HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

2D Diffusion Limited Aggregation

NGUYỄN TRUNG DŨNG	20226030
NGUYỄN PHƯƠNG ANH	20226011
NGUYỄN NGỌC HƯNG	20226044
NGUYỄN LÊ ĐĂNG KHOA	20226049

Semester: 20232
Course: IT3100E
Class ID: 147839

Supervisor: Dr. Trần Thế Hùng ________________

Hanoi, 06/2024

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

OOP	Object-oriented Programming
GUI	Graphic user interface
IT	Information technology

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This project is a starting step for our group in applying object-oriented programming(OOP). For this semester, we have developed an application demonstrating virus within its information, structure and how the virus affect the host. We have leanrt and applied 4 properties of OOP: inheritance, polymophism, encapsulation and abstration as well as JavaFX to create graphic user interface(GUI). Users can have more knowledge about viruses after using this app and have several analyzations to compare. We choose this Virus demonstration topic because science in general and viruses in detail is an unfamiliar field for IT students and having lots of information to explore.

DUTY ROSTER

No.	Job	Name
1	Coding	Trung Dung and Ngoc Hung
2	Research	Phuong Anh and Dang Khoa
3	Writing report	Trung Dung and Dang Khoa
4	Getting Data	Phuong Anh
5	Making slides	Dang Khoa
6	Presentation	All of us

CHAPTER 1. INTRODUCTION

1.1 Introduction

Object-Oriented Programming (OOP) is essential in modern software development due to its ability to improve code modularity, maintainability, and reusability. By organizing software design around objects that combine data and behavior, OOP makes it easier to manage complex systems and allows for more intuitive mapping of real-world scenarios into code. This is particularly useful in scientific research, where OOP facilitates the creation of robust simulations and models that can be easily modified and extended.

IT students need to learn OOP because it is fundamental to modern software development. OOP's principles of encapsulation, inheritance, and polymorphism allow students to create modular, maintainable, and reusable code, which are essential skills for developing complex software systems.

To practically learning OOP, we have done a project using Java and Eclipse IDE aiming to introduce viruses for user. Our application uses all properties that we have learnt from this course.

1.2 Problem Statements

Virus is a topic in medical field which is unfamiliar with students studying in SOICT, hence we choose this topic to solve this problem and bring the knowledge about viruses closer to students. To create an user-friendly and easy to manage app, we use JavaFX within basic Java properties.

1.3 Objectives

In this project, we aim to build a visual app for improving knowledge about viruses. We divide the collections into 2 types: Enveloped and Non-enveloped virus to explore. After the projects, we can master the theory and how OOP function. Moreover, users can have more information about viruses.

CHAPTER 2: UML diagram and Use case

2.1 Use case

To ideate the project, we come up with ideas helping users easily approach the information from our app. Therefore, this app is user-friendly and having basic functions for newbie students in this medical field.

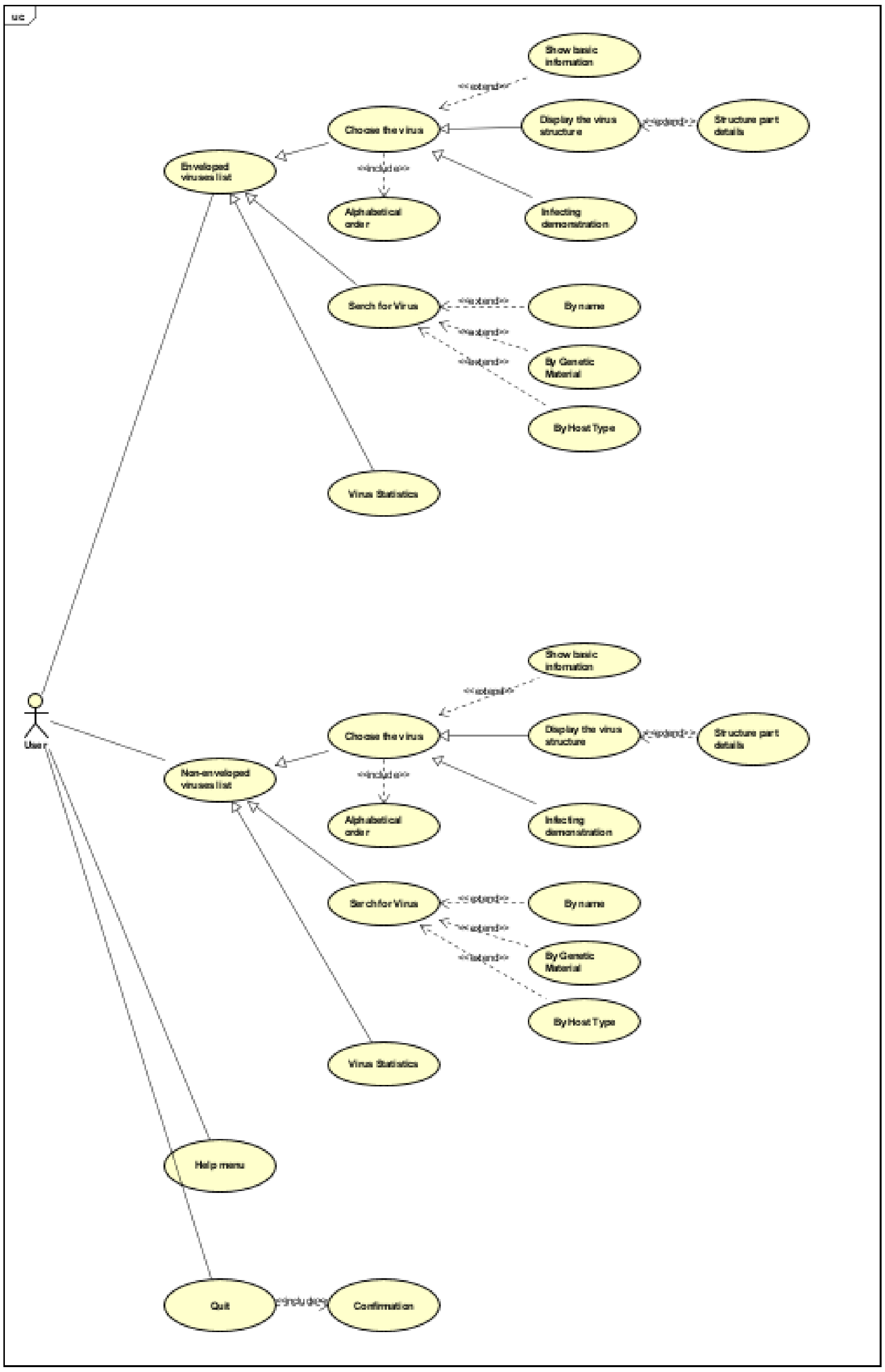
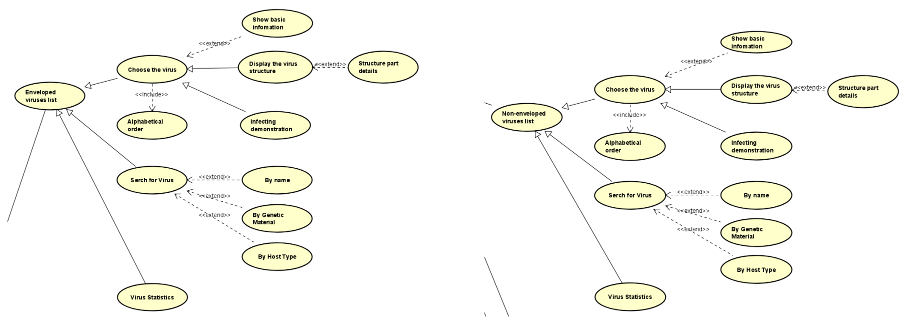


Figure 2.1 Overall Use Case Diagram

As can be seen in figure 2.1, this application is similar to a dictionary or a collection of virus as there is only interaction between users and the app without moderator. There are three main pages that users can open: Enveloped Virus page, Non-enveloped Virus page, Help page as well as a quit button.

2.1.1 Virus menu Pages

There are two pages to see the viruses in different types, enveloped and non-enveloped virus.



(a) Enveloped Page **(b) Non-enveloped page**

Figure 2.2 Use case diagram of content pages

Both pages have similar content as displaying the virus menu, showing its information and how it affect the host by a video. Moreover our application also supports searching by name, genetic material and host type. For further research we also implement a virus statistics section to generalize data.

2.1.2 Help and Quit function

If user have questions in using the app, they can enter the help section. Additionally, there is also a quit button with confirmation to shut the app down.

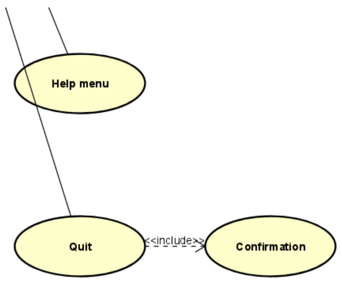


Figure 2.3 Help and Quit use case

2.2 Class Diagram

With respect to the feature listed in the use case diagram (figure 2.1) we have developed the app and this is our class diagram in figure 2.4

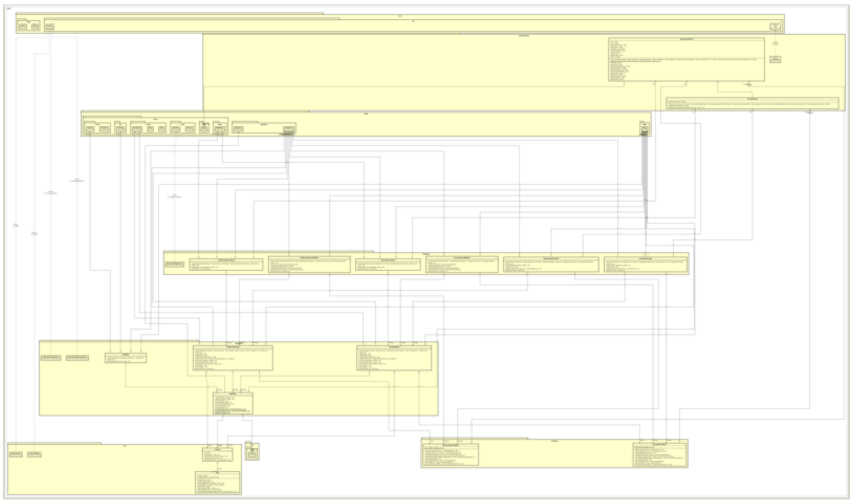
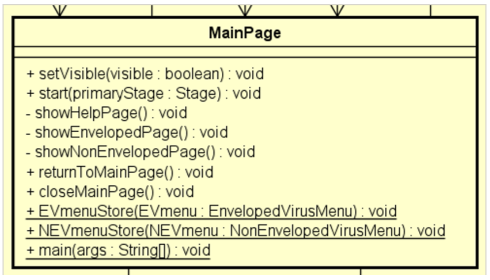


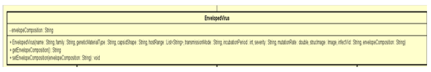
Figure 2.4 Class Diagram

This is a complicated class diagram autog-generated by the extension from Asthah.

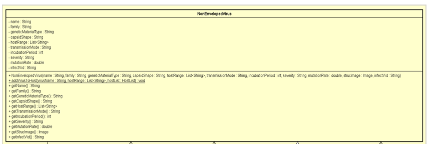
To see in details, we show you the class Main Page, Enveloped Page and Non-Enveloped Page in figure 2.5


MainPage
+ setVisible(visible : boolean) : void
+ start(primaryStage : Stage) : void
- showHelpPage() : void
- showEnvelopedPage() : void
- showNonEnvelopedPage() : void
+ returnToMainPage() : void
+ closeMainPage() : void
+ EVmenuStore(EVmenu : EnvelopedVirusMenu) : void
+ NEVmenuStore(NEVmenu : NonEnvelopedVirusMenu) : void
+ main(args : String[]) : void

(a) Main Page


EnvelopedVirus
- envelopeComposition : String
+ EnvelopedVirus(name : String, family : String, geneticMaterialType : String, capsidShape : String, hostRange : List<String>, transmissionMode : String, incubationPeriod : int, severity : String, mutationRate : double, strucImage : Image, infectVid : String, envelopeComposition : String)
+ getEnvelopeComposition() : String
+ setEnvelopeComposition(envelopeComposition : String) : void

(b) Enveloped Page


NonEnvelopedVirus
- name : String
- family : String
- geneticMaterialType : String
- capsidShape : String
- hostRange : List<String>
- transmissionMode : String
- incubationPeriod : int
- severity : String
- mutationRate : double
- infectVid : String
+ NonEnvelopedVirus(name : String, family : String, geneticMaterialType : String, capsidShape : String, hostRange : List<String>, transmissionMode : String, incubationPeriod : int, severity : String, mutationRate : double, strucImage : Image, infectVid : String)
+ addVirusToHost(virusName : String, hostRange : List<String>, hostList : HostList) : void
+ getName() : String
+ getFamily() : String
+ getGeneticMaterialType() : String
+ getCapsidShape() : String
+ getHostRange() : List<String>
+ getTransmissionMode() : String
+ getIncubationPeriod() : int
+ getSeverity() : String
+ getMutationRate() : double
+ getStrucImage() : Image
+ getInfectVid() : String

(c) Non-enveloped page

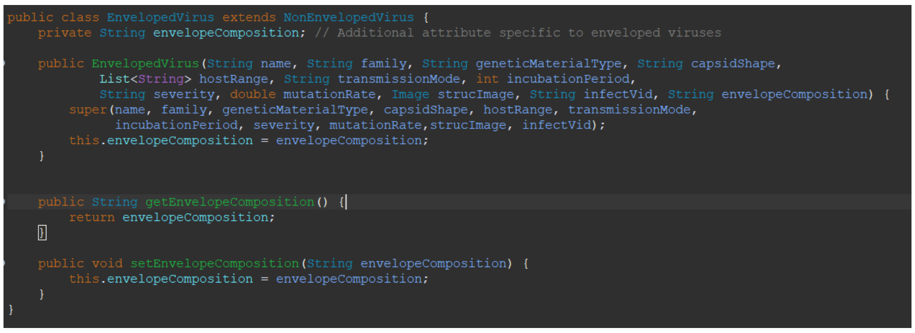
Figure 2.5 Class diagram of content pages

Class Diagram is an important step to review and determine the uses of functions and classes. This is a design to understand all attributes methods without reading codes.

CHAPTER 3: OBJECT-ORIENTED TECHNIQUES

3.1 Inheritance

The difference between enveloped and non-enveloped virus is the envelope composition, hence we set the Enveloped Virus inherits all the attributes from Non-enveloped Virus and add the envelope composition attribute.



```java
public class EnvelopedVirus extends NonEnvelopedVirus {
    private String envelopeComposition; // Additional attribute specific to enveloped viruses

    public EnvelopedVirus(String name, String family, String geneticMaterialType, String capsidShape,
            List<String> hostRange, String transmissionMode, int incubationPeriod,
            String severity, double mutationRate, Image strucImage, String infectVid, String envelopeComposition) {
        super(name, family, geneticMaterialType, capsidShape, hostRange, transmissionMode,
             incubationPeriod, severity, mutationRate,strucImage, infectVid);
        this.envelopeComposition = envelopeComposition;
    }


    public String getEnvelopeComposition() {
        return envelopeComposition;
    }

    public void setEnvelopeComposition(String envelopeComposition) {
        this.envelopeComposition = envelopeComposition;
    }
}
```

Figure 3.1 Enveloped Virus class

To run this application, the main page, which opens when we run the program, inherits the application from javafx.application.Application

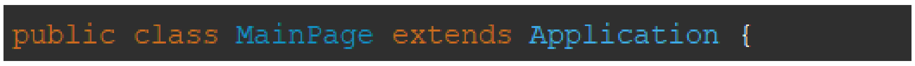
Stages like EnvelopedVirusPage or EnvelopedVirusMenu also inherit from Stage in javafx.stage.Stage, which is a primary window in JavaFX applications. This allows inheriting all the properties and methods of a ‘Stage‘, such as setting the scene and displaying the window.

3.2 Encapsulation

Every attributes and method is logically implemented. For example, in the EnvelopedVirusMenu class (figure 3.3), max items in menu is can not change and set as private, while objects in the menu is public as well as other functions like adding, searching and sort.

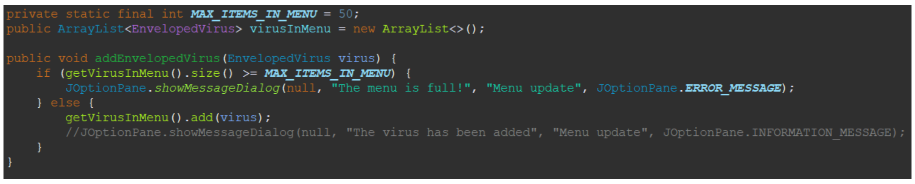
3.3 Polymorphism

Polymorphism is the ability of any data to be processed in more than one form and this is the key power of object oriented. For example, with the same name and different



```java
public class MainPage extends Application {
```

Figure 3.2 Main Page inherits Application

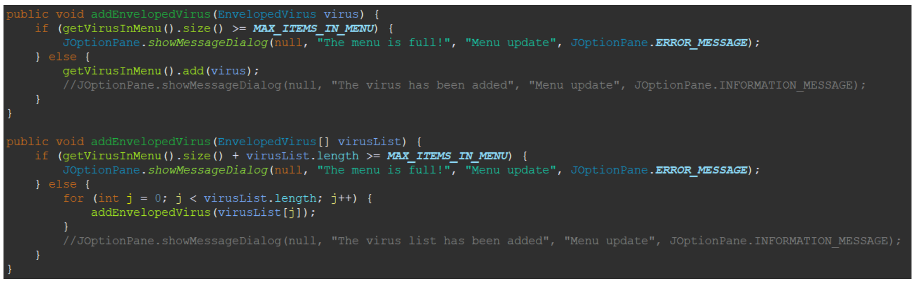


```java
private static final int MAX_ITEMS_IN_MENU = 50;
public ArrayList<EnvelopedVirus> virusInMenu = new ArrayList<>();

public void addEnvelopedVirus(EnvelopedVirus virus) {
    if (getVirusInMenu().size() >= MAX_ITEMS_IN_MENU) {
        JOptionPane.showMessageDialog(null, "The menu is full!", "Menu update", JOptionPane.ERROR_MESSAGE);
    } else {
        getVirusInMenu().add(virus);
        //JOptionPane.showMessageDialog(null, "The virus has been added", "Menu update", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Figure 3.3 Encapsulation in EnvelopedVirusMenu

attributes, we can create different methods like adding in the Menu class(figure 3.4)

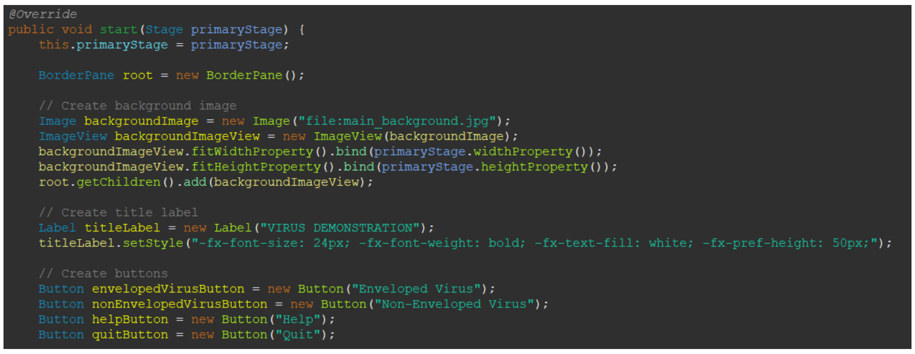


```java
public void addEnvelopedVirus(EnvelopedVirus virus) {
    if (getVirusInMenu().size() >= MAX_ITEMS_IN_MENU) {
        JOptionPane.showMessageDialog(null, "The menu is full!", "Menu update", JOptionPane.ERROR_MESSAGE);
    } else {
        getVirusInMenu().add(virus);
        //JOptionPane.showMessageDialog(null, "The virus has been added", "Menu update", JOptionPane.INFORMATION_MESSAGE);
    }
}

public void addEnvelopedVirus(EnvelopedVirus[] virusList) {
    if (getVirusInMenu().size() + virusList.length >= MAX_ITEMS_IN_MENU) {
        JOptionPane.showMessageDialog(null, "The menu is full!", "Menu update", JOptionPane.ERROR_MESSAGE);
    } else {
        for (int j = 0; j < virusList.length; j++) {
            addEnvelopedVirus(virusList[j]);
        }
        //JOptionPane.showMessageDialog(null, "The virus list has been added", "Menu update", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Figure 3.4 Methods with same name and different attributes

Another example for polymorphism in this project is overiding of the start method in the Main page. As the Main page inherit from the Application, we need to modify the start method to meet our demand.



```java
@Override
public void start(Stage primaryStage) {
    this.primaryStage = primaryStage;

    BorderPane root = new BorderPane();

    // Create background image
    Image backgroundImage = new Image("file:main_background.jpg");
    ImageView backgroundImageView = new ImageView(backgroundImage);
    backgroundImageView.fitWidthProperty().bind(primaryStage.widthProperty());
    backgroundImageView.fitHeightProperty().bind(primaryStage.heightProperty());
    root.getChildren().add(backgroundImageView);

    // Create title label
    Label titleLabel = new Label("VIRUS DEMONSTRATION");
    titleLabel.setStyle("-fx-font-size: 24px; -fx-font-weight: bold; -fx-text-fill: white; -fx-pref-height: 50px;");

    // Create buttons
    Button envelopedVirusButton = new Button("Enveloped Virus");
    Button nonEnvelopedVirusButton = new Button("Non-Enveloped Virus");
    Button helpButton = new Button("Help");
    Button quitButton = new Button("Quit");
```

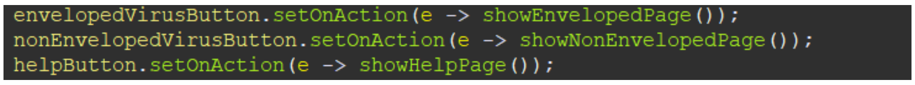
Figure 3.5 Overiding the "start" method

3.4 Aggregation

In Java, an Aggregation is a particular kind of connection between two classes. This connection between two or more entities is shown as a “has-a” relationship. In this project, we have the relation between the Virus menu class and the Virus class is aggregation as if we delete Virus class, the menu class still stays and reverse.

3.5 Composition

Composition in Java is a type of relationship between classes that shows a strong connection or association. It is also referred to as "Belongs-To association". The composite object cannot exist independently on its own, unlike association relationship. There are various cases in this project use composition such as the MainPage have the EnvelopedVirusPage, NonEnvelopedVirusPage and HelpPage (figure 3.6; or in the EnvelopedVirusPage we have the EnvelopedVirusInfo, EnvelopedVirusDemo and EnvelopedVirusStatistics.

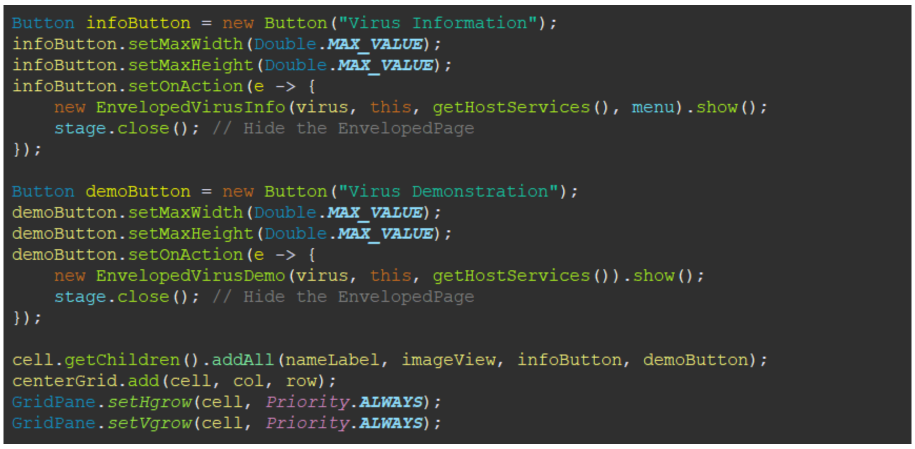


```
envelopedVirusButton.setOnAction(e -> showEnvelopedPage());
nonEnvelopedVirusButton.setOnAction(e -> showNonEnvelopedPage());
helpButton.setOnAction(e -> showHelpPage());
```

Figure 3.6 Composition in Main page

3.6 Interface

In the context of user interfaces, an interface in OOP defines a set of methods related to user interactions, such as handling button clicks, text input, or search bar functionality. For instance, in JavaFX, interfaces like EventHandler are used to create classes that handle specific events, such as a button press or a search query submission (figure 3.7). By implementing these interfaces, different components can respond to user actions in a consistent and interchangeable manner, enhancing the modularity and flexibility of the application's design.



```
Button infoButton = new Button("Virus Information");
infoButton.setMaxWidth(Double.MAX_VALUE);
infoButton.setMaxHeight(Double.MAX_VALUE);
infoButton.setOnAction(e -> {
    new EnvelopedVirusInfo(virus, this, getHostServices(), menu).show();
    stage.close(); // Hide the EnvelopedPage
});

Button demoButton = new Button("Virus Demonstration");
demoButton.setMaxWidth(Double.MAX_VALUE);
demoButton.setMaxHeight(Double.MAX_VALUE);
demoButton.setOnAction(e -> {
    new EnvelopedVirusDemo(virus, this, getHostServices()).show();
    stage.close(); // Hide the EnvelopedPage
});

cell.getChildren().addAll(nameLabel, imageView, infoButton, demoButton);
centerGrid.add(cell, col, row);
GridPane.setHgrow(cell, Priority.ALWAYS);
GridPane.setVgrow(cell, Priority.ALWAYS);
```

Figure 3.7 Buttons in EnvelopedPage

CHAPTER 4: GRAPHICAL RESULTS

4.1 Main Page

Once we run the application, the Main page is displayed with a virus back-ground and four buttons leading to other stages (figure 4.1).

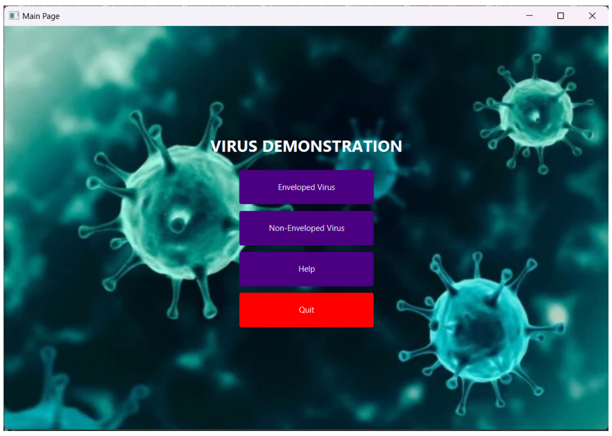


Figure 4.1 Main Page

4.2 Virus menu Pages

The two first buttons from the main page lead to two Virus menu pages: Enveloped Virus and Non-enveloped Virus. As these two pages are similar in graphic interface, I will show how Enveloped Virus page function only. Entering the page, users can see a list of virus in a scroll pane under the searching bar (figure 4.2).

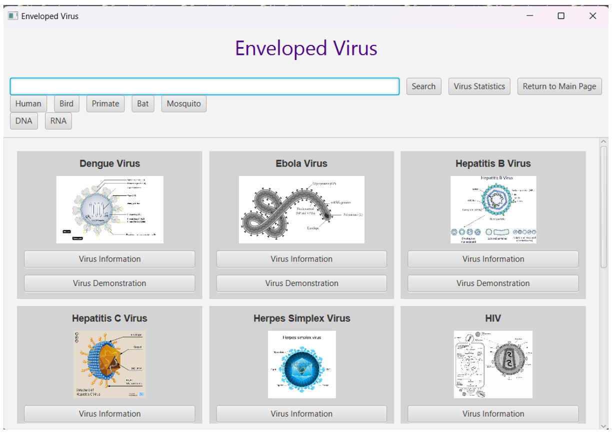
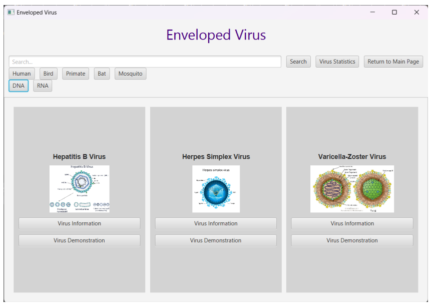
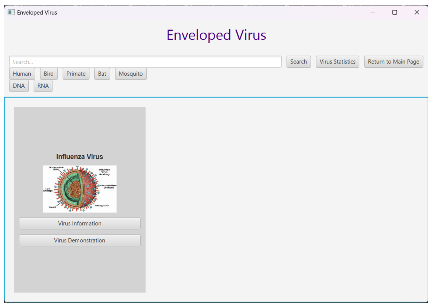


Figure 4.2 Enveloped Page

The scroll pane display all the virus in the menu within an image of the virus and two buttons lead to its information and demonstration. Above the menu is a search bar and we use three attributes to search for: name, host and genetic material type. Search result then be displayed in the pane (figure 4.3).



(a) Search by name

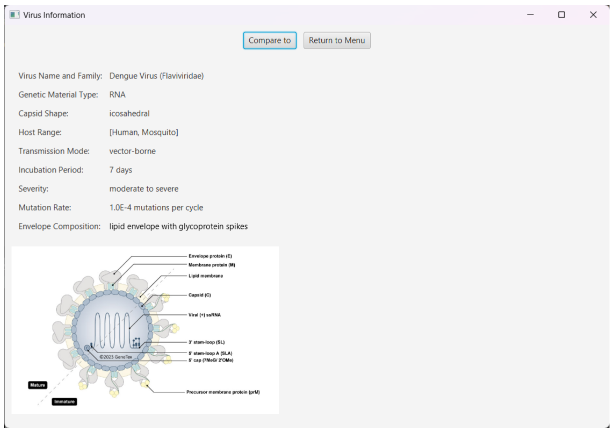


(b) Search by host

Figure 4.3 Searching result

4.3 Virus Information

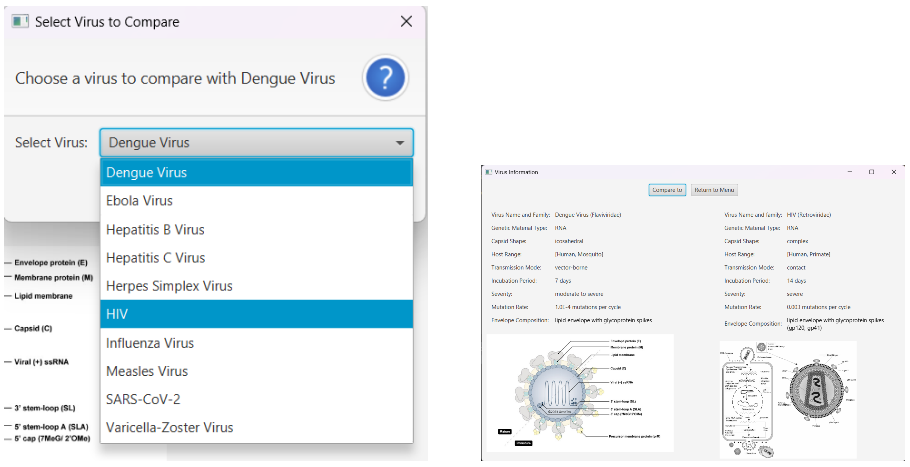
To know more information of a virus, users can click on the virus information button in its box. The virus information stage show you data included name, family, genetic material type, capsid shape, host range, transmission mode, incubation period, severity, mutation rate and enveloped composition if the virus is enveloped (figure 4.4). There is also an image of the Virus structure where you can click on it to zoom out.



Virus Name and Family: Dengue Virus (Flaviviridae)
Genetic Material Type: RNA
Capsid Shape: icosahedral
Host Range: [Human, Mosquito]
Transmission Mode: vector-borne
Incubation Period: 7 days
Severity: moderate to severe
Mutation Rate: 1.0E-4 mutations per cycle
Envelope Composition: lipid envelope with glycoprotein spikes

Figure 4.4 Virus information

To compare the displaying virus with another, users can click the "Compare to" button and choose which they want to compare to. A optional dialog of virus in the menu is displayed for choosing (figure 4.5a). Once two viruses are chosen, there information are displayed at the same time (figure 4.5b).



Select Virus to Compare

Choose a virus to compare with Dengue Virus

Select Virus: Dengue Virus

- Dengue Virus
- Ebola Virus
- Hepatitis B Virus
- Hepatitis C Virus
- Herpes Simplex Virus
- HIV
- Influenza Virus
- Measles Virus
- SARS-CoV-2
- Varicella-Zoster Virus

Virus Information

Compare to | Return to Menu

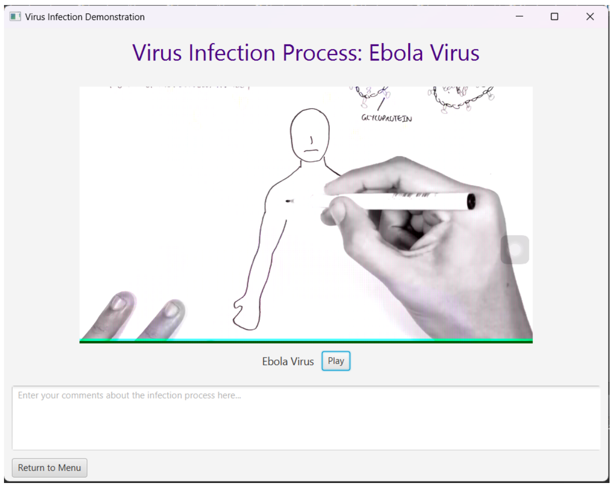
Virus Name and Family:	Dengue Virus (Flaviviridae)	Virus Name and family:	HIV (Retroviridae)
Genetic Material Type:	RNA	Genetic Material Type:	RNA
Capsid Shape:	icosahedral	Capsid Shape:	complex
Host Range:	[Human, Mosquito]	Host Range:	[Human, Primate]
Transmission Mode:	vector-borne	Transmission Mode:	contact
Incubation Period:	7 days	Incubation Period:	14 days
Severity:	moderate to severe	Severity:	severe
Mutation Rate:	1.0E-4 mutations per cycle	Mutation Rate:	0.003 mutations per cycle
Envelope Composition:	lipid envelope with glycoprotein spikes	Envelope Composition:	lipid envelope with glycoprotein spikes (gp120, gp41)

(a) Compare optional dialog **(b) Virus Comparison**

Figure 4.5 Comparison stage

4.4 Virus Demonstration

One other function in our application is virus demonstration. This stage contains a video show how the virus infect the host. To help user follow the content, we also install a play/pause button under the video (figure 4.6).



Virus Infection Demonstration

Virus Infection Process: Ebola Virus

Ebola Virus | Play

Enter your comments about the infection process here...

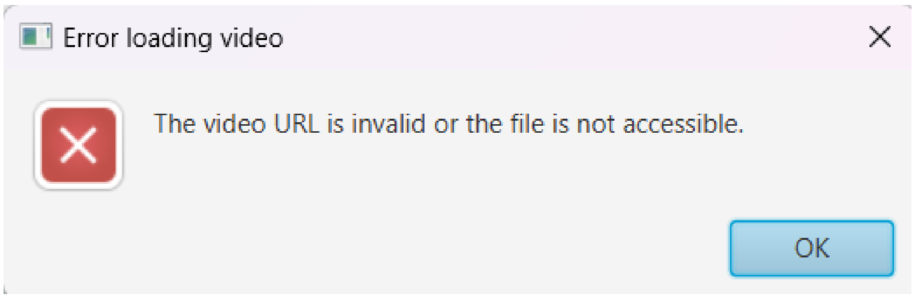
Return to Menu

Figure 4.6 Video of Virus Demonstration

However, some viruses do not have the demonstration video so if users want to open the demo, we make an alert that the video for this virus does not exist (figure 4.7).

4.5 Virus Statistics

The virus statistics page is where the users find graphs for some information: Virus severity, Genetic material type, Incubation Period and Mutation rate. We have installed bar graphs and a pie chart for the convenience in studying (figure 4.8).



The video URL is invalid or the file is not accessible.

Figure 4.7 Alert when the video does not exist

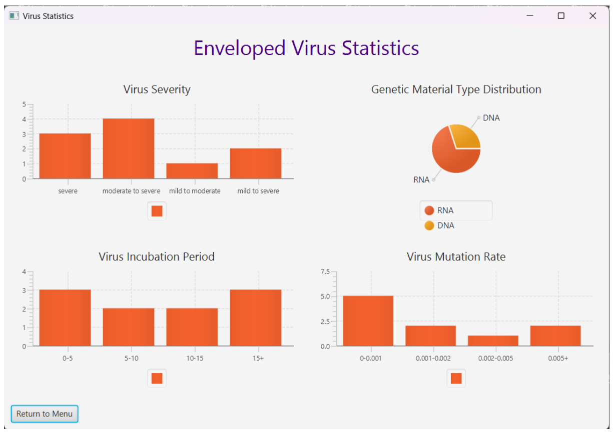
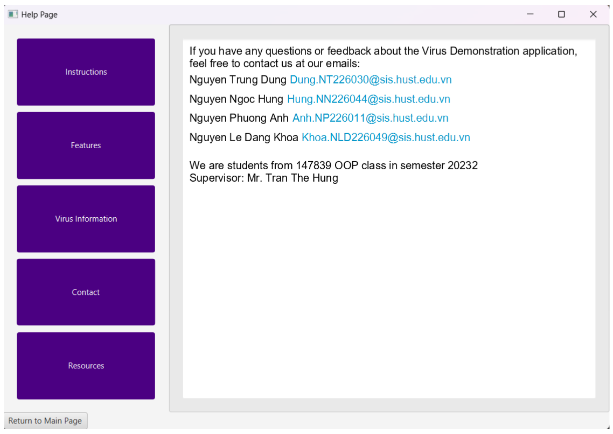


Figure 4.8 Virus Statistics

4.6 Help Page

In case of customer services, we implement a help page with five sections: Instructions, Features, Virus Information, Contact and Resources. These sections help users understand how the app works and the source of data for my project. Meanwhile, users know where to contact with the admins (figure 4.9).



If you have any questions or feedback about the Virus Demonstration application, feel free to contact us at our emails:

Nguyen Trung Dung Dung.NT226030@sis.hust.edu.vn

Nguyen Ngoc Hung Hung.NN226044@sis.hust.edu.vn

Nguyen Phuong Anh Anh.NP226011@sis.hust.edu.vn

Nguyen Le Dang Khoa Khoa.NLD226049@sis.hust.edu.vn

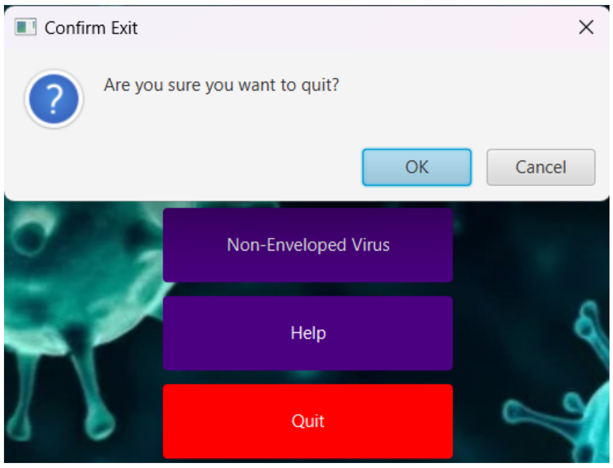
We are students from 147839 OOP class in semester 20232
Supervisor: Mr. Tran The Hung

Figure 4.9 Help page

4.7 Return and Quit

The Main page is the primary stage in this app, which immediately appears when we execute the code. All of the next three pages have a return to main button whenever they want to return. Meanwhile, every stages have a return button to its previous stage.

The Quit button in the Main page allows users close the app, however, they need to confirm before shutdown (figure 4.10).



Are you sure you want to quit?

Figure 4.10 Quit confirmation

4.8 Discussion

This project is logically visualize to be an user-friendly app. All class and stages are JavaFX based with imported libraries. For further presentation of this project, you can see the explaination video.