

Distributed Programming

MOD006128

A Distributed Multi-Player Racing Game

School of Computing and Information Technology

BEng Computer Science

Academic Year: 2023

Trimester: 2

Contents

[Task 1 - Spinning your cars 3](#_Toc132949623)

[Task 1 – Code 4](#_Toc132949624)

[Task 2 - Single player game 6](#_Toc132949625)

[Task 2 – Code 6](#_Toc132949626)

[Task 3 - Multi player game 15](#_Toc132949627)

[Task 3 – Code 15](#_Toc132949628)

[Appendix 23](#_Toc132949629)

[References 24](#_Toc132949630)

# Task 1 - Spinning your cars

For this part of the assignment the goal was to source pictures of a car, process them to be 50 by 50 pixels and gather sixteen pictures containing the car at 22.5° angles all the way to the initial position. Once these images have been processed the cars have to rotate inside a paint component while on a timer so that the car continuously rotates at the same speed.

The images in Figure 1 and Figure 2 have been selected for this task.

Diagram

Description automatically generatedA picture containing tool

Description automatically generated

Figure 1 - Green car Figure 2 - Blue Car

These images have then been processed into the sixteen separated angles seen in Figure 3 for the green car and Figure 4 for the blue car.

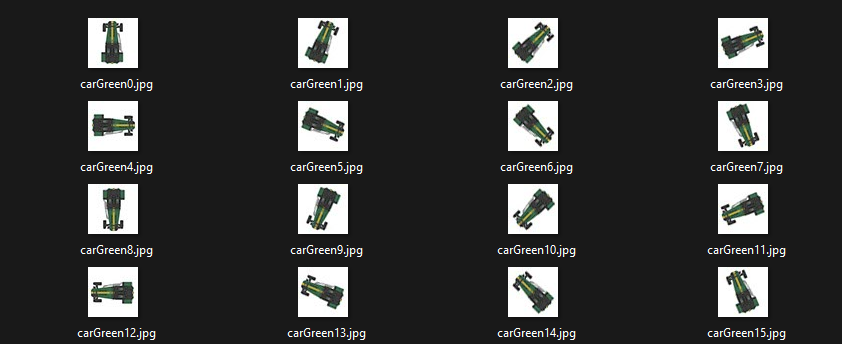


Figure 3 - Green car split angles.

Graphical user interface, application, Teams

Description automatically generated

Figure 4 - Blue car split angles.

## Task 1 – Code

In this task it was requested to use a Swing Timer class called “Timer” that will repeatedly delay the execution of the for loop that runs through the car angle iterations in ascending order from zero to fifteen. The swing timer is set to run every second creating a smooth rotation as shown in Figure 5.

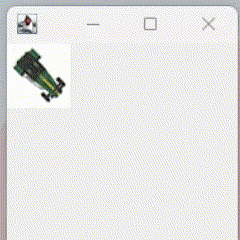


Figure 5 - Rotation Example

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
  
// For Timer we must implement Action Listener  
public class DrawCars extends JPanel implements ActionListener {  
  
 // Timer  
 private Timer spinner;  
 // Array containing images for each angle of the car  
 private ImageIcon imageArray[];  
 private int speed = 100, totalImages = 16, currentImage = 0;  
  
 // Draw the car image array  
 public DrawCars() {  
 imageArray = new ImageIcon[totalImages];  
  
 for (int i=0; i < imageArray.length; i++) {  
 imageArray[i] = new ImageIcon("GreenCar\\carGreen" + i + ".jpg");  
 }  
 // Timer set to run at the speed of 100  
 spinner = new Timer(speed, this);  
 spinner.start();  
 }  
  
 // Draws car to window  
 public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
  
 // When reaches end of spinner (array) restart  
 if (currentImage >= imageArray.length - 1) {  
 currentImage = 0;  
 }  
 currentImage++;  
 imageArray[currentImage].paintIcon(this, g, 0, 0);  
 }  
  
 // Wipes window allowing for new angle to show  
 public void actionPerformed(ActionEvent e) {  
 repaint();  
 }  
  
}

The action performed is to repaint the car image with the new angle.

In the main the frame for the car images is called and added to the rotation, then set to visible. The window is set to exit on close and the window size it set to 300 by 300.

import javax.swing.JFrame;  
  
class AFrame {  
 public static void main(String[] args) {  
 JFrame frame = new JFrame();  
 DrawCars rotation = new DrawCars();  
 frame.add(rotation);  
 frame.setVisible(true);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setSize(300,300);  
 }  
}

# Task 2 - Single player game

For task two the game had to be created, this included the ability to move the cars using four key commands each, create a menu for the game, and integrate the rules of the race ground such as the cars crashing or running over grass.

Starting with the cars, a change has been made, all sixteen images have been taken and overlapped onto a four-by-four image containing all the angles as seen in Figure 6. This image has an empty background allowing smooth visuals when corners are taken. This has been achieved using Adobe Photoshop and has been tested to be an optimal distancing between the cars and no other corners to show at specific angles while driving.

A picture containing several

Description automatically generated

Figure 6 - Green car overlapped image.

## Task 2 – Code

This has been implemented as with the below code where this image is taken then and loaded, at this point the whole image is available, further into the graphics of the game this image is being split. This can be found in the class “Media” as well as the code below.

import java.awt.image.BufferedImage;  
import java.io.IOException;  
import javax.imageio.ImageIO;  
  
public class CarFrames {  
 private int index; // Index  
 private BufferedImage[] frames; //Array of 'frames' of the car images  
 public CarFrames(BufferedImage[] frames) {  
 this.frames = frames;  
 index = 0; //First car image 0 degrees on start line  
 }  
 public BufferedImage getCurrentFrame(int index) {  
 return frames[index];  
 } //Used for the Move Sheet image  
 public static BufferedImage loadImage(String path) {  
 try {  
 return ImageIO.*read*(CarFrames.class.getResource(path)); // Returns car images from path  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*exit*(1); // End game if image can't load  
 }  
 return null;  
 }  
}

In class “Media” the image gets split into sixteen parts in the for loop for each of the cars and stored into the arrays “Buffered Image” of each car.

public static void initiate() {  
 Media sheetGreen = new Media(CarFrames.*loadImage*("/GreenCar/MoveSheet.png")); // Green car move sheet image path  
 Media sheetBlue = new Media(CarFrames.*loadImage*("/BlueCar/MoveSheet.png")); // Blue car move sheet image path  
 *playerGreen\_move* = new BufferedImage[16]; // Array for the 16 cropped images fo the car  
 *playerBlue\_move* = new BufferedImage[16]; // Array for the 16 cropped images of the car  
 // Loop for te Move Sheet image cropping the 16 images for the cars  
 for (int i = 0; i < *rows*; i++) {  
 for (int j = 0; j < *cols*; j++) {  
 *playerGreen\_move*[(i\**cols*) + j] = sheetGreen.cut(j\**width*, i\**height*, *width*, *height*);  
 *playerBlue\_move*[(i\**cols*) + j] = sheetBlue.cut(j\**width*, i\**height*, *width*, *height*);  
 }  
 }  
 *fire* = CarFrames.*loadImage*("/fire/fire.png"); // Fire picture path

Next the race ground has been created in the “Rache Graphics” class where collisions and checked between drivers and the driver commands are initiated, selecting the green car to be controlled by the “wasd” keys and the blue car by the arrows.

Then the grass, road, start line and checkpoint line are created using “Graphics 2D” and diverse colours, shapes, and lines.

import java.awt.Color;  
import java.awt.Graphics;  
import java.awt.Graphics2D;  
import java.awt.Rectangle;  
import java.awt.geom.Area;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import javax.swing.JOptionPane;  
  
public class RaceGraphics extends State {  
 private ManeuveringControl maneuveringControl;  
 private int collisions = 0; // Variable to check for crashing  
 public RaceGraphics(Handler handler) {  
 super(handler);  
 maneuveringControl = new ManeuveringControl(handler, new SpeedControl(handler, 700, 350, Media.*playerGreen\_move*, "wasd" ),  
 new SpeedControl(handler, 750, 350, Media.*playerBlue\_move*, "arrows") ); // Assigning key controls to each driver/car  
 }  
 @Override  
 public void readUserAction() {  
 maneuveringControl.maneuvers(); // Read keys  
 }  
 @Override  
 public void render(Graphics g) {  
 Graphics2D g2d = (Graphics2D) g.create();  
  
 Color c1 = new Color(0,153,0); // Dark Green color for the grass  
 g.setColor( c1 );  
 g.fillRect( 150, 200, 550, 300 ); // Inner grass  
  
 Color c2 = Color.*black*;  
 g.setColor( c2 );  
 g.drawRect(50, 100, 750, 500); // Outer edge of road  
 g.drawRect(150, 200, 550, 300); // Inner edge of road  
  
 Color c3 = Color.*yellow*;  
 g.setColor( c3 );  
 g.drawRect( 100, 150, 650, 400 ); // Road center line  
  
 Color c4 = Color.*white*;  
 g.setColor( c4 );  
 g.fillRect( 700, 350, 800, 5 ); // Start line  
  
 g.setColor(Color.*red*);  
 g.drawLine(50, 350, 150, 350 ); // Checkpoint  
  
 Area outer = new Area(new Rectangle(0, 0, 850, 650 )); // Outer grass area ends  
 Rectangle inner = new Rectangle(50, 100, 750, 500); // Outer grass area start  
 outer.subtract(new Area(inner)); // Subtract anything in the middle to fill the outer  
 g2d.setColor(new Color(0,153,0)); // Dark Green color for the grass  
 g2d.fill(outer); // Fill outer grass in dark green  
 maneuveringControl.render(g);  
 if (maneuveringControl.isCrash()) {gameOver(g);} // Check for car crash  
 }  
 public void gameOver(Graphics g) {  
 for (Maneuvering m : maneuveringControl.getDrivers()) {  
 g.drawImage(Media.*fire*, (int)m.x, (int)m.y,null); // Fire image at crash location  
 }  
 // Giving the system time to check and render fire on cars  
 if (collisions == 2) {  
 JOptionPane.*showMessageDialog*(null, "Game Over", "Game Over", JOptionPane.*YES\_NO\_OPTION*); // Game ends  
 System.*exit*(0); // Exit  
 }  
 collisions++;  
 }  
}

Inside the “Grass Collision” class the movement of the cars as well as the grass actions are implemented. In this class we mainly take care of the car angle image, the speed the car is currently in, and which direction is it going next. The first method is the laps checker, we created a small boundary around the grass meant to manage the fact that each car is not exactly five by five pixels and should get slightly closer to the grass than the edge of the car image. Then creating the start and checkpoint line, checking the checkpoint line has been passed then if the finish line has been passed user will be prompted with “Winner!”

public void laps(){  
  
 Rectangle grassBox = new Rectangle(((int)(x + 5)), ((int)(y + 5)), boundary.width, boundary.height); // Diver boundary  
 Rectangle endLine = new Rectangle( 700, 350, 800, 1 ); // End line/Finish line  
 Rectangle checkpoint = new Rectangle( 50, 350, 100, 1 ); // Ensure drivers race the lap and cross the checkpoint before end line  
 // Check if the checkpoint line has been passed  
 // Issue is this work if laps are done backwards as well as if checkpoint is passed then car turns around for the finish line  
 if (grassBox.intersects(checkpoint)) {  
 passCheckpoint = true;  
 }  
 // If the end line and checkpoint has been passed return 'Winner!'  
 if (grassBox.intersects(endLine) && passCheckpoint) {  
 System.*out*.println("Winner!");  
 laps++;  
 passCheckpoint = false;  
 }  
}

For checking if any grass has been touched, we go through a few if and else statements, if the grass has not been touched, then the car can continue in the direction x (left or right) and y (up or down) inside the available space. If the car has collided with the grass the car will stop and require the driver to reverse out slightly and redirection onto the race lanes. A grass sound will play as soon as the car touched the grass.

public void moveX() { // Grass touching detection  
  
 if (xMove > 0) { // Moving right  
 if (!grass.contains((int) (x + boundary.width), y + 5) && !grass.contains((x + boundary.width), boundary.height) && x - boundary.width < 710 ) {  
 x += xMove;  
 collision = false;  
 } else {  
 collision = true;  
 Media.*sound*.get("grass").play();  
 setSpeed(0); // When cars crash speed is set to 0  
 }  
 }else if (xMove < 0) { // When grass is touched the car stops allowing for driver to reverse if needed  
  
 if (!grass.contains(x + 5, y + 5) && !grass.contains(x + 5, boundary.height) && x > 50 ) {  
 x += xMove;  
 collision = false;  
 }else {  
 collision = true;  
 Media.*sound*.get("grass").play();  
 setSpeed(0); // When grass is touched the car stops allowing for driver to reverse if needed  
 }  
 }  
}  
public void moveY() {  
 if (yMove < 0) { // Moving up  
 if (!grass.contains(x + 5, y + 5) && !grass.contains(boundary.width, y + 5 ) && y > 100 ) {  
 y += yMove;  
 collision = false;  
 }else {  
 collision = true;  
 Media.*sound*.get("grass").play();  
 setSpeed(0); // When grass is touched the car stops allowing for driver to reverse if needed  
 }  
 } else if (yMove > 0) { // Moving down  
 if (!grass.contains(x + 5, y + boundary.height + 5) && !grass.contains(boundary.width, y + boundary.height + 5) && y < 550 ) {  
 y += yMove;  
 collision = false;  
 }else {  
 collision = true;  
 Media.*sound*.get("grass").play();  
 setSpeed(0); // When grass is touched the car stops allowing for driver to reverse if needed  
 }  
 }

For the next car image to be found depending on the keys pressed the following statements are followed such as each picture frame has a number reference from zero to sixteen.

public void direction(int currentImageIndex) {  
 // Driver moves based on current angle  
 if (currentImageIndex == 0 ) {  
 yMove = yMove - 2 \* speed;  
 } else if (currentImageIndex == 1 ) {  
 xMove = xMove + 1 \* speed;  
 yMove = yMove - 2 \* speed;  
 } else if (currentImageIndex == 2 ) {  
 xMove = xMove + 2 \* speed;  
 yMove = yMove - 2 \* speed;  
 } else if (currentImageIndex == 3 ) {  
 xMove = xMove + 2 \* speed;  
 yMove = yMove - 1 \* speed;  
 } else if (currentImageIndex == 4 ) {  
 xMove = xMove + 2 \* speed;  
 }else if (currentImageIndex == 5 ) {  
 xMove = xMove + 2 \* speed;  
 yMove = yMove + 1 \* speed;  
 }else if (currentImageIndex == 6 ) {  
 xMove = xMove + 2 \* speed;  
 yMove = yMove + 1 \* speed;  
 }else if (currentImageIndex == 7 ) {  
 xMove = xMove + 1 \* speed;  
 yMove = yMove + 2 \* speed;  
 }else if (currentImageIndex == 8 ) {  
 yMove = yMove + 2 \* speed;  
 }else if (currentImageIndex == 9 ) {  
 xMove = xMove - 1 \* speed;  
 yMove = yMove + 2 \* speed;  
 }else if (currentImageIndex == 10 ) {  
 xMove = xMove - 2 \* speed;  
 yMove = yMove + 2 \* speed;  
 }else if (currentImageIndex == 11 ) {  
 xMove = xMove - 2 \* speed;  
 yMove = yMove + 1 \* speed;  
 }else if (currentImageIndex == 12 ) {  
 xMove = xMove - 2 \* speed;  
 }else if (currentImageIndex == 13 ) {  
 xMove = xMove - 2 \* speed;  
 yMove = yMove - 1 \* speed;  
 }else if (currentImageIndex == 14 ) {  
 xMove = xMove - 2 \* speed;  
 yMove = yMove - 2 \* speed;  
 }else if (currentImageIndex == 15 ) {  
 xMove = xMove - 1 \* speed;  
 yMove = yMove - 2 \* speed;  
 }  
}

The Menu class has been included and is composed of a Start button, a Quit button, and a Help button. While the Menu window is open the Mario Kart music will play in the background until the game starts or the application is closed.

The Help option will give the user some small details on how to use the cars and what each key command does, how to win and how to lose the game.

@Override  
public void readUserAction() {  
 if(handler.getMouseManager().play) {  
 Media.*music*.get("menu").stop(); // Once play button clicked, menu music stops  
 Media.*sound*.get("start").play(); // Start sound starts  
 State.*setState*(handler.getGame().getGameState());  
 } else if (handler.getMouseManager().quit) {  
 System.*exit*(0); // If user selects exit, system exits  
 } else if (handler.getMouseManager().help) { // If user selects help will be prompted with bellow dialog box  
 String message = "Welcome to the Racing Game\n\n" +  
 "To control the cars you can use the following keys:\n" +  
 "1. Accelerate - W and Arrow UP \n" +  
 "2. Brake - S and Arrow DOWN\n" +  
 "3. Left - A and Arrow LEFT\n" +  
 "4. Right - D and Arrow RIGHT\n\n" +  
 "Crashing Rules:\n" +  
 "Cars that touch the grass will stop and might require \n" +  
 "reverse driving to be able to return on the track.\n" +  
 "When two cars crash into each other the game will end.";  
 // Using separate thread to main thread, for the flow of the moving background in the Menu  
 // Dialog box reference: https://www.developer.com/java/create-java-dialog-boxes/  
 Thread t = new Thread(new Runnable(){  
 public void run(){  
 JOptionPane.*showMessageDialog*(null, message, "Help", 1);  
 }});  
 t.start();  
 handler.getMouseManager().help = false; // Allows help button to be pressed again  
 }  
}  
@Override  
public void render(Graphics g) {  
 Color c1 = new Color(204,255,204); // Light green color for the title and menu  
 g.setColor(c1);  
 Graphics2D g2d = (Graphics2D) g.create();  
 g.setFont(new Font("Times Roman", Font.*BOLD*, 50)); // Selected font bold and text size  
 g.drawString("Racing Game - Part 2", 170, 100); // Title  
  
 g.setFont(new Font("Times Roman", Font.*BOLD*, 30)); // Selecting font bold and text size for menu  
 g2d.draw(playButton);  
 g.drawString("Play", playButton.x + 15, playButton.y + 35); // Play Button  
 g2d.draw(helpButton);  
 g.drawString("Help", helpButton.x + 15, helpButton.y + 35); // Help button  
 g2d.draw(quitButton);  
 g.drawString("Quit", quitButton.x + 15, quitButton.y + 35); // Quit button  
 // Moving cars in menu  
 // The if loop runs until the cars are out of the screen  
 // Then the else if creates a continuous loop as x becomes 0 each time returning it to the if loop above  
 if (x <= 850) {  
 g.drawImage(Media.*playerBlue\_move*[4], x, 420, null);  
 g.drawImage(Media.*playerGreen\_move*[4], x, 280, null);  
 x++;  
 }  
 else if (x > 850) {  
 x = 0;  
 }  
 // Blur effect under title for both of the cars, intentionally having one longer than the other  
 for (int i = 850; i > 120; i--) {  
 g.drawImage(Media.*playerGreen\_move*[12], i, 105,null); // Printing the same image over itself very close together  
 }  
 for (int i = 850; i > 220; i--) {  
 g.drawImage(Media.*playerBlue\_move*[12], i, 140,null); // Printing the same image over itself very close together  
 }  
}

Graphical user interface, application

Description automatically generated

Figure 7 - Menu window.

# Task 3 - Multi player game

## Task 3 – Code

In this last part of the assignment the game created in task two had to be modified into being able to handle multi-player sessions. This has been attempted but not completely accomplished. This has been started with the creating on packets that will be sent across from a server to the client/s. The packets can have four statuses of invalid, login (connected), disconnected and move which is the status for when the cars are currently moving inside the racecourse.

public abstract class Packet {  
 public static enum PacketStatus {  
 *INVALID*(-1), *LOGIN*(00), *DISCONNECT*(01), *MOVE*(02); // Packet status  
 private int packetID; // ID  
 private PacketStatus(int packetID) {  
 this.packetID = packetID;  
 }  
 public int getId() {  
 return packetID;  
 } // Get ID  
 }  
 public byte packetID;  
 public Packet(int packetID) {  
 this.packetID = (byte) packetID;  
 } // get the packet relevant to ID  
 public abstract void writeData(Client client); // Sends to specific client  
 public abstract void writeData(Server server); //Sends to server  
 public String readData (byte[] data) { // Read  
 String message = new String(data).trim();  
 return message.substring(2);  
 }  
 public abstract byte[] getData(); // Get  
 public static PacketStatus searchPacket(String packetId) { try { return *searchPacket*(Integer.*parseInt*(packetId)); //Ensures correct code is given back  
 } catch (NumberFormatException e) { return PacketStatus.*INVALID*;}  
 }  
 public static PacketStatus searchPacket(int id) { // Search packet status  
 for (PacketStatus p : PacketStatus.*values*()) { if (p.getId() == id) { return p;}  
 }return PacketStatus.*INVALID*;  
 }  
}

In the Server class the packets are received and parsed, then depending on their status they will trigger and error, it will start a game session, it will close the current game session, or it will move the car into the requested position.

The rest of those methods are then created below the packet parsing method.

import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
import java.net.SocketException;  
import java.util.ArrayList;  
public class Server extends Thread {  
 private DatagramSocket socket;  
 private RaceGround race;  
 private ArrayList<Driver> connectedDriver = new ArrayList<Driver>();  
 public Server(RaceGround race) {  
 this.race = race;  
 try { this.socket = new DatagramSocket(1331);  
 } catch (SocketException e) { e.printStackTrace(); }  
 }  
 public void run() {  
 while (true) {  
 byte[] data = new byte[1024];  
 DatagramPacket packet = new DatagramPacket(data, data.length);  
 try { socket.receive(packet);  
 } catch (IOException e) { e.printStackTrace();  
 } this.parsePacket(packet.getData(), packet.getAddress(), packet.getPort());  
 }  
 }  
 private void parsePacket(byte[] data, InetAddress address, int port) {  
 String message = new String(data).trim();  
 Packet.PacketStatus type = Packet.*searchPacket*(message.substring(0, 2));  
 Packet packet = null;  
 switch(type) {  
 default:  
 case *INVALID*: break;  
 case *LOGIN*:  
 packet = new PacketLogin(data);  
 System.*out*.println( "["+address.getHostAddress()+":"+port+"] " + ((PacketLogin)packet).getUsername()+ " has connected...");  
 Driver playerGreen = new Driver(race, 750, 350, ((PacketLogin) packet).getUsername() ,Media.*playerGreen\_move*, address, port);  
// PlayerMP playerBlue = new PlayerMP(race, 750, 350, ((PacketLogin) packet).getUsername() ,Media.playerBlue\_move, address, port);  
 this.addConnection(playerGreen, (PacketLogin)packet);  
// this.addConnection(playerBlue, (PacketLogin)packet);  
 break;  
 case *DISCONNECT*:  
 packet = new PacketDisconnect(data);  
 System.*out*.println( "["+address.getHostAddress()+":"+port+"] " + ((PacketDisconnect)packet).getUsername()+ " has left the game...");  
 this.removeConnection((PacketDisconnect)packet);  
 break;  
 case *MOVE*:  
 packet = new PacketMove(data);  
 //System.out.println(((Packet02Move)packet).getUsername()+" has moved to " +((Packet02Move)packet).getX()+","+ ((Packet02Move)packet).getY());  
 this.handleMove((PacketMove)packet);  
 }  
 }  
 public void removeConnection(PacketDisconnect packet) {  
 this.connectedDriver.remove(getDriverID(packet.getUsername()));  
 packet.writeData(this);  
 }  
 public Driver getDriver(String username) {  
 for(Driver player: this.connectedDriver) {  
 if(player.getUsername().equals(username)){  
 return player;  
 }  
 } return null;  
 }  
 public int getDriverID(String username) {  
 int id = 0;  
 for(Driver player: this.connectedDriver) {  
 if(player.getUsername().equals(username)){  
 break;  
 } id++;  
 } return id;  
 }  
 public void addConnection(Driver player, PacketLogin packet) {  
 boolean alreadyConnected = false;  
  
 for (Driver p : this.connectedDriver) {  
 if (player.getUsername().equalsIgnoreCase(p.getUsername())) {  
 if (p.ipAddress == null) {  
 p.ipAddress = player.ipAddress;  
 }  
 if (p.port == -1) {  
 p.port = player.port;  
 }  
 alreadyConnected = true;  
 } else {  
 sendData(packet.getData(), p.ipAddress, p.port); // Tells current player about new player  
 packet = new PacketLogin(p.getUsername(), p.x, p.y);  
 sendData(packet.getData(), player.ipAddress, player.port); // Tells new player about current player  
 }  
 } if(!alreadyConnected) {  
 this.connectedDriver.add(player);  
 //packet.writeData(this);  
 }  
 }  
 public void sendData(byte[] data, InetAddress ipAddress, int port) {  
 DatagramPacket packet = new DatagramPacket(data, data.length, ipAddress, port);  
 try { this.socket.send(packet);  
 } catch (IOException e) { e.printStackTrace(); }  
 }  
 public void sendDataToAllClients(byte[] data) {  
 for (Driver p : connectedDriver) { sendData(data, p.ipAddress, p.port); }  
 }  
 private void handleMove(PacketMove packet) {  
 if (getDriver(packet.getUsername()) != null) {  
 int index = getDriverID(packet.getUsername());  
 Driver player = this.connectedDriver.get(index);  
 player.x = packet.getX();  
 player.y = packet.getY();  
 player.setCurrentImageIndex(packet.getCurrentImageIndex()); // Player direction information  
 packet.writeData(this);  
 }  
 }  
}

Then the Client class was created where the game is initiated fir each player containing their IP address as well as their port. Depending on the packets parsed the client will connect, disconnect, or move the cars on this specific client screen. As for the server the methods are then expanded below.

import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
import java.net.SocketException;  
import java.net.UnknownHostException;  
public class Client extends Thread {  
 private InetAddress ipAddress;  
 private DatagramSocket socket;  
 private RaceGround race;  
 public Client(RaceGround race, String ipAddress) {  
 this.race = race;  
 try { this.socket = new DatagramSocket();  
 this.ipAddress = InetAddress.*getByName*(ipAddress);  
 } catch (SocketException e) { e.printStackTrace();  
 } catch (UnknownHostException e) { e.printStackTrace();  
 }  
 }  
 public void run() {  
 while (true) {  
 byte[] data = new byte[1024];  
 DatagramPacket packet = new DatagramPacket(data, data.length);  
 try { socket.receive(packet);  
 } catch (IOException e) { e.printStackTrace();  
 } this.parsePacket(packet.getData(), packet.getAddress(), packet.getPort());  
 }  
 }  
 private void parsePacket(byte[] data, InetAddress address, int port) {  
 String message = new String(data).trim();  
 Packet.PacketStatus status = Packet.*searchPacket*(message.substring(0, 2));  
 Packet packet = null;  
 switch(status) {  
 default:  
 case *INVALID*: break;  
 case *LOGIN*:  
 packet = new PacketLogin(data);  
 handleLogin((PacketLogin)packet, address, port);  
 break;  
 case *DISCONNECT*:  
 packet = new PacketDisconnect(data);  
 System.*out*.println( "["+address.getHostAddress()+":"+port+"] " + ((PacketDisconnect)packet).getUsername()+ " has disconnected...");  
 race.maneuvering.removeDriver(((PacketDisconnect) packet).getUsername());  
 break;  
 case *MOVE*:  
 packet = new PacketMove(data);  
 handleMove((PacketMove)packet);  
 }  
 }  
 private void handleMove(PacketMove packet) {  
 this.race.maneuvering.moveDriver(packet.getUsername(), packet.getX(), packet.getY(), packet.getCurrentImageIndex());  
 }  
 private void handleLogin(PacketLogin packet, InetAddress address, int port) {  
 System.*out*.println( "["+address.getHostAddress()+":"+port+"] " + packet.getUsername()+ " has joined the game...");  
 Driver playerGreen = new Driver(race, packet.getX(), packet.getY(), packet.getUsername(),Media.*playerGreen\_move*, "arrows",address, port);  
 Driver playerBlue = new Driver(race, packet.getX(), packet.getY(), packet.getUsername(),Media.*playerBlue\_move*, "wasd",address, port);  
 race.maneuvering.addDriver(playerGreen); // Join game  
 race.maneuvering.addDriver(playerBlue); // Join game  
 }  
 public void sendData(byte[] data) {  
 DatagramPacket packet = new DatagramPacket(data, data.length, ipAddress, 1331);  
 try { socket.send(packet);  
 } catch (IOException e) { e.printStackTrace();  
 }  
 }  
}

These are three separate classes for each of the packets statuses other than invalid. The disconnection packet only required the username to action the user connection cut. For the login or connection status it’s required a username and a user location on the race ground as x and y coordinates. Lastly, the move packet status includes the username, location, and car image index for display.

public class PacketDisconnect extends Packet {  
 private String username;  
 public PacketDisconnect(byte[] data) { super(01);  
 this.username = readData(data);  
 }  
 public PacketDisconnect(String username) { super(01);  
 this.username = username;  
 }  
 @Override  
 public void writeData(Client client) {  
 client.sendData(getData());  
 } // Sends to client  
 @Override  
 public void writeData(Server server) {  
 server.sendDataToAllClients(getData());  
 } // Sends to server  
 @Override  
 public byte[] getData() {  
 return ("01" + this.username).getBytes();  
 } // Return packet status  
 public String getUsername() {  
 return username;  
 } // Return username  
}

public class PacketLogin extends Packet {  
 private String username;  
 private float x,y;  
 public PacketLogin(byte[] data) {  
 super(00);  
 String[] dataArray = readData(data).split(",");  
 this.username = dataArray[0];  
 this.x = Float.*parseFloat*(dataArray[1]);  
 this.y = Float.*parseFloat*(dataArray[2]);  
 }  
 public PacketLogin(String username, float x, float y) {  
 super(00);  
 this.username = username;  
 this.x = x;  
 this.y = y;  
 }  
 @Override  
 public void writeData(Client client) {  
 client.sendData(getData());  
 } // Sends to client  
 @Override  
 public void writeData(Server server) {  
 server.sendDataToAllClients(getData());  
 } // Sends to server  
 @Override  
 public byte[] getData() {  
 return ("00" + this.username+","+getX()+","+getY()).getBytes();  
 } // Return packet status  
 public String getUsername() { return username; }  
 public float getX() {  
 return x;  
 }  
 public float getY() {  
 return y;  
 }  
}

public class PacketMove extends Packet {  
 private String username;  
 private float x,y;  
 private int currentImageIndex;  
 public PacketMove(byte[] data) {  
 super(02);  
 String[] dataArray = readData(data).split(",");  
 this.username = dataArray[0];  
 this.x = Float.*parseFloat*(dataArray[1]);  
 this.y = Float.*parseFloat*(dataArray[2]);  
 this.currentImageIndex = Integer.*parseInt*(dataArray[3]);  
 }  
 public PacketMove(String username, float x, float y, int currentImageIndex) {  
 super(02);  
 this.username = username;  
 this.x = x;  
 this.y = y;  
 this.currentImageIndex = currentImageIndex;  
 }  
 @Override  
 public void writeData(Client client) {  
 client.sendData(getData());  
 } // Sends to client  
 @Override  
 public void writeData(Server server) {  
 server.sendDataToAllClients(getData());  
 } // Sends to server  
 @Override  
 public byte[] getData() {  
 return ("02" + this.username + ","+this.x +","+this.y+","+this.currentImageIndex).getBytes(); // Get  
 }  
 public String getUsername() {  
 return username;  
 } // Get username  
 public float getX() {  
 return this.x;  
 } // Return x position  
 public float getY() {  
 return this.y;  
 } // Return y position  
 public int getCurrentImageIndex() {  
 return currentImageIndex;  
 } // Return car image  
}

For this stage only the control of one car was possible from two sessions, but one session will not allow input while the last created session will. If the second client session is closed, then the initial session gains full control of the car.

Further research and experiments are required for this to be functional at the requested state.

# Appendix

Code has been attached in .zip files.

Folders separated into Task1, Task2 and Task3.

A read me files has been attached in the .zip file.

# References

‘Answer to “Handling WASD and Arrows simultaneously in JavaScript game”’ (2020) *Stack Overflow*. Available at: <https://stackoverflow.com/a/63059619> (Accessed: 14 March 2023).

*API reference for Java Platform, Standard Edition* (no date). Available at: <https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/Serializable.html> (Accessed: 2 April 2023).

Artokun (2016) ‘Answer to “Using WASD and arrow keys simultaneously”’, *Stack Overflow*. Available at: <https://stackoverflow.com/a/38022775> (Accessed: 14 March 2023).

*Blue Car BluePrint.jpg (400×283)* (no date). Available at: <https://www.the-blueprints.com/modules/vectordrawings/preview/22575-mid.jpg> (Accessed: 21 April 2023).

Brannon (2013) ‘Using WASD and arrow keys simultaneously’, *Stack Overflow*. Available at: <https://stackoverflow.com/q/14824786> (Accessed: 2 April 2023).

*DataLine.Info (Java Platform SE 7 )* (no date). Available at: <https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/DataLine.Info.html> (Accessed: 1 April 2023).

Fery, P. (2022) ‘Answer to “Cannot invoke ‘java.net.URL.toExternalForm()’ because ‘location’ is null Java Netbeans Graphics2D”’, *Stack Overflow*. Available at: <https://stackoverflow.com/a/74549939> (Accessed: 2 April 2023).

*Green Car BluePrint.jpg (400×283)* (no date). Available at: <https://www.the-blueprints.com/modules/vectordrawings/preview/20357-mid.jpg> (Accessed: 3 March 2023).

‘Java Swing - JPanel With Examples’ (2018) *GeeksforGeeks*, 5 June. Available at: <https://www.geeksforgeeks.org/java-swing-jpanel-with-examples/> (Accessed: 3 March 2023).

Jon (2017) ‘Answer to “Handling WASD and Arrows simultaneously in JavaScript game”’, *Stack Overflow*. Available at: <https://stackoverflow.com/a/44492605> (Accessed: 11 February 2023).

Kamermans, M. ‘Pomax’ (2013) ‘Answer to “Using WASD and arrow keys simultaneously”’, *Stack Overflow*. Available at: <https://stackoverflow.com/a/14836059> (Accessed: 9 February 2023).

Nol (2023) ‘Answer to “Cannot invoke ‘java.net.URL.toExternalForm()’ because ‘location’ is null Java Netbeans Graphics2D”’, *Stack Overflow*. Available at: <https://stackoverflow.com/a/75084980> (Accessed: 17 March 2023).

Patashu (2013) ‘Answer to “Using WASD and arrow keys simultaneously”’, *Stack Overflow*. Available at: <https://stackoverflow.com/a/14827876> (Accessed: 3 April 2023).

Queeg (2022) ‘Answer to “Java Socket Programming MultiPlayer Kart Racing Game”’, *Stack Overflow*. Available at: <https://stackoverflow.com/a/71464339> (Accessed: 11 April 2023).

Radiodef (2014) ‘Answer to “Java Sound plays once and then doesn’t play again”’, *Stack Overflow*. Available at: <https://stackoverflow.com/a/27583249> (Accessed: 20 March 2023).

Seeker, H. (2022) ‘Java Socket Programming MultiPlayer Kart Racing Game’, *Stack Overflow*. Available at: <https://stackoverflow.com/q/71463901> (Accessed: 8 February 2023).

*Settings Page, Project Designer* (2010). Available at: <https://learn.microsoft.com/en-us/previous-versions/cftf714c(v=vs.100)> (Accessed: 21 April 2023).

Sousa, P. (2022) ‘Cannot invoke “java.net.URL.toExternalForm()” because “location” is null Java Netbeans Graphics2D’, *Stack Overflow*. Available at: <https://stackoverflow.com/q/74549838> (Accessed: 11 March 2023).

splash27 (2017) ‘Handling WASD and Arrows simultaneously in JavaScript game’, *Stack Overflow*. Available at: <https://stackoverflow.com/q/44470075> (Accessed: 1 April 2023).

user2279603 (2014) ‘Java Sound plays once and then doesn’t play again’, *Stack Overflow*. Available at: <https://stackoverflow.com/q/27583006> (Accessed: 2 February 2023).