HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Khoa Đa Phương Tiện

BÁO CÁO GIỮA KÌ

NGÔN NGỮ LẬP TRÌNH JAVA

Đề tài : "Lập trình game 2D-MegamanX"

Giảng viên hướng dẫn	**:**	**Vũ Hữu Tiến**
Nhóm môn học	**:**	**01**
Tổ bài tập	**:**	
Nhóm sinh viên – Mã sinh viên	**:**	**Hoàng Văn Hùng – B21DCPT123**
		Vũ Thanh Phong – B21DCPT183

HÀ NỘI - 2023

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Khoa Đa Phương Tiện

BÁO CÁO GIỮA KÌ

NGÔN NGỮ LẬP TRÌNH JAVA

Đề tài : "Lập trình game 2D-MegamanX"

Giảng viên hướng dẫn : Vũ Hữu Tiến

Nhóm môn học : 01

Tổ bài tập :

Nhóm sinh viên – Mã sinh viên : Hoàng Văn Hùng – B21DCPT123

Vũ Thanh Phong – B21DCPT183

HÀ NỘI - 2023

MỤC LỤC

LỜI NÓI ĐẦU

Lời đầu tiên, nhóm sinh viên xin được gửi lời cảm ơn sâu sắc **đến Học viện Công nghệ Bưu chính Viễn thông** đã đưa học phần **Ngôn ngữ lập trình Java** vào trong chương trình giảng dạy. Đặc biệt, chúng em xin gửi lời cảm ơn chân thành nhất đến giảng viên bộ môn – thầy **Vũ Hữu Tiến**, vì đã truyền đạt những kiến thức quý báu trong suốt thời gian học tập vừa qua.

Bài báo cáo của nhóm khó tránh những thiếu sót, vậy nên nhóm rất mong nhận được sự thông cảm, cũng như đóng góp chân thành của quý thầy cô để báo cáo của nhóm được hoàn thiện hơn ! Nhóm hi vọng sau báo này có thể rút được những kinh nghiệm quý báu cho những môn học lập trình sắp tới.

Chúng em xin chân thành cảm ơn!

Hà Nội, ngày 18 tháng 11 năm 2023

Nhóm sinh viên thực hiện

Hoàng Văn Hùng- B21DCPT123

Vũ Thanh Phong - B21DCPT183

CHƯƠNG 1 : MỞ ĐẦU
(Xác định các yêu cầu)

I. Lý do lựa chọn đề tài.

Trong những năm gần đây, nền công nghệ thông tin nước ta đã có nhiều phát triển trên mọi lĩnh vực trong cuộc sống cũng như trong lĩnh vực quản lý xã hội khác. Một trong những lĩnh vực được nhiều người quan tâm là lĩnh vực giải trí. Game vừa là một cách giải trí, giúp chúng ta có khả năng tư duy, đầu óc nhạy bén. Ngoài ra, game còn giúp con người tập trung cao độ hơn, rèn luyện tính quyết tâm, kiên trì, tích lũy kinh nghiệm để xử lí vấn đề...

Nhóm đã chọn đề tài ***"Lập trình game 2D-MegamanX"*** . Đây là một dạng game đã xuất hiện lâu trên thế giới, nhưng ưu điểm của Game mà nhóm đã thiết kế là tốc độ nhanh, không yêu cầu cấu hình mạnh, lối chơi đơn giản nhưng không nhàm chán, thân thuộc và dễ cuốn hút người dùng.

II. Tổng quan.

1. Giới thiệu tổng quan về game MegaManX.

MegaManX là một tựa game hành động và phưu lưu nổi tiếng do Capcom phát triểu và phát hành. Được phát hành vào năm 1993 cho hệ máy trò chơi Super Nintendo Entertaiment System (SNES), MegaMan X là phần mở rộng và cải tiến từ seires game MegaMan gốc.

Cốt truyện:

MegaMan X đặt người chơi vào vai một Robot có khả năng cải tiến và học hỏi. Trong một thế giới nơi các Robot được gọi là "Reploids" tồn tại, có sự phân biệt giữa các Reploid tốt và xấu. X là một Robot chiến binh mạnh mẽ, cùng Zero chống lại các thế lực xấu xa, ngăn cản kế hoạch của chúng tiêu diệt loài người và chiếm đoạt thế giới.

Đặc điểm GamePlay:

Hành đông và nhanh nhẹn: MegaMan X kế thừa phong cách gameplay hành động nhanh nhẹn từ series gốc với đặc điểm chính là nâng cấp và năng lực thông qua quá trình chiến đấu.

Chọn lựa các chế độ: Người chơi có thể lựa chọn chế độ mà họ muốn khám phá. Mỗi chế độ có một Reploid boss cuối cùng mà X và Zero cần đánh bại.

Đồ họa: MegaMan X sử dụng đồ họa 8-bit, mục đích để tái hiện lại những trải nghiệm của người chơi so với bản gốc.

Hình 1.1. Ảnh nhân vật MegaMan gốc

Hình 1.2. Ảnh nhân vật MegaMan X

MegaMan X được đánh giá cao hơn về đồ họa. Nó đã tạo ra một sự tiếp nối thành công cho series MegaMan. Từ đó MegaMan X đã phát triển với các phần tiếp theo, mở rộng thêm nội dung và cải thiện gameplay, vẫn tiếp tục thu hút được sự quan tâm của người chơi yêu thích thể loại game hành động.

2. Mục đích của lập trình game MegaMan X.
   Học tập và nâng cao kỹ năng lập trình:
   Một trong những múc đích chính có thể là đê thực hành và cải thiện kỹ năng lập trình của nhóm. Việc phát triển một trò chơi tương đối khó như MegaMan có thể giúp nhóm nâng cao kỹ năng lập trình, thiết kế game, quản lý dự án.
   Áp dụng kiến thức học tập:
   Nhóm áp dụng những kiến thức đã được học từ giảng viên hướng dẫn và tìm hiểu thêm các video hướng dẫn lập trình ở trên Internet. Các kiến thức đã được giảng viên hướng dẫn đã được áp dụng vào trong dự án game MegaMan X này. Bao gồm:
   - Lớp và đối tượng trong Java.
   - Tính kế thừa trong Java.
   - Xử lí nhập / xuất trong Java.
   - Xử lí ngoại lệ Java.
   - Xử lí đa luồng trong Java.
   - Lập trình giao diện trong Java.

   Hướng đến sáng tạo và đổi mới:

   Nhóm tạo ra một trò MegaMan X mới, mang sự độc đáo khi có sự xuất hiện của chế độ MultiPlayer, cùng với giao diện bắt mắt hơn nhằm đem lại trải nghiệm mới cho người chơi.

III. Mục tiêu

1. Hoàn thiện và phát triển các kỹ năng.
   Mục tiêu chính là cải thiện và áp dụng kiến thức lập trình vào thực tế, bao gồm việc sử dụng ngôn ngữ lập trình đã được nêu ở trên, thiết kế giao diện thông qua Figma.
2. Tạo ra trò chơi đảm bảo đúng tính chất Gameplay của tự game ban đầu.
3. Hiểu rõ về quy trình phát triển game.
   Hiểu rõ về quy trình phát triển game, từ việc lên ý tưởng, thiết kế giao diện, lập trình, kiểm thử, hoàn thiện sản phẩm.
4. Tăng cường kỹ năng làm việc nhóm.
   Đảm bảo được hiệu quả khi làm việc nhóm, quản lí dự án, phân chia công việc, đảm bảo tương tác phối hợp giữa các thành viên.

5. Tạo ra sản phẩm nhỏ để thêm vào portfolio phục vụ nhu cầu sau khi ra trường.

IV. Xác định yêu cầu thiết kế Game

1. **Yêu cầu về Game**.

1.1. GamePlay.

- MegaMan X được xây dựng 2 chế độ chơi, đó là Single Player Mode (chế độ một người chơi) và Multi Player Mode (chế độ hai người chơi).
- Với chế độ Single Mode, người chơi có thể lựa chọn giữa hai nhân vật đó là X và Zero.
- Với chế độ Multi Player Mode, Player1 sẽ điều khiển Zero và Player2 điều khiển Zero.
- Game cần được đảm bảo răng cấu trúc gameplay cho mỗi chế độ là linh hoạt để người chơi có thể lựa chọn và tận hương trải nghiệm một cách tốt nhất.

1.2. Character Design (Thiết kế nhân vật).

- Thế giới trò chơi MegaMan được xây dựng dựa trên các Robot chia thành các phe khác nhau, bao gồm phe phản diện gồm có các robot phản diện và các boss cuối (Final Boss), và phe anh hùng gồm có MeganMan X và Zero.
- Với sự xuất hiện của hai phe khác nhau, cần phải đảm bảo cân bằng giữa các nhân vật để trải nghiệm chơi game không bị lệch về một phía gây nhàm chán cho người chơi.

1.3. Level Design (Thiết kế mức độ).

- Thiết kế các màn chơi với sự phức tạp và thách thức phù hợp cho cả 2 chế độ chơi đơn và chế độ chơi 2 người.
- Với chế độ chơi đơn, số lượng robot phản diện sẽ được giảm bớt đi, đồng thời chỉ có 1 FinalBoss. Còn với chế đố 2 người chơi, số lượng robot phản diện tăng, đồng thời có 2 FinalBoss.

1.4. Tương tác giữa người chơi với người chơi.

- Xác định và thiết kế cách mà hai người chơi tương tác với nhau trong trò chơi. Trong MegaManX, hai người chơi tương tác với nhau trong vai trò là hợp tác cùng nhau chiến đấu.

1.5. Tương tác giữa người chơi và máy.

Người chơi tương tác với máy bằng cách nhấn chuột để lựa chọn chế độ, dùng bàn phím để trải nghiệm chơi game.

- Chế độ một người chơi:
  Di chuyển: sử dụng các nút A W S D để di chuyển.
  Tấn công: Space, L;
- Chế độ hai người chơi
  - Player1:
    Di chuyển: Sử dụng A W S D để di chuyển.
    Tấn công: Space, L;
  - Player2:
    Di chuyển: Sử dụng A W S D để di chuyển.
    Tấn công: 0,Enter,+;

2. Yêu cầu về giao diện người dùng (UI).

2.1 Menu chính.

- Xây dựng menu chính một cách trực quan và dễ dàng sử dụng cho cả chế độ đơn và chế dộ đôi cũng như lựa chọn nhân vật.
- Bao gồm các Button chính: Single(chế độ đơn), Mul (chế độ 2 người), Map, Mission(Nhiệm vụ), Equid (Trang bị), Character(Lựa chọn nhân vật), Lab (Phòng thí nghiệm).

2.2 Chế độ chơi.

- Tạo ra các nút bấm (Button) để người chơi dễ dàng chuyển đổi giữa hai chế độ chơi một cách dễ dàng.
- Hiển thị thông tin chế độ chơi một cách rõ ràng bằng cách setIcon cho từng chế độ riêng biệt.

2.3 Giao diện người chơi.

- Thiết kế giao diện cho mỗi người chơi khi chơi 2 người để hiển thị thông tin về thanh máu, nội năng, số mạng còn lại.
- Đảm bảo thông tin của mỗi người chơi được hiển thị rõ ràng và dễ nhìn

2.4 Thiết kế thông minh.

- Đảm bảo răng giao diện người dùng không quá rối mắt và phức tạp, tối ưu hóa việc sử dụng màn hình và không gian hiển thị.
- Sử dụng màu sắc, biểu tượng và đồ họa phù hợp để làm nổi bật thông tin quan trọng và dễ nhận biết. Bao gồm sự khác biệt của thanh máu của người chơi và boss, icon nhân vật người chơi và icon Final Boss.
- Hiệu ứng Hover khi di chuột vào Button.

Như vậy, việc tạo ra các Menu dễ sử dụng cho đến việc cung cấp thông tin chi tiết về từng chế độ chơi, việc thiết kế giao diện người dùng trò chơi MegaMan X sẽ giúp người chơi dễ dàng chuyển đổi giữa chế độ đơn và đôi một cách thuận tiện.

V. Kiến thức kỹ thuật.

1. Ngôn ngữ lập trình Java.
   - Nhóm sử dụng ngôn ngữ lập trình Java vì nó tương đối phổ biến và được sử dụng khá rộng rãi cũng như liên quan trực tiếp đến môn học. Java với các đặc điểm nổi bật như:
   - Tài nguyên học tập chất lượng cao: Được ra mắt lâu nên tài liệu liên quan cũng như hướng dẫn chi tiết có thể tìm kiếm được trên Internet.
   - Các chức năng và thư viện sẵn có: Khi sử dụng Java không cần phải viết mọi chức năng mới, thay vào đó java cung cấp một hệ sinh thái phong phú gồm các chức năng và thư viện sẵn có để phát triển hàng loạt ứng dụng đa dạng.
   - Độc lập với nền tảng: Java chạy được trên nhiều nền tảng khác nhau như Window, MacOS, Linux..
2. Xử lí sự kiện và điều khiển.
   Lập trình xử lí các sự kiện cho các hành động như: KeyEvent, Mount_Click, Mount_Enter, Mount_Exit.
3. Đồ họa và thiết kế giao diện người dùng.
   - Sử dụng Photoshop, Illutrator, để thiết kế các Button.
   - Sử dụng Figma để thiết kế các Frame, import các ảnh Button, xây dựng Prototype cho game.
4. Gameplay và Logic.
   - Xây dựng các cơ chế gameplay như di chuyển, tấn công, va chạm và các yếu tố khác.

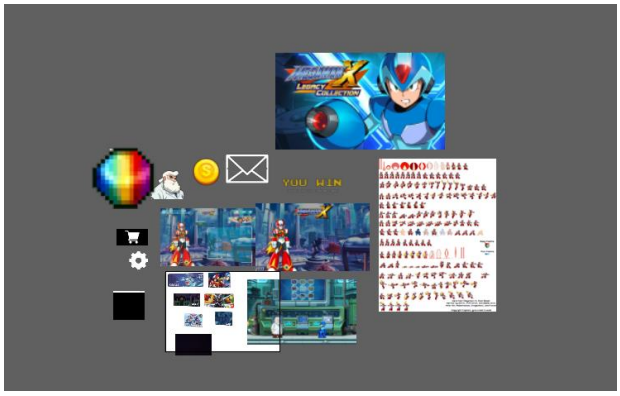
- Tính toán, xử lí logic cho các sự kiện trong trò chơi như va chạm, kết quả và phản ứng của người chơi.

5. Kỹ thuật đồ họa.
   - Các kỹ thuật vẽ đồ họa như drawImage để vẽ một khung hình, từ đó xây dựng Animation bằng các vẽ liên tiếp các khung hình.
   - Xử lí âm thanh, hiệu ứng âm thanh.

CHƯƠNG 2: THIẾT KẾ

I. Chuẩn bị dữ liệu.

Tìm dữ liệu cho quá trình thiết kế trên Internet.



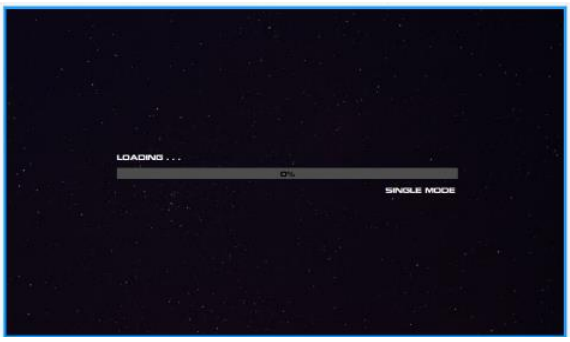
Hình 1.3. Quá trình chuẩn bị và xử lí dữ liệu cho quá trình thiết kế.

II. Thiết kế giao diện Game và các màn.

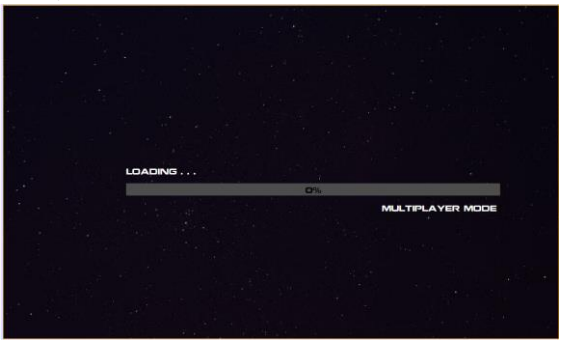
Hình 1.4. Thiết kế giao diện Loading_Game.

Hình 1.5. Thiết kế giao diện Loading_Game.

Hình 1.6: Thiết kế giao diện Menu chính.



Hình 1.7: Màn hình Loading vào GameMode(1)



Hình 1.8: Màn hình Loading vào GameMode(2)

Hình 1.9: Màn hình Turtorial.

Hình 1.10. Màn hình GamePlay(1)

Hình 1.11. Màn hình GamePlay(2)

Hình 1.12: Màn hình gặp Final Boss(2)

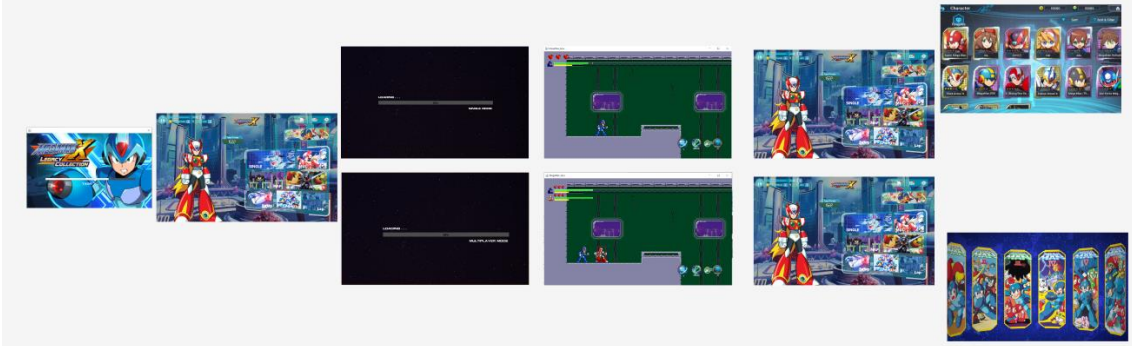
Hình 1.13: Màn hình gặp Final Boss(1)

Hình 1.14: Màn hình GameWin

Hình 1.14: Màn hình Game Over

Hình 1.15: Lựa chọn nhân vật

Hình 1.16: Màn hình lựa chọn Mission

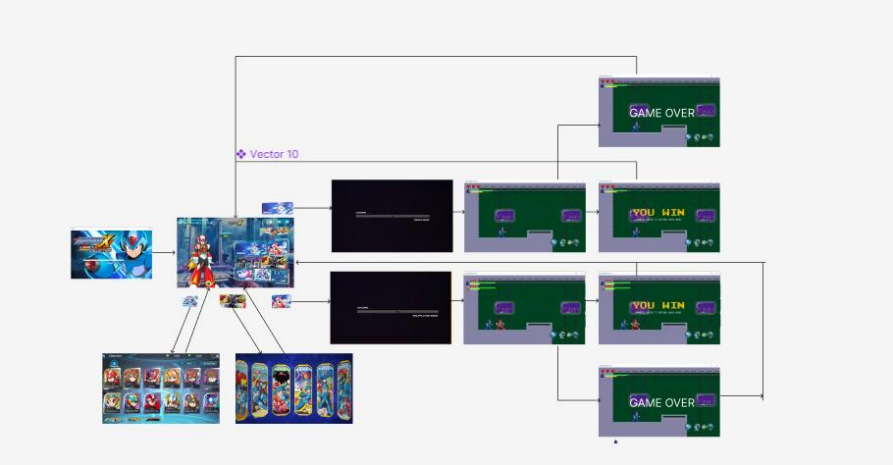


Hình 1.17: Tổng hợp giao diện Game.

Sau khi thiết kế xong giao diện của các chắc năng, sử dụng Figma để xây dựng Prototye và căn chỉnh kích thước Frame chuẩn để chuẩn bị cho giai đoạn phát triển game.

CHƯƠNG 3: PHÁT TRIỂN GAME

I. Phát triển giao diện game.

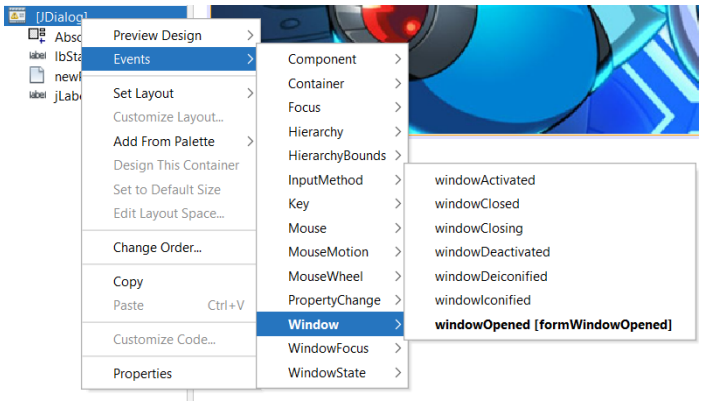


Hình 3.0. Mô tả kịch bản giao diện game

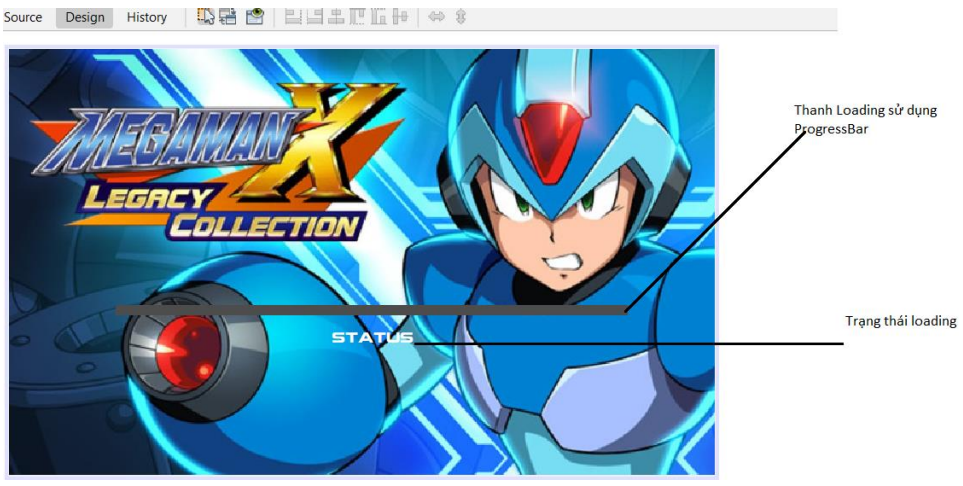
1. Giao diện Loading Game (Hình 1.4).

Sử dụng JDiaLog để tạo ra màn hình game với kích thước đặt ra là 667x400px.
Cài đặt Layout: Absolute layout.
Xét sự kiện (Event) JDialog: Window-WindowOpened.

Hình 3.1. Xét sự kiện cho JDialogFrame



Sử dụng Jlabel và setIcon để tạo ra BackGround cho màn hình.
Sử dụng class ProgressBarCusTom để tạo thanh Loading cho màn hình.
Tạo một Jlabel để cài đặt trạng thái Loading (Task1,2,3….).
Tạo một phương thức doTask để cài status và setValue cho ProgressBar.



Hình 3.2. Màn hình loading.

Phương thức doTask:

```
private void doTask(String name,int progress) throws Exception{
    lbStatus.setText(text: name);
    Thread.sleep( millis:500);
    newProgressBarCustom1.setValue(n: progress);

}
```

Xử lí sự kiện WindowOpened:

```
private void formWindowOpened(java.awt.event.WindowEvent evt) {
  new Thread(new Runnable() {
      @Override
      public void run() {
          try {
                doTask(name: "Connect To Database ...", progress: 10);
                doTask(name: "Task 1 ...", progress: 20);
                doTask(name: "Task 2 ...", progress: 30);
                doTask(name: "Task 3 ...", progress: 40);
                doTask(name: "Task 4 ...", progress: 50);
                doTask(name: "Task 5 ...", progress: 60);
                doTask(name: "Task 6 ...", progress: 70);
                doTask(name: "Task 7 ...", progress: 80);
                doTask(name: "Task 8 ...", progress: 90);
                doTask(name: "Task 9 ...", progress: 100);
                doTask(name: "Done ...", progress: 100);
                dispose();
          } catch (Exception e) {
          }

      }
  }).start();
    // TODO add your handling code here:
}
```

Status

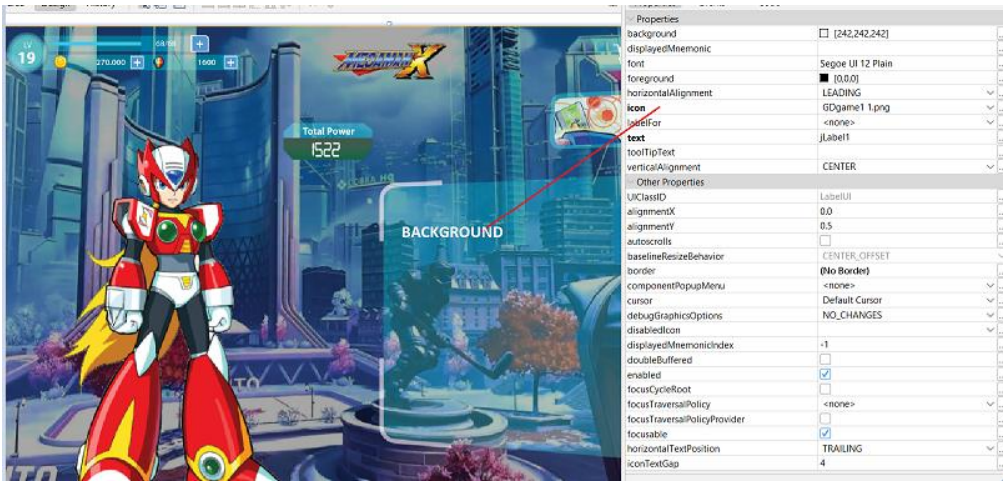
Value

Chạy phương thức doTask.

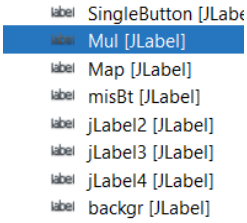
Đóng cửa sổ khi Value đạt đến 100

Giải thích: Trong phương thức doTask, Jlabale chứa trạng thái sẽ lấy text và xét giá trị cho ProgressBar ở trong **formWindowOpened,** cứ sau 500ms sẽ thực hiện lại phương thức doTask tiếp theo.

2. Màn hình Menu chính của Game.
   Tiếp tục sử dụng JDialog để tạo ra Menu game, cài kích thước là 1000x600px.
   Sử dụng Jlabel và setIcon để tạo background cho Menu.



Tương tự như setIcon của Background, dùng thêm các Jlabel để tạo ra các Button của menu.



Xét Event cho từng Button:

- Event Mouse_MouseEnter và Mouse_Exit:

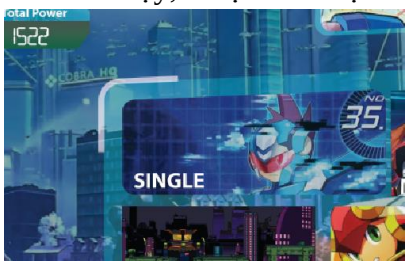
Xử lí sự kiện MouseEntered:

```
private void SingleButtonMouseEntered(java.awt.event.MouseEvent evt) {
    SingleButton.setIcon(new ImageIcon(location: this.getClass().getResource(name: "SignleButtom_test.png")));
    // TODO add your handling code here:
}
```

Xử lí sự kiện MouseExited:

```
private void SingleButtonMouseExited(java.awt.event.MouseEvent evt) {
    SingleButton.setIcon(new ImageIcon(location: this.getClass().getResource(name: "Single 1.png")));
    // TODO add your handling code here:
}
```

Giải thích: Khi ta chỉ con trỏ chuột đến vị trí của Jlabel, Jlabel đó sẽ được setIcon mới và khi con trỏ chuột rời khỏi vị trí của Jlabel đó, Jlabel sẽ được setIcon ban đầu. Chính vì vậy, sẽ tạo nên hiệu ứng Hover.



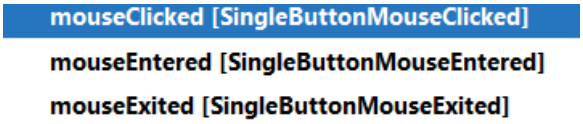
Hình 3.3. setIcon mặc định

Hình 3.4. setIcon của hiệu ứng Hover

Tương tự với các Button khác, ta thu được:

Hình 3.5. Menu hoàn chỉnh khi thêm hiệu ứng Hover

- Event Mouse_Clicked:



Hình 3.6. Các sự kiện Mouse_Event của Button.

Xử lí sự kiện Mouse_Clicked:

```
private void SingleButtonMouseClicked(java.awt.event.MouseEvent evt) {
    dispose();
    Thread thread1 = new Thread(() -> {
        // Lệnh thực hiện trong luồng thứ nhất
        try {
            new ManHinhLoadingSm(parent:null, modal: true).setVisible(b: true);

        } catch (Exception e) {
        }

    });
    thread1.start();

}
```

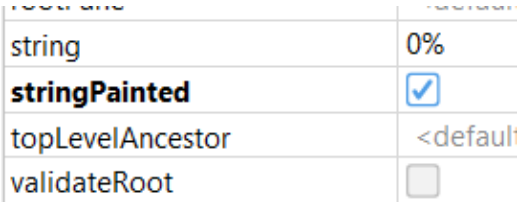
Đóng cửa sổ

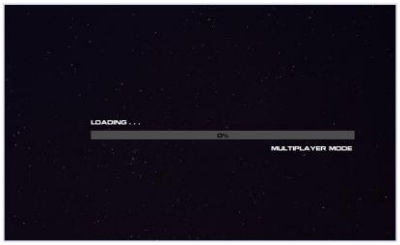
Màn hình mới xuất hiện.

Xử lí đa luồng nếu có

Giải thích: Khi người chơi ấn chuột vào một Button trong Menu, màn hình Menu sẽ được đóng lại, thay vào đó là một màn hình khác.
Ví dụ: với Button_Single khi ấn sẽ chuyển sang Màn hình Loading để vào màn chơi.

3. Màn hình Loading vào màn chơi.
Sử dụng Jdialog để tạo ra màn hình Loading vào chế đội chơi. Cài đặt kích thước 1000x600px.
Sử dụng Jlabel và setIcon để tạo background tương tự như màn hình Menu. Sử dụng các Jlabel khác và Edit Text để tạo chữ trên màn hình.
Sử dụng ProgressBar để tạo thanh Loading đồng thời bật chế độ StringPaint để hiện Value của ProgressBar.





Xét sự kiện cho Jdialog, Event **Window_WindowOpen**:
Tương tự như màn hình Loading đầu game, ta setValue cho ProgressBar:

Gọi phương thức Chay():

```
private void Chay(int i) throws Exception{

    while(i<=100){
        i+=5;
        progressBarCustom1.setValue(n: i);
        try {
            Thread.sleep(millis:190);
        } catch (Exception e) {
        }
    }
```

cài giá trị cho ProgressBar

Xử lí sự kiện WindowOpen:

```
private void Chay(int i) throws Exception{

    while(i<=100){                                  Value i tăng từ 0-100
        i+=5;
        progressBarCustom1.setValue(n: i);
        try {
            Thread.sleep(millis:190);
        } catch (Exception e) {
        }
    }

    dispose();                                      khi i>=100 đóng màn hình
    GameFrame_Multi_Player_Mode gameFrame = new GameFrame_Multi_Player_Mode();
    gameFrame.setTitle(title: "MegaMan_Java");
    gameFrame.setVisible(b: true);                  chuyển sang màn chơi
    gameFrame.startGame();                          game

}
```

Giải thích: Khi màn hình khởi động, thanh ProgressBar được gán giá trị từ 0 đến 100, khi giá trị i chưa nhỏ hơn 100 thì tiếp tục tăng và tiếp tục gán, đến khi giá trị của i =100 thì dừng lại, đóng và chuyển sang màn hình chơi game

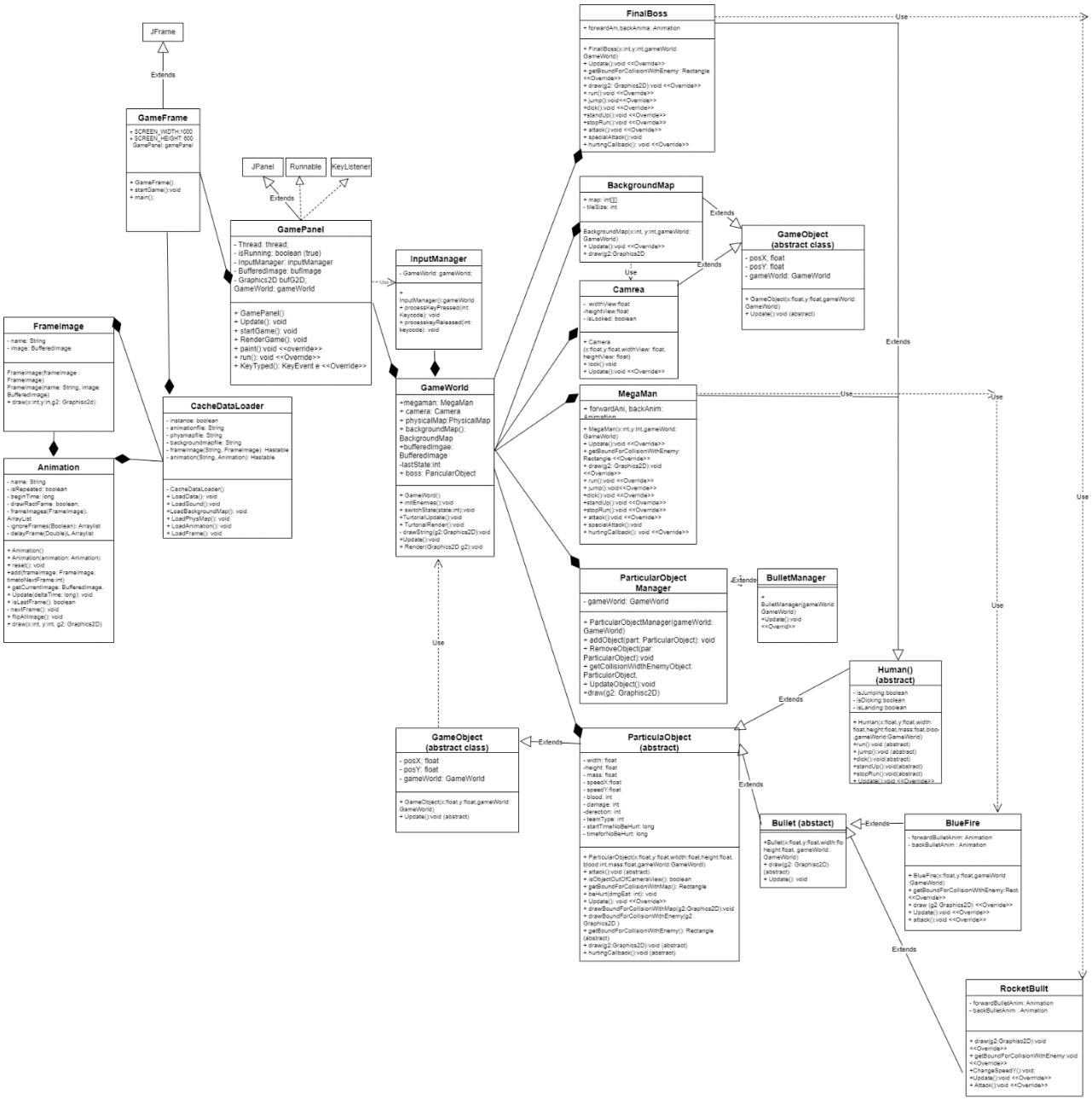
4. Màn hình chọn Character.
   Tương tự như các Button trên, ta sử dụng Jdialog và Jlabel để tạo ra background, cài kích thước 1000x600px.
   Đồng thời setIcon cho Jlable để có đc Background.
   Thêm các Jlabel khác và setIcon để có Button nhấn chọn vào nhân vật.
   Thêm hiệu ứng Hover tương tự như cách làm của Menu.
   Xử lí sự kiện Mouse_Click, Mouse_Enter, Mouse_Exited tương tự như Menu.
   Xử lí sự kiện:

```
private void jLabel2MouseEntered(java.awt.event.MouseEvent evt) {
    jLabel2.setIcon(new ImageIcon(location: this.getClass().getResource(name: "Zere_Bt 1.png")));
}

private void jLabel2MouseExited(java.awt.event.MouseEvent evt) {
    jLabel2.setIcon(new ImageIcon(location: this.getClass().getResource(name: "Zere_BtFN.png")));
    // TODO add your handling code here:
}

private void jLabel3MouseEntered(java.awt.event.MouseEvent evt) {
    jLabel3.setIcon(new ImageIcon(location: this.getClass().getResource(name: "MegaX_Buttom 1.png")));
    // TODO add your handling code here:
}

private void jLabel3MouseExited(java.awt.event.MouseEvent evt) {
    jLabel3.setIcon(new ImageIcon(location: this.getClass().getResource(name: "MegaX_ButtomFN.png")));
}
```

5. Màn hình lựa chọn Mission.
   Xử lí tương tự như mấy phần trên, về tạo Jdialog, xử lí sự kiện cho từng Button.

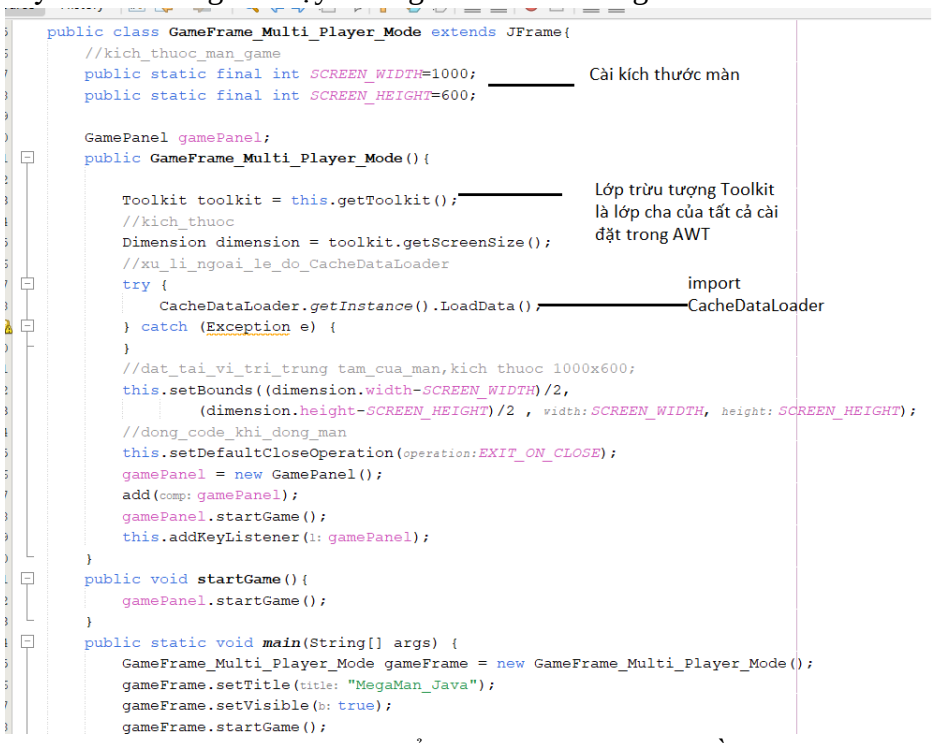
II. Phát triển GamePlay.

Xây dựng sơ đồ Class



Từ sơ đồ class đượ xây dựng như ở trên việc triển khai và quản lí các class được diễn ra một cách đơn giản.

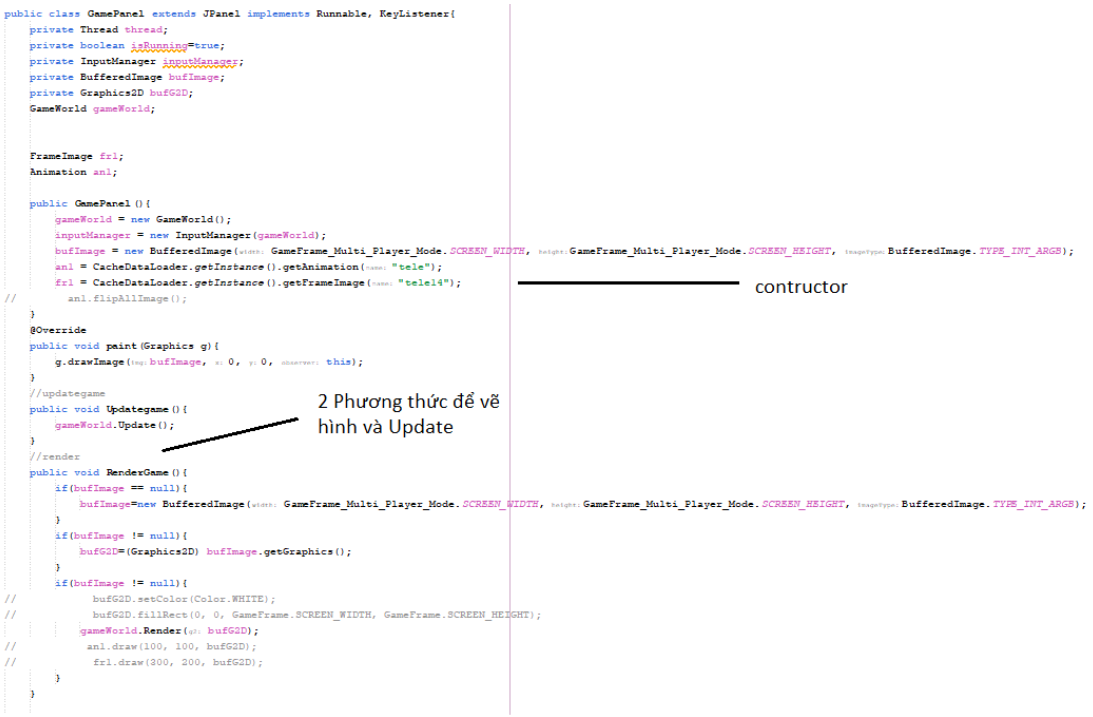
1. GameFrame:
   Đây là class dùng để chạy trương trình chính của game.




```
public class GameFrame_Multi_Player_Mode extends JFrame{
    //kich_thuoc_man_game
    public static final int SCREEN_WIDTH=1000;
    public static final int SCREEN_HEIGHT=600;

    GamePanel gamePanel;
    public GameFrame_Multi_Player_Mode(){

        Toolkit toolkit = this.getToolkit();
        //kich_thuoc
        Dimension dimension = toolkit.getScreenSize();
        //xu_li_ngoai_le_do_CacheDataLoader
        try {
            CacheDataLoader.getInstance().LoadData();
        } catch (Exception e) {
        }
        //dat_tai_vi_tri_trung tam_cua_man,kich thuoc 1000x600;
        this.setBounds((dimension.width-SCREEN_WIDTH)/2,
                (dimension.height-SCREEN_HEIGHT)/2 , width: SCREEN_WIDTH, height: SCREEN_HEIGHT);
        //dong_code_khi_dong_man
        this.setDefaultCloseOperation(operation:EXIT_ON_CLOSE);
        gamePanel = new GamePanel();
        add(comp: gamePanel);
        gamePanel.startGame();
        this.addKeyListener(l: gamePanel);
    }
    public void startGame(){
        gamePanel.startGame();
    }
    public static void main(String[] args) {
        GameFrame_Multi_Player_Mode gameFrame = new GameFrame_Multi_Player_Mode();
        gameFrame.setTitle(title: "MegaMan_Java");
        gameFrame.setVisible(b: true);
        gameFrame.startGame();
```

Giải thích: Class GameFrame để tạo ra mà hình chơi, đồng thời vị trí setBound để cài đặt vị trí và kích thước cho Frame, đặt cho màn hình của GameFrame ở vị trí trung tâm của màn hình máy tính.

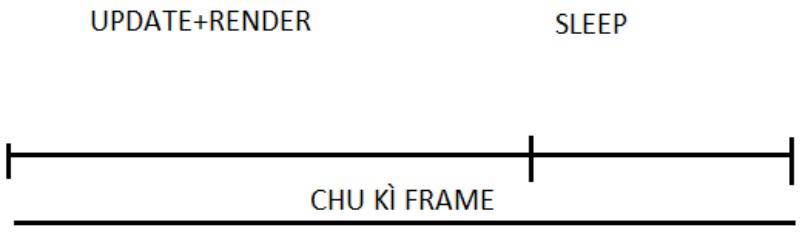
2. GamePanel:




```
public class GamePanel extends JPanel implements Runnable, KeyListener{
    private Thread thread;
    private boolean isRunning=true;
    private InputManager inputManager;
    private BufferedImage bufImage;
    private Graphics2D bufG2D;
    GameWorld gameWorld;


    FrameImage frl;
    Animation anl;

    public GamePanel (){
        gameWorld = new GameWorld();
        inputManager = new InputManager(gameWorld);
        bufImage = new BufferedImage(width: GameFrame_Multi_Player_Mode.SCREEN_WIDTH, height: GameFrame_Multi_Player_Mode.SCREEN_HEIGHT, imageType: BufferedImage.TYPE_INT_ARGB);
        anl = CacheDataLoader.getInstance().getAnimation(name: "tele");
        frl = CacheDataLoader.getInstance().getFrameImage(name: "tele14");
//        anl.flipAllImage();
    }
    @Override
    public void paint(Graphics g){
        g.drawImage(img: bufImage, x: 0, y: 0, observer: this);
    }
    //updategame
    public void Updategame(){
        gameWorld.Update();
    }
    //render
    public void RenderGame(){
        if(bufImage == null){
            bufImage=new BufferedImage(width: GameFrame_Multi_Player_Mode.SCREEN_WIDTH, height: GameFrame_Multi_Player_Mode.SCREEN_HEIGHT, imageType: BufferedImage.TYPE_INT_ARGB);
        }
        if(bufImage != null){
            bufG2D=(Graphics2D) bufImage.getGraphics();
        }
        if(bufImage != null){
//            bufG2D.setColor(Color.WHITE);
//            bufG2D.fillRect(0, 0, GameFrame.SCREEN_WIDTH, GameFrame.SCREEN_HEIGHT);
            gameWorld.Render(g: bufG2D);
//            anl.draw(100, 100, bufG2D);
//            frl.draw(300, 200, bufG2D);
        }
    }
```

```
public void startGame(){
    if(thread == null){
        thread = new Thread( target: this);
        thread.start();
    }
}
@Override
public void run(){
    long FPS = 120;
    long period=1000*1000000/FPS;
    long beginTime;
    long sleepTime;
    int a =1;
    beginTime=System.nanoTime();
    while(isRunning){
        //Update
        //Render
        Updategame();
        RenderGame();

        repaint();
        anl.Update( deltaTime: System.nanoTime());

        long deltaTime = System.nanoTime()-beginTime;
        sleepTime=period - deltaTime;
        try {
            if(sleepTime >0){
                Thread.sleep(sleepTime/1000000);
            }
            else Thread.sleep(period/2000000);

        } catch (InterruptedException ex) {
        }
        beginTime = System.nanoTime();


    }
}
```

Giải thích: Trong quá trình thực hiện, mỗi Frame đều diễn ra 2 quá trình chính là Update, Render (draw) và thời gian nghỉ (sleep), để đảm bảo cho quá trình Render và Update mượt mà, thực hiện quá trình nghỉ của hệ thống, deltaTime chính là thời gian mà thực hiện 2 quá trình Update và Render. deltaTime được tính bằng thời gian hiện tại trừ đi thời gian khởi tại trong đó System.nanatime() chính là thời gian của máy tính. Thời gian ngủ chính bằng thời gian chu kì period – deltaTime



Tuy nhiên sẽ có thời gian Update và Render sẽ vượt qua chu kì Frame thì ta sẽ cho hệ thống một khoảng thời gian ngắn hơn để đảm bảo game không bị delay.

```
@Override
public void keyTyped(KeyEvent e) {
}

@Override
public void keyPressed(KeyEvent e) {
    inputManager.processKeyPressed( keyCode: e.getKeyCode());

}

@Override
public void keyReleased(KeyEvent e) {
    inputManager.processkeyReleased( keyCode: e.getKeyCode());
}
```

Do implement KeyEvent vì vậy có các phương thức của KeyEvent.

3. InputManager

Lớp này có chức năng quản lí, xử lí dữ liệu mà người dùng nhập từ bàn phím.
Có 2 phương thức chính đó là ProcessKeyPress (nhấn phím) và ProcessKeyRelease (thả phím).
Với KeyPress: Điều khiển quá trình di chuyển của nhân vật và tấn công. Sử dụng switch-case để xét trường hợp nhập phím.

```
public void processKeyPressed(int keyCode){
    switch (keyCode) {
        case KeyEvent.VK_A:
            gameWorld.megaman.setDirection(dir:MegaMan.LEFT_DIR);
            gameWorld.megaman.run();
            break;
        case KeyEvent.VK_W:
            gameWorld.megaman.jump();


            break;
        case KeyEvent.VK_D:
            gameWorld.megaman.setDirection(dir:MegaMan.RIGHT_DIR);
            gameWorld.megaman.run();
            break;
        case KeyEvent.VK_S:
            gameWorld.megaman.dick();
            break;
        case KeyEvent.VK_ESCAPE:
            break;
        case KeyEvent.VK_ENTER:
            if(gameWorld.state == GameWorld.INIT_GAME){
                if(gameWorld.previousState == GameWorld.GAMEPLAY)
                    gameWorld.switchState(state: GameWorld.GAMEPLAY);
                else gameWorld.switchState(state: GameWorld.TUTORIAL);
            }
```

Với KeyRelese: Xét sự kiện nhả phím của người chơi. Dùng để dừng hành động chạy (stopRun), ngồi của nhân vật (standUp):

```
}
public void processkeyReleased(int keyCode){
    switch (keyCode) {
        case KeyEvent.VK_A:
            System.out.println(x: "RlLeft");
            if(gameWorld.megaman.getSpeedX() < 0)
                gameWorld.megaman.stopRun();
            break;
        case KeyEvent.VK_W:
            System.out.println(x: "RlUp");
            break;
        case KeyEvent.VK_D:
            System.out.println(x: "RlRight");
            if(gameWorld.megaman.getSpeedX() > 0)
                gameWorld.megaman.stopRun();
            break;
        case KeyEvent.VK_S:
            System.out.println(x: "RlDown");
            gameWorld.megaman.standUp();
            break;
        case KeyEvent.VK_ENTER:
            System.out.println(x: "RlEnter");
            break;
        case KeyEvent.VK_SPACE:

            System.out.println(x: "RlSpace");
            break;
```

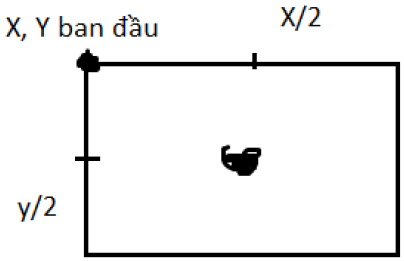
4. FrameImage.

   Như cái tên của lớp, FrameImage giúp lấy được một khung hình của bức ảnh. Điều này phục vụ cho quá trình làm Animation và vẽ các khung hình bất kì lên trên màn hình chơi game.

```
public FrameImage(String name, BufferedImage image){
    this.name = name;
    this.image = image;
}

public FrameImage(FrameImage frameImage){
    image = new BufferedImage(width: frameImage.getWidthImage(),
            height: frameImage.getHeightImage(), imageType: frameImage.image.getType());
    Graphics g = image.getGraphics();
    g.drawImage(img:frameImage.image, x: 0, y: 0, observer: null);
    name = frameImage.name;
}

public void draw(int x, int y, Graphics2D g2){

    g2.drawImage(img:image, x - image.getWidth()/2, y - image.getHeight()/2, observer: null);

}

public FrameImage(){
    this.name = null;
    image = null;
}

public int getWidthImage(){
    return image.getWidth();
}

public int getHeightImage(){
    return image.getHeight();
}

public void setName(String name){
    this.name = name;
}
public String getName(){
    return name;
}

public BufferedImage getImage(){
    return image;
}
public void setImage(BufferedImage image){
    this.image = image;
}
```

Giải thích: Tạo contructor và các hàm Get-Set (name,imgae). Ở đây ta gọi lại hàm FrameImage(FrameImage frameImage) 2 lần vì giống như con trỏ trong C++, là

cùng là một đối tượng FrameImage nhưng khác nhau về dữ liệu. Nếu không thì ta chỉ vẽ được một FrameImage duy nhất vì ta đã gán cho một dữ liệu duy nhất.
Hàm **draw** trong đây mục đích vẽ bức hình ở tọa độ chính giữa

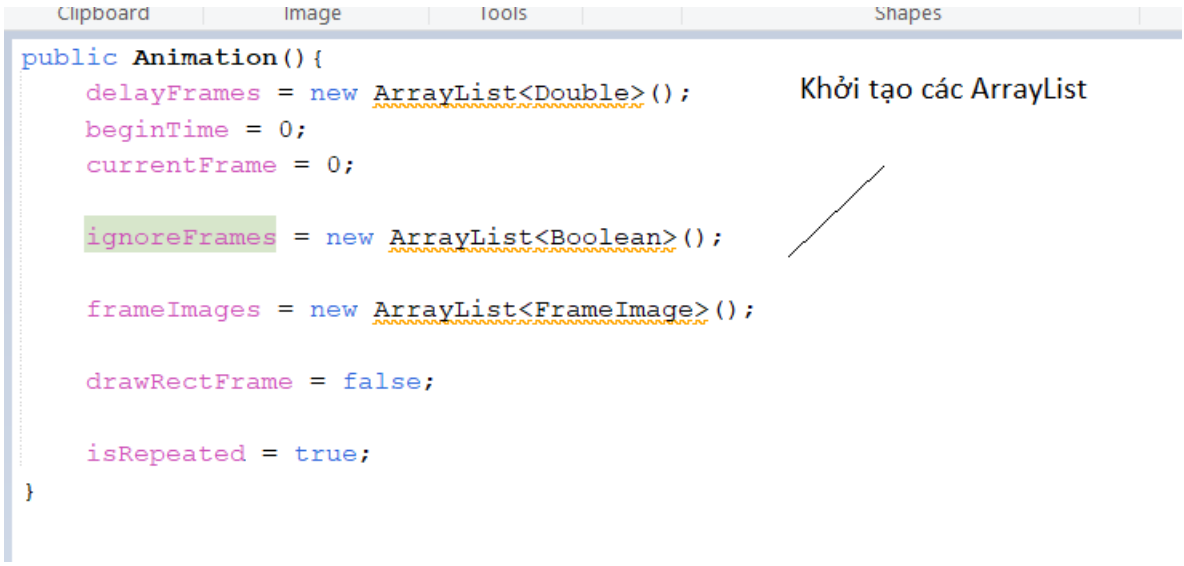


5. Animation

Đây là hàm dùng để tạo ra các animation cho nhân vật. Nguyên tắc là các Frame được vẽ liên tiếp để đánh lừa thị giác của con người về sự chuyển động. Khi thời gian đủ nhanh sẽ tạo ra chuyển động.

```
public class Animation {
    private String name;
    private boolean isRepeated;
    private ArrayList<FrameImage> frameImages;
    private int currentFrame;
    private ArrayList<Boolean> ignoreFrames;
    private ArrayList<Double> delayFrames;
    private long beginTime;
    private boolean drawRectFrame;
```

Giải thích: Mỗi animation đều có tên, và kiểm tra xem mỗi Frame trong Animation ấy có bị lặp lại hay không (isRepeated), ta cần các ArrayList để lưu trữ các FrameImage, thời gian tồn tại của mỗi Frame (delayFrame), bỏ qua các Frame(IgnoreFrames). Ví dụ trong trường hợp nhân vật từ trạng thái đứng im sang chạy, sẽ cần nhún chân, nhưng khi nhân vật đã chạy ta bỏ qua giai đoạn nhún chân, đó chính là ignoreFrames.



```
public Animation(Animation animation){
    beginTime = animation.beginTime;
    currentFrame = animation.currentFrame;
    drawRectFrame = animation.drawRectFrame;
    isRepeated = animation.isRepeated;
    delayFrames = new ArrayList<Double>();
    for(Double d : animation.delayFrames){
        delayFrames.add(e: d);
    }
    ignoreFrames = new ArrayList<Boolean>();
    for(boolean b : animation.ignoreFrames){
        ignoreFrames.add(e: b);
    }
    frameImages = new ArrayList<FrameImage>();
    for(FrameImage f : animation.frameImages){
        frameImages.add(new FrameImage(frameImage: f));
    }
}
```

Giải thích: Tương tự như FrameImage, ta tạo ra phương thức Animation giống hệt nó với mục đích tạo ra các Animation cùng một đối tượng Animation nhưng khác nhau về giá trị.

```
public boolean isIgnoreFrame(int id){
    return ignoreFrames.get(index: id);
}

public void setIgnoreFrame(int id){
    if(id >= 0 && id < ignoreFrames.size())
        ignoreFrames.set(index: id, element: true);
}

public void unIgnoreFrame(int id){
    if(id >= 0 && id < ignoreFrames.size())
        ignoreFrames.set(index: id, element: false);
}
```

Giải thích: Kiểm tra xem các ignoreFrame và inIgnoreFrame có hợp lệ hay không.

```
public void setCurrentFrame(int currentFrame){
    if(currentFrame >= 0 && currentFrame < frameImages.size())
        this.currentFrame = currentFrame;
    else this.currentFrame = 0;
}
public int getCurrentFrame(){
    return this.currentFrame;
}
```

Giải thích: Kiểm tra Frame hiện tại có hợp lệ không, nếu không trả về Frame đầu tiên.

```
public void add(FrameImage frameImage, double timeToNextFrame){

    ignoreFrames.add(e: false);
    frameImages.add(e: frameImage);
    delayFrames.add(new Double(value: timeToNextFrame));

}
```

Giải thích: Các ArrayList được khai báo ở đầu tiên có cùng độ lớn. Với ArrayList của ignoreFrame sẽ được thêm boolean mới bằng false, frameImage sẽ được thêm một Frame mới, timToNextFrame chính là delayFrame.

```
public void Update(long deltaTime){
    if(beginTime == 0) beginTime = deltaTime;
    else{

        if(deltaTime - beginTime > delayFrames.get(index: currentFrame)){
            nextFrame();
            beginTime = deltaTime;
        }
    }

}
```

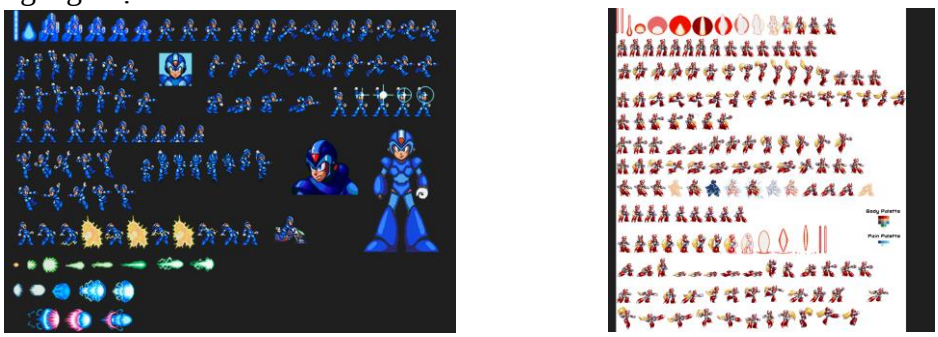
Giải thích: Phương thức Update này tương đối quan trọng vì nó quyết định đến frame tiếp theo của animation. Trong đó, deltaTime chính là thời gian mà ta truyền vào. Nếu thời gian bắt đầu bằng 0, gắn thời gian bắt đầu bằng thời gian truyền. Nếu beginTime khác 0, kiểm tra nếu hiệu số của thời gian truyền vào và thời gian bắt đầu lớn hơn thời gian delay, ta sẽ chuyển sang frame tiếp theo. Và cuối cùng gán beginTime=deltatime.

```
private void nextFrame(){

    if(currentFrame >= frameImages.size() - 1){

        if(isRepeated) currentFrame = 0;
    }
    else currentFrame++;

    if(ignoreFrames.get(index: currentFrame)) nextFrame();

}
```

Giải thích: Phương thức này mục đích chuyển sang khung hình tiếp theo, nếu đã ở khung hình cuối và có hình động lặp lại thì sẽ chuyển CurrentFrame bằng 0 và tiếp tục thực hiện vẽ animation.

```
public void flipAllImage(){

    for(int i = 0;i < frameImages.size(); i++){

        BufferedImage image = frameImages.get(index: i).getImage();

        AffineTransform tx = AffineTransform.getScaleInstance(sx: -1, sy: 1);
        tx.translate(-image.getWidth(), ty: 0);

        AffineTransformOp op = new AffineTransformOp(xform: tx,
        interpolationType:AffineTransformOp.TYPE_BILINEAR);
        image = op.filter(src: image, dst: null);

        frameImages.get(index: i).setImage(image);

    }

}

public void draw(int x, int y, Graphics2D g2){

    BufferedImage image = getCurrentImage();

    g2.drawImage(img: image, x - image.getWidth()/2, y - image.getHeight()/2, observer:null);
    if(drawRectFrame)
        g2.drawRect(x - image.getWidth()/2, x - image.getWidth()/2, width: image.getWidth(), height: image.getHeight());

}
```

Giải thích: Với phương thức flipAllImage dùng để lật ngược hình ảnh theo chiều ngang. Cụ thể như sau:



Hình 3.7. Hình ảnh minh họa cho nhân vật

Với file dữ liệu bạn đầu, nhân vật megaMan không có frame quay bên trái vì vậy các hình động run mà hướng bên trái sẽ không có vì vậy ta chỉ cần thực hiện quay ngược frame theo chiều ngang thay vì cắt các frame mới rồi thực hiện animation lại từ đầu.

6. CacheDataLoader:

Mục đích chính của lớp này chính là đọc dữ liệu từ file bên ngoài bao gồm, đọc animation, đọc frameImage, đọc file chứa sound, đọc physcalmap, backgroundmap.

```
public class CacheDataLoader {
    private static CacheDataLoader instance = null;

    private String framefile = "data/frame.txt";
    private String animationfile = "data/animation.txt";
    private String physmapfile = "data/phys_map.txt";
    private String backgroundmapfile = "data/background_map.txt";
    private String soundfile = "data/sounds.txt";

    private Hashtable<String, FrameImage> frameImages;
    private Hashtable<String, Animation> animations;
    private Hashtable<String, AudioClip> sounds;

    private int[][] phys_map;
    private int[][] background_map;

    private CacheDataLoader() {}

    public static CacheDataLoader getInstance(){
        if(instance == null)
            instance  = new CacheDataLoader();
        return instance;
    }
```

Giải thích: Import các tên file từ bên ngoài.

```
public AudioClip getSound(String name){
    return instance.sounds.get(key: name);
}

public Animation getAnimation(String name){

    Animation animation = new Animation(animation: instance.animations.get(key: name));
    return animation;

}

public FrameImage getFrameImage(String name){

    FrameImage frameImage = new FrameImage(frameImage: instance.frameImages.get(key: name));
    return frameImage;

}
```

Trả về Animation, frameimage bằng cách tìm tên của Animation, FrameImage trong file dữ liệu bên ngoài. Cụ thể:

```
158

idle1
image data/megasprite.png
x 530
y 40
w 80
h 90

idle2
image data/megasprite.png
x 616
y 40
w 80
h 90
```

```
87
idle
idle1 70000000 idle2 70000000 idle3 70000000 idle2 70000000 idle1 1500000000

idleshoot
idleshoot1 500000000 idleshoot1 70000000

run
run1 100000000 run2 100000000 run3 100000000 run4 100000000 run5 100000000 run6 100000000 run7

runshoot
runshoot1 1 runshoot2 1 runshoot3 1 runshoot4 1 runshoot5 5 runshoot6 1 runshoot7 1 runshoot8 1

dick
dick1 70000000 dick3 70000000 dick4 70000000 dick3 70000000 dick1 1500000000

flyingup
flyingup1 70000000 flyingup2 70000000 flyingup3 70000000 flyingup4 70000000 flyingup5 70000000

flyingupshoot
```

Hình 3.8. Ảnh minh họa tên FrameImgae và Animation

LoadFrame:

```
public void LoadFrame() throws IOException{

    frameImages = new Hashtable<String, FrameImage>();

    FileReader fr = new FileReader(fileName: framefile);
    BufferedReader br = new BufferedReader(in: fr);

    String line = null;
```

Giải thích: Khởi tạo Hastabale mới và sử dụng kỹ thuật đọc file đã được học.

```
if(br.readLine()==null) {
    System.out.println(x: "No data");
    throw new IOException();
}
else {

    fr = new FileReader(fileName:framefile);
    br = new BufferedReader(in: fr);

    while((line = br.readLine()).equals(anObject:""));

    int n = Integer.parseInt(s: line);
    String path = null;
    BufferedImage imageData = null;
    int i2 = 0;
    for(int i = 0;i < n; i ++){

        FrameImage frame = new FrameImage();
        while((line = br.readLine()).equals(anObject:""));
        frame.setName(name:line);

        while((line = br.readLine()).equals(anObject:""));
        String[] str = line.split(regex:" ");

        boolean refreshImage = (path == null || !path.equals(str[1]));
        path = str[1];

        while((line = br.readLine()).equals(anObject:""));
        str = line.split(regex:" ");
        int x = Integer.parseInt(str[1]);

        while((line = br.readLine()).equals(anObject:""));
        str = line.split(regex:" ");
        int y = Integer.parseInt(str[1]);
```

```
158

idle1
image data/megasprite.png
x 530
y 40
w 80
h 90
```

Giải thích: Trong một file dữ liệu bên ngoài, file đó sẽ được đọc trong phương thức LoadFrame, nếu file trống ném ra một ngoại lệ báo “No Data”. Nếu không gán n bằng tổng số frame sẵn có trong file. Cho biến i chạy từ 0 đến n, với mỗi i tạo ra một FrameImage khác nhau, sau khi đọc xong số n, dòng sẽ nhảy xuống, so sánh với “”, nếu không “” thì gắn tên bằng dòng tiếp theo. Dòng tiếp theo là đọc nguồn của ảnh, như trong hình sẽ lấy đến nguồn tên “megasprite.png”. Tiếp đến dòng tiếp theo gắn cho biến String str bằng giá trị thứ 2 trong dòng. Ví dụ như tọa độ x 530 gắn cho str=530, tương tự với y,w,h.

```
            if(refreshImage) {
                refreshImage = false;
                imageData = ImageIO.read(new File(pathname:path));
            }
            if(imageData != null) {
                BufferedImage image = imageData.getSubimage(x, y, w, h);
                frame.setImage(image);
            }

            instance.frameImages.put(key: frame.getName(), value: frame);
        }

    }

    br.close();

}
```

Giải thích: Nếu Frame tồn tại sẽ đẩy vào trong ArrayList ban đầu. Khi toàn tất, đóng file.

Tương tự với LoadAnimation, LoadBackGroundMap và LoadPhysicalMap. Tuy nhiên trong LoadPhysicalMap và LoadBackGroundMap ta thay bằng đọc file chứ mảng 2 chiều.

Hình 3.9. Minh họa cho đọc Physical Map

7. GameObject.
   Đây là một lớp abstract, mọi chủ thể trong game đều gồm có 2 tọa độ x,y vì tựa game đang xét là game 2D vì vậy nó có 2 chiều, các chủ thể về sau sẽ có thể kề thừa từ lớp GameObject này.

```
public abstract class GameObject {

        private float posX;
        private float posY;

        private GameWorld gameWorld;

        public GameObject(float x, float y, GameWorld gameWorld){
                posX = x;
                posY = y;
                this.gameWorld = gameWorld;
        }

        public void setPosX(float x){
                posX = x;
        }

        public float getPosX(){
                return posX;
        }

        public void setPosY(float y){
                posY = y;
        }

        public float getPosY(){
                return posY;
        }

        public GameWorld getGameWorld(){
                return gameWorld;
        }

        public abstract void Update();

}
```

Giải thích: mỗi chủ thể đều có 2 giá trị vì vậy khởi tạo 2 giá trị vị trí x và vị trí y, đồng thời gọi thêm 1 lớp GameWorld để dễ dàng can thiệt vào lớp GameWorld đó. Tạo 1 contructor đồng thời get-set. Ta tạo thêm một phương thức abstract Update để mọi class kế thừa GameObject đều phải định nghĩa lại Update.

8. Camera.
   Đây là một lớp kế thừa lại từ GameObject mục đích tạo một camera di chuyển theo nhân vật trong quá trình chơi game.

```
package GameObject;
public class Camera extends GameObject{
    private float widthView;
    private float heightView;

    private boolean isLocked = false;

    public Camera(float x, float y, float widthView, float heightView, GameWorld gameWorld) {
        super(x, y, gameWorld);
        this.widthView = widthView;
        this.heightView = heightView;
    }

    public void lock(){
        isLocked = true;
    }

    public void unlock(){
        isLocked = false;
    }
```

```
public float getWidthView() {
    return widthView;
}

public void setWidthView(float widthView) {
    this.widthView = widthView;
}

public float getHeightView() {
    return heightView;
}

public void setHeightView(float heightView) {
    this.heightView = heightView;
}
```

Giải thích: Khởi tạo 2 giá trị độ rộng và độ cao cho camera. Tạo một contructor đồng thời thêm phương thức get-set để lấy giá trị width và height. Thêm 2 phương thức trả về boolean kiểm tra camera khóa hay không. Nếu lock thì true và ngược lại.

```
@Override
public void Update() {
    if(!isLocked){
        MegaMan mainCharacter = getGameWorld().megaman;

        if(mainCharacter.getPosX() - getPosX() > 400) setPosX(mainCharacter.getPosX() - 400);
        if(mainCharacter.getPosX() - getPosX() < 200) setPosX(mainCharacter.getPosX() - 200);
        if(mainCharacter.getPosY() - getPosY() > 400) setPosY(mainCharacter.getPosY() - 400); // bottom
        else if(mainCharacter.getPosY() - getPosY() < 250) setPosY(mainCharacter.getPosY() - 250);// top
    }

}
```

Giải thích: Định nghĩa lại phương thức Update() của GameObject. Cập nhật vị trí của Camera theo vị trí nhân vật megaman bằng cách getPos của megaman và setPos cho bản thân camera.

9. PhysicalMap

PhysicalMap được kế thừa từ lớp GameOject. Mục tiêu tạo ra một không gian giới hạn để nhân vật có thể di chuyển.

```
public class PhysicalMap extends GameObject{
    public int[][] phys_map;
    private int tileSize;

    public PhysicalMap(float x,float y,GameWorld gameWorld){
        super(x,y, gameWorld);
        this.tileSize=30;
        phys_map = CacheDataLoader.getInstance().getPhysicalMap();
    }
    public int getTileSize(){
        return tileSize;
    }
```

Giải thích: gọi một mảng hai chiều phys_map và gán cho nó bằng file được đọc từ folder date và tạo một contructor, get-set.

```
public void draw(Graphics2D g2){
    Camera camera = getGameWorld().camera;
    g2.setColor(Color.GRAY);
    for(int i=0;i<phys_map.length;i++){
        for(int j=0;j<phys_map[0].length;j++){
            if(phys_map[i][j] !=0) g2.fillRect((int)getPosX()+ j*tileSize-(int)camera.getPosX(), (int) getPosY() +i*tileSize-(int) camera.getPosY(), width:tileSize,height: tileSize);
        }
    }
}
```

Giải thích: Trong file phys_map dữ liệu là mảng 2 chiều nhận 2 giá trị là 0 và 1 vì vậy ta đọc từng hàng và từng cột, nếu giá trị [i][j] =1 thì ta vẽ màu GRAY(xám) kích thước 30px(tileSize).

```
public Rectangle haveCollisionWithLand(Rectangle rect){

    int posX1 = rect.x/tileSize;
    posX1 -= 2;
    int posX2 = (rect.x + rect.width)/tileSize;
    posX2 += 2;

    int posY1 = (rect.y + rect.height)/tileSize;

    if(posX1 < 0) posX1 = 0;

    if(posX2 >= phys_map[0].length) posX2 = phys_map[0].length - 1;
    for(int y = posY1; y < phys_map.length;y++){
        for(int x = posX1; x <= posX2; x++){

            if(phys_map[y][x] == 1){
                Rectangle r = new Rectangle((int) getPosX() + x * tileSize, (int) getPosY()+ y * tileSize, width:tileSize, height: tileSize);
                if(rect.intersects(r))
                    return r;
            }
        }
    }
    return null;
}
```

Hàm kiểm tra va chạm với mặt đất
trả về giá trị là một hình chữ nhật

```
//kiemtravachamvoituongbenphai
public Rectangle haveCollisionWithRightWall(Rectangle rect){


    int posY1 = rect.y/tileSize;
    posY1-=2;
    int posY2 = (rect.y + rect.height)/tileSize;
    posY2+=2;

    int posX1 = (rect.x + rect.width)/tileSize;
    int posX2 = posX1 + 3;
    if(posX2 >= phys_map[0].length) posX2 = phys_map[0].length - 1;

    if(posY1 < 0) posY1 = 0;
    if(posY2 >= phys_map.length) posY2 = phys_map.length - 1;


    for(int x = posX1; x <= posX2; x++){
        for(int y = posY1; y <= posY2;y++){
            if(phys_map[y][x] == 1){
                Rectangle r = new Rectangle((int) getPosX() + x * tileSize, (int) getPosY() + y * tileSize, width:tileSize, height: tileSize);
                if(r.y < rect.y + rect.height - 1 && rect.intersects(r))
                    return r;
            }
        }
    }
    return null;

}
```

Hàm kiểm tra va chạm với tường
Trả về là một hình chữ nhật

```
//kiemtravachamvoituongtren
public Rectangle haveCollisionWithTop(Rectangle rect){

    int posX1 = rect.x/tileSize;
    posX1 -= 2;
    int posX2 = (rect.x + rect.width)/tileSize;
    posX2 += 2;

    //int posY = (rect.y + rect.height)/tileSize;
    int posY = rect.y/tileSize;

    if(posX1 < 0) posX1 = 0;

    if(posX2 >= phys_map[0].length) posX2 = phys_map[0].length - 1;

    for(int y = posY; y >= 0; y--){
        for(int x = posX1; x <= posX2; x++){

            if(phys_map[y][x] == 1){
                Rectangle r = new Rectangle((int) getPosX() + x * tileSize, (int) getPosY() + y * tileSize, width:tileSize, height: tileSize);
                if(rect.intersects(r))
                    return r;
            }
        }
    }
    return null;
}
```

Kiểm tra va chạm với tường trên

Để kiểm tra va chạm với tường bên trái ta làm tương tự.

10. BackgroundMap.

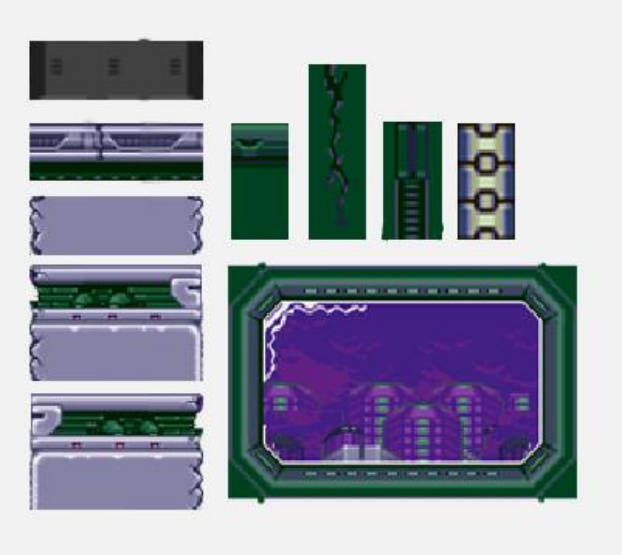
Tương tự như Camera, đây là class được kế thừa từ GameObject. Mục tiêu là vẽ ra BackGround của màn chơi.

```
public class BackgroundMap extends GameObject {

    public int[][] map;
    private int tileSize;

    public BackgroundMap(float x, float y, GameWorld gameWorld) {
        super(x, y, gameWorld);
        map = CacheDataLoader.getInstance().getBackgroundMap();
        tileSize = 30;
    }

    @Override
    public void Update() {}

    public void draw(Graphics2D g2){

        Camera camera = getGameWorld().camera;

        g2.setColor( c: Color.RED);
        for(int i = 0;i< map.length;i++)
            for(int j = 0;j<map[0].length;j++)
                if(map[i][j]!=0 && j*tileSize - camera.getPosX() > -30 && j*tileSize - camera.getPosX() < GameFrame_Multi_Player_Mode.SCREEN_WIDTH
                        && i*tileSize - camera.getPosY() > -30 && i*tileSize - camera.getPosY() < GameFrame_Multi_Player_Mode.SCREEN_HEIGHT){
                    g2.drawImage( img: CacheDataLoader.getInstance().getFrameImage("tiled"+map[i][j]).getImage(), (int) getPosX() + j*tileSize - (int) camera.getPosX(),
                            (int) getPosY() + i*tileSize - (int) camera.getPosY(), observer: null);
                }

    }
```

Contructor

Vẽ các mảnh của BackGround

Giải thích: Tạo ra mảng 2 chiều map và gán map bằng file được đọc từ folder data. Tạo contructor và gắn kích thước 1 mảnh nhỏ = 30px(tilesize). Trong phương thức Update() lặp theo số hàng và số cột của file Map, kiểm tra xem những vị trí nào trong phạm vi của Camera và nhỏ hơn kích thước của GameFrame thì sẽ được vẽ, ta sẽ kiểm tra theo cả 2 chiều x,y. Nếu thỏa mãn, vẽ ra frame để tạo ra background.
Ví dụ:



Hình 3.10. Các mảnh của background

11. ParticularObject

Tương tự như GameObject, đây cũng là một lớp trừu tượng. Tuy nhiên, class chi tiết hơn về đối tượng và nó kế thừa từ GameObject.

```
public abstract class ParticularObject extends GameObject{

    public static final int LEAGUE_TEAM = 1;
    public static final int ENEMY_TEAM = 2;

    public static final int LEFT_DIR = 0;
    public static final int RIGHT_DIR = 1;
    public static final int TELE = 9;
    public static final int ALIVE = 0;
    public static final int BEHURT = 1;
    public static final int FEY = 2;
    public static final int DEATH = 3;
    public static final int NOBEHURT = 4;
    private int state = ALIVE;

    private float width;
    private float height;
    private float mass;
    private float speedX;
    private float speedY;
    private int blood;

    private int damage;

    private int direction;

    protected Animation behurtForwardAnim, behurtBackAnim;
    protected Animation teleForwardAnim;

    private int teamType;

    private long startTimeNoBeHurt;
    private long timeForNoBeHurt;
```

Giải thích: Trong một đối tượng cụ thể hơn so với GameObject chính là các Robot bao gồm các trạng thái, thuộc tính như sau: Trạng thái team (cùng hoặc khác team), hướng di chuyển (trái-phải) ảnh hưởng đến hướng chuyển động của animation, vị trí, sát thương, trọng lượng, tốc độ, máu (blood), các trạng thái khác như: Bị tấn công (beHurt), chết (DEATH), vẫn còn sống (ALIVE).

```
public ParticularObject(float x, float y, float width, float height, float mass, int blood, GameWorld gameWorld){

    // posX and posY are the middle coordinate of the object
    super(x, y, gameWorld);
    setWidth(width);
    setHeight(height);
    setMass(mass);
    setBlood(blood);

    direction = RIGHT_DIR;

}
```

Tạo contructor và các phương thức get-set.

```
public boolean isObjectOutOfCameraView(){
    if(getPosX() - getGameWorld().camera.getPosX() > getGameWorld().camera.getWidthView() ||
            getPosX() - getGameWorld().camera.getPosX() < -50
        ||getPosY() - getGameWorld().camera.getPosY() > getGameWorld().camera.getHeightView()
                ||getPosY() - getGameWorld().camera.getPosY() < -50)
        return true;
    else return false;
}
```

Giải thích: Kiểm tra chủ thể có bị out khỏi camera hay không, nếu có trả về giá trị true, không thì false.

```
public Rectangle getBoundForCollisionWithMap(){
    Rectangle bound = new Rectangle();
    bound.x = (int) (getPosX() - (getWidth()/2));
    bound.y = (int) (getPosY() - (getHeight()/2));
    bound.width = (int) getWidth();
    bound.height = (int) getHeight();
    return bound;
}
```

Đưa tọa độ về trung tâm

Vẽ một hình chữ nhật bao quanh nhân vật. Để dễ dàng kiểm tra va chạm.

```
public void beHurt(int damgeEat){
    setBlood(getBlood() - damgeEat);
    state = BEHURT;
    hurtingCallback();
}
```

Phương thức kiểm tra bị tấn công và trả về máu mới (máu mới bằng máu cũ – sát thương)

```
public void Update(){
    switch(state){
        case TELE:
            teleForwardAnim.Update(deltaTime:System.nanoTime());
        case ALIVE:
            ParticularObject object = getGameWorld().particularObjectManager.getCollisionWidthEnemyObject(object:this);
            if(object!=null){
                if(object.getDamage() > 0){
                    System.out.println("eat damage.... from collision with enemy..... "+object.getDamage());
                    beHurt(damgeEat: object.getDamage());
                }

            }
            break;
        case BEHURT:
            if(behurtBackAnim == null){
                state = NOBEHURT;
                startTimeNoBeHurt = System.nanoTime();
                if(getBlood() == 0)
                    state = FEY;

            } else {
                behurtForwardAnim.Update(deltaTime:System.nanoTime());
                if(behurtForwardAnim.isLastFrame()){
                    behurtForwardAnim.reset();
                    state = NOBEHURT;
                    if(getBlood() == 0)
                        state = FEY;
                    startTimeNoBeHurt = System.nanoTime();
                }
            }
            break;

        case FEY:
            state = DEATH;
            break;
        case DEATH:
```

Giải thích: Phương thức Update() được định nghĩa lại, trong phương thức này dựa vào trạng thái mà Update(). Nếu trạng thái còn sống mà bị tấn công sẽ gọi lên phương thức behurt() ở trên để trừ máu và chuyển trạng thái thành beHurt. Nếu trạng thái thành BEHURT, trước tiên chuyển trạng thái sáng NOBEHURT vì đã BEHURT rồi, đồng thời cài cho thời gian gắn thời gian bắt đầu bị tổn thương bằng thời gian hệ thống. Nếu Blood ==0 thì nhân vật đã chết chuyển trạng thái sang DEATH.

```
case NOBEHURT:
    System.out.println(x: "state = nobehurt");
    if(System.nanoTime() - startTimeNoBeHurt > timeForNoBeHurt)
        state = ALIVE;
    break;
```

Giải thích: Nếu hiệu thời gian hệ thống và thời gian bắt đầu bị tấn công lớn hơn thời gian không bị tổn thời sẽ chuyển lại trạng thái sang ALIVE và tiếp tục Update();

```
public void drawBoundForCollisionWithMap(Graphics2D g2){
    Rectangle rect = getBoundForCollisionWithMap();
    g2.setColor(c: Color.BLUE);
    g2.drawRect(rect.x - (int) getGameWorld().camera.getPosX(), rect.y - (int) getGameWorld().camera.getPosY(), width: rect.width, height: rect.height);
}

public void drawBoundForCollisionWithEnemy(Graphics2D g2){
    Rectangle rect = getBoundForCollisionWithEnemy();
    g2.setColor(c: Color.RED);
    g2.drawRect(rect.x - (int) getGameWorld().camera.getPosX(), rect.y - (int) getGameWorld().camera.getPosY(), width: rect.width, height: rect.height);
}
```

Vẽ hình chữ nhật để dễ kiểm soát va chạm với địch mà Map.

```
public abstract Rectangle getBoundForCollisionWithEnemy();

public abstract void draw(Graphics2D g2);

public void hurtingCallback(){};
```

Tạo ra các phương thứ abstract để các đối tượng Extend phải định nghĩa lại.

12. Human

Human kế thừa từ lớp cha là ParticularObject, nó hướng đến một đối tượng cụ thể hơn đó là các Robot.

```
public abstract  class Human extends ParticularObject{
    private boolean isJumping;
    private boolean isDicking;
    private boolean isLanding;

    public Human(float x, float y, float width, float height, float mass, int blood, GameWorld gameWorld) {
        super(x, y, width, height, mass, blood, gameWorld);
        setState(state: ALIVE);
    }
```

Tạo contructor và các biến check và get-set cho chúng.

```
public abstract void run();

public abstract void jump();

public abstract void dick();

public abstract void standUp();

public abstract void stopRun();
```

Tạo ra các abstract để các class kế thừa phải định nghĩa lại.

```
@Override
public void Update(){
    super.Update();
    if(getState() == ALIVE || getState() == NOBEHURT){
        if(!isLanding){
            setPosX(getPosX() + getSpeedX());
            if(getDirection() == LEFT_DIR &&
                    getGameWorld().physicalMap.haveCollisionWithLeftWall(rect: getBoundForCollisionWithMap())!=null){
                Rectangle rectLeftWall = getGameWorld().physicalMap.haveCollisionWithLeftWall(rect: getBoundForCollisionWithMap());
                setPosX(rectLeftWall.x + rectLeftWall.width + getWidth()/2);
            }
            if(getDirection() == RIGHT_DIR &&
                    getGameWorld().physicalMap.haveCollisionWithRightWall(rect: getBoundForCollisionWithMap())!=null){

                Rectangle rectRightWall = getGameWorld().physicalMap.haveCollisionWithRightWall(rect: getBoundForCollisionWithMap());
                setPosX(rectRightWall.x - getWidth()/2);
            }
            Rectangle boundForCollisionWithMapFuture = getBoundForCollisionWithMap();
            boundForCollisionWithMapFuture.y += (getSpeedY()!=0?getSpeedY(): 2);
            Rectangle rectLand = getGameWorld().physicalMap.haveCollisionWithLand(rect: boundForCollisionWithMapFuture);
            Rectangle rectTop = getGameWorld().physicalMap.haveCollisionWithTop(rect: boundForCollisionWithMapFuture);
            if(rectTop !=null){
                setSpeedY(speedY: 0);
                setPosY(rectTop.y + getGameWorld().physicalMap.getTileSize() + getHeight()/2);

            }else if(rectLand != null){
                setIsJumping(isJumping: false);
                if(getSpeedY() > 0) setIsLanding(b: true);
                setSpeedY(speedY: 0);
                setPosY(rectLand.y - getHeight()/2 - 1);
            }else{
                setIsJumping(isJumping: true);
                setSpeedY(getSpeedY() + getMass());
                setPosY(getPosY() + getSpeedY());
            }
        }
    }
}
```

Giải thích: Định nghĩa lại phương thức Update. Nếu trạng thái còn sống và không bị tấn công, đồng thời không chạm đất, sẽ kiểm tra va chạm với tường, mặt đất, trần nhà và trả về cái hình chữ nhật ở bên lớp physicalMap. Nếu chủ thể va chạm với trần, tốc độ trục y = 0, nếu đang bay, tốc độ bằng tốc độ cũ + trọng lượng cơ thể.

13. Megaman.

    Đây là lớp chủ thể chính của game nó kế thừa từ lớp Human, vì vậy nó có những hành động của một Human, ngoài ra còn có thể phương thức attack();

```
public class MegaMan extends Human {

    public static final int RUNSPEED = 3;

    private Animation runForwardAnim, runBackAnim, runShootingForwarAnim, runShootingBackAnim;
    private Animation idleForwardAnim, idleBackAnim, idleShootingForwardAnim, idleShootingBackAnim;
    private Animation dickForwardAnim, dickBackAnim;
    private Animation flyForwardAnim, flyBackAnim, flyShootingForwardAnim, flyShootingBackAnim;
    private Animation landingForwardAnim, landingBackAnim;
    private Animation teleForwardAnim, teleBackAnim;

    private Animation climWallForward, climWallBack;

    private long lastShootingTime;
    private boolean isShooting = false;

    private AudioClip hurtingSound;
    private AudioClip shooting1;
```

Khởi tạo các Animation liên quan đến MegaMan.

```
public MegaMan(float x, float y, GameWorld gameWorld) {
    super(x, y, width: 70, height:90, mass: 0.1f, blood: 200, gameWorld);

    shooting1 = CacheDataLoader.getInstance().getSound(name: "bluefireshooting");
    hurtingSound = CacheDataLoader.getInstance().getSound(name: "megamanhurt");

    setTeamType(team:LEAGUE_TEAM);

    setTimeForNoBehurt(2000*1000000);

    runForwardAnim = CacheDataLoader.getInstance().getAnimation(name: "run");
    runBackAnim = CacheDataLoader.getInstance().getAnimation(name: "run");
    runBackAnim.flipAllImage();

    idleForwardAnim = CacheDataLoader.getInstance().getAnimation(name: "idle");
    idleBackAnim = CacheDataLoader.getInstance().getAnimation(name: "idle");
    idleBackAnim.flipAllImage();

    dickForwardAnim = CacheDataLoader.getInstance().getAnimation(name: "dick");
    dickBackAnim = CacheDataLoader.getInstance().getAnimation(name: "dick");
    dickBackAnim.flipAllImage();
```

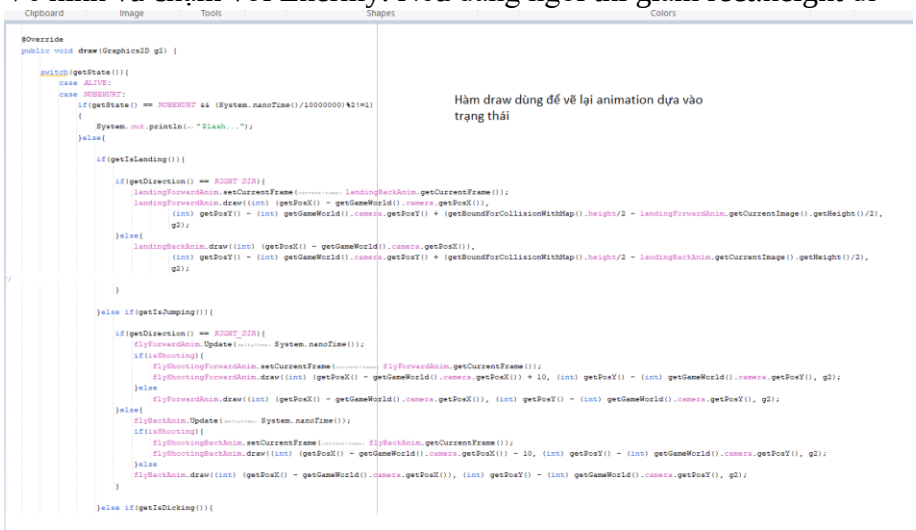
Tạo contructor và LoadAnimation của các hành động, đồng thời setTeamType.

```
@Override
public void Update() {

    super.Update();

    if(isShooting){
        if(System.nanoTime() - lastShootingTime > 500*1000000){
            isShooting = false;
        }
    }

    if(getIsLanding()){
        landingBackAnim.Update(deltaTime:System.nanoTime());
        if(landingBackAnim.isLastFrame()) {
            setIsLanding(b: false);
            landingBackAnim.reset();
            runForwardAnim.reset();
            runBackAnim.reset();
        }
    }

}
```

Giải thích: Nếu thời gian hệ thống trừ đi thời gian phát bắn cuối cùng lớn hơn 0.5s thì chủ thể đã bắn xong. Nếu chủ thể đang đứng, Animation đứng (idle) được update(). Nếu đã đến Frame cuối cùng thì reset lại từ đầu.\

```
@Override
public Rectangle getBoundForCollisionWithEnemy() {
    // TODO Auto-generated method stub
    Rectangle rect = getBoundForCollisionWithMap();

    if(getIsDicking()){
        rect.x = (int) getPosX() - 22;
        rect.y = (int) getPosY() - 20;
        rect.width = 44;
        rect.height = 65;
    }else{
        rect.x = (int) getPosX() - 22;
        rect.y = (int) getPosY() - 40;
        rect.width = 44;
        rect.height = 80;
    }

    return rect;
}
```

Vẽ hình va chạm với Enermy. Nếu đang ngồi thì giảm rect.height đi



```
@Override
public void run() {
    if(getDirection() == LEFT_DIR)
        setSpeedX(speedX: -4);
    else setSpeedX(speedX: 4);
}

@Override
public void jump() {

    if(!getIsJumping()){
        setIsJumping(isJumping: true);
        setSpeedY(-5.0f);
        flyBackAnim.reset();
        flyForwardAnim.reset();
    }
    // for clim wall
    else{
        Rectangle rectRightWall = getBoundForCollisionWithMap();
        rectRightWall.x += 1;
        Rectangle rectLeftWall = getBoundForCollisionWithMap();
        rectLeftWall.x -= 1;

        if(getGameWorld().physicalMap.haveCollisionWithRightWall(rect: rectRightWall)!=null && getSpeedX() > 0){
            setSpeedY(-5.0f);
            //setSpeedX(-1);
            flyBackAnim.reset();
            flyForwardAnim.reset();
            //setDirection(LEFT_DIR);
        }else if(getGameWorld().physicalMap.haveCollisionWithLeftWall(rect: rectLeftWall)!=null && getSpeedX() < 0){
            setSpeedY(-5.0f);
            //setSpeedX(1);
            flyBackAnim.reset();
            flyForwardAnim.reset();
            //setDirection(RIGHT_DIR);
        }

    }
}
```

Định nghĩa lại các hành đông chạy nhảy ngồi của nhân vật.

```
@Override
public void dick() {
    if(!getIsJumping())
        setIsDicking(isDicking: true);
}

@Override
public void standUp() {
    setIsDicking(isDicking: false);
    idleForwardAnim.reset();
    idleBackAnim.reset();
    dickForwardAnim.reset();
    dickBackAnim.reset();
}

@Override
public void stopRun() {
    setSpeedX(speedX: 0);
    runForwardAnim.reset();
    runBackAnim.reset();
    runForwardAnim.unIgnoreFrame(id: 0);
    runBackAnim.unIgnoreFrame(id: 0);
}
@Override
public void attack() {

    if(!isShooting && !getIsDicking()){

        shooting1.play();

        Bullet bullet = new BlueFire(x: getPosX(), y: getPosY(), gameWorld: getGameWorld());
        if(getDirection() == LEFT_DIR) {
            bullet.setSpeedX(speedX: -10);
            bullet.setPosX(bullet.getPosX() - 40);
            if(getSpeedX() != 0 && getSpeedY() == 0){
                bullet.setPosX(bullet.getPosX() - 10);
                bullet.setPosY(bullet.getPosY() - 5);
            }
        }else {
            bullet.setSpeedX(speedX: 10);
            bullet.setPosX(bullet.getPosX() + 40);
            if(getSpeedX() != 0 && getSpeedY() == 0){
                bullet.setPosX(bullet.getPosX() + 10);
                bullet.setPosY(bullet.getPosY() - 5);
            }
        }
        if(getIsJumping())
            bullet.setPosY(bullet.getPosY() - 20);
```

```
    }
    public void specialAttack(){
        if(!isShooting && !getIsDicking()){

            shooting1.play();

            Bullet bullet = new SpecialBullet(x: getPosX(), y: getPosY(), gameWorld: getGameWorld());
            if(getDirection() == LEFT_DIR) {
                bullet.setSpeedX(speedX: -10);
                bullet.setPosX(bullet.getPosX() - 40);
                if(getSpeedX() != 0 && getSpeedY() == 0){
                    //truong hop dang chay ban sung
                    bullet.setPosX(bullet.getPosX() - 10);
                    bullet.setPosY(bullet.getPosY() - 5);
                }
            }else {
                bullet.setSpeedX(speedX: 10);
                bullet.setPosX(bullet.getPosX() + 40);
                if(getSpeedX() != 0 && getSpeedY() == 0){
                    bullet.setPosX(bullet.getPosX() + 10);
                    bullet.setPosY(bullet.getPosY() - 5);
                }
            }
            if(getIsJumping())
                //set_vi_tri_cho_dung_sung
                bullet.setPosY(bullet.getPosY() - 20);
            //cung_team_khong_tinh)sat_thuong
            bullet.setTeamType(team: getTeamType());
            getGameWorld().bulletManager.addObject(particularObject: bullet);
            lastShootingTime = System.nanoTime();
            isShooting = true;

        }
    }
    @Override
    public void hurtingCallback(){
        System.out.println(x: "Call back hurting");
        hurtingSound.play();
    }

}
```

Mỗi nhân vật khác nhau sẽ có một chiêu thức tấn công đặc biệt khác nhau (specialAttack) ta thay đổi bằng cách thay đổi animation và setDamage.

14. Bullet

Đây là lớp định nghĩa đạn của cả nhân vật và địch, nó kế thừa ParticulaObject.

```
 */
public abstract class Bullet extends ParticularObject{

    public Bullet(float x, float y, float width, float height, float mass, int damage, GameWorld gameWorld) {
        super(x, y, width, height, mass,blood: 1, gameWorld);
        setDamage(damage);
    }

    public abstract void draw(Graphics2D g2);

    public void Update(){
         super.Update();
            setPosX(getPosX()+getSpeedX());
            setPosY(getPosY()+getSpeedY());
            ParticularObject object = getGameWorld().particularObjectManager.getCollisionWidthEnemyObject(object:this);
         if(object!=null && object.getState() == ALIVE){
            setBlood(blood: 0);
            object.beHurt(damgeEat: getDamage());
            System.out.println(x: "Bullet set behurt for enemy");
          }
    }

}
```

Giải thích: Đạn sẽ có tốc độ bằng cách getPos+getspeed. Nếu không có chủ thể sát thương bằng 0, và mỗi chủ thể sẽ chịu một lượng sát thương nhất định tùy vào ta setDamage cho đạn.

15. BlueFire

Lớn này được kế thừa từ lớp Bullet trên.

16. ParticularObjectManager

Lớp này dùng để quản lí tất cả các đối tượng của game bảo gồm nhân vật của game. Nó thực hiện thêm đối tượng, xóa đối tượng, kiểm tra đối tượng va chạm, Update() các đối tượng và vẽ các tượng.

```
public ParticularObject getCollisionWidthEnemyObject(ParticularObject object){
    synchronized(particularObjects){
        for(int id = 0; id < particularObjects.size(); id++){

            ParticularObject objectInList = particularObjects.get(index: id);

            if(object.getTeamType() != objectInList.getTeamType() &&
                    object.getBoundForCollisionWithEnemy().intersects(r: objectInList.getBoundForCollisionWithEnemy())){
                return objectInList;
            }
        }
    }
    return null;
}

public void UpdateObjects(){

    synchronized(particularObjects){
        for(int id = 0; id < particularObjects.size(); id++){

            ParticularObject object = particularObjects.get(index: id);


            if(!object.isObjectOutOfCameraView()) object.Update();

            if(object.getState() == ParticularObject.DEATH){
                particularObjects.remove(index: id);
            }
        }
    }

    //System.out.println("Camerawidth  = "+camera.getWidth());

}
```

Kiểm tra va chạm

Update

```
public void draw(Graphics2D g2){
    synchronized(particularObjects){
        for(ParticularObject object: particularObjects)
            if(!object.isObjectOutOfCameraView()) object.draw(g2);
    }
}
```

17. BulletManager

    Kế thừa từ ParticularObjectManager.

```
 */
public class BulletManager  extends ParticularObjectManager{

    public BulletManager(GameWorld gameWorld) {
        super(gameWorld);
    }
    @Override
    public void UpdateObjects(){
        super.UpdateObjects();
        synchronized (particularObjects) {
            //kiem tra doi tuong out khoi man hinh
            for(int id=0;id<particularObjects.size();id++){
                ParticularObject object = particularObjects.get(index: id);
                if(!object.isObjectOutOfCameraView()) object.Update();
                if(object.isObjectOutOfCameraView() || object.getState() == ParticularObject.DEATH){
                    particularObjects.remove(index: id);
                }
            }
        }

    }

}
```

18. FinaBoss và các robot khác (tương tự như lớp Megaman thay đổi các chỉ số máu, vị trí, di chuyển, phương thức tấn công).
19. GameWorld

    Là lớp chứa toàn bộ Object của trò chơi.

```
public class GameWorld {
    public MegaMan megaman ;
    public MegaMan1 megaman1;
    public Camera camera ;
    public PhysicalMap physicalMap;
    public BulletManager bulletManager;
    public BackgroundMap backgroundMap;
    public AudioClip bgMusic;
    private BufferedImage bufferedImage;
    private int lastState;
```

Các chủ thể đều được import vào trong class.

```
public ParticularObjectManager particularObjectManager;
public  GameWorld(){

    texts1[0] = "Doctor, I heard he has returned to destroy our city ! \n" +
and his power is terrible along with the large number of monsters.";
    texts1[1] = "What should I do now, Dr. Thomas . . . .?";
    texts1[2] = "Go to the city and help those in need !\nNow is the time for us, kill it and get freedom . !....";
    texts1[3] = "I will add someone to help you. They will come soon. . . . .!";
    texts1[4] = "Really doctor, I will destroy all of you, just wait . . . .  ! ! !"  ;
    texts1[5] = "      LET'S GO!.....";
    texts1[6] = "      GOOD LUCK !.....";




    textTutorial = texts1[0];


    bufferedImage = new BufferedImage(width: GameFrame_Multi_Player_Mode.SCREEN_WIDTH, height:GameFrame_Multi_Player_Mode.SCREEN_HEIGHT, imageType:B
    megaman = new MegaMan(x: 200, y: 300, gameWorld:this);
    megaman1 = new MegaMan1(x: 300, y: 300, gameWorld:this);
    megaman.setTeamType(team: ParticularObject.LEAGUE_TEAM);
    megaman1.setTeamType(team: ParticularObject.LEAGUE_TEAM);
    physicalMap = new PhysicalMap(x: 0, y: 0,gameWorld:this);
    backgroundMap = new BackgroundMap(x: 0, y: 0, gameWorld:this);
    camera = new Camera(x: 0, y: 0, widthView:GameFrame_Multi_Player_Mode.SCREEN_WIDTH, heightView:GameFrame_Multi_Player_Mode.SCREEN_HEIGHT, gameWorld:
/       camera1 = new Camera1(0, 0, GameFrame.SCREEN_WIDTH, GameFrame.SCREEN_HEIGHT, this);
    bulletManager = new BulletManager(gameWorld:this);
    particularObjectManager = new ParticularObjectManager(gameWorld:this);
   particularObjectManager.addObject(particularObject:megaman);
   particularObjectManager.addObject(particularObject:megaman1);
   initEnemies();
   bgMusic= CacheDataLoader.getInstance().getSound(name: "bgmusic");
```

Khởi tạo các chủ thể.

```
private void initEnemies(){
    ParticularObject redeye = new RedEyeDevil(x: 1250, y: 410, gameWorld:this);
    redeye.setDirection(dir:ParticularObject.LEFT_DIR);
    redeye.setTeamType(team: ParticularObject.ENEMY_TEAM);
    particularObjectManager.addObject(particularObject:redeye);

    ParticularObject redeye2 = new RedEyeDevil(x: 2500, y: 500, gameWorld:this);
    redeye2.setDirection(dir:ParticularObject.LEFT_DIR);
    redeye2.setTeamType(team: ParticularObject.ENEMY_TEAM);
    particularObjectManager.addObject(particularObject:redeye2);

    ParticularObject redeye3 = new RedEyeDevil(x: 3450, y: 500, gameWorld:this);
    redeye3.setDirection(dir:ParticularObject.LEFT_DIR);
    redeye3.setTeamType(team: ParticularObject.ENEMY_TEAM);
    particularObjectManager.addObject(particularObject:redeye3);

    ParticularObject redeye4 = new RedEyeDevil(x: 500, y: 1190, gameWorld:this);
    redeye4.setDirection(dir:ParticularObject.RIGHT_DIR);
    redeye4.setTeamType(team: ParticularObject.ENEMY_TEAM);
    particularObjectManager.addObject(particularObject:redeye4);



    ParticularObject robotR = new RobotR(x: 900, y: 400, gameWorld:this);
    robotR.setTeamType(team: ParticularObject.ENEMY_TEAM);
    particularObjectManager.addObject(particularObject:robotR);

    ParticularObject robotR2 = new RobotR(x: 3400, y: 350, gameWorld:this);
    robotR2.setTeamType(team: ParticularObject.ENEMY_TEAM);
    particularObjectManager.addObject(particularObject:robotR2);
```

Thêm các chủ thể khác phe.

```
public void switchState(int state){
    previousState = this.state;
    this.state = state;
}


private void TutorialUpdate(){
    switch(tutorialState){
        case INTROGAME:

            if(storyTutorial == 0){
                if(openIntroGameY < 450) {
                    openIntroGameY+=4;
                }else storyTutorial ++;

            }else{
                if(currentSize < textTutorial.length()) currentSize++;
            }

            break;
        case MEETFINALBOSS:
            break;
    }
}
private void drawString(Graphics2D g2, String text, int x, int y){
    for(String str : text.split(regex: "\n"))
        g2.drawString(str, x, y+=g2.getFontMetrics().getHeight());
}
```

Cài đặt trạng thái để Update() và Render() cho từng trạng thái.
Các thuật toán được sử dụng để vẽ hình đồng thời xử lí trạng thái khi máu của nhân vật đã hết hoặc Finalboss được đánh bại sẽ được chuyển sang trạng thái mới để Update và Render() tùy thuộc vào sở thích của mỗi người.

CHƯƠNG 4: TỔNG KẾT

Như vậy, nhóm đã hoàn thành xong project code game MegaMan của kì học lần này. Thông qua bài tập lớn này, giúp cho nhóm có thêm nhiều bài học, kinh nghiệm cũng như kĩ năng lập trình. MegaMan là một tựa game tương đối phức tạp, tuy nhiên thông qua nó mà nhón nhận ra những điểm được và chưa được.

Thứ nhất, kĩ năng lập trình được cải thiện, xử lí tốt được các sự kiện cũng như hiểu biết thêm về thuật toán khi gặp trường hợp cụ thể.

Thứ hai, hoàn thiện thêm về mặt kiến thức cũng như công cụ mà thầy đã dạy trong suốt kì học vùa qua.

Thứ ba, hoàn thiện thêm được khả năng làm việc nhóm, khả năng phối hợp.

Tuy nhiên, trong bài tập lớn lần này, nhóm nhận ra được những thiết sót vì gameplay chưa đủ cuốn hút, cũng như màn chơi còn sơ sài, cũng một phần do tính chất game tương đối nặng và thời gian ngắn, nên những thiếu sót mong được thầy thông cảm.

CHƯƠNG 5: TÀI LIỆU THAM KHẢO

https://www.youtube.com/watch?v=IDN-hIEFAcE&t=345s

https://www.youtube.com/watch?v=UkQWBfTCVbw&list=PLOgQJY7VjpBTYqgm46gdO7NEQMT72zrH7

https://yandex.com/images/?utm_source=yandex&utm_medium=com&utm_campaign=morda

HẾT