Introduction

OOP is a programming language paradigm that uses objects and classes as its core components, allowing the programmer to think in terms of real-life objects. This makes the code cleaner, reusable, maintainable and scalable. Grasping them is key to understanding how Java works. Basically, Java OOP concepts let us create working methods and variables, then re-use all or part of them without compromising security.

Graphical User Interfaces (GUIs) on the other hand, are mechanisms for allowing users to enter data in the most economical and straightforward manner possible. In many aspects the gui is the most important part of the program. The gui can determine if the user enjoys or hates using the program. If the user hates using the program he will stop using it. Interesting that in industry there are generally no gui expert; all programmers are expected to be able to write good and effective gui.

In this individual project, I have decided to implement my knowledges of OOP Concepts and GUI in Java by recreating the classic game 'Snake'. The GUI designed was simple and straightforward, inherits the quality of a minimalist. I choose to do snake because I belived its fun and challenging to do. It is algorithmic, interactive and fun to look at than a simple office tool.

Objective

1. To create a simple Graphical User Interface of a Snake Game.
2. To create a system that stores data efficiently and safely.
3. To create a system that allows user to manipulate, edit and delete stored data.
4. To study the current system identifying its inefficiencies.
5. To implement proper concepts and method in this Java Application

Project Scope

PROJECT

- Type of Project
- Name of Project
- Problem statements
- Deciding Objectives

CODING

- Implementation of Object Orientated Programming
- Array
- Inheritence
- Encapsulation
- Interface
- Abstraction
- Read and Write Implementation

GUI

- Improving mistakes
- Application of GUI interfaces
- Creation of PNG for game icons

REPORT

- Analyzing Project
- Creaying game manual
- GUI Screenshots

PRESENTATION

- Recheck system
- Finalization

Java Code

Main.java

```java
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JButton;

public class Main{

   public static void main(String [] args) {
      JFrame obj = new JFrame();
      Gameplay gameplay = new Gameplay();

      obj.setBounds(10, 10, 905, 700);
      obj.setBackground(Color.DARK_GRAY);
      obj.setResizable(false);
      obj.setVisible(true);
      obj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      obj.add(gameplay);

   }
}
```

Gameplay.java

```java
import java.util.Random;
import javax.swing.*;
import java.awt.*;
import java.awt.Font;
import java.awt.event.*;
import java.lang.Object;
import java.io.*;

public class Gameplay extends JPanel implements KeyListener,
ActionListener{

   private int[] snakexlength = new int[750];
   private int[] snakeylength = new int[750];

   private boolean left = false;
   private boolean right = true;
   private boolean up = false;
   private boolean down = false;

   private ImageIcon rightmouth;
   private ImageIcon upmouth;
   private ImageIcon downmouth;
   private ImageIcon leftmouth;
   private ImageIcon snakeimage;
   private ImageIcon titleImage;
   private ImageIcon enemyimage;
   private Random random = new Random();
   private Timer timer;
   private int lengthofsnake = 3;
   private int delay = 100;
   private int moves = 0;
   private int xpos = random.nextInt(34);
   private int ypos = random.nextInt(23);
   private int counter = 0;

   private JButton startButton;
   private JButton continueButton;
   private JPanel menu;

   private int[] foodxpos =
{25,50,75,100,125,150,175,200,225,250,275,300,325,350,375,400,425,450,4
75,500,525,550,575,600,625,650,675,700,725,750,775,800,825,850};
   private int[] foodypos =
{75,100,125,150,175,200,225,250,275,300,325,350,375,400,425,450,475,500,
525,550,575,600,625};
   private int score = 0;
   private int highscore = 0;
   private int z = 1;

   public Gameplay()
   {
      addKeyListener(this);
      setFocusable(true);
      setFocusTraversalKeysEnabled(false);
      timer = new Timer(delay, this);
      timer.start();
      try{
         FileInputStream saveFile = new FileInputStream("saveFile.sav");
         ObjectInputStream save = new ObjectInputStream(saveFile);
         highscore = (Integer) save.readObject();
         save.close();
      }
      catch(Exception exc){
         exc.printStackTrace();//if theres an error, print it out.
      }
   }

   public void paint(Graphics g)
   {
      //snake default position
      if(moves == 0)
```

```
{
   snakexlength[2] = 50;
   snakexlength[1] = 75;
   snakexlength[0] = 100;

   snakeylength[2] = 100;
   snakeylength[1] = 100;
   snakeylength[0] = 100;
}

//title image border
g.setColor(Color.WHITE);
g.drawRect(24, 10, 851, 55);

//title image
titleImage = new ImageIcon("snaketitle.jpg");
titleImage.paintIcon(this, g, 25, 11);

//Border for gameplay
g.setColor(Color.WHITE);
g.drawRect(24, 74, 851, 576);

//draw background
g.setColor(Color.black);
g.fillRect(25, 75, 850, 575);

//score
g.setColor(Color.white);
g.setFont(new Font("arial", Font.PLAIN, 14));
g.drawString("Score: "+score, 780, 30);

//highest score
g.setColor(Color.white);
g.setFont(new Font("arial", Font.PLAIN, 14));
g.drawString("Highscore: "+highscore, 780, 50);

//draw snake
rightmouth = new ImageIcon("rightmouth.png");
rightmouth.paintIcon(this, g, snakexlength[0], snakeylength[0]);
for(int i=0; i<lengthofsnake; i++)
{
   if(i==0 && right)
   {
      rightmouth = new ImageIcon("rightmouth.png");
      rightmouth.paintIcon(this, g, snakexlength[i], snakeylength[i]);
   }
   if(i==0 && left)
   {
      leftmouth = new ImageIcon("leftmouth.png");
      leftmouth.paintIcon(this, g, snakexlength[i], snakeylength[i]);
```

```
            }
            if(i==0 && up)
            {
               upmouth = new ImageIcon("upmouth.png");
               upmouth.paintIcon(this, g, snakexlength[i], snakeylength[i]);
            }
            if(i==0 && down)
            {
               downmouth = new ImageIcon("downmouth.png");
               downmouth.paintIcon(this, g, snakexlength[i], snakeylength[i]);
            }
            if(i!=0)
            {
               snakeimage = new ImageIcon("snakeimage.png");
               snakeimage.paintIcon(this, g, snakexlength[i], snakeylength[i]);
            }
         }

         //draw snake food
         enemyimage = new ImageIcon("enemy.png");
         enemyimage.paintIcon(this, g, foodxpos[xpos], foodypos[ypos]);
         if((foodxpos[xpos] == snakexlength[0]) && (foodypos[ypos] ==
snakeylength[0]))
         {
            score = score + 5;
            lengthofsnake ++;
            xpos = random.nextInt(34);
            ypos = random.nextInt(23);
         }

         //Start manu
         if(moves == 0)
         {
            g.setColor(Color.WHITE);
            g.drawRect(239, 199, 446, 176);
            g.setColor(Color.DARK_GRAY);
            g.fillRect(240, 200, 445, 175);

            g.setColor(Color.white);
            g.setFont(new Font("Tahoma", Font.PLAIN, 23));
            g.drawString("Feed Esther's Snake and make it grow!", 265, 255);

            g.setFont(new Font("arial", Font.BOLD, 15));
            g.drawString("Press any arrow key to move", 350, 320);

            g.setFont(new Font("arial", Font.BOLD, 15));
            g.drawString("Press 'R' to reset highscore", 350, 345);


            if(z == 0)
```

```
        {
          g.setColor(Color.DARK_GRAY);
          g.fillRect(255,220,420,50);
          g.setColor(Color.white);
          g.setFont(new Font("arial", Font.BOLD, 20));
          g.drawString("High score is Reset!", 355, 255);
        }

      }

      //gameover screen
      for(int i = 1; i<lengthofsnake; i ++)
      {
        if((snakexlength[i] == snakexlength[0]) && (snakeylength[i] ==
snakeylength[0]))
        {
          right = false;
          left = false;
          up = false;
          down = false;

          g.setColor(Color.WHITE);
          g.drawRect(284, 199, 306, 176);
          g.setColor(Color.DARK_GRAY);
          g.fillRect(285, 200, 305, 175);

          g.setColor(Color.RED);
          g.setFont(new Font("arial", Font.BOLD, 50));
          g.drawString("Game Over", 305, 275);

          g.setColor(Color.WHITE);
          g.setFont(new Font("arial", Font.PLAIN, 20));
          g.drawString("Press SPACE to restart", 330, 325);
          counter++;

        }
      }
      g.dispose();
    }

    @Override
    public void actionPerformed(ActionEvent e){
      timer.start();

      if(right){
        for(int r = lengthofsnake-1; r>=0; r--)
        {
          snakeylength[r+1] = snakeylength[r];
        }
        for(int r = lengthofsnake; r>=0; r--)
        {
          if(r==0){
            snakexlength[r] = snakexlength[r]+25;
          }
          else{
            snakexlength[r] = snakexlength[r-1];
          }
          if(snakexlength[r] > 850){
            snakexlength[r] = 25;
          }
        }
        repaint();
      }
      if(left){
        for(int r = lengthofsnake-1; r>=0; r--)
        {
          snakeylength[r+1] = snakeylength[r];
        }
        for(int r = lengthofsnake; r>=0; r--)
        {
          if(r==0){
            snakexlength[r] = snakexlength[r]-25;
          }
          else{
            snakexlength[r] = snakexlength[r-1];
          }
          if(snakexlength[r] < 25){
            snakexlength[r] = 850;
          }
        }
        repaint();
      }

      if(up)  {
        for(int r = lengthofsnake-1; r>=0; r--)
        {
          snakexlength[r+1] = snakexlength[r];
        }
        for(int r = lengthofsnake; r>=0; r--)
        {
          if(r==0){
            snakeylength[r] = snakeylength[r]-25;
          }
          else{
            snakeylength[r] = snakeylength[r-1];
          }
          if(snakeylength[r] < 75){
            snakeylength[r] = 625;
          }
        }
```

```java
            repaint();
        }


        if(down){
            for(int r = lengthofsnake-1; r>=0; r--)
            {
                snakexlength[r+1] = snakexlength[r];
            }
            for(int r = lengthofsnake; r>=0; r--)
            {
                if(r==0){
                    snakeylength[r] = snakeylength[r]+25;
                }
                else{
                    snakeylength[r] = snakeylength[r-1];
                }
                if(snakeylength[r] > 625){
                    snakeylength[r] = 75;
                }
            }
            repaint();
        }
    }

    @Override
    public void keyTyped(KeyEvent e) {
    }

    @Override
    public void keyPressed(KeyEvent e) {
        int key = e.getKeyCode();

        if(key == KeyEvent.VK_SPACE)
        {
         if(score > highscore){
            try{
              FileOutputStream saveFile = new FileOutputStream("saveFile.sav");
              ObjectOutputStream save = new ObjectOutputStream(saveFile);
              save.writeObject(score);
              save.close();
            }
            catch(Exception exc){
                exc.printStackTrace();
            }
         }
         try{
             FileInputStream saveFile = new FileInputStream("saveFile.sav");
             ObjectInputStream save = new ObjectInputStream(saveFile);
             highscore = (Integer) save.readObject();
```

```
     save.close();
   }
   catch(Exception exc){
     exc.printStackTrace();
   }

   counter=0;
   moves=0;
   score=0;
   lengthofsnake = 3;
   repaint();
     right = true;
     left = false;
     up = false;
     down = false;
  }

  if(counter == 0){
    if(key == KeyEvent.VK_RIGHT)
    {
      moves++;
      z++;
      right = true;
      if(!left)
      {
        right = true;
      }
      else
      {
        right = false;
        left = true;
      }
      up = false;
      down = false;
    }

    if(key == KeyEvent.VK_LEFT)
    {
      if(moves!=0){
        moves++;
        z++;
        left = true;
        if(!right)
        {
          left = true;
        }
        else
        {
          left = false;
          right = true;
```

```
            }
            up = false;
            down = false;
        }
    }

    if(key == KeyEvent.VK_UP)
    {
        moves++;
        z++;
        up = true;
        if(!down)
        {
            up = true;
        }
        else
        {
            up = false;
            down = true;
        }
        left = false;
        right = false;
    }

    if(key == KeyEvent.VK_DOWN)
    {
        moves++;
        z++;
        down = true;
        if(!up)
        {
            down = true;
        }
        else
        {
            down = false;
            up = true;
        }
        left = false;
        right = false;
    }

    if(key == KeyEvent.VK_R)
    {
        highscore = 0;
        counter=0;
        moves=0;
        score=0;
        lengthofsnake = 3;
        z = 0;
```

```
            try{
             FileOutputStream saveFile = new FileOutputStream("saveFile.sav");
             ObjectOutputStream save = new ObjectOutputStream(saveFile);
             save.writeObject(z);
             save.close();
            }
            catch(Exception exc){
             exc.printStackTrace();
            }

            right = true;
            left = false;
            up = false;
            down = false;

            repaint();
          }
      }
  }
```

OOP Concept Implementation

Encapsulation

This is the practice of keeping fields within a class private, then providing access to them via public methods. It's a protective barrier that keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.

Encapsulation lets us re-use functionality without jeopardizing security. It's a powerful OOP concept in Java because it helps us save a lot of time. For example, we may create a piece of code that calls specific data from a database. It may be useful to reuse that code with other databases or processes. Encapsulation lets us do that while keeping our original data private. It also lets us alter our original code without breaking it for others who have adopted it in the meantime.

In the example below, taken from the project code, encapsulation is demonstrated as an OOP concept in Java. Here, the variable "snakexlength", "snakeylength" and "moves" is kept private or "encapsulated", which is being called again from another funtion to perform methods.

```
public class Gameplay extends JPanel implements KeyListener, ActionListener{
   private int[] snakexlength = new int[750];
   private int[] snakeylength = new int[750];
    private int moves = 0;
   ...
}
   public void paint(Graphics g){
      if(moves == 0){
      snakexlength[2] = 50;
      snakexlength[1] = 75;
      snakexlength[0] = 100;
      snakeylength[2] = 100;
      snakeylength[1] = 100;
      snakeylength[0] = 100;
   }
}
```

<u>Interface</u>

An interface is a blueprint of a class, which can be declared by using interface keyword. Interfaces can contain only constants and abstract methods. Like abstract classes, Interfaces cannot be instantiated, they can only be implemented by classes or extended by other interfaces. Interface is a common way to achieve full abstraction in Java. We use implements keyword while implementing an interface (similar to extending a class with extends keyword).

```
public class Gameplay extends JPanel implements KeyListener,ActionListener{
    public Gameplay(){
        addKeyListener(this);
        setFocusable(true);
        ...
    }
}
```

In the above example, a class called Gameplay that extends JPanel and implements KeyListener and ActionListener was written.

`addKeyListener` is a method in the Component class, inherited through many levels of the class hierarchy. It accepts data type KeyListener as an arguement. In order to become a KeyListener, a class must implement the KeyListener interface and override its methods. This ensures that any class that will use a KeyListener to catch keyboard events includes methods that detail how the application will respond to them.

<u>Inheritence</u>

This feature lets programmers create new classes that share some of the attributes of existing classes. This lets programmers to rebuild on previous work without reinventing the wheel.

It works by letting a new class adopt the properties of another. Its called the inheriting class a subclass or a child class. The original class is often called the parent. The keyword extends is used to define a new class that inherits properties from an old class.
In the below example, inheritence has been simply achieved by using the **extends** keyword:

```
public class Gameplay extends JPanel implements KeyListener,ActionListener{
}
```

Based on the code fragment above, a new class called “Gameplay” has been created while inheriting the properties of “JPanel”.

Abstraction

Abstraction means using simple things to represent complexity. In Java, abstraction means simple things like objects, classes, and variables represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple times. It’s used to create a boundary between the application and the client programs. Example code from main.java;

```
    public static void main(String [] args) {
      JFrame obj = new JFrame();
      Gameplay gameplay = new Gameplay();
      obj.setBounds(10, 10, 905, 700);
      obj.setBackground(Color.DARK_GRAY);
      obj.add(gameplay);
    }
```

Based on this simple example , Gameplay.java is the base interface, or Abstract class. It contains parameters and variables that is hidden from the client program. The client only knows that the gameplay class has been added into the JFrame obj that allows the game to function normally.

Object and Classes

Everything in Java is associated with classes and objects, along with its attributes and methods. From a programming point of view, an object can include a data structure, a variable, or a function. It has a memory location allocated. The object is designed as class hierarchies. A Class is like an object constructor, or a "blueprint" for creating objects. It is a basic concept of Object-Oriented Programming which revolve around the real-life entities. Class in Java determines how an object will behave and what the object will contain.

```
public class Gameplay extends JPanel {
    private int lengthofsnake = 3;
    private int delay = 100;
    private int moves = 0;
    private int counter = 0;
    ...
}
public static void main(String [] args) {
      JFrame obj = new JFrame();
      Gameplay play = new Gameplay();
      obj.add(play);
}
```

In the example above, the concept of objects and classes were implemented in the code that makes up the program. A class named “`Gameplay`” was created. It was used to create an object called “`play`” by using the keyword `new`. The object was added into the boudaries of another object from the JFrame class.

Read and Write Implementation

This program provides an autosave feature of the highscore everytime the user runs the game. The program will save the value of the score to seperate external file provided if its higher than the previous highscore. This is done by using the class FileInputStream and FileOutputStream.

```
public Gameplay()
   {
      try{
         FileInputStream saveFile = new FileInputStream("saveFile.sav");
         ObjectInputStream save = new ObjectInputStream(saveFile);
         highscore = (Integer) save.readObject();
         save.close();
      }
      catch(Exception exc){
         exc.printStackTrace();//if theres an error, print it out.
      }
   }
```

As the game starts, the program will check the existence of the file named “saveFile.sav”, if it doesnt exist it will create a new file with said name. The program will then proceeds to read the data from the file and stores it into a private variable “highscore”. This feature is to mainly preserve the highscore even after the program is closed.

```
public void keyPressed(KeyEvent e) {
   int key = e.getKeyCode();
   if(key == KeyEvent.VK_SPACE)
      {
      if(score > highscore){
          try{
            FileOutputStreamsaveFile = new FileOutputStream ("saveFile.sav");
            ObjectOutputStream save = new ObjectOutputStream(saveFile);
            save.writeObject(score);
            save.close();
          }
          catch(Exception exc){
             exc.printStackTrace();
          }
        }

        try{
          FileInputStream saveFile = new FileInputStream("saveFile.sav");
          ObjectInputStream save = new ObjectInputStream(saveFile);
          highscore = (Integer) save.readObject();
          save.close();
        }
       catch(Exception exc){
          exc.printStackTrace();
        }
```

If the player restarted the game by pressing the spacebar key, the program will check through an if/else conditional statement for the value of score to be higher than the value of highscore. If its true, it will then save the data value of score into the file. It will then reopen the file to store its value into the declared variable highscore. The value is shown at the top right corner of the game.

```
if(key == KeyEvent.VK_R)
{
        highscore = 0;
        z = 0;
        try{
           FileOutputStream saveFile = new FileOutputStream("saveFile.sav");
           ObjectOutputStream save = new ObjectOutputStream(saveFile);
           save.writeObject(z);
           save.close();
        }
        catch(Exception exc){
           exc.printStackTrace();
        }
}
```

Players are able to manipulate the highscore data by pressing the R key to reset its value back to zero. A very useful and simple concept.

ULAR2020 Game Manual

Instructions

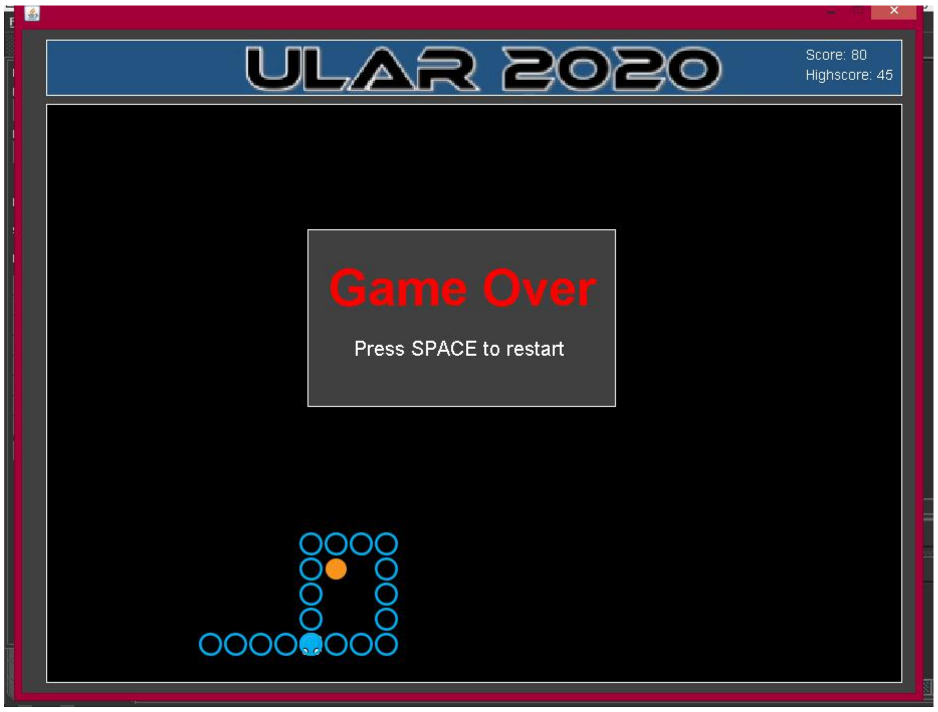
This application is a replication of the classic game 'Snake'. ULAR2020 takes place within an environment of 23x33 block area. The user controls a snake which continuously moves in the direction of the head is facing. If the snake runs into its' own body, it will result in fatal injury to the snake and the player will lose the game as a result. It can however portal through the the border to the opposite side. Snake food will always be randomly spawned on the map. The snake will then 'eat' the spawned food by running over them. When that happens, the snake will grow 1 unit longer and another block of snake food will spawn in another randomly generated location.

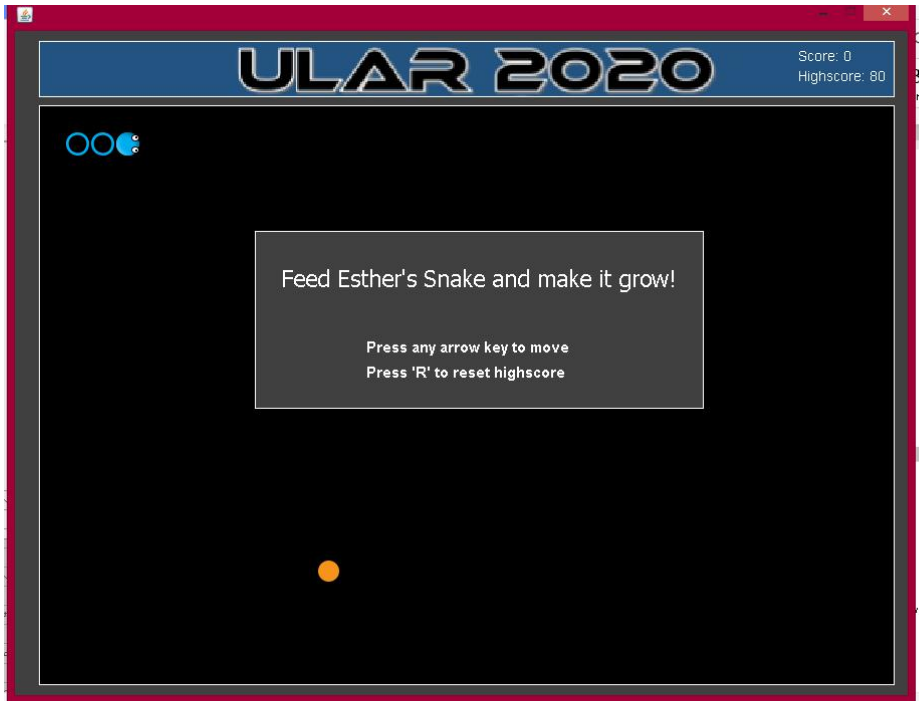
The goal of the game is to control the snake in such a way that it eats as many food as possible without dying. The game gets harder as the snake gets longer, because there will be less available space for the snake to move in its' environment, so the game gets very strategic as the snake approaches longer lengths.

Control Keys

◆ SPACE BAR

Pressing the space bar restarts the game. You can restart at any point during the game.





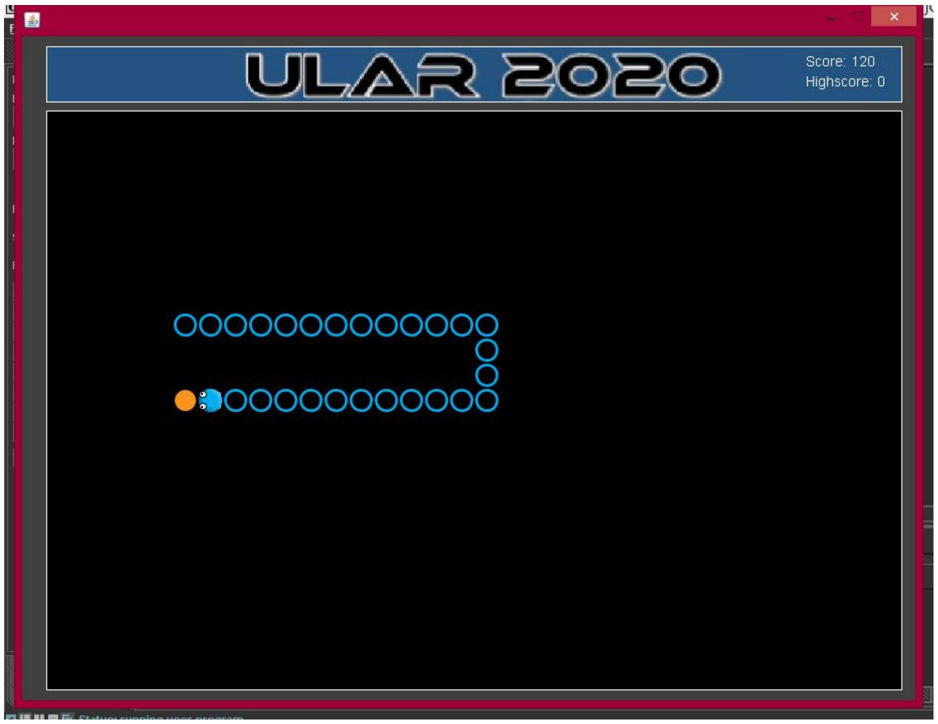
Feed Esther's Snake and make it grow!

Press any arrow key to move
Press 'R' to reset highscore

The game will then restart and the scores will be set to 0. The game will show the previous highscore if it has been beaten

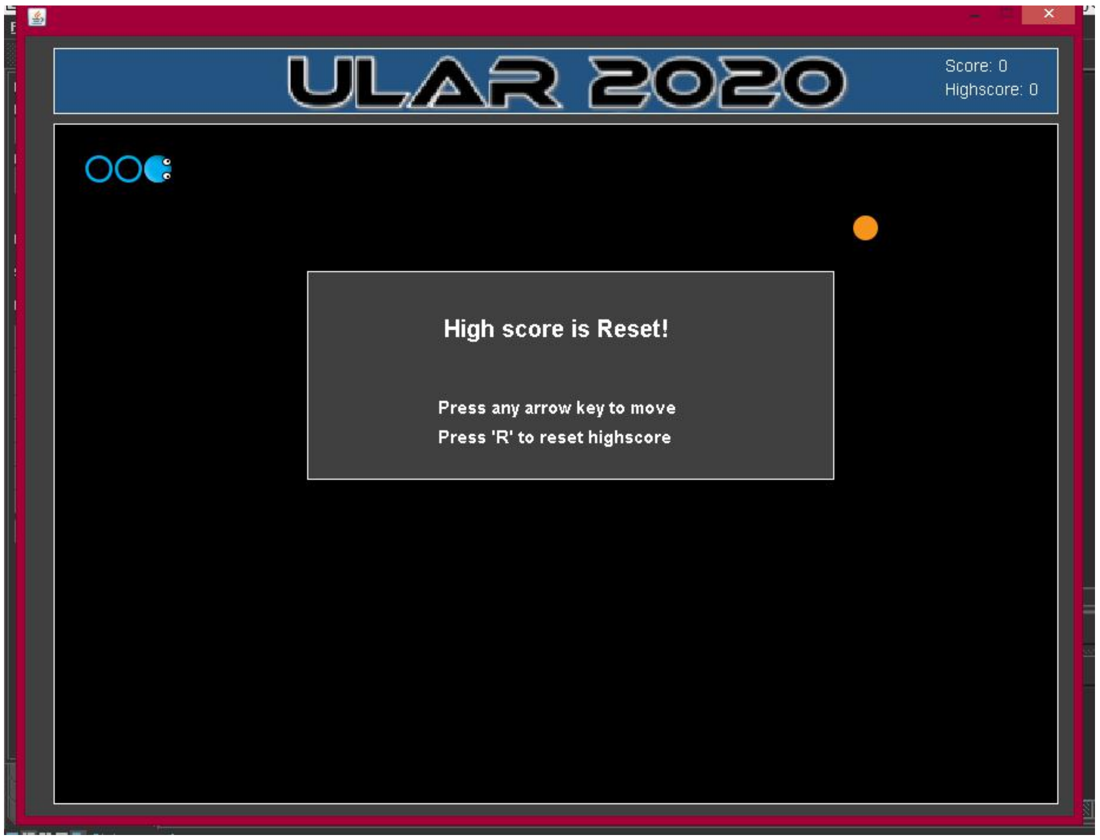
- ARROW KEYS

  Pressing the arrow keys will make the snake face and move to the respective direction of the arrow key. Note: The snake cannot instantly turn backwards, it must go around in a semi circle.



- ‵ R’ KEY

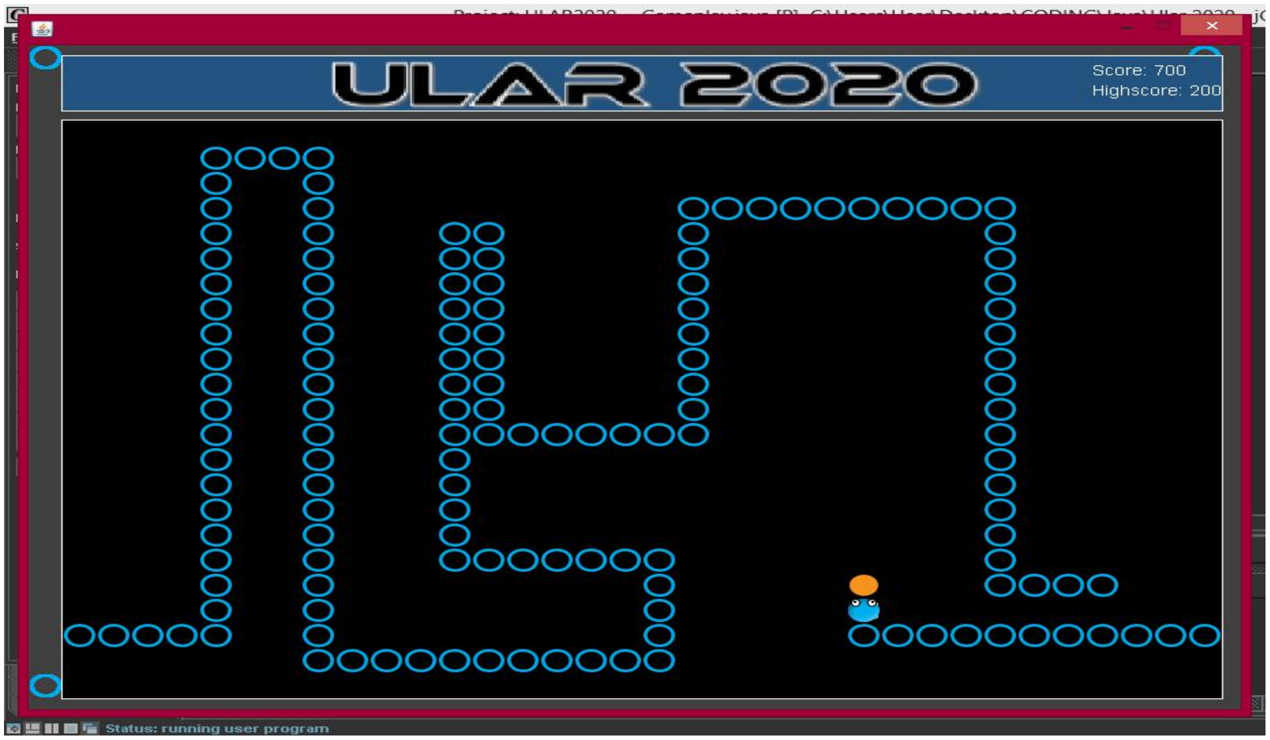
  Pressing it resets the game and highscore back to zero. This allows the user to manipulate the data stored in the hardware.

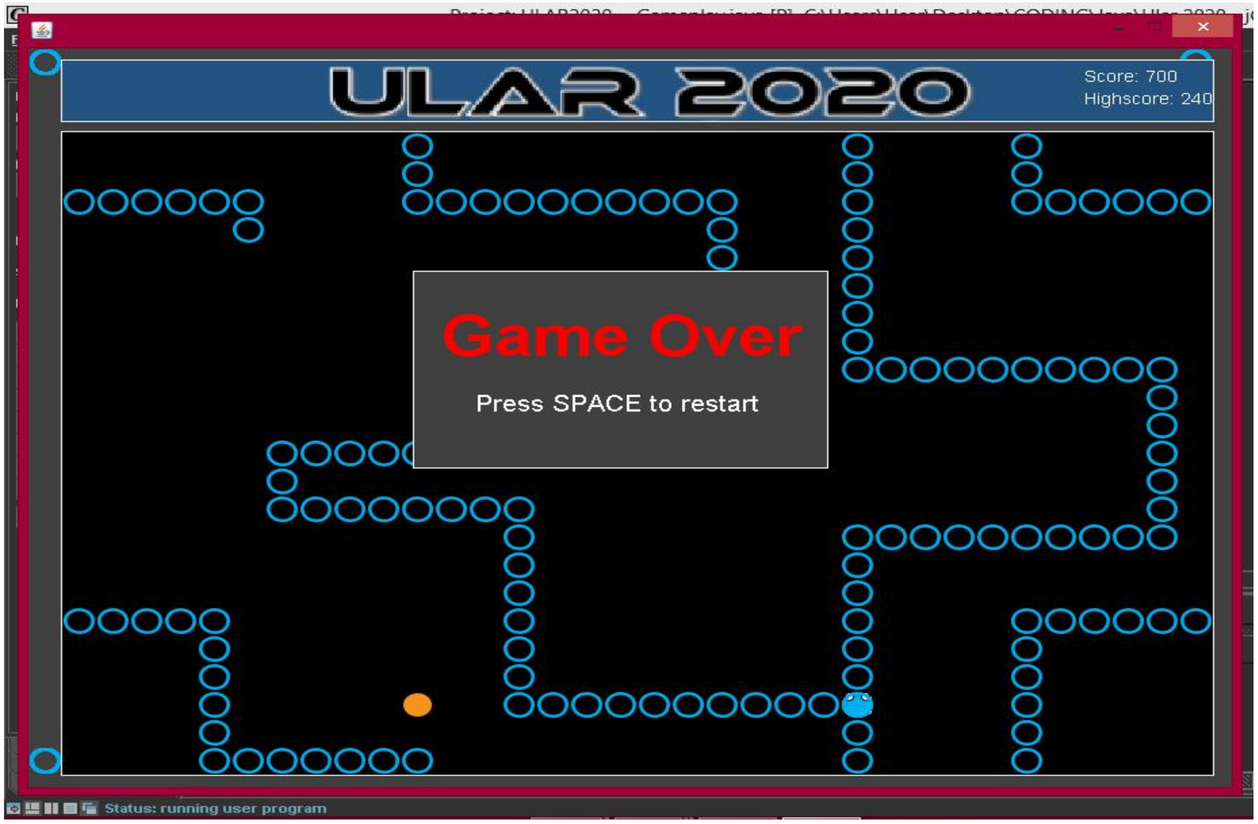


High score is Reset!

Press any arrow key to move
Press 'R' to reset highscore

The program GUI





Conclusion

Object oriented programming has become a fundamental part of software development. Through object oriented programming language, the software maintenance is easier than as compared with structured oriented programming as much time has been invested while doing the planning of the code so minimum flaws are found.

As well as the project designed, OOP helps the system works so much better. Besides that, the project which was presented in graphical user interface looks very attractive and it is much better than command driven interface which has many drawbacks. It is because the simple icon in GUI uses multiple instructions in the back end. Hence, it easy for the programmer to code in GUI languages.