**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



**BÁO CÁO BÀI TẬP LỚN**

**--------------------**

**Môn học: Công nghệ Java**

**Đề Tài: Xây dựng game Pikachu**

**Giảng viên hướng dẫn: Lê Thị Thanh Thủy**

**Nhóm thực hiện: Nhóm 6 – CNTT3-K62**

Phạm Minh Hiếu

Đặng Bá Tài

Nguyễn Viết Tú

**Năm học 2022-2023**

Mục lục

[Lời Nói Đầu 3](#_Toc134021032)

[**CHƯƠNG 1 TÓM TẮT ĐỀ TÀI** 4](#_Toc134021033)

[**1.1** **Giới thiệu game pikachu** 4](#_Toc134021034)

[**1.2** **Phạm vi nghiên cứu** 4](#_Toc134021035)

[**1.3** **Chức năng công việc đã làm được** 5](#_Toc134021036)

[**CHƯƠNG 2 PHÂN TÍCH MÔ HÌNH GAME** 5](#_Toc134021037)

[**2.1Các công nghệ sử dụng** 5](#_Toc134021038)

[**2.2Mô hình game Pikachu** 6](#_Toc134021039)

[**2.3 MÔ HÌNH TRIỂN KHAI** 6](#_Toc134021040)

[**CHƯƠNG 3: THIẾT KẾ VÀ LẬP TRÌNH** 8](#_Toc134021041)

[**GIỚI THIỆU SƠ QUA CÁC LỚP CÓ TRONG TRÒ CHƠI** 8](#_Toc134021042)

[**3.1: Xây dựng giao diện của game Pikachu** 12](#_Toc134021043)

[**3.1.1. *Xây dựng giao diện trang Home Pikachu*** 12](#_Toc134021044)

[**3.1.*2*. Xử lý thuật toán đối với các Icon** 15](#_Toc134021045)

[**3.1.*3*. Thêm các Icon vào khối để hiển thị** 19](#_Toc134021046)

[**3.1.*4*. Chạy chương trình và hiển thị các Icon** 22](#_Toc134021047)

[**3.2 Hoàn thiện giao diện game** và **Thao tác xử lý cơ bản với các Icon** 23](#_Toc134021048)

[**3.2.1 Hoàn thành giao diện** 23](#_Toc134021049)

[3.2.2. Thao tác xử lý cơ bản với các Icon 31](#_Toc134021050)

[**3.3 Thuật toán tìm kiếm giữa 2 Icon giống nhau** 36](#_Toc134021051)

[**3.3.1. Ý tưởng** 36](#_Toc134021052)

[**Thuật toán tìm kiếm đường đi giữa 2 icon giống nhau** 37](#_Toc134021053)

[**3.3.3. Hoàn thiện hàm checkTwoPoint(Point p1, Point p2).** 44](#_Toc134021054)

[**3.4 Hoàn thiện game với các hiệu ứng âm thanh** 45](#_Toc134021055)

# Lời Nói Đầu

Em rất vui khi được viết báo cáo này về trò chơi Pikachu trên nền tảng Java. Như chúng ta đã biết, Pikachu là một trò chơi giải đố được yêu thích trên toàn thế giới. Nó được phát triển bởi một công ty trò chơi Nhật Bản và được giới thiệu vào năm 1996. Từ đó đến nay, trò chơi đã trở thành một trong những trò chơi giải đố phổ biến nhất trên thế giới và đã được phát triển trên nhiều nền tảng khác nhau.

Trong báo cáo này, em sẽ giới thiệu cho bạn về quá trình phát triển trò chơi Pikachu trên nền tảng Java. em sẽ bắt đầu bằng cách giải thích các khái niệm cơ bản của lập trình Java, như lớp, đối tượng, kế thừa, đa hình, giao diện và nhiều khái niệm khác. Sau đó, em sẽ trình bày các bước để xây dựng trò chơi Pikachu bằng ngôn ngữ lập trình Java, bao gồm:

1. Thiết kế giao diện người dùng - Đây là bước đầu tiên để tạo ra một trò chơi Pikachu trên nền tảng Java. Em sẽ giới thiệu cho bạn cách thiết kế giao diện người dùng đơn giản và trực quan cho trò chơi này.

2. Xử lý sự kiện - Trong bước này, em sẽ giải thích cách xử lý các sự kiện như di chuyển chuột, nhấn phím và các sự kiện khác liên quan đến trò chơi Pikachu.

3. Xây dựng cấu trúc dữ liệu - Đây là bước quan trọng nhất trong việc xây dựng trò chơi Pikachu. Em sẽ giới thiệu cho bạn cách tạo ra một cấu trúc dữ liệu phù hợp để lưu trữ và quản lý các hình ảnh và vị trí của các con Pikachu.

4. Tạo hiệu ứng âm thanh - Trong bước này, em sẽ giới thiệu cho bạn cách tạo ra các hiệu ứng âm thanh như âm thanh khi điểm số tăng lên hoặc khi người chơi kết thúc trò chơi.

5. Kiểm tra điều kiện thắng cuộc - Trong bước cuối cùng, em sẽ giải thích cách kiểm tra điều kiện thắng

Em hy vọng rằng báo cáo này sẽ giúp các bạn hiểu rõ hơn về quá trình phát triển trò chơi Pikachu và giúp các bạn có thêm kiến thức và kinh nghiệm về lập trình trò chơi. Hy vọng rằng báo cáo này sẽ giúp các bạn đạt được mục tiêu của mình trong việc tìm hiểu về lập trình game và cải thiện kỹ năng của mình.

Cảm ơn cô đã đọc báo cáo của em.

# **CHƯƠNG 1 TÓM TẮT ĐỀ TÀI**

## **Giới thiệu game pikachu**

* Game Pikachu là một game puzzle được phát triển bằng ngôn ngữ lập trình Java.
* Mục đích của game là loại bỏ hết các cặp hình Pikachu giống nhau bằng cách kết nối chúng với nhau thông qua các đường nối không quá ba góc.
* Game có một số tính năng như cấp độ khó tăng dần, thời gian giới hạn, gợi ý, điểm số và hình ảnh đẹp.
* Luật game pikachu như sau:
  + Trên màn hình sẽ hiển thị một bảng chứa các hình ảnh của các loài Pikachu khác nhau.
  + Nhiệm vụ của người chơi là tìm và kết nối các cặp hình ảnh Pikachu giống nhau bằng cách vẽ các đường nối giữa chúng.
  + Đường nối chỉ có thể được vẽ nếu nó không vượt qua bất kỳ hình ảnh Pikachu nào khác và không có quá ba góc.
  + Các cặp hình ảnh Pikachu giống nhau sẽ bị loại bỏ khỏi bảng và điểm số của người chơi sẽ được cộng thêm.
  + Người chơi chiến thắng khi đã loại bỏ hết tất cả các cặp hình ảnh Pikachu và điểm số của họ được tính dựa trên thời gian còn lại.
  + Nếu người chơi không thể tìm thấy bất kỳ cặp hình ảnh Pikachu nào nữa hoặc thời gian đã hết, thì người chơi sẽ thua cuộc.

## **Phạm vi nghiên cứu**

* Phạm vi nghiên cứu: Tự nghiên cứu
* Phạm vi ứng dụng: Cá nhân

## **Chức năng công việc đã làm được**

* + - Thiết kế giao diện: Tạo ra giao diện cho trò chơi với các thành phần như bảng điểm, thời gian chơi, màn hình chính, màn hình chơi, hình ảnh các loại thú Pikachu.
    - Xây dựng thuật toán pikachu: xây dựng ma trân chứa các ô pikachu một các ngẫu nhiên, thuật toán tìm đường đi giữa 2 icon giống nhau
    - Xử lý sự kiện: Xử lý các sự kiện như click chuột, di chuyển các ô trống, kiểm tra sự khớp nhau giữa các ô Pikachu để tạo ra điểm số.
    - Thời gian chơi: Tạo ra tính năng đếm thời gian để người chơi có thể theo dõi thời gian chơi và kết thúc game khi hết giờ.
    - Các cấp độ khác nhau: Tạo ra các cấp độ khác nhau với độ khó tăng dần, bao gồm 3 cấp độ chơi khác nhau và cho người chơi chọn lưới game phù hợp.
    - Điểm số và bảng xếp hạng: Cập nhật điểm số của người chơi và hiển thị trên bảng xếp hạng.
    - Âm thanh: Thêm các hiệu ứng âm thanh cho các hoạt động như lựa chọn, khớp thú Pikachu, hoàn thành trò chơi.
    - Hiệu ứng: khi click vào ô hoặc khi hover vào sẽ set border cho các ô đó
    - Hiện thị bảng thông báo khi người chơi chiến thắng, hết giờ hoặc các option trên thanh menu
    - Tạm dừng trò chơi: thì nhạc và thời gian sẽ được tạm dừng và được tiếp chạy tiếp khi người chơi bấm tiếp tục
    - Tối ưu hóa trò chơi: Cải thiện hiệu suất trò chơi bằng cách sử dụng cấu trúc dữ liệu và giải thuật tối ưu để giảm thiểu thời gian xử lý và tối đa hóa trải nghiệm người chơi.

# **CHƯƠNG 2 PHÂN TÍCH MÔ HÌNH GAME**

## **2.1Các công nghệ sử dụng**

* Ngôn ngữ lập trình Java Core
* Java Swing để tạo giao diện đồ họa
* Java AWT để xử lý sự kiện đầu vào
* Java sound: Xử lí phần âm thanh

## **2.2Mô hình game Pikachu**

* Khi bắt đầu game sẽ hiện ra phần menu game: gồm các option lựa chọ độ khó(ở đây là giới hạn về mặt thời gian khi độ khó càng tăng thời gian càng giảm), chọn kích cỡ (ô ma trận) phù hợp với người chơi, khi đã chọn hết option thì nhất nút băt đầu game để chơi
* Hình ảnh của game Pikachu được lưu trữ dưới dạng các file ảnh PNG.
* Khi bắt đầu game, các hình ảnh sẽ được tải lên và đưa vào một ma trận lưu trữ vị trí của các hình ảnh, lúc này thời gian và ẩm thanh sẽ chạy song song
* Khi người chơi chọn một cặp hình Pikachu, chương trình sẽ kiểm tra xem có đường nối nào giữa hai hình ảnh đó không. Nếu có, chương trình sẽ loại bỏ cặp hình ảnh đó, tăng điểm số, phát ra âm thanh khi hai ô đó match và tiếp tục chơi.
* Nếu các ô được loại bỏ hết trước khi hết thời gian sẽ phát ra đoạn nhạc chiến thắng hoặc chưa loại bỏ được hết các ô khi thời gian kết thức sẽ phát ra đoạn nhạc thua game và hiện ra một bảng thông báo hỏi người chơi có muốn tiếp tục game hoặc thoát
* Các hình ảnh Pikachu có thể được xếp ngẫu nhiên trong ma trận mỗi khi khởi động game hoặc khi chơi lại.
* Trên thanh option có phần chưa năng New Game để bắt đầu một ván mới, Quay trở lại trang home hoặc thoát

## **2.3 MÔ HÌNH TRIỂN KHAI**

* Mô hình MVC (Model-View-Controller) là một kiến trúc phần mềm phổ biến trong lập trình. Nó giúp phân tách chức năng của một ứng dụng thành ba phần chính: model, view và controller. Trong đó, model đại diện cho các dữ liệu và logic của ứng dụng, view đại diện cho giao diện người dùng và controller đại diện cho các hành động xử lý sự kiện của người dùng.
* Trong dự án game pikachu, mô hình MVC được áp dụng để phân tách chức năng của game thành ba phần chính:
* Model: Model trong game pikachu bao gồm các dữ liệu như ma trận ô vuông, các loại thú Pikachu và điểm số của người chơi. Ngoài ra, model còn chứa các phương thức xử lý logic của game, chẳng hạn như kiểm tra sự khớp nhau giữa các thú Pikachu, tính điểm số, tạo ra các cấp độ khác nhau,...
* View: View trong game pikachu đại diện cho giao diện người dùng của trò chơi. Nó được thiết kế để hiển thị các thành phần như bảng điểm, màn hình chính, màn hình chơi, hình ảnh các loại thú Pikachu,... View còn có nhiệm vụ hiển thị thông tin về thời gian chơi và điểm số của người chơi.
* Controller: Controller trong game pikachu xử lý các sự kiện người dùng và thay đổi trạng thái của model. Ví dụ, khi người chơi click vào một ô vuông, controller sẽ xử lý sự kiện này để kiểm tra xem ô vuông đó có chứa thú Pikachu hay không. Nếu có, controller sẽ ghi nhận điểm số của người chơi và xóa các thú Pikachu tương ứng trên ma trận.
* Với mô hình MVC, việc phát triển và bảo trì dự án trở nên dễ dàng hơn, vì các thành phần trong mô hình hoạt động độc lập và không phụ thuộc vào nhau quá nhiều. Nếu có thay đổi trong logic của game, chúng ta chỉ cần chỉnh sửa phần model mà không ảnh hưởng đến phần view hoặc controller. Tương tự, nếu có thay đổi trong giao diện của game, chúng ta chỉ cần chỉnh sửa phần view mà không ảnh hưởng đến phần model hay controller.

# **CHƯƠNG 3: THIẾT KẾ VÀ LẬP TRÌNH**

## **GIỚI THIỆU SƠ QUA CÁC LỚP CÓ TRONG TRÒ CHƠI**

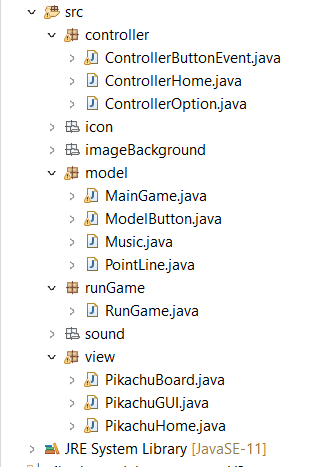
* Ta đi tạo 3 package theo mô hình MVC:

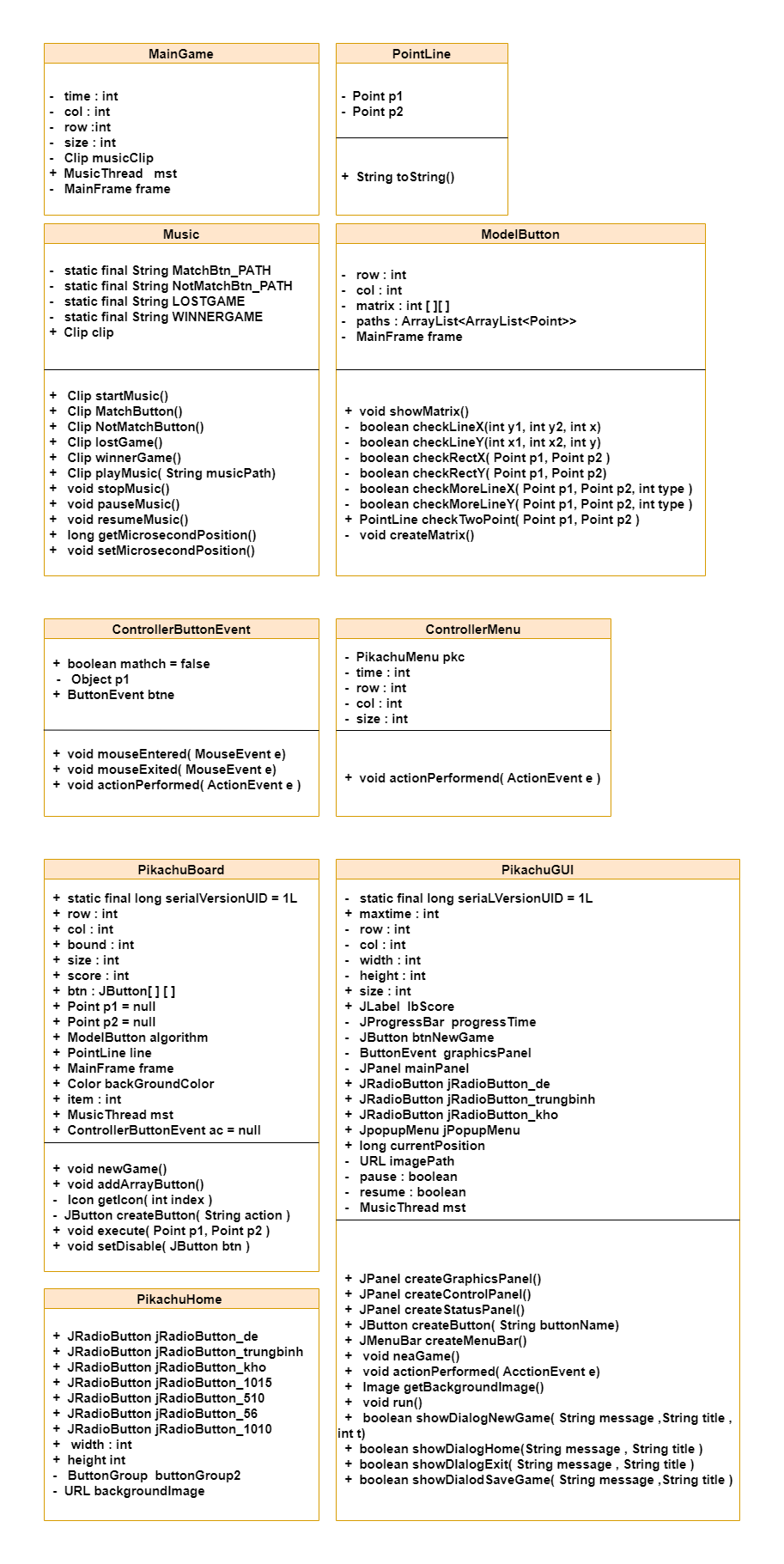
+ Đầu tiên tạo package Model trong đó:

* Lớp ModelButton: là một model để thực hiện các xử lý logic cho trò chơi. Cụ thể, class này thể hiện ma trận các ô vuông trong trò chơi và xử lý các thao tác kiểm tra đường đi giữa các ô vuông, kiểm tra xem có hình chữ nhật nằm giữa hai ô vuông hay không, kiểm tra thêm các đường đi khác nếu hình chữ nhật được xác định, và hiển thị ma trận lên màn hình.
* Class PointLine: định nghĩa một đối tượng biểu diễn một đoạn thẳng trong không gian hai chiều bằng cách lưu trữ hai điểm p1 và p2 của đoạn thẳng đó
* Class Music: có vai trò là quản lý âm thanh cho ứng dụng. Nó chứa các phương thức để phát nhạc nền, âm thanh khi click các nút trong game, âm thanh khi thắng hoặc thua, tạm dừng, tiếp tục phát nhạc. Ngoài ra, class cũng hỗ trợ các phương thức để thao tác với audio clip như stop, pause, resume và thay đổi vị trí thời gian của audio clip. Class Music sử dụng thư viện Java Audio để phát các file audio.
* Class MainGame: có vai trò quản lý trò chơi Pikachu, bao gồm thời gian chơi, số hàng và số cột của bảng, kích thước của các ô, khởi tạo giao diện chính MainFrame và thực hiện việc bắt đầu chơi trò chơi. Nó cũng có vai trò quản lý âm thanh trong trò chơi bằng cách sử dụng lớp MusicThread để phát nhạc và lặp lại khi nhạc kết thúc. Ngoài ra, MainGame cũng có vai trò tính toán và đếm thời gian chơi và hiển thị cửa sổ thông báo khi kết thúc trò chơi hoặc người chơi muốn chơi lại.

+ Tạo package View trong đó:

* Class PikachuHome là một lớp định nghĩa giao diện trang Home cho trò chơi Pikachu. Nó mở rộng từ lớp JFrame để tạo ra một cửa sổ giao diện người dùng. Class này bao gồm các thành phần trên giao diện như các nút bấm bắt đầu , các nút radio, nhãn, các thành phần này được sử dụng để lựa chọn các option trước khi bắt đầu chơi game pikachu
* Class PikachuBoard là một JPanel trong game Pikachu, chứa toàn bộ các ô vuông được tạo thành bằng JButton để hiển thị giao diện game. Nó được sử dụng để quản lý các JButton, lắng nghe sự kiện click chuột trên JButton, cập nhật giao diện cho JButton và thực hiện các hành động xoá các cặp ô vuông khi được chọn đúng. Class này còn chứa các thông số cấu hình của game như số lượng dòng, cột, kích thước ô vuông, điểm số và các đối tượng hỗ trợ game như ModelButton và MusicThread.
* Class PikachuGUI trong trò chơi Pikachu là lớp đại diện cho giao diện người dùng (GUI), kế thừa từ lớp JFrame và quản lý các thành phần giao diện như JProgressBar, JLabel, JButton, JPanel, JMenuBar, v.v... Nó cung cấp các phương thức để tạo ra các thành phần GUI và xử lý sự kiện người dùng thông qua các listener và các controller khác nhau. PikachuGUI cũng quản lý các thông tin của trò chơi như thời gian, điểm số, vị trí hiện tại của Pikachu, kích thước của board game, v.v. và truyền chúng cho các đối tượng khác trong trò chơi để xử lý các logic và tính toán phù hợp. Nó là một lớp trung tâm trong việc tạo ra các đối tượng khác của trò chơi và kết nối chúng với nhau, giúp điều khiển hành vi của trò chơi.

 Cấu trúc thư mục project



### **3.1: Xây dựng giao diện của game Pikachu**

### **3.1.1. *Xây dựng giao diện trang Home Pikachu***

Trong tổng quan, lớp PikachuHome đóng vai trò quan trọng trong việc tạo ra giao diện người dùng cho trò chơi Pikachu. Nó cung cấp cho người dùng các tùy chọn để chọn mức độ chơi và kích thước lưới trò chơi, và sử dụng một ActionListener để xử lý các sự kiện khi người dùng tương tác với giao diện người dùng. Khi người dùng đã chọn mức độ và kích thước lưới, lớp này sẽ bắt đầu trò chơi bằng cách tạo một đối tượng PikachuGame và khởi động trò chơi.

Class PikachuHome là một lớp kế thừa từ lớp JFrame để tạo ra một giao diện đồ họa cho trò chơi Pikachu.

Đối tượng PikachuHome bao gồm một số thành phần chính bao gồm:

Các đối tượng JRadioButton, được sử dụng để chọn mức độ chơi và kích thước lưới trò chơi. Các đối tượng này được khởi tạo và thiết lập tùy chọn mặc định của chúng trong phương thức khởi tạo của lớp.

Một URL backgroundImage được sử dụng để tải hình nền cho giao diện người dùng từ đường dẫn đã cho.

Một hình ảnh Pikachu được thêm vào label jLabel bằng cách sử dụng đối tượng ImageIcon.

Các đối tượng JPanel được sử dụng để chứa các thành phần giao diện người dùng khác.

Một ActionListener được khởi tạo để xử lý các sự kiện khi người dùng tương tác với giao diện người dùng.

Một số phương thức đơn giản được sử dụng để thiết lập và hiển thị giao diện người dùng.

Dưới đây là một số thuộc tính và phương thức của lớp PikachuHome:

public JRadioButton jRadioButton\_de: Đối tượng JRadioButton dùng để chọn mức độ "Dễ".

public JRadioButton jRadioButton\_trungbinh: Đối tượng JRadioButton dùng để chọn mức độ "Trung Bình".

public JRadioButton jRadioButton\_kho: Đối tượng JRadioButton dùng để chọn mức độ "Khó".

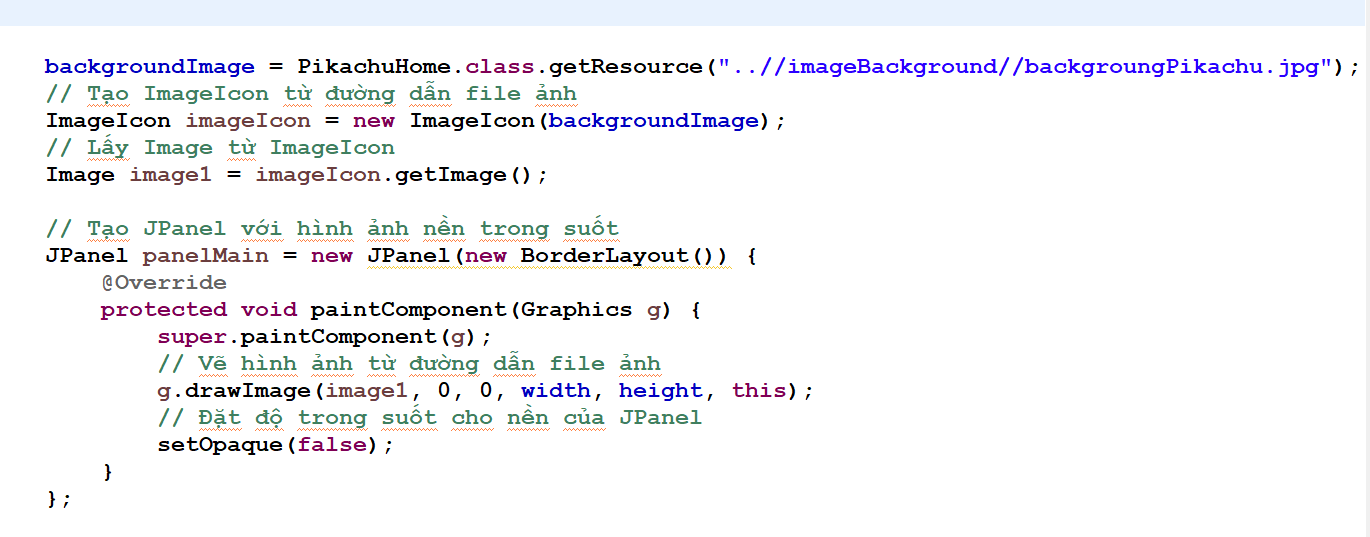
public JRadioButton jRadioButton\_1015: Đối tượng JRadioButton dùng để chọn kích thước lưới 10x15

public JRadioButton jRadioButton\_510: Đối tượng JRadioButton dùng để chọn kích thước lưới 5x10.

public JRadioButton jRadioButton\_56: Đối tượng JRadioButton dùng để chọn kích thước lưới 5x6.

public JRdioButton jRadioButton\_1010: Đối tượng JRadioButton dùng để chọn kích thước lưới 10x10.

private ButtonGroup buttonGroup2: Đối tượng ButtonGroup dùng để quản lý các JRadioButton cho kích thước lưới.



Cụ thể, đoạn code này sử dụng hình ảnh được định nghĩa trước và được lưu trữ trong thư mục của ứng dụng để đặt làm hình nền cho JPanel.

Dòng đầu tiên sử dụng phương thức getResource của class PikachuHome để lấy đường dẫn tới file ảnh nền được lưu trữ trong thư mục imageBackground của ứng dụng. Đường dẫn này được lưu trữ trong một biến kiểu URL.

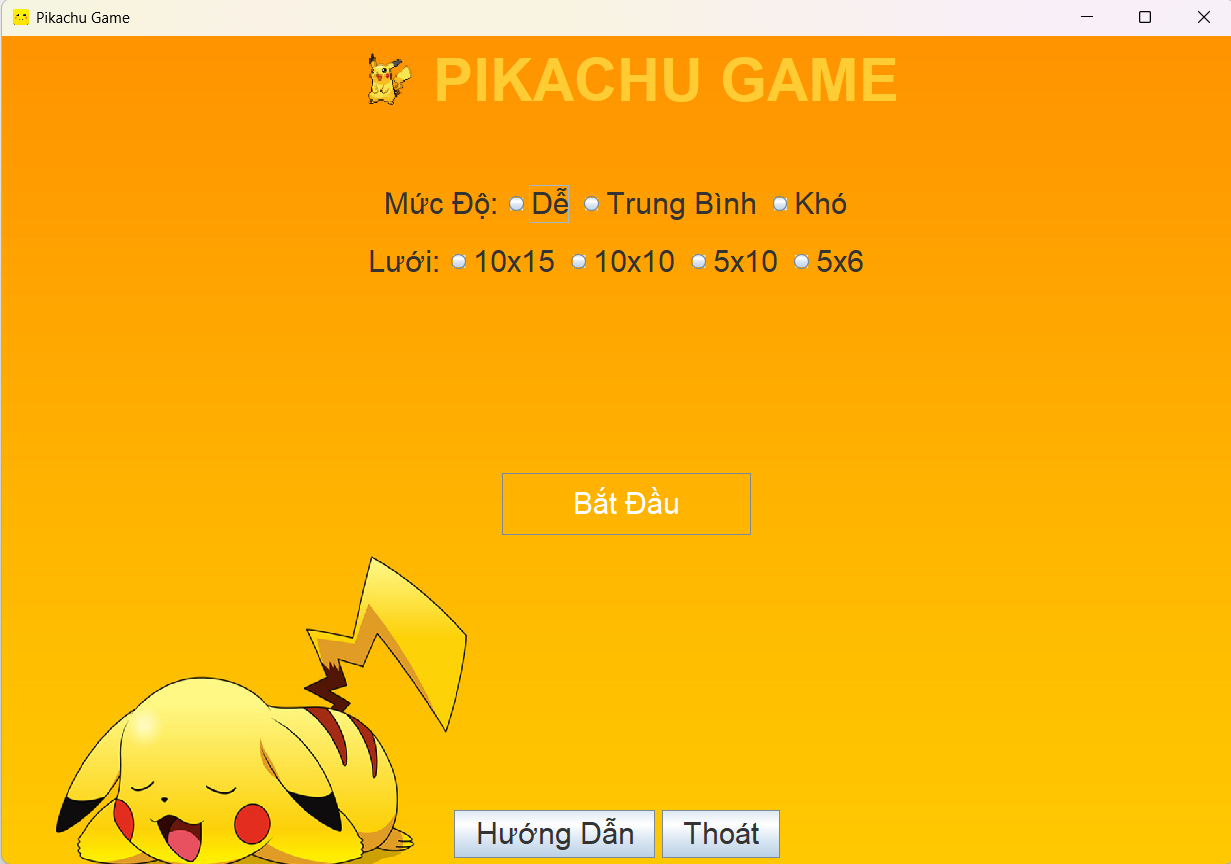
Dòng thứ hai tạo một đối tượng ImageIcon từ đường dẫn đến file ảnh nền.

Dòng thứ ba lấy Image từ ImageIcon đã tạo.

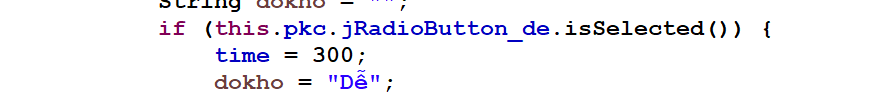
Dòng tiếp theo tạo một đối tượng JPanel và thiết lập giao diện cho nó bằng cách sử dụng BorderLayout.

Dòng thứ năm override phương thức paintComponent của JPanel. Phương thức này được gọi bởi hệ thống khi nội dung của JPanel cần được vẽ lại. Trong phương thức này, ta sử dụng đối tượng Graphics để vẽ hình ảnh nền lên JPanel.

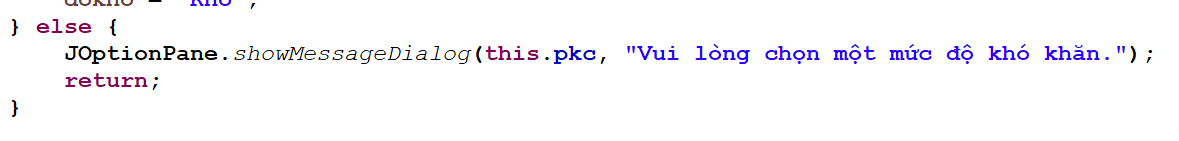
Dòng thứ sáu đặt độ trong suốt cho nền của JPanel bằng cách gọi phương thức setOpaque(false) của JPanel. Điều này sẽ làm cho nền của JPanel trong suốt, để cho hình ảnh nền được vẽ lên phía sau JPanel.



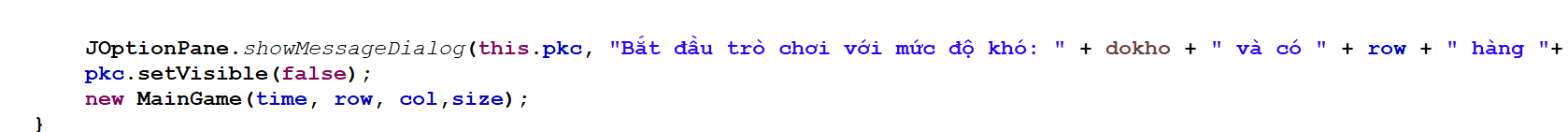
* Sử lí sự kiện cho trang home pikachu:
  + Class ControllerHome là một lớp điều khiển (controller) trong ứng dụng Pikachu. Nó được triển khai từ ActionListener để lắng nghe các sự kiện tương tác từ người dùng và xử lý chúng. Class này có các thuộc tính
  + private PikachuHome pkc: đối tượng PikachuHome, là màn hình chính của trò chơi Pikachu.
  + public int time: thời gian của trò chơi, được thiết lập bởi người chơi khi bắt đầu trò chơi.
  + public int row: số hàng của lưới trong trò chơi, được thiết lập bởi người chơi khi bắt đầu trò chơi.
  + public int col: số cột của lưới trong trò chơi, được thiết lập bởi người chơi khi bắt đầu trò chơi.
  + public int size: kích thước của mỗi ô trong lưới, được tính toán dựa trên số hàng và số cột của lưới.
  + Constructor của lớp này nhận đối tượng PikachuHome như một tham số và lưu trữ nó vào thuộc tính pkc.
  + Khi người chơi chọn mức chơi dễ thời gian sẽ set là 300s



với mức khó hơn thì thời gian giảm dần, nếu như người chơi chưa chọn mức độ mà bấm bắt đầu chơi game thì hiện thông báo và sẽ không cho chơi game



* + Khi người chọn lưới là 10x10 lúc này sẽ set col, row bằng 10
  + Khi ngươi đã chọ được mức độ và lưới thì bắt đầu chơi game

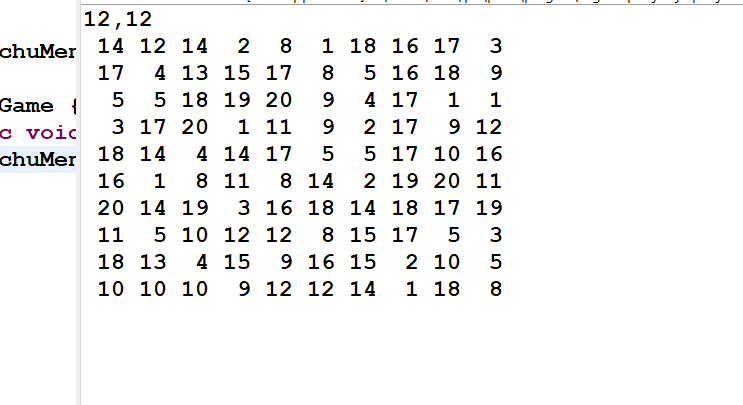


Sẽ ẩn đi trang home pikachu và khởi tạo hàm MainGame với các đối số tương ứng khi tùy chọn

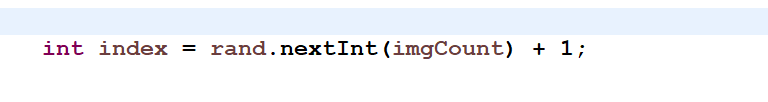
### **3.1.*2*. Xử lý thuật toán đối với các Icon**

Để xử lý vấn đề này,đầu tiên, em tạo class **ModelButton** bên trong package Model để xử lý những thuật toán của game. Mình coi mỗi Icon là 1 **Point** với 2 tọa độ **x,y**và tạo ra một ma trận **matrix[row][col]**với **row,col**là số hàng và cột của các Icon được tạo ra. Mục đích của việc tạo ra ma trận này là để tạo ra một phiên bản thu nhỏ Game Pikachu của chúng ta, chỉ là thay các Icon bằng các con số. Mỗi Icon sẽ là một con số riêng, hai con số giống nhau chính là hai Icon giống nhau. Bằng việc tạo ra ma trận này, chúng ta có thể xử lý những thuật toán của game gián tiếp qua các con số và vị trí của chúng trong ma trận.

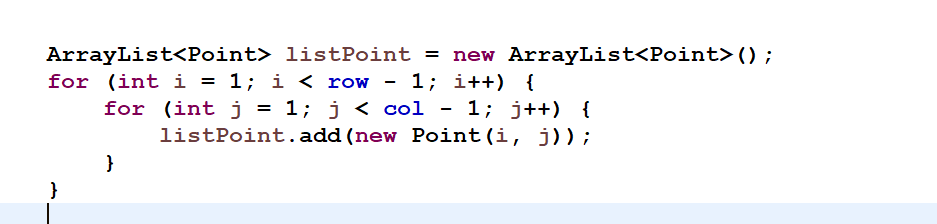




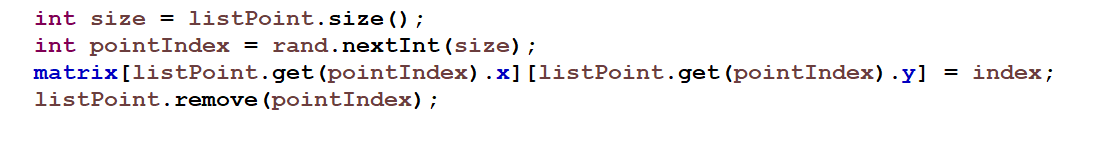
Từ bây giờ, ta sẽ coi như các Icon xuất hiện trong game cũng giống như các con số xuất hiện trong ma trận. Để các phần tử trong **matrix**nhận giá trị ngẫu nhiên, mình dùng lớp **Random** để tạo ra các số ngẫu nhiên. Ở trong phần code của mình, em có 21 icon , nên mình tạo ra một biến **imgCount = 21** và 1 biến **index** được chọn ngẫu nhiên trong các số từ 1 đến 21 với câu lệnh **int index = rand.nextInt(imgCount) + 1;**Đây là giá trị của các ô trong ma trận mình tạo ra, và đồng thời tương ứng là giá trị của các Icon dùng để so sánh với nhau.



Em tạo ra một ArrayList<Point>listPoint và thêm vào listPoint các Point với vị trí tương ứng với từng ô trong ma trận bằng 2 vòng for cùng 2 biến i,j với i chạy từ 0 đến row -1 và j chạy từ 0 đến col - 1. Khi đó, listPoint sẽ có số phần tử bằng với số phần tử của ma trận matrix.



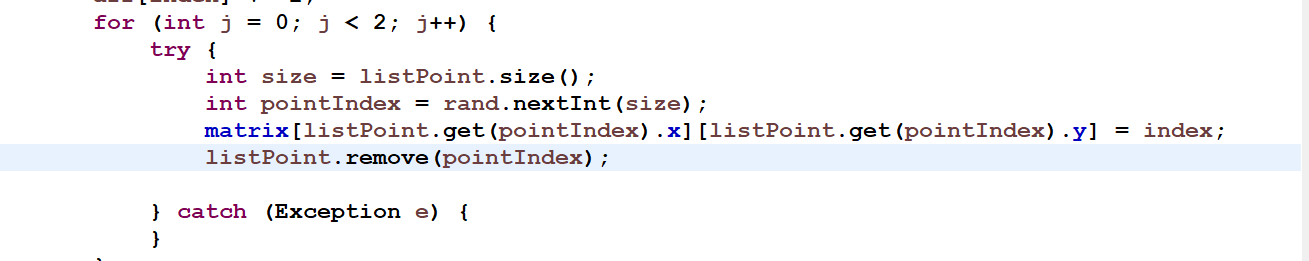
Sau đó, em sẽ dùng 1 biến **pointIndex**được lấy giá trị **random** theo giá trị **size()**của **listPoint**. Với mỗi **pointIndex**được lấy ra, chúng ta gán giá trị của **matrix[pointIndex.x][pointIndex.y] = index;**đồng thời remove giá trị **pointIndex**này ra khỏi **listPoint .**Việc remove này giúp chúng ta bỏ qua những ô đã được gán giá trị trong ma trận, tránh việc vị trí các ô được lấy qua hàm **random**trùng lặp nhau, qua đó tất cả các ô sẽ đều được nhận 1 giá trị ngẫu nhiên và không bỏ qua 1 ô nào cả.



Bằng cách này, em sẽ khởi tạo được vị trí ngẫu nhiên của các Icon game được hiển thị ra, và đồng thời đảm bảo tất cả các phần tử của **matrix**đều nhận những giá trị ngẫu nhiên.

Tuy nhiên, còn một vấn đề nữa với các Icon mà em phải giải quyết, đó là làm sao để số các Icon giống nhau luôn là chẵn. Vậy nếu tất cả đều là ngẫu nhiên, các Icon hiển thị ra bị lẻ, thì chẳng phải game sẽ không thể thắng

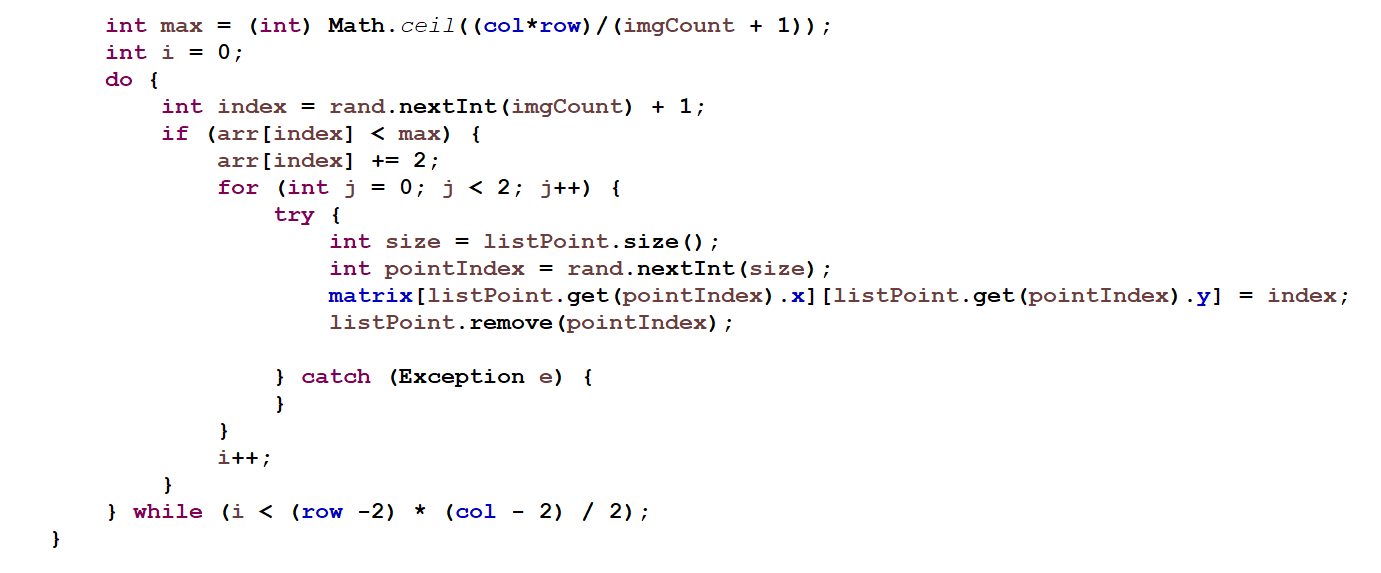
Để giải quyết điều này, mình đặt tất cả quá trình gán giá trị ngẫu nhiên cho các phần tử trong **matrix**trong 1 vòng **for**chạy 2 lần.



Khi đặt quá trình gán giá trị ngẫu nhiên cho các phần tử trong **matrix**trong vòng **for** ấy, mỗi khi 1 icon được tạo ra với giá trị tương ứng là **index** trong **matrix,** sẽ luôn có 1 Icon nữa có giá trị cũng bằng **index**ở 1 vị trí khác. Nói cách khác, khi đó, chúng ta sẽ không bao giờ tạo ra các Icon đơn lẻ, mà luôn tạo ra các cặp icon giống nhau. Vấn đề hoàn toàn được giải quyết.

Còn một vấn đề nữa đối với việc khởi tạo giá trị của các icon theo ma trận **matrix**, đó là vì tất cả đều ngẫu nhiên, cho nên sẽ có trường hợp các icon bị trùng lặp quá nhiều, có khi cả trò chơi có 100 ô thì đến 50 ô đều là 1 icon giống nhau thì…, Game sẽ nhàm chán lắm. Để giải quyết vấn đề này, mình quan tâm đến 1 giá trị là **row\*col/imgCount.**Tại sao lại là giá trị này? Chúng ta hãy tưởng tượng 1 ma trận với kích cỡ là **row\*col**được lấp đầy bằng tổn số Icon là **imgCount**, nghĩa là trung bình mỗi Icon sẽ xuất hiện là **row\*col/imgCount**lần. Kết hợp với việc số lần mà 1 Icon xuất hiện luôn phải là chẵn, mình rút ra số lần tối đa mà 1 Icon xuất hiện sẽ là giá trị chẵn giữa **row\*col/imgCount+1**hoặc **row\*col/imgCount+2.**

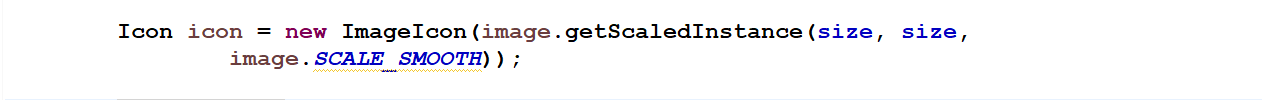
Đối với bài toán có 21 Icon và muốn tạo ra 1 bảng 8x8, nên số lần tối đa mà 1 Icon xuất hiện của mình là **8\*8/21+1=4  .** Sau khi tính được giá trị này, mình tạo ra 1 mảng **int []arr**với các giá trị khởi tạo đều là 0 và một biến **max=4.**Sau đó với mỗi giá trị **index**được tạo ra ngẫu nhiên, mình cộng thêm **2** vào giá trị của **arr[index]** (Bởi vì mỗi Index của mình đều đặt trong vòng **for**chạy 2 lần nên luôn có 2 Icon xuất hiện cùng lúc ứng với mỗi giá trị **index**)**.** Ta sẽ kiểm tra nếu **arr[Index] >= max**, Icon ứng với giá trị của **index**đó sẽ không được tiếp tục tạo ra nữa. Khi đó, mỗi giá trị **index**ứng với mỗi Icon được tạo ra luôn được giới hạn , như trường hợp của mình, một Icon sẽ xuất hiện tối đa 4 lần.

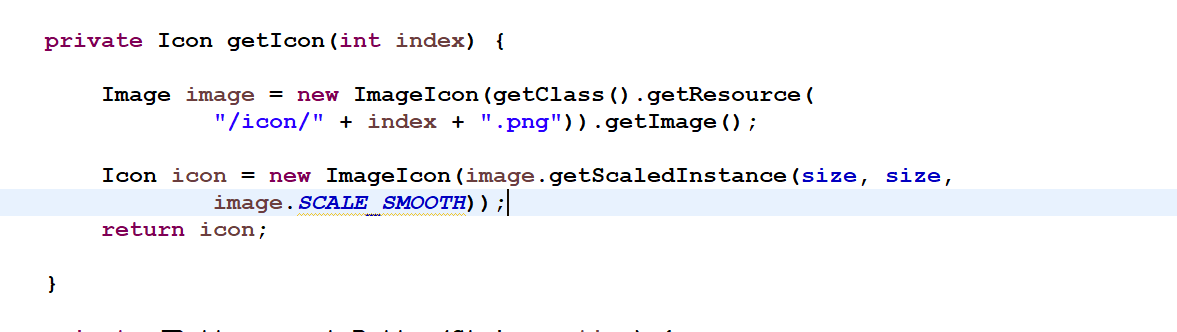


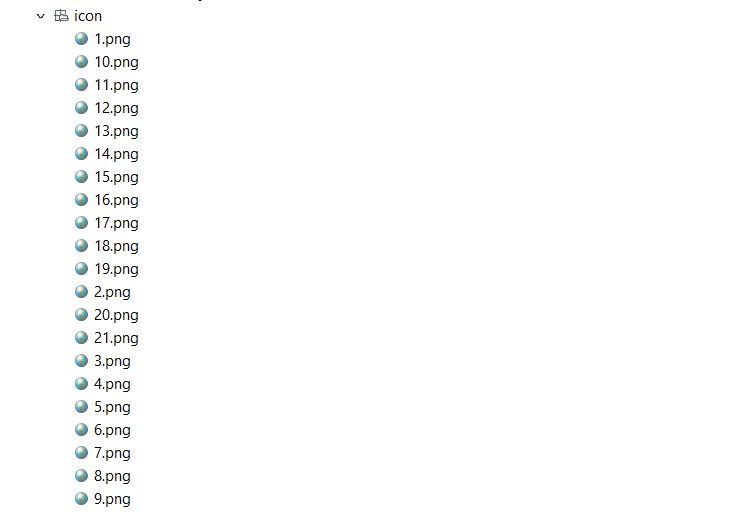
Tất cả phần khởi tạo ở trên em đặt trong 1 hàm while với biến **i**nhận giá trị ban đầu là 0. Với mỗi lần khởi tạo ra giá trị cho 2 ô mới của ma trận, câu lệnh **i++**sẽ được chạy, và việc khởi tạo giá trị của các ô sẽ kết thúc khi **i=row\*col/2**,cũng là khi chúng ta đã tạo đủ giá trị cho tất cả các ô trong ma trận .

### **3.1.*3*. Thêm các Icon vào khối để hiển thị**

Để có thể tạo ra được các icon, mình tạo 1 class mới là **PikachuBoard** extends từ **JPanel** trong packpage **View**. Mình viết 1 hàm **getIcon(int index)**để lấy ra các icon theo **index**(vị trí) của chúng. Ở đây, em dùng thư viện **Image**để lấy ra ảnh đã được lưu trong packpage **icon**, rồi sau đó dùng thư viện **Icon**để tạo ra các icon với ảnh được lấy ra cùng kích cỡ **width, height**có thể tùy chỉnh. Việc lấy các ảnh ra bằng **index:**

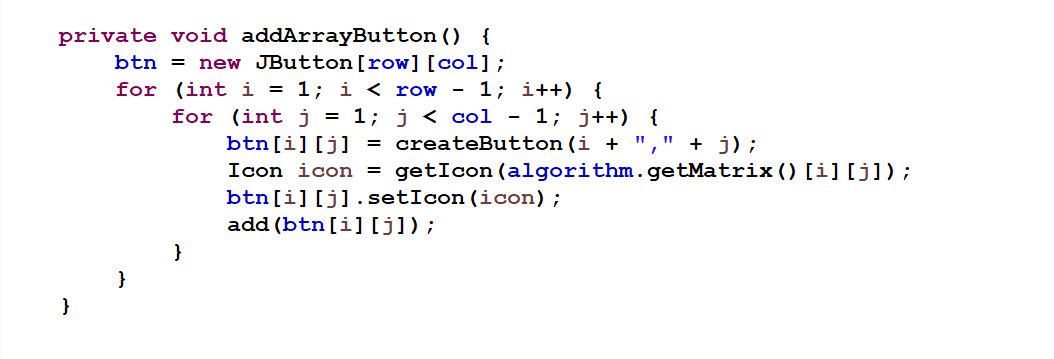




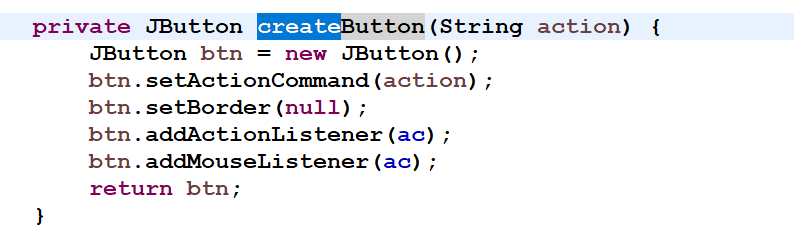


Ý nghĩa của câu lệnh là chúng ta sẽ lấy ra ảnh có tên là **index**trong packpage **icon,**đó là lý do mà chúng ta đặt tên các icon là các số từ 1 đến hết mà em đã trình bày ở trên

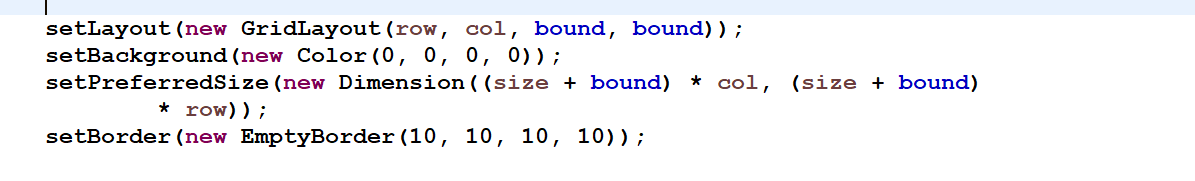
Sau khi lấy ra được các icon rồi, mình viết hàm **addArrayButton()** thực hiện việc tạo ra các **button** trong bảng, và **setIcon**cho các **button** được tạo ra. Chúng ta cần tạo ra đủ số button tương ứng với kích cỡ của bảng, và các icon cho mỗi **button**được lấy ra bằng hàm **getIcon(int index)**với giá trị **index**chính là giá trị tương ứng của ma trận tại vị trí của **button**đó. Bằng cách này, ma trận của chúng ta bây giờ chính thức là 1 game thu nhỏ của game Pikachu, giúp ta có thể xử lý rất nhiều thứ về thuật toán của game tại ma trận này.



Để hoàn thiện phần **PikachuBoard** hiển thị lên các icon của game,em bổ sung thêm 1 hàm **createButton(String action)** để tạo sự kiện cho các icon trong các **Button** mà em tạo ra.



Sét Layout và backgroup cho class PikachuBoard:



**setLayout(new GridLayout(row, col, bound, bound));** Set Layout cho khối JPanel được tạo ra bằng GridLayout (sắp xếp các component thành dạng bảng) với các thamsố hàng, cột và khoảng cách giữa các component theo chiều ngang và chiều dọc bằng **bound=2.**

**setBackground** Cài đặt màu nền cho khối JPanel.

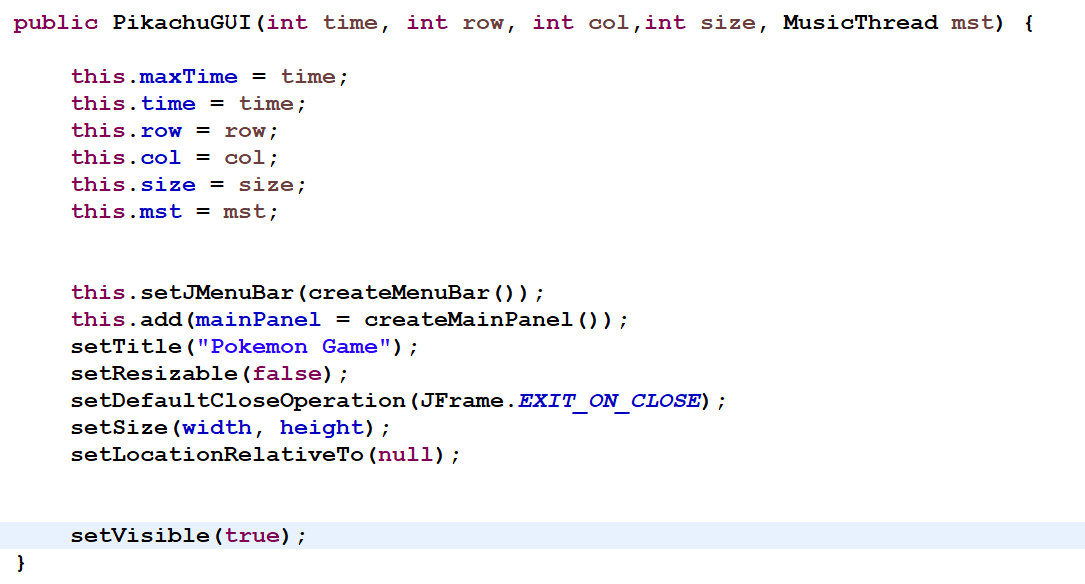
**setPreferredSize(new Dimension((size + bound) \* col, (size + bound)\* row));**Cài đặt kích cỡ của khối Panel.

**setBorder(new EmptyBorder(10, 10, 10, 10));**  Thêm viền bên ngoài của khối Panel.

### **3.1.*4*. Chạy chương trình và hiển thị các Icon**

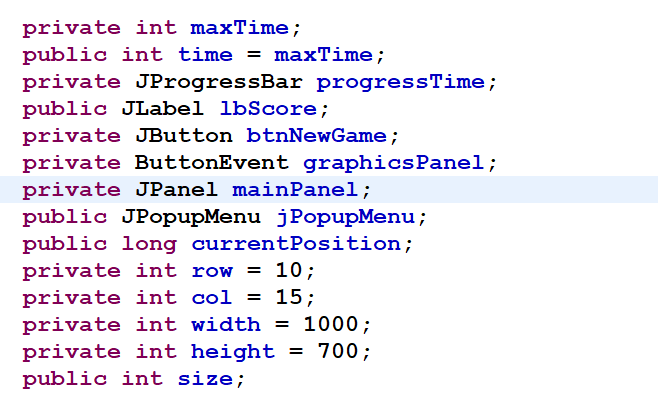
Sau khi hoàn thành 2 phần trên, mình tạo class **PikachuGUI** extends từ JFrame trong packpage **View** để làm màn hình chính của game. Trong class **PikachuGUI**này, em tạo 2 hàm **createmainPanel(),** và **createGraphicsPanel()** để thêm vào các **Button**chứa các icon của game.





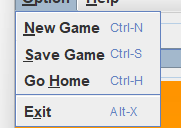
### **3.2 Hoàn thiện giao diện game** và **Thao tác xử lý cơ bản với các Icon**

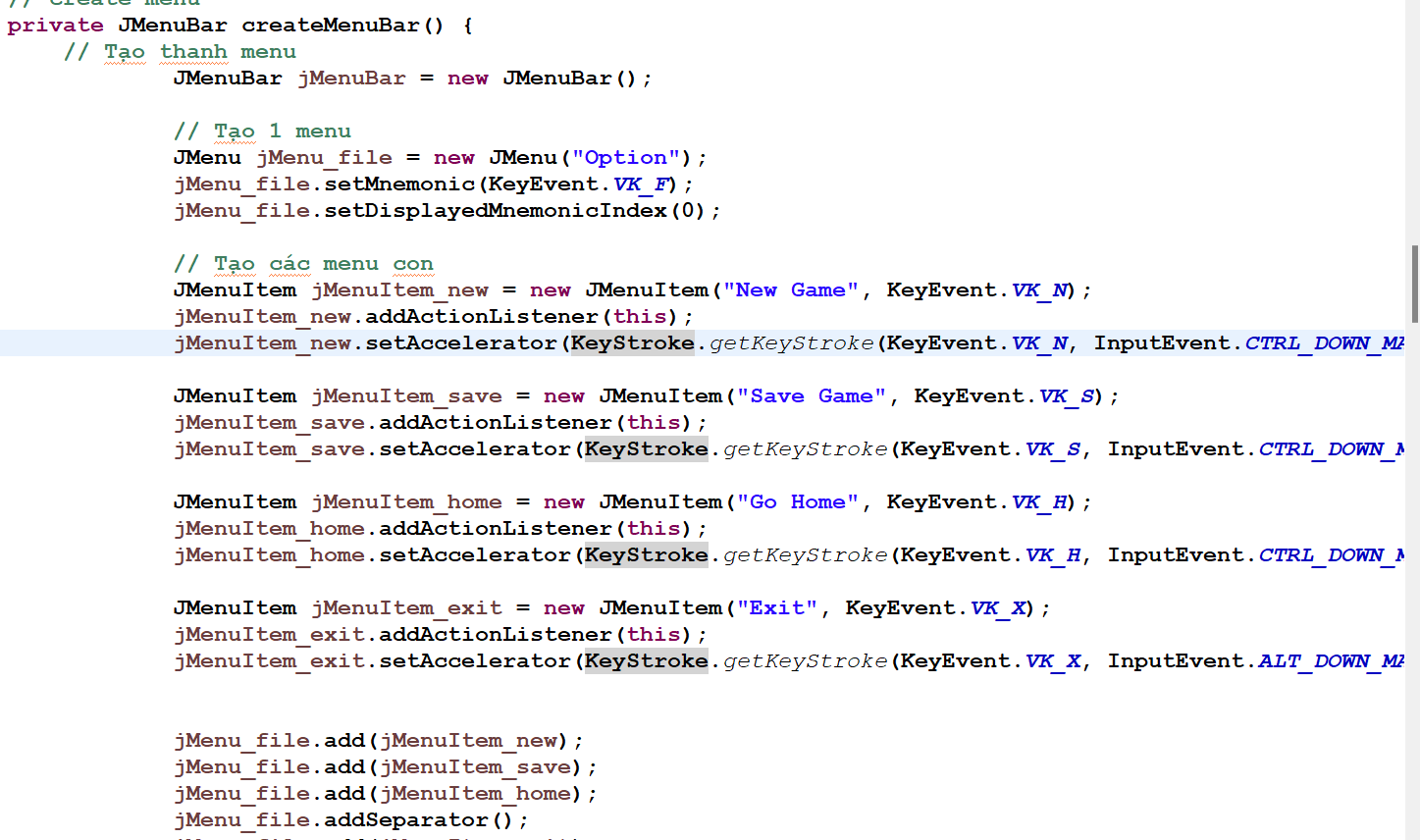
### **3.2.1 Hoàn thành giao diện**

Để hoàn thiện giao diện game, trong lớp **PikachuGUI(),**em tạo thêm một hàm **createControlPanel()** để khởi tạo các phần còn thiếu gồm: Thanh thời gian, ô tính điểm, và **button New game**; đồng thời, em cũng tạo ra 1 biến **MAX\_TIME**sẽ một số int là thời gian người dùng sẽ chọn ở phần home , 1 biến **time= maxTime**và các **Component** em dùng để khởi tạo. 

Em để 2 biến **time**và **lbScore**là public vì đây sẽ là 2 biến em sẽ dùng để xử lý và thay đổi giá trị tại các lớp khác.

Tiếp theo, em viết hàm **createmenuBar dùng để tạo thanh menu bar gồm các option NewGame, Home, Exit, và cho phép sử dụng phim tắt để chọn các chức năng nhanh chóng**





Và để hoàn thiện phần giao diện này, em hoàn thiện hàm **createControlPanel()**để khởi tạo các phần còn thiếu của giao diện.



Trong phương thức **createControlPanel()**, trên dòng đầu tiên **lbScore** được khởi tạo là một **JLabel** với nội dung là chuỗi "0". Tiếp theo, progressTime được khởi tạo là một JProgressBar với giá trị tối thiểu là 0 và giá trị tối đa là 100, và giá trị hiện tại được đặt là 100.

Tiếp theo, **panelLeft** được khởi tạo là một JPanel sử dụng **GridLayout** với 1 hàng và 2 cột và khoảng cách 5 đơn vị giữa các thành phần. Một JLabel mới với nội dung là chuỗi "Score:" được thêm vào **panelLeft** với vị trí đầu tiên và **lbScore** được thêm vào với vị trí thứ hai.

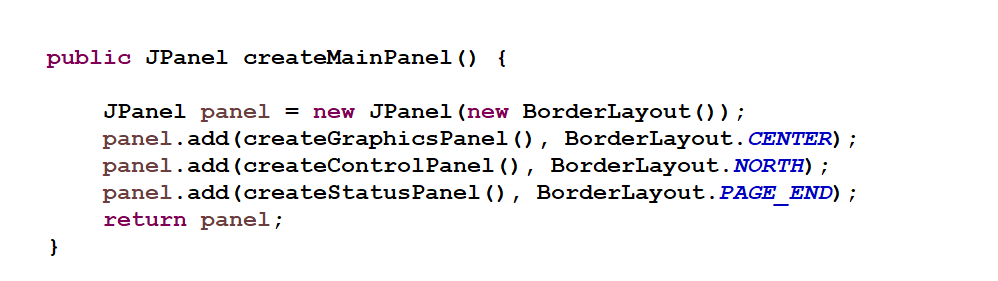
panelCenter cũng được khởi tạo là một **JPanel** sử dụng **GridLayout** với 1 hàng và 2 cột và khoảng cách 5 đơn vị giữa các thành phần. progressTime được thêm vào panelCenter.

**panelScoreAndTime** được khởi tạo là một JPanel sử dụng BorderLayout với khoảng cách 10 đơn vị giữa các thành phần. panelLeft được thêm vào với vị trí phía tây (**WEST**) của **panelScoreAndTime**, trong khi panelCenter được thêm vào với vị trí trung tâm (**CENTER**) của nó.

**panelControl** được khởi tạo là một JPanel sử dụng **BorderLayout** với khoảng cách 10 đơn vị giữa các thành phần. Một đường viền rỗng (**EmptyBorder**) được đặt cho **panelControl**, với các kích thước tương ứng là 10 đơn vị về phía bắc (top), 3 đơn vị về phía tây (left), 5 đơn vị về phía nam (**bottom**) và 3 đơn vị về phía đông (**right**). panelScoreAndTime được thêm vào với vị trí trung tâm (**CENTER**) của panelControl.

Cuối cùng, panel được khởi tạo là một JPanel sử dụng **BorderLayout**, với một tiêu đề (**TitledBorder**) là chuỗi "**Status**". **panelControl** được thêm vào với vị trí phía đầu (**PAGE\_START**) của panel.

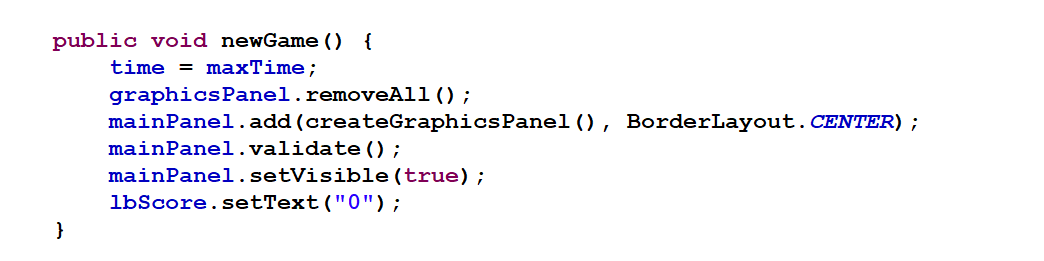
Sau khi hoàn thành hàm **createControlPanel() ,** chúng ta sẽ khởi tạo nó tại hàm **createmainPanel()**, phần controlPanel(gồm thanh menu và status ở trên cùng) , các ô ma trân ở trung tâm



Khi đó, màn hình game sẽ thay đổi thành như hình ảnh sau:



Tiếp theo xây dựng nút **New game.**Để thêm chức năng cho nút **New game**, mình viết hàm **newGame()**. Mỗi khi gọi đến hàm này, tất cả các dữ liệu về **time, score,**hay các Icon đều sẽ được cài lại mặc định ban đầu.



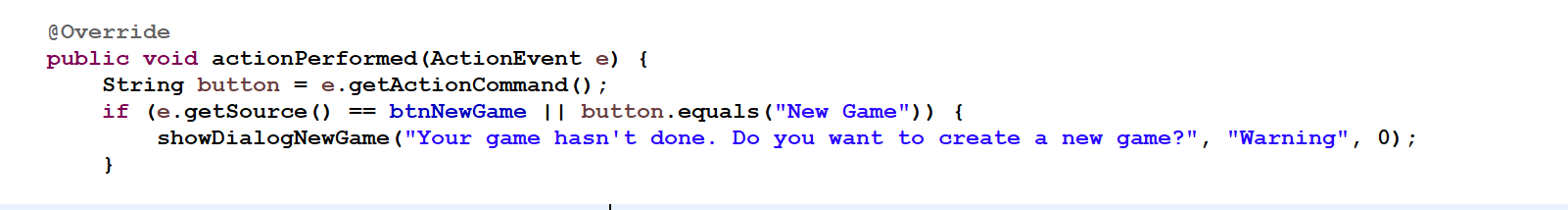
Hàm **newGame()**của em sẽ trả lại **time**bằng giá trị **MAX\_TIME**(Thời gian ban đầu), **score**trả về giá trị 0, và **graphicsPanel**chính là ô chứa các Icon sẽ **removeAll()**(gỡ hết các Icon đi) và thêm lại 1 bảng các Icon mới. **validate()**dùng để set thuộc tính hợp lệ cho **Component**được tạo ra.

Sau đó, em tạo ra hàm **showDialogNewGame().**Đây là một hàm quan trọng, vì nó sẽ được sử dụng khá nhiều, khi bạn thắng, khi bạn thua, hoặc khi bạn ấn vào nút **New Game**.



Hàm **showDialogNewGame**() này sẽ hiển thị ra 1 message bằng **JOptionPane.YES\_NO\_OPTION** với 3 giá trị truyền vào lần lượt là: message - thông báo hiển thị ra của hộp thoại JOptionPane, **tittle** là tiêu đề của hộp thoại, cùng biến t để kiểm tra nguồn xuất hiện của hộp thoại. Nguồn xuất hiện ở đây mình phân biệt từ 2 nguồn, hoặc là sau khi thắng game hoặc thua game, khi đó giá trị truyền vào của t sẽ là 1; hoặc là sau khi ấn vào nút **New game** khi trò chơi vẫn đang chạy, khi đó giá trị truyền vào của **t** sẽ là 0. Với t=1, nghĩa là sau khi bạn thắng game hoặc thua game, nếu bạn không muốn tiếp tục trò chơi, chọn No thì game sẽ thoát bằng hàm (0);. Còn nếu t=0, là khi game vẫn đang diễn ra, nếu bạn chọn No thì sẽ không xảy ra gì cả, hộp thoại biến mất và game sẽ được tiếp tục với trò chơi hiện tại của bạn. Trong cả 2 trường hợp, nếu chọn **Yes** , hàm **newGame**() sẽ được chạy, và chúng ta sẽ có 1 trò chơi mới.

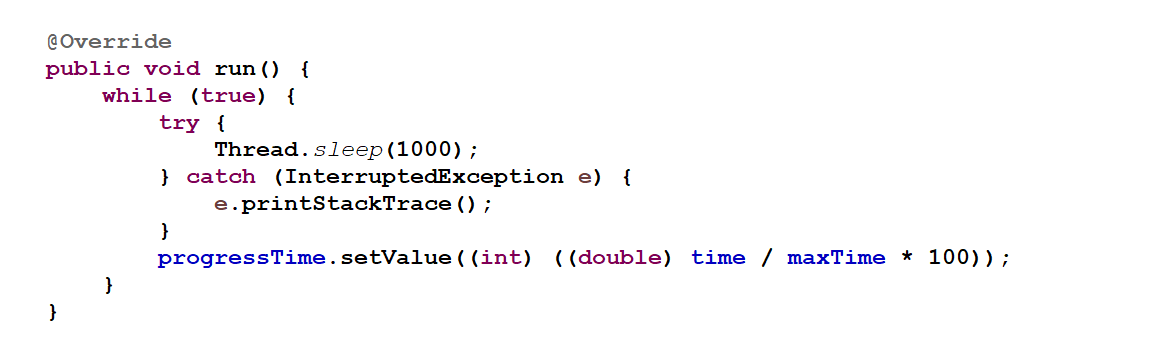
Cuối cùng, việc còn lại là thêm **action** cho nút **New Game** của lớp **PikachuGUI**được khởi tạo từ lớp **ControllerOption** thuộc package controller.

****

Em dùng **e.getSource()** để kiểm tra nếu nút **New Game** (có tên là **btnNewGame**) của mình được click vào, khi đó mình sẽ gọi đến hàm **showDialogNewGame**, hiển thị ra 1 hộp thoại nhắc nhở với các tham số truyền vào như trên. Với tham số **t=0**, nếu chọn **No**, game của mình sẽ không được tạo mới, và vẫn tiếp tục bình thường.

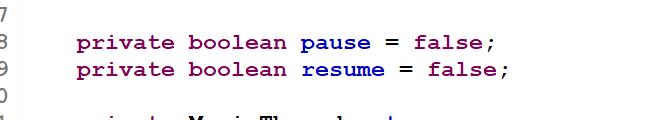
Sau khi xử lý action thành công cho nút **New Game**, chúng ta sẽ tiếp tục xử lý về thời gian của game.  Như trong giao diện game của chúng ta, **time**được hiển thị dưới dạng một thanh **JProgressBar**. Để thay đổi thời gian, chúng ta cần liên tục thay đổi giá trị của thanh **JProgressBar**này. Ở trên, chúng ta đã can thiệp vào nút **New Game**bằng phần **override**của phương thức **actionPerformed**, đối với phần xử lý **time**này, em sẽ can thiệp vào phần **override** còn lại, đó là phương thức **run().**

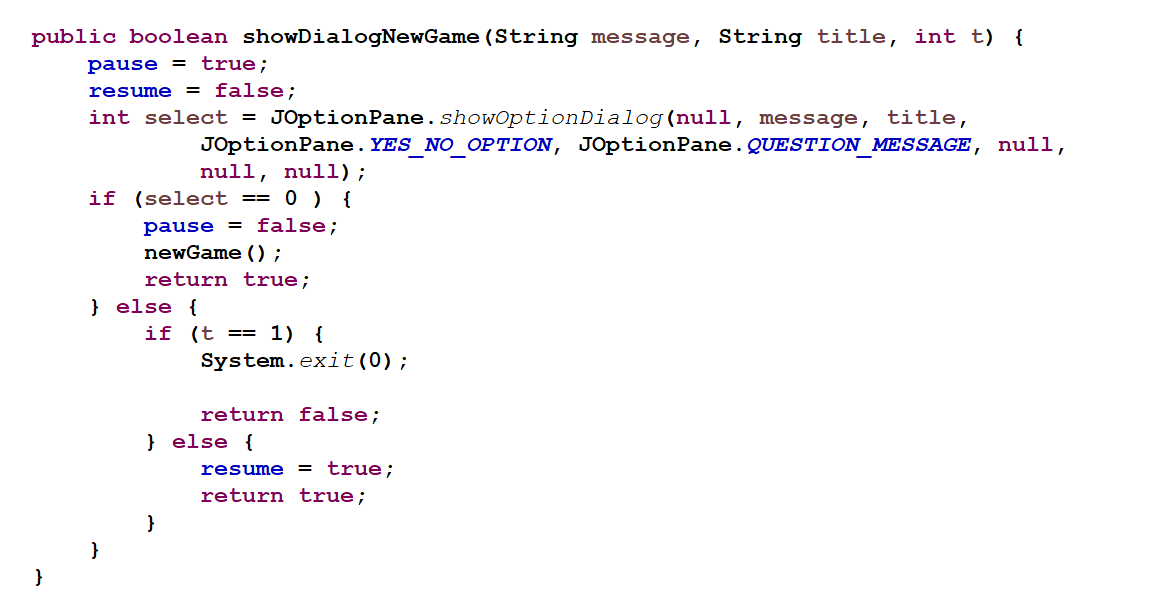
Ý tưởng của em rất đơn giản, đó là liên tục cập nhập giá trị của biến **progressTime** được khởi tạo từ thanh **JProgressBar**ở trên bằng tỉ lệ của sự thay đổi của biến **time** với thời gian tối đa của game (**maxTime**) và nó sẽ được cập nhập mỗi giây (tương ứng với 1000 ms). Theo dõi đoạn code của mình để rõ hơn.

****

Giá trị tối đa của **progressTime**là 100 (Tương ứng với 100%).Em muốn game của mình có thời gian là giá trị của **MAX\_TIME**, đồng thời cũng là giá trị khởi tạo của **time**. Việc thay đổi giá trị của **progressTime**dựa vào tỉ lệ của 2 biến **time**và **MAX\_TIME**cần đặt ép kiểu **(double)**ở trước vì **time**và **MAX\_TIME** đều là kiểu **int**, nên nếu không ép kiểu, việc thời gian (**time**) bị giảm đi, giá trị của **time/MAX\_TIME**sẽ trở thành 0, khi đó việc nhân với 100 ở sau sẽ trở nên vô nghĩa. Và dĩ nhiên, sau khi ép kiểu, để có thể **setValue()** cho **progressTime**, chúng ta cần ép kiểu 1 lần nữa giá trị của **(double) time/MAX\_TIME\*100** về kiểu  **int**.

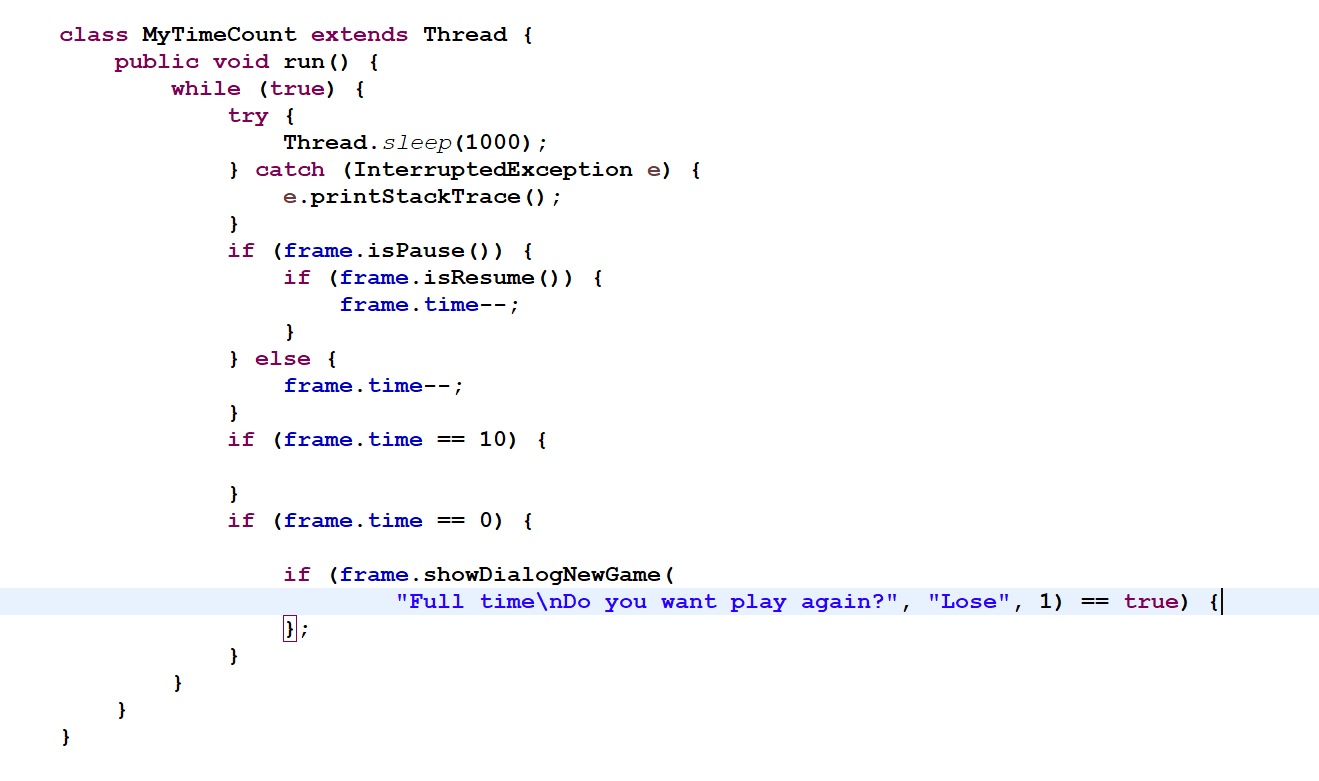
Để xử lý thời gian của game, mình muốn mỗi khi Popup được gọi ra từ hàm **showDialogNewGame**() hiển thị, thời gian của game sẽ dừng lại, và khi ta muốn tiếp tục trò chơi của mình, thì thời gian của game mới tiếp tục. Để làm được điều đó, mình khai báo thêm 2 biến pause và resume với giá trị khởi tạo là false và thực hiện việc thay đổi các giá trị này trong hàm **showDialogNewGame**(). Để có thể sử dụng 2 biến pause và resume tại các lớp khác để xử lý

****

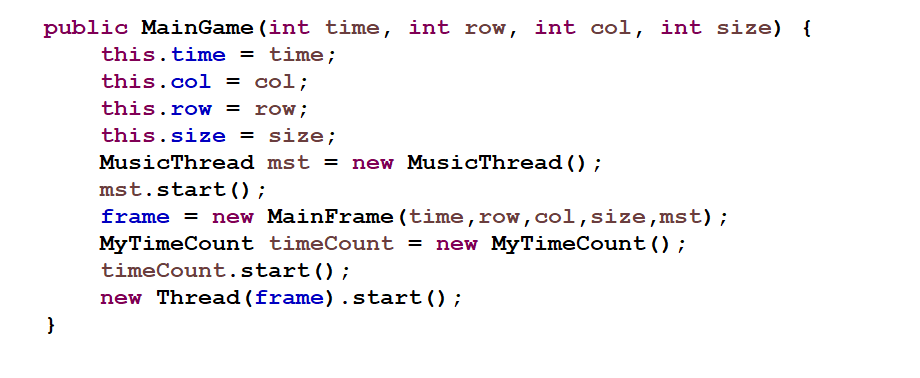
****

Việc thay đổi trong hàm **showDialogNewGame()** đơn giản là mỗi khi gọi đến hàm này, để thời gian dừng lại, mình thay đổi biến **pause** về giá trị **true**và **resume**về giá trị **false**. Nếu hàm **showDialogNewGame()** được gọi từ nút **New Game**, việc chọn **No** để quay lại game sẽ tiếp tục thời gian đã bị dừng lại của mình. Để làm được điều đó, mình thay đổi giá trị **resume**thành **true**, khi đó, thời gian sẽ tiếp tục; Khi chọn **Yes**để tạo game mới, biến **pause**của mình khi này đang có giá trị **true**ngay khi hộp thoại xuất hiện, vậy nên để sau khi khởi tạo lại, thời gian của game chạy bình thường, mình đặt lại giá trị của biến **pause**về **false.**

Tiếp theo, để hoàn thiện việc xử lý thời gian của game, cũng như để thay đổi giá trị của **time**, tại class MainGame, em sẽ tạo **class Time** extends từ lớp **Thread.**

****

Class**Time** chính là phần xử lý chính của biến **time**, cũng là phần xử lý chính đối với thời gian của game. Trong class**Time** này, mình cập nhập liên tục giá trị biến **time** của lớp **PikachuGUI** mỗi giây. Trong khi cập nhập, kiểm tra biến **pause**của lớp **PikachuGUI** bằng hàm **isPause()**để kiểm tra xem game có đang tạm dừng không. Nếu game tạm dừng, **time** sẽ không thay đổi cho đến khi mình kiểm tra game đã được tiếp tục bằng hàm **isResume().** Tại phần cập nhập này, **time** sẽ giảm đi 1 mỗi giây. Nếu **time=0**, mình sẽ gọi đến hàm **showDialogNewGame()** của lớp **PikachuGUI**để thông báo cho người chơi rằng họ đã thua vì thời gian đã hết, và hỏi xem có muốn chơi lại không. Việc xử lý của hàm **showDialogNewGame** này mình đã nói ở trên. Và cuối cùng, khởi tạo **class Time**để thay đổi **time**và **Thread**của lớp **PikachuGUI** để thay đổi giá trị của **progressBar**trong phần khởi tạo của lớp **MainGame()**, sau đó để hai **Thread**này chạy ngay khi game được bắt đầu.

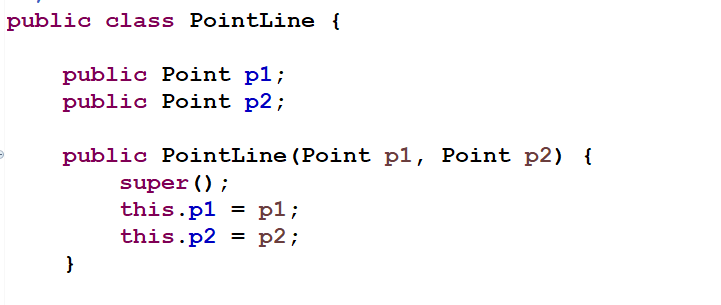
****

### **3.2.2. Thao tác xử lý cơ bản với các Icon**

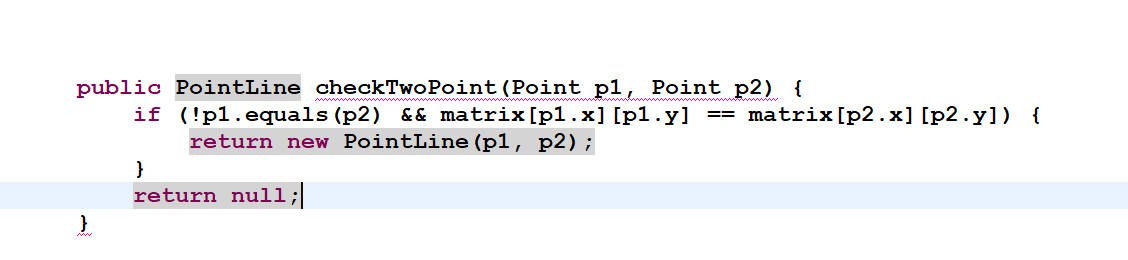
Các Icon được khởi tạo trên game đồng thời với một ma trận **matrix** với các giá trị của từng ô trong **matrix**tương ứng là các giá trị của các Icon. Các Icon sẽ được xử lý dựa trên ma trận này, và phần xử lý ấy nói đơn giản sẽ gồm 2 bước:

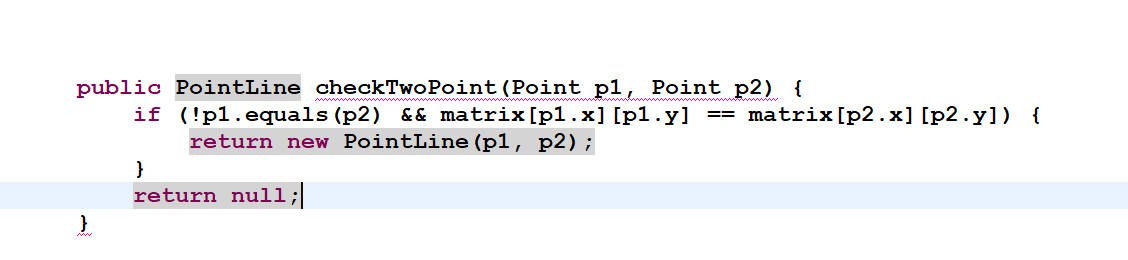
* **Bước 1:**Tìm 2 Icon giống nhau (Cũng là tìm 2 Icon có giá trị bằng nhau). Chú ý rằng các Icon được lấy ra theo giá trị **index**, đó là giá trị của ô tại vị trí tương ứng của Icon trong ma trận **matrix.**
* **Bước 2:** Kiểm tra xem 2 Icon có nằm trên vị trí có thể kết nối được với nhau hay không.

Trong packgage model tạo một lớp PointLine, mỗi Icon mình coi là một **Point** với 2 tọa độ **x,y**, class **PointLine** này được tạo ra để định nghĩa việc kết nối 2 Icon với nhau. Cụ thể, một **PointLine** sẽ gồm hai **Point p1, p2**, tương ứng là hai Icon trong game.



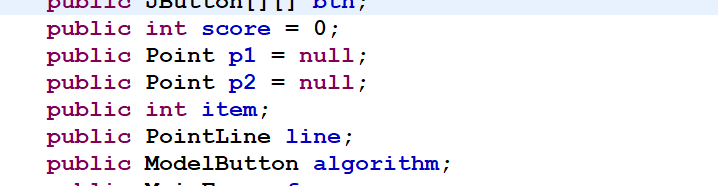
Em viết thêm 1 hàm **checkTwoPoint(Point p1, Point p2)**  trong class ModelButton với 2 tham số nhận vào là hai **Point p1, p2** để kiểm tra giá trị của 2 **Point** này. Bởi vì mục đích của em hiện tại đơn giản chỉ là kiểm tra 2 Icon giống nhau, nên mình chỉ cần kiểm tra 2 điều kiện, đó là: **p1** và **p2** không được trùng nhau (Nghĩa là không cùng chọn vào 1 Icon), và giá trị của 2 Icon tương ứng tại ma trận là bằng nhau.



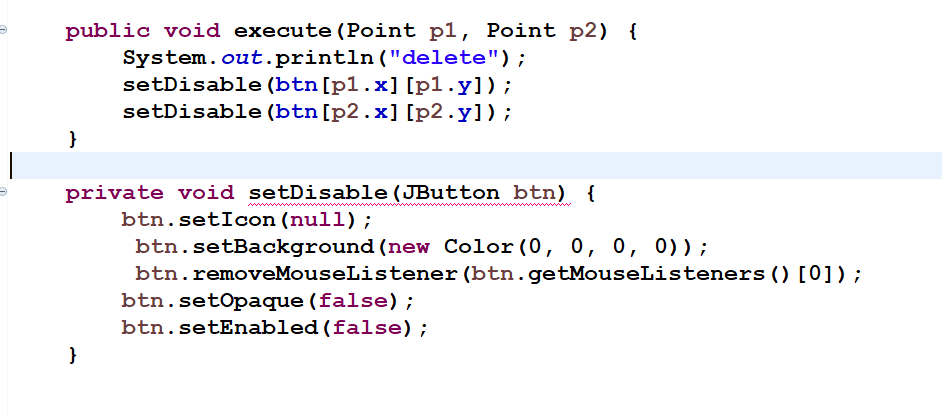


Khi đó, 2 Icon được kiểm tra nếu thỏa mãn điều kiện trên sẽ trả về 1 **PointLine** mới của 2 **Point** ứng với 2 Icon đó, còn nếu không thỏa mãn, sẽ trả về **null**.

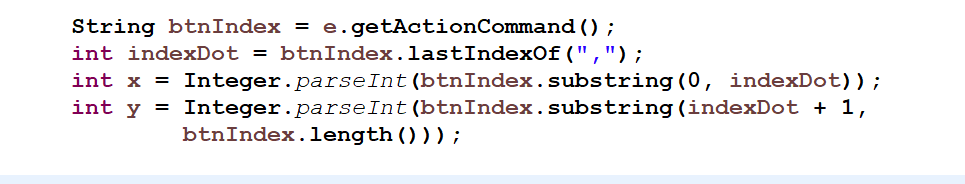
Tiếp theo xóa đi 2 Icon giống nhau, mình tiếp tục viết thêm vào class **PikachuBoard** của packpage **View.** Đầu tiên,em tạo ra 2 biến **Point** **p1, p2** với giá trị khởi tạo là **null**, 1 biến **PointLine** **line**, 1 biến **int score=0** để tính điểm và 1 biến **int** **item** với giá trị khởi tạo là **row\*col/2.** Bởi vì các Icon được khởi tạo theo cặp, và khi biến mất, chúng cũng biến mất theo cặp, nên em sẽ xử lý các Icon này theo các cặp Icon với tổng số cặp Icon chính là **item**. Ngắn gọn hơn, **item** chính là **tổng số** **cặp Icon**



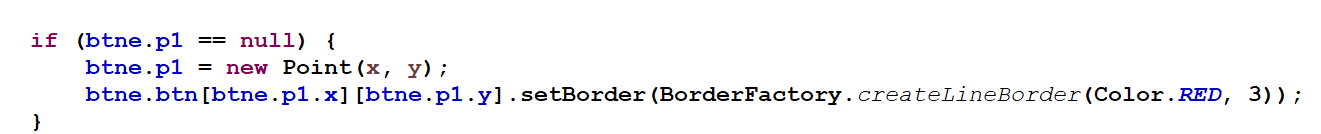
Tiếp theo, mình tạo thêm 2 hàm **excute()** và **setDisable()** trong lớp PikachuBoard để thực hiện việc xóa đi các Icon. Trong đó, tại hàm **setDisable()**, các Icon tương ứng trên các **Button** sẽ bị xóa đi, các **Button**ấy sẽ bị vô hiệu hóa và đồng thời **background** sẽ trả về **background**của cả khối **Panel**được tạo ra. Còn hàm **Excute()**sẽ nhận 2 tham số truyền vào là hai **Point p1, p2**, để **setDisable()** cho các nút tương ứng với 2 **Point**này.



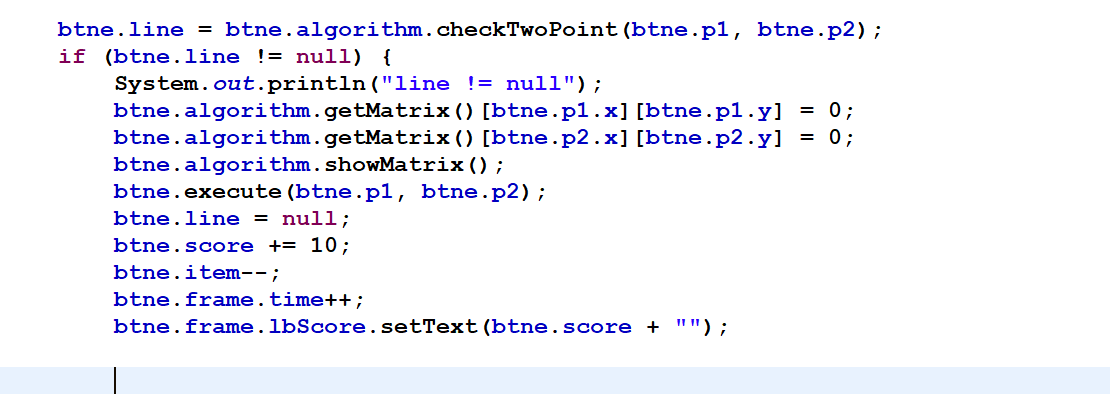
Em sẽ tạo Class **ControllerButton** trong package **Controller** và implement **ActionListner** và Overiide lại phương thức **actionPerformed(ActionEvent e)**để xử lý **Event**của các **Button** tương ứng với các Icon được tạo ra. Khi chúng ta click vào 1 Icon, hàm **e.getActionCommand()** sẽ trả về tọa độ của Icon mà chúng ta vừa click vào. Mình lưu tọa độ ấy vào 1 biến String là **btnIndex**, sau đó mình tách tọa độ ấy ra qua dấu **","** rồi lần lượt lưu vào 2 biến **x**và **y.**



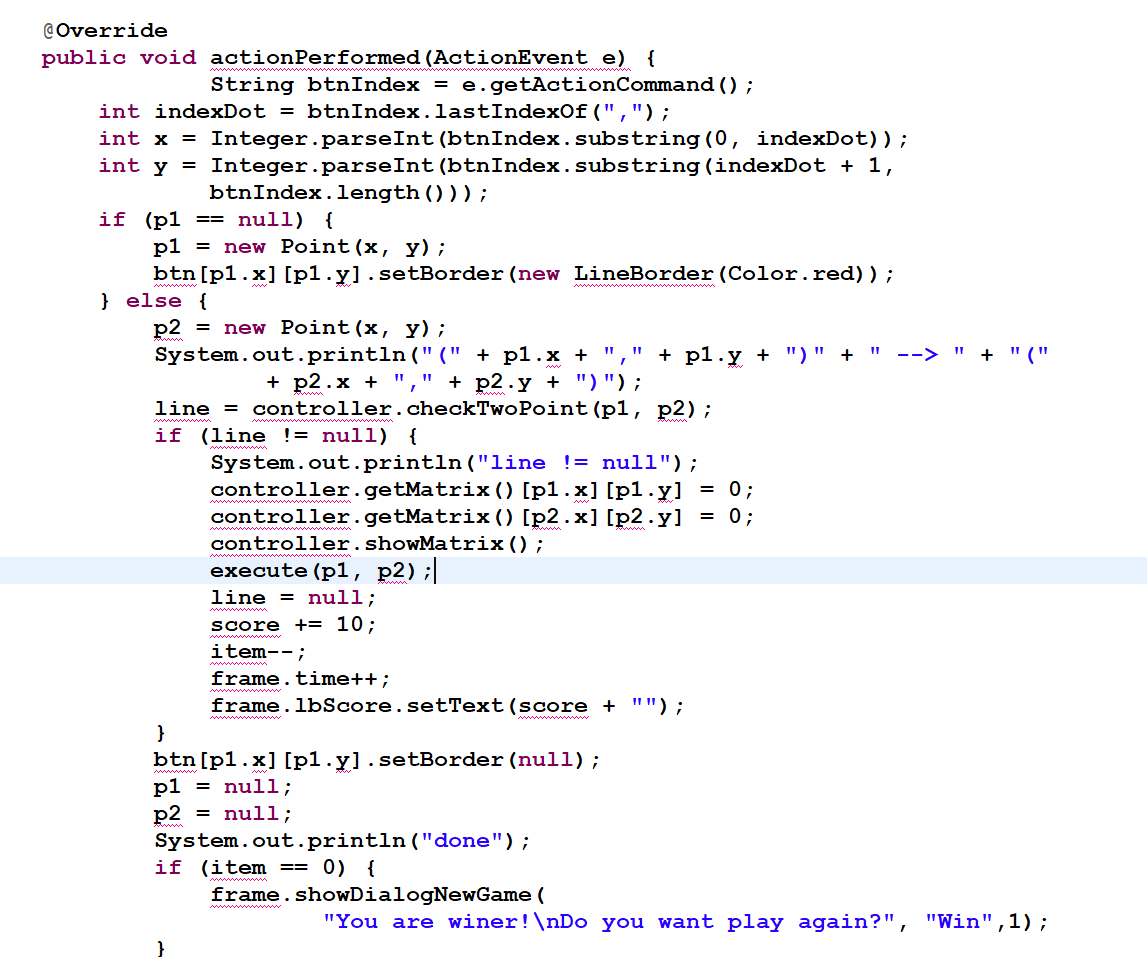
Tiếp theo, kiểm tra biến **p1,** em đang xét các Icon theo cặp, nghĩa là chúng ta sẽ xét lần lượt 2 Icon được chọn. Nếu **p1=null**, nghĩa là chưa có Icon nào được chọn, Icon mình đang click vào là Icon đầu tiên của cặp Icon mình đang xét. Khi đó, mình gán giá trị cho **p1**bằng một **Point** mới với tọa độ **x,y** vừa lấy từ **Button**được chọn và **p1**chính là Icon với tọa độ **x,y**đó. Đồng thời, mình cũng tạo hiệu ứng để thể hiện Icon đang được chọn bằng cách **setBorder()**cho **Button** tương ứng của Icon ấy.



Khi đã có 1 Icon được chọn rồi, và Icon ở **Button** mà mình đang click vào, chính là Icon còn lại trong cặp Icon mà mình đang xét. Khi đó, mình sẽ tạo ra 1 **Point** mới với tọa độ **x,y** lấy từ **Button**được chọn và gán giá trị đó vào **p2**. Mình gọi hàm **checkTwoPoint()** từ lớp **Controller** để kiểm tra 2 điểm **p1, p2**và gán giá trị trả về của hàm **checkTwoPoint(p1, p2)**  vào biến **line** được khởi tạo ở trên. Nếu **line** không bằng **null**, nghĩa là 2 điểm này thỏa mãn điều kiện kiểm tra, nói cách khác, 2 Icon được chọn là 2 Icon giống nhau. Khi đó, mình sẽ gọi hàm **excute()**với 2 **Button** tương ứng với 2 điểm **p1, p2**; đồng thời, mình cũng đặt giá trị của những phần tử trong ma trận tương ứng với 2 điểm **p1, p2** này về 0. Trong ma trận của mình, 0 sẽ thể hiện cho các ô trống, đồng thời cũng là các Icon bị mất đi. Đây là 1 phần rất quan trọng để xử lý thuật toán về vị trí của các Icon mà mình sẽ giới thiệu với các bạn ở phần sau. Sau khi xóa thành công 2 Icon được chọn, **line**sẽ được trả về **null**, điểm số của chúng ta sẽ được tăng thêm 10 điểm, số cặp Icon là **item** sẽ trừ đi 1, thời gian **time** trong lớp **PikachuGUI**sẽ được cộng thêm 1 để là phần thưởng cùng với **score**và mình sẽ hiển thị lại số điểm trên **lbScore** của lớp **PikachuGUI** chính là giá trị của biến **score**của mình.



Sau khi kiểm tra 2 **Button** ứng với 2 điểm **p1, p2,**mình sẽ đặt lại **Border** của **Button** ứng với điểm **p1**mà mình đã thêm **Border** ở trên về **null.** Nếu trường hợp 2 Icon được chọn không giống nhau, hoặc ta chọn cùng 1 Icon hai lần, thì ta cần có 1 tín hiệu rằng mình đã chọn sai, việc kiểm tra 2 Icon được chọn đã kết thúc và tiếp theo sẽ là một cặp 2 Icon mới, đúng không? Đây chính là tín hiệu đó. Đồng thời, việc này sẽ giúp người chơi luôn biết mình đang chọn Icon nào, để tìm Icon tương tự. Sau đó, các điểm **p1, p2** sẽ được trả về **null**để tiếp tục với các cặp Icon mới. Cuối cùng, mình kiểm tra số lượng cặp Icon còn lại bằng biến **item**. Nếu **item=0**, nghĩa là không còn Icon nào cả, mình sẽ gọi hàm **showDialogNewGame()** của lớp **PikachuGUI** để thông báo chiến thắng đến người chơi



### **3.3 Thuật toán tìm kiếm giữa 2 Icon giống nhau**

.  

Ở bên trái là 1 trường hợp thỏa mãn khi 2 Icon đường nối với nhau bằng 3 đoạn thẳng, còn ở bên phải, với việc cần ít nhất 4 đoạn thẳng để nối giữa 2 icon này, nên dù có giống nhau chúng vẫn không thỏa mãn điều kiện về đường đi. Nói cách khác, đường đi thỏa mãn 2 icon ở ảnh bên phải đã bị CHẶN’

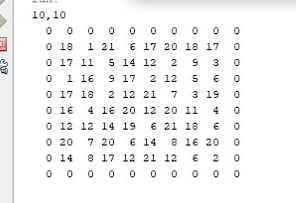
### **3.3.1. Ý tưởng**

 Trong 2 bức ảnh ấy, em đã vẽ đường đi cụ thể của 2 icon giống nhau, và nó sẽ men theo vị trí của các ô trống để tìm đường đi bằng các đoạn thẳng vuông góc với nhau. Để làm được điều ấy, mình cần xác định được vị trí của các ô trống. Như trong những phần trước, chúng ta đã dùng một ma trận **matrix** để thể hiện cho bảng của các icon trong game. Tại ma trận, mỗi icon sẽ có 1 số tương ứng với giá trị của icon đó; và mỗi khi 2 icon bị biến mất, giá trị của cả 2 icon đều được trả về 0. Nói cách khác, số 0 chính là giá trị của các ô trống không có icon

Tiếp tục quan sát 2 bức ảnh trên, các đoạn thẳng nối 2 icon còn vượt ra ngoài cả bảng, vậy nghĩa là vượt ra cả ma trận

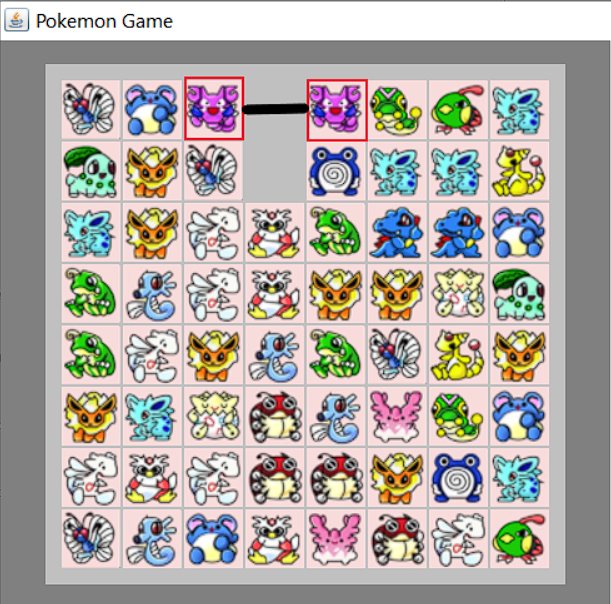


Để tạo ra các đoạn nối thỏa mãn điều đó, ý tưởng của mình là bọc ma trận chính của mình trong 1 vòng gồm các số 0, mình gọi đó là các lề. Việc tạo ra 4 lề tương ứng với tất cả các ô trống với giá trị 0 sẽ bao quanh bảng của mình bằng các vị trí trống, từ đó chúng ta có thể tìm được các đường đi thỏa mãn.

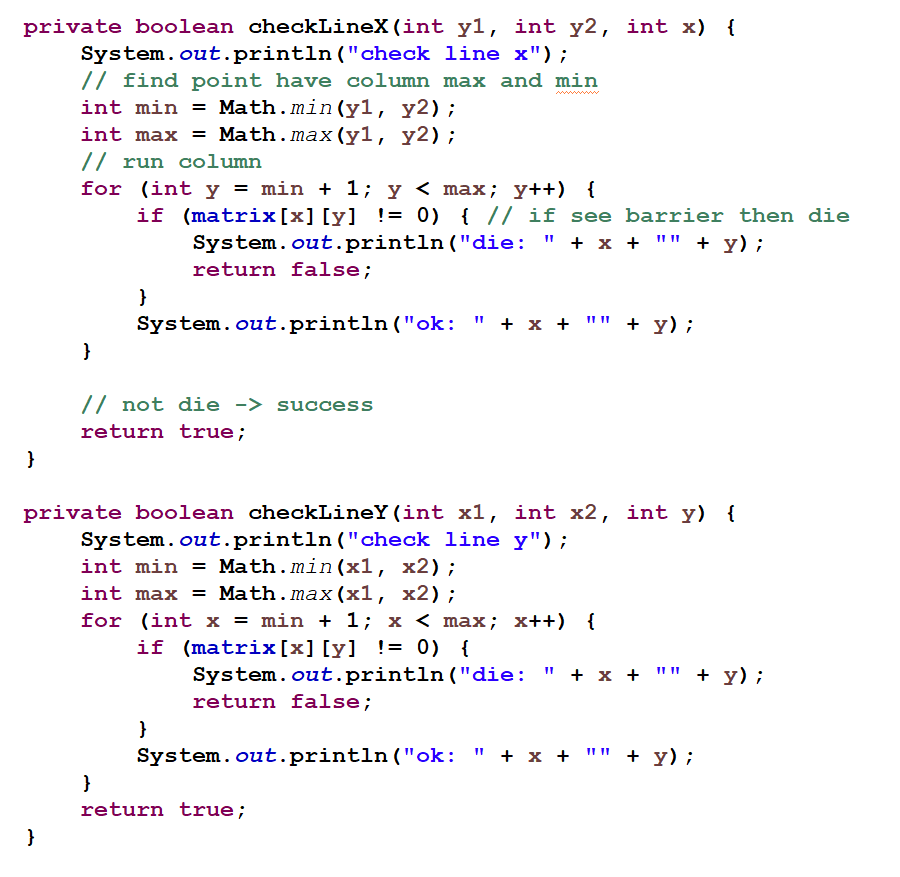


### **Thuật toán tìm kiếm đường đi giữa 2 icon giống nhau**

* **Nằm trên cùng một cạnh:**
* Đây là trường hợp đơn giản và dễ xét nhất. Mình chia làm 2 trường hợp: Nằm trên cùng 1 hàng ngang (**x1 = x2**) hoặc nằm trên cùng 1 hàng dọc (**y1 = y2)**. Xét trường hợp 2 icon nằm trên cùng 1 hàng ngang (**x1 = x2**), mình xét 2 tọa độ còn lại là **y1**và **y2** để tìm ra điểm có hoành độ nhỏ hơn (giả sử trả về **y1 < y2**), rồi xét liên tục các ô theo hàng ngang **x1**từ vị trí **y1** đến **y2**, nếu các ô được xét đều là ô trống (có giá trị là 0), thì đây là 2 icon nằm ở vị trí thỏa mãn. Đường nối tạo ra sẽ có dạng như sau.



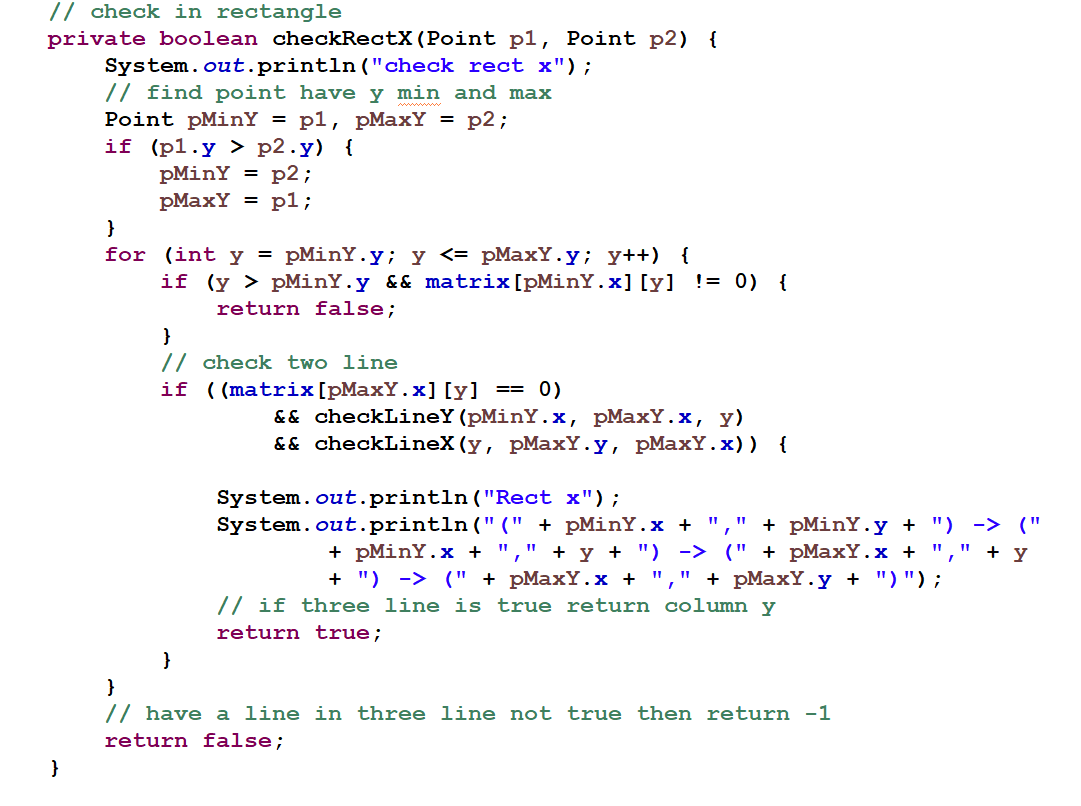
* Tương tự với trường hợp 2 icon nằm trên cùng 1 hàng dọc. Và để thực hiện việc này, mình viết 2 hàm mới là **checkLineX()**và **checkLineY()** trong **class ModelButton**, tương ứng với 2 trường hợp 2 icon nằm trên cùng 1 hàng ngang, và nằm trên cùng 1 hàng dọc.



* **. Được nối bằng tối đa 3 đoạn thẳng trong phạm vi hình chữ nhật hình thành từ tọa độ của 2 icon:**
* Xét 2 điểm chỉ trong phạm vi được bọc bởi giới hạn từ 2 tọa độ của 2 icon ứng với 2 đầu của hình chữ nhật. Với trường hợp này, mình cũng chia làm 2 trường hợp nhỏ:
* Đó là xét **theo chiều ngang** và **chiều dọc.**
* trường hợp này hoàn toàn tương tự nhau về cách xét, chỉ là khác về hướng đi. Mục đích của việc chia làm 2 trường hợp là để mình có thể bao quát toàn bộ các con đường với các hướng đi khác nhau. Giống như ở 1 ngã tư, chúng ta có 4 hướng đi, và muốn biết hướng đi nào dẫn đến đích, chúng ta cần bao quát được con đường từ cả 4 hướng.
* Cùng nhau xét 2 điểm **p1, p2**với trường hợp xét theo chiều ngang trước nhé. Với việc bắt đầu tìm hướng đi theo chiều ngang, mình lấy ra 2 hoành độ của 2 điểm là **y1**và **y2**. Sau đó, mình so sánh 2 hoành độ này, tìm ra hoành độ nhỏ hơn, và bắt đầu tịnh tiến từ ô có hoành độ nhỏ hơn sang phía ô có hoành độ lớn hơn với vòng **for** và giá trị tăng là 1. Với mỗi lần tịnh tiến **y** thêm 1 đơn vị ấy, mình liên tục kiểm tra xem ô nằm tại vị trí kiểm tra có phải là ô trống không. Nếu đó là ô trống, mình sẽ kiểm tra thêm 2 đoạn thẳng còn lại để nối 2 điểm ấy có thỏa mãn không bằng 2 hàm **checkLineX()** và **checkLineY()** với các tham số truyền vào tương ứng là điểm cuối cùng trong mỗi đoạn thẳng được vẽ:

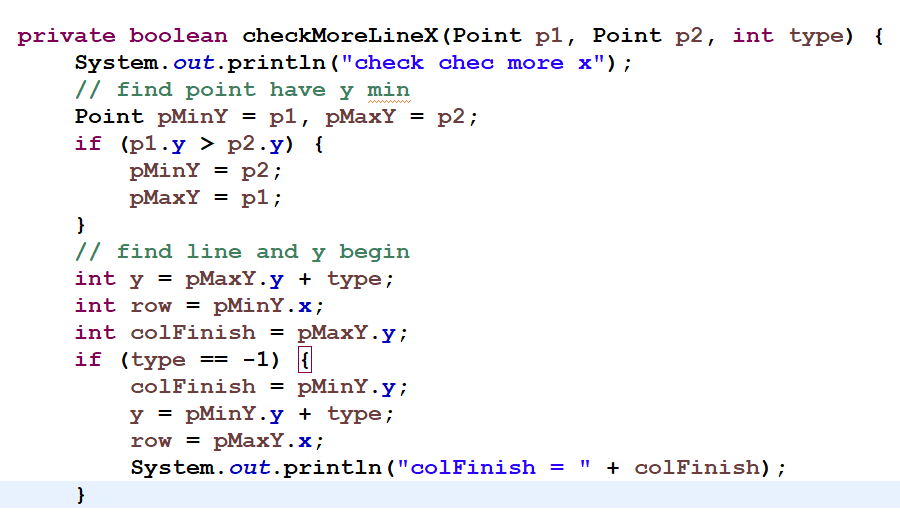


* Nếu kiểm tra được 1 đường đi thỏa mãn, hàm kiểm tra sẽ trả về **true**. Ngược lại nếu gặp 1 icon khác trước khi tìm được đường đi thỏa mãn, hàm sẽ trả về **false**. Chú ý rằng, mình kiểm tra liên tục bằng vòng **for**, nên nếu gặp 1 icon khác khi chưa tìm được đường đi thỏa mãn, điều đó nghĩa là tất cả các trường hợp trước đều không có đường đi đúng, đó là cơ sở bảo đảm cho việc trả về **false** của mình. Và dĩ nhiên, khi kiểm tra hết đường đi đến giới hạn của hình chữ nhật, nếu không có trường hợp nào đúng sẽ mặc định trả về **false**. Tương tự với phần xét 2 icon theo chiều dọc.
* Sẽ có trường hợp 2 icon chỉ được nối với nhau bằng 2 đoạn, nghĩa là chỉ cần xét 2 đoạn là đã tìm được icon còn lại rồi. Với trường hợp ấy, khi chạy hàm **checkLineX()**và **checkLineY()** sẽ có 1 trong 2 hàm xét 2 điểm trùng nhau. Tuy nhiên như mình nói ở trên, khi 2 điểm trùng nhau, hàm này vẫn sẽ trả về **true**, nhờ vậy, việc kiểm tra sẽ không bị ảnh hưởng.
* Trong hình trên là trường hợp mình xét 2 icon theo chiều dọc.Em đã đánh dấu những đoạn nối và hàm **checkLineY()**sẽ là hàm với vị trí của 2 icon được xét trùng nhau. Và với những suy nghĩ ở trên, em đã viết thêm 2 hàm mới **checkRectX()**và **checkRectY()**trong **class Controller**.



#### **Được nối bằng tối đa 3 đoạn thẳng vượt ra ngoài phạm vi hình chữ nhật hình thành từ tọa độ của 2 icon.**

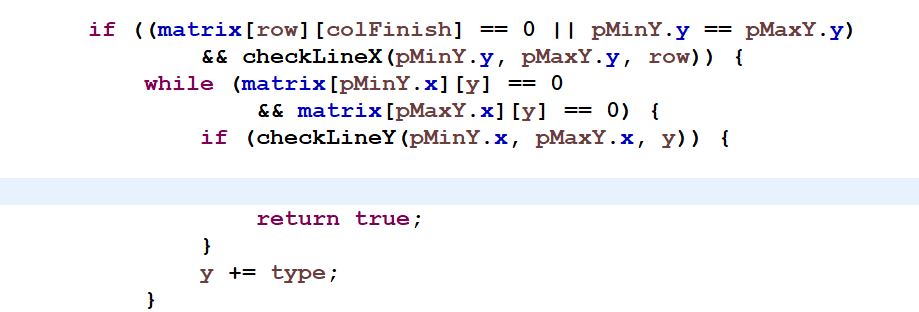
* Đối với trường hợp này, mình tạo ra hàm **checkMoreLineX()**trong **class Controller**. Để giải quyết vấn đề đoạn thẳng vượt ra ngoài hình chữ nhật hình thành từ đọa độ của 2 icon, tạo ra biến **type** trong tham số truyền vào của hàm **checkMoreLineX()**. Biến **type** chỉ nhận 2 giá trị: 1 và -1, tương ứng là 2 hướng đi về phía trước, hoặc ngược lại. So sánh 2 icon rồi tìm ra icon có hoành độ nhỏ hơn tương ứng là **pMinY** và **pMaxY**. Tương ứng với giá trị của **type**, nếu **type = 1**, tạo ra 1 biến **y**nhận giá trị bằng **pMaxY.y + type** , ngược lại, nếu **type =** -1, biến **y** của mình sẽ nhận giá trị bằng **pMinY.y + type**. Đối với việc xét theo chiều ngang này, với 2 biến **pMaxY.y**và **pMinY.y** chính là giới hạn hoành độ của hình chữ nhật tạo thành bởi 2 icon đang xét. Việc cộng thêm **type** vào 2 biến chính là việc mở rộng đường tìm kiếm ra ngoài phạm vi của hình chữ nhật đó.



Trong phần code trên,khởi tạo thêm 2 biến đó là **row**và **colFinish**. Quá trình tìm kiếm luôn có 2 điểm: Điểm bắt đầu tìm kiếm và điểm kết thúc tìm kiếm. Ví dụ như ở những phần trên, luôn là chạy từ hoành độ nhỏ hơn (**y1**) đến hoành độ lớn hơn (**y2**). Ở đây, **row** và **colFinish**cũng đóng vai trò tương đương như vậy. **row** chính là tung độ (**x**) của điểm bắt đầu tìm kiếm, và **colFinish** là hoành độ (**y**) của điểm kết thúc tìm kiếm. Và ứng với 2 trường hợp của biến **type**, **row**và **colFinish** sẽ nhận những giá trị khác nhau.

****

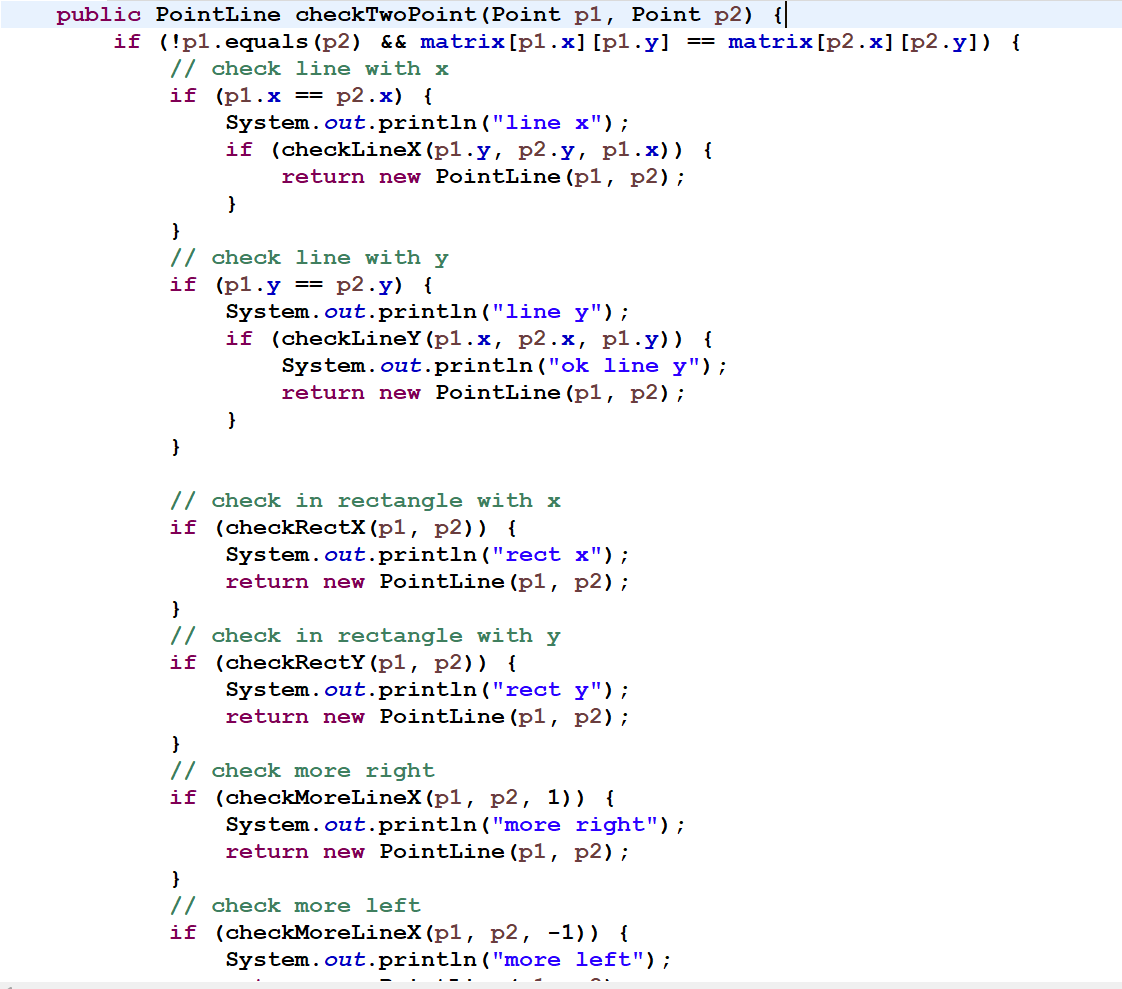
Tại đây mình có 3 điểm nối bởi vì trường hợp đó là xét 2 icon và các đoạn nối sẽ vượt ra ngoài phạm vi của hình chữ nhật tạo từ 2 icon ấy. Cần chú ý đến điểm giới hạn, cũng là điểm nối thứ nhất, đó là 1 góc hình chữ nhật mà chúng ta đang xét. Mình kiểm tra giá trị của ô tại điểm nối này, chính là ô **matrix[row][colFinish]**, ô này bắt buộc phải có giá trị bằng 0 hoặc ô này chính là 1 trong 2 icon mà chúng ta đang xét, điều đó cũng có nghĩa là 2 icon nằm ở vị trí cùng 1 cột. Khi đó chúng ta mới tiếp tục xét đến các ô nằm ở vị trí trong khoảng từ ô bắt đầu kiểm tra đến ô tại điểm nối thứ nhất. Đường đi này không được bị chặn bởi 1 icon nào, nghĩa là tất cả các giá trị của các ô trong khoảng đó đều sẽ bằng 0, kiểm tra nó bằng hàm **checkLineX(pMinY.y, pMaxY.y, row)**.



* Sau khi đã kiểm tra và vẽ được đoạn nối đến điểm nối thứ nhất thỏa mãn, chúng ta sẽ tiếp tục với điểm nối thứ 2 và thứ 3. Về phần này, vì không còn giới hạn phạm vi, nên không thể dùng vòng **for**được. Thay vào đó, với biến **y** đã được khởi tạo ở trên, mình dùng vòng **while**để kiểm tra xem chừng nào 2 ô **matrix[pMinY.x][y]** và **matrix[pMaxY.x][y]**sẽ còn cùng có giá trị bằng 0. Trong vòng **while**, mình liên tục kiểm tra đoạn nối giữa 2 điểm ứng với 2 ô ma trận đang xét bằng hàm **checkLineY(pMinY.x, pMaxY.x,** **y)**, nếu đoạn nối ấy thỏa mãn, nghĩa là đã tìm được đường đi thỏa mãn. Khi này, hàm **checkMoreLineX()** sẽ trả về **true**, nếu không, giá trị của **y**sẽ tiếp tục được cộng vào 1 giá trị bằng **type**(Tiếp tục tăng lên hoặc tiếp tục giảm đi). Nếu vòng **While**kết thúc mà vẫn chưa tìm được đường đi thỏa mãn, hàm sẽ trả về **false**.

### **3.3.3. Hoàn thiện hàm checkTwoPoint(Point p1, Point p2).**

Thứ tự kiểm tra sẽ là: Kiểm tra 2 icon xem chúng có nằm trên cùng 1 hàng hoặc 1 cột không, rồi đến kiểm tra xem chúng có được nối với nhau bằng đường zigzag nằm trong phạm vi hình chữ nhật tạo bởi 2 icon đó không, và cuối cùng là kiểm tra 2 icon có được nối bằng đường zigzag nằm ngoài phạm vi hình chữ nhật ấy không. Việc kiểm tra theo thứ tự này sẽ giúp chúng ta kiểm tra tuần tự, tiết kiệm thời gian và không bị bỏ sót trường hợp nào cả. Và dưới đây là hàm **checkTwoPoint(Point p1, Point p2)**hoàn chỉnh:



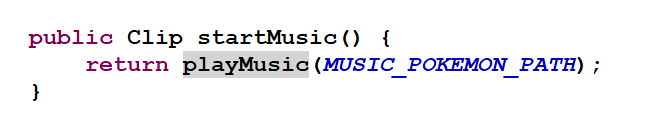
### **3.4 Hoàn thiện game với các hiệu ứng âm thanh**

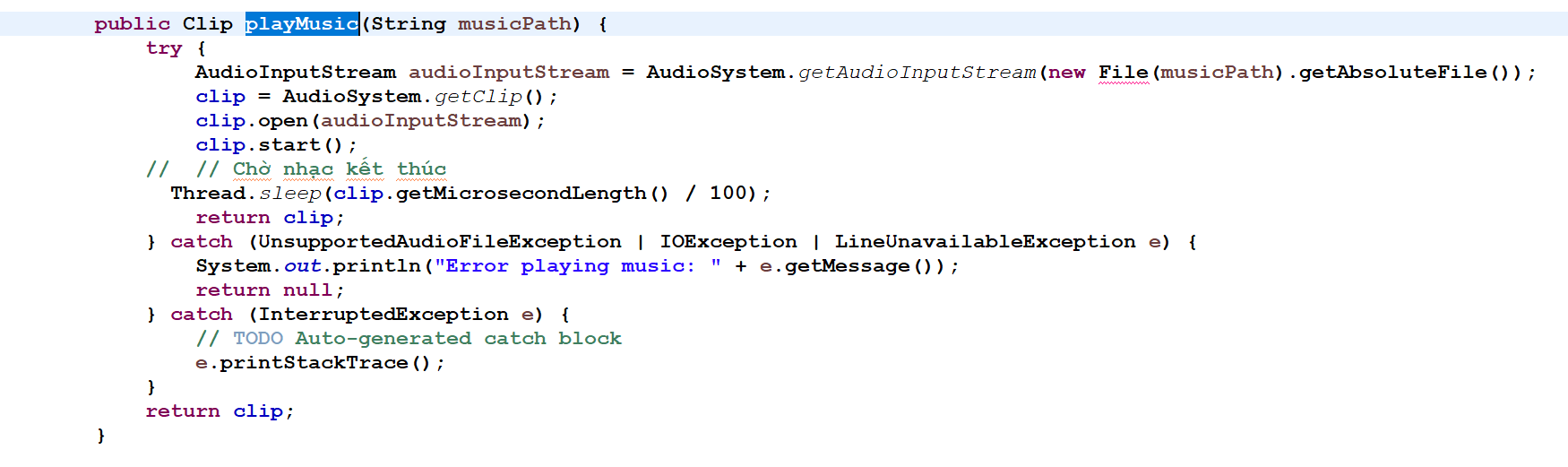
* **Cụ thể, lớp Music có các phương thức để chơi nhạc nền, phát âm thanh cho các trường hợp khác nhau trong game như nút Match và NotMatch, và các trường hợp thắng hoặc thua game.**
* **Ngoài ra, lớp cũng cung cấp các phương thức để dừng, tạm dừng và tiếp tục phát nhạc, lấy và đặt vị trí hiện tại của âm thanh, và một phương thức main() để thử nghiệm các phương thức trong lớp**
* **Để hoàn thiện phần âm thanh trong game, chúng ta có thể sử dụng các phương thức trong lớp Music để phát các tệp âm thanh phù hợp với từng trường hợp trong game.**
* **Khi bắt đầu game, chúng ta có thể gọi phương thức startMusic() để phát nhạc nền.**

****

**Đoạn code trên định nghĩa một lớp MusicThread là một luồng (thread) để chơi nhạc trong game. Lớp này kế thừa từ lớp Thread và ghi đè phương thức run() của lớp Thread để thực hiện việc chơi nhạc và lặp lại khi nhạc kết thúc.**

**Trong phương thức run(), đầu tiên ta khởi tạo một đối tượng Music từ lớp Music để sử dụng các phương thức để chơi nhạc. Tiếp theo, trong vòng lặp vô hạn, ta gọi phương thức startMusic() của đối tượng Music để chơi nhạc. Sau đó, ta dùng một vòng lặp while để kiểm tra xem nhạc có đang được phát hay không. Nếu nhạc đang được phát, vòng lặp sẽ tiếp tục chạy và Thread sẽ sleep trong 1 giây trước khi kiểm tra lại. Nếu nhạc đã dừng, phương thức isRunning() của đối tượng Clip sẽ trả về giá trị false và vòng lặp sẽ kết thúc. Sau đó, vòng lặp bên ngoài sẽ tiếp tục và lại gọi phương thức startMusic() để chơi lại nhạc.**

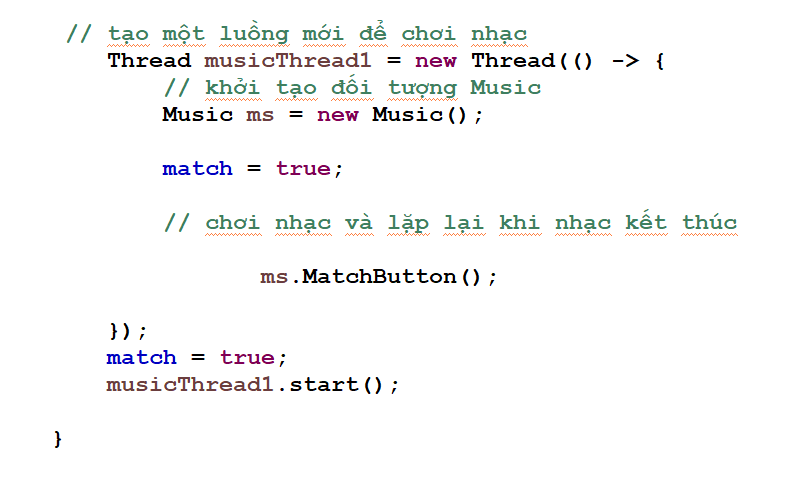
****

****

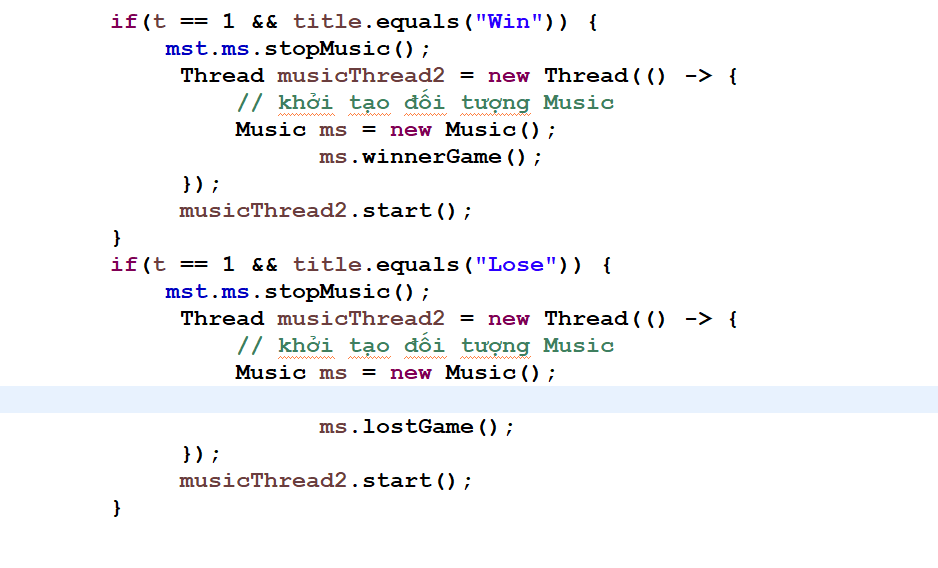
* **Hàm playMusic được sử dụng để chơi một file nhạc với đường dẫn được truyền vào dưới dạng tham số. Hàm sử dụng AudioSystem để tạo một AudioInputStream từ file nhạc, sau đó sử dụng AudioSystem.getClip() để tạo ra một Clip để chơi nhạc.**
* **Sau đó, hàm gọi phương thức start() để chơi nhạc, và sử dụng Thread.sleep() để tạm dừng luồng thực thi của chương trình trong thời gian nhạc đang được phát, được tính bằng độ dài của nhạc (được chia cho 100 để đổi sang đơn vị micro giây).**

**Nếu có lỗi khi chơi nhạc, hàm sẽ in ra một thông báo lỗi và trả về giá trị null. Nếu có lỗi trong quá trình Thread.sleep(), hàm cũng sẽ in ra thông báo lỗi và in ra stack trace của lỗi, và trả về giá trị null.**

* **Khi người chơi tìm thấy một cặp Pikachu khớp nhau, chúng ta có thể gọi phương thức MatchButton() để phát âm thanh tương ứng.**

****

* **Khi người chơi không tìm thấy cặp Pikachu khớp nhau, chúng ta có thể gọi phương thức NotMatchButton() để phát âm thanh tương ứng tương tự với trường hợp match.**
* **Khi người chơi thua cuộc, chúng ta có thể gọi phương thức lostGame() để phát âm thanh báo hiệu thua cuộc hoặc khi thăng cuộc ta sẽ gọi phương thức winnerGame().**

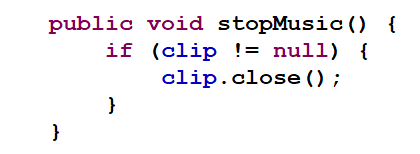
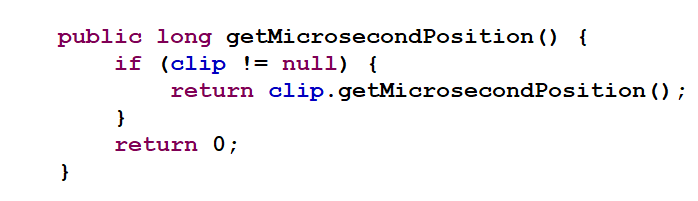
****

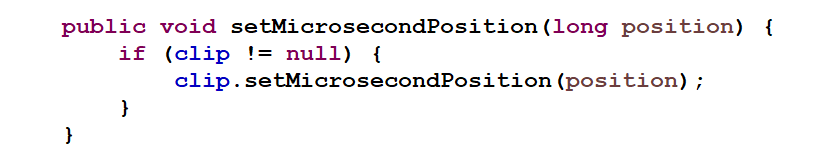
**Lúc ta sẽ phãi dựng phát nhạc nền bằng hàm stopMusic và tạo một luồng chạy hàm winnerGame() nếu chiền thắng hoặc thua thì gọi hàm loseGame(), và không có vòng lặp while vì âm thanh chỉ được phát một lần**

* **Chúng ta cũng có thể sử dụng các phương thức stopMusic() dừng nhạc lại, pausemusic() để tạm dừng và resumemusic() tiếp tục phát nhạc, để kiểm soát việc phát nhạc, và các phương thức getMicrosecondPosition() và setMicrosecondPosition() để lấy và đặt vị trí hiện tại của âm thanh.**
* **Khi người chơi chọn 1 trong các option trong thanh menu thì lúc này sẽ hiễn ra một JoptionPanel lúc này nhạc sẽ được tạm dừng**

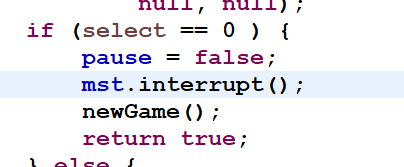
****

**Hàm pausemusic() được gọi để tạm dừng nhạc, getMicrosecondPosition() dung để lấy ra vị trí nhạc đang phát và sét vị trí âm thanh hiện tại của âm thanh bằng hàm getMicrosecondPosition**

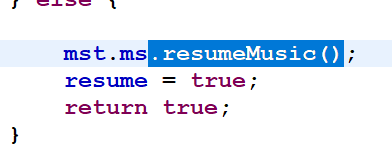
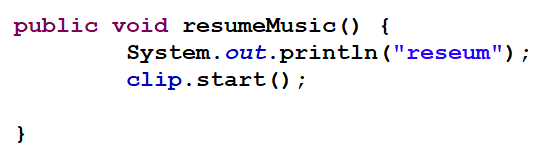
**** ****

****

**Khi người chơi chọn option Yes thì lúc này thread phát âm thanh sẽ bị huy bằng hàm interrupt()**

****

**Khi người chơi chọn option No thì lúc này tiếp tục được phát lại bằng hàm resumemusic()**

**** ****