VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI, KARNATAKA-590 018.

A MINI PROJECT REPORT

ON

TOWER OF HANOI SIMULATION

Submitted in partial fulfilment of the requirements for the ***Mini Project (18CSL67)*** *course of the 6th semester.*

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

By

K M ISHA	**1JS20CS075**
KARAN SAURAV	**1JS20CS076**

Under the guidance of

Dr. Pavithra G S

Asst. Professor, Dept. of CSE

Department of Computer Science and Engineering

JSS ACADEMY OF TECHNICAL EDUCATION, BENGALURU

2022 – 2023

JSS MAHAVIDYAPEETHA, MYSURU

JSS Academy of Technical Education

JSS Campus, Uttarahalli Kengeri Main Road, Bengaluru – 560060

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the mini project work entitled **“TOWER OF HANOI SIMULATION”** is a benefited work carried out by **K M ISHA** bearing USN **1JS20CS075, KARAN SAURAV** bearing USN **1JS20CS076** bonafide student of **JSS Academy of Technical Education** in the partial fulfillment for the award of the **Bachelor of Engineering** in **Computer Science & Engineering** of the **Visvesvaraya Technological University**, Belgaum, during the year 2022-23. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of mini-Project work prescribed for the said degree.

Dr. Pavithra G S	**Dr. P B Mallikarjuna**
Asst. Professor, Dept. of CSE	Assoc. Prof & HOD, Dept. of CSE

Name of the Examiners	**Signature with date**
1.	....
2.	....

ACKNOWLEDGEMENT

I, take this opportunity to thank one and all involved in helping me build this project. Firstly, I would like to thank the college for providing me an opportunity to work on this project.

I thank the management of the **JSS Academy of Technical Education** for providing all the resources required for the project.

I wish to acknowledge my sincere gratitude to our **Principal, Dr. Bhimasen Soragaon** for his constant encouragement and for providing us with all the facilities required for the accomplishment of this project.

The project would not have been possible if not for the constant support of our Associate Professor and Head of the Computer Science Department, **Dr. Mallikarjuna P B.**

I also am highly grateful to the guidance offered by **Dr. Pavithra G S, Asst. Professor, Department of Computer Science,** who has been very generous in assisting and supporting, to do this project named **"Tower of Hanoi Simulation"**, which formally started as just a rough idea and now has resulted in the form of this project.

I also would like to thank all the other teaching and non-teaching staff members who had extended their hand for support and co-operation while bringing up this project.

K M ISHA (1JS20CS075)
KARAN SAURAV (1JS20CS076)

ABSTRACT

This mini project aims to develop a captivating and immersive Tower of Hanoi Simulation with C++ programming language and OpenGL graphics library. The project leverages fundamental concepts of computer graphics and visualization to create an engaging gaming experience.

The Tower of Hanoi is a classic mathematical puzzle that challenges the problem-solving skills of individuals. This project presents an interactive simulation of the Tower of Hanoi using the OpenGL graphics library. The aim is to create a visually appealing and engaging environment that allows users to understand and interact with the Tower of Hanoi problem.

The tower is represented by a series of stacked disks of varying sizes, which need to be moved from one tower to another while following the puzzle's rules. The user can control the movement of the disks through intuitive mouse or keyboard inputs.

Furthermore, the project aims to serve as an educational tool for individuals interested in learning about algorithmic problem-solving and recursion. By visually demonstrating the Tower of Hanoi puzzle, the simulation enables users to observe and internalize the underlying principles and strategies required to solve the problem efficiently. Key Features include choosing the no of disks varying from 1 to 9, changing the viewing angles and rotating the view and altering the speed as per the requirements of the user

Overall, this Tower of Hanoi simulation project combines the power of OpenGL graphics with interactive elements to create an engaging and educational experience. Through its visual representation, interactivity, and educational value, the simulation offers users a unique opportunity to explore and master the Tower of Hanoi puzzle in an interactive and enjoyable manner.

CONTENTS

LIST OF FIGURES

Chapter 1: Introduction:

The Tower of Hanoi Simulation is a computer graphics and visualization project that utilizes the power of C++ programming language and the OpenGL graphics library to create an immersive experience. In this section, we will provide an overview of the project and its objectives.

The main goal of the project is to develop a captivating experience where the user witnesses the working of the Tower of Hanoi problem. The simulation environment will be rendered using OpenGL, offering visually appealing graphics and realistic movements.

1.1 About OpenGL:

OpenGL (Open Graphics Library) is a cross-platform graphics API (Application Programming Interface) that allows developers to create interactive and high-performance 2D and 3D graphics applications. It provides a set of functions for rendering and manipulating graphical objects, enabling developers to harness the full potential of modern graphics hardware.

1.2.1 OpenGL Commands and Primitives:

OpenGL provides a wide range of commands and primitives that developers can use to create and manipulate graphical objects. Primitives are basic geometric shapes such as points, lines, and polygons. These primitives can be combined and transformed to create complex objects in the game environment.

OpenGL commands allow developers to control the rendering process, specifying various attributes such as color, texture, and lighting effects. For example, developers can use commands to set the background color, define the position and orientation of objects, and apply textures to surfaces.

1.2.2 OpenGL Rendering Pipeline:

The OpenGL rendering pipeline is a series of stages that transform the input data (vertices, textures, etc.) into the final image displayed on the screen. Understanding the rendering pipeline is crucial for efficientgraphics programming. The pipeline consists of several stages, including vertex processing, primitive assembly, rasterization, fragment processing, and framebuffer operations. Each stage performs specific tasks such as transforming vertices, interpolating colors, applying textures, and performing depth testing.By properly understanding and utilizing the rendering pipeline, developers can optimize their code and achieve better performance in rendering complex scenes.

1.2.3 OpenGL - GLUT and OpenGL Utility Libraries:

GLUT (OpenGL Utility Toolkit) and OpenGL Utility Libraries are additional tools and libraries that provide extended functionality and convenience for OpenGL programming. GLUT simplifies theprocess of creating windows, handling user input, and managing events such as mouse clicks and keyboard inputs. It provides a platform-independent interface for interacting with the underlyingoperating system, allowing developers to focus on graphics programming without worrying about platform-specific details.

- In the Tower of Hanoi Simulation project, we will leverage the power of GLUT and OpenGL Utility Libraries to streamline the development process and create a user-friendly interface.
- OpenGL Utility Libraries, such as GLU (OpenGL Utility Library) and GLM (OpenGL Mathematics), offer additional utilities and functions for common tasks like loading 3D models, handling matrix transformations, and performing mathematical operations. These libraries enhance the development process and facilitate efficient and robust code implementation.
- By utilizing OpenGL, along with its commands, rendering pipeline, and utility libraries, the UFO Alien Game project aims to provide an immersive gaming experience with visually appealing graphics, smooth animations, and realistic physics-based movements. This project offers an excellent opportunity for students to explore the fundamentals of computer graphics and visualization while developing their programming skills in C++ and OpenGL.

1.3 Advantages of Using OpenGL:

OpenGL offers several advantages that make it a popular choice for computer graphics programming:

- Cross-Platform Compatibility: One of the significant advantages of OpenGL is its cross-platform compatibility. It is supported on various operating systems, including Windows, macOS, and Linux, allowing developers to create applications that can run seamlessly across differentplatforms without extensive modifications.

- Hardware Acceleration: OpenGL takes advantage of the capabilities of modern graphics hardware to accelerate the rendering process. By offloading computations to the GPU (Graphics Processing Unit), OpenGL can achieve faster rendering and deliver high-performance graphics,making it ideal for real-time applications like games.
- Wide Industry Adoption: OpenGL has been widely adopted in various industries, including gaming, virtual reality, scientific visualization, and computer-aided design. Its extensive use in the industry ensures a wealth of resources, tutorials, and community support, making it easier for developers to find help and learn from others' experiences.

- Flexibility and Customization: OpenGL provides developers with a high degree of flexibility and customization options. It allows fine-grained control over the rendering pipeline, enabling developers to optimize their code for specific requirements. Additionally, OpenGL supports extensions that allow developers to access advanced graphics features and tailor their applicationsto specific hardware capabilities.

- Integration with Other Libraries: OpenGL can be easily integrated with other libraries and frameworks to enhance functionality. For example, developers can combine OpenGL with libraries like OpenAL for audio, OpenCL for parallel computing, or ImGui for creating graphical user interfaces. This flexibility allows developers to leverage the strengths of different libraries and create powerful applications.

1.4 C++ and OpenGL:

C++ is a versatile and widely used programming language known for its performance, efficiency, and extensive support for object-oriented programming. When combined with OpenGL, a powerful graphics library, it becomes a formidable tool for computer graphics programming. In this section, we will explorethe benefits and capabilities of using C++ in conjunction with OpenGL.

C++ supports object-oriented programming paradigms, enabling developers to design modular and reusable code structures. This is advantageous in graphics programming, as it allows for the creation of classes and objects that encapsulate graphics entities, such as meshes, textures, and shaders. Object- oriented design promotes code organization, readability, and maintainability, making it easier to developand maintain large-scale graphics applications.

1.4.1 Performance and Efficiency:

C++ is renowned for its ability to deliver high-performance code. Its low-level control and direct memorymanagement allow developers to optimize their programs for efficiency. This is particularly important in graphics programming, where real-time rendering and complex computations are required. With C++,developers have fine-grained control over memory allocation, data structures, and algorithm implementations, resulting in faster and more efficient code execution.

1.4.2 Object-Oriented Programming:

C++ supports object-oriented programming paradigms, enabling developers to design modular and reusable code structures. This is advantageous in graphics programming, as it allows for the creation of classes and objects that encapsulate graphics entities, such as meshes, textures, and shaders. Object- oriented design promotes code organization, readability, and maintainability, making it easier to developand maintain large-scale graphics applications.

1.4.3 Compatibility and Portability:

C++ is a widely supported language, with compilers available for various platforms and operating systems. This ensures that C++ code written for graphics programming using OpenGL can be easily ported to different systems without significant modifications. This cross-platform compatibility enablesdevelopers to reach a broader audience and deploy their applications on multiple platforms, such as Windows, macOS, and Linux.

1.4.4. Leveraging the Features of OpenGL:

- **Graphics Rendering:** OpenGL provides a comprehensive set of functions and features for rendering 2D and 3D graphics. It supports a variety of rendering primitives, including points, lines, and polygons, which can be transformed and textured to create complex objects. OpenGL also offers advanced rendering techniques such as vertex and fragment shaders, which allow developers to manipulate vertices and fragments at the GPU level, achieving stunning visual effects.

- **Platform-Independent API:** OpenGL is designed to be a platform-independent graphics API. It provides a consistent interface across different operating systems, allowing developers to writegraphics code that can be executed on multiple platforms without modification. By utilizing OpenGL in C++, developers can write graphics code that is portable and can seamlessly run on various systems, ensuring broad compatibility and reach.

- **Extensibility and Compatibility:** OpenGL is an extensible API that supports the use of extensions. These extensions provide additional features and functionalities beyond the core OpenGL specification. This extensibility allows developers to leverage the latest graphics capabilities and hardware advancements. Furthermore, OpenGL maintains backward compatibility, ensuring that applications written using older versions of the API can still run on newer systems with minimal adjustments.

- **Libraries and Frameworks:** C++ and OpenGL benefit from a vast array of libraries and frameworks that facilitate graphics programming. These include GLM (OpenGL Mathematics) for vector and matrix operations, GLFW (Graphics Library Framework) for window creation anduser input handling, and Assimp for loading and processing 3D model files. These libraries enhance development productivity by providing pre-built functionality and simplifying commontasks.

- **IDEs and Debugging Tools:** C++ is supported by numerous Integrated Development Environments (IDEs) that offer powerful features for code editing, debugging, and profiling. Popular IDEs such as Visual Studio, CLion, and Code::Blocks provide comprehensive development environments tailored for C++ programming. Additionally, debugging tools like GDB (GNU Debugger) enable developers to identify and fix errors efficiently during thedevelopment process.

  C++ and OpenGL have thriving communities of developers, enthusiasts, and experts who actively contribute to online forums, discussion boards, and tutorial websites. These communities provide a wealth of knowledge, code samples, and troubleshooting advice, making it easier for developers to learn, solve problems, and stay up-to-date with the latest advancements in graphics programming.

In conclusion, combining C++ with OpenGL offers a powerful and flexible approach to graphics programming. C++ provides performance, efficiency, and compatibility, while OpenGL delivers a comprehensive set of graphics rendering capabilities. Together, they form a formidable duo that allows developers to create visually stunning and high-performance graphics applications across multiple platforms. By leveraging the features of

both C++ and OpenGL, developers can unlock the full potential of computer graphics and bring their visions to life.

1.5 Future of OpenGL:

While OpenGL has been a dominant graphics API for many years, its future is evolving with the emergence of new technologies and APIs. The Khronos Group, the organization responsible for OpenGL's development, ha`s introduced Vulkan as a next-generation graphics API designed to provide even greater performance and efficiency.

Vulkan, also known as OpenGL Next or OpenGL 4.6+, offers lower-level access to graphics hardware,allowing developers to have more control over the rendering process and achieve higher performance. Vulkan aims to address the limitations of OpenGL and provide a more modern and efficient graphics API.

However, despite the rise of Vulkan, OpenGL remains relevant and widely used, especially in legacy systems and applications. Many existing applications, including games, continue to rely on OpenGL, and support for the API is expected to continue for the foreseeable future. Furthermore, OpenGL continues to evolve, with periodic updates and extensions being introduced to add new features and improve functionality. Developers who are familiar with OpenGL can easily transition to Vulkan or other APIs as needed, leveraging their existing knowledge and experience.

In conclusion, while the future of graphics programming may be shifting towards newer APIs like Vulkan, OpenGL remains a powerful and widely adopted graphics library with cross-platform compatibility, hardware acceleration, and a flexible development environment. Its ease of use, extensive resources, and broad industry support make it an excellent choice for the UFO Alien Game project, providing a solid foundation for creating captivating and visually stunning graphics.

Chapter 2: Requirement Specifications

2.1 SOFTWARE SPECIFICATION

- **Operating System:** Windows 11 and Linux Mint
- **Libraries:** OpenGL Libraries
- **IDE:** Visual Studio Code

2.2 HARDWARE SPECIFICATION

- **Processor:** x86 compatible processor with 1.7 GHz Clock Speed
- **RAM:** 512 MB or greater
- **Hard Disk:** 20 GB or grater
- **Monitor:** VGA/SVGA
- **Keyboard:** 104 keys standard
- **Mouse:** 2/3 button. Optical/Mechanical.

2.3 USER CHARACTERISTICS

Every user:

- Should be comfortable with basic working of the computer
- Must have basic knowledge of English
- Must carry a login ID and password used for authentication

Chapter 3: Design Phase

Steps to follow to compile the program code:

- Initialize the necessary libraries and variables.
- Define constants for screen size, speed, directions, and other parameters.
- Create arrays to store the coordinates of various discs used in simulation.
- Implement a function to display raster text on the screen.
- Create an initialization function to set up the OpenGL environment.
- Implement a function to draw disks and the pins.
- Implement a function to perform the tower of hanoi algorithm.
- Implement a function to display the action.
- Implement a function to display the instructions.
- Implement functions to reset to default.
- Set up the main display function to handle different view pages and call appropriate functionsaccordingly.
- Implement keyboard and mouse input handling functions.
- Set up the main function to initialize the OpenGL environment, register callback functions, andenter the main loop.

3.1 Algorithm

- A key to solving this puzzle is to recognize that it can be solved by breaking the problem down into a collection of smaller problems and further breaking those problems down into even smaller problems until a solution is reached. For example:
  - Label the pegs as A, B and C
  - Let N be the total number of discs
  - Number the discs from 1 (smallest, topmost) to N (largest, bottommost) To move N discs from peg A to peg C:
- 1.Move N−1 discs from A to B. This leaves disc N alone on peg A
- 2.Move disc N from A to C
- 3.Move N−1 discs from B to C so they sit on disc N

The above is a recursive algorithm, to carry out steps 1 and 3, apply the same algorithm again for N−1. The entire procedure is a finite number of steps, since at some point the algorithm will be required for N = 1. This step, moving a single disc from peg A to peg C, is trivial. This approach can be given a rigorous mathematical formalism with the theory of dynamic programming and is often used as an example of recursion when teaching programming.

- **void hanoi(actions* queue, const int n, const char pin1, const char pin2, const char pin3):**

A function to perform the algorithm to determine each movement

- **void push(stack* pin, disk* item):**

  To push disks into the specified pin

- **disk* pop(stack* pin):**

  To remove disks from the specified pin

- **void drawDisk(GLUquadricObj** quadric, const GLfloat outer, const GLfloat inner):**

  To draw the individual pins

- **void Reshape(int width, int height):**

This function is called when window is resized

To reshape and resize the pins and the disks

- **void hanoiinit(void):**

A function to animate the movement of disks onto the pins

- **void reset():**

  To reset the view to default front view

  Move the disks to Pin A

- **void drawPin(GLUquadricObj** quadric, const GLfloat radius, const GLfloat height):**

A function to visualize the individual pins

- **void drawAllPins(GLUquadricObj** quadric, const GLfloat radius, const GLfloat height, const GLfloat gap):**

A function to visualize all the pins together as a single unit

- **init():**

Set the clear color to white.
Set the current matrix mode to `GL_PROJECTION`. - Load the identity matrix.
Set the orthographic projection using `gluOrtho2D`. - Set the matrix mode to `GL_MODELVIEW`.

- **main(int argc, char** argv)**

Initialize GLUT and create a window.

Set the display mode and window properties. - Register the callback functions.

Call `init` to set up the OpenGL environment. - Enter the main loop using `glutMainLoop

Flow Diagram:

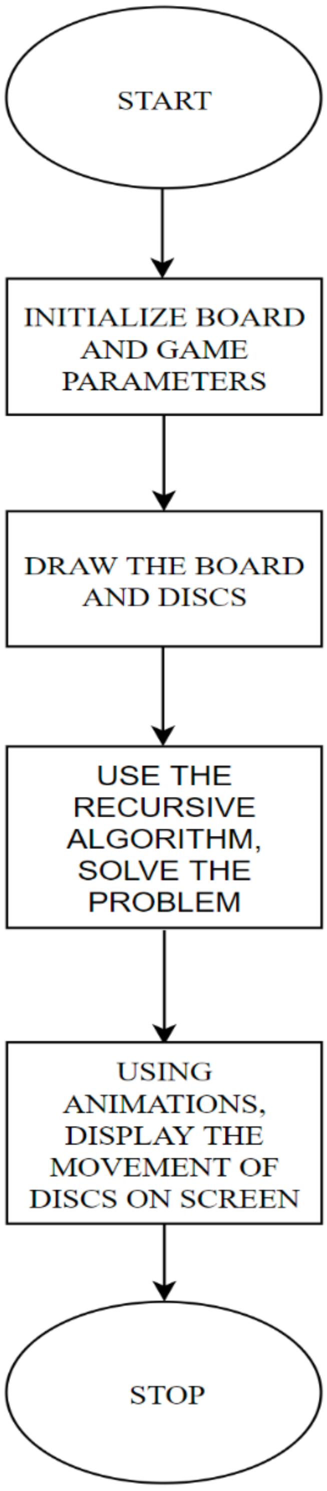

START
INITIALIZE BOARD AND GAME PARAMETERS
DRAW THE BOARD AND DISCS
USE THE RECURSIVE ALGORITHM, SOLVE THE PROBLEM
USING ANIMATIONS, DISPLAY THE MOVEMENT OF DISCS ON SCREEN
STOP

Figure 1 : Flow Chart of the program Code

Chapter 4: Implementation

4.1 Implementation of OpenGL built-in functions:

OpenGL provides a set of built-in functions that can be used to perform various graphics operations. These functions are implemented in the OpenGL library and can be accessed by including the appropriate headers and linking the OpenGL library with your code. Here are some examples of how these functionsare typically implemented:

- **glClearColor(float red, float green, float blue, float alpha):** This function sets the clear colorfor the color buffer. It determines the color that is used to clear the color buffer when `glClear(GL_COLOR_BUFFER_BIT)` is called. The implementation of this function wouldinvolve storing the specified color values internally and using them during the clearing process.

- **glClear(int mask)**: This function clears the specified buffers based on the provided mask. Forexample, `glClear(GL_COLOR_BUFFER_BIT)` clears the color buffer, `glClear(GL_DEPTH_BUFFER_BIT)` clears the depth buffer, and `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)` clears both buffers. The implementation of this function would involve accessing the specified buffers and setting their values to the clear color.

- **glBegin(int mode)**: This function marks the beginning of a group of vertices that define a geometric primitive. The `mode` parameter specifies the type of primitive being drawn, such as points, lines, triangles, etc. The implementation of this function would involve setting up internal state variables to track the current primitive mode and preparing the OpenGL pipeline for vertexdata.

- **glVertex2f(float x, float y)**: This function specifies a vertex with 2D coordinates. It is typicallycalled multiple times between `glBegin` and `glEnd` to define a set of vertices that form a primitive. The implementation of this function would involve storing the specified vertex coordinates internally and processing them during subsequent rendering operations.

- **glEnd()**: This function marks the end of a group of vertices and completes the definition of a geometric primitive. It is called after specifying the vertices using `glVertex` calls. The implementation of this function would involve finalizing the rendering of the primitive and performing any necessary cleanup.

These functions are implemented in the OpenGL library and can be accessed by including the appropriateheaders and linking the OpenGL library with your code. User-defined functions are functions that you define in your code to perform specific tasks or implement custom functionality. The implementation ofthese functions depends on the programming language you are using.

4.2 Implementation of user-defined functions:

User-defined functions are functions that you define in your code to perform specific tasks or implement custom functionality. The implementation of these functions depends on the programming language youare using. Here's an example of how a user-defined function can be implemented:

c++ code:

```
void displayRasterText(float x, float y, float z, const char* stringToDisplay)
{
        glRasterPos3f(x, y, z);
        for (const char* c = stringToDisplay; *c != '\0'; c++)
        {
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, *c);
        }
}
```

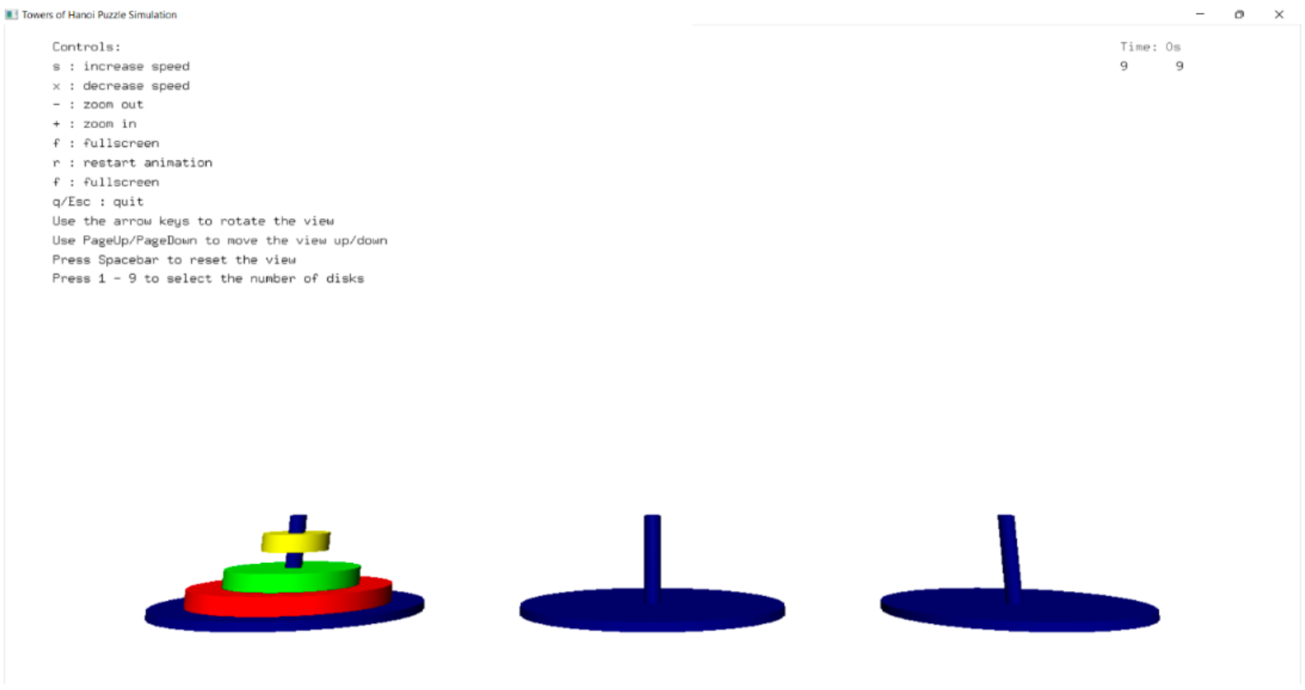
- we define a function called `displayRasterText` that takes the coordinates (`x`, `y`, `z`) and astring (`stringToDisplay`) as parameters.
- The function sets the raster position using `glRasterPos3f` and then iterates through eachcharacter in the string.
- It uses `glutBitmapCharacter` to render each character on the screen using the GLUT bitmapfont.
- User-defined functions can be implemented in a similar manner by defining the necessary logicand operations based on your specific requirements.

- The implementation details will vary depending on the functionality you want to achieve.

  In this example, we define a function called `displayRasterText` that takes the coordinates (`x`, `y`, `z`) and a string (`stringToDisplay`) as parameters. The function sets the raster position using `glRasterPos3f` and then iterates through each character in the string. It uses `glutBitmapCharacter` to render each character on the screen using the GLUT bitmap font.

  User-defined functions can be implemented in a similar manner by defining the necessary logic and operations based on your specific requirements. The implementation details will vary depending on thefunctionality you want to achieve.

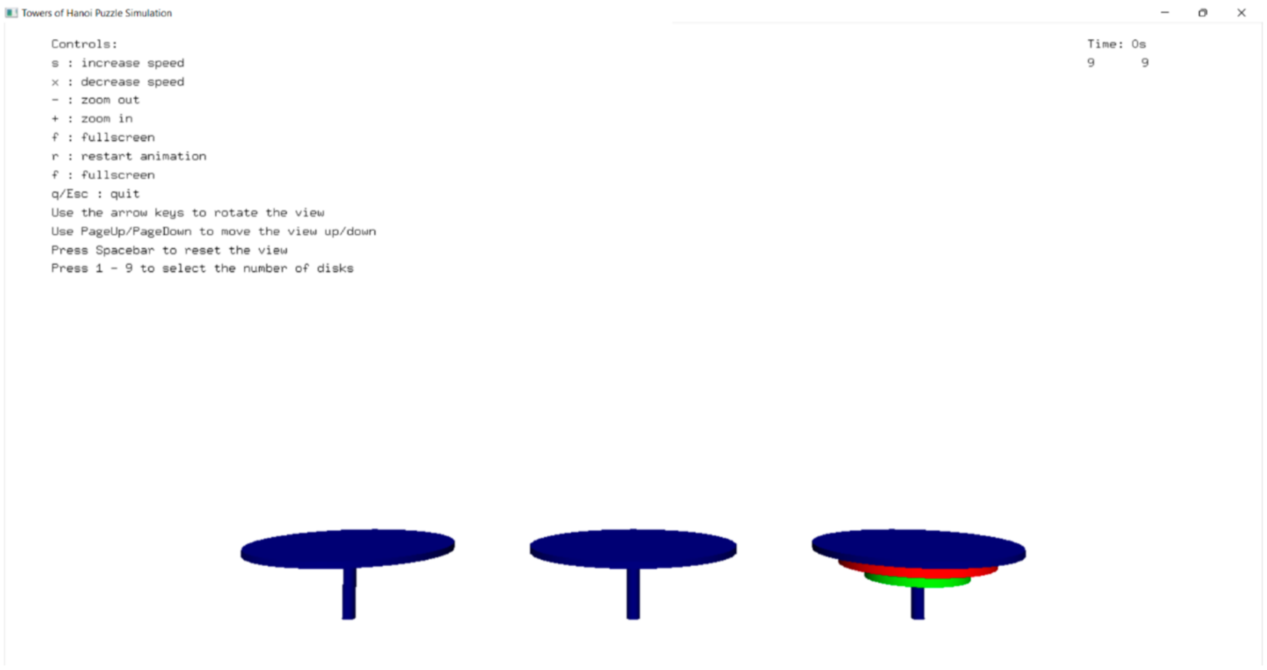
Chapter 5: Test Cases and Snapshots:



```
Controls:
s : increase speed
x : decrease speed
- : zoom out
+ : zoom in
f : fullscreen
r : restart animation
f : fullscreen
q/Esc : quit
Use the arrow keys to rotate the view
Use PageUp/PageDown to move the view up/down
Press Spacebar to reset the view
Press 1 - 9 to select the number of disks
```

```
Time: 0s
9      9
```

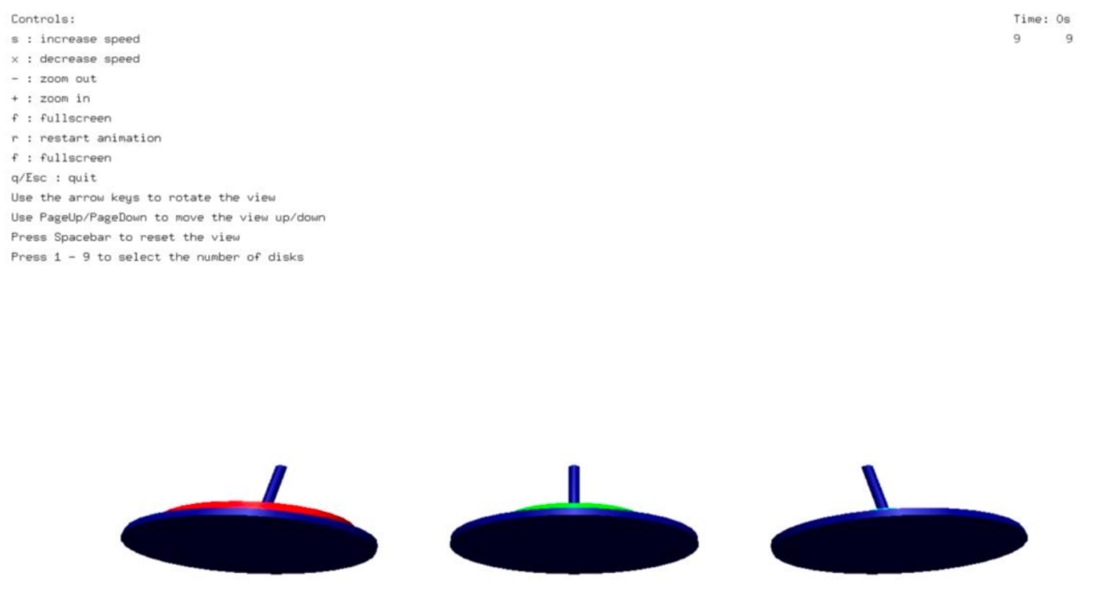
Figure 2: Main Page with instructions



```
Controls:
s : increase speed
x : decrease speed
- : zoom out
+ : zoom in
f : fullscreen
r : restart animation
f : fullscreen
q/Esc : quit
Use the arrow keys to rotate the view
Use PageUp/PageDown to move the view up/down
Press Spacebar to reset the view
Press 1 - 9 to select the number of disks
```

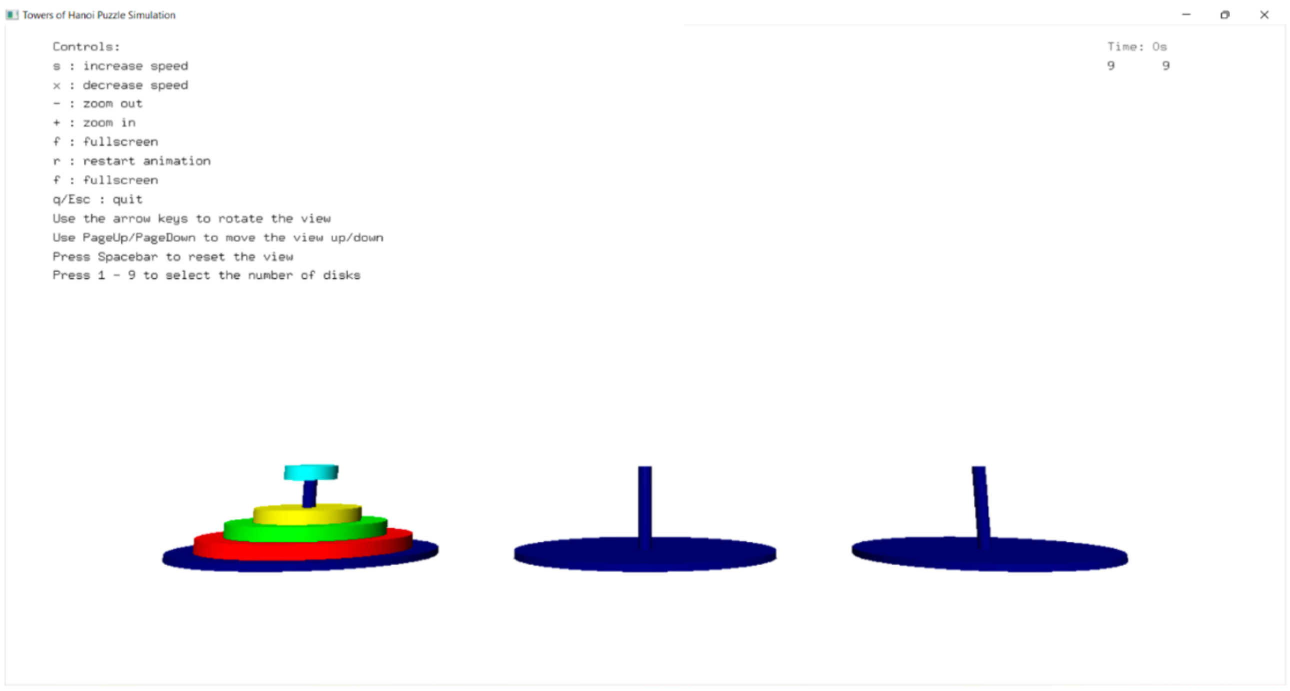
```
Time: 0s
9      9
```

Figure 3: Tower rotated downwards using the DOWN navigation key



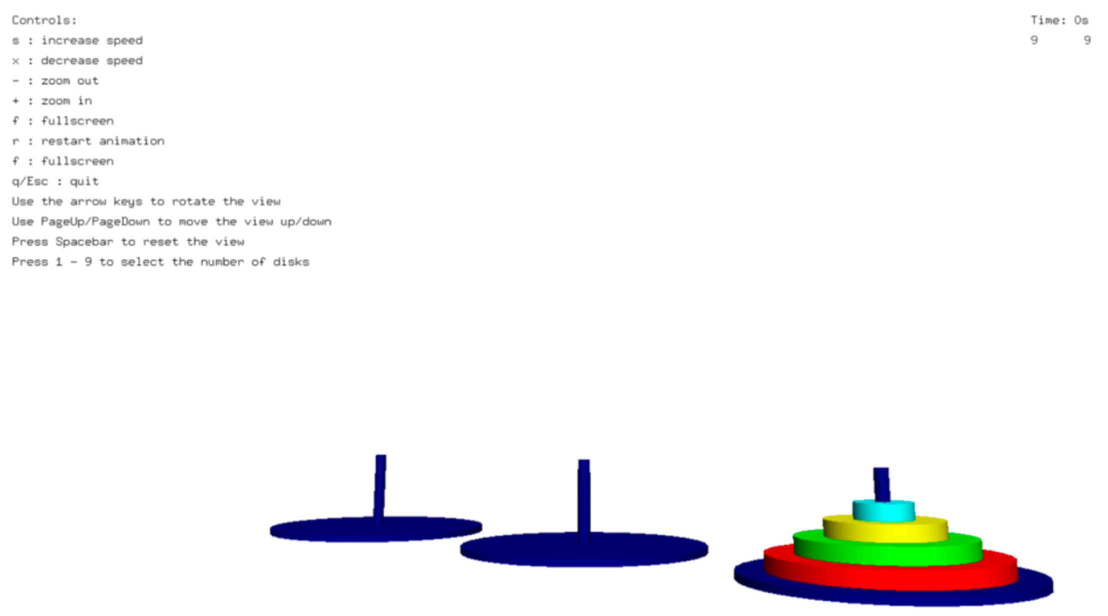
```
Controls:                                                          Time: 0s
s : increase speed                                                 9     9
x : decrease speed
- : zoom out
+ : zoom in
f : fullscreen
r : restart animation
f : fullscreen
q/Esc : quit
Use the arrow keys to rotate the view
Use PageUp/PageDown to move the view up/down
Press Spacebar to reset the view
Press 1 - 9 to select the number of disks
```

Figure 4 : Tower rotated upward with UP navigation key



```
Controls:                                                          Time: 0s
s : increase speed                                                 9     9
x : decrease speed
- : zoom out
+ : zoom in
f : fullscreen
r : restart animation
f : fullscreen
q/Esc : quit
Use the arrow keys to rotate the view
Use PageUp/PageDown to move the view up/down
Press Spacebar to reset the view
Press 1 - 9 to select the number of disks
```

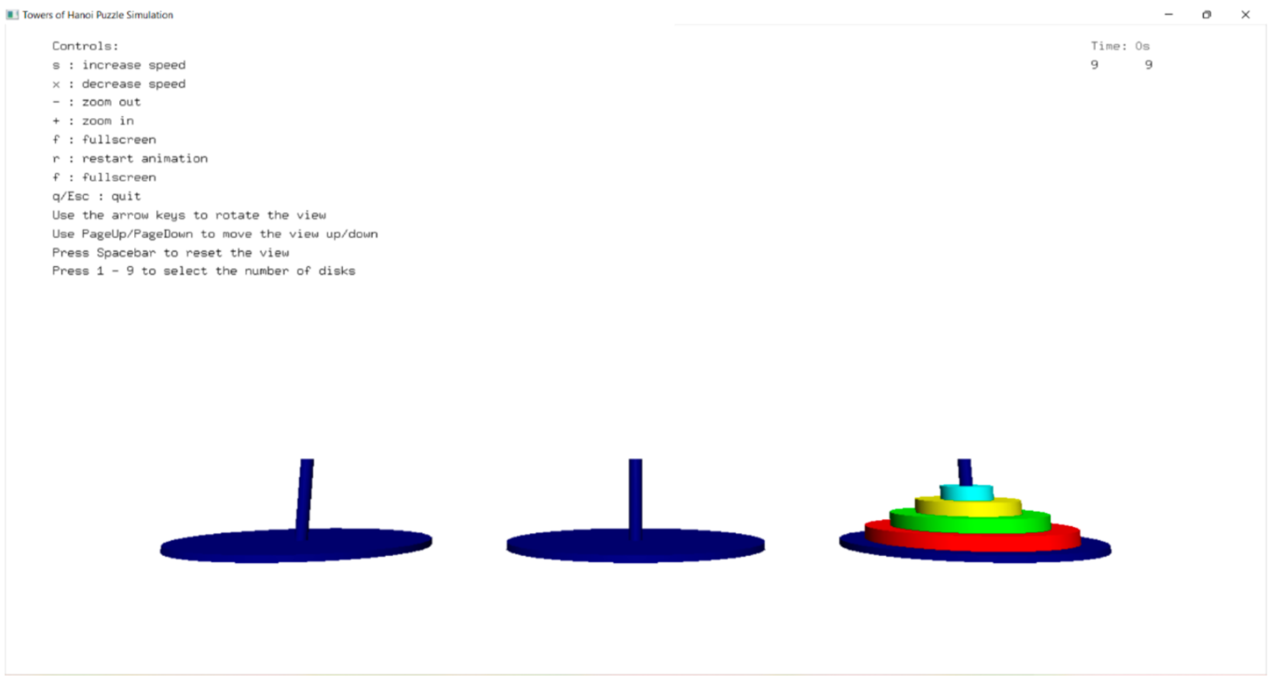
Figure 5 : No of Disks selected as 4



```
Controls:
s : increase speed
x : decrease speed
- : zoom out
+ : zoom in
f : fullscreen
r : restart animation
f : fullscreen
q/Esc : quit
Use the arrow keys to rotate the view
Use PageUp/PageDown to move the view up/down
Press Spacebar to reset the view
Press 1 - 9 to select the number of disks
```

```
Time: 0s
9      9
```

Figure 6 : View rotated clockwise using LEFT navigation key



```
Controls:
s : increase speed
x : decrease speed
- : zoom out
+ : zoom in
f : fullscreen
r : restart animation
f : fullscreen
q/Esc : quit
Use the arrow keys to rotate the view
Use PageUp/PageDown to move the view up/down
Press Spacebar to reset the view
Press 1 - 9 to select the number of disks
```

```
Time: 0s
9      9
```

Figure 7 : Result

Conclusion

- In conclusion, developing a Tower of Hanoi simulation project offers a valuable opportunity to engage users in a challenging and intellectually stimulating puzzle. By creating an interactive and visually appealing simulation, users can enhance their problem-solving skills while exploring the fascinating mathematical concepts behind the Tower of Hanoi problem.

- Through the implementation of various enhancements, such as a user-friendly GUI, animations, difficulty levels, time tracking, and scoring, the project can provide an immersive and enjoyable experience for users of all skill levels. Additionally, features like a hint system, customization options, leaderboards, and achievements add depth and replayability, promoting healthy competition and continuous learning.

- Moreover, incorporating educational content within the simulation allows users to deepen their understanding of the mathematical principles underlying the Tower of Hanoi problem. This educational aspect can further contribute to the project's value by encouraging users to explore and expand their knowledge while having fun.

- Finally, by optimizing the simulation for mobile devices, the Tower of Hanoi project can reach a broader audience, ensuring accessibility across various platforms.

- Overall, a well-developed Tower of Hanoi simulation project not only entertains and challenges users but also serves as an effective tool for learning and honing problem-solving skills. It can be a rewarding experience for users and a valuable addition to the field of educational games and simulations.

Future Enhancement

There are several potential future enhancements that could be considered for the Tower of Hanoi Simulation:

- **Difficulty Level:** Introduce different difficulty levels to cater to users of varying skill levels. For instance, you can start with a smaller number of disks for beginners and gradually increase the difficulty by adding more disks. This would provide a more challenging experience for advanced users.

- **Enhanced Visuals and Audio:** Improving the game's visual and audio elements can greatly enhance the player's immersion. Adding particle effects, dynamic lighting, and realistic sound effects would create a more engaging and immersive experience.

- **Customization Options**: Allowing players to customize their rings and pins with different skins, colors,or accessories would provide a personal touch.

- **Time Tracking and Scoring:** Implement a timer to track the time taken to solve the puzzle. You can also introduce a scoring system that rewards users based on their performance, such as completing the puzzle in the shortest time or with the fewest moves.

- **Mobile and VR Support:** Adapting the simulation for mobile devices or virtual reality platforms would expand its reach and provide a more immersive experience for players who prefer these platforms. This would require optimizing the game's controls and visuals for the target platform.

References:

1. Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version,3rd / 4th Edition,Pearson Education,2011.
2. Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5thedition. Pearson Education, 2008.
3. Wikipedia: The free encyclopaedia.
4. Geeks for Geeks and TutorialsPoint.