End Semester Project Report

School: SEECS

Class: BSCS-11C

Course: OOP

Instructor: Dr.Imran Mehmood

Group: D

Syed Hamza Ali Shah (396358)

Email: shaahah.bscs21seecs@seecs.edu.pk

Mohammad Irfan (373668)

Email: mirfan.bscs21seecs@seecs.edu.pk

Ali Murad (372401)

Email: abajeer.bscs21seecs@seecs.edu.pk

Hamaz Hamza (365670)

Email: hmalik.bscs21seecs@seecs.edu.pk

Omer Farooq (377187)

Email: ofarooq.bscs21seecs@seecs.edu.pk

Table of Contents:

Brief Introduction: 3
Architecture: 3
Presentation Layer: GUI 4
Logic Layer: Simulation 7
Data Layer: Data update and storage 8
Conclusion: 9

Brief Introduction:

Our end semester project is a 2D Gravity Simulator. It is based on the law of Gravity:

$$F = \frac{Gm_1m_2}{r^2}$$

By using this formula, a set of planets can be created that mimic the solar system with planets revolving around a central planet or a star. The simulation is shown on the screen in which new planets can be added. As the simulation runs, the details of the simulation are added to a database at the backend. (Each part is explained in detail later in the report).

Instead of procedural programming we have utilized OOP(Object Orientated Programming) along with its SOLID Principles:

Single Responsibility: All the classes and methods are designed to handle a single task instead of performing multiple functions making them highly specific.

Open to extension Closed for modification: By utilizing this principle new features can be added to our project without causing disturbance to the rest of the code.

Liskov Substitution Principle: The classes in the project can be replaced with their subclasses or child classes without changing the overall behavior of the program.

Interface Segregation Principle: Our project does not include any interfaces but the classes themselves are still specific in their functionality.

Architecture:

The Gravity Simulator uses a multi-layer architecture instead of a multi-layer architecture. This is because it can easily run on a normal desktop or a laptop without the need of high performance and expensive hardware. Due to this reason, it is divided into layers which all run on a single machine instead of tiers which run on separate hardware. The project is divided into following layers:

Presentation Layer: GUI

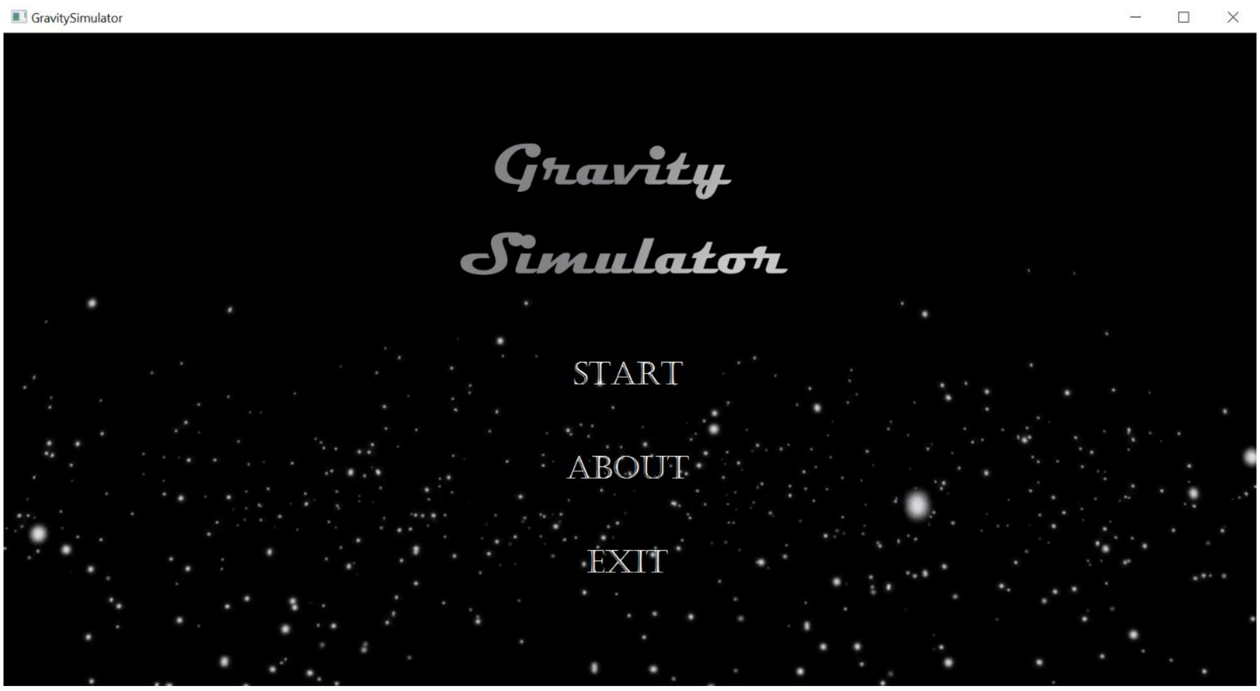
Logic Layer: Simulation

Data Layer: Data update and storage

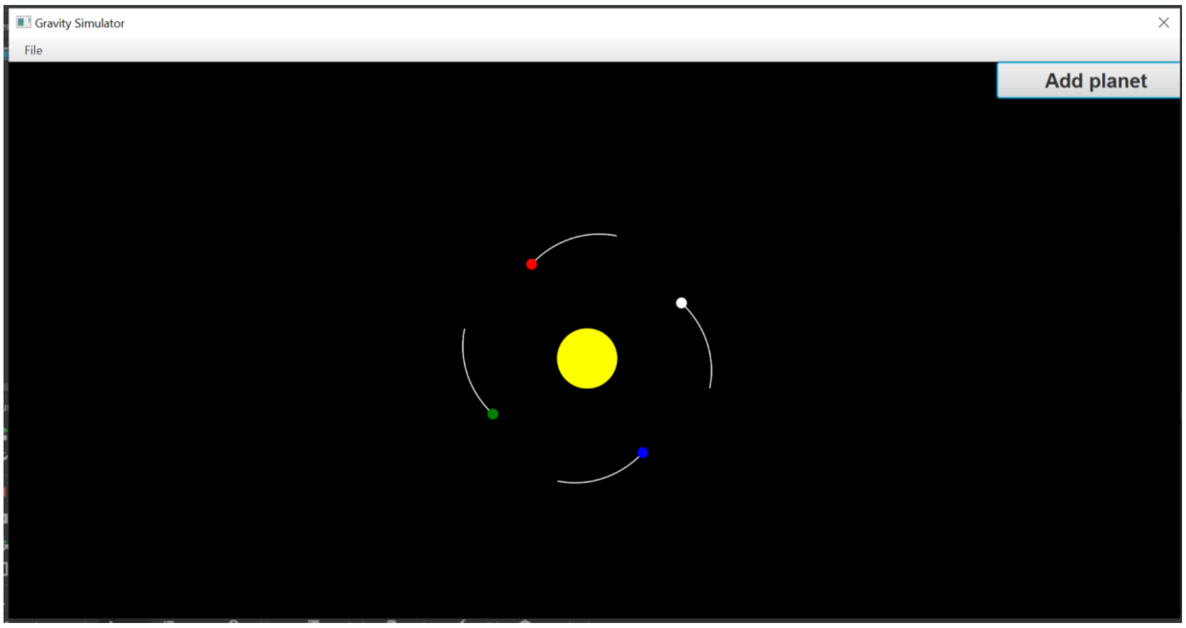
Presentation Layer: GUI

The presentation layer is mainly the graphical user interface that allows the user to add new planets to the simulation. Following are the components of the GUI:

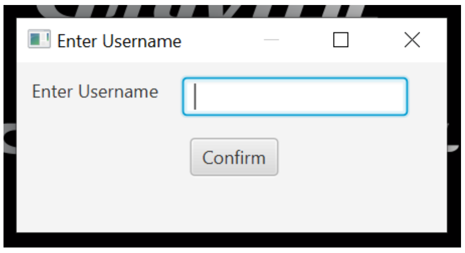
Title Screen:



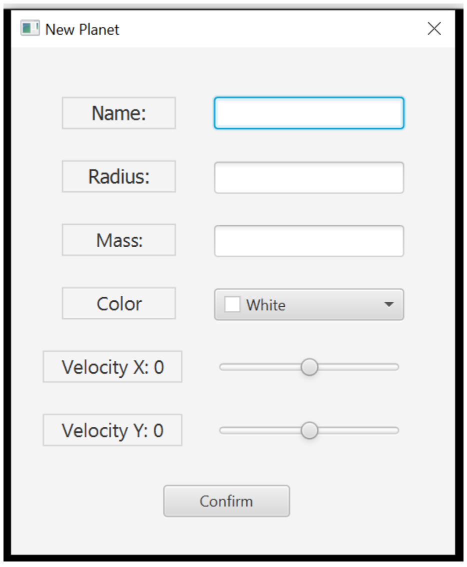
Main Simulation Window:



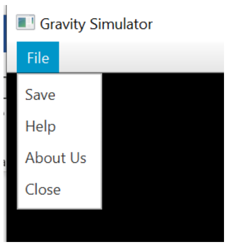
Username popup:



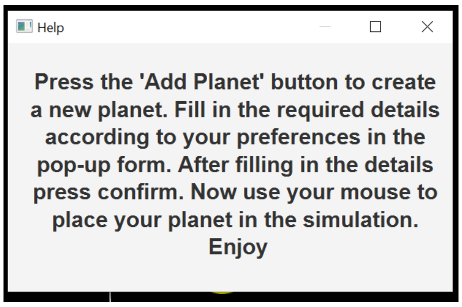
New Planet Form:



Menu Bar:

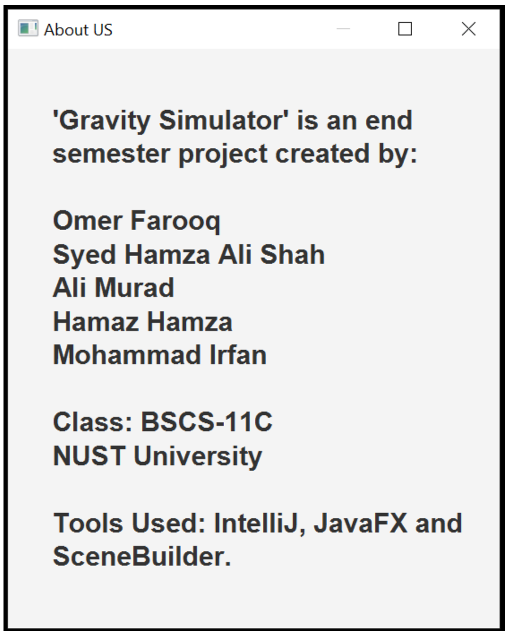


Help Window:



Press the 'Add Planet' button to create a new planet. Fill in the required details according to your preferences in the pop-up form. After filling in the details press confirm. Now use your mouse to place your planet in the simulation. Enjoy

About us Window:



'Gravity Simulator' is an end semester project created by:

Omer Farooq
Syed Hamza Ali Shah
Ali Murad
Hamaz Hamza
Mohammad Irfan

Class: BSCS-11C
NUST University

Tools Used: IntelliJ, JavaFX and SceneBuilder.

Explanation:

The gui for this project has been made using javafx. It has three main windows: Title Screen, Main window and a new planet form window. The title screen will open when the gravity simulator is opened. Once you press start the main window appears. It has a black box which will be the place for the simulation to run and a “add new planet” button for adding new planets. When the button is pressed a popup window will show taking multiple inputs: name, radius, mass, color and x and y component of initial velocity for the planet that will be added to the simulation. When the confirm button is pressed, a message hovers over the mouse cursor: “click to add”. Wherever the user clicks on the simulation, at that location the new planet is placed. The menu bar gives the option to save the data of the simulation, a help button which shows the instructions of how to use the gravity simulator, a About us button that show the details of the students who designed it and a close button to close the simulation.

The gui has been made by javafx and javafx scene builder. The scene builder allows us to design the gui by drag and drop and also configure the location and styling. On the back end it generates a .fxml file that has a css format code. The java code combines all the windows and fxml files as a one gui. The action listeners are written in java that are linked with the fxml files and allow buttons to perform action when clicked.

Logic Layer: Simulation

The basic logic behind this simulation revolves around the use of physics equations, vectors and a basic understanding of how bodies in space attract one another.

Firstly, we decided to make the simulation in 2-dimensions, hence we needed two variables: X and Y to represent the positions, accelerations, and velocities of all planets, and also the forces acting on the planets.

The X and Y variables follow traditional behavior i.e. X represents the horizontal component and Y represents the vertical component. To make things easier, we combined these two variables into a new data structure called a "Vector" instead of dealing with them individually. For positions, vectors can be used as coordinates of the object relative to the origin point (in our case it's the center of the screen). For velocities, the X and Y values stored within the vectors represent the horizontal and vertical components of the velocity/acceleration/force of/on the planets.

Secondly, since simulations are models of the real world, we needed to implement real world physics into our simulation. We achieved this through the help of conventional physics equations for gravitational pull, newton's laws of motion and projectile motion.

For gravitation pull, we used the equation Force = G.M.m/r2 where F is the force, G is the gravitational constant, M represents the mass of the planet on which the force is acting and m is the mass of the planet applying the force, and r represents the distance between the two planets. To find an appropriate value of G, we used trial and error and picked the value which allowed the simulation to mimic real-world planetary orbits the best. After experimentation we found that setting G equal to 100 works best. For newton's laws of motion, we used the conventional Force = mass*acceleration equation, because all real-world objects follow this law. This equation allowed us to find the acceleration acting on each planet due to gravitational pulls from all other planets. Finally, projectile motion equations define how acceleration changes the velocity of the object and how the velocity changes the position of the object. The projectile motion equations required a value of 'time elapsed' in them. The smaller the value of time elapsed, the more accurate the values of velocity and position will be, because calculations will be done more times per frame. For our simulation, we set 'time elapsed' to a default value of 0.01.

All these equations combined allow us to define the path an object will follow under gravitation influence from other bodies. For the paths, we just drew a line between the initial and final positions of the planets at each frame for a duration of 120 frames, so the paths were neither too short nor too long. Due to the planets changing positions many times per second, the lines were very small and had slight changes in direction, giving an effect of curvature for the paths followed by the planets.

Furthermore, a collision mechanic was also implemented, the rules of which are as follows:

If a planet of less mass collides with a planet of larger mass, the planet of less mass gets destroyed and is removed from the simulation.

Upon collision the planet with the larger mass, gains the mass of the planet of the lesser mass.

If the colliding planets are of the same mass, both planets are destroyed from the simulation.

Lastly, we coded a default planet arrangement that would run whenever a new simulation began, but we also allowed the user to add their own custom planets, with their own names, radii, masses, colors, velocities and at their specified location so that they could test different scenarios on the simulation for recreational or educational purposes.

Data Layer: Data update and storage

Our project, the Gravity Simulator, utilizes the MySQL Database by storing the details of the simulation. The database of our project is called "simulator_db" which contains two main tables. Each simulation uses these two tables: One table is for general info(named "Simulation_Info") that stores the user's name, No. of planets created and total duration of the simulation. And the other table(named " Simulation_Log_'number' ") stores specific details of the simulation such as at what instance of time which planet was created along with all of its details. It is equivalent to a log file. For each simulation the Log table ("Simulation_Log_'number' ") is separate but the general info table("Simulation_Info") is the same. For example, the first simulation will have "Simulation_Log_1" and the second simulation will have "Simulation_Log_2" but both will have the "Simulation_Info" table. We decided to design it in this way because we wanted to keep the details of every simulation separate from the details of the other simulations, hence creating a different one for each.

Our project has a DataBase.java file that has the methods to attain the above-mentioned functionality:

DB_Info_Insert:

This method is attached to the close button. Whenever the close button is pressed the general details of the current simulation are inserted into the "Simulation_Info" table.

DB_Detail_Insert:

This method is attached to the Confirm button of the New Planet Form. Whenever the Confirm button is pressed, and the details of the planets are inserted to the Log table (" Simulation_Log_'number' ").

DB_Create_Table:

This method is also attached to the close button. Whenever the close button is pressed a new log table is created that will be used by the next simulation to insert its timed details. In this way the details of every simulation will be stored separately. The details include the Planet name, radius, mass, color, Initial Coordinates and Initial Velocity.

The above-mentioned methods are static and are directly used in the simulation whereas the are two other methods:

getLastId:

This method counts the number of rows in the info table and is used by the DB_Info_Insert method to know in which row the data goes into for the current simulation.

DB_getTotalTables:

This method counts the number of tables in the database and is used by the DB_Create_Table method to know the number of the next log table: "Simialtion_Log_number".

These two methods are not directly used in the simulation but are used by the data layer itself to know where the data will go for the running simulation and the next simulation.

Conclusion:

The Gravity Simulator can be used by kids from age 7 – 15 as an educational tool to understand the law of gravity and how two large bodies possessing mass attract each other. It is a great educational application especially in the sense that kids get hands on experience and visual representation of planetary system. Also it may me an encouragement and inspiration for older kids to learn programming and make their own applications in which they express their imagination in their own way.

For us it was the direct application of OOP and JAVA along with GUI and Database Management. It was a great learning experience and also made us realize how much we still need to learn.