**ASSESSMENT TASK 2 COLLEGE OF SINGAPORE**

 **INDIVIDUAL TASK COVER SHEET**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Student*  *Please sign, date and attach cover sheet to front of assessment task for all hard copy submissions* | | | | | | | | | |
| **SUBJECT CODE** | CP2406 | | | | | | | | |
| **STUDENT FAMILY NAME** | **Student Given Name** | **JCU Student Number** | | | | | | | |
| Nguyen | Duc Tuan | 1 | 3 | 5 | 4 | 7 | 3 | 7 | 0 |
| **ASSESSMENT TITLE** | Assignment 2 | | | | | | | | |
| **DUE DATE** | 21 September 2018 | | | | | | | | |
| **LECTURER NAME** | JAIKISHIN LAKHYANI | | | | | | | | |
| **TUTOR NAME** | JAIKISHIN LAKHYANI | | | | | | | | |
| **Student Declaration**     1. This assignment is my original work and no part has been copied/ reproduced from any other person’s work or from any other source, except where acknowledgement has been made (see *Learning, Teaching and Assessment Policy*   *5.1*).   1. This work has not been submitted for any other course/subject (see *Learning, Teaching and Assessment Policy 5.9*). 2. This assignment has not been written for me. 3. I hold a copy of this assignment and can produce a copy if requested. 4. This work may be used for the purposes of moderation and identifying plagiarism. 5. I give permission for a copy of this marked assignment to be retained by the College for benchmarking and course review and accreditation purposes.       [Learning, Teaching and Assessment Policy](http://www.jcu.edu.au/policy/allitoz/JCU_076643.html) 5.1. A student who submits work containing plagiarised material for assessment will be subject to the provisions of th[e Student Academic Misconduct Requirements.](http://www.jcu.edu.au/policy/allitoz/JCUDEV_005375.html)    **Note definition of plagiarism and self plagiarism in Learning, Teaching and Assessment Policy:**  **Plagiarism:** reproduction without acknowledgement of another person’s words, work or expressed thoughts from any source. The definition of words, works and thoughts includes such representations as diagrams, drawings, sketches, pictures, objects, text, lecture hand-outs, artistic works and other such expressions of ideas, but hereafter the term ‘work’ is used to embrace all of these. Plagiarism comprises not only direct copying of aspects of another person’s work but also the reproduction, even if slightly rewritten or adapted, of someone else’s ideas. In both cases, someone else’s work is presented as the student’s own. Under the Australian *Copyright Act 1968* a copyright owner can take legal action in the courts against a party who has infringed their copyright.  **Self Plagiarism:** theuse of one’s own previously assessed material being resubmitted without acknowledgement or citing of the original.    **Student Signature**  **NGUYEN DUC TUAN**  **…………………………………………………… Submission date 21/09/2018** | | | | | | | | | |

Link to GitHub:

<https://github.com/CP2406/cp2406-project-ductuan1999>

User stories

As a user, I can input my name to my light cycle to play the game

Estimate: 5s to register

Priority: high

As a user, I can restart the game when the game ends

Estimate: 1s

Priority: Medium

As a user, I can play with my friends on the same computer

Estimate: Immediately when the game starts

Priority: High

As a user, I can choose the color for my light cycle

Estimate: 5s

Priority: Low

As a user, I can be able to identify when the game is over and the name of the winner of the game

Estimate: 5 seconds

Priority: High

As a user, I can look at the score to see how much I have scored while moving the light cycle

Estimate: Immediately when the game starts

Priority: High

# Spike Stories:

As a user, I can look at the leaderboard and the high scores of the game

Estimate: Right when the game starts

Priority: High

\

As a user, I can save my scores to a file

Estimate: Right when the game ends

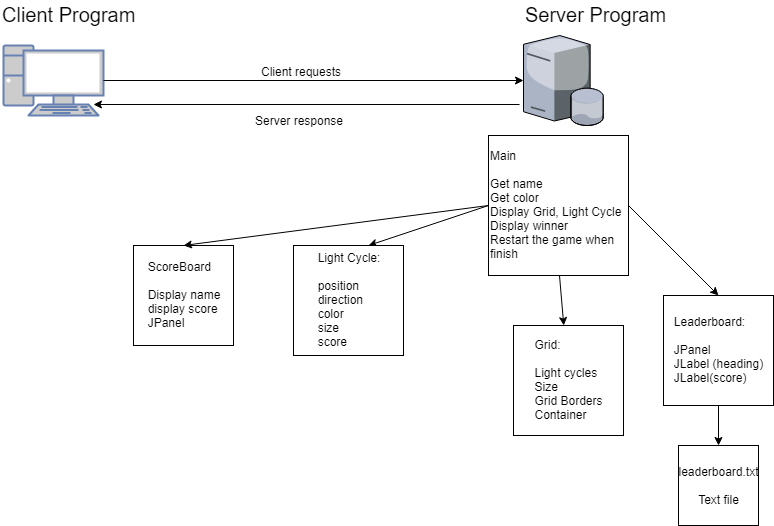
Priority: High

As an admin, I can display the high scores on the game screen

Estimate: Right when the game starts

Priority: High

UML Class Diagram:

s

Program code:

Main.java

*//import modules***import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.KeyAdapter;  
**import** java.awt.event.KeyEvent;  
  
  
*// this is what an anonymous inner class does... kind of :)***class** fAnon **implements** ActionListener {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 }  
}  
  
*//Main class***public class** Main {  
 **public static void** main(String[] args) {  
  
 *//getting input for name and color of two players* String a = JOptionPane.*showInputDialog*(**"Enter 1st player name: "**);  
  
  
 Color c = JColorChooser.*showDialog*(**null**, **"Choose a color for your Light Cycle"**, Color.***RED***);  
  
 String b = JOptionPane.*showInputDialog*(**"Enter 2nd player name: "**);  
  
 Color d = JColorChooser.*showDialog*(**null**, **"Choose a color for your Light Cycle"**, Color.***RED***);  
  
  
 *//Create frame of game, game title* JFrame frame = **new** JFrame(**"Light Cycle"**);  
 Grid grid = **new** Grid();  
  
 *//Setting up two light cycles* LightCycle car1 = **new** LightCycle(5, c);  
 LightCycle car2 = **new** LightCycle(5, d);  
  
  
 *//display names, scores and leaderboard on screen* JLabel name = **new** JLabel(**"Name: "** + a + **" | "** );  
 JLabel name2= **new** JLabel(**"Name: "** + b + **" | "** );  
  
 **final** JLabel scoreDisplay = **new** JLabel(**"Score: "** );  
 **final** JLabel scoreDisplay2 = **new** JLabel(**"Score: "**);  
 LeaderBoard leaderBoard = **new** LeaderBoard();  
  
  
 name.setForeground(Color.***BLACK***);  
 name2.setForeground(Color.***BLACK***);  
  
  
 name.setBackground(Color.***WHITE***);  
 name2.setBackground(Color.***WHITE***);  
 frame.setSize(600,800);  
  
 ScoreBoard scoreBoard = **new** ScoreBoard();  
 scoreBoard.add(name, BorderLayout.***NORTH***);  
  
  
 scoreBoard.add(scoreDisplay, BorderLayout.***SOUTH***);  
  
 scoreBoard.add(name2,BorderLayout.***NORTH***);  
 scoreBoard.add(scoreDisplay2,BorderLayout.***SOUTH***);  
  
 */\*\*scoreBoard.add(button, BorderLayout.EAST);\*\*/* frame.add(scoreBoard, BorderLayout.***SOUTH***);  
 frame.add(leaderBoard, BorderLayout.***EAST***);  
  
  
 *//get random position of light cyclels in grid* car1.getRandomStart();  
 car2.getRandomStart();  
  
  
  
 *// frame-by-frame animation loop* Timer timer = **new** Timer(50, **new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 car1.move(grid.getSize(),car2.getPoints()); *//move car1* car2.move(grid.getSize(), car1.getPoints()); *//move car2  
  
 //display scores* scoreDisplay.setText(**"Score: "** + String.*valueOf*(car1.getScore()) + **" ------ "**);  
 scoreDisplay2.setText(**"Score: "** + String.*valueOf*(car2.getScore()));  
  
  
 *//when car 1 dies* **if** (!car1.getAlive()) {  
  
 car1.leaderBoard(); *//save scores to leaderboard* car2.leaderBoard();  
  
 *//display winner* **int** selectedOption = JOptionPane.*showConfirmDialog*(frame, b + **" won!"** +  
 **"\n Do you want to play again?"**);  
  
 *//restart* **if** (selectedOption == JOptionPane.***YES\_OPTION***) {  
 car1.restart();  
 car2.restart();  
  
 *//close System* }  
 **if** (selectedOption == JOptionPane.***NO\_OPTION*** || selectedOption == JOptionPane.***CANCEL\_OPTION***){  
 System.*exit*(0);  
 }  
 }  
  
 *//if car2 dies* **if** (!car2.getAlive()) {  
  
 car1.leaderBoard(); *//save scores to leaderboard* car2.leaderBoard();  
  
 *//display winner* **int** selectedOption = JOptionPane.*showConfirmDialog*(frame, a + **" won!"** +  
 **"\n Do you want to play again?"**);  
  
 *//restart* **if** (selectedOption == JOptionPane.***YES\_OPTION***) {  
 car1.restart();  
 car2.restart();  
  
 }  
  
 *//close system* **if** (selectedOption == JOptionPane.***NO\_OPTION*** || selectedOption == JOptionPane.***CANCEL\_OPTION***){  
 System.*exit*(0);  
 }  
 }  
  
 *//repaint for each frame per second* grid.repaint();  
  
 }  
 });  
  
 *//start timer* timer.start();  
  
  
  
  
 *//Setting up key interaction with light cycle for car1* grid.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyPressed(KeyEvent e){  
 e.getKeyCode();  
 **if** (e.getKeyCode() == KeyEvent.***VK\_W***){  
 car1.up();  
 }  
 **if** (e.getKeyCode() == KeyEvent.***VK\_S***){  
 car1.down();  
 }  
 **if** (e.getKeyCode() == KeyEvent.***VK\_A***){  
 car1.left();  
 }  
 **if** (e.getKeyCode() == KeyEvent.***VK\_D***){  
 car1.right();  
 }  
  
 }  
 });  
  
 *//Setting up key interaction with light cycle for car2* grid.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyPressed(KeyEvent e){  
 e.getKeyCode();  
 **if** (e.getKeyCode() == KeyEvent.***VK\_UP***){  
 car2.up();  
 }  
 **if** (e.getKeyCode() == KeyEvent.***VK\_DOWN***){  
 car2.down();  
 }  
 **if** (e.getKeyCode() == KeyEvent.***VK\_LEFT***){  
 car2.left();  
 }  
 **if** (e.getKeyCode() == KeyEvent.***VK\_RIGHT***){  
 car2.right();  
 }  
  
 }  
 });  
  
  
  
  
 *//add grid to frame* frame.add(grid, BorderLayout.***CENTER***);  
  
 *//add light cycles to grid* grid.addLightCycle(car1);  
 grid.addLightCycle(car2);  
  
 *//set frame's properties* frame.pack();  
 frame.setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
 frame.setVisible(**true**);  
 frame.setLocationRelativeTo(**null**);  
 frame.setResizable(**false**);  
 }  
}

Grid.java

*//import modules***import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.util.ArrayList;  
**import** java.util.Random;  
  
  
*//Create jpanel of grid***public class** Grid **extends** JPanel {  
  
 *//list of light cycles in the grid* **private int i** = 0;  
 **private** ArrayList<LightCycle> **lightCycles** = **new** ArrayList<LightCycle>();  
  
 **private static final** Font ***font*** = **new** Font(**"Arial"**,Font.***BOLD***, 24);  
  
  
 *//constructor* Grid() {  
 setFocusable(**true**); *// is false by default...* setPreferredSize(**new** Dimension(600,600));  
 setBackground(**new** Color(0,0,0));  
 setBorder(BorderFactory.*createLineBorder*(Color.***WHITE***));  
 setFocusable(**true**);  
  
 }  
  
 *//add light cycles to grid* **void** addLightCycle(LightCycle lightCycle) {  
 **lightCycles**.add(lightCycle);  
 }  
  
  
  
  
  
 *//paint graphics for grid and light cycle* @Override  
 **protected void** paintComponent(Graphics graphics) {  
 **super**.paintComponent(graphics);  
  
 **for**(LightCycle lightCycle : **lightCycles**) {  
 lightCycle.draw(graphics);  
 }  
  
 }  
}

LightCycle.java

*//import modules***import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.io.BufferedOutputStream;  
**import** java.io.IOException;  
**import** java.io.OutputStream;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.ArrayList;  
**import** java.util.Formatter;  
**import** java.util.Random;  
**import static** java.nio.file.StandardOpenOption.\*;  
  
  
**class** LightCycle {  
  
 *//dim private variables and constants* **private int x**,**y**; *//position of lightcycle* **private int xDir**, **yDir**; *//direction of lightcycle* **private final int VELOCITY** = 5; *//velocity of lightcycle* **private boolean alive** = **true**; *//alive light cycle (uncrashed)* **private** ArrayList<Point> **points** = **new** ArrayList<Point>(); *//List of points travelled by lightcycle* **private int size**; *//size of light cycle* **private** Color **color**; *//color of light cycle* **private static final** Random ***random*** = **new** Random(); *//generate random  
  
 //constructor* LightCycle(**int** size, Color color) {  
 **x** = 0;  
 **y** = 0;  
 **xDir** = **VELOCITY**;  
 **yDir** = 0;  
 **this**.**size** = size;  
 **this**.**color** = color;  
 }  
  
 *//directions modification of light cycle (UP, DOWN, LEFT, RIGHT)* **public void** up() {  
 **xDir** = 0;  
 **yDir** = -**VELOCITY**;  
 }  
  
 **public void** down() {  
 **xDir** = 0;  
 **yDir** = **VELOCITY**;  
 }  
  
 **public void** left(){  
 **xDir** = -**VELOCITY**;  
 **yDir** = 0;  
  
 }  
  
 **public void** right(){  
 **xDir** = **VELOCITY**;  
 **yDir** = 0;  
  
 }  
  
 *//get points travelled by light cycle* **public** ArrayList<Point> getPoints(){  
 **return points**;  
 }  
  
 *//move light cycle* **void** move(Dimension size, ArrayList<Point> opponentPoints) {  
 *// update position based on current direction* **int** ox = **x**; *//distance from origin* **int** oy = **y**; *//distance from origin  
  
 //distance + direction travelled* **x** = **x** + **xDir**;  
 **y** = **y** + **yDir**;  
  
 *//stop when touch the line* **for** (**int** i = 0; i < **points**.size(); i++) {  
 Point point = **points**.get(i);  
 **if** (**x** == point.**x** && **y** == point.**y**) {  
 **xDir** = 0;  
 **yDir** = 0;  
 **x**= ox;  
 **y** = oy;  
  
 **break**;  
 }  
 }  
  
  
 *//stop when touch opponent's line* **for** (**int** i = 0; i < opponentPoints.size(); i++) {  
 Point point = opponentPoints.get(i);  
 **if** (**x** == point.**x** && **y** == point.**y**) {  
 **xDir** = 0;  
 **yDir** = 0;  
 **x**= ox;  
 **y** = oy;  
  
 **break**;  
 }  
 }  
  
 *//stop when touch the wall* **if**(**x** < 0 || **x** > 600) {  
 **xDir** = 0;  
 }  
 **if** (**y** < 0 || **y** > 600){  
 **yDir** = 0;  
 }  
  
 *//crash* crash();  
  
 *//add points to list* **if** (**alive**){  
 Point point = **new** Point(ox, oy);  
 **points**.add(point);  
 }  
 }  
  
 *//when crash* **void** crash(){  
 **if** (**xDir** == 0 && **yDir** == 0){  
 **alive** = **false**;  
 }  
 }  
  
 *//random initial position* **void** getRandomStart(){  
 **x** = **size**\****random***.nextInt(50);  
 **y** = **size**\****random***.nextInt(50);  
 }  
  
  
*//get score* **int** getScore(){  
 **int** score = **points**.size();  
 **return** score;  
  
 }  
  
 *//write file to leaderboard* **void** leaderBoard(){  
 **int** sco = getScore();  
 Path file = Paths.*get*(**"leaderboard.txt"**);  
 String s = String.*valueOf*(sco) + **"\n"** ;  
 **byte** [] data = s.getBytes();  
 OutputStream output = **null**;  
 **try** {  
 output = **new** BufferedOutputStream(Files.*newOutputStream*(file,***APPEND***));  
 output.write(data );  
  
 output.flush();  
 output.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 **boolean** getAlive(){  
 **return alive**;  
 }  
  
 *//draw light cycle, draw line* **void** draw(Graphics graphics) {  
  
 graphics.setColor(**color**);  
 graphics.fillRect(**x**,**y**,**size**,**size**);  
  
 **for**(Point point : **points**) {  
 graphics.fillRect(point.**x**,point.**y**,**size**,**size**);  
 }  
  
 }  
*//restart* **void** restart(){  
 getRandomStart();  
 **xDir** = **VELOCITY**;  
 **yDir** = 0;  
 **this**.**size** = **size**;  
 **this**.**color** = **color**;  
 **alive** = **true**;  
 **points**.clear();  
 }  
}

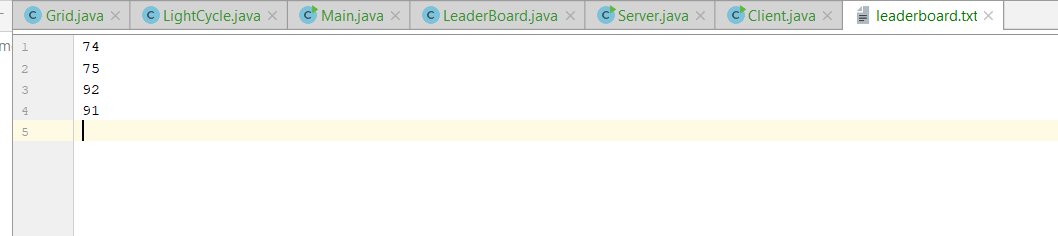
ScoreBoard.java

*//import modules***import** javax.swing.\*;  
**import** java.awt.\*;  
  
*//draw scoreboard on screen***public class** ScoreBoard **extends** JPanel  
{  
 *//constructor* **public** ScoreBoard(){  
 setPreferredSize(**new** Dimension(600,50  
 ));  
 setBackground(Color.***WHITE***);  
  
  
  
  
 }  
  
}

LeaderBoard.java

*//import modules***import** javax.swing.\*;  
**import** javax.swing.border.Border;  
**import** java.awt.\*;  
**import** java.io.\*;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.ArrayList;  
**import static** java.nio.file.StandardOpenOption.\*;  
  
*//Create a panel for the leaderboard***public class** LeaderBoard **extends** JPanel  
{  
 JLabel **heading** = **new** JLabel(**"Leaderboard: \n"**); *//The leaderboard header* JLabel **savedscore** = **new** JLabel(**""**); *//The score saved in the leaderboard* LightCycle **lightCycle**; *//Calling class LightCycle  
  
 //LeaderBoard constructor (designing the panel and labels)* **public** LeaderBoard(){  
 setPreferredSize(**new** Dimension(100,600));  
 setBackground(Color.***BLACK***);  
 **heading**.setForeground(Color.***WHITE***);  
 **savedscore**.setForeground(Color.***WHITE***);  
  
 add(**heading**, BorderLayout.***NORTH***);  
  
 *//Opening the leaderboard file* Path path = Paths.*get*(**"leaderboard.txt"**);  
 InputStream input = **null**;  
 **try**{  
 input = Files.*newInputStream*(path);  
 BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(input));  
 String s = **null**;  
 s = reader.readLine();  
 **savedscore**.setText(s);  
 add(**savedscore**, BorderLayout.***CENTER***);  
  
 input.close();  
 }  
 **catch** (IOException e) {  
 System.***out***.println(e);  
 }  
 }  
}

leaderboard.txt



Server

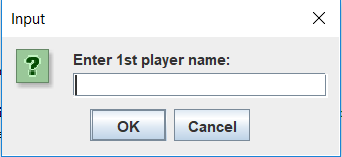
**import** javax.swing.\*;  
**import** java.io.DataInputStream;  
**import** java.io.DataOutputStream;  
**import** java.io.IOException;  
**import** java.io.PrintStream;  
**import** java.net.InetAddress;  
**import** java.net.MulticastSocket;  
**import** java.net.ServerSocket;  
**import** java.net.Socket;  
**import** java.util.Scanner;  
  
**public class** Server  
{  
 **static** ServerSocket *ss*;  
 **static** Socket *s*;  
 **static** DataOutputStream *dout*;  
 **static** DataInputStream *din*;  
 **public** Server() {  
  
 }  
  
 **public static void** main(String[] args) {  
  
 String a = JOptionPane.*showInputDialog*(**"Enter number of clients: "**);  
 **int** num = Integer.*parseInt*(a);  
 **try** {  
 *ss* = **new** ServerSocket(1201);  
 *s* = *ss*.accept();  
 *din* = **new** DataInputStream(*s*.getInputStream());  
 *dout* = **new** DataOutputStream(*s*.getOutputStream());  
  
 **while** (a != **null**){  
 *dout*.writeInt(num);  
 }  
  
  
  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
  
  
  
  
 }  
}

Client

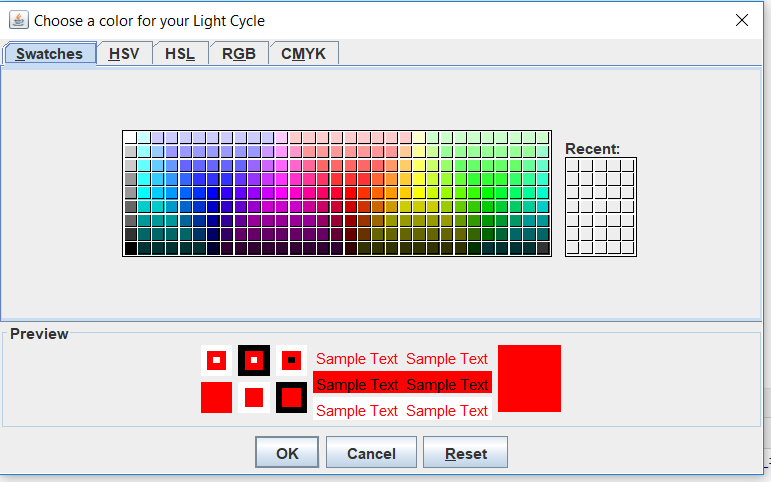
**import** java.util.\*;  
**import** javax.swing.\*;  
**import** java.io.DataInputStream;  
**import** java.io.DataOutputStream;  
**import** java.io.IOException;  
**import** java.net.\*;  
  
**public class** Client  
{  
 **static** Socket *s*;  
 **static** DataInputStream *din*;  
 **static** DataOutputStream *dout*;  
 **public** Client() {  
  
 }  
  
  
 **public static void** main(String[] args) **throws** IOException, UnknownHostException  
 {  
 **int** num;  
 *s* = **new** Socket(**"127.0.0.1"**, 1201);  
 *din* = **new** DataInputStream(*s*.getInputStream());  
 *dout* = **new** DataOutputStream(*s*.getOutputStream());  
  
 Scanner sc = **new** Scanner(*s*.getInputStream());  
 num = sc.nextInt();  
  
  
 }  
  
}

Screen shots of the GUI

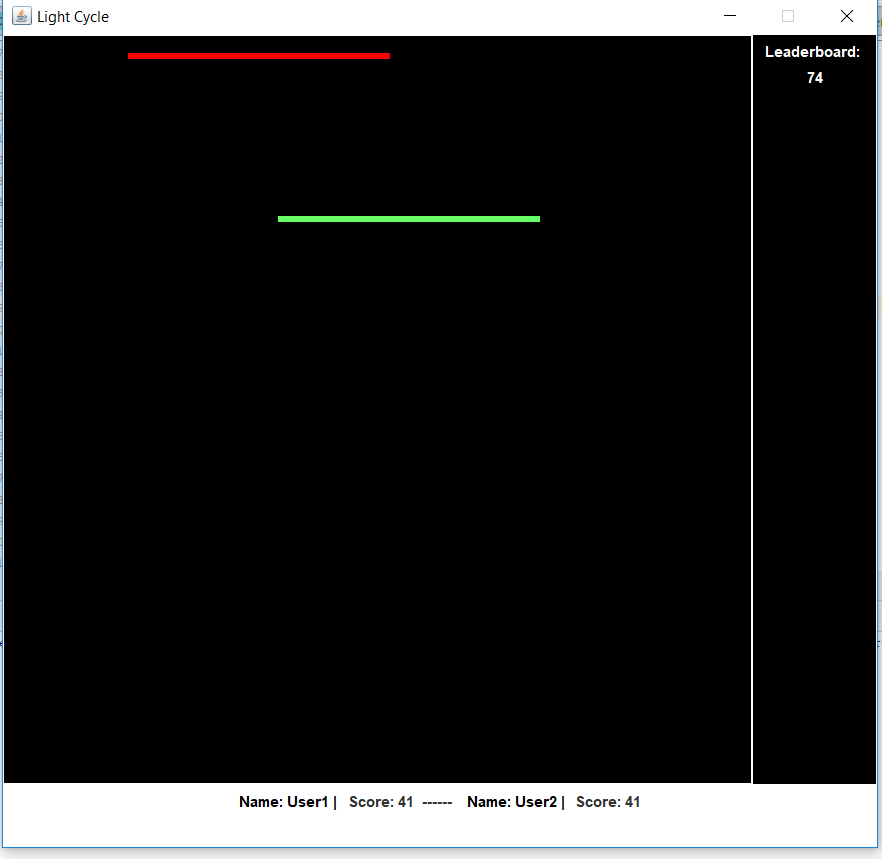
Get name:



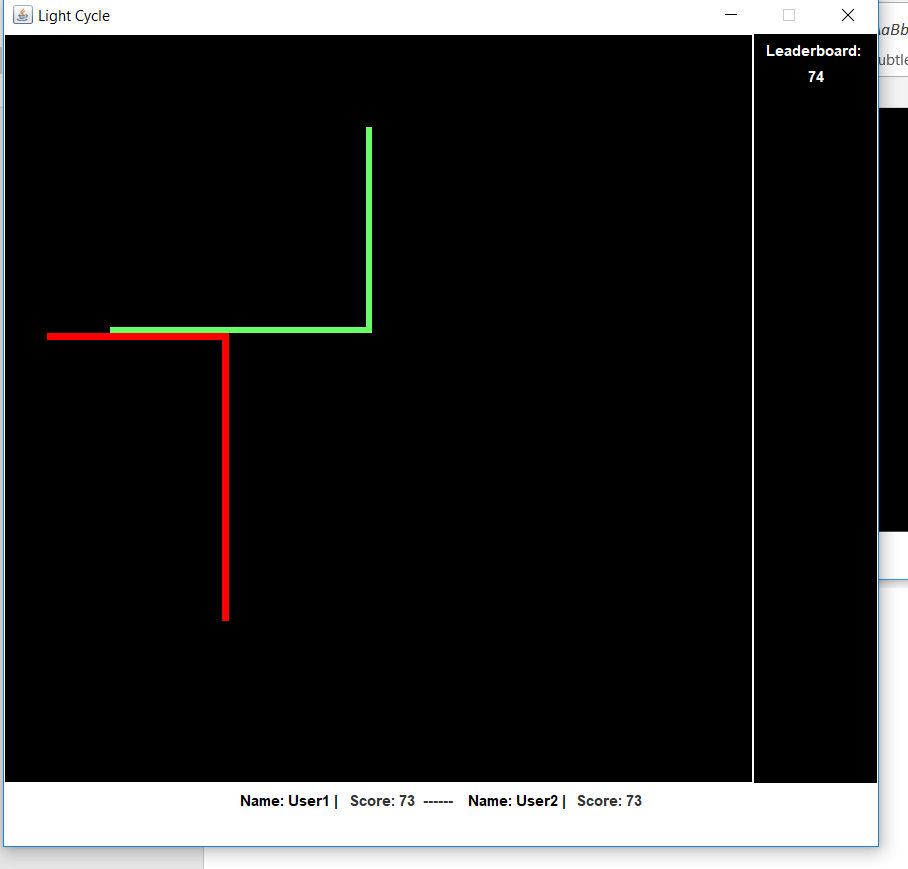
Get Color:



Game Start:



Change Direction:



Crash:

