CE203 Application Programming Assignment 2(2020-2021)

Module Supervisor: Faiyaz Doctor

Program Testing Output

Registration Number: 1905770

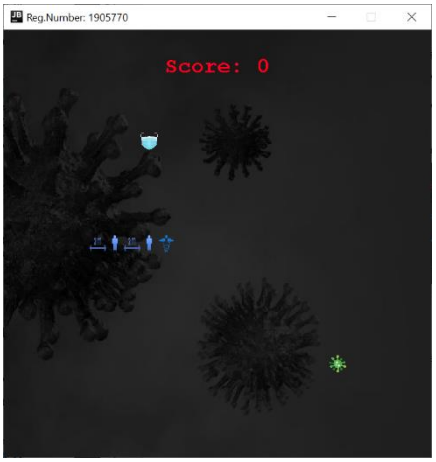
COVID-19 SNAKE GAME

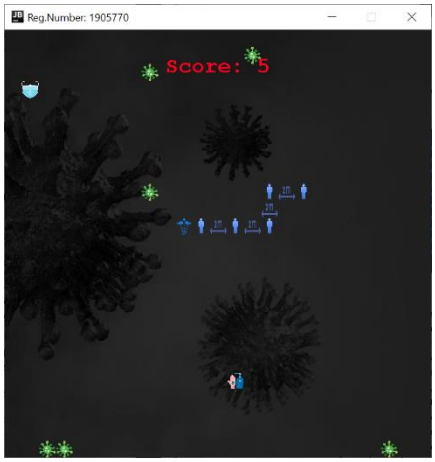
The Covid-19 Snake Game: The purpose of the game is to entertain the users as they play it, but also provide them some useful guidance and messages for Covid-19 pandemic. The game is based on the classic snake games we all know, but with Covid-19 theme. The goal of the user is to destroy the Covid-19. There are 2 stages. The first one is to collect masks, and the stage two, to collect vaccines. As you collect masks, covid-19 objects are generated to make your game harder. If you collect 15 masks without getting a covid, the masks will stop spawning and the vaccines will start to spawn. For every vaccine you get, a covid will be destroyed. The game finishes when you destroy all the covids. Each mask and vaccine worth a point. If you collide with a covid you lose. If the snake head touches the rest of the body, you lose as well. The faster you finish the game, the better score you get. Time is a part of the scoring system. At each five points the gel sanitizer/antiseptic is spawn on the panel. If you get it, you get an extra bonus point. If you do not get it on time, you must wait for the next 5 scored points because it will disappear (it will disappear if you get a mask or vaccine). At the end you can see the score you got with the time taken. You can also see the best 5 scores achieved. Snake object consists of three parts. The head is shown by the medical sign logo and the body is shown with the 2m distance icons and human icons (advice to keep distance).

Part 1: Demonstrating the features of the game (a to d):

a) Game was created by using a fixed grid (500x500). All the objects used in the game are drawn in the fixed grid of filled scores with a fixed size. All the objects of the game have the same size. The same size also applies for each square of the grid. The game consists of three screens.
-The first is the game start screen which is shown when you run the game:

-The second screen is the game playing screen which is shown when you start the game (by pressing space or clicking the panel):





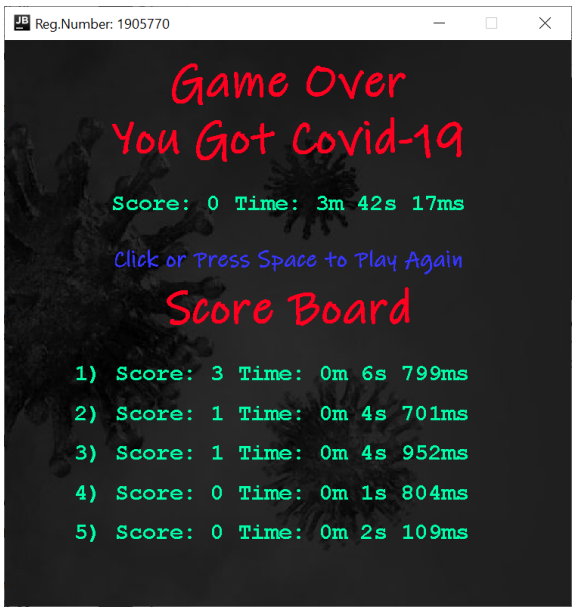
The second screen is where the objects are drawn. As you can see in the screenshots above, all the objects are drawn in a fixed size square of the grid. In the screenshots you can see that the snake can be manipulated to rotate according to the keyboard events. Also, the snake is manipulated to move on the panel. Every snake body object is saved in two arrays (snakeX[], snakeY[]), so it can be drawn accordingly. To move the snake, you have to change these coordinates every time, so a move on the panel can be made. To rotate the snake with the help of keyboard events, you have to change X or Y of the head of the snake to the direction you want (user changes startPos with key events). With these two ways the snake can be moved and rotated on the panel.

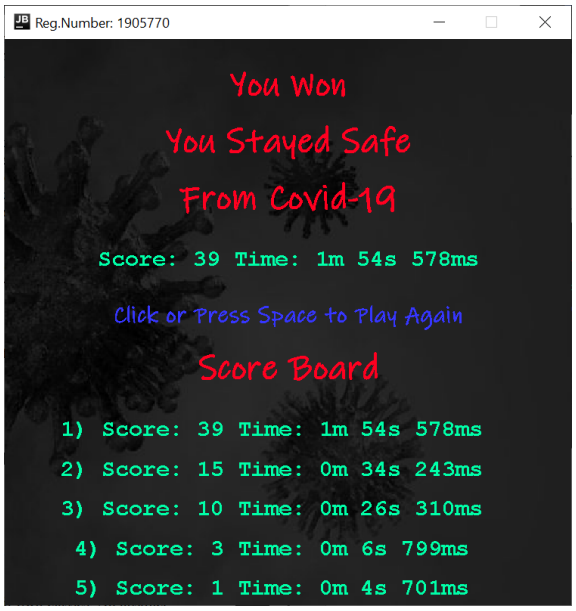
```
//make the snake to be able to move
public void snakeBodyMove(){
    //used for manipulating snake movement (manipulating snake object coordinates)
   for(int i=snakeBody; i>0; i--){
       snakeX[i] = snakeX[i-1];
       snakeY[i] = snakeY[i-1];
   }
   //how the snake will start at the start of the game and when keyboard key is pressed
manipulates snake object by rotating it. Object_size is the size of the fixed grid squares.
   if(startPos=="UP"){
       snakeY[0] = snakeY[0]-OBJECTS_SIZE;
   }
   else if(startPos=="DOWN"){
       snakeY[0] = snakeY[0]+OBJECTS_SIZE;
   }
   else if(startPos=="RIGHT"){
       snakeX[0] = snakeX[0]+OBJECTS_SIZE;
   }
   else if(startPos=="LEFT"){
       snakeX[0] = snakeX[0]-OBJECTS_SIZE;
    }
```

All the other objects (masks, vaccines, covids, antiseptics), are immovable objects on the panel. They are used for the game scoring

system. The snake can collide with all of these (mask also increments the body size of the snake).

-The third screen is the ending game screen which is shown when you end the game. End screen consists of two screens. The first screen is the win screen and the second one is the game over screen:

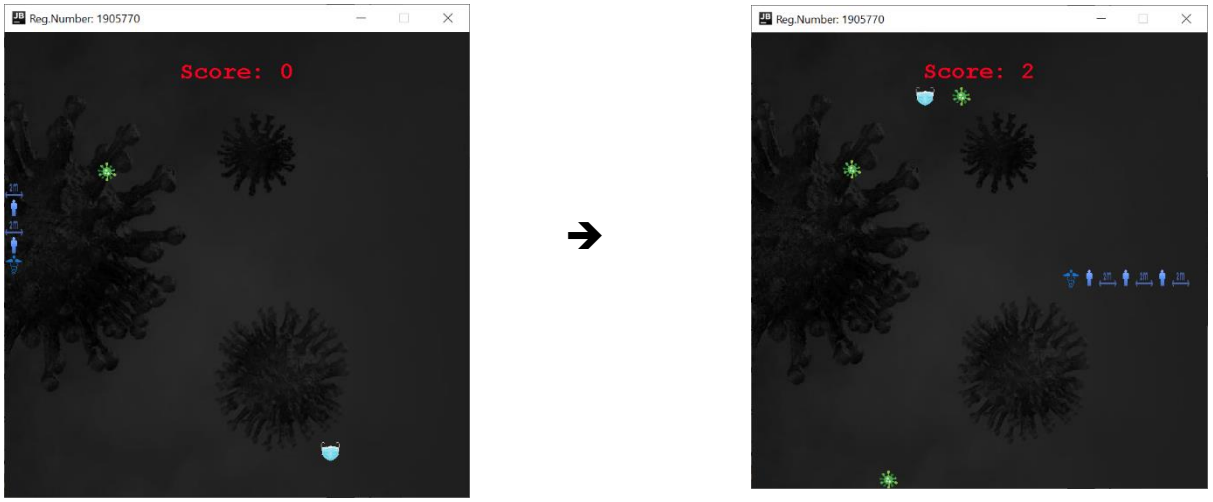




b) I have created an abstract class called GameObjects which is extended by five object classes (Covid, CovidAntiseptic, CovidMask, CovidVaccine, Snake). All the objects are drawn with the use of 2D image (g.drawImage() method). Each of these classes encapsulate a draw method, which is used to draw the objects on the panel. Furthermore, the object classes get the sizeX and sizeY of the panel which are always fixed for all the objects. When an object is created, the coordinate X and coordinate Y of the object are also passed in the class constructor (these coordinates are used to draw the object at a specific position on the panel; random generated). Draw image method gets the coordinate X and Y which is passed along with the fixed sizes of the panel. The fixed sizes of the panel are encapsulated using super() key.

All objects are drawn inside the fixed panel size but with random coordinates. Covid objects are also saved in the collection. Covid objects collection is incremented by one if a mask is collected and decremented by one if a vaccine is collected. To accomplish that, all the covid object should be saved in the collection (ArrayList). By saving them in a list, we can draw every covid which is in the list. When we destroy a covid object, it will be removed from the list. Therefore, it will be removed from the panel as well. If the ArrayList is empty, then the game is over. When game starts, a covid and a mask are drawn.

e.g., If you get two masks, two more covids will be created.

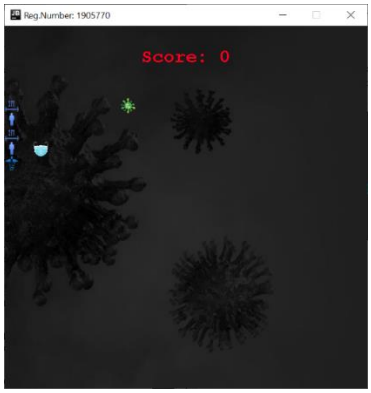


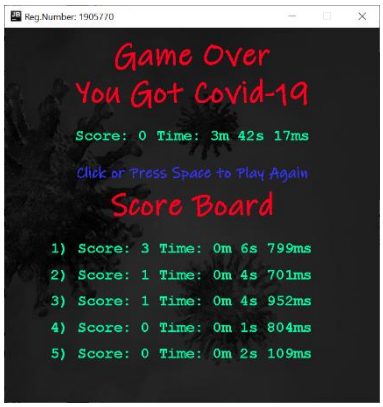
If you get all the masks (15), vaccines will start spawning. For each you get, one covid will be destroyed. At 15 collected masks, 16 covids are drawn. Next, the first vaccine will be drawn. If you get one vaccine, one covid will be destroyed (16 covids -> 15 covids).

c) Two event handler classes are created (KeyBoardListener, MouseClickListener). Keyboard listener is used to start the game, play the game, and play again the game (three screens exists as mentioned on a)). Mouse listener is used to start the game and play game again. Mouse listener click event is available on the start screen and on the end screen. If you click the starting panel you can start the game. If game is over, mouse event can be used to play the game again. On the game running screen nothing will happen if you click the panel.

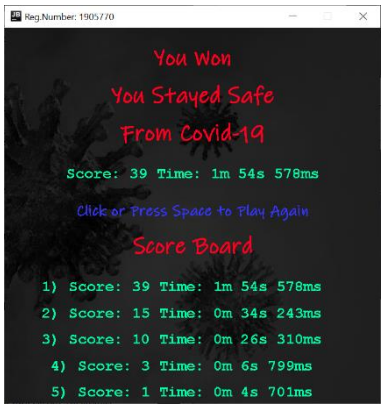
Click Starts Game

➔

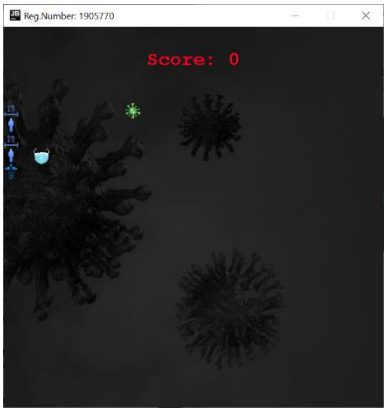




OR



Click Starts Game Again ➔

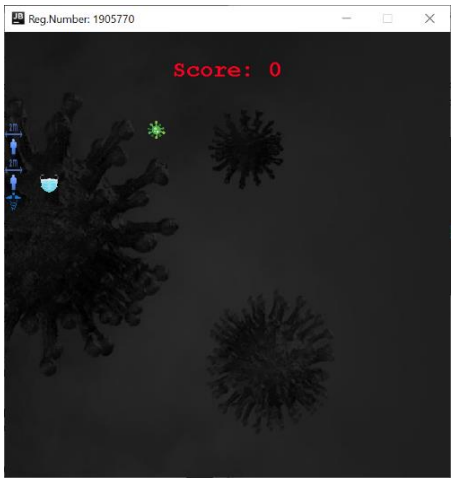


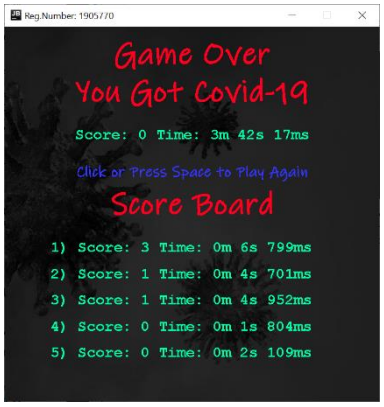
Keyboard event is using 5 keys; SPACE and the four arrows (UP,DOWN,RIGHT,LEFT). SPACE key has the same functionalities as mouse click event. The 4 arrows (UP,DOWN,LEFT,RIGHT) are used to control the snake. Cases control the pressed keys so; a continuous pressing key cannot be made. Also, cases have been used to prevent opposite arrows to be pressed (e.g., when snake goes up, down arrow cannot be used (applies for UP-DOWN, RIGHT-LEFT). E.g.:

```
case KeyEvent.VK_UP:
    //Cases to make sure that a no 180 degree can be made and not continuously press of a key can happen
    if(game.startPos != "DOWN"){
        if(up==true) {
            down=true;
            right=true;
            left=true;
            game.startPos = "UP";
            game.snakeBodyMove();
            up=false;
        }
    }
    break;
```

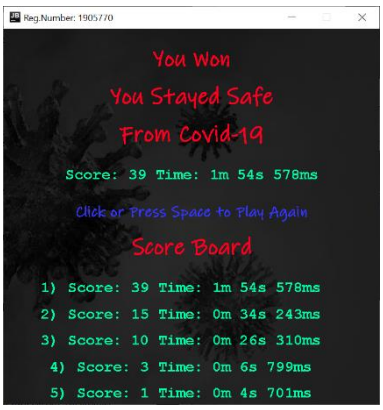
SPACE Starts Game

➔



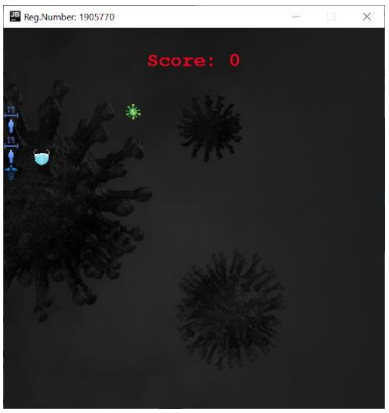


OR



SPACE Starts Game

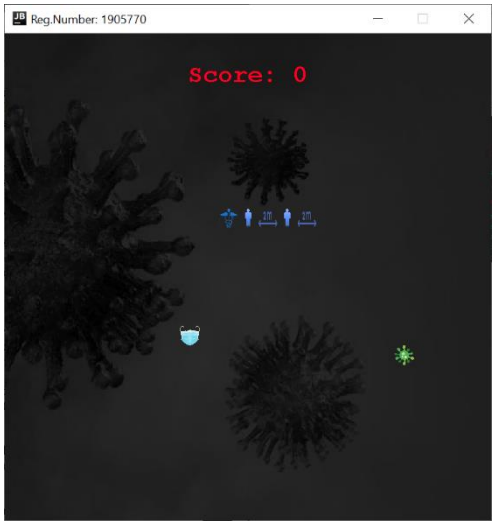
Again ➔



Snake goes down with down arrow:

Snake goes up with up arrow:

Snake goes left with left arrow:



Snake goes right with right arrow:

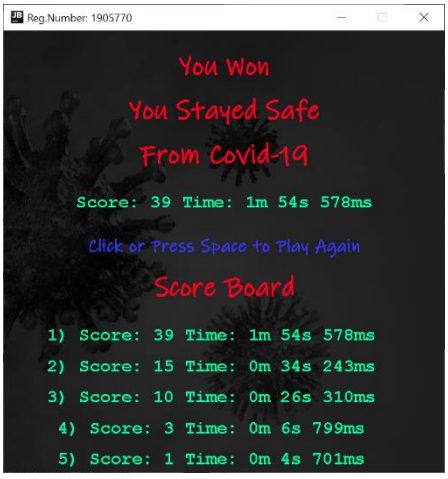
d) The game keeps records of scores via a text file. The created text file saves two records per game: score achieved, and time taken to finish the game. When the program runs, a new file is created (with the help of a method) or if it has already created, uses the same and appends the previous records. Every time the game is finished, the records are saved on a GameScore list (containing two strings and two integers for score and time) and at the same time list is written to the file. To get the 5 top scores we must sort the list and then display the top 5 records of it; With the use of another method, program reads the records of the file, splits the records to different variables and saves them in a new GameScore list. The new GameScore list is used to sort records, display the top 5 scores, and resave the sorted records in the file. The Collection. Sort() is used to sort the records. A new class (scoresCompare) which implements Comparator is used for the sorting. The compare method sorts the records first by the score achieved and then by time taken. We cannot use the same list created on the first method, as it saves the score instantly in the file. To display the top 5 scores achieved, program iterates five time through the second created list which has the sorted records. The program draws the records of the GameScore list on the screen with the g.drawString() method.

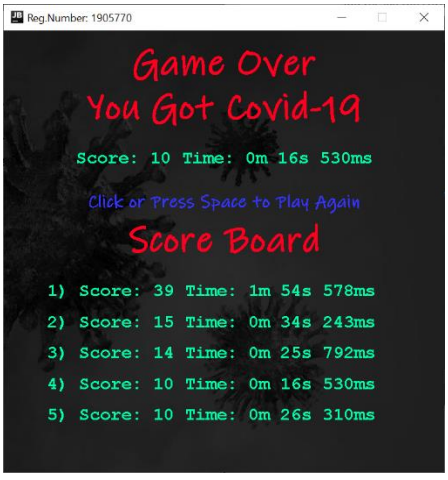
e.g.

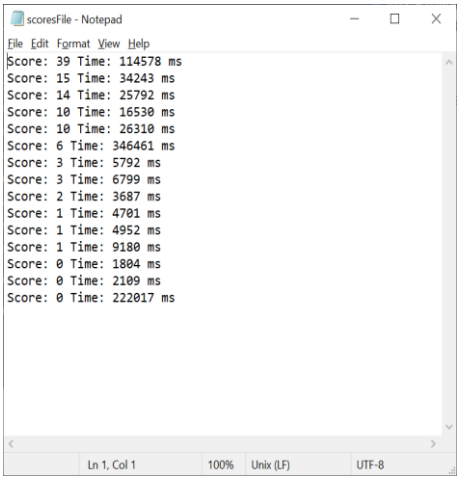
-A new high score achieved and displayed on the top of scoreboard.

-If score is the same, sorting according to the time taken (see 4th and 5th records on the scoreboard).

-Txt file created. All records are saved. Records are sorted as well.







Part 2: Implement scoring with SQL DB using JDBC:

The game could extend implemented scoring feature so it can store player scores in an SQL database using JDBC. First, we should import the relevant classes from java.sql. We need to load a JDBC driver for the DB system, establish a connection and last, create a statement object which will be used to execute SQL statements. There are different systems to load a driver class for the database system. The driver class for MySQL is called→com.mysql.jdbc.Driver. After that we need to connect to the database. This can be done by using getConnection method from the DriverManager class in java.sql package. In the getConnection method, the path often consists of the database URL, the user id and the password as an authentication.
-Connection conn = DriverManager.getConnection(<path>);
In order to establish the connection with the database, we should use try/catch blocks. SQLException will be thrown if the connection cannot be established or a ClassNotFoundException will be thrown if the driver cannot be found. Next, we should create a statement object to execute SQL statements. This can be done by using createStatement method. An SQLException could be thrown again (try/catch blocks required again).
-statement = connection.createStatement();
When the statement has been created, we can use executeUpdate() method to write the update query we want to execute. A SQLException could be thrown so a try/catch block should be used.
In the Covid-19 Snake Game, we assume that a database exists, and the connection is established. We also assume that the table has auto incremented id column as a primary key (assume that the table exists as well with the name 'scores'). We could declare the connection and statement objects in the Game class where most of the variables are declared.
The connection's try/catch block could be put in the Game's class constructor as it is run when the game is compiled. It is also used as the main class of the whole program. Under the connection's try/catch blocks we could put the statement's try/catch block. ExecuteUpdate() method should be called inside the readScoresFile method in the EndGame class. In there, we read the file and we save the records of the file (for every line) in variables so we can add them in the GameScore list. We can just add a line of code after we add the records in the list to insert the values (records) to the DB table. After the while loop is finished, we can call a new method (call it closeSQL(Game game)) in which we close the connection and the statement (should be put in try/catch block as a

SQLException can be thrown). The readScoresFile method should also throw a SQLException along with the IOException.
Before getting in the readScoresFile method, methods winGame() and endgame() are called, so SQLException should be thrown in the methods as well. We should put one more catch block for SQLException when we call the winGame and endgame methods (inside paintComponent method).

Pseudo code to extend implemented scoring feature so it can store player scores in an SQL database using JDBC :

Assume that the database table is called scores.

import java.sql.*;

Declare in the Game class the two objects (connection and statement)

Connection connection = null;

Statement statement = null;

In the Game() constructor:

try/catch block for connection.

```
try {
    connection = Connection conn = DriverManager.getConnection( <path> );
    path consists of databaseURL, user id, password.
        }
 catch {
         ClassNotFoundException
         driver not found exception.
        }
 catch  {
         SQLException
         failed to connect to the database exception.
        }
```

try/catch block for statement.

```
try {
      statement = connection.createStatement();
        }
catch {
         SQLException
         fail to access database exception.
         }
```

inside the readScoresFile method in the EndGame class
and inside the while loop
right after adding records in the gameScore list,
we can access the statement object with game.statement

game.statement.execute ("INSERT INTO scores (idColumn, scoreStringColumn, scoreColumn, timeStringColumn, timeTakenColumn) VALUES (?,"+scoreString+", "+score+", '"+timeString+"', "+timeTaken+")"); *//id is auto incremented*
after the while loop is finished
closeSQL(game); *//method is called.*

Inside closeSQL method
method closeSQL(Game game){
try {
close statement
close connection
}
catch {
SQLException
problems closing connection exception
}
}

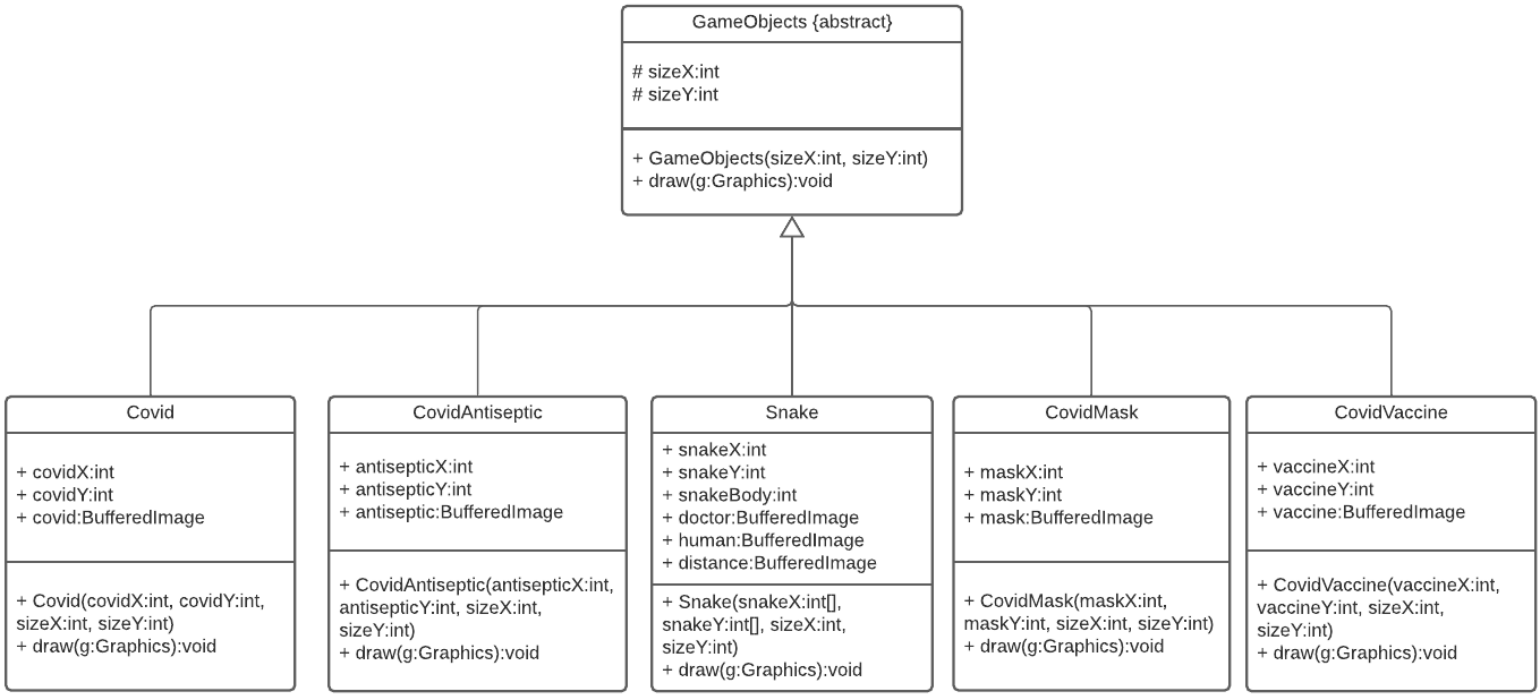
We could create the table, if it doesn't exist by adding in the Game() constructor, after statement's try/catch block, a new try/catch block(SQLException should be catch) with the following code of line:

statement.executeUpdate("CREATE TABLE IF NOT EXISTS scores("
+ "ID INT PRIMARY KEY AUTO_INCREMENT, " *//auto increment id*
+ "stringScore VARCHAR (20) NOT NULL, "
+ "intScore INT NOT NULL, "
+ "stringTime VARCHAR (20) NOT NULL,
+ " intTime TIME NOT NULL")");

Part 3: Underlying code feature based on design patterns:

A design pattern is a description of a known problem in software design along with a solution.
One possible design pattern is GRASP Pattern: Polymorphism. In the snake covid-19 game, polymorphism is used for the game objects. A super class (GameObjects) class is created, which is extended by five sub-classes (Covid, CovidMask, CovidVaccine, CovidAntiseptic, Snake). The five classes inherit a draw method that allows the class objects to be drawn on the panel (polymorphism). These classes also use super() key to inherit the width and height of the panel. The design patter's responsibility is to make the game capable to draw the objects on the panel. It is collaborating with the Game

class; initialisation of the objects and drawing them as it is the class used for the game panel.

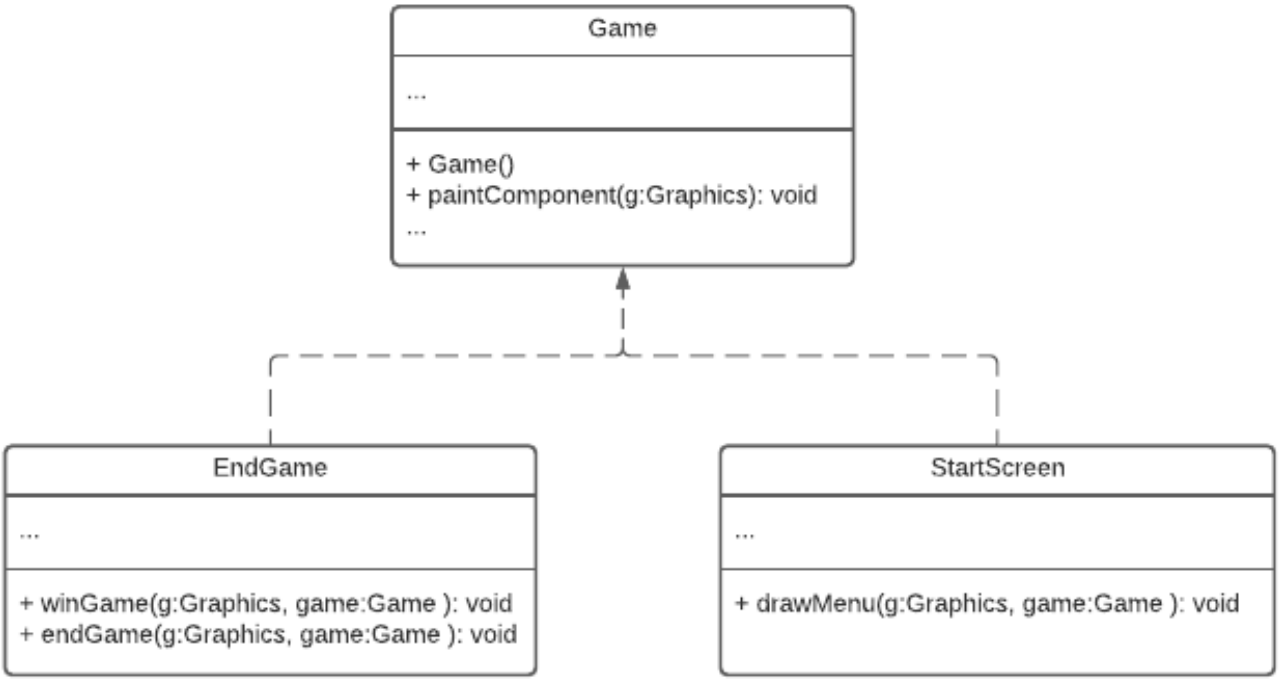


GameObjects super-class:
GameObjects super-class is an abstract class. It contains a draw method that will be inherited from all sub-class (polymorphism). Also, the sizeX and sizeY will also be inherited. These two sizes will be used as the panel dimensions.
Sub-classes (CovidVaccine,CovidMask,CovidAntiseptic,Covid,Snake):
Draw method is inherited. With the use of drawImage we can draw the objects on the panel but with a 2D image (images loaded from a file and saved as a Buffered Image object). The five classes get four parameters in the constructor (objectX,objectY,sizeX,sizeY). ObjectX and objectY are used as the draw coordinates on the panel, while the sizeX,sizeY are the dimensions of the panel. The same thing applies for the other four sub-classes as well (different variable names for x, y object coordinates).
Snake sub-class constructor takes one more parameter (snakeBody). SnakeBody variable is used to draw the length of the snake. Three images are also created in this class to show the head of the snake (doctor), and the two

other images to show the body of the snake (human and distance sign alternately).

Another possible design pattern is GRASP Pattern: High Cohesion. In the covid-19 snake game, complexity is kept manageable because some of the responsibilities of the program are assigned under other classes (subset of methods are grouped separately in other classes). In the game, the main class that is responsible for painting the panel (the one that extends JPanel), has two separate classes which are responsible for painting the start screen and the end screen (the game play screen is drawn in the main class). The main draw class in the game is Game class while the two other classes are StartScreen and EndGame (StartScreen used to draw start menu screen, and EndGame used to draw the two ending screens; win screen and game over screen).

UML Diagram:



In the given example of cohesion above, the public interface of the Game class is cohesive because the draw feature is related to the concept that the EndGame and StartScreen classes represent.

References:

1) Image of the head of the snake (doctor object): https://www.pngkey.com/png/full/33-337638_medicine-logo-png-1-medical-logo.png
2) Image of the body of the snake (distance object): https://cdn2.iconfinder.com/data/icons/the-new-normal-2/64/social_distancing_2m_distance_user-128.png
3) Image of the body of the snake (human object): https://www.freeiconspng.com/img/1927
4) Image of the covid object: https://freepikpsd.com/wp-content/uploads/2020/02/coronavirus_PNG-Images-Clipart.png
5) Image of the antiseptic object: https://cdn0.iconfinder.com/data/icons/covid-19-37/512/UseHandSanitizer-handantiseptic-handdisinfectant-covid19-disinfection-hygiene-prevention-512.png
6) Image of the mask object: https://i.pinimg.com/originals/4e/c1/19/4ec119f7a6cf0088dce44614696f15bb.png
7) Image of the vaccine object: https://images.squarespace-cdn.com/content/5a7adf55f6576ee0160b5a58/1523964614321-ZTGQTIY55KJNYQDNFVJZ/vaccine.png?content-type=image%2Fpng
8) Image of the game background: https://static.physoc.org/app/uploads/2020/03/30091715/COVID19-video-background-942x500.jpg
9) For basic coding for the snake a video was used for help (basic movement of the snake): https://www.youtube.com/watch?v=bI6e6qjJ8JQ&t=1276s