**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**Đề tài: Thiết kế game Brick Breakout**

**BÁO CÁO BÀI TẬP LỚN MÔN CÔNG NGHỆ JAVA**

**Giảng viên hướng dẫn:**

|  |  |
| --- | --- |
| **Sinh viên thực hiện** | **Mã sinh viên** |
| **Nguyễn Đăng Đức** | **211202200** |

**Nhóm: 7**

**Lớp: CNTT6- K62 (N08)**

**Hà Nội, Tháng 4 năm 2023**

Mục lục

[**I. Mô tả bài toán và phương hướng giải quyết bài toán** 3](#_Toc133926910)

[**1. Mô tả bài toán** 3](#_Toc133926911)

[**2. Phương hướng giải quyết bài toán** 3](#_Toc133926912)

[**II. Các lớp của chương trình** 4](#_Toc133926913)

[**1. Lớp General** 5](#_Toc133926914)

[**2. Lớp Controller** 5](#_Toc133926915)

[**3. Lớp Level** 6](#_Toc133926916)

[**4. Lớp Main** 7](#_Toc133926917)

[**5. Lớp IModel** 7](#_Toc133926918)

[**6. Lớp Ball** 8](#_Toc133926919)

[**7. Lớp Brick** 9](#_Toc133926920)

[**8. Lớp Paddle** 9](#_Toc133926921)

[**9. Lớp Constants** 10](#_Toc133926922)

[**10. Lớp Menu** 11](#_Toc133926923)

[**11. Lớp View** 11](#_Toc133926924)

[**III. Phân tích một số thuộc tính trong lớp Controller** 12](#_Toc133926925)

[**1. initial()** 12](#_Toc133926926)

[**2. gameInit()** 13](#_Toc133926927)

[**3. MapGerenation()** 14](#_Toc133926928)

[**4. drawObjects()** 15](#_Toc133926929)

[**5. checkCollision()** 16](#_Toc133926930)

[**6. gameFinished()** 20](#_Toc133926931)

[**7. doGameCycle() && stopGame()** 21](#_Toc133926932)

[**8. continueGame()** 22](#_Toc133926933)

[**IV. Xây dựng chương trình** 23](#_Toc133926934)

[**1. Khởi tạo trò chơi** 23](#_Toc133926935)

[**2. Xây dựng chương trình** 23](#_Toc133926936)

[**3. Bắt đầu trò chơi** 24](#_Toc133926937)

[**V. Hướng dẫn chạy mã lệnh trên một máy tính khác** 25](#_Toc133926938)

[**VI. Link Github** 25](#_Toc133926939)

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

## **1. Mô tả bài toán**

Brik Breaker (hay còn gọi là Brick Breaker) là một trò chơi điện tử kinh điển, thường được chơi trên các thiết bị di động hoặc máy tính. Bài toán game Brik Breaker là tiêu diệt các khối gạch bằng cách điều khiển một viên bi và đưa nó va vào các khối gạch để phá huỷ chúng. Mục tiêu của trò chơi là phá hủy tất cả các khối gạch để điểm số càng cao càng tốt.

Trong trò chơi, người chơi sẽ điều khiển một thanh ngang ở đáy màn hình để ném viên bi lên trên. Viên bi sẽ di chuyển ngược lại khi chạm vào các thành phía trên và hai bên của màn hình. Khi viên bi chạm vào một khối gạch, khối gạch sẽ bị phá hủy và người chơi sẽ được cộng điểm. Nếu viên bi rơi xuống đáy màn hình mà không được bắt bởi thanh ngang, người chơi sẽ mất một lượt chơi.

Trò chơi Brik Breaker thường có nhiều cấp độ khác nhau với các mức độ khó khác nhau. Nếu người chơi phá hủy hết tất cả các khối gạch trong một cấp độ, họ sẽ tiến tới cấp độ tiếp theo với số lượng khối gạch tăng dần và độ khó cũng tăng lên.

## **2. Phương hướng giải quyết bài toán**

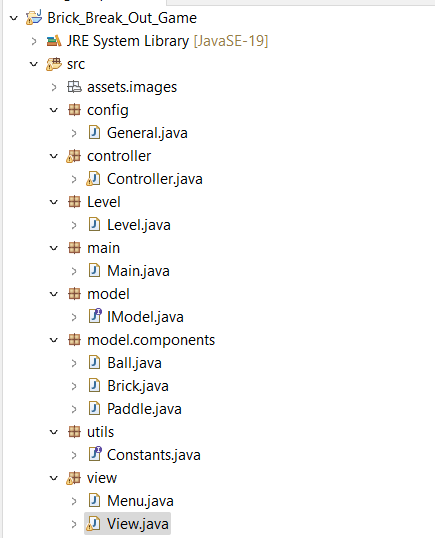
Để giải quyết bài toán game Brik Breaker, ta có thể sử dụng các phương pháp lập trình cơ bản để tạo ra một game đơn giản. Đầu tiên, ta cần thiết kế các thành phần cơ bản của game như viên bi, thanh ngang, khối gạch, điểm số, số lượt chơi còn lại, v.v. Sau đó, ta sẽ tạo ra các hàm để xử lý các tương tác giữa các thành phần đó.

Cụ thể, ta có thể sử dụng các thư viện đồ họa để vẽ các thành phần lên màn hình, sử dụng các hàm để xử lý các va chạm giữa viên bi và các khối gạch, viên bi và thanh ngang, hay viên bi và các cạnh của màn hình. Ta cũng cần quản lý điểm số và số lượt chơi còn lại của người chơi thông qua các hàm tương ứng.

Ngoài ra, để tăng tính thú vị và độ khó của trò chơi, ta có thể thêm vào các tính năng như các power-up (như tăng tốc độ bi, phá hủy tất cả các khối gạch trong một vùng nhất định, v.v.) hoặc các chướng ngại vật khác (như các cửa sổ xuất hiện đột ngột trong màn hình, các khối gạch di chuyển, v.v.). Tất cả những tính năng này sẽ giúp trò chơi thêm phần hấp dẫn và thú vị hơn đối với người chơi.

Tóm lại, để giải quyết bài toán game Brik Breaker, ta cần thiết kế các thành phần cơ bản của game, xử lý các tương tác giữa các thành phần đó và thêm vào các tính năng để tăng tính thú vị và độ khó của trò chơi.

# **II. Các lớp của chương trình**

****

## **1. Lớp General**

Lớp này định nghĩa các thuộc tính và phương thức chung cho các lớp khác trong game.

Các thuộc tính bao gồm:

x: vị trí theo trục hoành

y: vị trí theo trục tung

imageWidth: chiều rộng của hình ảnh

imageHeight: chiều cao của hình ảnh

image: hình ảnh

Các phương thức bao gồm:

getX (), setX(), getY(), setY(): các phương thức get và set cho các thuộc tính x và y

getImageWidth(), setImageWidth(), getImageHeight(), setImageHeight(): các phương thức get và set cho các thuộc tính imageWidth và imageHeight

getImage(), setImage(): các phương thức get và set cho thuộc tính image

getRectangle(): trả về một hình chữ nhật có vị trí và kích thước của đối tượng

getImageDimensions(): lấy kích thước của hình ảnh và cập nhật các thuộc tính imageWidth và imageHeight

## **2. Lớp Controller**

Lớp "Controller" được thiết kế bằng Java và sử dụng framework Swing để tạo ra một trò chơi đơn giản. Lớp "Controller" mở rộng lớp JPanel và thực hiện logic của trò chơi.

Lớp "Controller" có trách nhiệm khởi tạo và quản lý các phần tử trong trò chơi như bóng, gậy, gạch và các cấp độ trò chơi. Lớp cũng xử lý vòng lặp game và cập nhật trạng thái của trò chơi dựa trên đầu vào của người dùng.

Trạng thái của trò chơi được lưu trữ trong các biến khác nhau như inGame, message, score, combo\_score và Timer\_Display\_Combo. Biến inGame là một boolean được thiết lập thành true khi trò chơi đang chạy và false khi trò chơi kết thúc. Biến message chứa thông báo khi trò chơi kết thúc, và biến score chứa điểm số hiện tại của người chơi. Biến combo\_score chứa điểm combo hiện tại, và biến Timer\_Display\_Combo là một bộ đếm thời gian điều khiển thời gian hiển thị điểm combo trên màn hình.

Lớp "Controller" sử dụng các phương thức khác nhau để khởi tạo và cập nhật trò chơi. Phương thức "initial" được gọi khi đối tượng "Controller" được tạo ra, và nó thêm một trình lắng nghe phím vào đối tượng JPanel để bắt lấy đầu vào của người dùng. Phương thức "gameInit" khởi tạo các phần tử của trò chơi như bóng, gậy, gạch và các cấp độ trò chơi. Phương thức "MapGeneration" tạo ra các cấp độ trò chơi bằng cách sử dụng lớp "Level". Phương thức "paintComponent" có trách nhiệm vẽ các phần tử của trò chơi trên đối tượng JPanel. Phương thức "drawObjects" vẽ bóng, gậy và gạch trên màn hình. Phương thức "gameFinished" được gọi khi trò chơi kết thúc và hiển thị thông báo khi trò chơi kết thúc.

Lớp "Controller" sử dụng lớp "GameCycle" để quản lý vòng lặp trò chơi. Lớp "GameCycle" implements giao diện ActionListener, và phương thức "actionPerformed" của nó được gọi mỗi khi bộ đếm thời gian hết hạn. Phương thức "actionPerformed" cập nhật trạng thái trò chơi bằng cách di chuyển quả bóng và thanh chắn, kiểm tra va chạm và cập nhật điểm số.

Tóm lại, lớp "Controller" chịu trách nhiệm quản lý các thành phần và logic của trò chơi, và lớp "GameCycle" chịu trách nhiệm quản lý vòng lặp trò chơi. Vòng lặp trò chơi cập nhật trạng thái trò chơi và vẽ các thành phần trò chơi trên màn hình bằng phương thức "paintComponent".

## **3. Lớp Level**

Lớp **Level** định nghĩa một mảng 2 chiều **arr** để lưu trữ các cấp độ của trò chơi. Có 2 phương thức được cung cấp trong lớp:

**getLevel(int level):** Trả về mảng con tương ứng với cấp độ **level** truyền vào.

**getLevelMax():** Trả về số lượng cấp độ hiện có.

Để sử dụng lớp **Level,** bạn có thể làm như sau:

Tạo một đối tượng của lớp **Level: Level level = new Level();**

Sử dụng phương thức **getLevel(int level)** để lấy mảng tương ứng với cấp độ **level: int[] levelArray = level.getLevel(1);**

Sử dụng mảng được trả về để hiển thị cấp độ trên giao diện người dùng của trò chơi của bạn.

## **4. Lớp Main**

Đây là file **Main.java** trong package **main**. File này chỉ chứa phương thức **main()** để khởi chạy chương trình.

Khi chạy chương trình, phương thức **main()** sẽ tạo một đối tượng **Menu** và gọi hàm khởi tạo của nó để hiển thị menu cho người dùng.

Để cài đặt, bạn có thể tạo một file mới trong package **main**, đặt tên là **Main.java** và copy đoạn code trên vào file này. Sau đó, bạn có thể chạy chương trình bằng cách chọn **Run As -> Java Application** hoặc nhấn nút Run trong IDE của bạn.

## **5. Lớp IModel**

Đây là một interface tên IModel trong package model, có các method sau:

**initial():** không có tham số, dùng để khởi tạo trạng thái ban đầu của đối tượng.

**initial(int x, int y):** có hai tham số kiểu int là x và y, dùng để khởi tạo trạng thái ban đầu của đối tượng với vị trí (x,y).

**loadImage():** không có tham số, dùng để tải hình ảnh của đối tượng.

**reset():** không có tham số, dùng để reset lại trạng thái của đối tượng về trạng thái ban đầu.

**move():** không có tham số, dùng để di chuyển đối tượng.

**update():** không có tham số, dùng để cập nhật trạng thái của đối tượng sau khi di chuyển.

**repaint():** không có tham số, dùng để vẽ lại đối tượng trên màn hình.

**collision():** không có tham số, trả về kiểu boolean, dùng để kiểm tra xem đối tượng đã va chạm với đối tượng khác hay chưa.

**bounce():** không có tham số, trả về kiểu boolean, dùng để kiểm tra xem đối tượng có nảy lại sau khi va chạm hay không.

**destroy():** không có tham số, dùng để hủy bỏ đối tượng.

## **6. Lớp Ball**

Đây là lớp **Ball** được triển khai từ interface **IModel** và chứa các thuộc tính và phương thức liên quan đến bóng trong trò chơi.

Các phương thức và thuộc tính trong lớp **Ball:**

Thuộc tính:

**xDirection:** hướng di chuyển theo trục x của bóng.

**yDirection:** hướng di chuyển theo trục y của bóng.

Phương thức:

**getxDirection() và setxDirection():** lấy và thiết lập hướng di chuyển của bóng theo trục x.

**getyDirection()** và **setyDirection():** lấy và thiết lập hướng di chuyển của bóng theo trục y.

**initial():** khởi tạo bóng với hướng di chuyển ban đầu theo trục x và y, tải ảnh của bóng và reset vị trí ban đầu.

**initial(int x, int y):** phương thức trống không được triển khai.

**loadImage():** tải ảnh của bóng.

**reset():** đặt lại vị trí ban đầu của bóng.

**move():** di chuyển bóng theo hướng di chuyển hiện tại và xử lý nếu bóng chạm biên trên, biên dưới, biên trái hoặc biên phải của khung trò chơi.

**update():** phương thức trống không được triển khai.

**repaint():** phương thức trống không được triển khai.

**collision():** phương thức trống không được triển khai.

**bounce():** xử lý nếu bóng chạm biên trên, biên dưới, biên trái hoặc biên phải của khung trò chơi và đổi hướng di chuyển của bóng tương ứng.

**destroy():** phương thức trống không được triển khai.

## **7. Lớp Brick**

Đây là một đoạn mã Java định nghĩa lớp **Brick** trong gói **model.components**. Lớp này được kế thừa từ lớp **General** và implements interface **IModel.**

Lớp **Brick** đại diện cho các viên gạch trong trò chơi. Các thuộc tính và phương thức được định nghĩa như sau:

Thuộc tính **destroyed** để xác định liệu viên gạch đã bị phá hủy hay chưa.

Phương thức **initial()** và **initial(int x, int y)** để khởi tạo đối tượng viên gạch với tọa độ x, y và hình ảnh tương ứng.

Phương thức **loadImage()** để tải hình ảnh của viên gạch lên.

Phương thức **randomNumber()** để trả về một số nguyên ngẫu nhiên trong khoảng từ 0 đến 2, để chọn một hình ảnh ngẫu nhiên cho viên gạch.

Các phương thức **reset(), move(), update(), repaint(), collision(),** **bounce(),** và **destroy()** được định nghĩa để triển khai interface **IModel**, tuy nhiên các phương thức này đang không được sử dụng trong lớp **Brick.**

Một số hình ảnh của viên gạch được lưu trữ trong các chuỗi **Image\_\_Brick, Brick\_\_red, Brick\_\_blue, Brick\_\_yellow,** và **Brick\_\_green.**

## **8. Lớp Paddle**

Đây là đoạn mã Java trong package **model.components** được viết để định nghĩa lớp **Paddle** (thanh chắn) trong trò chơi. Lớp **Paddle** kế thừa từ lớp **General** và implements interface **IModel.**

Các phương thức trong lớp **Paddle** bao gồm:

**initial:** Phương thức được gọi để khởi tạo thanh chắn, gọi phương thức **loadImage** để tải hình ảnh và phương thức **getImageDimensions** để lấy kích thước hình ảnh.

**initial(int x, int y):** Phương thức này được gọi để khởi tạo thanh chắn với vị trí xác định bởi tham số **x** và **y**. Tuy nhiên, phương thức này không được sử dụng trong đoạn mã.

**loadImage:** Phương thức này được sử dụng để tải hình ảnh của thanh chắn từ đường dẫn **src/assets/images/paddle.png.**

**reset:** Phương thức được gọi để reset lại vị trí của thanh chắn về giá trị ban đầu.

**move:** Phương thức này được sử dụng để di chuyển thanh chắn dựa trên giá trị của biến **xDirection.** Nếu giá trị **x** của thanh chắn chạm vào cạnh của màn hình thì phương thức **bounce** sẽ được gọi để đảo chiều di chuyển của thanh chắn.

**update, repaint, collision, destroy:** Các phương thức này không được sử dụng trong đoạn mã.

**bounce:** Phương thức này được gọi để đảm bảo rằng thanh chắn không bị di chuyển ra khỏi màn hình bằng cách kiểm tra xem vị trí **x** của thanh chắn có vượt quá kích thước của màn hình không. Nếu vượt quá thì giá trị **x** của thanh chắn được đặt lại để nó không chạm vào cạnh của màn hình.

**keyPressed, keyReleased:** Phương thức này được sử dụng để xử lý sự kiện khi người dùng ấn hoặc thả phím trên bàn phím. Nếu người dùng ấn phím mũi tên trái hoặc "A", biến **xDirection** sẽ được đặt về giá trị âm để thanh chắn di chuyển về bên trái. Tương tự, nếu người dùng ấn phím mũi tên phải hoặc "D", biến **xDirection** sẽ được đặt về giá trị dương để thanh chắn di chuyển về

## **9. Lớp Constants**

Đây là một giao diện Constants trong Java, chứa các hằng số được sử dụng trong trò chơi. Các giá trị cụ thể được xác định như sau:

message: chuỗi thông báo "Game Over".

SCREEN\_WIDTH: chiều rộng của màn hình trò chơi.

SCREEN\_HEIGHT: chiều cao của màn hình trò chơi.

NUMBER\_OF\_BRICK: số lượng gạch ban đầu trong trò chơi.

INITIAL\_PADDLE\_X: tọa độ x ban đầu của vợt.

INITIAL\_PADDLE\_Y: tọa độ y ban đầu của vợt.

INITIAL\_BALL\_X: tọa độ x ban đầu của quả bóng.

INITIAL\_BALL\_Y: tọa độ y ban đầu của quả bóng.

SPEED\_BALL: tốc độ ban đầu của quả bóng.

SPEED\_PADDLE: tốc độ của vợt.

BOTOM\_EDGE: đường biên dưới của màn hình trò chơi.

PERIOD: thời gian trễ giữa các lần cập nhật trong trò chơi.

HEALPOINT: điểm số ban đầu.

SCORE: điểm số hiện tại của người chơi.

CONBO\_SCORE: điểm số liên tục của người chơi.

LEVEL: cấp độ hiện tại của trò chơi.

MAX\_LEVEL: cấp độ tối đa của trò chơi.

## **10. Lớp Menu**

Đây là lớp Menu của giao diện người dùng. Trong đó:

Lớp này thực hiện việc tạo ra menu cho game bao gồm ảnh nền, logo và nút chơi game.

Trong phương thức tạo của lớp Menu, ảnh nền và logo được tải và hiển thị trên khung hình JFrame.

Nút chơi game được tạo ra từ một hình ảnh, được thu nhỏ và được hiển thị trên ảnh nền.

Nếu người dùng bấm vào nút chơi game, lớp sẽ đóng menu hiện tại và mở màn hình chơi game mới.

## **11. Lớp View**

Đây là class **View** của package **view** trong ứng dụng Breakout. Class này kế thừa từ **JFrame** của thư viện Swing để tạo ra một cửa sổ cho ứng dụng. Trong constructor của class, nó tạo ra một instance của class **Controller** (một JPanel) và thêm nó vào trong cửa sổ bằng phương thức **add().** Các thuộc tính của cửa sổ được đặt bằng các phương thức set như **setTitle(), setDefaultCloseOperation(), setLocation(), setResizable()** và **pack().** Cụ thể:

**setTitle()** đặt tiêu đề cho cửa sổ là "Breakout".

**setDefaultCloseOperation()** đặt phương thức xử lý sự kiện khi người dùng đóng cửa sổ là **EXIT\_ON\_CLOSE**, tức là khi người dùng đóng cửa sổ, ứng dụng sẽ thoát.

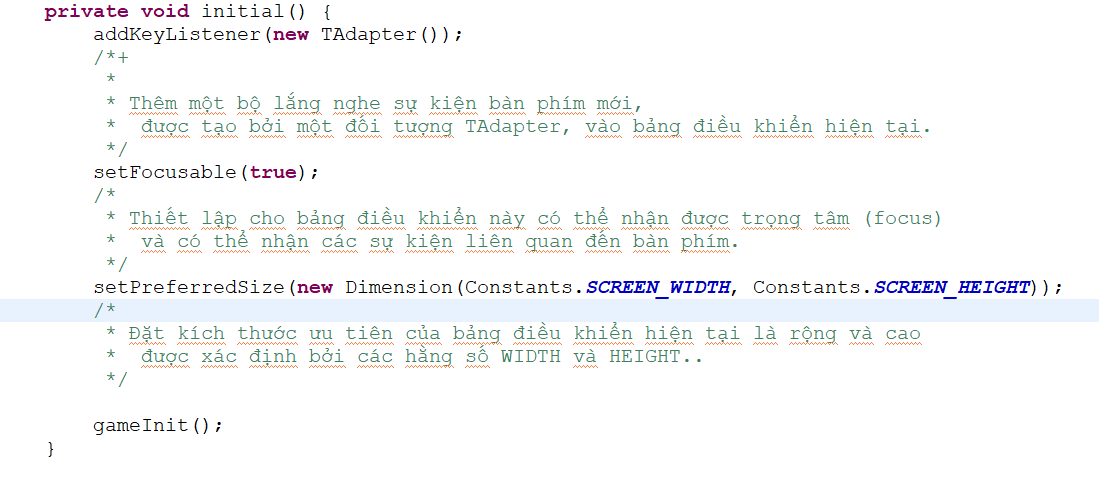
**setLocation()** đặt vị trí của cửa sổ trên màn hình.

**setResizable()** khóa kích thước của cửa sổ không cho phép người dùng thay đổi kích thước.

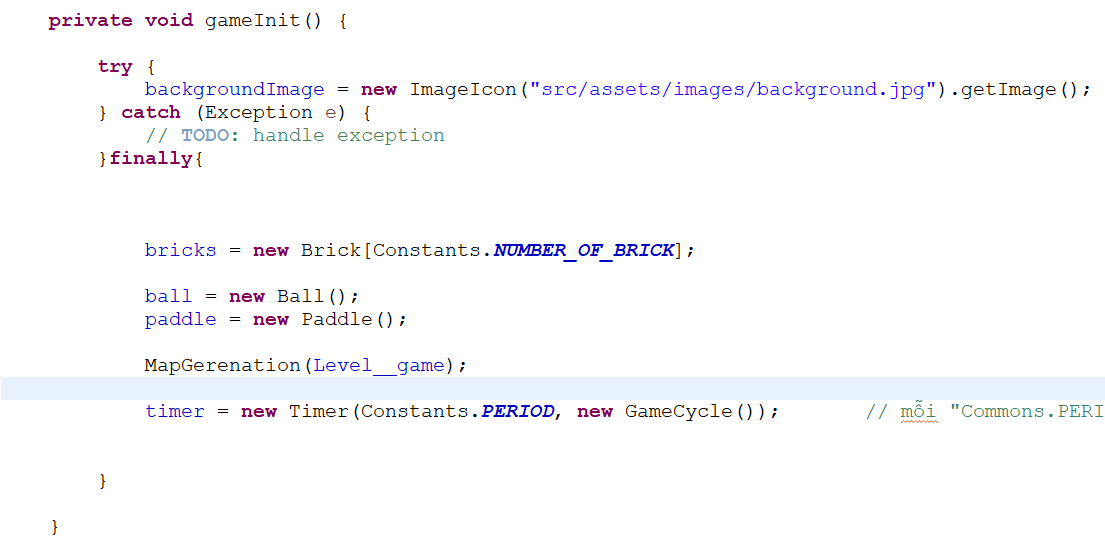
**pack()** điều chỉnh kích thước của cửa sổ để phù hợp với nội dung của nó.

# **III. Phân tích một số thuộc tính trong lớp Controller**

## **1. initial()**

****

## **2. gameInit()**

****

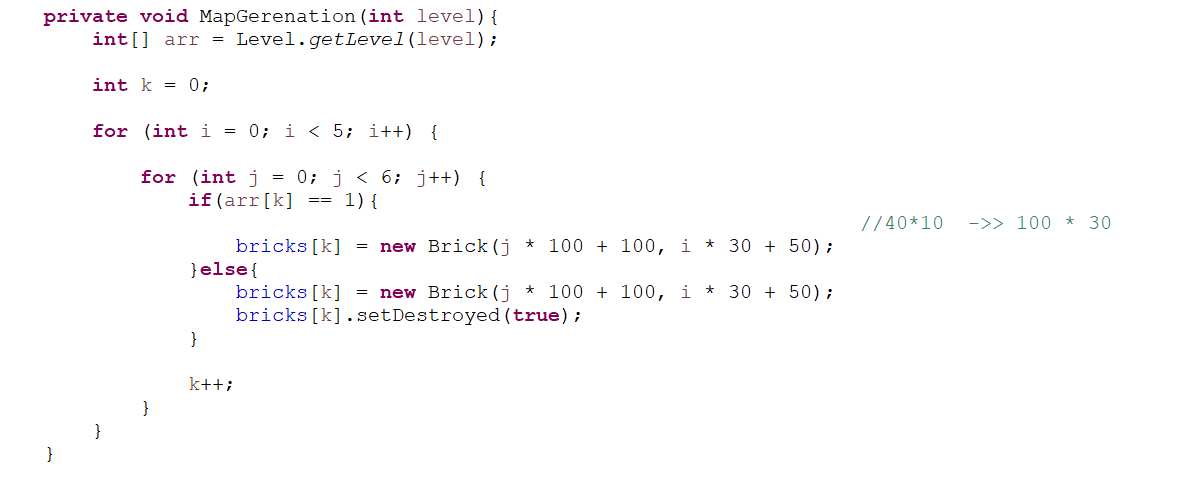
Phương thức **gameInit()** được sử dụng để khởi tạo các thành phần cần thiết của trò chơi.

Đầu tiên, nó tạo một đối tượng hình ảnh **backgroundImage** bằng cách đọc tệp ảnh **background.jpg** từ thư mục **src/assets/images/.**

Sau đó, nó tạo ra các đối tượng **bricks, ball** và **paddle. bricks** là một mảng các đối tượng gạch, **ball** là một đối tượng quả bóng và **paddle** là một đối tượng vợt.

Phương thức **MapGerenation(Level\_\_game)** được sử dụng để tạo ra các màn chơi. Cuối cùng, phương thức này tạo một đối tượng **Timer** với một khoảng thời gian cập nhật là **Constants.PERIOD** và một đối tượng **GameCycle** được chuyển vào để quản lý vòng lặp trò chơi.

## **3. MapGerenation()**

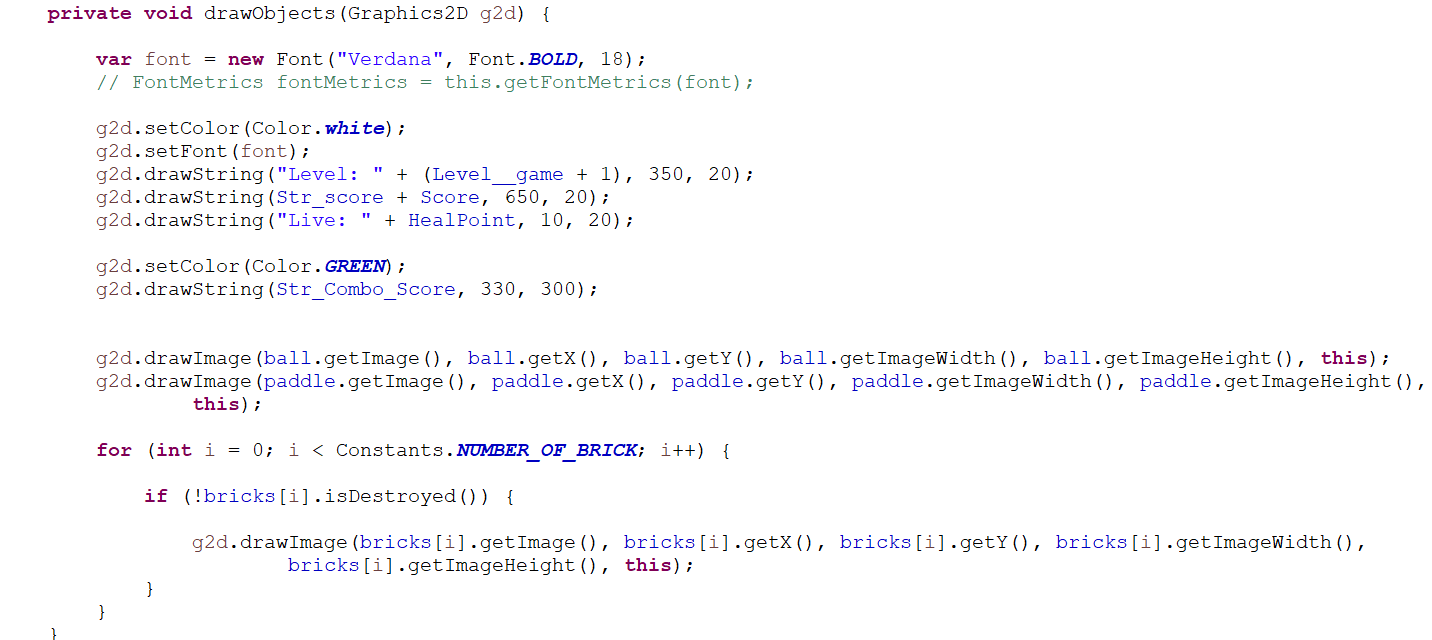
****

Phương thức **MapGeneration** được sử dụng để tạo ra các cấp độ cho trò chơi, thông qua lớp **Level.** Biến **level** được sử dụng để xác định cấp độ hiện tại của trò chơi.

Phương thức tạo ra một mảng **arr** chứa thông tin về trạng thái của các viên gạch trong cấp độ hiện tại. Mỗi phần tử của mảng tương ứng với một viên gạch, và giá trị của nó là 0 hoặc 1. Nếu giá trị của phần tử là 1, thì viên gạch tương ứng sẽ được tạo ra và được đặt tại một vị trí cụ thể trên màn hình. Nếu giá trị của phần tử là 0, thì viên gạch tương ứng sẽ không được tạo ra và sẽ được đánh dấu là bị phá hủy.

Sau đó, một vòng lặp lồng nhau được sử dụng để tạo ra các viên gạch và đặt chúng vào đúng vị trí trên màn hình. Cuối cùng, mảng **bricks** sẽ chứa các đối tượng **Brick** tương ứng với các viên gạch đã được tạo ra.

## **4. drawObjects()**

****

Phương thức **drawObjects(Graphics2D g2d)** được sử dụng để vẽ các đối tượng trong trò chơi trên màn hình. Cụ thể, nó vẽ các đối tượng sau:

Vẽ chuỗi ký tự "Level: X" trên cùng bên trái màn hình, trong đó X là mức độ hiện tại của trò chơi.

Vẽ điểm số của người chơi, được đặt trong biến **Score,** trên cùng bên phải màn hình.

Vẽ số mạng của người chơi, được đặt trong biến **HealPoint,** ở góc trái dưới màn hình.

Vẽ chuỗi ký tự "Combo: X" ở giữa màn hình khi người chơi đánh bại một hoặc nhiều viên gạch cùng một lúc, trong đó X là số điểm thưởng được tính dựa trên số viên gạch đánh bại trong combo hiện tại.

Vẽ bóng và thanh trượt ở vị trí hiện tại của chúng trên màn hình.

Vẽ tất cả các viên gạch còn lại trên màn hình, nếu chúng chưa bị phá hủy.

## **5. checkCollision()**

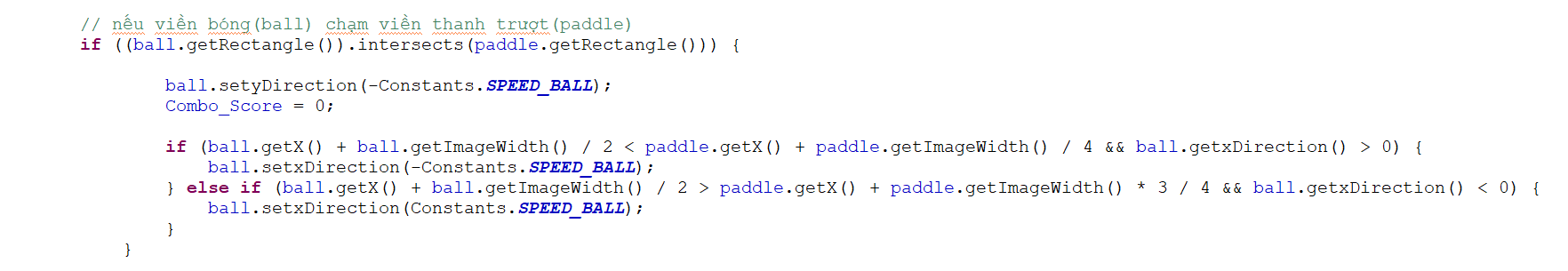
****

Phần code này kiểm tra va chạm của quả bóng và xử lý các trường hợp tương ứng. Đầu tiên, nếu quả bóng chạm đến phía dưới cùng của màn hình, số lượt chơi sẽ giảm đi 1. Nếu số lượt chơi bằng 0, trò chơi sẽ kết thúc, ngược lại thì trò chơi sẽ tiếp tục.

Tiếp theo, vòng lặp for sẽ kiểm tra xem có bao nhiêu viên gạch đã bị phá hủy. Nếu số lượng này bằng với tổng số viên gạch, mức độ chơi sẽ được tăng lên 1. Nếu mức độ chơi đã đạt đến mức tối đa, trò chơi sẽ kết thúc và thông báo "Victory". Nếu không, sẽ khởi động lại trò chơi.

Sau đó, phần code kiểm tra xem quả bóng có va chạm với thanh trượt hay không. Nếu có, quả bóng sẽ di chuyển lên trên và Combo\_Score sẽ được đặt lại bằng 0. Nếu viền bóng chạm vào viền bên trái của thanh trượt, quả bóng sẽ di chuyển sang trái, và ngược lại với viền bên phải của thanh trượt.

Cuối cùng, phần code kiểm tra xem quả bóng có va chạm với viên gạch nào không. Nếu có, các điểm cực đại của quả bóng sẽ được tính toán và xác định hướng di chuyển của quả bóng tương ứng với vị trí va chạm. Nếu viên gạch chưa bị phá hủy, nó sẽ bị phá hủy và Combo\_Score sẽ được tăng lên 1. Score sẽ được tính toán dựa trên số lượng combo đã thực hiện.

****

Đoạn code này là phần xử lý va chạm giữa bóng và thanh trượt, và cũng xử lý việc va chạm giữa bóng và các viên gạch.

Trong phần va chạm giữa bóng và thanh trượt, nếu viền của bóng chạm vào viền của thanh trượt, thì code sẽ thực hiện các hành động sau:

Đổi hướng di chuyển theo trục y của bóng, để bóng quay trở lại.

Đặt Combo\_Score về 0, đây là biến tính điểm khi liên tiếp đập được các viên gạch, và sẽ được tính lại sau khi bóng đập vào thanh trượt.

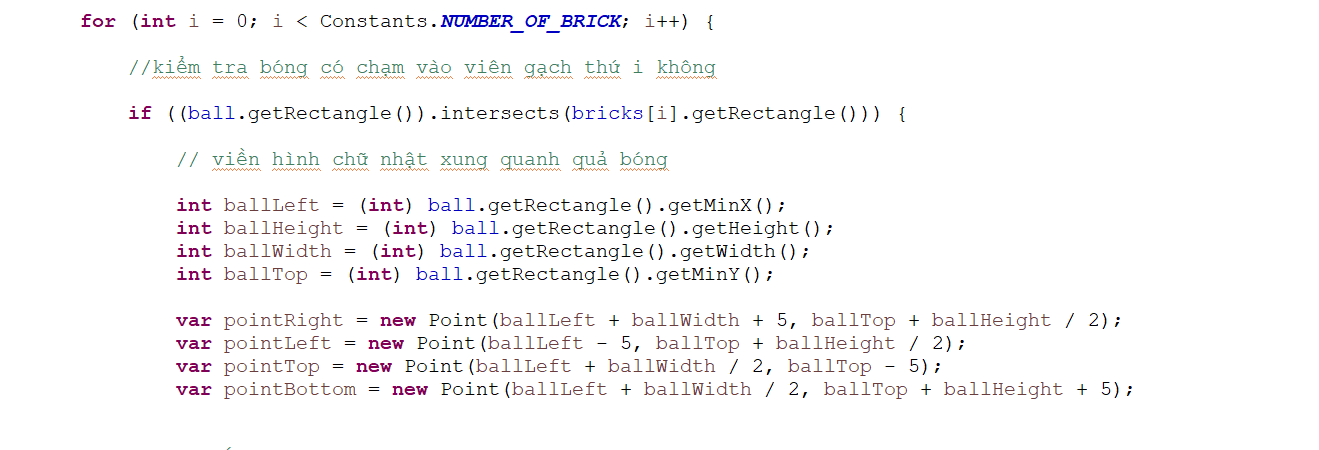
Kiểm tra xem vị trí của bóng so với thanh trượt. Nếu vị trí của bóng nằm trong 1/4 phía trái của thanh trượt và hướng di chuyển của bóng là về phía bên phải, thì sẽ đổi hướng di chuyển theo trục x của bóng sang trái. Ngược lại, nếu vị trí của bóng nằm trong 3/4 phía phải của thanh trượt và hướng di chuyển của bóng là về phía bên trái, thì sẽ đổi hướng di chuyển theo trục x của bóng sang phải.

Sau đó, trong vòng lặp for, code sẽ kiểm tra va chạm giữa bóng và từng viên gạch. Nếu một viên gạch đã bị phá hủy, biến đếm j sẽ tăng lên. Nếu tất cả các viên gạch đều đã bị phá hủy, tức j = số lượng viên gạch, thì code sẽ thực hiện các hành động sau:

Tăng cấp độ Level\_\_game lên 1.

Nếu cấp độ Level\_\_game đã đạt tối đa, thì đặt thông báo "Victory" và dừng trò chơi.

Nếu cấp độ Level\_\_game chưa đạt tối đa, thì khởi tạo lại các giá trị và dừng đồng hồ đếm thời gian.



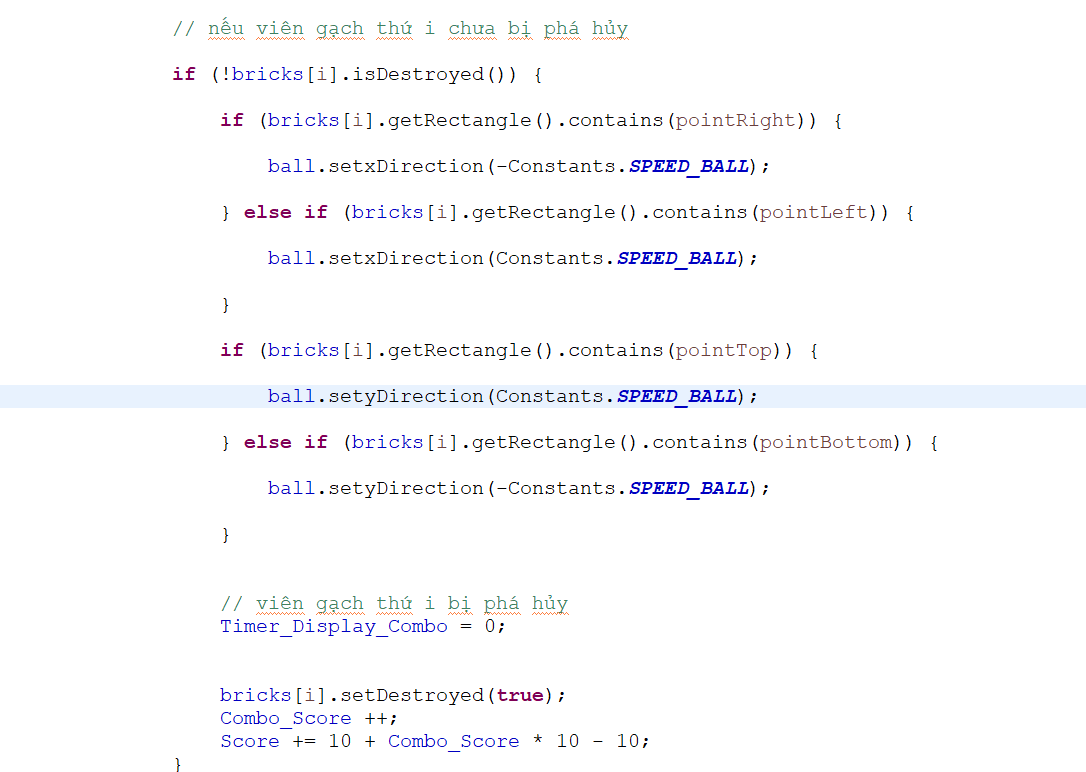
Đoạn mã trên là một đoạn mã trong ngôn ngữ lập trình Java, được sử dụng để kiểm tra va chạm giữa quả bóng và các viên gạch trong trò chơi Breakout.

Vòng lặp for được sử dụng để lặp qua tất cả các viên gạch trong mảng bricks. Số lượng viên gạch được xác định bằng hằng số NUMBER\_OF\_BRICK trong lớp Constants.

Trong khối if, phương thức intersects() được sử dụng để kiểm tra xem hình chữ nhật bao quanh quả bóng (được xác định bằng phương thức getRectangle()) có chồng lên hình chữ nhật bao quanh viên gạch thứ i hay không (được xác định bằng phương thức getRectangle() của đối tượng bricks[i]).

Nếu có va chạm xảy ra, vị trí của các điểm trên, dưới, trái và phải của quả bóng được tính toán bằng cách sử dụng các phương thức của lớp Rectangle (getMinX(), getMinY(), getHeight() và getWidth()).

Các điểm này được sử dụng để xác định hướng di chuyển của quả bóng sau khi va chạm xảy ra. Cụ thể, các điểm pointRight, pointLeft, pointTop và pointBottom tương ứng với các điểm bên phải, bên trái, phía trên và phía dưới của quả bóng, và được dịch chuyển một khoảng cách 5 pixel để xác định hướng di chuyển của quả bóng.

****

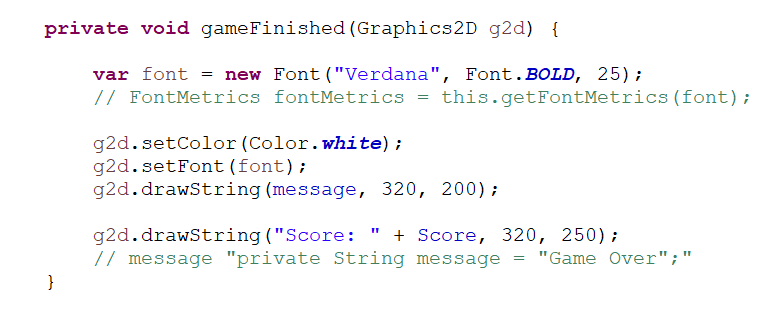
Khối lệnh if bên trong vòng lặp kiểm tra xem viên gạch thứ i đã bị phá hủy chưa. Nếu chưa bị phá hủy, vị trí của các điểm được kiểm tra để xác định hướng di chuyển của quả bóng sau khi va chạm với viên gạch.

Nếu điểm pointRight nằm trong hình chữ nhật bao quanh viên gạch thứ i, hướng di chuyển của quả bóng được đặt là hướng ngược lại (-Constants.SPEED\_BALL). Nếu điểm pointLeft nằm trong hình chữ nhật bao quanh viên gạch thứ i, hướng di chuyển của quả bóng được đặt là hướng dương (+Constants.SPEED\_BALL).

Tương tự, nếu điểm pointTop nằm trong hình chữ nhật bao quanh viên gạch thứ i, hướng di chuyển của quả bóng được đặt là hướng dương (+Constants.SPEED\_BALL). Nếu điểm pointBottom nằm trong hình chữ nhật bao quanh viên gạch thứ i, hướng di chuyển của quả bóng được đặt là hướng ngược lại (-Constants.SPEED\_BALL).

Sau đó, biến Timer\_Display\_Combo được đặt về giá trị 0 để chuẩn bị tính toán Combo\_Score. Combo\_Score được tăng lên một đơn vị và điểm số được tính toán dựa trên công thức: Score = 10 + Combo\_Score \* 10 - 10. Cụ thể, điểm số được tính bằng 10 điểm cộng với số lần liên tiếp đập được gạch (Combo\_Score), nhân với 10 điểm cho mỗi lần đập và trừ đi 10 điểm ban đầu.

## **6. gameFinished()**



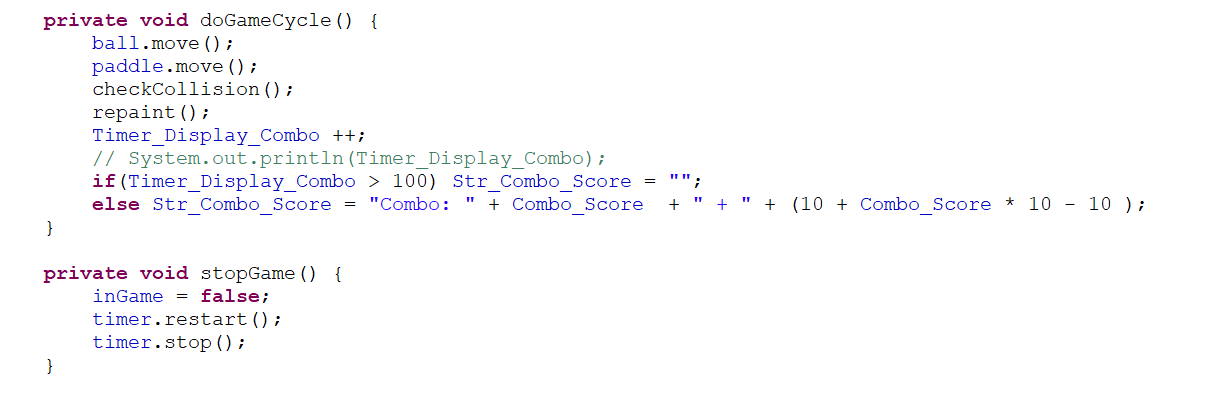
Phương thức này có một tham số là một đối tượng của lớp **Graphics2D,** được sử dụng để vẽ đồ họa trên một thành phần giao diện người dùng (GUI).

Phương thức này đầu tiên tạo một đối tượng Font mới với font Verdana, độ dày in đậm là 25. Tiếp theo, nó đặt màu của đối tượng Graphics2D thành màu trắng và đặt font hiện tại của đối tượng Graphics2D thành font vừa tạo.

Phương thức tiếp theo vẽ các chuỗi văn bản trên đối tượng Graphics2D. Cụ thể là nó vẽ chuỗi **message** (được khai báo như là một chuỗi private trong lớp) tại vị trí (320, 200) trên đối tượng Graphics2D. Sau đó, nó vẽ chuỗi "Score: " cộng với giá trị của biến Score, tại vị trí (320, 250) trên đối tượng Graphics2D.

Tóm lại, phương thức này được sử dụng để vẽ thông báo kết thúc trò chơi và điểm số của người chơi trên đối tượng Graphics2D.

## **7. doGameCycle() && stopGame()**



Phương thức **doGameCycle()** được gọi trong vòng lặp game để thực hiện một chu kỳ của game. Các bước trong chu kỳ này bao gồm:

Di chuyển quả bóng (ball.move()).

Di chuyển thanh chắn (paddle.move()).

Kiểm tra va chạm giữa quả bóng và thanh chắn cùng các viên gạch trên màn hình (check DiCollision()).

Vẽ lại toàn bộ màn hình (repaint()).

Tăng giá trị của biến **Timer\_Display\_Combo** lên 1 đơn vị, để đếm thời gian hiển thị thông tin combo trên màn hình.

Nếu giá trị của **Timer\_Display\_Combo** lớn hơn 100, chuỗi **Str\_Combo\_Score** sẽ được xóa, ngược lại, nó sẽ được cập nhật với thông tin combo và điểm số của người chơi.

Phương thức **stopGame()** được gọi khi trò chơi kết thúc. Trong phương thức này, biến **inGame** được đặt thành false để dừng trò chơi, sau đó timer được khởi động lại và dừng để chuẩn bị cho trò chơi tiếp theo.

Tóm lại, phương thức **doGameCycle()** được sử dụng để thực hiện các bước trong một chu kỳ của game, trong khi phương thức **stopGame()** được sử dụng để dừng trò chơi khi người chơi thua.

## **8. continueGame()**



Phương thức **continueGame()** được gọi khi người chơi muốn tiếp tục chơi sau khi đã thua một lần. Trong phương thức này, đầu tiên thanh chắn (paddle) và quả bóng (ball) sẽ được reset về vị trí ban đầu. Tiếp theo, một **KeyListener** mới sẽ được tạo ra để theo dõi phím được nhấn bởi người chơi.

Nếu người chơi nhấn phím mũi tên trái hoặc phím 'A', hướng di chuyển của quả bóng sẽ được đặt là hướng sang phải với tốc độ là **Constants.SPEED\_BALL.** Nếu người chơi nhấn phím mũi tên phải hoặc phím 'D', hướng di chuyển của quả bóng sẽ được đặt là hướng sang trái với tốc độ là **Constants.SPEED\_BALL.**

Cuối cùng, timer sẽ được dừng để chuẩn bị cho trò chơi tiếp theo. Tuy nhiên, mặc dù có một **KeyListener** được tạo ra, nhưng nó không được sử dụng bất cứ đâu trong phương thức này, vì vậy sẽ không có tác dụng gì khi người chơi nhấn phím.

# **IV. Xây dựng chương trình**

## **1. Khởi tạo trò chơi**

Để khởi tạo trò chơi Brick Breaker, chúng ta cần tạo một cửa sổ game và các đối tượng cần thiết cho trò chơi. Các đối tượng chính trong trò chơi này bao gồm thanh va chạm, bóng, các viên gạch và điểm số.

Dưới đây là một số bước cần thiết để khởi tạo trò chơi Brick Breaker:

1. Tạo một cửa sổ game với kích thước và tên phù hợp.
2. Tạo một đối tượng thanh va chạm và đặt vị trí ban đầu.
3. Tạo một đối tượng bóng và đặt vị trí ban đầu.
4. Tạo một danh sách các viên gạch và đặt vị trí ban đầu cho chúng.
5. Tạo một đối tượng điểm số và đặt giá trị ban đầu là 0.
6. Xử lý các sự kiện như di chuyển thanh va chạm, va chạm với bóng và các viên gạch, và tính điểm số khi phá hủy các viên gạch.
7. Vẽ các đối tượng trên cửa sổ game.

Sau khi hoàn thành các bước trên, trò chơi Brick Breaker đã sẵn sàng để chơi.

## **2. Xây dựng chương trình**

Để xây dựng một chương trình game Brick Breakout, ta có thể sử dụng ngôn ngữ lập trình Java và sử dụng thư viện đồ họa Swing.

Bước 1: Thiết kế giao diện Ta có thể sử dụng lớp **JPanel** để tạo một vùng đồ họa để hiển thị trò chơi. Ta cũng có thể sử dụng các thành phần như **JLabel, JButton** để thêm các phần tử như điểm số, số mạng còn lại, nút bắt đầu/ dừng trò chơi,...

Bước 2: Tạo các đối tượng game Ta cần tạo các đối tượng game như: thanh chắn, quả bóng, các viên gạch, nền của trò chơi,...

Bước 3: Xử lý sự kiện Ta cần lắng nghe các sự kiện từ người dùng như nhấn phím, click chuột để di chuyển thanh chắn. Ta cũng cần xử lý va chạm giữa các đối tượng game như quả bóng va chạm với các viên gạch, thanh chắn,...

Bước 4: Cập nhật trạng thái trò chơi Mỗi khi xử lý được một sự kiện từ người dùng hay xử lý được một va chạm giữa các đối tượng game, ta cần cập nhật lại trạng thái của trò chơi, ví dụ như thay đổi vị trí của quả bóng, số điểm, số mạng còn lại,...

Bước 5: Hiển thị trạng thái trò chơi Sau khi cập nhật trạng thái của trò chơi, ta cần vẽ lại các đối tượng game, điểm số, số mạng còn lại,... để hiển thị lên giao diện người dùng.

Bước 6: Kết thúc trò chơi Nếu số mạng còn lại của người chơi bằng 0 hoặc hết tất cả các viên gạch, ta cần dừng trò chơi và hiển thị điểm số của người chơi.

## **3. Bắt đầu trò chơi**

Sau khi khởi tạo trò chơi Brick Breaker, chúng ta cần bắt đầu trò chơi bằng cách thực hiện các bước sau:

1. Hiển thị cửa sổ game.
2. Tạo một vòng lặp vô hạn để cập nhật và vẽ các đối tượng trên cửa sổ game.
3. Trong vòng lặp, xử lý các sự kiện như di chuyển thanh va chạm và va chạm với bóng.
4. Kiểm tra xem trò chơi có kết thúc chưa (ví dụ: bóng rơi xuống đáy cửa sổ game).
5. Cập nhật điểm số và hiển thị lên cửa sổ game.
6. Vẽ lại các đối tượng trên cửa sổ game.
7. Nếu trò chơi kết thúc, hiển thị thông báo kết thúc trò chơi và cho phép người chơi chơi lại hoặc thoát khỏi trò chơi.

# **V. Hướng dẫn chạy mã lệnh trên một máy tính khác**

Để chạy mã lệnh trên một máy tính khác mà không cần cài đặt bất kỳ công cụ hoặc thư viện bổ sung nào ngoài JDK, bạn cần thực hiện các bước sau đây:

1. Sao chép toàn bộ mã lệnh của bạn vào một tệp văn bản mới. Bạn có thể sử dụng trình chỉnh sửa văn bản nào đó để làm điều này, chẳng hạn như Notepad.
2. Lưu tệp tin văn bản của bạn với định dạng .java. Chẳng hạn, nếu tên tệp của bạn là "MyProgram", bạn nên lưu nó dưới dạng "MyProgram.java".
3. Đảm bảo rằng JDK đã được cài đặt trên máy tính mà bạn muốn chạy mã lệnh của mình trên. Nếu chưa có, bạn cần tải xuống và cài đặt JDK từ trang web của Oracle.
4. Mở một cửa sổ dòng lệnh trên máy tính đó. Bạn có thể làm điều này bằng cách nhấn phím Windows + R và nhập "cmd" vào hộp thoại Run.
5. Điều hướng đến thư mục chứa tệp tin văn bản của bạn bằng cách sử dụng lệnh "cd" trong dòng lệnh. Ví dụ: nếu tệp tin của bạn nằm trong thư mục C:\MyPrograms, bạn có thể nhập "cd C:\MyPrograms" để điều hướng đến thư mục đó.
6. Sử dụng lệnh "javac" để biên dịch mã lệnh của bạn. Ví dụ: nếu tên tệp tin của bạn là "MyProgram.java", bạn có thể nhập "javac MyProgram.java" để biên dịch chương trình của mình.
7. Sau khi mã lệnh của bạn đã được biên dịch thành công, sử dụng lệnh "java" để chạy chương trình của bạn. Ví dụ: nếu tên của lớp chứa phương thức main trong chương trình của bạn là "MyProgram", bạn có thể nhập "java MyProgram" để chạy chương trình đó.

# **VI. Link Github**

https://github.com/nguyendangduc2003/Brick\_Break\_Out\_Game.git