Projekt **Pogramowanie obiektowe (Java)**

Wydział Elektrotechniki Automatyki i Informatyki

Politechnika Świętokrzyska

Studia: **Stacjonarne I stopnia**	Kierunek: **Informatyka**
Temat projektu: **Gra UNO**	Grupa: **2ID11A**
	1. Yevhenii Stavnychyi 2. Szymon Paweł Baran

1 Ogólny opis projektu

1.1 Co to jest Uno?

UNO – amerykańska gra karciana, zbliżona koncepcyjnie do Pan. Do gry w UNO używa się specjalnej talii kart.

Talia liczy w sumie 108 kart i składa się z:

Kart zwykłych:

- 19 czerwonych kart ponumerowanych od 0 do 9 (jedna karta z numerem 0 oraz po dwie karty z numerami od 1 do 9)
- 19 zielonych kart ponumerowanych od 0 do 9 (jedna karta z numerem 0 oraz po dwie karty z numerami od 1 do 9)
- 19 niebieskich kart ponumerowanych od 0 do 9 (jedna karta z numerem 0 oraz po dwie karty z numerami od 1 do 9)
- 19 żółtych kart ponumerowanych od 0 do 9 (jedna karta z numerem 0 oraz po dwie karty z numerami od 1 do 9)

oraz kart funkcyjnych:

- 8 kart stopu (Skip) po dwie z każdego koloru
- 8 kart zmiany kierunku (Revers) po dwie z każdego koloru
- 8 kart +2 (Draw two) po dwie z każdego koloru
- 4 czarne karty +4 ze zmianą koloru (Wild +4)
- 4 czarne karty zmiana koloru (Wild)

1.2 Zasady gry

Graczy może być od 2 do 4. Na początku gry rozdaje się po 6 kart każdemu graczowi i jedną z talii kładzie się na środek. Gracz musi dopasować swoją kartę numerem, kolorem lub symbolem do odkrytej karty. Jeżeli gracz nie posiada żadnej karty pasującej do tej odkrytej, musi pociągnąć kartę z talii. Jeśli wyciągnięta karta pasuje do odkrytej, jeszcze w tej samej kolejce gracz może ją dołożyć. Jeżeli nie - ruch ma kolejny gracz. Nie ma przymusu w dokładaniu kart. Gracz, który jako pierwszy zostanie bez kart, wygra(2 gracz na grugim miejscu i td.).

W talii są także karty specjalne takie jak Skip, Reverse, Draw two, Wild + 4, Wild.

- Skip - następny gracz traci (stoi) kolejkę

- Reverse - karta zmieniająca kierunek gry
- Draw two - następny gracz bierze dwie karty
- Wild + 4 - zagrywający kartę deklaruje zmianę koloru na dowolnie przez siebie wybrany, następny gracz bierze 4 karty.
- Wild - zagrywający kartę deklaruje zmianę koloru na dowolnie przez siebie wybrany (jeden z kolorów dostępnych w grze).

Wygrywa ten, kto pierwszy pozbędzie się wszystkich kart.

1.3 Zastosowane technologie

Do wykonania projektu zastosowano środowisko programistyczne Eclipse IDE.

1.4 Uruchamianie

Aplikację uruchamia się przez program Eclipse IDE.

2 Pakiety aplikacji

2.1 Pakiet Uno

2.1.1 UnoApp.java

UnoApp.java jest programem pozwala na uruchomienie aplikacji.

Ten program jest połączony z innymi programami:

- Uno.game.GameManagement

Ten program zawiera implementację aplikacji i przeciążoną metodę startThread() klasy GameManagement.

2.1.2 Deal.java

Deal.java jest programem, który zawarte są tam metody umożliwiającą przyszłe „play again”.

Biblioteki użyte w programie Deal.java:

- Java.util.ArrayList

Ten program zawiera class Deal który dziedziczy CardHadler, tablicę kart(w ręce) i inne metody: metodę addCard, removeCard, getCard, getSize, printArray, getLast.

```
public void addCard(CardHandler deck) {
		hand.add(0,deck.getLast());
		deck.removeLast();
	}

	public void addCard(Uno.Card addCard) {
		hand.add(addCard);
	}
```

- dodawana jest karta do talli gracza (przez talie lub przez dodanie karty).

```
	public int getSize(){
	  return hand.size();
    }
```

- zwracana jest ilość kart w talii.

2.1.3 Player.java

Player.java jest programem, który inicjuje działanie gry gracza.

Biblioteki użyte w programie Player.java:

- java.util.InputMismatchException;
- java.util.Random;
- java.util.Scanner;

Ten program zawiera metody które realizują działania gracza: enemyGame, getEnemyChoice, findWild, findDTwo, findSkip, findReverse, findDrawFour, findPlayable, hasWild, hasSkip, hasReverse, hasDrawTwo, hasDrawFour, hasPlayable, wildEnemyColor, checkDraw, draw2, draw4, getCardNumber, printDiscard, wildColor, printHand.

To była implementacja gry w konsoli, która nas wzorowała jak ma wyglądać ogólnie gra.

2.1.4 CardHandler.java

CardHadler.java jest programem, który zawarte są tam talia kart i metody.

Biblioteki użyte w programie CardHandler.java:

- Java.util.ArrayList;
- Java.util.Collections;

Ten program zawiera własną tablicę kart i inne metody: shuffleDeck, getLast, addCard, printDeck, getSize, removeLast.

```
public Card getLast() {

		if(cards.size()==0) return null;
		return cards.get(cards.size() - 1);
	}
```

- otrzymywana jest ostatnia karta ze talii, jeżeli talia jest pusta jest otrzymywane null.

```
public void removeLast() {
		if(cards.size()!=0)
			cards.remove(cards.size() - 1);
	}
```

- usuwana jest ostatnia karta ze talii.

2.1.5 Card.java

Card.java jest programem do obsługi kart(wagi i kolor).

Na tym programie nie użyto żadnej biblioteki.

Ten program zawiera konstruktor klasy Card i jej inne metody: numRet, colorRet, toString, clone.

```
 public Card(int cNumber, char cColor){
		cardNumber = cNumber;
		cardColor = cColor;
	}
```

- Tworzona jest karta o danym numerze i kolerze (litera w char).

```
 public int numRet(){
		return cardNumber;
	}
	public char colorRet(){
```

```
        return cardColor;
    }
```

- Pierwsza funckja zwraca numer karty druga kolor.

2.2 Pakiet Uno.game

2.2.1 GameManagement.java

GameManagement.java jest programem, który inicjuje okno aplikacji które jest w Uno.window.AppWindow.

Biblioteki użyte w programie GameManagement.java:

- Java.awt.Font;
- Java.awt.FonrMetrics;
- Java.awt.Graphics;
- Java.awt.image.BufferStrategy;
- Java.awt.image.BufferedImage;

Ten program jest połączony z innymi programami:

- Uno.game.hendler.GameHandlerer;
- Uno.game.input.*;
- Uno.window.*;
- Uno.window.screens.*;

Ten program zawiera klasę która odpowiada za przechowywanie rzeczy które są potrzebne do naszej aplikacji takich jak: listener do klawiatury,myszki,okna menu,connection itp.

```
public synchronized void startThread()
      {
            setUp();
            if(live==false)
            {
                  live=true;
                  thread = new Thread(this);
                  thread.start();
```

```
        }
    }

    public synchronized void stopThread()
    {
        if(live==true)
        {
            live=false;
            window.exitWindow();
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        }
```

- pierwsza funkcja zaczyna a druga zakańcza działanie progromu (okna).

```
public void run() {
        int FPS=60;
        double timePerTick = 1000000000/FPS;
        double DELTA = 0;
        long CURRENT;
        long LAST = System.nanoTime();

        long TIMER = 0;
        long TICKS = 0;

        while(live==true)
        {
            CURRENT = System.nanoTime();
            DELTA += (CURRENT-LAST)/timePerTick;
            TIMER += CURRENT-LAST;
            LAST=CURRENT;
```

```
                if(DELTA >=1)
                {
                        render();
                        update();
                        TICKS++;
                        DELTA--;
                }
                if(TIMER>=1000000000)
                {
                        System.out.println("FPS:"+TICKS);
                        TICKS=0;
                        TIMER=0;
                }
        }
        stopThread();
    }
```

- Główna pętla gry/programu. Sprawia że program będzie chodził najprawdopodnobnie w tym samym czasie na wszystkich komputerach, wywołuje także metody render i update, których działanie bedzie pozniej wymienione.

```
private void render()
    {
        buffer = window.canvasRet().getBufferStrategy();
        if(buffer == null)
        {
            window.canvasRet().createBufferStrategy(3);
            return;
        }

        g = buffer.getDrawGraphics();

        g.clearRect(0, 0, WIDHT, HEIGHT);

        g.drawImage(backround,0,0,WIDHT,HEIGHT,null);
```

```
            if(Screens.getScreen()!=null)
                  Screens.getScreen().render(g);


            buffer.show();
            g.dispose();
      }
```

- render na początku czyści ekran (płótno) a później rysuje komponenty teraźniejszego screena/ekranu (Screens) za pomoca grafik.

```
private void update()
      {           if(Screens.getScreen()!=null)
                  Screens.getScreen().update();
      }
```

- Uaktualnia dane ekranu (Screens) dopóki on nie jest null

2.3 Uno.game.handler

2.3.1 GameHadlerer.java

GameHandlerer jest programem, który zawiera klasę ktorą używa jako medium komunikacji miedzy inymi klasami(np. Aby zdobyć zmienna od jednej klasy i wykorzystać jej w drugiej).

Biblioteki użyte w programie GameHandlerer:

- Java.awt.Font;
- Java.awt.FontMetrics;
- Java.awt.image.BufferedImage;

Ten program jest połączony z innimy programami:

- Uno.Card;
- Uno.Deal;
- Uno.game.GameManagement;
- Uno.game.input.KeyboardMangaer;

- Uno.game.input.MouseManager;
- Uno.net.client.GameClient;
- Uno.net.server.GameServer;
- Uno.window.screens.ConnectionScreen;
- Uno.window.screens.Screens;
- Uno.window.screens.Screens;
- Uno.window.ui.manager.UIManagerS;
- Uno.window.CardLoader;
- Uno.window.ImagesLoader;

```
public void CreateServer()
	{
		if(gameServer==null)
		{
			gameServer = new GameServer(1331);
			gameServer.start();
		}

	}

	public void CreateClient(String ip)
	{

		if(gameClient==null)
		{
			gameClient = new GameClient(ip,1331,this);
			gameClient.start();
			gameClient.SendMessageToServer("1");
		}

	}
```

- pierwsza funckja tworzy i uruchamia server, druga tworzy i uruchamia klienta oraz wysyła informacje do servera (o podanym ip) o logowaniu. Obie funckje przyjmują port 1331 i nie uruchomią się jeżeli istenieje już ich instacja. Klientowi jest także dawany GameHandlerer jako ostatni argument.

```
public void startGameServer()
	{


		if(gameServer!=null)
		{
			gameServer.startGame();
			gameMan.newGame();


		}


	}
```

- funkcja ta mówi serverowi by zaczoł nową rozgrywkę oraz GameManager-owi by zrobił nowe okno gry(GameScreen).

```
public void setUIM(UIManagerS uim)
	{
		gameMan.getMouseManager().setUIM(uim);
		gameMan.getKeyManager().setUIM(uim);
	}
```

- funckja ta jest używana przy zmianie Screen-a i zmienia działanie klawiatury i myszki tak by działy na nowym Screen-ie i, przez co, przestały na starym.

2.4 Uno.game.input

2.4.1 KeyboardMangaer.java

KeyboardManagaer.java jest programem, który nasłuchuje zdarzeń z klawiatury.

Biblioteki użyte w programie KeyboardManagaer.java:

- Java.awt.event.KeyEvent;

- Java.awt.event.KeyListener;

Ten program jest połączony z innymi programami:

- Uno.window.ui.manager.UIManagerS;

```
public void keyTyped(KeyEvent e) {
		if(curUIM!=null)
		{
			curUIM.keyTyped(e);
		}
	}
```

- Wysyła naciśnięty klawisz do UIManager-a.

2.4.2 MouseManager.java

MouseManager.java jest programem, który nasłuchuje zdarzeń z myszy.

Biblioteki użyte w programie KeyboardManagaer.java:

- Java.awt.event.MouseEvent;
- Java.awt.event.MouseListener;
- Java.awt.event.MouseMotionListener;

Ten program jest połączony z innymi programami:

- Uno.window.ui.manager.UIManagerS;

```
public void mouseMoved(MouseEvent arg0) {
		mouseX = arg0.getX();
		mouseY = arg0.getY();

		if(curUIM!=null)
		{
			curUIM.mouseMoved(arg0);
		}
	}
```

- ustawia zmienne X i Y współrzędnych myszki (potrzebne do testów poprzenich) i wysyła dane myszki do UIManager-a.

2.5 Uno.net.client

2.5.1 GameClient.java

GameClient.java jest programem, za pomocą którego klient otrzymuje informacje z serwera.

Biblioteki użyte w programie GameClient.java:

- java.io.BufferedReader;
- java.io.IOException;
- java.io.InputStreamReader;
- java.io.PrintWriter;
- java.net.Socket;
- java.net.UnknownHostException;

Ten program jest połączony z innymi programami:

- Uno.Card;
- Uno.Deal;
- Uno.game.handler.GameHandlerer;
- Uno.net.packets.Packet;
- Uno.net.packets.Packet.PacketsIDs;
- Uno.net.packets.PacketDisconnect;
- Uno.net.packets.PacketGetCard;
- Uno.net.packets.PacketLogin;
- Uno.net.packets.PacketSetCard;
- Uno.net.packets.PacketSizeP;
- Uno.net.packets.PacketUpdate;

```
private void run()
```

- zbyt długi by wkleić. Dostaje informacje/wiadomość od servera i zależnie od 1 litery (zmienionej na cyfre) decyduje jaki rodzaj informacji dostał i robi odpowiednie akcje.

```
private void eventLogin(PacketLogin packet)
	{
		name = packet.getName();
		System.out.println("[Client]Aquaried name:"+name);
	}
```

- Dostaje swoją nazwe od wiadomości z pakietu dostaną od servera.

```
private void eventUpdate(PacketUpdate packet)
	{
		nameList = packet.getNameList();
	}
```

- Uaktualnia liste graczy (ich nazwy).

```
private void eventGetCard(PacketGetCard packet)
	{
		String cardData = packet.getCardData();
		if(!cardData.equals("-1"))
		{

			Card tmpCard = Card.getCardfromString(cardData);
			clientDeck.addCard(tmpCard);
			System.out.println("[Client]:Aquaried
card:"+tmpCard.TransalteToString());

		}
		else
			System.out.println("[Client]No more cards in pile from server");

	}
```

- dostaje karte od servera i wstasowuje je do swojej talii, chyba że takiej karty nie ma (talia servera jest pusta).

```
private void eventSetCard(PacketSetCard packet)
	{
		centerCard = Card.getCardfromString(packet.getCardData());
```

```
            System.out.println("[Client]:New center
card:"+centerCard.TransalteToString());
      }
```

- wysyła prośbe o dostanie karty

2.6 Uno.net.packets

Wszystkie programy w tym pakiecie są potrzebne do wysyłania informacji do serwera/klienta, aby coś zrobić, pierwszy charakter z wiadomości mówi jaki to pakiet.

```
public static PacketsIDs getPacketID(int id)
       {
             for (PacketsIDs i : PacketsIDs.values()) {
                 if (i.getId() == id) {
                     return i;
                 }
              }
             return PacketsIDs.INVALID;
       }
```

- znajduje pakiet z wartości enum-a PacketIDs i zwraca go chyba że jest błędny to zwraca wartość INVALID(0).

```
public String readData(String data) {
              return data.substring(1);
       }
```

- odczytuje pierszą litere z wiadomości i ją zwraca

```
public abstract String getData()
```

- abstrakcyjna klasa, w innych pakietach jest implementowana jako zwrot wiadomości, pierwsza litera z niej jest ot numer pakietu, a następne, jeżeli takie są, inne informacje

2.7 Uno.net.server

2.7.1 ClientHandlerer.java

ClientHandlerer.java jest programem(server), który jest połaczony z klientem, wysyła do niego komunikaty.

Biblioteki użyte w programie ClientHandlerer.java:

- java.io.BufferedReader;
- java.io.IOException;
- java.io.InputStreamReader;
- java.io.PrintWriter;
- java.net.Socket;

Ten program jest połączony z innymi programami:

- Uno.Card;
- Uno.Deal;
- Uno.net.packets.Packet;
- Uno.net.packets.PacketLogin;
- Uno.net.packets.Packet.PacketsIDs;

```
public void run()
```

- Tak samo jak u klienta lecz dostaje wiadomości od klienta

2.7.2 GameServer.java

GameServer.java jest programem, w którym klient dostaje inforamcje od serwera.

Biblioteki użyte w programie ClientHandlerer.java:

- java.io.IOException;
- java.net.ServerSocket;
- java.util.ArrayList;
- java.util.Random;

Ten program jest połączony z innymi programami:

- Uno.Card;
- Uno.CardHandler;
- Uno.net.packets.PacketDisconnect;
- Uno.net.packets.PacketGetCard;
- Uno.net.packets.PacketSetCard;
- Uno.net.packets.PacketSizeP;

- Uno.net.packets.PacketStartGame;
- Uno.net.packets.PacketTurn;
- Uno.net.packets.PacketUpdate;

```
public void run()
```

- dostaje przychodzące połączenia od klientów dopóki nie rozpocznie się gra lub nie przekroczą limitu miejsc

```
public void sendToAllClients(String msg)
	{
		for(ClientHandlerer c : clientList)
		{
			c.sendMessageToClient(msg);
		}
	}
```

- wysyła informacje wszytskim klientom z listy

```
public void sendCardToClient(ClientHandlerer c,boolean changeT)
	{
		Card tmpcard = deck.getLast();
		String msgD = "";
		if(tmpcard==null)
		{
			msgD = new PacketGetCard("-1").getData();
			c.sendMessageToClient(msgD);
			return;
		}
		else
			msgD = new PacketGetCard(tmpcard.TransalteToString()).getData();
		c.sendMessageToClient(msgD);
		c.addToDeck(tmpcard);
		deck.removeLast();

		if(changeT)
		{
```

```
                nextTurn(false);
                updatePSize();
                sendToAllClients(new PacketSizeP(pSize).getData());
            }

        }
```

- funckja ta bierze karte z talii servera i wysyła ją do klienta, chyba że jest pusta to wtedy wysyła wiadomość powiadającą że już nie ma kart, dodaje także tą samą kartę do ClientHandelrera. Jeżeli jest taka potrzeba wysyła informacje do wszyskich klientów o zmianie ilości kart w taliach klientów i zmienia ture

2.8 Uno.window

2.8.1 AppWindow

AppWindow jest programem, który inicjuje działanie okna aplikacji.

Biblioteki użyte w programie AppWindow.java:

- Java.awt.Canvas;
- Java.awt.Dimension;
- Java.awt.event.WindowEvent;
- Javax.swing.JFrame;

Ten program zawiera konstruktor klasy AppWindow i inne metody: canvasRet, windowRet, resizeWindow, exitWindow.

```
public AppWindow(int widht,int height)
        {

            window =new JFrame("Uno - Project");
            window.setSize(widht, height);
            window.setLocationRelativeTo(null);
            window.setResizable(false);
            window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            window.setVisible(true);

            canvas = new Canvas();
```

```
            canvas.setPreferredSize(new Dimension(widht,height));
            canvas.setFocusable(false);//bez tego nie dziala input

            window.add(canvas);
        }
```

- konstruktor ten tworzy okno z płótnem (na którym się rysuje) o podanej nazwie szerokościach oraz operacji wyjścia z ekranu. Także wyświetla okno.

2.8.2 CardLoader.java

CardLoader.java jest programem, potrzebne do załadowania, pokolorowania, obrócenia obrazów kart.

Biblioteki użyte w programie CardLoader.java:

- Java.awt.Color;
- Java.awt.Graphics;
- Java.awt.Graphics2D;
- Java.awt.imge.BufferedImage;

Ten program jest połączony z innymi programami:

- Uno.Card;
- Uno.game.handler.GameHandlerer;

Ten program zawiera klas CardLoader i metody: loadRotatedCards, loadCards, paintOver, RotateNCards.

2.8.3 ImagesLoader.java

ImagesLoader.java jest programem, który ładuje zdjecia do aplikacji.

Biblioteki użyte w programie ImagesLoadre.java:

- Javax.imageio.ImageIO;
- Java.awt.image.BufferedImage;
- Java.io.IOException;

Ten program zawiera klas ImagesLoader i metody: loadImage, cropOutImage.

```
public static BufferedImage loadImage(String path)
        {
```

```
            try {
                  return ImageIO.read(ImagesLoader.class.getResource(path));
            } catch (IOException e) {
                  e.printStackTrace();
                  System.out.println("Could not laod image from path:"+path);
            }
            return null;

      }
```

- funckja ta ładuje i zwraca obraz o podanej ścieżce

```
public static BufferedImage cropOutImage(int x,int y,int w,int h,BufferedImage sheet)
      {
            return sheet.getSubimage(x, y, w, h);
      }
```

- funkcja ta zwraca kawałek podanego rozmiaru o podanych z niego współrzędnych i wielkości

2.8.4 SheetHolder.java

SheetHolder.java jest programem, który trzyma sheests i zwraca je.

Biblioteki użyte w programie SheetHolder.java:

- Java.awt.image.BufferedImage;

Ten program zawiera klas SheetHolder i metody: getCardSheet, setCardSheet, getButtonSheet, setButtonSheet.

2.8.4 WindowHandlerer.java

WindowHandlerer.java jest programem, który odpowiedzialny za wyłączenie serwera i klienta(jeśli są włączone), gdy okna programu są zamknięte.

Biblioteki użyte w programie SheetHolder.java:

- Java.awt.event.WindowEvent;
- Java.awt.event.WindowListener;

Ten program jest połączony z innymi programami:

- Uno.game.handler.GameHadlerer;

2.9 Uno.window.screens

2.9.1 ConnectionScreen.java

ConnectionScreen.java jest programem, który służące do połaczenia lub hostowania gry.

Biblioteki użyte w programie ConnectionScreen.java:

- java.awt.Color;
- java.awt.Font;
- java.awt.Graphics;
- java.awt.Window;
- java.awt.image.BufferedImage;
- javax.swing.JLabel;
- javax.swing.JOptionPane;
- javax.swing.SwingUtilities;

Ten program jest połączony z innymi programami:

- Uno.game.handler.GameHandlerer;
- Uno.window.ui.UIButton;
- Uno.window.ui.UIButtonImage;
- Uno.window.ui.UIClicker;
- Uno.window.ui.UITextEnter;
- Uno.window.ui.manager.UIManagerS;

Ten program zawiera class ConnectionScreen który dziedziczy Screens, konstruktor klasy ConnectionScreen, metody: addComponents, update, render, create_Label, create_Dialog, disposeOptionDialog, startGameForClient.

2.9.2 GameScreen.java

GameScreen.java jest programem, który inicjuje okno w którym rozgrywa się gra.

Biblioteki użyte w programie GameScreen.java:

- java.awt.Color;
- java.awt.Graphics;
- java.awt.image.BufferedImage;
- java.util.ArrayList;
- javax.swing.JOptionPane;

Ten program jest połączony z innymi programami:

- Uno.Card;
- Uno.Deal;
- Uno.CardHandler;
- Uno.game.handler.GameHandlerer;
- Uno.window.ui.UIButtonImage;
- Uno.window.ui.UICard;
- Uno.window.ui.UIClicker;
- Uno.window.ui.UIHolder;
- Uno.window.ui.UITakeCard;
- Uno.window.ui.manager.UIManagerS;

Ten program zawiera class GameScreen który dzidziczy Screens, konstruktor klasy GameScreen, metody: addComponent, update, render, drawPacketCards, exchangeCenter, addCardtoUI, rearangeCards, drawOnTop, commitAction, UpdateDeck, checkIfFinished, assignNames.

2.9.3 MenuScreen.java

MenuScreen.java jest programem, który inicjuje okno „Menu”.

Biblioteki użyte w programie MenuScreen.java:

- java.awt.Color;
- java.awt.Font;
- java.awt.Graphics;
- java.awt.image.BufferedImage;

Ten program jest połączony z innymi programami:

- Uno.game.handler.GameHandlerer;
- Uno.window.ui.UIButtonImage;
- Uno.window.ui.UIClicker;
- Uno.window.ui.manager.UIManagerS;

Ten program klas MenuScreen który dzidziczy Screens, konstruktor klasy MenuScreen, metody: addComponent, update, render.

2.9.4 Screens.java

Screens.java jest programem, który inicjuje abstrakcyjne klasa po której wszystkie screeny dziedziczą, definiuje rzeczy które są potrzebne dla każdego z nich(zmienne,funkcje).

Biblioteki użyte w programie Screens.java:

- Java.awt.Graphics;

Ten program jest połączony z innymi programami:

- Uno.game.handler.GameHandlerer;
- Uno.window.ui.manager.UIManagerS;

```
public abstract void update();

public abstract void render(Graphics g);
```

- Dwie abstakcyjne funkcje które słożą za uaktualnanie danych w oknie i renderowanie orbazów. Każdy screen ma ich swoja implementacje

2.10 Uno.window.ui

Wszystkie programy w tym pakiecie są odpowiada za user interface programu.

2.11 Uno.window.ui.manager

2.11.1 UIManagerS.java

UIManagerS.java jest programem, który robi wszystkie rzeczy dla obiektów z uiLista (każdy screen ma innego uiLista) kiedy jakieś wydarzenie się pojawi (renderuje, updatuje, kiedy jest ruch myszy lub klawiatury itp.).

Biblioteki użyte w programie UIManagerS.java:

- java.awt.Graphics;

- java.awt.event.KeyEvent;
- java.awt.event.MouseEvent;
- java.util.ArrayList;

Ten program jest połączony z innymi programami:

- Uno.game.handler.GameHandlerer;
- Uno.window.ui.UiComponent;

```
public void update()
	{
		for(UIComponent i: uiList)
			i.update();

	}
```

- funckja ta uaktualnia wszytskie komponenty które znajdują się w liście obiektów UI, w tej klasie są inne funckje które działaja na takiej samiej podstawie, z różnicą taką że przyjmują parametr który jest potrzebny do zrobienia innej funkcji np. przy uaktualnieniu grafik jest przekazywany Graphics

3 Server

Aby więc było spotkanie co najmniej dwóch obiektów na terytorium jednego serweru - serwer musi zająć określony port, a drugi musi znaleźć miejsce spotkania znając adres IP.

Port - to unikalny numer, z którym powiązane jest określone socker, innymi słowami, jest zajęty przez określoną usługę, aby móc się z nią skontaktować.

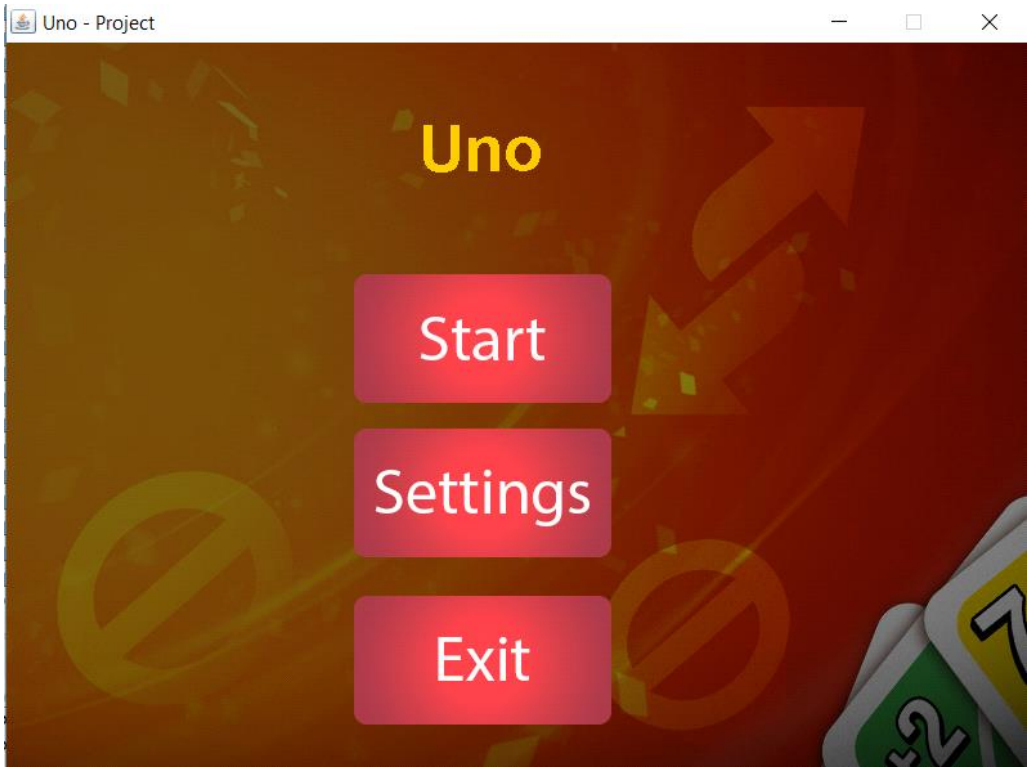
clientSocket – implementuje ideę gniazda. Poprzez kanały wejściowe/wyjściowe klient komunikuje się z serwerem.

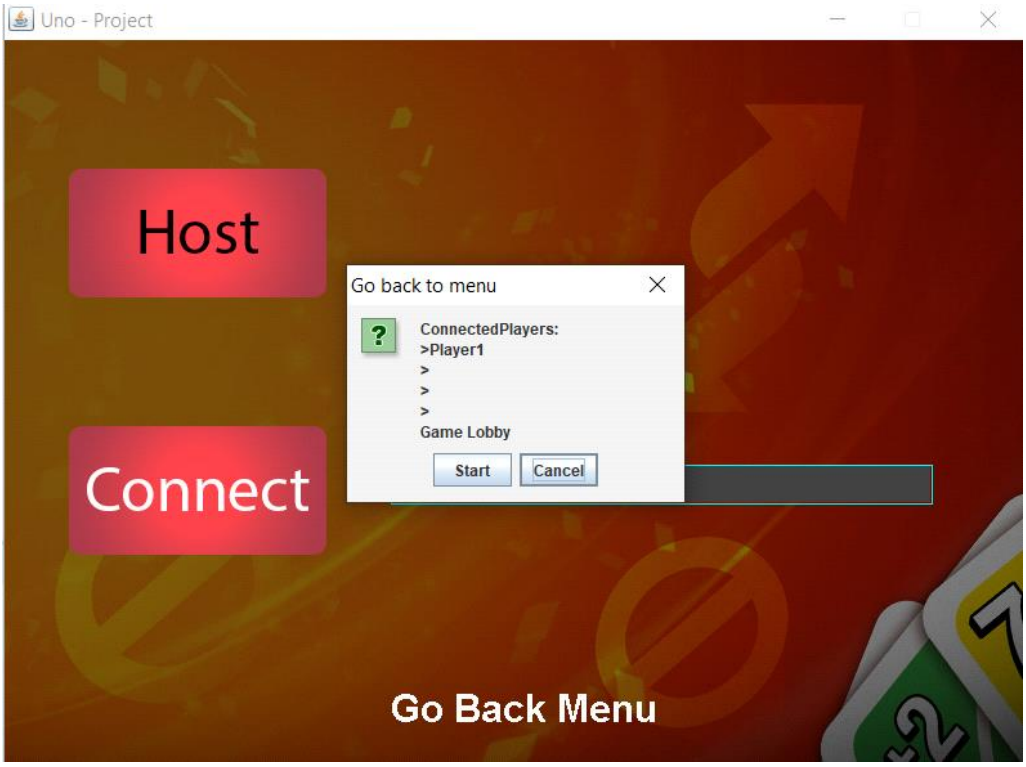
ClientSocket jest zadklarowany po stronie klienta, a serwer odtwarza ją, odbierając sygnał połączenia. Tak odbywa się komunikacja sieciowa.

serverSocket – to to samo co clientSocket, tylko dla serwera. Ale odgrywa zupełnie inną rolę niż clientSocket.

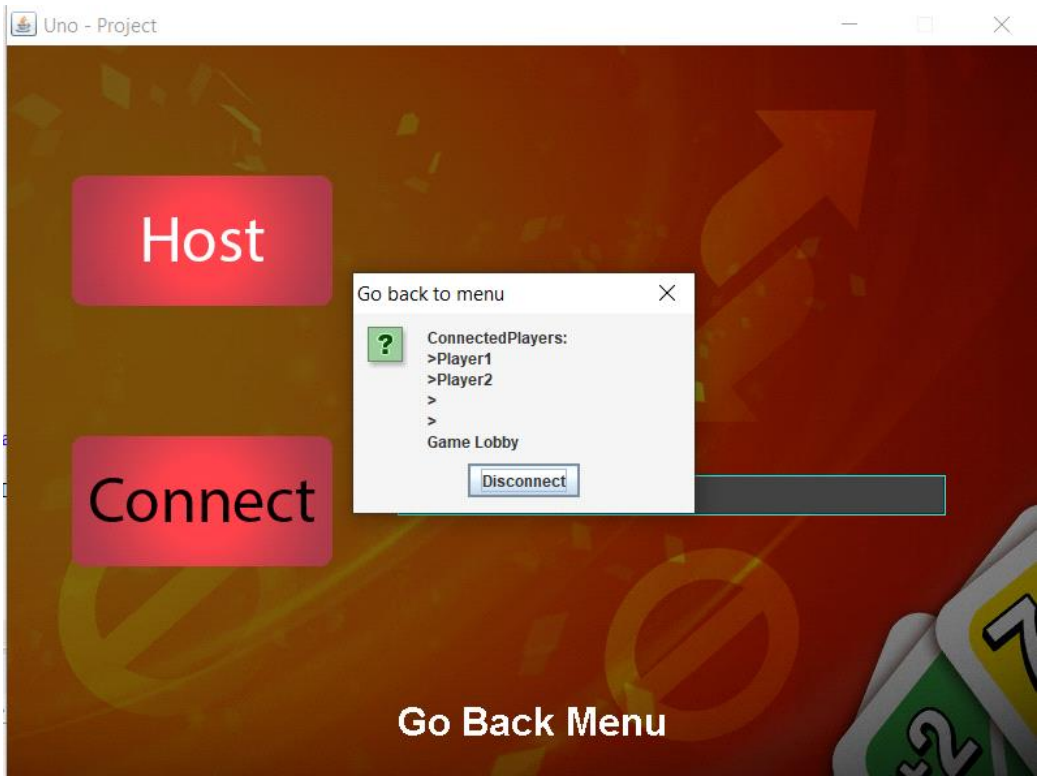
Używamy TCP.

4 Interfejs gry

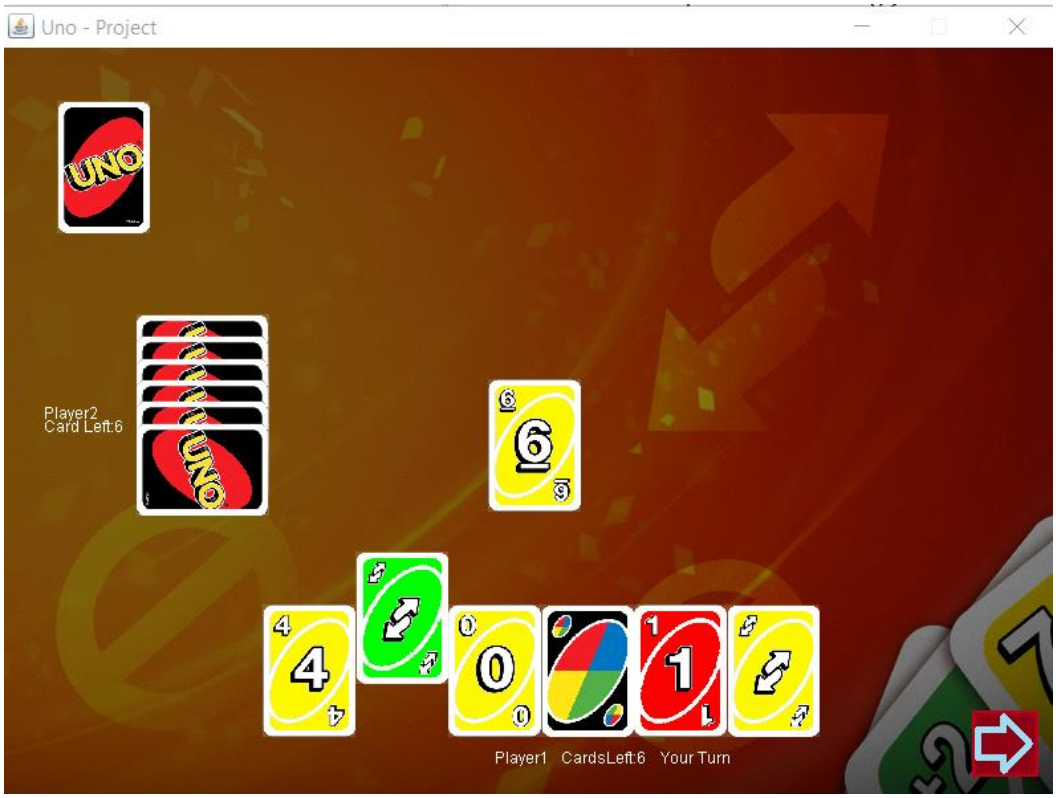

Uno - Project
Uno
Start
Settings
Exit


Uno - Project
Host
Connect
Go back to menu
ConnectedPlayers:
>Player1
>
>
>
Game Lobby
Start
Cancel
Go Back Menu

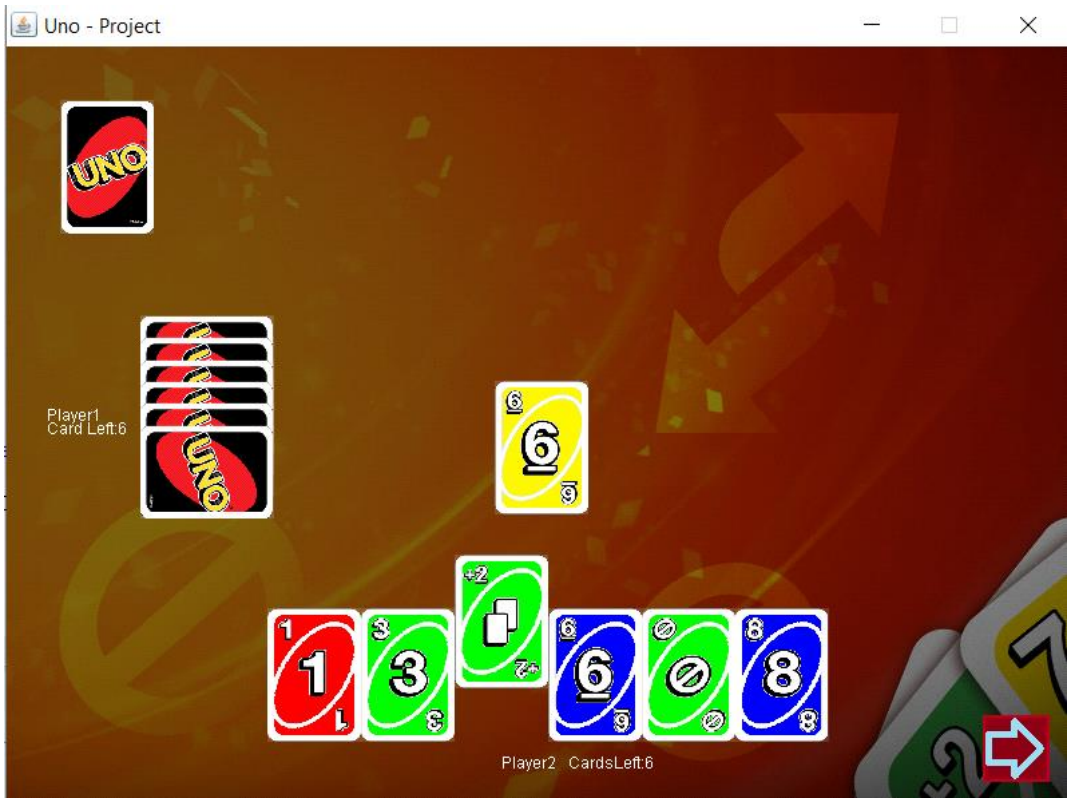
Serwer



Klient



Talia kart serwera



Talia kart klienta

Uwaga! Z gry należy kolejno wyjść najpierw klienci, a następnie serwer.

5 Wniosek

Wszystkie postawione cele zostały wykonane. Zrealizowałyśmy projekt wykorzystujący architekturę klient-serwer napisany w języku Java. Nam zajeło dużo czasu żeby zrozumić jak realizować prawidlowo server i zrobić to w programie. Dla wykonania potrzebnym było duże rzeczy znaleźć i nauczyć się temu. Ale na końcu zdążyłyśmy realizować wszysko co było potrzebne.

Wspólnymi siłami daliśmy radę stworzyć aplikacje, z której możemy być dumni.