TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
KHOA ĐIỆN TỬ VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN LẬP TRÌNH JAVA

Đề tài : Thiết kế game đua xe

Giảng viên hướng dẫn : **TS. Phạm Huy Hoàng**

Sinh viên thực hiện : **Phạm Duy Kiên 20111726**

Trần Mạnh Toàn 20112344

Hà Nội, 5-2015

Contents

I. Mô tả bài toán và phương hướng giải quyết bài toán.

Lập trình phần mềm trò chơi đua ô tô với các yêu cầu:

1: Sử dụng kỹ thuật lập trình hướng đối tượng

2: Có các xe đối thủ cạnh tranh trên đường đua

3: Các chướng ngại vật

4: Khi người chơi đạt tới điểm số quy định sẽ tự động nâng mức khó của trò chơi

5: Cho phép người chơi ghi lại số điểm đạt được.

1. Mô tả bài toán.

Game đua xe ôtô trên máy tính cũng cần phải có:

- Khởi tạo trò chơi: là một Frame cho phép người chơi biết một số thông tin cần thiết, các thức điều khiển trò chơi.

- Người điều khiển xe đua: người chơi sử dụng máy tính để điều khiển xe.

- Đường đua: nơi các xe tham gia đua. Trên đường đua gồm có xe của người chơi, xe của các đối thủ, các chướng ngại vật mà những người tham gia đua xe cần phải vượt qua.

- Vỉa hè: là những rào cản bên ngoài, nó có nhiệm vụ giới hạn các xe chỉ được phép đi trên vị trí lòng đường đã quy định, không được phép vượt ra ngoài phạm vi này

- Chướng ngại vật: một điều không thể thiếu khi nói đến trò chơi đua xe ôtô. Ngoài xe của người chơi còn có xe của những đối thủ cùng tham gia trò chơi, các chướng ngại vật trên đường, nếu những xe tham gia giao thông gặp nhau sẽ gây ra va chạm và xe của người chơi không thể đi thẳng tiếp.

- Điểm số và tên người chơi: tệp tin Score.txt và NamePlayer.txt trên máy tính cho phép người chơi lưu lại tên của mình khi người chơi đạt tới một

điểm số nhất định. Có thể có nhiều người chơi đạt được yêu cầu nên cần phải có một danh sách lưu lại điểm và người chơi. Mỗi người chơi cùng với điểm số tương ứng sẽ được lưu trên một dòng văn bản duy nhất.

- Mức độ khó của trò chơi: với bài toán này, do có yêu cầu tự động nâng mức độ khó của trò chơi nên bối cảnh của đường đua xe thay đổi tùy theo các mức độ khó của trò chơi.

2. Phương hướng giải quyết bài toán

Trên cơ sở đã phân tích bài toán, dựa trên kiểu lập trình hướng đối tượng, chúng em bắt tay vào xây dựng những lớp, những thuộc tính, những phương thức tương ứng với khi phân tích bài toán

- Khởi tạo trò chơi: để cho phép người chơi khởi tạo trò chơi mới, cần xây dựng một Frame riêng, đáp ứng một cách tốt nhất những trợ giúp cho người chơi mới lần đầu sử dụng.

- Người chơi: cái mà người chơi nhìn thấy chỉ là giao diện bề ngoài của chương trình, muốn người chơi cảm thấy hứng thú khi chơi ngoài tính năng đặc biệt, trước tiên ta cần phải thể hiện một giao diện màn hình ưa nhìn, không quá cầu kì, quá phức tạp gây rối mắt nhưng cũng không được quá sơ sai, đơn điệu dễ gây nhàn chán cho người chơi.

- Đường đua và vỉa hè: để tạo ra đường đua và vỉa hè trong chương trình này đã load file có tên là map1.txt, map2.txt, map3.txt…. từ thư mục map, tương ứng với mỗi mức độ khó dễ của trò chơi là một tệp tin mapx.txt khác nhau. Khi đọc nhưng tệp tin đó, dựa trên những kí tự có trong tệp tin, chương trình sẽ nhận đinh kí tự nào tương ứng với làn đường đua, kí tự nào tương ứng với vỉa hè, kí tự nào tương ứng với vị trí khởi tạo trò chơi, kí tự nào tương ứng với các đối thủ đua cùng người chơi, kí tự nào ở vị trí nào sẽ là các chướng ngại vật gây cản trở cho người chơi, kí tự nào thể hiện vị trí kết thúc một mức của trò chơi để bước sang mức chơi mới.

- Xe ôtô: trong chương trình này, phương tiên tham gia giao thông chỉ có duy nhất là xe ôtô. Nhưng ôtô này là những file ảnh được load từ thư muc images. Ảnh khác nhau sẽ tương ứng cho nhưng xe khác nhau, riêng xe của người chơi là duy nhất, còn nhưng xe đối thủ là những xe riêng, nhưng phương tiện hay là những chướng ngại vật cũng là nhưng file ảnh khác để phân biệt chúng với những xe khác cùng tham gia.

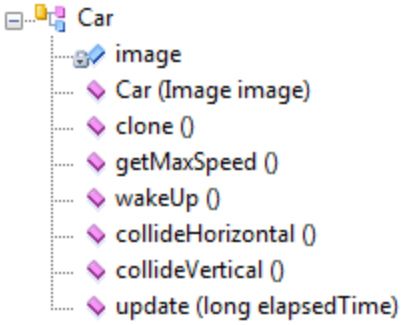
- Lưu điểm số và tên của người chơi: trong chương trình này sử dụng tệp tin có tên là Score.txt để lưu tên của những người chơi và file NamePlayer.txt đề lưu tên người chơi khi khởi tạo .

II. Các lớp của chương trình

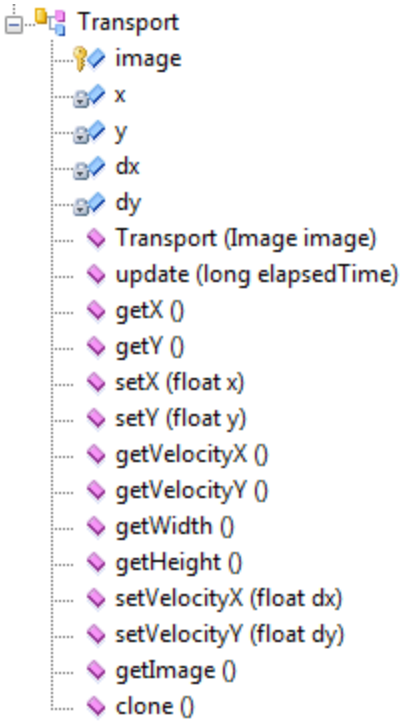
Để xây dựng lên được game đua xe ôtô chúng em đã xây dựng 17 lớp.

1. Lớp Car

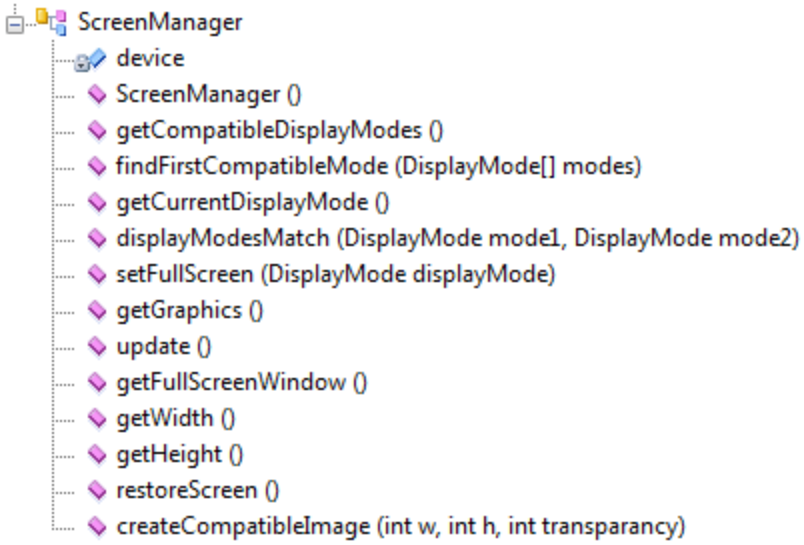
Lớp Car kế thừa từ lớp Transport



2. Lớp Transport

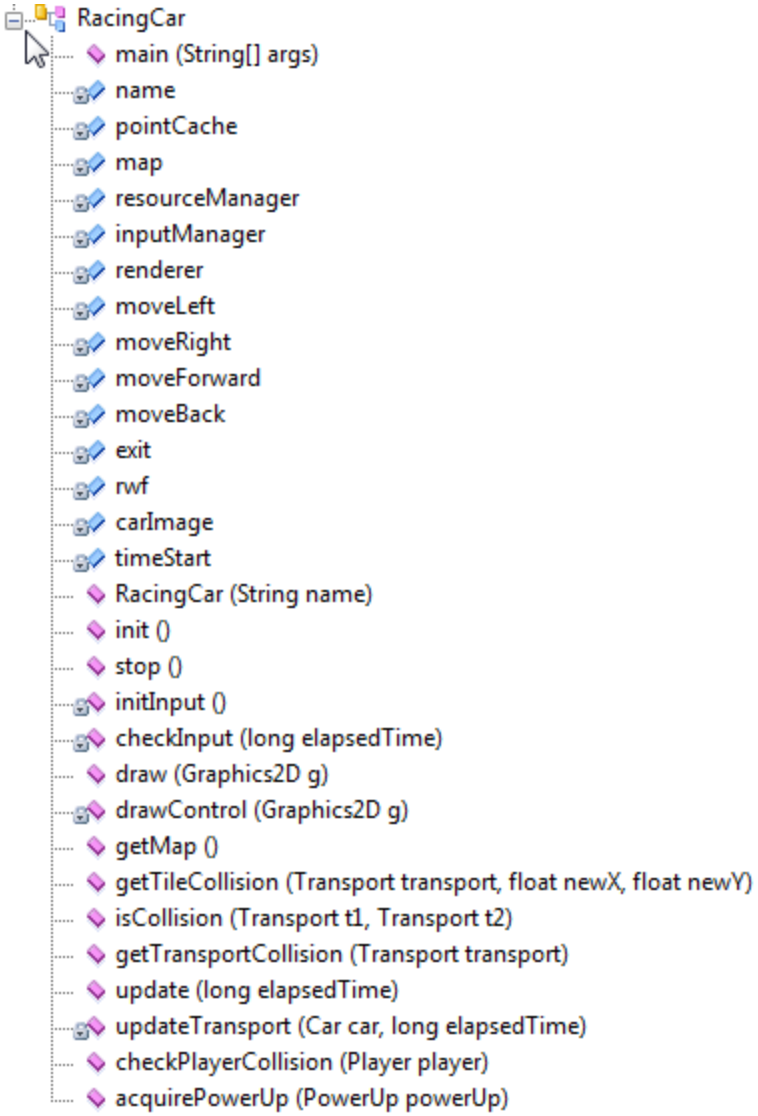


3. Lớp ScreenManager

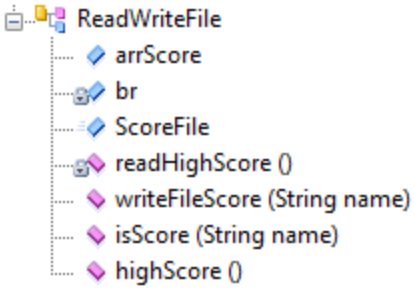


4. Lớp RacingCar

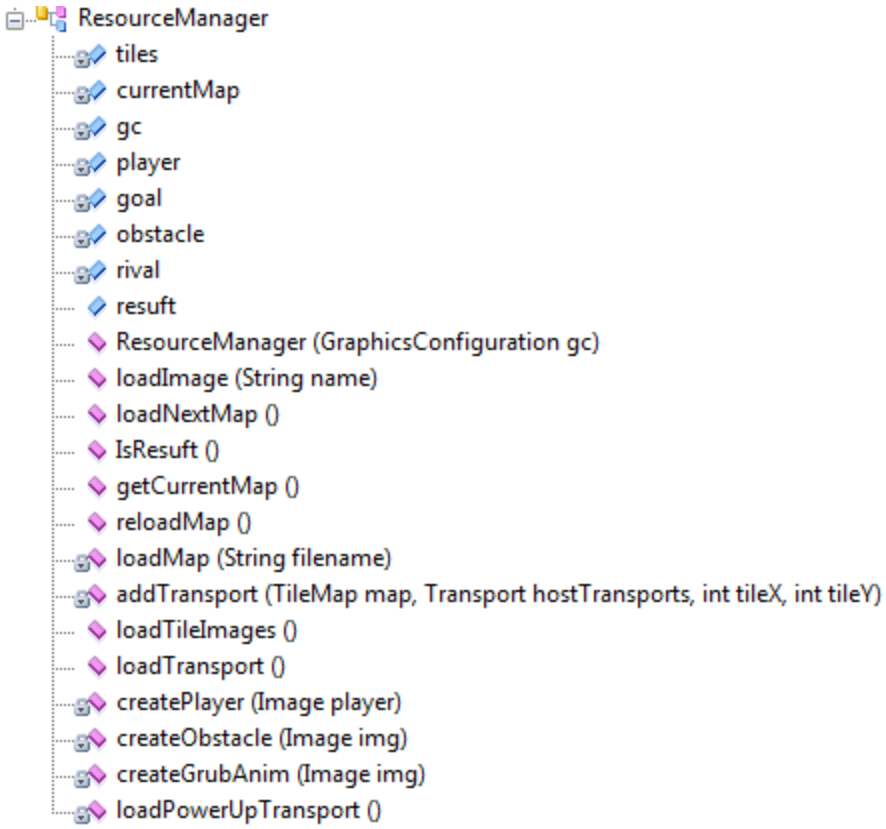
Lớp RacingCar kế thừa từ lớp GameCore



5. Lớp ReadWriteFile

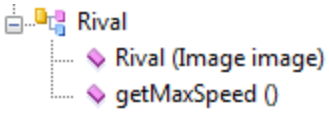


6. Lớp ResourceManager



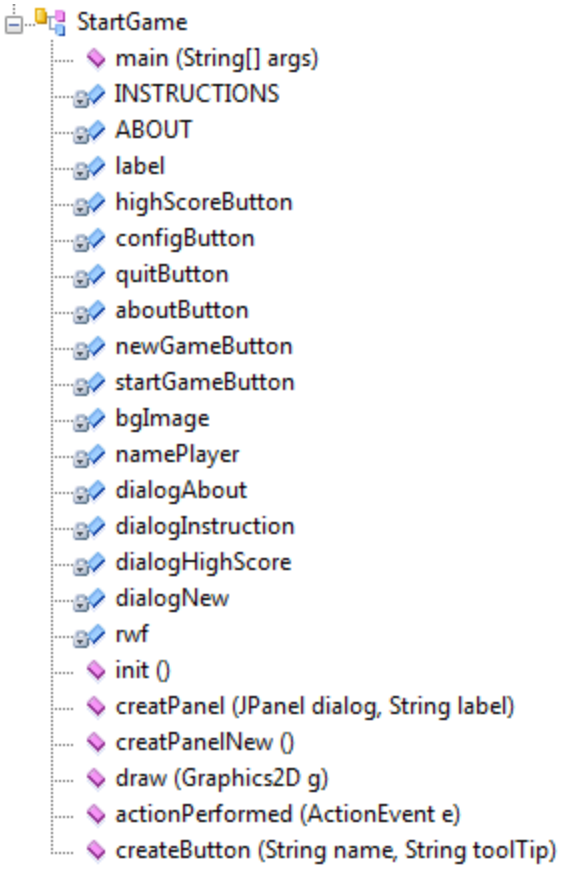
7. Lớp Rival

Lớp Rival kế thừa từ lớp Car

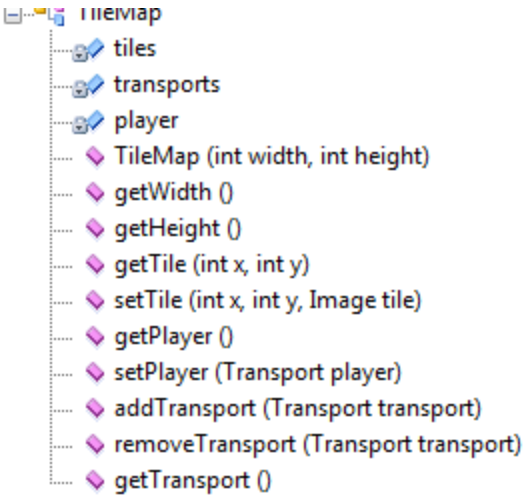


8. Lớp StartGame

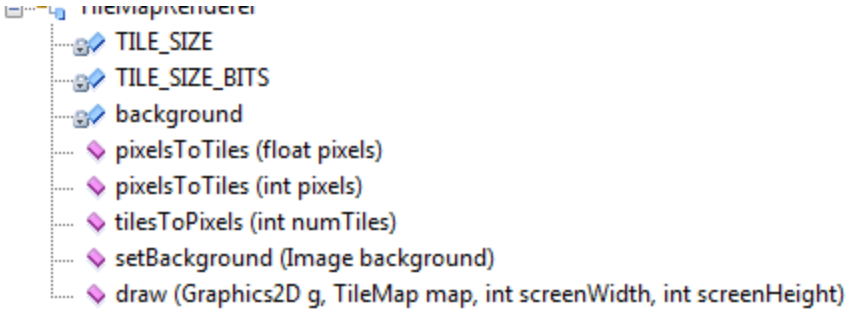
Lớp StartGame kế thừa từ lớp GameCore và lắng nghe sự kiện ActionListener



9. Lớp TileMap

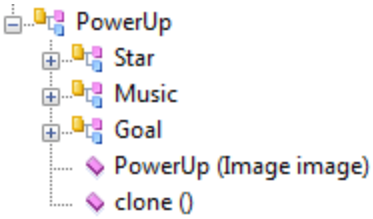


10. Lớp TileMapRenderer



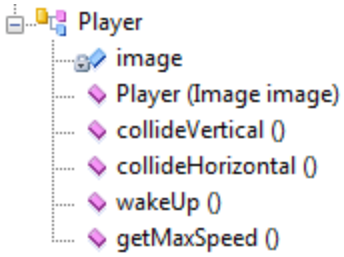
11. Lớp PowerUp

Lớp PowerUp kế thừa từ lớp Transport



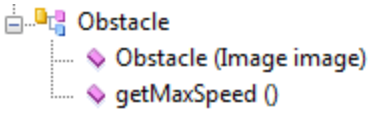
12. Lớp Player

Lớp Player kế thừa từ lớp Car



13. Lớp Obstacle

Lớp Obstacle kế thừa từ lớp Car

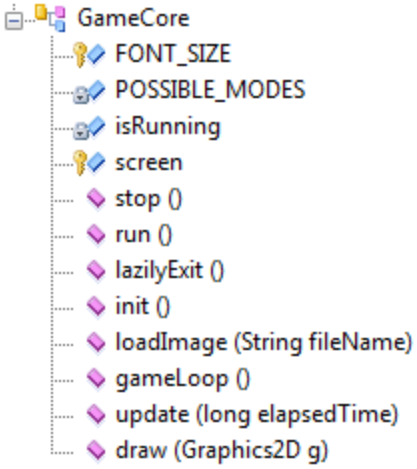


14. Lớp NullRepaintManager

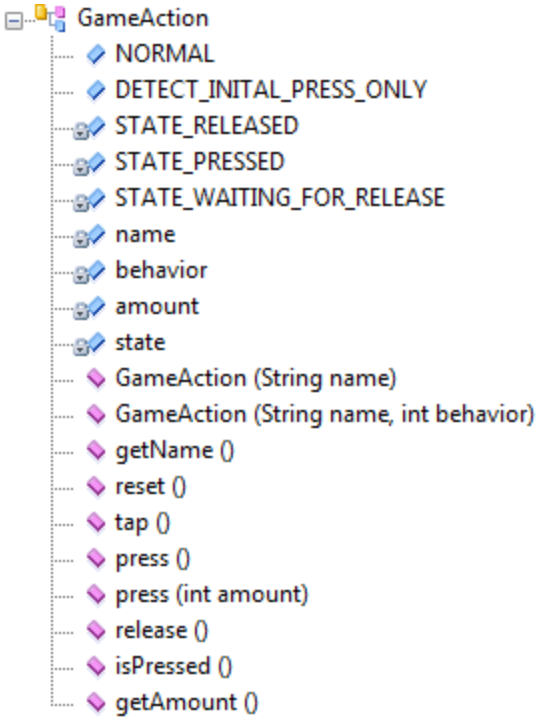
Lớp NullRepaintManager kế thừa từ lớp RepaintManager

15. Lớp GameCore

Lớp GameCore kế thừa từ lớp JFrame



16. Lớp GameAction



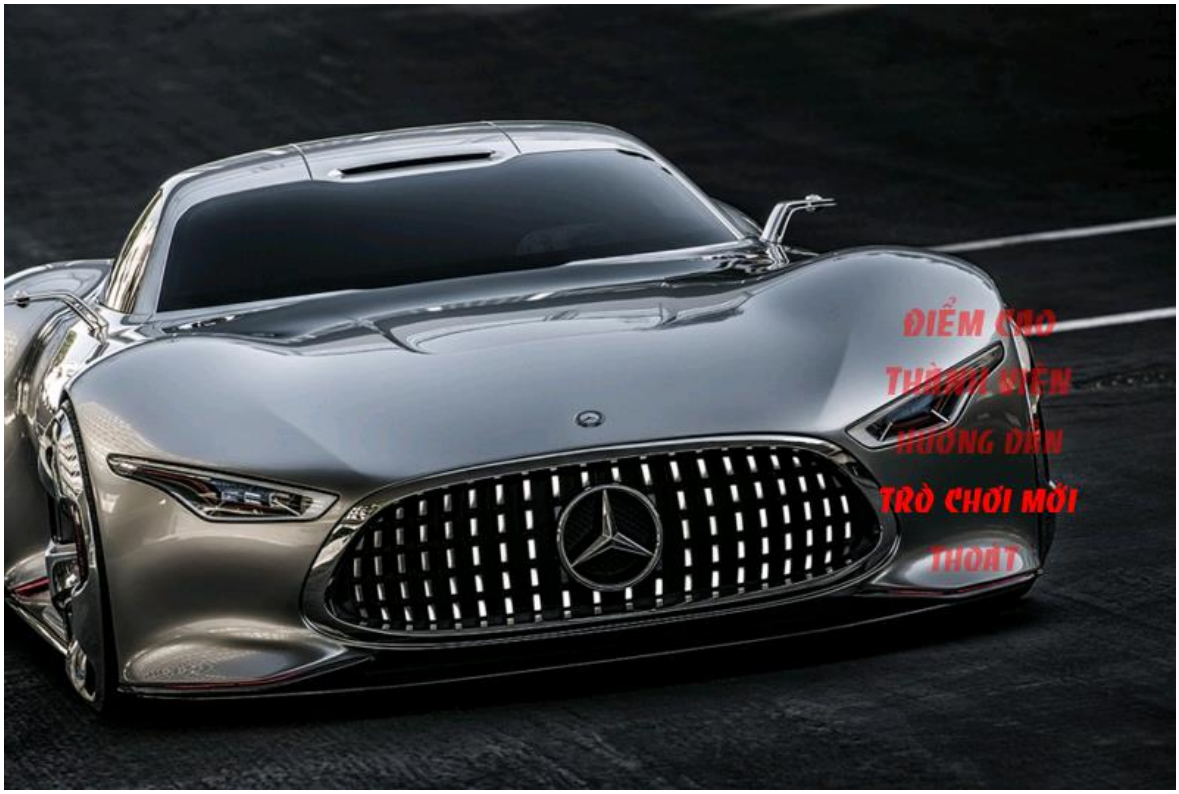
17. Lớp InputManager

Lớp InputManager lắng nghe các sự kiên KeyListener, MouseListener, MouseMotionListener và MouseWheelListener

III. Xây dựng chương trình

1. Khởi tạo trò chơi.

Lớp StartGame là lớp dùng để khởi tạo chò trơi. Lớp này có giao diện đồ họa dễ dàng sử dụng. Ở giao diện đồ họa này người chơi biết biết được cách thức điều khiển xe, tên những người chơi đã chơi trước đó và hoàn thành trò choi, thông tin về tác giả. Người chơi có thể khởi tạo trò chơi mới với tên được đăng kí do người dùng tự đặt. Giao điện bằng tiếng Việt, chỉ vài lần click chuột là người chơi có thể tự mình tham gia trò chơi.



Để có được giao diện và thực hiện những hành động đó lớp StartGame đã kế thừa từ lớp GameCore và lắng nghe sự kiện ActionListener.

```
public ScreenManager() {
    GraphicsEnvironment environment =
      GraphicsEnvironment.getLocalGraphicsEnvironment();
    device = environment.getDefaultScreenDevice();
  }
```

Lớp ScreenManager sẽ lấy các thiết bị đồ họa trên một môi trường tương ứng với từng máy tính khác nhau. Với mỗi một máy tính khác nhau nó sẽ có

môi trường đồ học khác nhau. Trình biên dịch của Java sẽ ánh xạ sang từng máy để nó trả về môi trường cụ thể.

Phương thức *init()* trong lớp ScreenManager được gọi sẽ khởi tao ra một cửa sổ giao diện mà hình. Nó sẽ lấy độ phân giải cụ thể của từng máy sau đó sẽ trả về một giao diện đầy màn hình trên từng máy

```
public void init()
  {
        screen = new ScreenManager();
        DisplayMode displayMode =
        screen.findFirstCompatibleMode(POSSIBLE_MODES);
        screen.setFullScreen(displayMode);
        Window window = screen.getFullScreenWindow();
        window.setFont(new Font("Dialog", Font.PLAIN, FONT_SIZE));
        window.setBackground(Color.blue);
         window.setForeground(Color.white);
         isRunning = true;
  }
```

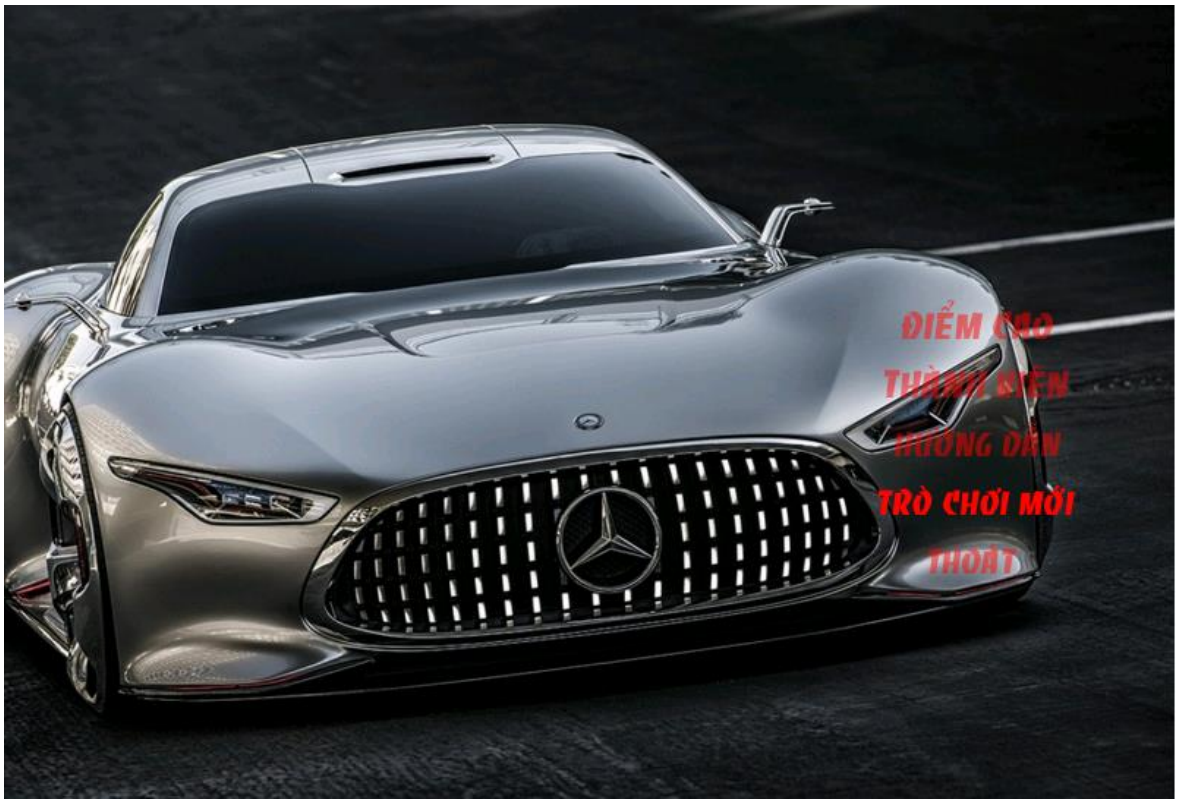
Sau khi lấy được quyền điều khiển các thiết bị đồ họa trên máy tính, phương thức *gameLoop()* sẽ được gọi. Phương thức này có nhiệm vụ load các file ảnh, cập nhật liên tục màn hình để cho người dùng có cảm giác như không có sự thay đổi gì khác biệt. Graphics2D sẽ trả về giá trị thông qua lớp ScreenManager, nó sẽ lấy đồ họa và hiển thị toàn màn hình. Phương thức *gameLoop()* sẽ thức hiện liên tục và được quản lí thông qua thuộc tính *isRunning*, nó chỉ dừng sau khi biến *isRunning* này được gán giá trị false.

```
public void gameLoop()
 {
  long startTime = System.currentTimeMillis();
long currTime = startTime;

 while (isRunning) {
 long elapsedTime = System.currentTimeMillis() - currTime;
```

```
            currTime += elapsedTime;
            update(elapsedTime);
            Graphics2D g = screen.getGraphics();
            draw(g);
            g.dispose();
            screen.update();
        }
        }
```

Phương thức *install()* trong lớp NullRepaintManager kế thừa từ lớp RepaintManager sẽ cắt bỏ những màu dư thừa không cần thiết trong khi người dùng di chuyển chuột tới những Button. Thay vào đó nó sẽ thay thế bằng hình ảnh nền của nó.




```
    public static void install() {
        RepaintManager repaintManager = new NullRepaintManager();
        repaintManager.setDoubleBufferingEnabled(false);
        RepaintManager.setCurrentManager(repaintManager);
    }
```

Những Button sẽ được tạo bằng cách lấy một file ảnh từ folder images tương ứng khi ta truyền vào cho, khi người dùng di chuyển chuột, ấn chuột, nhả chuột thì Button này cũng thay đổi màu sắc. File ảnh được tao sẽ chỉ lấy phần ảnh có màu, còn những phần không có màu nó sẽ được cập nhật bằng ảnh của nền màn hình.

```
public JButton createButton(String name, String toolTip) {

    String imagePath = "images/" + name + ".png";
    ImageIcon iconRollover = new ImageIcon(imagePath);
    int w = iconRollover.getIconWidth();
    int h = iconRollover.getIconHeight();

    Cursor cursor =
      Cursor.getPredefinedCursor(Cursor.HAND_CURSOR);

    Image image = screen.createCompatibleImage(w, h,
      Transparency.TRANSLUCENT);
    Graphics2D g = (Graphics2D)image.getGraphics();
    Composite alpha = AlphaComposite.getInstance(
      AlphaComposite.SRC_OVER, .5f);
    g.setComposite(alpha);
    g.drawImage(iconRollover.getImage(), 0, 0, null);
    g.dispose();
    ImageIcon iconDefault = new ImageIcon(image);

    image = screen.createCompatibleImage(w, h,
      Transparency.TRANSLUCENT);
    g = (Graphics2D)image.getGraphics();
    g.drawImage(iconRollover.getImage(), 2, 2, null);
    g.dispose();
    ImageIcon iconPressed = new ImageIcon(image);

    JButton button = new JButton();
    button.addActionListener(this);
    button.setIgnoreRepaint(true);
```

```
        button.setFocusable(false);
        button.setToolTipText(toolTip);
        button.setBorder(null);
        button.setContentAreaFilled(false);
        button.setCursor(cursor);
        button.setIcon(iconDefault);
        button.setRolloverIcon(iconRollover);
        button.setPressedIcon(iconPressed);

        return button;
    }
```

Các Button sẽ được sắp xếp theo kiểu GridLayount

```
contentPane.setLayout(new GridLayout(3,3));
```

Lớp ScreenManager sẽ lắng nghe các sự kiện khi người chơi click chuột vào các Button. Ban đầu các Frame được khởi tạo và gán định cho nó ở trạng thái ẩn *dialog.setVisible(false)*. Khi người chơi click chuột vào một Button nào đó thì ngay lập tức Frame tương ứng với với sự kiện đó được đặt ở trạng thái hiện. Trong khi đó, các Button còn lại sẽ được thiết lập trạng thái Enabled vô hiệu hóa nó, người chơi chỉ có thể sử dụng các Button đó khi hoàn thành xong các thao tác tương ứng.

```
if (src == quitButton) {
        System.exit(0);
        }
        else if (src == configButton) {
          boolean enable = ! quitButton.isEnabled();
          aboutButton.setEnabled(enable);
          highScoreButton.setEnabled(enable);
          quitButton.setEnabled(enable);
          newGameButton.setEnabled(enable);
                dialogInstruction.setVisible(!enable);
        }
```

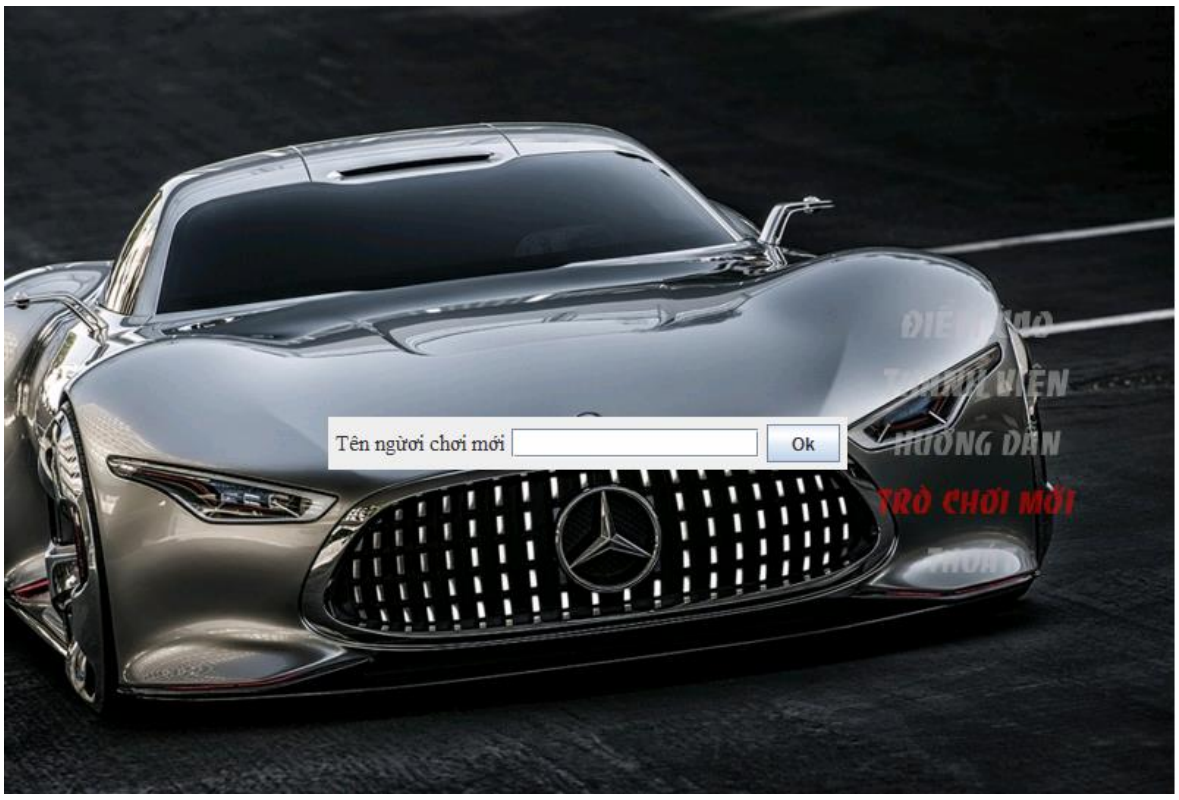
Danh sách tên của những người đang chơi trước và hoàn thành được trò chơi sẽ được lưu trong một file có tên là Score.txt. Lớp ReadWriteFile sẽ đọc thông tin từ file đó vào trả về một chuỗi gồm nội dung chứa đựng trong đó.

Ngăn cách giữa tên của các người chơi là một "
" và biến j đếm số dòng trên file đó (tức là danh sách người chơi hiện có) để trình bày giao diện dễ nhìn hơn trong khi người chơi xem danh sách người chơi trước họ.

```
public String highScore()
{
        readHighScore();
        String str = "<html> \u0110i\u1ec3m cao :";
        int j = 0;
        for(int i = 0; i < arrScore.size(); i++)
    str += "<br>" + ++j + " : " + arrScore.get(i);
    return str;

}
```

Trong những Button được xây dựng, ngoài một Button dùng để khởi tạo trò chơi mới là những Button có tính năng xem thông tin như thành viên nhóm thực hiện, cách sử dụng trò chơi, thoát, danh sách người chơi. Khi được gọi, các Frame tương ứng sẽ được hiện ra. Khi người chơi chọn khởi tạo trò chơi mới, sau khi nhập tên người chơi xong sẽ bắt đầu load trò chơi. Lớp RacingCar sẽ được gọi.



2. Bắt đầu trò chơi.

Lớp Transport được khởi tạo sẽ truyên vào một biến có kiểu dữ liệu là image. Ảnh được truyền vào chính là đại diện cho phương tiện giao thông có trong đường đua xe mà người chơi sẽ thấy xuất hiện trên màn hình. Lớp Transport có các phương thức như: *update(long elapsedTime)*, *getX()*, *getY()*, *setX(float x)*, *setY(float y)*, *getWidth()*, *getHeight()* *setVelocityX(float dx)*, *setVelocityY(float dy)*, *getImage()*, *clone()*... Sẽ dùng để lấy các tọa độ, tộc độ và gán tọa độ, tộc độ cho phương tiện giao thông đó. Riêng phương thức *update(long elapsedTime)* sẽ cập nhật lại tọa độ của phương tiện giao thông đó theo một khoảng thời gian

```
public void update(long elapsedTime)
  {
     x += dx * elapsedTime;
     y += dy * elapsedTime;
  }
```

Lớp Car là một lớp trừu tượng, nó được kế thừa từ lớp Transport, ngoài một số phương thức như đã có của lớp Transport nó còn được bổ xung thêm nhưng phương thức riêng biệt dành riêng cho nó như: *collideVertical()*, *collideHorizontal()*, *getMaxSpeed()*

Lớp Rival và lớp Obstacle kết thừa từ lớp Car. Lớp Rival, phương thức *getMaxSpeed()* sẽ thiết lập tốc độ tối đa cho các xe là đối thủ của người chơi, trong khi đó phương thức này ở lớp Obstacle sẽ thiết lập tốc độ di truyển của những chướng ngại vật trên đường.

```
private TileMap loadMap(String filename) throws IOException
  {
     ArrayList lines = new ArrayList();
     int width = 0;
     int height = 0;
     int xstart = 0;
     int ystart = 0;
     BufferedReader reader = new BufferedReader(new FileReader(filename));
     while (true) {
```

```
            String line = reader.readLine();
            if (line == null) {
                reader.close();
                break;
            }

            if (!line.startsWith("#")) {
                lines.add(line);
                height = Math.max(height, line.length());
            }
        }
        width = lines.size();
        TileMap newMap = new TileMap(width, height);
        for (int x=0; x<width; x++) {
            String line = (String)lines.get(x);
            for (int y=0; y<line.length(); y++) {
                char ch = line.charAt(y);
                int tile = ch - 'A';
                if (tile >= 0 && tile < tiles.size()) {
                    newMap.setTile(x, y, (Image)tiles.get(tile));
                }
                else if (ch == 'o') {
                    addTransport(newMap, obstacle, x, y);
                }
                else if (ch == '*') {
                    addTransport(newMap, goal, x, y);
                }
                else if (ch == '1') {
                    addTransport(newMap, rival, x, y);
                }
                else if (ch == '2') {
                    addTransport(newMap, rival, x, y);
                }
                else if (ch == 's'){
                        xstart = x;
                        ystart = y;
                        System.out.println (x + " : " + y);
```

```
            }
        }
    }
    Transport playerTransport = (Transport)player.clone();
    playerTransport.setX(TileMapRenderer.tilesToPixels(xstart) +
        (TileMapRenderer.tilesToPixels(1) -
        playerTransport.getWidth()) / 2);
    playerTransport.setY(TileMapRenderer.tilesToPixels(ystart + 1) -
        playerTransport.getHeight());
    newMap.setPlayer(playerTransport);
    return newMap;
}
```

Phương thức *loadMap()* của lớp ResourceManager sẽ trả về một TileMap. Khi được truyền vào một đối số, tham số này chính là tên của tệp tin là bản đồ đường đi khi điều khiển xe. Dựa trên số truyền vào là một chuỗi, chương trình sẽ tìm và đọc những dòng có nội dung bắt đầu không phải là # (những dòng chữ bắt đầu bằng # dùng để ghi chú thích cho tệp tin đó). Chiều dài của đường đua chính là chiều dài của dòng có nhiều kí tự nhất, chiều rộng của đường đua lại là số dòng tìm được trong tệp tin đó. Căn cứ vào tệp tin mà nó tìm thấy chương trình sẽ đọc từng kí tự, nếu trong tệp tin có kí tự 's' tức là nó đã tìm thấy vị trí xuất phát của xe được người chơi điều khiển, còn kí tự 'A', 'B' là hai bên vỉa hè của đường đua, kí tự '1', '2' là vị trí xuất phát của những đối thủ đua cùng người chơi, các kí tư 'o', 'O' là những chướng ngại vật xuất hiện trên đường đua, nó sẽ gây cản trở đến tốc độ của các xe đua, còn có kí tự '*', nếu xe của người chơi đạt tới vị trí của '*' này đồng nghĩa với việc xe đã di chuyển đến đích. Tương ứng với mỗi kí tự đó và những bức ảnh thể hiện đối tượng được đặt vào bản đồ giao diện.

Cũng trong lớp này, khi người chơi đạt tới vị trí được gọi là đích thì phương thức *loadNextMap()* được gọi. Phương thức này sẽ tăng biến toàn cụ *currentMap()* lên thêm 1 trước khi nó gọi đến phương thức *loadMap()*, biến *currentMap()* chính là biến quản lức mức của người chơi.

Trong trường hợp người chơi đã di chơi qua tất cả các mức độ của trò chơ, một biến có tên là *resuft* được gán giá trị true, đồng thời giá trị của *currentMap* được gán là 0, để nó lại quai về giá trị ban đâu.

TileMap là lớp dùng để thiết lập bản đồ, tức là chiều dài và chiều rộng của đường đua. Thông qua hai biến *height* và *width* chúng ta có thể biết được những thông số của bản đồ. Những xe phương tiên giao thông và xe của người chơi cũng được thể hiện.

```
public void draw(Graphics2D g, TileMap map,
int screenWidth, int screenHeight)
  {
    Transport player = map.getPlayer();
    int mapHeight = tilesToPixels(map.getHeight());
    int offsetY = screenHeight / 2 -
      Math.round(player.getY()) - TILE_SIZE;
    offsetY = Math.min(offsetY, 0);
    offsetY = Math.max(offsetY, screenHeight - mapHeight);
    int offsetX = screenWidth -
      tilesToPixels(map.getWidth());

    if (background == null || screenHeight > background.getHeight(null))
    {
      g.setColor(Color.gray);
      g.fillRect(0, 0, screenWidth, screenHeight);
    }

    if (background != null) {
      int y = offsetY *
        (screenHeight - background.getHeight(null)) /
        (screenHeight - mapHeight);
      int x = screenWidth - background.getWidth(null);
      g.drawImage(background, x, y, null);
    }
    int firstTileY = pixelsToTiles(-offsetY);
    int lastTileY = firstTileY +
      pixelsToTiles(screenHeight) + 1;
    for (int x = 0; x < map.getWidth(); x++) {
      for (int y = firstTileY; y <= lastTileY; y++) {

        Image image = map.getTile(x, y);
        if (image != null) {
          g.drawImage(image,
            tilesToPixels(x) + offsetX,
            tilesToPixels(y) + offsetY,
            null);
        }
      }
    }
```

```
        // draw player
        g.drawImage(player.getImage(),
           Math.round(player.getX()) + offsetX,
           Math.round(player.getY()) + offsetY,
           null);
        // draw Cars
        Iterator i = map.getTransport();
        while (i.hasNext()) {
           Transport transport = (Transport)i.next();
           int x = Math.round(transport.getX()) + offsetX;
           int y = Math.round(transport.getY()) + offsetY;
           g.drawImage(transport.getImage(), x, y, null);
           if (transport instanceof Car &&
              x >= 0 && x < screenWidth)
           {
              ((Car)transport).wakeUp();
           }
        }
     }
```

Phương thức *draw()* của lớp TileMapRenderer sẽ cập nhật liên tục màn hình. Khi xe ôtô của người chơi di chuyển thì màn hình cũng dịch chuyển theo và cập nhật vị trí của các xe khác có trên đường đua.

Lớp InputManager lắng nghe các sự kiện KeyListener, MouseListener, MouseMotionListener và MouseWheelListener. Lớp có phương thức *setCursor()* có nhiệm vụ khóa con trỏ, chúng ta sẽ không thể di chuyển chuột trong quá trình sử dụng trò chơi.

```
public void setCursor(Cursor cursor) {

    comp.setCursor(cursor);

}
```

Luôn lắng nghe sự kiện có liên quan đến bàn phím như ấn, nhả một hay nhiều phím khác nhau. Môi một phím được ấn hay nhả đều được kiểm tra và nó thiết lập phương thức *press()* và *ralease()* của lớp GameAction

```
public void keyPressed(KeyEvent e) {
    GameAction gameAction = getKeyAction(e);
    if (gameAction != null) {
        gameAction.press();
    }
    e.consume();
}


public void keyReleased(KeyEvent e) {
    GameAction gameAction = getKeyAction(e);
    if (gameAction != null) {
        gameAction.release();
    }
    e.consume();
}
```

Phương thức *checkInput()* kiểm tra liên tục các sự kiện nhập từ bàn phím thông qua lớp InputManager. Khi các phím có liên quan được ấn thì tọa độ hay là vị trì của xe được thay đổi theo bằng cách thiết lập một tốc độ cho xe của người chơi.

```
    private void checkInput(long elapsedTime)
    {

    if (exit.isPressed()) {
        stop();
    }

    Player player = (Player)map.getPlayer();
        {
        float velocityX = 0;
        float velocityY = 0;
        if (moveLeft.isPressed()) {
            velocityX-=player.getMaxSpeed();
        }
        if (moveRight.isPressed()) {
            velocityX+=player.getMaxSpeed();
        }
        if (moveForward.isPressed()) {
            velocityY-=player.getMaxSpeed();
        }
        if (moveBack.isPressed()) {
            velocityY+=player.getMaxSpeed();
        }
        player.setVelocityX(velocityX);
        player.setVelocityY(velocityY);
    }

  }
```

Phương thức *update()* có nhiệm vụ cập nhật và thay đổi vị trí của các xe, bối cảnh trong chương trình bằng cách gọi lại các hàm tương ứng với nhứng nhiệm đó.

```
public void update(long elapsedTime) {
    Car player = (Car)map.getPlayer();


    checkInput(elapsedTime);

    updateTransport(player, elapsedTime);
    player.update(elapsedTime);

    Iterator i = map.getTransport();
    while (i.hasNext()) {
        Transport transport = (Transport)i.next();
        if (transport instanceof Car) {
            Car car = (Car)transport;
            updateTransport(car, elapsedTime);
        }
        transport.update(elapsedTime);
    }
}
```

Phương thức sẽ quán lí tọa độ các xe là đối thủ của người chơi và các xe có trên đường đua mà người chơi đang điều khiển xe.

```
private void updateTransport(Car car, long elapsedTime)
 {
    // change x
    float dx = car.getVelocityX();
    float oldX = car.getX();
    float newX = oldX + dx * elapsedTime;
    Point tile =
       getTileCollision(car,  newX, car.getY());
    if (tile == null) {
       car.setX(newX);
    }
    else {
       if (dx > 0) {
          car.setX(
             TileMapRenderer.tilesToPixels(tile.x)  -
             car.getWidth());
       }
       else if (dx < 0) {
          car.setX(
             TileMapRenderer.tilesToPixels(tile.x  + 1));
       }
       car.collideHorizontal();
    }
    if (car instanceof Player) {
       checkPlayerCollision((Player)car);
    }
    // change y
    float dy = car.getVelocityY();
    float oldY = car.getY();
    float newY = oldY + dy * elapsedTime;
    tile = getTileCollision(car,  car.getX(), newY);
    if (tile == null) {
       car.setY(newY);
    }
    else {
       if (dy > 0) {
          car.setY(
             TileMapRenderer.tilesToPixels(tile.y)  -
             car.getHeight());
       }
       else if (dy < 0) {
          car.setY(
             TileMapRenderer.tilesToPixels(tile.y  + 1));
       }
       car.collideVertical();
    }
    if (car instanceof Player) {
       checkPlayerCollision((Player)car);
    }
```

```
}
```

Phương thức *getTileCollision()* sẽ chuyển đổi tọa độ của TileMap thành tọa độ của màn hình

```
public Point getTileCollision(Transport transport,
    float newX, float newY)
{
    float fromX = Math.min(transport.getX(), newX);
    float fromY = Math.min(transport.getY(), newY);
    float toX = Math.max(transport.getX(), newX);
    float toY = Math.max(transport.getY(), newY);


    int fromTileX = TileMapRenderer.pixelsToTiles(fromX);
    int fromTileY = TileMapRenderer.pixelsToTiles(fromY);
    int toTileX = TileMapRenderer.pixelsToTiles(
       toX + transport.getWidth() - 1);
    int toTileY = TileMapRenderer.pixelsToTiles(
       toY + transport.getHeight() - 1);


    for (int y = fromTileY; y <= toTileY; y++){
       for (int x = fromTileX; x <= toTileX; x++) {
          if (y < 0 || y >= map.getHeight() ||
                x < 0 || x >= map.getWidth() ||
                map.getTile(x, y) != null)
          {
             pointCache.setLocation(x, y);
             return pointCache;
          }
       }
    }

    return null;
}
```

Hiển thị thời gian tính từ khi bắt đầu trò chơi đến thời điểm hiện tại, mức độ mà người chơi đang chơi và in ra màn hình thông báo đó để người chơi biết. Nó cũng quán lí và cho biết người chơi đã đạt yêu cầu để chuyển sang mức chơi mới chưa. Nếu người chơi về đến đích quá thời gian yêu cầu thì mức độ vẫn giữ nguyên ở hiện tại và bản đồ sẽ load lại tệp tin ứng với mức chơi đó.

```
private void drawControl(Graphics2D g)
{
g.drawImage(resourceManager.loadImage("10.png"), 0 , 0, null);
g.setColor(Color.BLUE);
g.drawString("Time : " + String.valueOf(System.currentTimeMillis() - timeStart),
10, 30);
 g.drawString("Level : " + String.valueOf(resourceManager.getCurrentMap()), 10,
80);
```

```
if(resourceManager.IsResuft())
                    g.drawString(" You complete the game ", 10,
screen.getHeight()/2);

}
```

Các chướng ngại vật xuất hiện ngẫu nhiên làm cản trở đến tốc độ xe của người chơi, phương *checkPlayerCollision()* sẽ làm điều này

```
public void checkPlayerCollision(Player player)
  {

    Transport collisionTransport = getTransportCollision(player);

    if (collisionTransport instanceof PowerUp) {
      acquirePowerUp((PowerUp)collisionTransport);
    }
    else if (collisionTransport instanceof Car) {
      Transport transport = (Transport)collisionTransport;

      if(transport.getY() < player.getY())
      {
            player.setY(transport.getY() + player.getHeight());
      }
      else if(transport.getY() > player.getY())
      {
            transport.setY(player.getY() + transport.getHeight());
      }
      else if(transport.getX() > player.getX())
      {
            player.setX(transport.getX() - player.getWidth());
      }
      else if(transport.getX() < player.getX())
      {
            player.setX(transport.getX() + player.getWidth());
      }

    }

  }
```

Khi gặp các chướng ngại, nếu chướng ngại vật đó mà là cái đích đến thì nó sẽ gọi đến phương thức *acquirePowerUp()*, phương thức này sẽ tự động chuyển bản đồ mới, hay đường đua mới. Nhưng để làm điều đó thành công thì nó kiểm tra xem thời gian về đích của người chơi có đạt yêu cầu không, nếu đạt tới yêu cầu về thời gian thì nó sẽ tự động nâng bài, còn trong trường hợp ngược lại chương trình sẽ không nâng bài mới mà thay vào đó chương trình sẽ load lại đường đua mà người chơi vừa chưa hoàn thành mục tiêu.

```
public void acquirePowerUp(PowerUp powerUp) {
```

```
            map.removeTransport(powerUp);

            else if (powerUp instanceof PowerUp.Goal) {

                if(System.currentTimeMillis() - timeStart < 40000)
                {

                        map = resourceManager.loadNextMap();
                        timeStart = System.currentTimeMillis();
                }
                else
                {
                        map = resourceManager.reloadMap();
                        timeStart = System.currentTimeMillis();
                }

                if(resourceManager.getCurrentMap() == 3 && !rwf.isScore(name))
                        rwf.writeFileScore(name);

            }
        }
```

Bằng cách kiểm tra phương thức *IsResuft()* của lớp ResourceManager liên tục. Khi đã quan hết các mức của trò chơi, thì sẽ in ra màn hình dòng thông báo "You complete the game!"

```
            public boolean IsResuft()
            {
                    return resuft;
            }

            public int getCurrentMap()
            {
                    return currentMap;
            }
```

IV. Kết luận

Trong thời gian thực hiện bài tập lớn, nhờ sự chỉ bảo của thầy Trần Huy Hoàng và sự giúp đỡ của các bạn trong lớp, nhóm chúng em đã thu được nhiều kết quả trong việc học lập trình Java.

Do thời gian và khả năng có hạn nên bài tập lớn của nhóm chúng em còn rất nhiều thiếu sót, chúng em rất mong nhận được sự góp ý, giúp đỡ của thầy và các bạn để bài tập của chúng em được hoàn thiện hơn