vs robot by setting the human player count to zero.
- supports java 6+
- recommended java 8+

2. How to play

2.1 Uno Rules

Setup: The game is for 2-10 players. Every player starts with seven cards, and they are dealt face down. The rest of the cards are placed in a Draw Pile face down. Next to the pile space should be assigned for a Discard Pile. The top card should be placed in the Discard Pile, and the game begins!

Game Play: The first player is normally the player to the left of the dealer and gameplay follows a clockwise direction. Every player views their cards and tries to match the card in the Discard Pile.

You have to match either by the number, colour, or the symbol. For instance, if the Discard Pile has a blue card that is a 5 you have to place either a blue card or a card with a 5 on it. You can also play a Wild card (black card) which can alter current colour in play.

If the player has no matches or they choose not to play any of their cards even though they might have a match, they must draw a card from the Draw pile. The game then moves on to the next person in turn.

To start the game, you turn over the card at the top of the deck and place it in the discard pile

You may not stack two or more cards together on the same turn.

You can place a wild card on any other card and select the colour you wish to use.

Once a player has no cards remaining, the game round is over, points are scored, and the game begins over again. Normally, everyone tries to be the first one to achieve 500 points, but you can also choose whatever points number to win the game.

Scoring and Winning: There are two ways to play Uno, option A is to not keep score and the winner is whoever wins each round.

Option B keeping score:

When a player no longer has any cards and the game ends, he/she receives points. All opponents' cards are given to the winner and points are counted. The first player to attain 500 points is the winner, but you can also choose whatever points number to win the game.

The scoring for the cards is as follows:

Numbered cards (0-9) – Face value

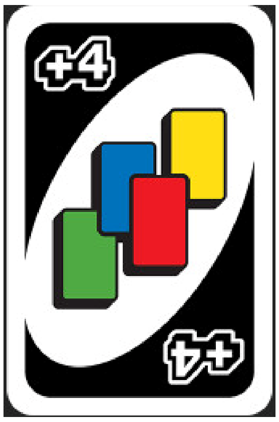
Draw Two/Skip/Reverse – 20 points each.

Wild/Wild Draw Four – 50 points each

Action Cards: Besides the number of cards, several other cards help mix up the game. These are called Action or Symbol cards. There are two of each action cards of each colour in the deck, so there is a total of 24 action cards.

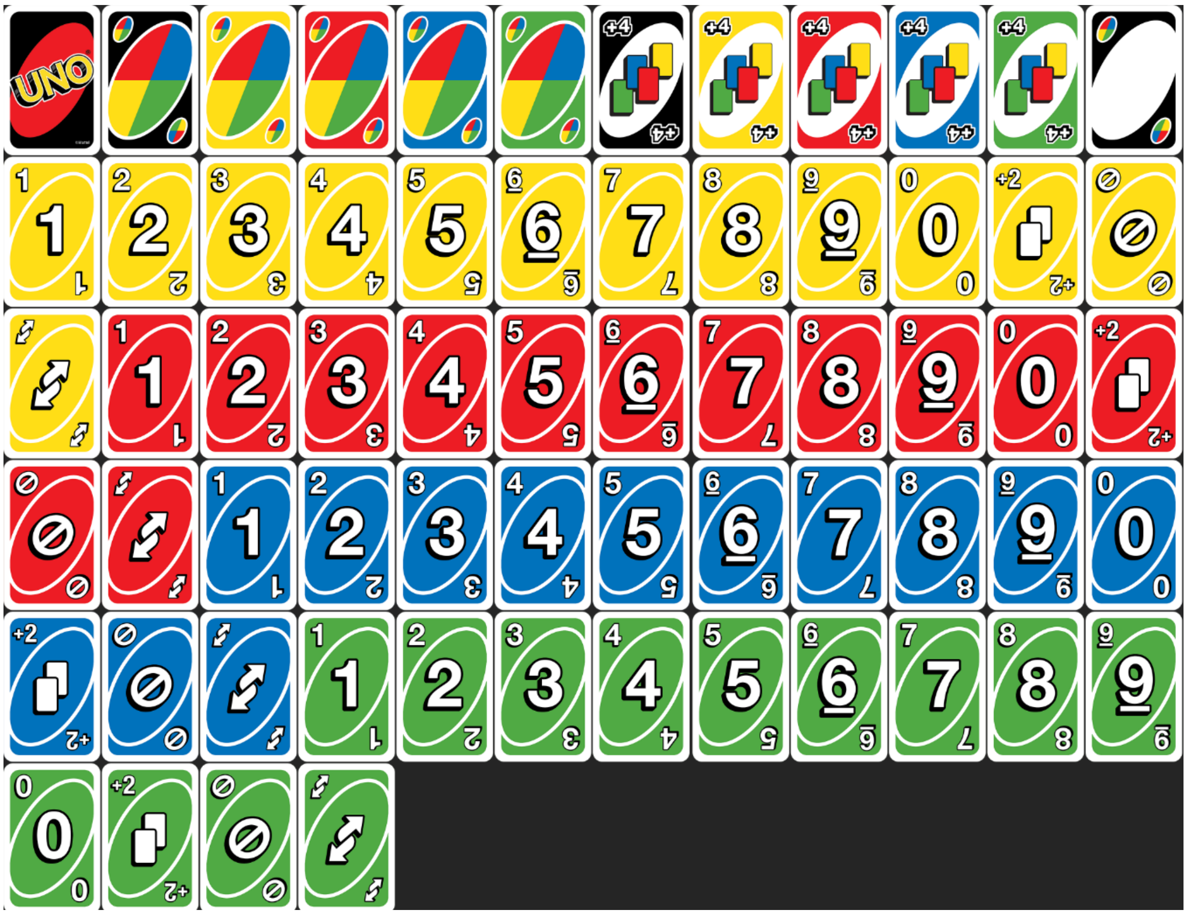
Reverse: reverses the direction that the games play. For example, if the game is playing clockwise this card would reverse it to counterclockwise. There are two of these cards for each colour	**Skip:** When a player places this card, the next player has to skip their turn.	**Draw Two:** When a person places this card, the next player will have to pick up two cards and forfeit his/her turn.

Wild Cards: Can be placed on any card and can change the colour of play.



Wild Draw Four - This card can be placed on top of any card. When a person places this card, the next player will have to pick up four cards and forfeit his/her turn. You may choose the colour of this card after you play it.	Wild/Colour Change - This card can be placed on top of any card. You may choose the colour of this card after you play it.

2.2 Deck

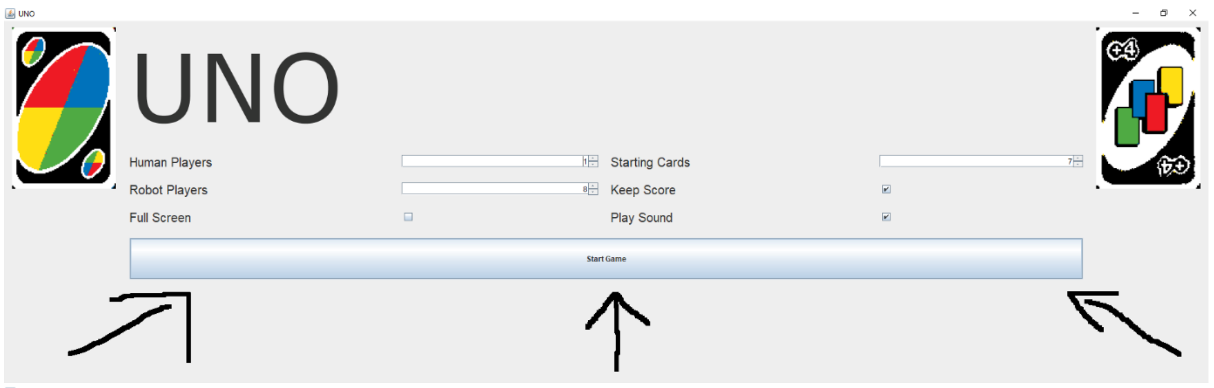


The above image is used for all the cards in the game it is downloaded from https://www.textures-resource.com/fullview/8671/.

The Uno deck contains 108 cards.

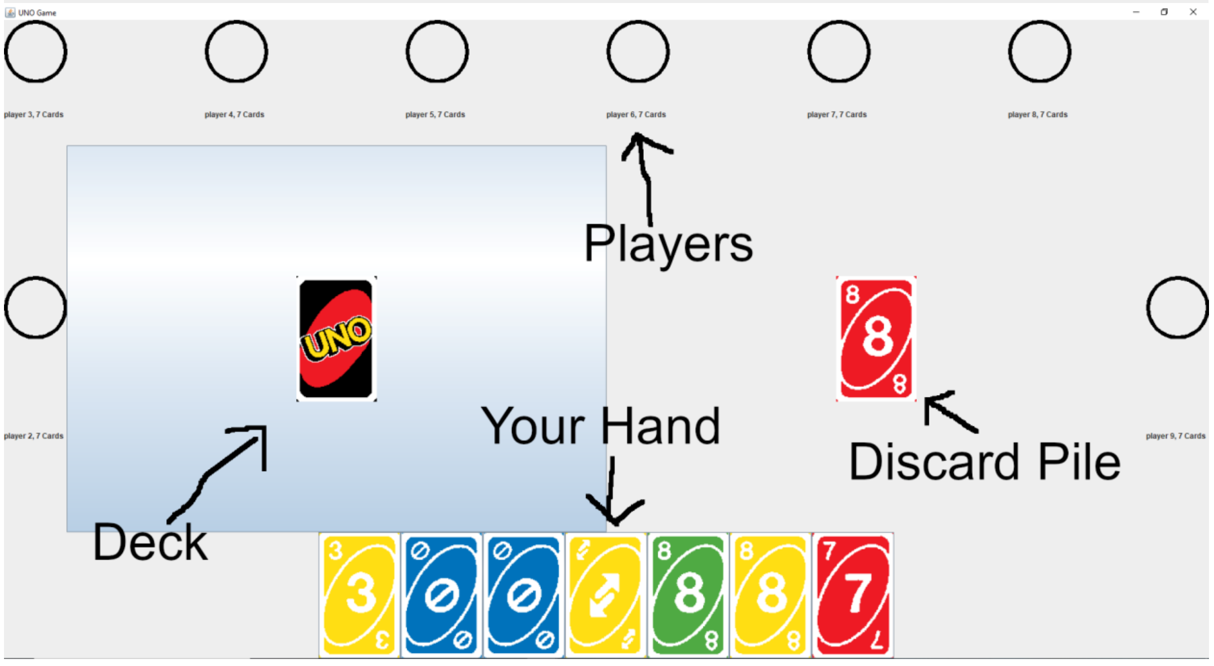
0	1 of each card for each colour	4 total
1-9	2 of each card for each colour	8*9=72
Reverse	2 of each card for each colour	8
Skip	2 of each card for each colour	8
Draw two	2 of each card for each colour	8
Wild Draw Four	4 cards total	4
Wild/Change Colour	4 cards total	4

2.3 Java Game



UNO

Human Players	1	Starting Cards	7
Robot Players	8	Keep Score	☑
Full Screen	☐	Play Sound	☑



My Uno java game is straight forward, on the starting screen you select what settings you want then you click Start Game.

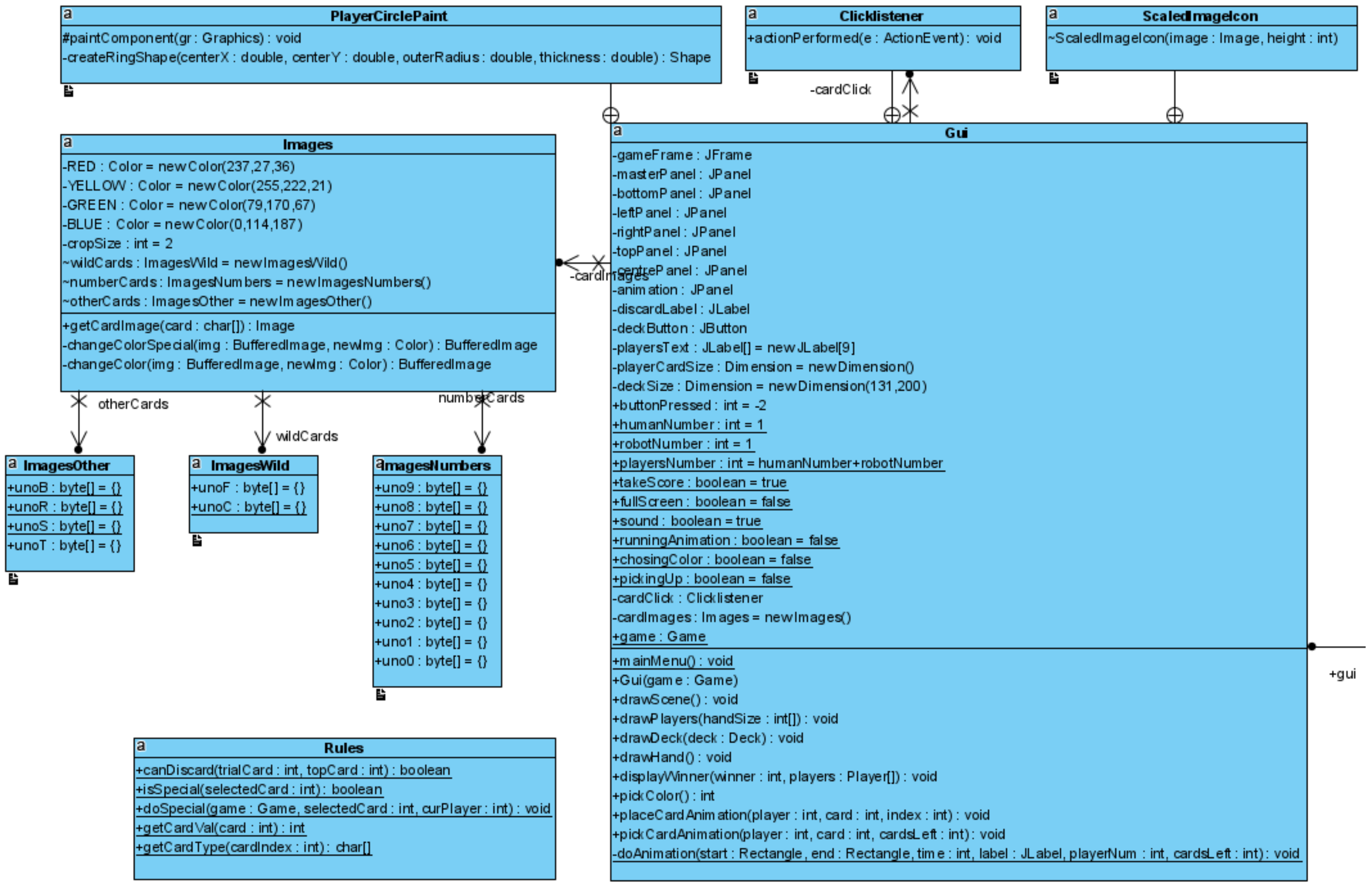
Then a new window opens with the game, push the card in your hand which you would like to place. If you do not want or can't play any card, click the deck to pick from the deck.

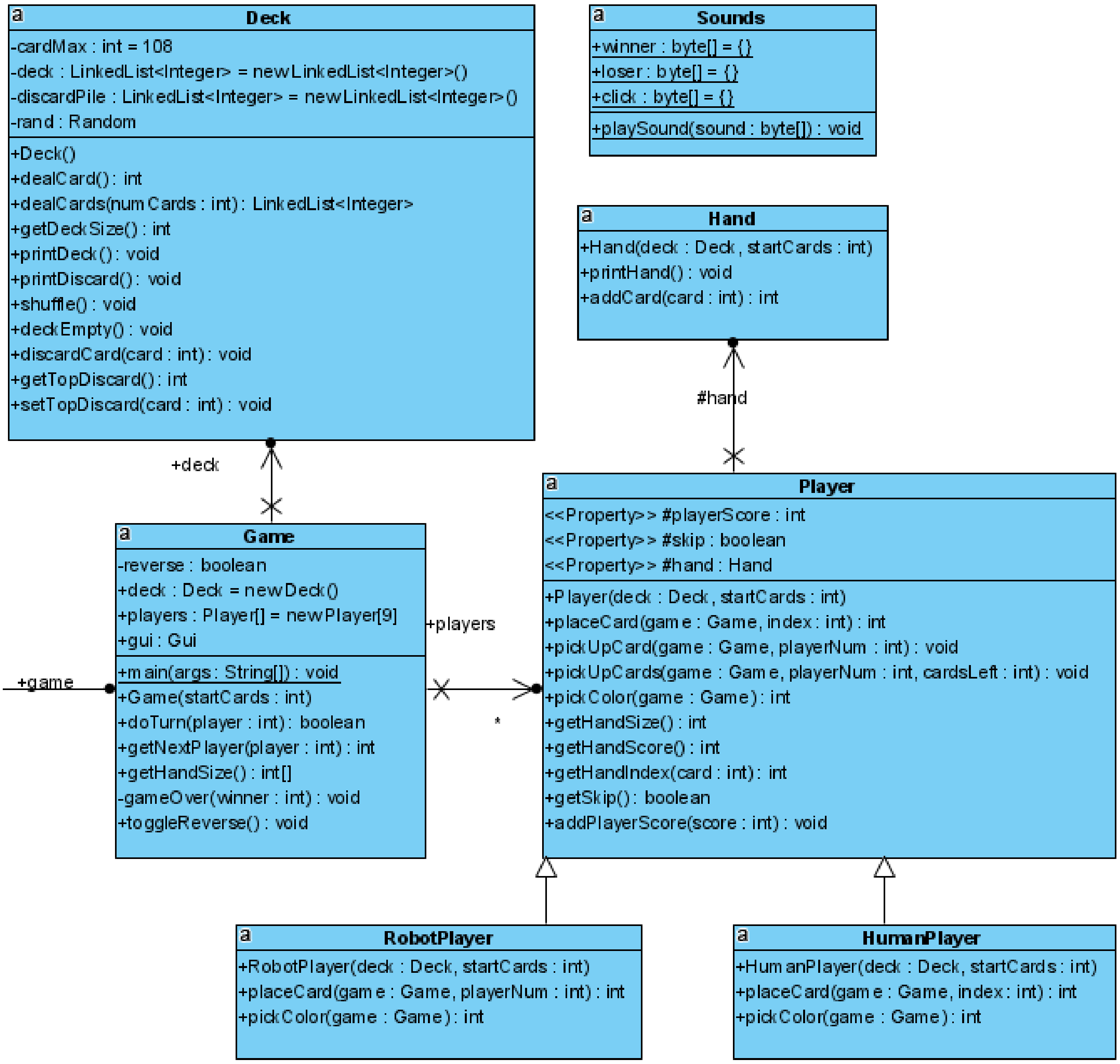
occasionally a popup will appear asking you to select the colour, just select the colour from the dropdown and push ok.

When the round finishes you will get a popup saying either you have won or lost and your score. You should write down your score on a piece of paper so that you can keep track of it. When you push ok the game will close.

If you would like to play robot vs robot you select 0 human players and then push start and watch

3. Code structure


PlayerCirclePaint
#paintComponent(gr : Graphics) : void
-createRingShape(centerX : double, centerY : double, outerRadius : double, thickness : double) : Shape
Clicklistener
+actionPerformed(e : ActionEvent) : void
ScaledImageIcon
~ScaledImageIcon(image : Image, height : int)
-cardClick
Images
-RED : Color = new Color(237,27,36)
-YELLOW : Color = new Color(255,222,21)
-GREEN : Color = new Color(79,170,67)
-BLUE : Color = new Color(0,114,187)
-cropSize : int = 2
~wildCards : ImagesWild = new ImagesWild()
~numberCards : ImagesNumbers = new ImagesNumbers()
~otherCards : ImagesOther = new ImagesOther()
+getCardImage(card : char[]) : Image
-changeColorSpecial(img : BufferedImage, newImg : Color) : BufferedImage
-changeColor(img : BufferedImage, newImg : Color) : BufferedImage
-cardImages
otherCards
wildCards
numberCards
ImagesOther
+unoB : byte[] = {}
+unoR : byte[] = {}
+unoS : byte[] = {}
+unoT : byte[] = {}
ImagesWild
+unoF : byte[] = {}
+unoC : byte[] = {}
ImagesNumbers
+uno9 : byte[] = {}
+uno8 : byte[] = {}
+uno7 : byte[] = {}
+uno6 : byte[] = {}
+uno5 : byte[] = {}
+uno4 : byte[] = {}
+uno3 : byte[] = {}
+uno2 : byte[] = {}
+uno1 : byte[] = {}
+uno0 : byte[] = {}
Rules
+canDiscard(trialCard : int, topCard : int) : boolean
+isSpecial(selectedCard : int) : boolean
+doSpecial(game : Game, selectedCard : int, curPlayer : int) : void
+getCardVal(card : int) : int
+getCardType(cardIndex : int) : char[]
Gui
-gameFrame : JFrame
-masterPanel : JPanel
-bottomPanel : JPanel
-leftPanel : JPanel
-rightPanel : JPanel
-topPanel : JPanel
-centrePanel : JPanel
-animation : JPanel
-discardLabel : JLabel
-deckButton : JButton
-playersText : JLabel[] = new JLabel[9]
-playerCardSize : Dimension = new Dimension()
-deckSize : Dimension = new Dimension(131,200)
+buttonPressed : int = -2
+humanNumber : int = 1
+robotNumber : int = 1
+playersNumber : int = humanNumber+robotNumber
+takeScore : boolean = true
+fullScreen : boolean = false
+sound : boolean = true
+runningAnimation : boolean = false
+chosingColor : boolean = false
+pickingUp : boolean = false
-cardClick : Clicklistener
-cardImages : Images = new Images()
+game : Game
+mainMenu() : void
+Gui(game : Game)
+drawScene() : void
+drawPlayers(handSize : int[]) : void
+drawDeck(deck : Deck) : void
+drawHand() : void
+displayWinner(winner : int, players : Player[]) : void
+pickColor() : int
+placeCardAnimation(player : int, card : int, index : int) : void
+pickCardAnimation(player : int, card : int, cardsLeft : int) : void
-doAnimation(start : Rectangle, end : Rectangle, time : int, label : JLabel, playerNum : int, cardsLeft : int) : void
+gui


Deck
-cardMax : int = 108
-deck : LinkedList<Integer> = newLinkedList<Integer>()
-discardPile : LinkedList<Integer> = newLinkedList<Integer>()
-rand : Random
+Deck()
+dealCard() : int
+dealCards(numCards : int) : LinkedList<Integer>
+getDeckSize() : int
+printDeck() : void
+printDiscard() : void
+shuffle() : void
+deckEmpty() : void
+discardCard(card : int) : void
+getTopDiscard() : int
+setTopDiscard(card : int) : void
Sounds
+winner : byte[] = {}
+loser : byte[] = {}
+click : byte[] = {}
+playSound(sound : byte[]) : void
Hand
+Hand(deck : Deck, startCards : int)
+printHand() : void
+addCard(card : int) : int
#hand
+deck
Game
-reverse : boolean
+deck : Deck = newDeck()
+players : Player[] = newPlayer[9]
+gui : Gui
+main(args : String[]) : void
+Game(startCards : int)
+doTurn(player : int) : boolean
+getNextPlayer(player : int) : int
+getHandSize() : int[]
-gameOver(winner : int) : void
+toggleReverse() : void
+game
+players
*
Player
<<Property>> #playerScore : int
<<Property>> #skip : boolean
<<Property>> #hand : Hand
+Player(deck : Deck, startCards : int)
+placeCard(game : Game, index : int) : int
+pickUpCard(game : Game, playerNum : int) : void
+pickUpCards(game : Game, playerNum : int, cardsLeft : int) : void
+pickColor(game : Game) : int
+getHandSize() : int
+getHandScore() : int
+getHandIndex(card : int) : int
+getSkip() : boolean
+addPlayerScore(score : int) : void
RobotPlayer
+RobotPlayer(deck : Deck, startCards : int)
+placeCard(game : Game, playerNum : int) : int
+pickColor(game : Game) : int
HumanPlayer
+HumanPlayer(deck : Deck, startCards : int)
+placeCard(game : Game, index : int) : int
+pickColor(game : Game) : int

4. Data structures and algorithms used

4.1 Overview of Data Structures

Throughout the game, I use linked lists because I am constantly manipulating both the deck and the discard pile and the user's hands. The linked lists also allow me to dynamically change the size of the lists as I do not know how large any of the lists will be at start-up.

I mostly access the data sequentially, insert and delete items often meaning that linked lists are ideally suited.

4.2 Representing each card

To represent each card throughout the game I use an int which is unique to every card. This means I designated a unique int for each card e.g. The 4 wild plus four cards are numbers 0-3.

I use a lookup table called getCardType which is in the rules class, to convert from my number to the name and colour of the card.

The name and colour of a card are represented by a char array the first item in the array is the number or symbol of the card. The second char in the array is the colour.

The symbols I use are

Symbol	Meaning
F	Wild plus 4
T	Plus 2
C	Wild/Colour Change
S	Skip
R	Reverse

Symbol	Meaning
R	Red
Y	Yellow
G	Green
B	Blue
W	Wild/Black

The lookup table to convert from my number representation of a card.

0-107 are the 108 cards of the official Uno.

108-115 are colour variants of the wild cards only used for when the player selects a colour.

116 is the back of the playing card used for the deck and dealing cards.

Number Range (Inclusive)	Card range (Inclusive)	Colour Code		Number Range (Inclusive)	Card range (Inclusive)	Colour Code
0-3	Wild +4	Wild				
4-7	Wild	Wild				
8	0	Red		58	0	Green
9-17	1-9	Red		59-67	1-9	Green
18-26	1-9	Red		68-76	1-9	Green
27-28	Skip	Red		77-78	Skip	Green
29-30	+2	Red		79-80	+2	Green
31-32	Reverse	Red		81-82	Reverse	Green
33	0	Yellow		83	0	Blue
34-42	1-9	Yellow		84-92	1-9	Blue
43-51	1-9	Yellow		93-101	1-9	Blue
52-53	Skip	Yellow		102-103	Skip	Blue
54-55	+2	Yellow		104-105	+2	Blue
56-57	Reverse	Yellow		106-107	Reverse	Blue
				108-111	Wild +4	R-B
				112-115	Wild	R-B
				116	Back of card	W

4.3 Deck Data Structure

```
private LinkedList<Integer> deck = new LinkedList<Integer>();
private LinkedList<Integer> discardPile = new LinkedList<Integer>();
```

The deck class stores two linked lists one for the deck where players draw their cards from. The other is the discard pile which stores all the cards which the players discard.

The reason I need to store the discard pile is that in the case that the deck runs out the game will automatically move the discard pile to the deck and reshuffle.

If the deck and the discard pile is empty, meaning that the players have all 108 cards in their hands a new deck will be added.

4.4 Hand Data Structure

```
public Player players[] = new Player[9];
```

The main game class stores an array of the player class, each player class has an instance of hand.

```
protected Hand hand;         //stores the player's hand
```

the hand class is an extension of the LinkedList class.

```
public class Hand extends LinkedList<Integer>
```

Therefore, the player's hand is stored in a custom linked list which is inside the player class.

The reason for this complexity is to allow me to add a `printHand` method and `addCard` which returns the item which was added.

4.5 Images Data and Algorithm

As I had to deliver this app in a single.java file as opposed to a .jar it meant that I had to embed all the images inside the code, which lead to much complexity and workarounds.

The main limitation I ran into is that a class cannot have more than 65535 bytes of static variables and cannot have over 65535 bytes for the local variable.

This meant that I had to store the images in the minimum amount of storage.

The first thing I did was split the image of all the Uno cards into 64 images (some of the 108 cards are duplicates).

The storage limitation meant that I could not have the full 64 images which were of all Uno cards in all colours. The way I reduced the number of images needed was to get rid of the colours. This was done by converting all the red cards into black and white versions. The three wild cards which had multiple colours were left in multiple colours.

At this point, I have reduced the number of cards from 64 to 16 cards, but they still need to be compressed as in their current jpg format they were still taking 29KB each card.

I found that the most efficient format was a .tiff as it is the only format which supports a 1-bit pallet meaning black or white (not greyscale). This also meant that I could use CCIT fax compression. This greatly reduced the image size down to 700 Bytes per images.

The problem was after doing that for all the images I then realized that .tiff was only compatible with Java 9+.

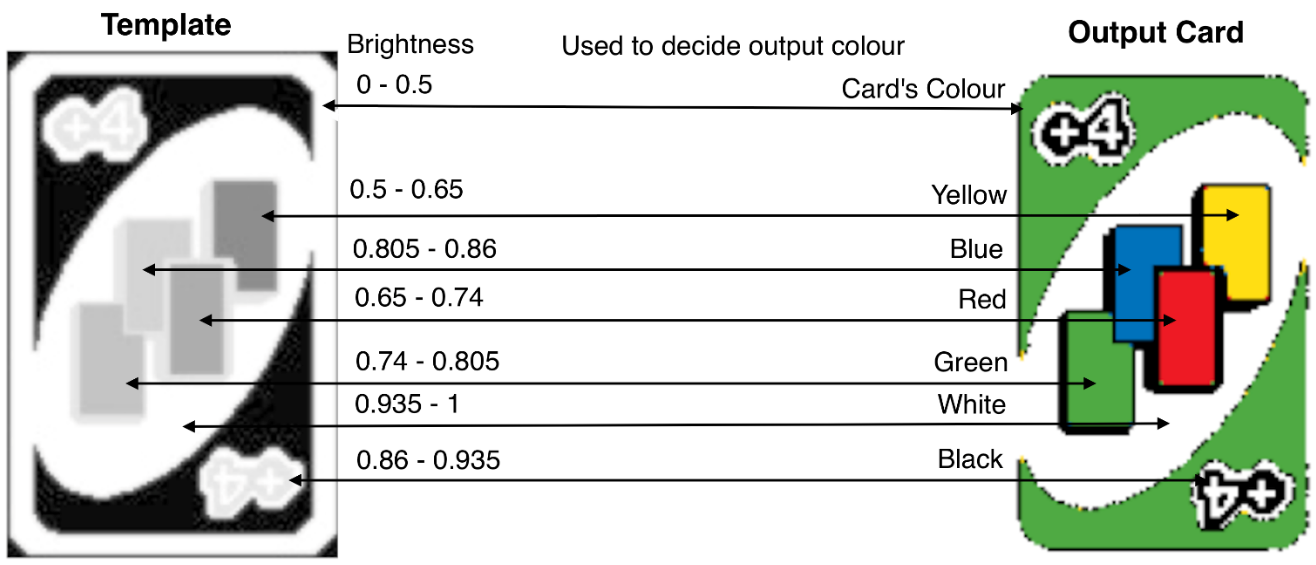
So, I had to switch to .jpg which only supports greyscale, this would mean that I would need a lot of compressions to get the file size small enough. I first converted all the images to greyscale.

For the images which had multiple colours, I made up a conversion from colour to greyscale, so that in java I could convert them back to colour. By hand, I set the images to arbitrary greyscale values to represent a specific colour.

I then reduced the dimensions of the images and corrected them by hand before exporting them with the highest level of compression. Now each file was 2KB which is much larger than the .tiff which much worse image quality. Because there were so many compression anomalies in the image, I had to program my algorithm to remove them.

Here is an example of my algorithm working.

Colourisation



The Colourisation of each image for a card is done at RunTime because of size limitations.

I then convert the .jpg to hex using https://hexed.it/. I then inputted the hex into a java program I wrote to convert it to decimal and then an unsigned int and then a byte array.

The output of the program looked like this:

```
public static final byte [] click = {77,84,104,100,0,0,0,6,0,1,0,2,1,-
128,77,84,114,107,0,0,0,19,0,     -1,88,4,4,2,24,8,0,-1,81,3,8,82,-82,0,-
1,47,0,77,84,114,107, 0,0,0,27,0,-1,3,8,68,114,117,109,32,75,105,116,0,-
55,0,0,-103, 85,50,96,-119,85,0,0,-1,47,0,};
```

I then pasted this into a new java class in my main app.

I used gimp for all image editing.

4.6 Sound Data Structure

Sounds are stored very similarly to the way pictures are stored. The only difference is the file type used is midi. Midi is the most efficient file type as it only stores the instruments identifier and the notes they should play. This means that it is not saving a recording instead it is interpreted at runtime. This means that it sounds slightly different on each device.

An example of the click sound is displayed in the section above.

4.7 Robot Player Algorithm

There are two robot algorithms which are used. the first is for placeCard and the second is pickColour.

The place card algorithm works by starting at index 0 of the player's hand and moving across the hand. The first card it finds which can be placed according to the Uno rules it will place. If there are no cards which can be placed, it will pick up a card

This means that it will always place cards from left to right so the order of the cards in the robot's hand matters.

The pick colour algorithm works by counting how many reds cards are in the player's hand then it moves on to yellow and continues until it has the colour of the largest number of cards. It selects this colour.

This means that the robot will always place a wild card in the colour which has the largest amount of.

4.8 Other algorithm

canDiscard is a basic algorithm in the rules class which allows the game to tell if the action is according to the rules of Uno.

5. Learning outcomes

During this project, I have learnt many things because of the challenge imposed by the size limitations. This means that I learned ways of getting around them by using clever data structures, algorithms and compression.

I also learnt how to write java GUIs, sounds, draw images, Data structures, class hierarchy and multi-threading.

6. Appendix

6.1 Code for generating images

```
package gui;


import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.nio.file.Files;


public class ConvertHexToDecimal {


	public static void main(String[] args) throws FileNotFoundException, IOException {

		BufferedReader csvReader = new BufferedReader(new
FileReader("UnoCards/unoCardsHex.csv"));

		String row;

		System.out.print("public static final byte [] uno = {");

		while ((row = csvReader.readLine()) != null) {

		    String[] data = row.split(",");

		    for (int i=0; i<data.length; i++) {

		    	data[i] = data[i].substring(2,4);

		         System.out.print((byte)Integer.parseInt(data[i],16));

		       System.out.print(",");

		    }

		}

		System.out.print("};");

		csvReader.close();

	}


	public static byte[] extractBytes (String ImageName) {
        File fi = new File(ImageName);

        byte[] fileContent = {};

            try {

            fileContent = Files.readAllBytes(fi.toPath());

        } catch (IOException e) {

            e.printStackTrace();

        }


        try (FileOutputStream fos = new FileOutputStream("uno0output.txt")) {

          fos.write(fileContent);

          //fos.close(); There is no more need for this line since you had created the instance of
"fos" inside the try. And this will automatically close the OutputStream

        } catch (FileNotFoundException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        } catch (IOException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }


        return fileContent;

    }


}
```

6.2 Code For Game

```
package game;

//By Donovan Webb - 1933061

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;

import java.util.*;
import java.util.List;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import javax.swing.*;

import javax.imageio.ImageIO;

import javax.sound.midi.*;


public class Game {
    private boolean reverse;
    public Deck deck = new Deck();
    public Player players[] = new Player[9];
      public Gui gui;

      public static void main(String[] args) {
            //Schedule a job for the event-dispatching thread:
         //creating and showing this application's GUI.
         javax.swing.SwingUtilities.invokeLater(new Runnable() {
             public void run() {
                  System.out.println(System.getProperty("java.version"));
//prints out the java version
                  System.out.println("this is a single .java file");
                 Gui.mainMenu(); //starts the gui
             }
         });
      }

      public Game(int startCards) {
            //Initializes players, designed for any number of human players
            for (int i=0; i<Gui.humanNumber; i++) {
                  players[i] = new HumanPlayer(deck, startCards);
            }
            for (int i=Gui.humanNumber; i<Gui.robotNumber+Gui.humanNumber; i++)
{
                  players[i] = new RobotPlayer(deck, startCards);
            }

            /*
             * turns a card over from the top of the pile
             * if the card turned over was a special card
             * turn over another card
             */
            int topCard = deck.dealCard();
            deck.discardCard(topCard);
            while (Rules.isSpecial(topCard)) {
                  topCard = deck.dealCard();
                  deck.discardCard(topCard);
            }

            //Initializes the gui
            gui = new Gui(this);
```

```
		//refreshes the gui
		gui.drawScene();

	}

	/*
	 * main recursive function which calls most other functions
	 *
	 * each time doTurn is called it conducts the go of the specified player
	 * it then calls place card which will trigger an animation in a new
thread
	 * it the returns back while the animation is running
	 * and calls dospecial which implements the special cards in uno then it
returns
	 * after the animation thread finishes it will call doTurn again which
creates a recursive loop
	 * the loop ends when it reaches the human player
	 */
	public boolean doTurn(int player) {
		int discardedCard;
		if (player<0) {
			player=Gui.playersNumber-1;
		}
		if (player>Gui.playersNumber-1) {
			player=0;
		}
		if (Gui.runningAnimation || Gui.chosingColor) {
			return false;
		}

		gui.drawScene();

		int index =  player;
		if (player==0 && Gui.humanNumber == 1) {
			index =  gui.buttonPressed;
		}

		discardedCard = players[player].placeCard(this, index);
		if (discardedCard >= 0) {
			Rules.doSpecial(this, discardedCard, player);
			System.out.println("Player "+(player+1)+" placed a
"+Rules.getCardType(discardedCard)[0]+Rules.getCardType(discardedCard)[1]);
		} else if (discardedCard ==-244) {
			if (getNextPlayer(player) == 0 && Gui.humanNumber == 1) {
			return false;
		 }
			doTurn(getNextPlayer(player));
		}
		return false;
	}

	//returns the next player number which should be called in the loop
	public int getNextPlayer(int player) {
		if (reverse) {
			if (player==0) {
				return Gui.playersNumber-1;
			} else {
				return player-1;
			}
		} else {
			if (player==Gui.playersNumber-1) {
				return 0;
			} else {
				return player+1;
			}
		}
	}

	//gets the hand size of all players and returns them in an array
	public int[] getHandSize() {
		int handSize[] = new int[Gui.playersNumber];
		for (int x=0; x<Gui.playersNumber; x++) {
			handSize[x] = players[x].getHandSize();
			if (handSize[x]==0) {
				System.out.println("Player "+(x+1)+" Won");
				gameOver(x);
			}
		}
		return handSize;
	}

	//called when someone wins the game
	//calculates the score if appropriate
	//and calls displayWinner in gui class
	private void gameOver(int winner) {
		if (Gui.takeScore) {
			int totalScore = 0;
			for (int i = 0; i < Gui.playersNumber; i++) {
				if (i != winner) {
					totalScore += players[i].getHandScore();
				}
			}
			players[winner].addPlayerScore(totalScore);
		}
		gui.displayWinner(winner, players);
	}

	//toggles the reverse variable
	public void toggleReverse() {
		if (reverse) {
			reverse = false;
		} else {
			reverse = true;
		}
```

```
      }
}
```

```
class Gui {
      //the panels for the game view
      private JFrame gameFrame;
   private JPanel masterPanel;
   private JPanel bottomPanel;
   private JPanel leftPanel;
   private JPanel rightPanel;
   private JPanel topPanel;
   private JPanel centrePanel;
   private JPanel animation;

   //saves the location of discard and deck for animations
   private JLabel discardLabel;
      private JButton deckButton;

      private Clicklistener cardClick; //calls this class whenever a card is
pressed
   private Images cardImages = new Images(); //Initializes the images
   private JLabel [] playersText = new JLabel[9];//saves the location of the
players
      private Dimension playerCardSize = new Dimension();//saves the size of
the players cards for animations
      private final Dimension deckSize = new Dimension(131,200); // the size of
the deck the width = 200*0.655=131
      public int buttonPressed = -2;  //stores the card the user clicked on

      //stores the values set on the menu screen
      public static int humanNumber = 1;
      public static int robotNumber = 1;
      public static int playersNumber = humanNumber+robotNumber;
      public static boolean takeScore = true;
      public static boolean fullScreen = false;
      public static boolean sound = true;

      public static boolean runningAnimation = false; //used to block the game
continuing during a animation
   public static boolean chosingColor = false; //used to block the game
continuing while waiting for user input
```

```
   public static boolean pickingUp = false; //used to block the game continuing
while a user is picking up from a special card

   //stores the instance of the Game class
   public static Game game;

   //draws the main menu
   public static void mainMenu() {
      JButton button;
        JLabel label;
        SpinnerNumberModel model;

        //create a local instance of the images
           Images cardImage = new Images();
           Font title = new Font("Verdana", Font.PLAIN, 150);
           Font h3 = new Font("Arial", Font.PLAIN, 20);

      //Create and set the window.
        JFrame menuFrame = new JFrame("UNO");
        menuFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           menuFrame.pack();
        menuFrame.setMinimumSize(new Dimension(850,380));
        menuFrame.setSize(new Dimension(1000,800));
        menuFrame.setVisible(true);

        Container pane = menuFrame.getContentPane();
           pane.setLayout(new GridBagLayout());
           GridBagConstraints c = new GridBagConstraints();

           //Default constants
           c.fill = GridBagConstraints.HORIZONTAL;
           c.anchor = GridBagConstraints.NORTHWEST;
           c.insets = new Insets(10,10,10,10);

           //display a +4 card
           char [] card = { 'C','W' };
           label = new JLabel( new ScaledImageIcon(cardImage.getCardImage( card
),257) );
           label.setPreferredSize(new Dimension(169,257));
           c.weightx = 0;
           c.gridwidth = 1;
           c.gridheight = 4;
           c.gridx = 0;
           c.gridy = 0;
           pane.add(label, c);

           //display a Color Changing Card
           card[0] = 'F';
           card[1] = 'W';
           label = new JLabel( new ScaledImageIcon(cardImage.getCardImage( card
),257) );
           label.setPreferredSize(new Dimension(169,257));
           c.gridx = 5;
           pane.add(label, c);

           //display the text UNO
           label = new JLabel("UNO");
           label.setFont(title);
           c.weightx = 0.5;
           c.gridwidth = 4;
           c.gridheight = 1;
           c.gridx = 1;
           c.gridy = 0;
           pane.add(label, c);


           //select human number
           label = new JLabel("Human Players");
           label.setFont(h3);
           c.gridwidth = 1;
           c.gridy = 1;
           pane.add(label, c);


           model = new SpinnerNumberModel(1,1,1,1);
           //if you add the comment on the below line it block you to play robot
   vs robot
           //by setting human players to zero
           model = new SpinnerNumberModel(1,0,1,1);
           JSpinner humanSpinner = new JSpinner(model);
           c.gridx = 2;
           pane.add(humanSpinner, c);

           //select robot number
           label = new JLabel("Robot Players");
           label.setFont(h3);
           c.gridx = 1;
           c.gridy = 2;
           pane.add(label, c);

           model = new SpinnerNumberModel(1,1,8,1);
           JSpinner robotSpinner = new JSpinner(model);
           c.gridx = 2;
           pane.add(robotSpinner, c);

           //select the number of starting cards
           label = new JLabel("Starting Cards");
           label.setFont(h3);
           c.gridx = 3;
           c.gridy = 1;
           pane.add(label, c);

           model = new SpinnerNumberModel(7,1,99,1);
           JSpinner startCardSpinner = new JSpinner(model);
           c.gridx = 4;
           c.gridy = 1;
           pane.add(startCardSpinner, c);

           //chose weather to keep score
           label = new JLabel("Keep Score");
           label.setFont(h3);
           c.gridx = 3;
           c.gridy = 2;
           pane.add(label, c);

           JCheckBox scoreText = new JCheckBox("",true);
           c.gridx = 4;
           c.gridy = 2;
           pane.add(scoreText, c);


           //chose weather to launch in full screen
           label = new JLabel("Full Screen");
           label.setFont(h3);
           c.gridx = 1;
           c.gridy = 3;
           pane.add(label, c);

           JCheckBox fullScreenText = new JCheckBox("",false);
           c.gridx = 2;
           c.gridy = 3;
           pane.add(fullScreenText, c);


           //chose weather to play sounds
           label = new JLabel("Play Sound");
           label.setFont(h3);
           c.gridx = 3;
           c.gridy = 3;
           pane.add(label, c);

           JCheckBox soundText = new JCheckBox("",true);
           c.gridx = 4;
           c.gridy = 3;
           pane.add(soundText, c);


           //displays the start game button
           button = new JButton("Start Game");
           c.ipady = 40;      //make this component tall
           c.weightx = 0.5;
           c.weighty = 1;
           c.gridwidth = 4;
           c.gridheight = 1;
           c.gridx = 1;
           c.gridy = 4;
           pane.add(button, c);

           //if the start game button is pressed it will setup all the variables
           //based on the fields in the gui
           //and initialize a new instance of the Game class

           button.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                  humanNumber =
   Integer.parseInt(humanSpinner.getValue().toString());
                  robotNumber =
   Integer.parseInt(robotSpinner.getValue().toString());
                  playersNumber = humanNumber+robotNumber;
                  fullScreen = fullScreenText.isSelected();
                  sound = soundText.isSelected();
                  takeScore = scoreText.isSelected();

                  new Game(
   Integer.parseInt(startCardSpinner.getValue().toString()) );
                  if (humanNumber == 0) {
                       game.doTurn(0);
                  }
               }
           });
      }

      public Gui(Game game) {

       Gui.game = game;
       cardClick = new Clicklistener();

       //Create and set up the window.
           gameFrame = new JFrame("UNO Game");
           gameFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
           gameFrame.pack();
           gameFrame.setMinimumSize(new Dimension(600,700));
           gameFrame.setSize(1000,800);
           gameFrame.setVisible(true);

           //makes sure the game window is on top of the menu window
           gameFrame.setAlwaysOnTop(true);
           gameFrame.toFront();
           gameFrame.requestFocus();
           gameFrame.setAlwaysOnTop(false);
           if (fullScreen) {
               gameFrame.setExtendedState(JFrame.MAXIMIZED_BOTH);
           }
           //initilizes and adds all the panels to there correct positions
           masterPanel = new JPanel(new BorderLayout());
           topPanel = new JPanel(new GridLayout(2,6));
           leftPanel = new JPanel(new GridLayout(2,1));
           rightPanel = new JPanel(new GridLayout(2,1));
           centrePanel =new JPanel(new GridLayout(1,2));
           bottomPanel = new JPanel(new GridBagLayout());
           masterPanel.add(topPanel, BorderLayout.PAGE_START);
           masterPanel.add(leftPanel, BorderLayout.LINE_START);
           masterPanel.add(rightPanel, BorderLayout.LINE_END);
           masterPanel.add(centrePanel, BorderLayout.CENTER);
           masterPanel.add(bottomPanel, BorderLayout.PAGE_END);

           //sets up the animation glass Pane
           animation = new JPanel(null);
           gameFrame.setGlassPane(animation);
           animation.setVisible(true);
           animation.setOpaque(false);

           gameFrame.getContentPane().add(masterPanel); //adds the master panel to
    the frame

           gameFrame.addComponentListener(new ComponentAdapter() {
               public void componentResized(ComponentEvent componentEvent) {
                   /* TODO cache answer as this event is called many times during
    a window resize */
                   //draws hand whenever the window is resized
                   drawHand();
               }
           });

           //sets up all the circles representing the robot players
             for (int i=1; i<playersNumber; i++) {
               PlayerCirclePaint circle = new PlayerCirclePaint();
               if (i<7) {
                   circle.setPreferredSize(new Dimension(100,100));
                   topPanel.add(circle);
               } else if (i==7) {
                   circle.setPreferredSize(new
    Dimension(100,leftPanel.getHeight()/2));
                   leftPanel.add(circle);
               } else {
                   circle.setPreferredSize(new
    Dimension(100,leftPanel.getHeight()/2));
                   rightPanel.add(circle);
               }
           }

             //sets up all the text to accompany the robot players circles
         playersText[0] = new JLabel("player 1,  Cards");

         playersText[0].setLocation(masterPanel.getWidth(),masterPanel.getHeight()
    );
           for (int i=1; i<playersNumber; i++) {
               playersText[i] = new JLabel("player "+(i+1)+",  Cards");
               if (i<=7 && ( i>1 || playersNumber <= 7 )) {
                   topPanel.add(playersText[i]);
```

```
           } else if (i==8) {
                rightPanel.add(playersText[i]);
           } else {
                leftPanel.add(playersText[i]);
           }
       }

       //refreshes the layout of the view
         gameFrame.validate();
   }

   //renders the entire gameFrame
   public void drawScene() {
     drawPlayers(game.getHandSize());
           drawHand();
           drawDeck(game.deck);
   }

   //updates the robot players card count
   public void drawPlayers(int [] handSize) {
       for (int i=1; i<playersNumber; i++) {
           playersText[i].setText("player "+(i+1)+", "+handSize[i]+" Cards");
       }

         gameFrame.validate();
   }

   //draws the deck
   public void drawDeck(Deck deck) {
         if (deckButton == null || discardLabel == null) {
           //Initializes deckButton and discardLabel
           char [] card = { 'B','W' };
           deckButton = new JButton( new
ScaledImageIcon(cardImages.getCardImage( card ),deckSize.height));
           deckButton.setName("-1");
           deckButton.setPreferredSize(deckSize);
           deckButton.setMinimumSize(deckSize);
           deckButton.setMaximumSize(deckSize);
           deckButton.addActionListener(cardClick);
           centrePanel.add(deckButton);

           discardLabel = new JLabel( new
ScaledImageIcon(cardImages.getCardImage( Rules.getCardType(
deck.getTopDiscard() ) ),deckSize.height));
           discardLabel.setPreferredSize(deckSize);
           discardLabel.setMinimumSize(deckSize);
           discardLabel.setMaximumSize(deckSize);
           centrePanel.add(discardLabel);
         } else {
           //changes the card displayed on the discard pile
           discardLabel.setIcon(new ScaledImageIcon(cardImages.getCardImage(
Rules.getCardType( deck.getTopDiscard() ) ),deckSize.height));
```

```
        }
        gameFrame.validate();
    }

    //renders the players hand
    public void drawHand() {
        Hand hand = game.players[0].getHand();
        int handSize = hand.size();
        JButton button;
      JSeparator separator = new JSeparator(JSeparator.VERTICAL); //adds 
spacers to either side of the cards
        JPanel handPanel = new JPanel(new GridBagLayout());

        //clears the bottom panel and adds the new panel to it
        bottomPanel.removeAll();
        bottomPanel.setLayout(new BorderLayout());
        bottomPanel.add(handPanel, BorderLayout.CENTER);

        //calculates the width and height for each card in the players hand
        playerCardSize.width = bottomPanel.getWidth()/handSize;
        if (playerCardSize.width<50) {
         /* TODO add pages */
         playerCardSize.width=50;
        }
        if (playerCardSize.width>deckSize.width) {
         /* TODO add pages */
         playerCardSize.width=deckSize.width;
        }
        playerCardSize.height = (int)(playerCardSize.width*1.5357);

        //calculates the size required on each side of the players hand
        Dimension sepearatorSize = new Dimension((bottomPanel.getWidth()-
playerCardSize.width*handSize)/2, 0);

        //updates the location of player text to the end of the hand
        playersText[0].setLocation(playerCardSize.width*(handSize-
1)+sepearatorSize.width, masterPanel.getHeight()-playerCardSize.height);

        //sets the size and position of the start separator
        separator.setPreferredSize(sepearatorSize);
        bottomPanel.add(separator, BorderLayout.PAGE_START);

        //adds all the cards to the handPanel
        for (int i=0; i<handSize; i++) {
         button = new JButton( new ScaledImageIcon(cardImages.getCardImage(
Rules.getCardType(hand.get(i)) ),playerCardSize.height));
              button.setPreferredSize(playerCardSize);
              button.setMinimumSize(playerCardSize);
              button.setMaximumSize(playerCardSize);
              button.setSize(playerCardSize);
              button.addActionListener(cardClick);
              button.setName(""+i);
```

```
                handPanel.add(button);
            }

            //sets the size and position of the end seperator
            bottomPanel.add(separator, BorderLayout.PAGE_END);

            gameFrame.validate();
    }

        //updates the gui with the number of cards the players have
        //Displays a pop up with the winner and the score of the winner then
exits the game when the user pushes ok
        public void displayWinner(int winner, Player[] players) {
            int handSize[] = new int[Gui.playersNumber];
            String message = "Player "+(winner+1)+" has Won this round";
            if (Gui.takeScore) {
                message = "Player "+(winner+1)+" has Won this round with a
score of "+players[winner].getPlayerScore();
            }

            for (int x=0; x<Gui.playersNumber; x++) {
                handSize[x] = players[x].getHandSize();
            }
            drawPlayers(handSize);

            if (winner == 0) {
                Sounds.playSound(Sounds.winner);

                JTextArea winnerLabel = new JTextArea();
                JLabel rocket = new JLabel("🚀🎆");
                rocket.setFont(new Font(Font.SANS_SERIF,Font.BOLD, 150));
                winnerLabel.setEditable(false);
                winnerLabel.setText("\r\n"
                                +
"░██╗░░░░░░░██╗██╗███╗░░██╗███╗░░██╗███████╗██████╗░\r\n"
                                +
"░██║░░██╗░░██║██║████╗░██║████╗░██║██╔════╝██╔══██╗\r\n"
                                +
"░╚██╗████╗██╔╝██║██╔██╗██║██╔██╗██║█████╗░░██████╔╝\r\n"
                                +
"░░████╔═████║░██║██║╚████║██║╚████║██╔══╝░░██╔══██╗\r\n"
                                +
"░░╚██╔╝░╚██╔╝░██║██║░╚███║██║░╚███║███████╗██║░░██║\r\n"
                                +
"░░░╚═╝░░░╚═╝░░╚═╝╚═╝░░╚══╝╚═╝░░╚══╝╚══════╝╚═╝░░╚═╝\r\n");

                //Initializes and displays the dialog
                JPanel winningScreen = new JPanel(new FlowLayout());
                winningScreen.add(rocket);
                winningScreen.add(winnerLabel);
                winningScreen.add(new JLabel(message));
                JOptionPane.showConfirmDialog(gameFrame, winningScreen, 
"Player "+(winner+1)+" has Won", JOptionPane.DEFAULT_OPTION);
            } else {
                Sounds.playSound(Sounds.loser);
                
                JTextArea winnerLabel = new JTextArea();
                winnerLabel.setEditable(false);
                winnerLabel.setText("\r\n"
                        + "██╗░░░░░░██████╗░███████╗███████╗██████╗░\r\n"
                        + "██║░░░░░██╔═══██╗██╔════╝██╔════╝██╔══██╗\r\n"
                        + "██║░░░░░██║░░░██║███████╗█████╗░░██████╔╝\r\n"
                        + "██║░░░░░██║░░░██║╚════██║██╔══╝░░██╔══██╗\r\n"
                        + "███████╗╚██████╔╝███████║███████╗██║░░██║\r\n"
                        + "╚══════╝░╚═════╝░╚══════╝╚══════╝╚═╝░░╚═╝\r\n");
                
                //Initializes and displays the dialog
                JPanel winningScreen = new JPanel(new GridLayout(2,1));
                winningScreen.add(winnerLabel);
                winningScreen.add(new JLabel(message));
                JOptionPane.showConfirmDialog(gameFrame, winningScreen, 
"Player "+(winner+1)+" has Won", JOptionPane.DEFAULT_OPTION);
            }
            System.exit(0);
        }
        
        //displays the pick color dialog and waits for the user to chose then 
returns the users choice
        //can be called recursively
        public int pickColor() {
        int output = 1;
        Gui.chosingColor = true;  //blocks the game from preceding before the 
user chooses a color
            String[] possibleValues = { "Red", "Yellow", "Green", "Blue" };
            JComboBox<Object> colorF = new JComboBox<Object>(possibleValues);
            colorF.setSelectedIndex(0);
            
            //Initializes and displays the dialog
            JPanel question = new JPanel(new GridLayout(2,1));
            question.add(new JLabel("Choose your Color"));
            question.add(colorF);
            int result = JOptionPane.showConfirmDialog(gameFrame, question, 
"Choose your Color", JOptionPane.DEFAULT_OPTION);
            
            if (result == JOptionPane.OK_OPTION) {
                //if the user press OK then return the selected value
            String selectedVal = colorF.getSelectedItem().toString();
                switch (selectedVal) {
                    case "Red":
                        output = 1;
                        break;
                    case "Yellow":
                        output = 2;
                        break;
                    case "Green":
                        output = 3;
                        break;
                    case "Blue":
                        output = 4;
                        break;
                    default:
                        output = 1;  //default to red if there is a mistake
                        break;
                }
            } else {
              //if the user pushes the x on the dialog display another dialog
              return pickColor();
            }

              Gui.chosingColor = false;  //allow the game to continue

              return output;
        }

        /*
         * this calls the animation function for placing a card
         * player = the number of the player to deal cards from
         * card = the card to deal
         * index = the position in the players hand to take the card from only
used if player=0
         */
    public void placeCardAnimation(int player, int card, int index) {
        Dimension cardSize = playerCardSize;

          JLabel placeAnimLabel = new JLabel();
        placeAnimLabel.setIcon(new
ScaledImageIcon(cardImages.getCardImage(Rules.getCardType(card)),
deckSize.height));
        placeAnimLabel.setSize(deckSize);
        placeAnimLabel.setName(""+card);
          animation.add(placeAnimLabel);

        Point start = playersText[player].getLocation();

        //selects which location in the hand to take from
        if (player == 0) {
              start.x = start.x-playerCardSize.width*index;
              start.y = masterPanel.getHeight()-playerCardSize.height;
        }

        //sets the endpoint to the discard pile
        Point endTemp = new Point(discardLabel.getX()-discardLabel.getWidth()/2 ,
discardLabel.getY()+discardLabel.getHeight()/2);
        Point end = SwingUtilities.convertPoint(discardLabel, endTemp,
gameFrame);
```

```
      end.x -= deckSize.width/2;
      end.y -= deckSize.height/2;

      //changes both start and end to include the size of the card
      Rectangle startR = new
Rectangle(start.x,start.y,cardSize.width,cardSize.height);
      Rectangle endR = new
Rectangle(end.x,end.y,deckSize.width,deckSize.height);

            System.out.println("starting place animation , for
player="+player+", card="+card);
      doAnimation(startR, endR , 500, placeAnimLabel, player, 1);
   }

      /*
       * this calls the animation function for picking a card
       * player = the number of the player to deal cards to
       * card = the card to deal
       * cardsLeft = used for the recursive counter
       */
   public void pickCardAnimation(int player, int card, int cardsLeft) {
      Dimension cardSize = playerCardSize;

        JLabel pickAnimLabel = new JLabel();
      pickAnimLabel.setIcon(new
ScaledImageIcon(cardImages.getCardImage(Rules.getCardType(card)),
cardSize.height));
      pickAnimLabel.setSize(cardSize);
      pickAnimLabel.setName(""+card);
        animation.add(pickAnimLabel);

        //sets start to the location of the deck
      Point start = new Point(leftPanel.getWidth()+deckButton.getWidth()/2-
deckSize.width/2,topPanel.getHeight()+deckButton.getHeight()/2-
deckSize.height/2);

      //sets end to the players hand
      Point end = new Point(playersText[player].getX()+playerCardSize.width/2,
playersText[player].getY());

      //changes both start and end to include the size of the card
      Rectangle startR = new
Rectangle(start.x,start.y,deckSize.width,deckSize.height);
      Rectangle endR = new
Rectangle(end.x,end.y,cardSize.width,cardSize.height);

            System.out.println("starting pick animation, player="+player+",
ammount="+cardsLeft+", card="+card);
      doAnimation(startR, endR , 500, pickAnimLabel, player, cardsLeft);
   }

   /*
```

```
    * runs the animation using a swingWorker which means it runs on a new
thread
    * start = start coordinates and size of the card
    * end = end coordinates and size of the card
    * time = the duration in ms of the animation
    * label = the JLabel to be moved and scaled
    * playerNum = the players number
    * cardsLeft = counter for recursion it is the number of cards left to deal
if it is <1 it is ignored
    */

     private static void doAnimation(Rectangle start, Rectangle end, int time,
JLabel label, int playerNum, int cardsLeft)  {
           runningAnimation = true;  //blocks game perceding while running
           label.setBounds(start);
           label.setVisible(true);
       SwingWorker<Boolean,Rectangle2D.Double> animate = new
SwingWorker<Boolean,Rectangle2D.Double>() {
           private float xDelta = (float)(end.x-start.x)/time;  //the speed it
moves on the x
           private float yDelta = (float)(end.y-start.y)/time;  //the speed it
moves on the y
           private float widthDelta = (float)(end.width-start.width)/time;
//the speed it moves on the width
           private float heightDelta = (float)(end.height-start.height)/time;
//the speed it moves on the height
           private boolean running = true;  //makes sure that the while loop
dousent start again

           //changes the start coordinates into a double so that the location
can be calculated more accuratly
           private Rectangle2D.Double curLoc = new
Rectangle2D.Double(start.getX(), start.getY(), start.getWidth(),
start.getHeight());
           @Override
           protected Boolean doInBackground() throws Exception
           {
               while (running) {
                boolean xDone = false; //records when it has reached its
target x
                boolean yDone = false; //records when it has reached its
target y

                     if ( ((curLoc.getX()>=end.x) && (xDelta >= 0)) ||
((curLoc.getX()<=end.x) && (xDelta <= 0)) ) {
                      xDone = true;
                     }

                     if ( ( (curLoc.getY()>=end.y) && (yDelta >= 0) ) || (
(curLoc.getY()<=end.y) && (yDelta <= 0) ) ) {
                      yDone = true;
                     }
```

```
                        if (xDone && yDone) {
                        running = false;
                        return running;
                        }

                    if (!xDone) {
                        curLoc.x += xDelta;
                    }
                    if (!yDone){
                        curLoc.y += yDelta;
                    }
                    curLoc.width += widthDelta;
                    curLoc.height += heightDelta;

                        Thread.sleep(1);
                        //System.out.println(curLoc.toString()+",
xdelta="+xDelta+", ydelta="+yDelta);
                        publish(curLoc); //sends the data to proccesing
                }

                return true;
            }

            @Override
            protected void process(List<Rectangle2D.Double> chunks)
            {
                        /*
                         * sets the label to its new coordinates and size
                         * this is called in batches as in it could execute
                         * like this
doInBackground,doInBackground,doInBackground,doInBackground, process,
process,doInBackground, process, process, process
                         * that is why it has to get the next item in the list
                         */
                    Rectangle2D.Double val = (java.awt.geom.Rectangle2D.Double)
chunks.get(chunks.size()-1);
                label.setBounds(val.getBounds());;
            }

            @Override
            protected void done()
            {
                // this method is called when the background
                // thread finishes execution
                label.setVisible(false);
                runningAnimation = false;
                System.out.println("Finished");

                game.gui.drawScene(); //update gui

                    if (cardsLeft>1) {
                            //Recursively calls itself each time with one less card
left until it only has a card left when it finishes
                            game.players[playerNum].pickUpCards(game, playerNum,
cardsLeft-1);
                            return;
                    } else if (Gui.pickingUp) {
                            Gui.pickingUp = false;
                            return;
                    }



                    //makes the game wait for the humans input
                  if (game.getNextPlayer(playerNum) == 0 && Gui.humanNumber == 1)
{
                    if( game.players[0].getSkip() ) {
                            game.players[0].setSkip(false);
                                    game.doTurn(game.getNextPlayer(0));
                            }
                    return;
                  }

                  //Triggers the next robots turn
                  game.doTurn(game.getNextPlayer(playerNum));
            }
        };

        // executes the swingworker on worker thread
        animate.execute();
    }

      //used to read when a player pushes a card
      class Clicklistener implements ActionListener {
              public void actionPerformed(ActionEvent e) {
                      JButton source = (JButton) e.getSource();

                      Sounds.playSound(Sounds.click);

                      Gui.game.gui.buttonPressed =
Integer.parseInt(source.getName());
                      Gui.game.doTurn(0);
              }
      }

      //used to draw the circle representing a player
      @SuppressWarnings("serial")
      class PlayerCirclePaint extends JPanel
      {
              @Override
          protected void paintComponent(Graphics gr) {
                super.paintComponent(gr);
                Graphics2D g = (Graphics2D)gr;
```

```
			Shape ring = createRingShape(50, g.getClipBounds().getHeight()-50, 
50, 5);
			g.setColor(Color.BLACK);
			g.fill(ring);
			g.draw(ring);
		}

	//creates a ring with the specified dimensions
	//this is created by generating two circles and subtracting the smaller 
from the larger
	private Shape createRingShape( double centerX, double centerY, double 
outerRadius, double thickness) {
			Ellipse2D outer = new Ellipse2D.Double(
				centerX - outerRadius,
				centerY - outerRadius,
				outerRadius + outerRadius,
				outerRadius + outerRadius);
			Ellipse2D inner = new Ellipse2D.Double(
				centerX - outerRadius + thickness,
				centerY - outerRadius + thickness,
				outerRadius + outerRadius - thickness - thickness,
				outerRadius + outerRadius - thickness - thickness);
			Area area = new Area(outer);
			area.subtract(new Area(inner));
			return area;
		}
	}


	//used to scale the images to the specified height
	@SuppressWarnings("serial")
	static class ScaledImageIcon extends ImageIcon {
		ScaledImageIcon(Image image, int height) {
		 super(image.getScaledInstance(-1, height, Image.SCALE_SMOOTH));
		}
	}
}
```

```
class Rules {
	//checks if the specified card can be placed on top of the topCard
	public static boolean canDiscard(int trialCard, int topCard) {
		char top[] = getCardType(topCard);
		char trial[] = getCardType(trialCard);
		if (top[0] == trial[0]) {
			return true;
		}
		if (top[1] == trial[1]){
			return true;
		}
		if (trial[1] == 'W') {
			return true;
		}
		return false;
	}

	//returns true if the card is a special card
	//returns false otherwise
	public static boolean isSpecial(int selectedCard) {
		char card[] = getCardType(selectedCard);
		switch (card[0]) {
			case 'F':
				return true;
			case 'C':
				return true;
			case 'T':
				return true;
			case 'S':
				return true;
			case 'R':
				return true;
			default:
				return false;
		}
	}

	//checks if the card is special and does all the appropriate actions if
it is
	public static void doSpecial(Game game, int selectedCard, int curPlayer)
{
		char card[] = getCardType(selectedCard);
		int output = 0;
		switch (card[0]) {
			case 'F':
				//wild +4 card
				//prevents the player taking a go while picking up cards
or color

				Gui.pickingUp = true;

	game.players[game.getNextPlayer(curPlayer)].setSkip(true);//skips the
next users turn
				switch (game.players[curPlayer].pickColor(game)) {
					case 1:
						output = 108;
						break;
					case 2:
						output = 109;
```

```
                                        break;
                                case 3:
                                        output = 110;
                                        break;
                                case 4:
                                        output = 111;
                                        break;
                        }
                        //changes the color of the wild card so you can see the
chosen color
                        game.deck.setTopDiscard(output);
                        game.gui.drawScene();

        game.players[game.getNextPlayer(curPlayer)].pickUpCards(game,
game.getNextPlayer(curPlayer), 4);

        game.players[game.getNextPlayer(curPlayer)].setSkip(false);
                        break;
                case 'C':
                        //wild color changing card
                        switch (game.players[curPlayer].pickColor(game)) {
                                case 1:
                                        output = 112;
                                        break;
                                case 2:
                                        output = 113;
                                        break;
                                case 3:
                                        output = 114;
                                        break;
                                case 4:
                                        output = 115;
                                        break;
                        }
                        //changes the color of the wild card so you can see the
chosen color
                        game.deck.setTopDiscard(output);

                        game.gui.drawScene();
                        break;
                case 'T':
                        //+2 card
                        Gui.pickingUp = true;

        game.players[game.getNextPlayer(curPlayer)].pickUpCards(game,
game.getNextPlayer(curPlayer), 2);
                        break;
                case 'S':
                        //skip card

        game.players[game.getNextPlayer(curPlayer)].setSkip(true);//skips the
next users turn
```

```
					break;
				case 'R':
					//reverse card
					if (Gui.playersNumber != 2) {
						game.toggleReverse();
					}
					break;
			}
		}

		/* Reverts a colored wild card back to a wild card */
		public static void revertWildCard(Deck deck, int topCard) {
			char card[] = getCardType(topCard);
			switch (card[0]) {
				case 'F':
					//wild +4 card
					deck.setTopDiscard(0);
					break;
				case 'C':
					deck.setTopDiscard(4);
					break;
			}
		}

		//returns the score of the specified card according to the rule of uno
		public static int getCardVal(int card) {
			char [] cardType = getCardType(card);
			switch (cardType[0]) {
				case 'F':
					return 50;
				case 'C':
					return 50;
				case 'T':
					return 20;
				case 'S':
					return 20;
				case 'R':
					return 20;
				default:
					return Integer.valueOf(cardType[0]);
			}
		}

		//lookup table to convert the unique number of the card to a card type
and color
		public static char[] getCardType(int cardIndex) {
			final char cards[][] = {
					{'F', 'W'}, {'F', 'W'}, {'F', 'W'}, {'F', 'W'}, {'C',
'W'}, {'C', 'W'}, {'C', 'W'}, {'C', 'W'},

					{'0', 'R'}, {'1', 'R'},{'2', 'R'},{'3', 'R'},{'4',
'R'},{'5', 'R'},{'6', 'R'},{'7', 'R'},{'8', 'R'},{'9', 'R'},
                              {'1', 'R'},{'2', 'R'},{'3', 'R'},{'4', 'R'},{'5',
'R'},{'6', 'R'},{'7', 'R'},{'8', 'R'},{'9', 'R'},
                              {'S', 'R'},{'S', 'R'},{'T', 'R'},{'T', 'R'},{'R',
'R'},{'R', 'R'},

                              {'0', 'Y'},{'1', 'Y'},{'2', 'Y'},{'3', 'Y'},{'4',
'Y'},{'5', 'Y'},{'6', 'Y'},{'7', 'Y'},{'8', 'Y'},{'9', 'Y'},
                              {'1', 'Y'},{'2', 'Y'},{'3', 'Y'},{'4', 'Y'},{'5',
'Y'},{'6', 'Y'},{'7', 'Y'},{'8', 'Y'},{'9', 'Y'},
                              {'S', 'Y'},{'S', 'Y'},{'T', 'Y'},{'T', 'Y'},{'R',
'Y'},{'R', 'Y'},

                              {'0', 'G'},{'1', 'G'},{'2', 'G'},{'3', 'G'},{'4',
'G'},{'5', 'G'},{'6', 'G'},{'7', 'G'},{'8', 'G'},{'9', 'G'},
                              {'1', 'G'},{'2', 'G'},{'3', 'G'},{'4', 'G'},{'5',
'G'},{'6', 'G'},{'7', 'G'},{'8', 'G'},{'9', 'G'},
                              {'S', 'G'},{'S', 'G'},{'T', 'G'},{'T', 'G'},{'R',
'G'},{'R', 'G'},

                              {'0', 'B'},{'1', 'B'},{'2', 'B'},{'3', 'B'},{'4',
'B'},{'5', 'B'},{'6', 'B'},{'7', 'B'},{'8', 'B'},{'9', 'B'},
                              {'1', 'B'},{'2', 'B'},{'3', 'B'},{'4', 'B'},{'5',
'B'},{'6', 'B'},{'7', 'B'},{'8', 'B'},{'9', 'B'},
                              {'S', 'B'},{'S', 'B'},{'T', 'B'},{'T', 'B'},{'R',
'B'},{'R', 'B'},

                              {'F', 'R'}, {'F', 'Y'}, {'F', 'G'}, {'F', 'B'}, {'C',
'R'}, {'C', 'Y'}, {'C', 'G'}, {'C', 'B'},

                              {'B', 'W'}
                  };
                  return cards[cardIndex];
            }
}




class Deck {
      final private int cardMax = 108; //size of the deck
      private LinkedList<Integer> deck = new LinkedList<Integer>();
      //stores the deck
      private LinkedList<Integer> discardPile = new LinkedList<Integer>();
      //stores the discardPile
      private Random rand;

      public Deck() {
            rand = new Random();

            /*
```

```
		 * goes through each location in the deck and selects a random uno 
card to place there
		 * if that card is already in the array then keep selecting
		 * a new random card until the selected card is not in the array
		 * 
		 * could be more efficient but only triggers when the game starts
		 */
		
		for(int i=0; i<cardMax; i++) {
			int random = rand.nextInt(cardMax);
			while (deck.contains(random)) {
				random = rand.nextInt(cardMax);
			}
			deck.add(random);
		}
	}
	
	//deals a card
	//returns the top card from the deck
	//after removing it from the deck
	public int dealCard() {
		if (deck.isEmpty()) {
			deckEmpty();
		}
		return deck.pop();
	}
	
	/*
	 * deals numCards amount of cards
	 * same as calling dealCards multiple times
	 * 
	 * returns a linkedList of all the cards dealt
	 */
	
	public LinkedList<Integer> dealCards(int numCards) {
		if (deck.size()<numCards) {
			deckEmpty();
		}
		LinkedList<Integer> output = new LinkedList<Integer>();
		for(int i=0; i<numCards; i++) {
			output.add(deck.pop());
		}
		return output;
	}
	
	//gets the size of the deck
	public int getDeckSize() {
		return deck.size();
	}
	
	public void printDeck() {
		System.out.print("Deck = [");
		for (int i=0; i<deck.size(); i++) {
			System.out.print(Rules.getCardType(deck.get(i))[0]);
			System.out.print(Rules.getCardType(deck.get(i))[1]);
			System.out.print(", ");
		}
		System.out.println("]");
	}

	public void printDiscard() {
		System.out.print("Discard = [");
		for (int i=0; i<discardPile.size(); i++) {
			System.out.print(Rules.getCardType(discardPile.get(i))[0]);
			System.out.print(Rules.getCardType(discardPile.get(i))[1]);
			System.out.print(", ");
		}
		System.out.println("]");
	}

	//shuffles the deck
	public void shuffle() {
		Collections.shuffle(deck);
	}

	/*
	 * to be called if the deck runs out
	 * it will move the discard pile to the deck
	 * and then shuffle the deck 10 times
	 *
	 * if there still is not enough cards
	 * then it will add in a second deck
	 */
	public void deckEmpty() {
		while(discardPile.size()>1) {
			deck.add(discardPile.removeLast());
		}
		if (deck.size()<=4) {
			//adds a second deck in case of +4 card
			for(int i=0; i<cardMax; i++) {
				int random = rand.nextInt(cardMax);
				while (deck.contains(random)) {
					random = rand.nextInt(cardMax);
				}
				deck.add(random);
			}
		}

		for(int i=0; i<10; i++) {
			shuffle();
		}
	}

	//add the given card to the discard pile
	public void discardCard(int card) {
		discardPile.addFirst(card);
	}

	//returns the card at the top of the discard pile
	public int getTopDiscard() {
		return discardPile.getFirst();
	}

	//changes the card at the top of the discard to the specified card
	//used to set a wild card to a colored variant
	public void setTopDiscard(int card) {
		discardPile.set(0, card);
	}
}
```

```
@SuppressWarnings("serial")
class Hand extends LinkedList<Integer>{

	//Initializes the hand with startCards number
	//of randomly dealt cards
	public Hand(Deck deck, int startCards) {
		for (int i=0; i<startCards; i++) {
			super.add(deck.dealCard());
		}
	}

	//prints the content of the hand to console
	//used for debugging
	public void printHand() {
		System.out.print("Your hand = [");
		for (int i=0; i<super.size(); i++) {
			System.out.print(Rules.getCardType(super.get(i))[0]);
			System.out.print(Rules.getCardType(super.get(i))[1]);
			System.out.print(", ");
		}
		System.out.println("]");
	}

	//add a card to the end of the hand
	//returns the card it adds
	public int addCard(int card) {
		super.add(card);
		return card;
	}
	//adds multiple cards and returns all the cards added
	/*public LinkedList<Integer> addCards(LinkedList<Integer> cards) {
		int size = cards.size();
		LinkedList<Integer> output = new LinkedList<Integer>();
		
		for (int i=0; i<size; i++) {
			int curCard = cards.pop();
			output.add(curCard);
			super.add(curCard);
		}
		return output;
	}*/
}
```

```
class Player {
	protected Hand hand;		//stores the players hand
	protected int playerScore;	//stores the total score of the player
	protected boolean skip;	//identifies if the player should be skipped
	
	public Player(Deck deck, int startCards) {
		hand = new Hand(deck, startCards);
	}
	
	//place holder for the definitions in the sub classes
	public int placeCard(Game game, int index) {
		return -500;
	}
	
	/*
	 * deals a card to the players hand then plays the animation
	 * if the user is a robot it deals with the card face down
	 * 
	 * playerNum is the player who should pickup
	 */
	public void pickUpCard(Game game, int playerNum) {
		int card = hand.addCard(game.deck.dealCard());
		if (playerNum != 0 && Gui.humanNumber == 1) {
			card = 116;
		}
		game.gui.pickCardAnimation(playerNum, card, 1);
	}
	
	/*
```

```
	 * a recursive method which will deal cardsLeft number of cards then 
plays the
	 * animation if the user is a robot deals with the card face down and 
repeats
	 * this until cardsLeft == 1
	 * 
	 *  playerNum is the player who should pickup
	 *  cardsLeft is the number of cards to pickup
	 */
	public void pickUpCards(Game game, int playerNum, int cardsLeft) {
		int card = hand.addCard(game.deck.dealCard());
		if (playerNum != 0 && Gui.humanNumber == 1) {
			card = 116;
		}
		game.gui.pickCardAnimation(playerNum, card, cardsLeft);
	}
	
	//place holder for the definitions in the sub classes
	public int pickColor(Game game) {
		return 1;
	}
	
	//returns the size of the players hand
	public int getHandSize() {
		return hand.size();
	}
	
	//calculates the players score
	public int getHandScore() {
		int output = 0;
		for (int i=0; i<hand.size(); i++) {
			output += Rules.getCardVal(hand.get(i));
		}
		return output;
	}
	
	//calculates the players score
	public Hand getHand() {
		return hand;
	}
	
	//gets the first occurrence of the card in the players hand
	public int getHandIndex(int card) {
		return hand.indexOf(card);
	}
	
	//gets the players score
	public int getPlayerScore() {
		return playerScore;
	}
	
	//returns wether the player should be skipped
	public boolean getSkip() {
		return skip;
	}

	//adds score to the players score
	public void addPlayerScore(int score) {
		playerScore += score;
	}

//	sets the player score to score
	public void setPlayerScore(int score) {
		playerScore = score;
	}

	public void setSkip(boolean input) {
		skip = input;
	}
}


class HumanPlayer extends Player {

	public HumanPlayer(Deck deck, int startCards) {
		super(deck, startCards);
	}


	/*
	 * places the card at the specified index
	 *
	 * it removes the card at the specified index from the players hand
	 * and adds it to the discard pile then it calls the animation
	 *
	 * if the specified index is -1 then it picks up
	 *
	 * if the index is not valid then it returns -500
	 *
	 * returns -244 = skip
	 * returns -500 = pickup
	 * returns -501 = against rules
	 */
	public int placeCard(Game game, int index) {
		if (skip) {
			skip = false;
			return -244;
		}

		int topCard = game.deck.getTopDiscard();
		if (index==-1) {
			pickUpCard(game, 0);
			return -500;
		} else if (Rules.canDiscard(hand.get(index), topCard)) {
				//reverts special cards in the discard pile back to there 
original card
				if (Rules.isSpecial(topCard)) {
					Rules.revertWildCard(game.deck, topCard);
				}
				
				//can play card will place
				int selectedCard = hand.remove(index);
				game.deck.discardCard(selectedCard);
				game.gui.placeCardAnimation(0, selectedCard, (hand.size()+1)-
(index+1));
				return selectedCard;
			} else {
				//against rules will exit
				System.out.println("Against the rules");
				return -501;
			}
		}
		
		/*
		 * asks the use to chose a color and returns the color number chosen
		 * 
		 * returns
		 * 1 = Red
		 * 2 = Yellow
		 * 3 = Green
		 * 4 = Blue
		 */
		public int pickColor(Game game) {
			int choice = game.gui.pickColor();
			game.doTurn(game.getNextPlayer(0));
			return choice;
		}
}








class RobotPlayer extends Player {
		public RobotPlayer(Deck deck, int startCards) {
			super(deck, startCards);
			
			//just used for testing
//			hand.set(0, 54);
//			for(int i=1; i<startCards; i++) {
//				hand.set(i, 41+i);
//			}
```

```
	}

	/*
	 * decides which card the robot should place
	 * 
	 * it choose the first card in its hand which can be placed
	 * according to the rules of uno
	 * it then removes the card from the players hand
	 * and adds it to the discard pile then it calls the animation
	 * 
	 * if it is unable to find any valid cards then it will pick up one
	 * 
	 * playerNum is the player who is playing
	 * 
	 * returns -244 = skip
	 * returns -500 = pickup
	 */
	public int placeCard(Game game, int playerNum) {
		if (skip) {
			skip = false;
			return -244;
		}
		int topCard = game.deck.getTopDiscard();
		for (int index = 0; index<hand.size(); index++) {
			if (Rules.canDiscard(hand.get(index), topCard)) {
				//reverts special cards in the discard pile back to there 
original card
				if (Rules.isSpecial(topCard)) {
					Rules.revertWildCard(game.deck, topCard);
				}
				
				int selectedCard = hand.remove(index);
				game.deck.discardCard(selectedCard);
				game.gui.placeCardAnimation(playerNum, selectedCard, 
index);
				
				return selectedCard;
			}
		}
		pickUpCard(game, playerNum);
		return -500;
	}

	/*
	 * Chooses a color and returns the color number chosen
	 * it will chose the color of most of the cards in its hand
	 * 
	 * returns
	 * 1 = Red
	 * 2 = Yellow
	 * 3 = Green
	 * 4 = Blue
	 */
	public int pickColor(Game game) {
		int color[] = new int[4];
		char cur = ' ';
		for (int c=0; c<4; c++) {
			switch (c) {
				case 0:
					cur = 'R';
					break;
				case 1:
					cur = 'Y';
					break;
				case 2:
					cur = 'G';
					break;
				case 3:
					cur = 'B';
					break;
			}
			for (int index = 0; index<hand.size(); index++) {
				if (Rules.getCardType(hand.get(index))[1] == cur) {
					color[c] += 1;
				}
			}
		}
		int largestC = 0;
		int largestCVal = 0;
		for (int c=0; c<4; c++) {
			if (color[c]>largestCVal) {
				largestCVal = color[c];
				largestC = c;
			}
		}
		return largestC+1;
	}
}
```

```
class Images {
	//the official uno colors
	private final Color RED = new Color(237,27,36);
	private final Color YELLOW = new Color(255,222,21);
	private final Color GREEN = new Color(79,170,67);
	private final Color BLUE = new Color(0,114,187);
	//the ammount of pixels to crop off the edges of the images
	private final int cropSize = 2;

	//creates a local instance of these classes so that i can access
	//the local variables which hold some of the images
	ImagesWild wildCards = new ImagesWild();
	ImagesNumbers numberCards = new ImagesNumbers();
	ImagesOther otherCards = new ImagesOther();


	/*
	 * gets the specified byte array and then converts it to a image
	 * then changes it into the specified color,
	 * Then it fills in all the static colors which where previously
converted to grayscale,
	 * then returns the final image
	 */
	public Image getCardImage(char card[]) {
		byte [] cardImg;
		boolean isMultiColored = false;
		switch (card[0]) {
			case '0':
				cardImg = numberCards.uno0;
				break;
			case '1':
				cardImg = numberCards.uno1;
				break;
			case '2':
				cardImg = numberCards.uno2;
				break;
			case '3':
				cardImg = numberCards.uno3;
				break;
			case '4':
				cardImg = numberCards.uno4;
				break;
			case '5':
				cardImg = ImagesNumbers.uno5;
				break;
			case '6':
				cardImg = ImagesNumbers.uno6;
				break;
			case '7':
				cardImg = ImagesNumbers.uno7;
				break;
			case '8':
				cardImg = ImagesNumbers.uno8;
				break;
			case '9':
				cardImg = ImagesNumbers.uno9;
				break;
			case 'R':
				cardImg = ImagesOther.unoR;
				break;
			case 'S':
				cardImg = ImagesOther.unoS;
				break;
			case 'T':
				cardImg = otherCards.unoT;
				break;
			case 'C':
				cardImg = ImagesWild.unoC;
				isMultiColored = true;
				break;
			case 'F':
				cardImg = wildCards.unoF;
				isMultiColored = true;
				break;
			case 'B':
				cardImg = ImagesOther.unoB;
				isMultiColored = true;
				break;
			default:
				cardImg = null;
				break;
		}

		/* prints out supported image formats */
		/*
		 * for (String format : ImageIO.getReaderFormatNames()) {
		 * System.out.println("format = " + format); } for (String format :
		 * ImageIO.getReaderMIMETypes()) { System.out.println("format = " +
format); }
		 */

		BufferedImage img = null;
		try {
			/*
			 * TODO make compatible with jdk 8
			 * only works with java jdk version >= v9
			 *
			 */
			img = ImageIO.read(new ByteArrayInputStream(cardImg));
		} catch (IOException e) {
			e.printStackTrace();
		}

		Color cardColor;
		switch (card[1]) {
			case 'R':
				cardColor = RED;
				break;
			case 'Y':
```

```
                        cardColor = YELLOW;
                        break;
                   case 'G':
                        cardColor = GREEN;
                        break;
                   case 'B':
                        cardColor = BLUE;
                        break;
                   case 'W':
                        cardColor = Color.BLACK;
                        break;
                   default:
                        return img;
              }
              if (isMultiColored) {
                   return changeColorSpecial(img, cardColor);
              } else {
                   return changeColor(img, cardColor);
              }

         }

         /*
          * uses the specified img as a template to create a new image in the 
specified color
          * Then it fills in all the static colors which where previously 
converted to grayscale,
          * then returns the image
          */
         private BufferedImage changeColorSpecial(BufferedImage img, Color newImg) 
{
              BufferedImage output = new BufferedImage(img.getWidth()-cropSize*2, 
img.getHeight()-cropSize*2, 1);
              final int newRGB = newImg.getRGB();
              for (int x = cropSize; x < img.getWidth()-cropSize; x++) {
               int outX = x-cropSize;
                 for (int y = cropSize; y < img.getHeight()-cropSize; y++) {
                   int outY = y-cropSize;
                   Color pixel = new Color(img.getRGB(x, y));
                   float hsbVal[] = Color.RGBtoHSB(pixel.getRed(), 
pixel.getGreen(),  pixel.getBlue(), null);
                    if (hsbVal[2] < 0.5) {
                        output.setRGB(outX, outY, newRGB);
                    } else if (hsbVal[2] > 0.5 && hsbVal[2] <= 0.65) {
                        output.setRGB(outX, outY, YELLOW.getRGB());
                    } else if (hsbVal[2] > 0.65 && hsbVal[2] <= 0.74) {
                        output.setRGB(outX, outY, RED.getRGB());
                    } else if (hsbVal[2] > 0.74 && hsbVal[2] <= 0.805) {
                        output.setRGB(outX, outY, GREEN.getRGB());
                    } else if (hsbVal[2] > 0.805 && hsbVal[2] <= 0.86) {
                        output.setRGB(outX, outY, BLUE.getRGB());
                    } else if (hsbVal[2] > 0.86 && hsbVal[2] <= 0.935) {
                        output.setRGB(outX, outY, Color.BLACK.getRGB());
                    } else {
                        output.setRGB(outX, outY, Color.WHITE.getRGB());
                    }
                }
            }
            return output;
        }

        //uses the specified img as a template to create a new image in the
specified color
        //returns the image in the specified color
        private BufferedImage changeColor(BufferedImage img, Color newImg) {
             BufferedImage output = new BufferedImage(img.getWidth()-cropSize*2,
img.getHeight()-cropSize*2, 1);
            final int newRGB = newImg.getRGB();
            for (int x = cropSize; x < img.getWidth()-cropSize; x++) {
             int outX = x-cropSize;
               for (int y = cropSize; y < img.getHeight()-cropSize; y++) {
                  int outY = y-cropSize;
                  Color pixel = new Color(img.getRGB(x, y));
                  float hsbVal[] = Color.RGBtoHSB(pixel.getRed(),
pixel.getGreen(),  pixel.getBlue(), null);
                   if (hsbVal[2] < 0.8) {
                       output.setRGB(outX, outY, newRGB);
                   } else {
                        output.setRGB(outX, outY, Color.WHITE.getRGB());
                   }
               }
            }
            return output;
        }
}

/*
 * stores the images of the cards in a signed byte array
 * this is required as the code has to be delivered in a single .java file not
a .jar
 * there is a size limit of 65535 bytes for a static in a class
 * and another limit of 65535 bytes for the local variables of a class
 *
 * this is why i use both static and local varibles in the images classes so
that
 * i can take advantage of both those limits
 */

//the two wild cards
class ImagesWild {
//all images have been removed to reduce the size of the pdf report
        System.exit(1);
        public final byte [] unoF = {};
        public static final byte [] unoC = {};
```

```
}

//the numbered cards from 0-9
class ImagesNumbers{
	public static final byte [] uno9 = {};
	public static final byte [] uno8 = {};
	public static final byte [] uno7 = {};
	public static final byte [] uno6 = {};
	public static final byte [] uno5 = {};
	public final byte [] uno4 = {};
	public final byte [] uno3 = {};
	public final byte [] uno2 = {};
	public final byte [] uno1 = {};
	public final byte [] uno0 = {};
}

//the back of the cards, reverse, skip and +2 cards
class ImagesOther{
	public static final byte [] unoB = {};
	public static final byte [] unoR = {};
	public static final byte [] unoS = {};
	public final byte [] unoT = {};
}




class Sounds {
	/*
	 * these are the three sounds which are stored in byte arrays
	 * the audio is in .mid format so that it takes the minimum amount space
	 * this means that it is storing only the notes to play and the 
instruments
	 * 
	 * all the sounds where generated using 
	 * winner = pizza time 			- 
https://onlinesequencer.net/1496934#t36 - modified
	 * loser  = Beethoven 5th 	 	- https://onlinesequencer.net/275900
	 * click  = drumkit castanets 	- https://onlinesequencer.net/
	 */
	public static final byte [] winner = {};
	public static final byte [] loser = {};
	public static final byte [] click = {};
	
	//plays the sound from the given byte array
	public static void playSound(byte [] sound) {
		if(Gui.sound) {
			try {
			 Sequencer sequencer = MidiSystem.getSequencer(); // Get the 
default Sequencer
			 if (sequencer==null) {
```

```
                        System.err.println("Sequencer device not supported");
                        return;
                    }
                    sequencer.open(); // Open device
                    Sequence sequence = MidiSystem.getSequence(new
ByteArrayInputStream(sound));
                    sequencer.setSequence(sequence); // load it into sequencer
                    sequencer.start();  // start the playback
                } catch (MidiUnavailableException | InvalidMidiDataException |
IOException ex) {
                    ex.printStackTrace();
                }
            }
        }
}
```