COMP90015 Distributed System

Project 2

Distributed Race Car

Wei Han: 523979

Lei Tang: 518701

Table of Contents

1. System Description ……..…….. 3

2.System Design…….. ……..…….. 4

2.1 Client Side Design ……..……..…. 4

2.2 Server Side Design ……. 5

2.3 Communication ……. 6

3. System Implementation…….…. 7

3.1 Client Side ……... 7

3.2 Server Side …….…….. 10

4. Testing ……... 10

5. Group Member Contribution …… 13

6. Source Code ……..…… 14

6.1 Client Side ……... 14

6.2 Server Side …….…….. 16

1. System Description:

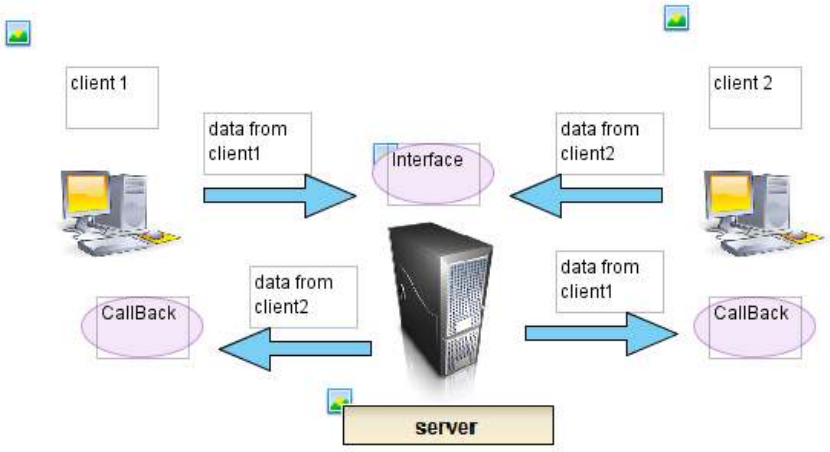
This project is design a distributed race car system which allows two persons to drive their cars on their own clients while both individuals can see two cars on the same interface. Support min. 2 participants in racing, but scalable design and implementation supporting larger number of participants will be well regarded.

The project is based on the Java RMI Remote Method Invocation technology. RMI is an evolution of the client/server architecture. Like most client/server architecture, RMI component have server side and client side. Compare to normal client/server architecture, RMI approach is a more flexible , for example, it is much easy to pass functionality from server to client, also easy to pass functionality from client to server with RMI.

In this application, server side acts as a communicated bridge between two clients. The server could receive the invocation from client 1 and then call the invoke function on the client 2, after that response the data from client 2 to client 1, and from that, the game could process simultaneously on both clients. So the functionality should exchange between the server and client, this make RMI suitable to this application.

For the application design based on RMI, the first step is to design an interface extends from Remote interface.

In the server side, the service interface class is: *RaceCarServiceInterface()*. It defines the interface for the server to relay the message between two clients. The interface class in client to server is called *Callback()*. It provides an interface for server to direct to invoke and control the client state and function. The system communication architecture can show below. The more details analysis will provided in the subsequent sections.



For a distributed game system, besides the communication interface design, the user interface design is also important. In this design, the user interface design is implemented in the client side. It is based on the javax.swing.* framework. The car display is drawn in the JPanel component. And other control components are implemented in the JFrame component. These control components including connect to server, starting game, stopping game and control the race car speed or direction. The speed and direction control keys are also implemented in the client side with JFrame extended class.

The race car view itself is designed with pre-made pictures. 16 different pictures are made to represent 16 directions of one car. With direction control button or control keys, the corresponding direction car view picture can display on the track. The car status information (position, crashed, win/lost) can exchange via server among different clients.

To run this program, firstly we should start the server, then start two clients on different PCs, after that, both clients should connect to the server, finally, when both clients start, we can enjoy the game.

2. System design:

2.1 Client Side Design:

The client side includes 7 classes located in four files, which includes four public classes:

CallBack class which provides the client interface to server;

RaceCar class which implements the Callback interface is the Race Car's state control class, every RaceCar instance corresponding to a Race Car, for RaceCar implements the Callback interface, so the server also can direct invoke RaceCar method and direct control Car's status;

RaceCarServiceInterface class: this is the RMI server interface to client;

RaceCarFrame class is the main class in the client side. This class process user input, output view display, car status (position, crash, collide or win/lost) detect and control, it also report the client's local car's status to the server via RaceCarServiceInterface interface;

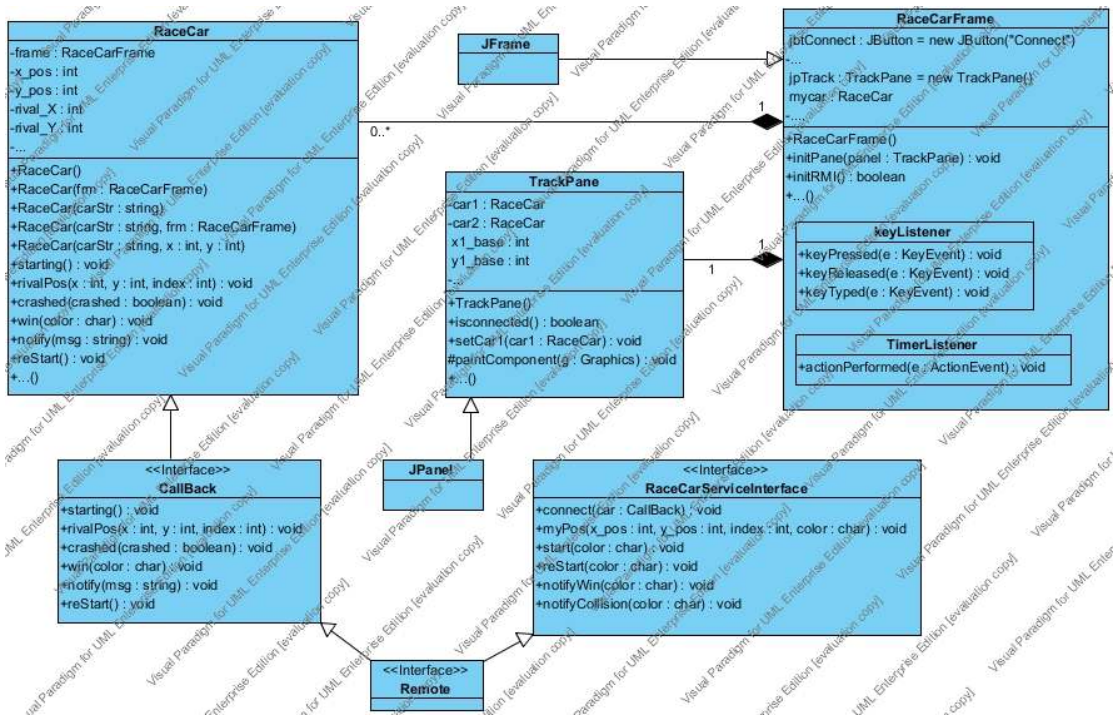
The fifth class: TrackPane class located in the same file of RaceCarFrame is just display the track and two Race Cares.

The last two classes are the inner class inside the RaceCarFrame class:

class KeyListener implements KeyListener: process key event.

class TimerListener implements ActionListener: process timer event.

The client side UML class diagram can show in below:

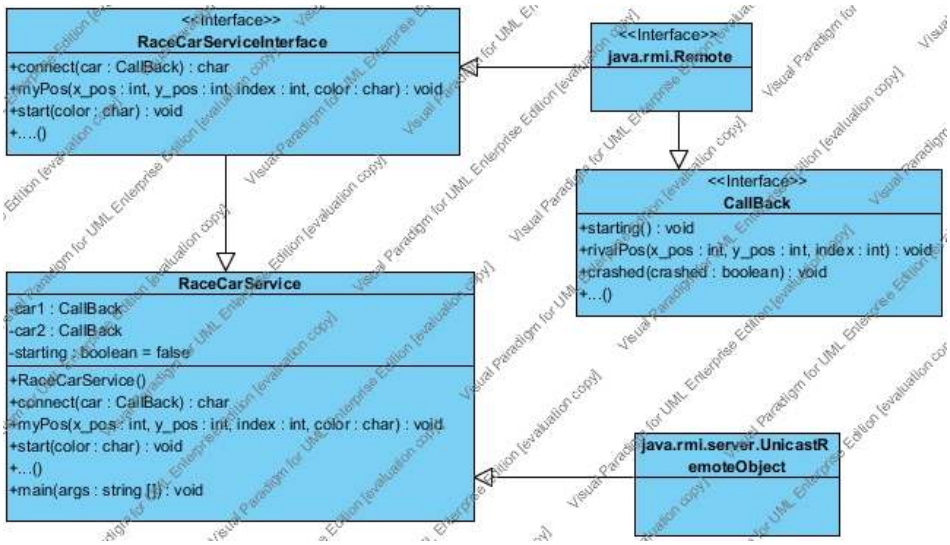


2.2. Server Side Design:

The server side includes 3 classes, while *CallBack* class and *RaceCarServiceInterface* class are same to server side.

The third class, *RaceCarService*, which implements the *RaceCarServiceInterface* interface, starts the RMI service. Its implements *RaceCarServiceInterface* methods can relay the message between two clients and made distributed Race Car game possible.

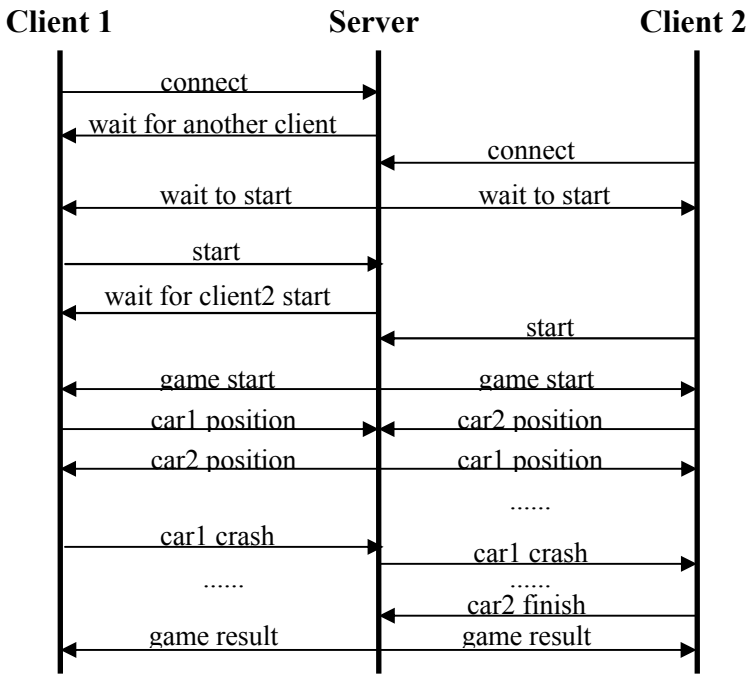
The UML classes diagram in server side is show in below:



2.3 Communication:

The communication in this project is kind of client/server model based on Java RMI. When there are some data need send from client to server, client should invoke server side RMI interface *RaceCarServiceInterface*; via the RMI communication mechanism, the process data will send from client to server. When the server receive the data from one client, in this application, based the message type, it will change the game status, or invoke the client side interface *CallBack* to send info to the peer client side.

The communication sequence could show in below:



As above communication sequence shows, the client 1 connects Server first, server will response "wait for another client". After client 2 connects, the server will tell both clients to start the game. Then, both clients can start game by pressing the "start button", and the server will start the game on both clients simultaneously. When the game is process, client1 could send the position of its car to client2 through the server, and the same as client2, this procedure will be executed repeatedly until one of the both cars finishes the game.

3. System Implementation:

From above system design, our system is implemented as following:

3.1 Client Side :

1). CallBack Interface:

This interface is used to be called by the other client through the server, to present the information of the other car, including the position, whether it is crashed, whether it wins .etc.

Main Methods In CallBack:

starting() void: Be called when the competitor start the game.
rivalPos(int, int, int) void: Be called every mini time to show the position of the competitor's car.
crashed(boolean) void: Be called when the competitor's car is crashed.
win(char) void: Be called when the competitor's car wins.
notify(String) void: Be called when the competitor want send some message.
reStart() void: Be called when the competitor require to restart the game.

2). RaceCar Class

This class is used to define the functions of keyboard events, add the listener for the keyevents.

Class Variables:
MaxCarNum(int) : Stand for the total picture numbers for one car.
frame(RaceCarFrame): Stand for the game interface object.
x_pos(int): Stand for the x-axis of the user's car.
....
index(int): Stand for the picture order of the user's car in total 16 pictures.
rival_index(int): Stand for the picture order of the competitor's car in total 16 pictures.
top(boolean): Stand for the finish line.

Class Methods:

…

rival_pos(int, int, int) void: Set the picture path, x-axis and y-axis for the competitor'car.

crashed(boolean) void: Set the result to crash, called by server.

win(char) void: Set the game result to win or lose, called by server.

notify(String) void: Present the message, called by server.

reStart() void: Restart game, init the tack and two cars again, called by server.

...

getX_pos() int: Get the x-axis of the current car.

setX_pos(int) void: Set the x-axis of the current car.

...

getLeftCar(String) String: Get the picture path of the car left to the current car.

getRightCar(String) String: Get the picture path of the car right to the current car.

getCar(String, int) String: Get the picture path of the needed car by the path of the current car and the current order.

getCarIndex(String) int: Get the picture order of the current car.

3).RaceCarFrame Class:

This class is mainly used to init the game interface, button, track and the RMI protocol.

Class Variables:

jbtConnect(JButton): Stand for a button used to connect the server.

jbtStart(JButton): Stand for a button for client to start the game.

....

PI(double): Stand for circumference

spinTimer(Timer): Stand for a Timer object

Class Methods:

RaceCarFrame(): contructed function. Init the interface of the game, including the functions of every button and adding the listener for each button.

initPane(TrackPane) void: Init the track of the game, including the border of the track and the cars of two competitor.

initRMI() void: init the RMI protocol.

collision(RaceCar, RaceCar) boolean: Judge whether two cars crash with each other.

collision(RaceCar) boolean: Judge whether one car crash to the border of the game.

main(String[]) void: the main function, init the original parameters of the frame.

4) class KeyListener implements KeyListener:

This class is used to define the functions of keyboard events, add the listener for the keyevents.

Methods:
keyPressed(KeyEvent) void: Add a listener for the press action for the key.
keyReleased(KeyEvent) void: Add a listener for the release action for the key.
keyTyped(KeyEvent) void: Add a listener for the type action for the key.

5) class TimerListener implements ActionListener:

Methodes:
actionPerformed(ActionEvent) void: Process the game, in particularly change the pictures and the positions for the race cars every mini time.

6) class TrackPane extends JPanel:

This class is used to init the track and race cars of the game, including the pictures and the positions of two cars.

Variables:
car1(RaceCar): Stand for the user's car.
....
Functions:
TrackPane(): constructed function.
isConnected() boolean: Return the result whether the client connect to the server.
setConnected(boolean) void: Set the result whether the client connect to the server.
setCar1(RaceCar) void: Set the coordinate data of car1's position.
setCar2(RaceCar) void: Set the coordinate data of car2's position.
paintComponent(Graphics) void: Set the twos pictures of these twos cars to the related position on the interface of the game and draw the track for the game.

7).RaceCarServiceInterface Interface:

This interface is used to call the other client through the server, to present the information of the current car, including the position, whether it is crashed, whether it wins .etc.

Functions:
myPos(int, int, int, char) void: Send the position of current car to the other client.
start(char) void: Call server when current user starts the game.
reStart(char) void: Call server when current user restarts the game.
notifyWin(char) void: Call server when current user wins the game.
notifyCollision(char) void: Call server when current car is crashed.

3.2 Server Side :

1) CallBack Interface:

This class is used to be called by the client to the server, to present the information of this client, and the sent to the other client by the server. Its definition is same as client side.

2). RaceCarService Class:

This class is used to the functions of the server.

Variables:

car1(CallBack) : Stand for the one client.
car2(CallBack): Stand for the other client.
starting(boolean): Stand for the result that whether the user start the game.

Functions:

RaceCarService(): constructed function.
connect(CallBack) char: Return the color of the connecting car.
myPos(int, int, int, char) void: Send the position of one client to the other.
start(char) void: Start the game.
reStart(char) void: Restart the game.
notifyWin(char) void: Send the game result to both clients.
notifyCollision(char) void: Send the crashed result to both clients if there is.
main(String[]) void: Start the server

3). RaceCarServiceInterface Interface

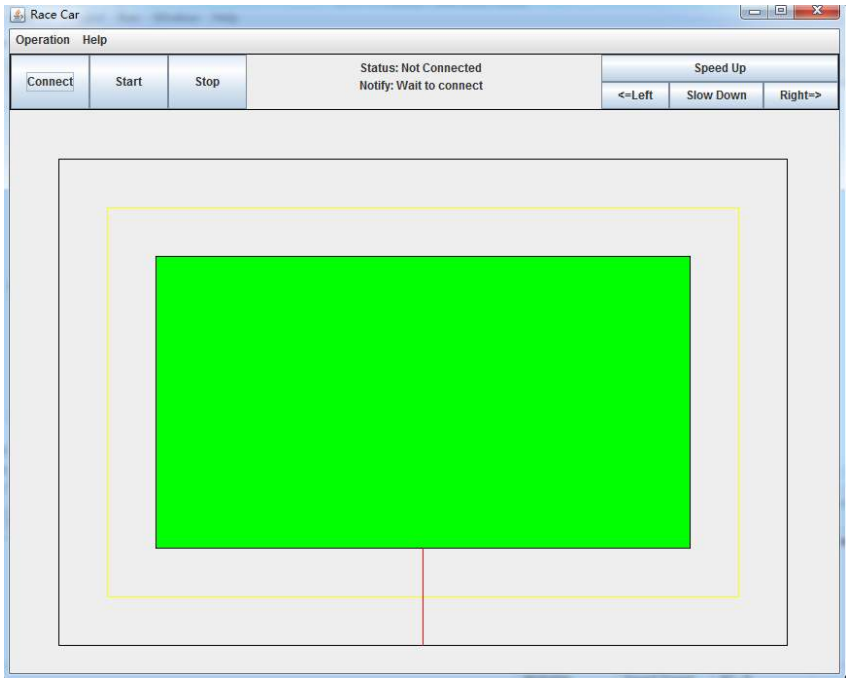
This class is used to call the other client through the server, to present the information of the current car, including the position, whether it is crashed, whether it wins .etc. The definition description is same as client side.

4. Testing

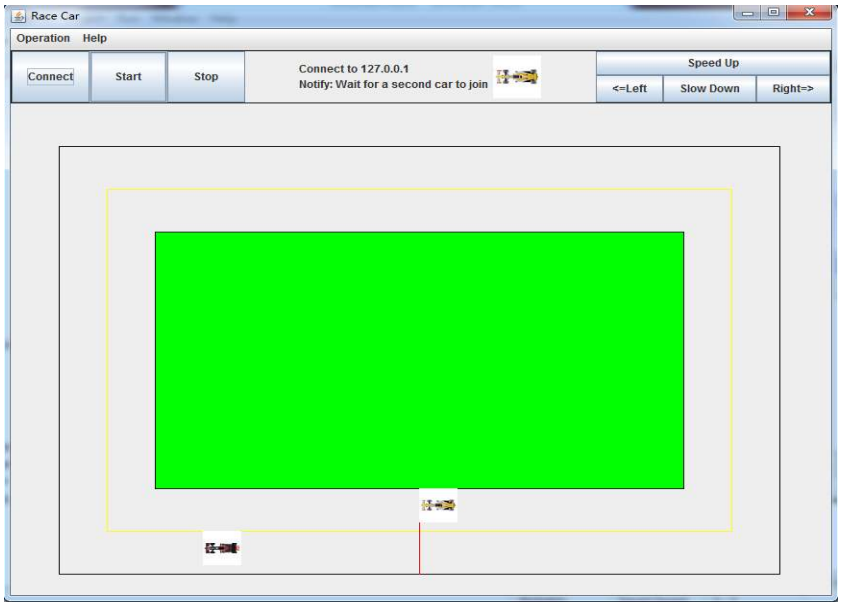
Firstly, start the server, if successful, this information will be presented:

"Server RaceCarService[UnicastServerRef [liveRef: [endpoint:[10.1.1.11:50494](local),objID:[-39ef6b3a:133381e94ea:-7fff, 3228638537215849796]]]] registered".

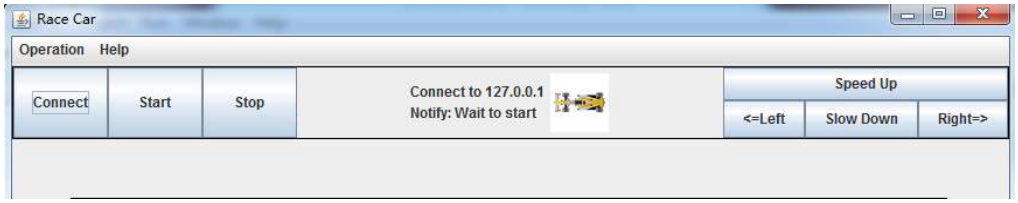
Secondly, Start twos clients, the picture below is the basic client interface(not connected): on top of the interface, there are 7 buttons, used to connecting server, start game, stop game, turn left, turn right, accelerate and decelerate expectedly



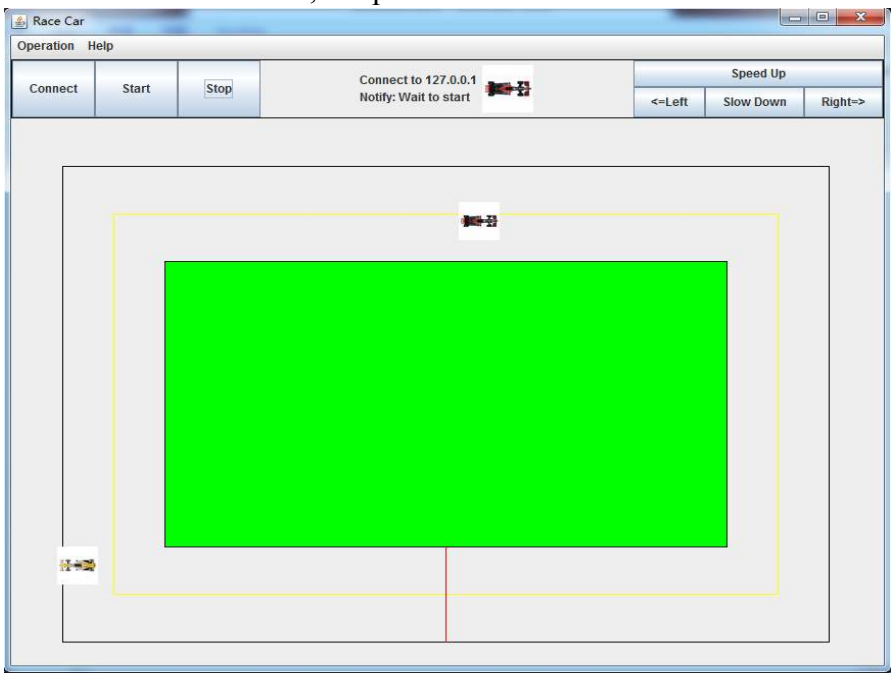
The third step: both clients should connect the server: this is the picture a client connected to the server, the server host will be presented, and notify to wait for the other user.



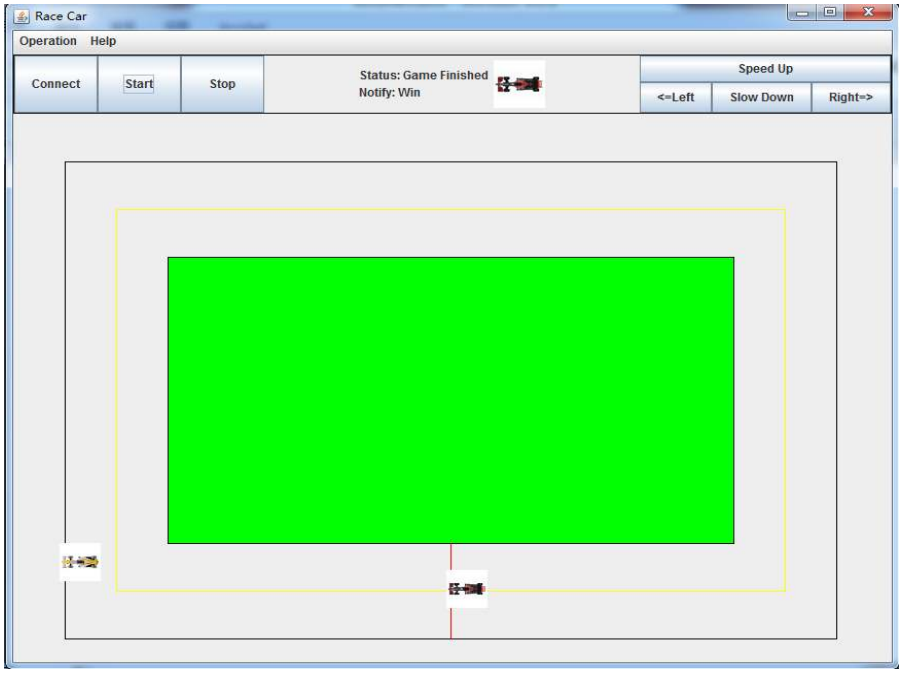
When both users connect, the information changes to "wait to start".



When one of the car crashed, the picture is like this :



This is the end of the game. The black car wins.



5. Group Member Contribution:

This project is finished by the group formed by: Wei Han (523979) and Lei Tang (518701). Their contribution to the project can show in below table:

	Wei Han (523979)	Lei Tang (518701)
Distributed Race Car Software System	Software system design and programming	Race Car views image pictures designing System testing
Project report	Modification for the report draft Section 1: System Description rewriting Section 2: System Design rewriting Final report's editing and revising	Project report draft writing System architecture and UML class diagrams drawing

6. Appendix – Source Code:

6.1 Sever Side:

RaceServiceInterface.java

```
/*
 * Wei Han's New File
 *    Created on 30/09/2011
 */
import java.rmi.*;
public interface RaceCarServiceInterface extends Remote{
        public char connect(CallBack car) throws RemoteException;
        public void myPos(int x_pos, int y_pos, int index, char color) throws RemoteException;
        public void start(char color) throws RemoteException;
        public void reStart(char color) throws RemoteException;
        public void notifyWin(char color)throws RemoteException;
        public void notifyCollision(char color) throws RemoteException;
}
```

Callback.java

```
/*
 * Wei Han's New File
 *    Created on 30/09/2011
 */
import java.rmi.*;
public interface CallBack extends Remote {
        public void starting() throws RemoteException;
        public void rivalPos(int x, int y, int index)throws RemoteException;
        public void crashed(boolean crashed) throws RemoteException;
        public void win(char color) throws RemoteException;
        public void notify(String msg) throws RemoteException;
        public void reStart() throws RemoteException;
}
```

RaceService.java

```
/*
 * Wei Han's New File
 *    Created on 30/09/2011
```

```
 */
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class RaceCarService extends UnicastRemoteObject implements RaceCarServiceInterface{
	private CallBack car1 = null;	//will be created as 'Y' car
	private CallBack car2 = null;	// will be created as 'G' car
	private boolean starting = false;
	public RaceCarService() throws RemoteException {
		super();
	}

	public char connect(CallBack car) throws RemoteException {
		if (car1 == null) {
		// car1 registered
			car1 = car;
			car1.notify("Wait for a second car to join");
			System.out.println("Car1 Connecting");
			return 'Y';	//yellow car
			}
		else if (car2 == null) {
		// car2 registered
			car2 = car;
			car2.notify("Wait to start");
			car1.notify("Wait to start");
			System.out.println("Car2 Connecting");
			return 'G';	//Green Car
			}
		else {
		// Already two cars
			car.notify("Two cars are already in the game");
			System.out.println("Another Car Connecting");
			return ' ';
			}
		}

	public void myPos(int x_pos, int y_pos,int index, char color) throws RemoteException{
		if(color == 'Y'){
			car2.rivalPos(x_pos, y_pos, index);
		}
		else {
			car1.rivalPos(x_pos, y_pos,index);
		}
```

```
    }
    public void start(char color) throws RemoteException {
        if(starting){
            car1.starting();
            car2.starting();
        }
        else starting = true;
    }
    public void reStart(char color) throws RemoteException {
        car1.reStart();
        car2.reStart();
        starting = false;
    }
    public void notifyWin(char color)throws RemoteException{
        car1.win(color);
        car2.win(color);
    }

    public void notifyCollision(char color) throws RemoteException{
        ;
    }
    public static void main(String[] args) {
        try {
            RaceCarServiceInterface server = new RaceCarService();
            Registry registry = LocateRegistry.createRegistry(1099); //.getRegistry(); is not work here ?

            if (registry == null) {
                System.out.println("server is null");
               throw new NullPointerException("cannot bind to null");
            }

            registry.rebind("RaceCarService", server);
            System.out.println("Server " + server + " registered");
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

6.2 Client Side

Callback.java

```
/*
 * Wei Han's New File
 *    Created on 30/09/2011
 */
import java.rmi.*;
public interface CallBack extends Remote {
        public void starting() throws RemoteException;
        public void rivalPos(int x, int y, int index)throws RemoteException;
        public void crashed(boolean crashed) throws RemoteException;
        public void win(char color) throws RemoteException;
        public void notify(String msg) throws RemoteException;
        public void reStart() throws RemoteException;
}
```

RaceCarServiceInterface.java

```
/*
 * Wei Han's New File
 *    Created on 30/09/2011
 */
import java.rmi.*;

public interface RaceCarServiceInterface extends Remote{
        public char connect(CallBack car) throws RemoteException;
        public void myPos(int x_pos, int y_pos,int index,    char color) throws RemoteException;
        public void start(char color) throws RemoteException;
        public void reStart(char color) throws RemoteException;
        public void notifyWin(char color)throws RemoteException;
        public void notifyCollision(char color) throws RemoteException;
}
```

RaceCarFrame.java

```
/*
 * Wei Han's New File
 *    Created on 30/09/2011
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.border.*;
import java.rmi.*;
import java.rmi.registry.*;

public class RaceCarFrame extends JFrame {
        JButton jbtConnect = new JButton("Connect");
        JButton jbtStart = new JButton("Start");
        JButton jbtStop    = new JButton("Stop");
        JButton jbtAcc     = new JButton("Speed Up");
        JButton jbtDec     = new JButton("Slow Down");
        JButton jbtLeft    = new JButton("<=Left");
        JButton jbtRight = new JButton("Right=>");

        JLabel jlblStatus = new JLabel("Status: Not Connected");
        JLabel jlblNotify = new JLabel("Notify: Wait to connect");
        JLabel jlblCar = new JLabel("");

        TrackPane jpTrack = new TrackPane();

        private JMenuItem jmiRestart,jmiAbout;

        private String yCarStr = "image/car1/car0.gif";
        private String gCarStr = "image/car2/car0.gif";

        private String hostName="";
        private RaceCarServiceInterface raceCarService;
        private char color;

        RaceCar myCar;
        RaceCar rivalCar;

        static final double PI = 3.14159;

        static Timer spinTimer;

        public RaceCarFrame(){
                JPanel jpStart = new JPanel();
                jpStart.setLayout(new GridLayout(1,3));
                jpStart.add(jbtConnect);
                jpStart.add(jbtStart);
                jpStart.add(jbtStop);

                JPanel jpControl = new JPanel();
                jpControl.setLayout(new BorderLayout());
```

```
jpControl.add(jbtLeft,BorderLayout.WEST);
jpControl.add(jbtRight,BorderLayout.EAST);
jpControl.add(jbtAcc,BorderLayout.NORTH);
jpControl.add(jbtDec,BorderLayout.CENTER);

JPanel jpStatus = new JPanel();
JPanel jpNotify = new JPanel();
jpNotify.setLayout(new GridLayout(2,1));
jpNotify.add(jlblStatus);
jpNotify.add(jlblNotify);
jpStatus.add(jpNotify,BorderLayout.CENTER);
jpStatus.add(jlblCar,BorderLayout.WEST);

JPanel jpButton = new JPanel();
jpButton.setLayout(new BorderLayout());
jpButton.add(jpStart,BorderLayout.WEST);
jpButton.add(jpStatus,BorderLayout.CENTER);
jpButton.add(jpControl,BorderLayout.EAST);
jpButton.setBorder(new LineBorder(Color.BLACK));

add(jpTrack,BorderLayout.CENTER);
add(jpButton,BorderLayout.NORTH);

JMenuBar jmb = new JMenuBar();
setJMenuBar(jmb);

JMenu operationMenu = new JMenu("Operation");
operationMenu.add(jmiRestart = new JMenuItem("Restart",'R'));

JMenu helpMenu = new JMenu("Help");
helpMenu.add(jmiAbout = new JMenuItem("About",'A'));

jmb.add(operationMenu);
jmb.add(helpMenu);
spinTimer = new Timer(50,new TimerListener());
addKeyListener(new keyListener());

jbtConnect.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){

        while(hostName.length()<1){
                hostName = JOptionPane.showInputDialog("Please input the Race Server Address:");
        }
        if(initRMI()){
```

```
                jpTrack.setConnected(true);
                initPane(jpTrack);
            }
        }
        });
        jbtConnect.addKeyListener(new keyListener());

        jbtStart.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                try{
                    raceCarService.start(myCar.getColor());
                }
                catch(RemoteException ex){
                }
            }
        });
        jbtStart.addKeyListener(new keyListener());

        jbtStop.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                spinTimer.stop();
            }
        });
        jbtStop.addKeyListener(new keyListener());

        jbtLeft.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                myCar.turnLeft();
            }
        });
        jbtLeft.addKeyListener(new keyListener());

        jbtRight.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                myCar.turnRight();
            }
        });
        jbtRight.addKeyListener(new keyListener());

        jbtAcc.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                myCar.speedUp();
            }
        });
```

```
        jbtAcc.addKeyListener(new keyListener());

        jbtDec.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                myCar.slowDown();
            }
        });
        jbtDec.addKeyListener(new keyListener());

        jmiRestart.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                try{
                    raceCarService.reStart(myCar.getColor());
                }
                catch(RemoteException ex){
                }
                initPane(jpTrack);
            }
        });
        jmiRestart.addKeyListener(new keyListener());

        jmiAbout.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                JOptionPane.showMessageDialog(null, "Race Car Game");
            }
        });
        jmiAbout.addKeyListener(new keyListener());
    }

    public void initPane(TrackPane panel){
        System.out.println("System.getProperty(\"user.dir\") is:"+System.getProperty("user.dir"));

        if(color == 'Y'){
            panel.setCar1(myCar);
            panel.setCar2(rivalCar);

            myCar.setCrashed(false);
            myCar.setCur_Car(yCarStr);
            rivalCar.setCur_Car(gCarStr);

            myCar.setWin(false);
            myCar.setTop(false);
        }
        else {
```

```
                panel.setCar2(myCar);
                panel.setCar1(rivalCar);

                myCar.setCrashed(false);
                myCar.setCur_Car(gCarStr);
                rivalCar.setCur_Car(yCarStr);

                myCar.setWin(false);
                myCar.setTop(false);
        }
        spinTimer.stop();
        panel.repaint();
}

public boolean initRMI() {
        try {
                Registry registry = LocateRegistry.getRegistry(hostName);
                raceCarService = (RaceCarServiceInterface)registry.lookup("RaceCarService");
                try{
                        myCar = new RaceCar(this);
                        rivalCar = new RaceCar();

                        color = raceCarService.connect(myCar);
                        myCar.setColor(color);

                        if(color == 'Y'){
                                myCar.setCur_Car(yCarStr);
                                myCar.setColor('Y');

                                rivalCar.setCur_Car(gCarStr);
                                rivalCar.setColor('G');
                                jlblStatus.setText("Connect to "+hostName);

                                ImageIcon yCarIcon = new ImageIcon(yCarStr);
                                jlblCar.setIcon(yCarIcon);
                        }
                        else if(color == 'G'){
                                myCar.setCur_Car(gCarStr);
                                myCar.setColor('G');

                                rivalCar.setCur_Car(yCarStr);
                                rivalCar.setColor('Y');
                                jlblStatus.setText("Connect to "+hostName);
```

```
                        ImageIcon gCarIcon = new ImageIcon(gCarStr);
                        jlblCar.setIcon(gCarIcon);
                    }
                    else return false;
                }
                catch(RemoteException e){
                    e.printStackTrace();
                    return false;
                }

                return true;
            }
            catch (Exception e){
                System.out.println(e);
                return false;
            }
        }

        class keyListener implements KeyListener{
            public void keyPressed(KeyEvent e){
                switch(e.getKeyCode()){
                    case KeyEvent.VK_DOWN:  myCar.slowDown();break;
                    case KeyEvent.VK_UP:    myCar.speedUp();break;
                    case KeyEvent.VK_LEFT:  myCar.turnLeft();break;
                    case KeyEvent.VK_RIGHT: myCar.turnRight();break;
                }
            }
            public void keyReleased(KeyEvent e){
            }
            public void keyTyped(KeyEvent e){
            }
        }

        class TimerListener implements ActionListener{
            public void actionPerformed(ActionEvent e){

                int car1Index = myCar.getIndex();
                int speed1 = myCar.getSpeed();
                int x_pos,y_pos;

                int car1DeltaX = 0-(int)(speed1*Math.cos(car1Index*22.5/360.0*2*PI));
                int car1DeltaY = 0-(int)(speed1*Math.sin(car1Index*22.5/360.0*2*PI));

                x_pos = myCar.getX_pos()+car1DeltaX;
```

```
                y_pos = myCar.getY_pos()+car1DeltaY;

                if(collision(myCar)) myCar.setCrashed(true);
                if(collision(rivalCar)) rivalCar.setCrashed(true);

                if(collision(myCar,rivalCar)){
                        myCar.setCrashed(true);
                        rivalCar.setCrashed(true);
                }

                if(!myCar.isCrashed()){
                        myCar.setX_pos(x_pos);
                        myCar.setY_pos(y_pos);

                        try{
                                raceCarService.myPos(x_pos, y_pos, myCar.getIndex(), myCar.getColor());
                        }
                        catch(RemoteException ex){

                        }
                }
                rivalCar.setX_pos(myCar.getRival_X());
                rivalCar.setY_pos(myCar.getRival_Y());
                rivalCar.setCur_Car(myCar.getCar(rivalCar.getCur_Car(), myCar.getRival_index()));
                if(myCar.isWin()){
                        try{
                                raceCarService.myPos(x_pos, y_pos,myCar.getIndex(), myCar.getColor());
                                raceCarService.notifyWin(myCar.getColor());
                        }
                        catch(RemoteException ex){
                        }
                }
                if(rivalCar.isWin()){
                }
                ImageIcon carIcon = new ImageIcon(myCar.getCur_Car());
                jlblCar.setIcon(carIcon);

                jpTrack.repaint();
        }
}
        public boolean collision(RaceCar car1,RaceCar car2){    //two car collision
                if((Math.abs(car1.getX_pos()-car2.getX_pos())<40)&&(Math.abs(car1.getY_pos()-car2.getY_pos())<40))return true;
                else return false;
        }
```

```
        public boolean collision(RaceCar car){    //collision with track
            int x_pos    = car.getX_pos();
            int y_pos    = car.getY_pos();

            if((x_pos<50)||(y_pos<50)||(x_pos>760)||(y_pos>510)) return true;
            else if((x_pos>110)&&(y_pos>110)&&(x_pos<700)&&(y_pos<450)) return true;
            else return false;
        }
    public static void main(String[] args) {
        RaceCarFrame frame = new RaceCarFrame();
        frame.setTitle("Race Car");
        frame.setSize(870,700);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
class TrackPane extends JPanel {
    private RaceCar car1;
    private RaceCar car2;
    int x1_base;
    int y1_base;
    int x2_base;
    int y2_base;
    boolean connected = false;
    public TrackPane(){
    }
    public boolean isConnected() {
        return connected;
    }
    public void setConnected(boolean connected) {
        this.connected = connected;
    }
    public void setCar1(RaceCar car1) {
        x1_base = 425;
        y1_base = 450;
        car1.setX_pos(x1_base);
        car1.setY_pos(y1_base);
        this.car1 = car1;
    }
    public void setCar2(RaceCar car2) {
        x2_base = 200;
        y2_base = 500;
```

```
            car2.setX_pos(x2_base);
            car2.setY_pos(y2_base);
            this.car2 = car2;
        }
        protected void paintComponent(Graphics g){
            super.paintComponent(g);
            Color c1 = Color.green;
            g.setColor(c1);
            g.fillRect(150, 150, 550, 300);    //inner side
            Color c2 = Color.black;
            g.setColor(c2);
            g.drawRect(50, 50, 750, 500);       //outer side
            g.drawRect(150, 150, 550, 300);
            Color c3 = Color.yellow;
            g.setColor(c3);
            g.drawRect(100, 100, 650, 400);
            Color c4 = Color.RED;
            g.setColor(c4);
            g.drawLine(425, 450, 425, 550);

            if(connected){
                ImageIcon car1Icon = new ImageIcon(this.car1.getCur_Car());
                ImageIcon car2Icon = new ImageIcon(this.car2.getCur_Car());

                Image car1 = car1Icon.getImage();
                Image car2 = car2Icon.getImage();

                g.drawImage(car1,this.car1.getX_pos(),this.car1.getY_pos(),40,40,this);
                g.drawImage(car2,this.car2.getX_pos(),this.car2.getY_pos(),40,40,this);
            }
        }
}
```

RaceCar.java

```
/*
 * Wei Han's New File
 *    Created on 30/09/2011
 */
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class RaceCar extends UnicastRemoteObject implements CallBack{
        final int MaxCarNum = 16;
        private RaceCarFrame frame;
```

```
private int x_pos;
private int y_pos;
private int rival_X;
private int rival_Y;
private int speed = 5;
private String cur_Car;
private char color;
private int index;
private int rival_index;
private boolean crashed = false;
private boolean win = false;
private boolean top = false;
public RaceCar() throws RemoteException{
}
public RaceCar(RaceCarFrame frm) throws RemoteException{
        this.frame = frm;
}
public RaceCar(String carStr) throws RemoteException{
        this.cur_Car = carStr;
}
public RaceCar(String carStr,RaceCarFrame frm) throws RemoteException{
        this.cur_Car = carStr;
        this.frame = frm;
}
public RaceCar(String carStr,int x, int y) throws RemoteException{
        this.cur_Car = carStr;
        this.x_pos = x;
        this.y_pos = y;
}
public void starting() throws RemoteException{
        RaceCarFrame.spinTimer.start();
}
public void rivalPos(int x, int y, int index)throws RemoteException{
        this.rival_X = x;
        this.rival_Y = y;
        this.rival_index = index;
}
public void crashed(boolean crashed) throws RemoteException{
        setCrashed(crashed);
}
public void win(char color) throws RemoteException{
        if(color == this.color){
                frame.jlblStatus.setText("Status: Game Finished");
```

```
                frame.jlblNotify.setText("Notify: Win");
        }
        else{
                frame.jlblStatus.setText("Status: Game Finished");
                frame.jlblNotify.setText("Notify: Lost");
        }
        RaceCarFrame.spinTimer.stop();
}
public void notify(String msg)throws RemoteException{
        frame.jlblNotify.setText("Notify: "+msg);
}
public void reStart() throws RemoteException{
        this.frame.initPane(this.frame.jpTrack);
}
public void setColor(char color) {
        this.color = color;
}
public char getColor() {
        return color;
}
public boolean isCrashed() {
        return crashed;
}
public void setCrashed(boolean crashed) {
        this.crashed = crashed;
}
public boolean isWin() {
        if(top){
                if(this.x_pos<=425&&this.y_pos>=450&&this.y_pos<=510){
                        this.win=true;
                        return win;
                }
                else {
                        return win;
                }
        }
        else {
                if(this.x_pos>=425&&this.y_pos>=50&&this.y_pos<=110){
                        this.top=true;
                }
                return win;
        }
}
public void setWin(boolean win) {
```

```
        this.win = win;
    }
    public void setTop(boolean top) {
        this.top = top;
    }
    public int getIndex() {
        index    = getCarIndex(cur_Car);
        return index;
    }
    public void setIndex(int index) {
        this.index = index;
    }
    public int getRival_index() {
        return rival_index;
    }
    public void setRival_index(int rival_index) {
        this.rival_index = rival_index;
    }
    public int getX_pos() {
        return x_pos;
    }
    public void setX_pos(int x_pos) {
        this.x_pos = x_pos;
    }
    public int getY_pos() {
        return y_pos;
    }
    public void setY_pos(int y_pos) {
        this.y_pos = y_pos;
    }
    public int getRival_X() {
        return rival_X;
    }
    public void setRival_X(int rival_X) {
        this.rival_X = rival_X;
    }
    public int getRival_Y() {
        return rival_Y;
    }
    public void setRival_Y(int rival_Y) {
        this.rival_Y = rival_Y;
    }
    public int getSpeed() {
        return speed;
```

```
}
public void setSpeed(int speed) {
        this.speed = speed;
}
public String getCur_Car() {
        return cur_Car;
}
public void setCur_Car(String cur_Car) {
        this.cur_Car = cur_Car;
}
void speedUp(){
        speed +=1;
        if(speed>10)speed =10;
}
void slowDown(){
        speed -=1;
        if(speed<0) speed = 0;
}
void turnLeft(){
        cur_Car = getLeftCar(cur_Car);
}
void turnRight(){
        cur_Car = getRightCar(cur_Car);
}
public String getLeftCar(String car){
        index = getCarIndex(car);
        if(index >= 0){
                index--;
                if(index<0) index=MaxCarNum-1;
                car = getCar(car,index);
        }
        return car;
}
public String getRightCar(String car){
        index = getCarIndex(car);
        if(index >= 0){
                index++;
                if(index >= MaxCarNum) index= 0;
                car = getCar(car,index);
        }
        return car;
}
String getCar(String car,int index){
        int idx = car.lastIndexOf('.');
```

```
            idx--;
            while(Character.isDigit(car.charAt(idx)))idx--;
            return (car.substring(0,idx+1)+index+".gif");
        }
        int getCarIndex(String car){
            int carIndex;
            int index = car.lastIndexOf('.');
            index-=2;
            if(Character.isDigit(car.charAt(index))){
                carIndex = Integer.parseInt(Character.toString(car.charAt(index)))*10;
                index++;
                carIndex += Integer.parseInt(Character.toString(car.charAt(index)));
                return carIndex;
            }
            else{
                index++;
                if(Character.isDigit(car.charAt(index))){
                    carIndex = Integer.parseInt(Character.toString(car.charAt(index)));
                    return carIndex;
                }
            }
            return -1;
        }
}
```