VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI - 590018

Computer Graphics Mini Project Report
On

N-QUEENS SIMULATION

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
In
Computer Science and Engineering

Submitted by
SMITHA S MAGANTI
1BG18CS110

Vidyayāmruthamashnuthe

B.N.M. Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE & Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82   Email: principal@bnmit.in, www.bnmit.org

Department of Computer Science and Engineering
2020-21

B.N.M. Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE & Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, www. bnmit.org

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vidyayāmruthamashnuthe

CERTIFICATE

Certified that the Mini Project entitled **N-Queens Simulation** carried out by **Ms. Smitha S Maganti** USN **1BG18CS110** a bona-fide student of VI Semester B.E., **B.N.M Institute of Technology** in partial fulfillment for the Bachelor of Engineering in COMPUTER SCIENCE AND ENGINEERING of the **Visvesvaraya Technological University**, Belagavi during the year 2020-21. It is certified that all corrections / suggestions indicated for internal Assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of Computer Graphics Mini Project Laboratory prescribed for the said degree.

Mrs. Akshitha Katkeri
Assistant Professor
Department of CSE
BNMIT, Bengaluru

Dr. Sahana D. Gowda
Professor and HOD
Department of CSE
BNMIT, Bengaluru

Name and Signature

Examiner 1:

Examiner 2:

ABSTRACT

The N-Queens problem is a popular classic puzzle where n number of queens are to be placed on an n x n chessboard such that no queen can attack any other queen. The queens can move diagonally or in a straight line. Therefore, to arrive at the solutions for this problem, no two queens should be placed in the same row or column or diagonal. A lot of solutions used for real word problems are tested out on this puzzle. The application integrates the concepts of computer graphics, Open GL API, C Programming with the algorithm of N-Queens problem which uses the concept of backtracking.

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project.

I would like to thank **Shri. Narayan Rao R Maanay**, Secretary, BNMEI, Bengaluru for providing the excellent environment and infrastructure in the college.

I would like to sincerely thank **Prof. T J Rama Murthy**, Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

I would like to sincerely thank **Dr. S Y Kulkarni** ,Additional Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

I would like to express my gratitude to **Prof. Eishwar N Maanay**, Dean, BNMIT, Bengaluru for his relentless support and encouragement.

I would like to thank **Dr. Krishnamurthy G N**, Principal, BNMIT, Bengaluru for his constant encouragement.

I would like to thank, **Dr. Sahana D. Gowda**, Professor & Head of the Department of Computer Science and Engineering for the encouragement and motivation she provides.

I would also like to thank **Mrs. Akshitha Katkeri,** Assistant Professor, Department of Computer Science and Engineering for providing me with her valuable insight and guidance wherever required throughout the course of the project and its successful completion.

Smitha S Maganti

1BG18CS110

Table of Contents

List of Figures

Chapter 1
INTRODUCTION

1.1 Overview

Interactive computer graphics provides us with the most natural means of communicating information through a computer. Over the years advancements in computer graphics have enabled us to not only make pictures of real-world objects but also visualize abstract, synthetic objects such as mathematical surfaces and of data that have no inherent geometry, such as survey results. Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. The computer on receiving signals from the input device can modify the displayed picture appropriately. To the user it appears that the picture is changing instantaneously in response to his commands. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

1.2 Problem Statement

The aim of this application is to show the implementation of N-Queens problem; a concept where N number of queens are to be placed on an N x N chessboard such that no queen can attack any other queen. The application will be implemented using the C programming language and the OpenGL API. The objective of the application is to demonstrate the simulation of N-Queens problem and arriving at all the solutions using the principles of Computer Graphics. The application will also include user interaction through mouse events; the user can choose to either view the entire simulation or just the possible solutions.

1.3 Motivation

N-Queens simulations take O(N!) for all operations which makes it an ideal option for computer scientists and software developers to test their algorithms on this concept before applying it in relevant real world problems. The ability to visualize how N-Queens solutions are obtained will give an insight on the working of the algorithm used. The ability to develop this visualization using C programming and the OpenGL API serves as a motivation to develop this application.

1.4 Computer Graphics

Computer graphics and multimedia technologies are becoming widely used in educational applications because they facilitate non-linear, self-learning environments that particularly suited to abstract concepts and technical information.

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as CGI. Important topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others.

The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

1.5 OpenGL API

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware- accelerated rendering. Silicon Graphics Inc., (SGI) began developing OpenGL in 1991 and released it on June 30, 1992; applications use it extensively in the fields of computer- aided design (CAD), virtual reality, scientific visualization, information visualization,

flight simulation, and video games. Since 2006 OpenGL has been managed by the non- profit technology consortium Khronos Group. The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware. The API is defined as a set of functions which may be called by the client program, alongside a set of named integer constants. In addition to being language-independent, OpenGL is also cross- platform.
Given that creating an OpenGL context is quite a complex process, and given that it varies between operating systems, automatic OpenGL context creation has become a common feature of several game-development and user-interface libraries, including SDL, Allegro, SFML, FLTK, and Qt. A few libraries have been designed solely to produce an OpenGL-capable window. The first such library was OpenGL Utility Toolkit (GLUT), later superseded by freeglut. GLFW is a newer alternative.

1.5.1 OpenGL API Architecture

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions.

Per Vertex Operations:

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

Primitive Assembly:

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half space, defined by a plane.

Pixel Operation:

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.

Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Colour and depth values are assigned for each fragment square.

Fragment Operations:

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

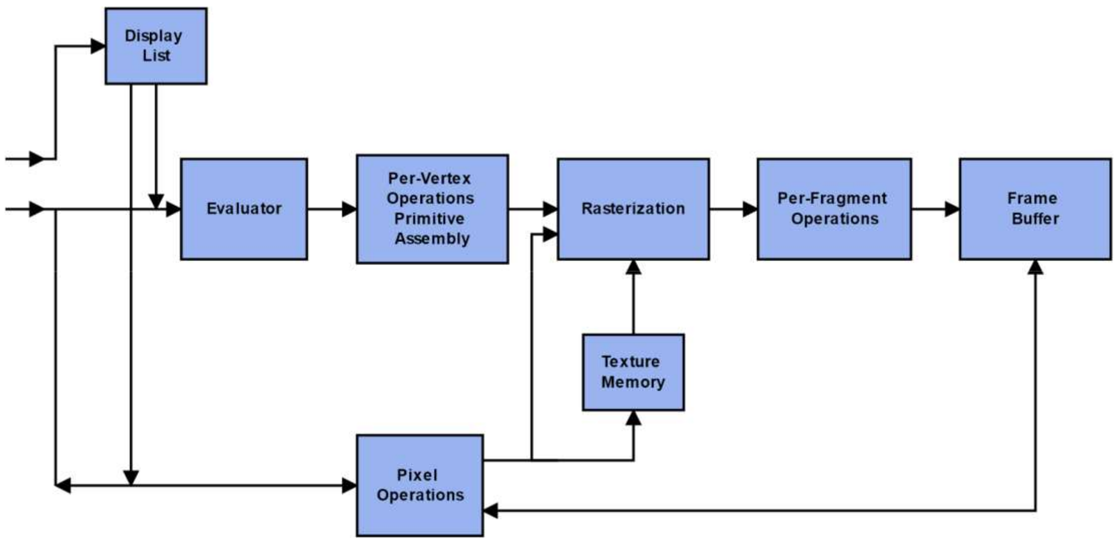


Figure 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

1.6 Applications of Computer Graphics

Although many applications span two, three, or even all of these areas, the development of the field was based, for the most part, on separate work in each domain. We can classify applications of computer graphics into four main areas:

1.6.1 Display of Information

Graphics has always been associated with the display of information. Examples of the use of orthographic projections to display floorplans of buildings can be found on 4000-year-old Babylonian stone tablets. Mechanical methods for creating perspective drawings were developed during the Renaissance. Countless engineering students have become familiar with interpreting data plotted on log paper. More recently, software packages that allow interactive design of charts incorporating colour, multiple data sets, and alternate plotting methods have become the norm. In fields such as architecture and mechanical design, hand drafting is being replaced by computer-based drafting systems using plotters and workstations. Medical imaging uses computer graphics in a number of exciting ways.

1.6.2 Design

Professions such as engineering and architecture are concerned with design. Although their applications vary, most designers face similar difficulties and use similar methodologies. One of the principal characteristics of most design problems is the lack of a unique solution. Hence, the designer will examine a potential design and then will modify it, possibly many times, in an attempt to achieve a better solution. Computer graphics has become an indispensable element in this iterative process.

1.6.3 Simulation

Some of the most impressive and familiar uses of computer graphics can be classified as simulations. Video games demonstrate both the visual appeal of computer graphics and our ability to generate complex imagery in real time. Computer-generated images are also the heart of flight simulators, which have become the standard method for training pilots.

1.6.4 User Interfaces

The interface between the human and the computer has been radically altered by the use of computer graphics. Consider the electronic office. The figures in this book were produced through just such an interface. A secretary sits at a workstation, rather than at a desk equipped with a typewriter. This user has a pointing device, such as a mouse, that allows him to communicate with the workstation.

Chapter 2

LITERATURE SURVEY

2.1 History of Computer Graphics

The term “computer graphics” was coined in 1960 by William Fetter, a designer at Boeing, to describe his own job, the field can be said to have first arrived with the publication in 1963 of Ivan Sutherland’s Sketchpad program, as part of his Ph.D. thesis at MIT. Sketchpad, as its name suggests, was a drawing program. Beyond the interactive drawing of primitives such as lines and circles and their manipulation – in particular, copying, moving and constraining – with use of the then recently invented light pen, Sketchpad had the first fully-functional graphical user interface (GUI) and the first algorithms for geometric operations such as clip and zoom. Interesting, as well, is that Sketchpad’s innovation of an object-instance model to store data for geometric primitives foretold object-oriented programming. Coincidentally, on the hardware side, the year 1963 saw the invention by Douglas Engelbart at the Stanford Research Institute of the mouse, the humble device even today carrying so much of GUI on its thin shoulders.

Subsequent advances through the sixties came thick and fast: raster algorithms, the implementation of parametric surfaces, hidden-surface algorithms and the representation of points by homogeneous coordinates, the latter crucially presaging the foundational role of projective geometry in 3D graphics, to name a few. Flight simulators were the killer app of the day and companies such as General Electric and Evans & Sutherland, 6 co-founded by Douglas Evans and Ivan Sutherland, wrote simulators with real-time graphics.

The seventies, brought the Z-buffer for hidden surface removal, texture mapping, Phong’s lighting model – all crucial components of the OpenGL API (Application Programming Interface) we’ll be using soon – as well as keyframe-based animation. Photorealistic rendering of animated movie keyframes almost invariably deploys ray tracers, which were born in the seventies too. Through the nineties, as well, the use of 3D effects in movies became pervasive.

The Terminator and Star Wars series, and Jurassic Park, were among the early movies to set the standard for CGI. Toy Story from Pixar, 8 released in 1995, has special importance in the history of 3D CGI as the first movie to be entirely computer- generated – no scene was ever pondered through a glass lens, nor any recorded on a photographic reel! It was cinema without film. Quake, released in 1996, the first of the hugely popular Quake series of games, was the first fully 3D game.

Another landmark from the nineties of particular relevance to us was the release in 1992 of OpenGL, the open-standard cross-platform and cross language 3D graphics API, by Silicon Graphics. OpenGL is actually a library of calls to perform 3D tasks, which can be accessed from programs written in various languages and running over various operating systems. That OpenGL was high-level (in that it frees the applications programmer from having to care about such low-level tasks as representing primitives like lines and triangles in the raster, or rendering them to the window) and easy to use (much more so than its predecessor 3D graphics API, PHIGS, standing for Programmer's Hierarchical Interactive Graphics System) first brought 3D graphics programming to the "masses". What till then had been the realm of a specialist was now open to a casual programmer following a fairly amicable learning curve.

Since its release OpenGL has been rapidly adopted throughout academia and industry. It's only among game developers that Microsoft's proprietary 3D API, Direct3D, which came soon after OpenGL bearing an odd similarity to it but optimized for Windows, is more popular.

The story of the past decade has been one of steady progress, rather than spectacular innovations in CG. Hardware continues to get faster, better, smaller and cheaper, continually pushing erstwhile high-end software down market, and raising the bar for new products. The almost complete displacement of CRT monitors by LCD and the emergence of high-definition television are familiar consequences of recent hardware evolution.

2.2 Related Work

Computer Aided Design (CAD):

Most of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, in almost all kinds of Industries. Its use in designing electronic systems is known as electronic design automation (EDA). In mechanical design it is known as mechanical design automation (MDA) or computer-aided drafting (CAD), which includes the process of creating a technical drawing with the use of computer software.

Computer Simulation

Computer simulation is the reproduction of the behavior of a system using a computer to simulate the outcomes of a mathematical model associated with said system. Since they allow to check the reliability of chosen mathematical models, computer simulations have become a useful tool for the mathematical modeling of many natural systems in physics (computational physics), astrophysics, climatology, chemistry, biology and manufacturing, human systems in economics, psychology, social science, health care and engineering. Simulation of a system is represented as the running of the system's model. It can be used to explore and gain new insights into new technology and to estimate the performance of systems too complex for analytical solutions.

Digital Art

Digital art is an artistic work or practice that uses digital technology as part of the creative or presentation process. Since the 1970s, various names have been used to describe the process, including computer art and multimedia art. Digital art is itself placed under the larger umbrella term new media art. With the rise of social media and the internet, digital art application of computer graphics. After some initial resistance, the impact of digital technology has transformed activities such as painting, drawing, sculpture and music/sound art, while new forms, such as net art, digital installation art, and virtual reality, have become recognized artistic practices. More generally the term digital artist is used to describe an artist who makes use of digital technologies in the production of art. In an expanded sense, "digital art" is contemporary art that uses the methods of mass production or digital media.

Virtual Reality

Virtual reality (VR) is an experience taking place within a computer-generated reality of immersive environments can be similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. gaming) and educational purposes (i.e. medical or military training). Other, distinct types of VR style technology include augmented reality and mixed reality. Currently standard virtual reality systems use either virtual reality headsets or multi-projected environments to generate realistic images, sounds and other sensations that simulate a user's physical presence in a virtual environment. A person using virtual reality equipment is able to look around the artificial world, move around in it, and interact with virtual features or items. The effect is commonly created by VR headsets consisting of a head-mounted display with a small screen in front of the eyes, but can also be created through specially designed rooms with multiple large screens. Virtual reality typically incorporates auditory and video feedback, but may also allow other types of sensory and force feedback through haptic technology.

Video Games

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a two- or three-dimensional video display device such as a TV screen, virtual reality headset or computer monitor. Since the 1980s, video games have become an increasingly important part of the entertainment industry, and whether they are also a form of art is a matter of dispute. The electronic systems used to play video games are called platforms. Video games are developed and released for one or several platforms and may not be available on others. Specialized platforms such as arcade games, which present the game in a large, typically coin-operated chassis, were common in the 1980s in video arcades, but declined in popularity as other, more affordable platforms became available. These include dedicated devices such as video game consoles, as well as general-purpose computers like a laptop, desktop or handheld computing devices.

Chapter 3

SYSTEM REQUIREMENTS

3.1 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

The following are the software requirements for the application :

- Operating System: Windows 10
- Compiler: GNU C/C++ Compiler
- Development Environment: Visual Studio 2019 Community Edition
- API: OpenGL API & Win32 API for User Interface and Interaction

3.2 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware.

- CPU: Intel or AMD processor
- Cores: Dual-Core (Quad-Core recommended)
- RAM: minimum 4GB (>4GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommend

Chapter 4

SYSTEM DESIGN

4.1 Proposed System

Chess composer Max Bezzel published the eight queens puzzle in 1848. Franz Nauck published the first solution for the eight queens puzzle in 1850. Nauck also extended the puzzle to the N-Queens problem, with queens on a chessboard of n×n squares. Since then, many mathematicians, including Carl Friedrich Gauss, have worked on both the eight queens puzzle and its generalized n-queens version

Finding all solutions to the N-Queens puzzle is a good example of a simple but nontrivial problem. For this reason, it is often used as an example problem for various programming techniques, including nontraditional approaches such as constraint programming, logic programming or genetic algorithms. Most often, it is used as an example of a problem that can be solved with a recursive and backtracking algorithm.

The backtracking depth-first search algorithm used in this application constructs the search tree by considering one row of the board at a time, eliminating most nonsolution board positions at a very early stage in their construction.

The time complexity of N-Queens simulation solved using the backtracking algorithm is 0(N!). The simulation starts and shows all the possible positions the queens can take on the chessboard without attacking each other.

The proposed system for the application aims to simulate and demonstrate the N-Queens problem using C Programming and OpenGL API.
The OpenGL API provides us with built in functions to construct primitives which will be used to build the model of the N-Queens. This will be

constructed in the memory. The application will recursively backtrack through the N-Queens search tree in the memory and draw the queens and the chessboard on the screen.

4.1.1 Data Flow Diagram

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data- flow diagram has no control flow, there are no decision rules and no loops. For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The refined representation of a process can be done in another data-flow diagram, which subdivides this process into sub-processes. The data-flow diagram is part of the structured-analysis modelling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram. A special form of data- flow plan is a site-oriented data-flow plan.

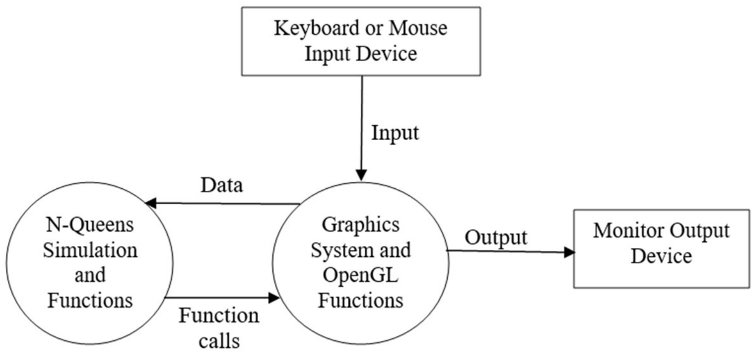

Keyboard or Mouse Input Device
Input
Data
N-Queens Simulation and Functions
Graphics System and OpenGL Functions
Output
Monitor Output Device
Function calls

Figure 4.1 Level 0 Dataflow Diagram of the Proposed System

Figure 4.1 shows the Level 0 Dataflow diagram of the proposed application for the N-Queens. The keyboard and mouse devices are used for input to the application. The graphics system processes these user keyboard/mouse interactions using built in OpenGL functions like glutKeyboardFunc(*void), glutMouseFunc(*void) and glutAttachMenu(*void). N-Queens simulation is constructed in the memory using the user inputs sent to it by the Graphics System. The simulation built in the memory is used by the graphics system to draw the chessboard and queens onto the screen using OpenGL Functions.

4.2 Flowchart

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows.

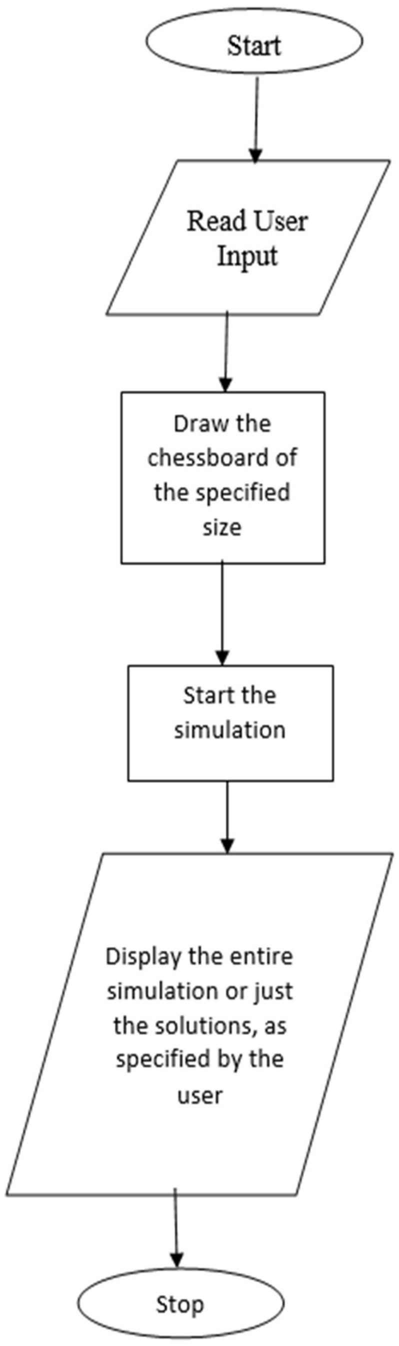

Start
Read User Input
Draw the chessboard of the specified size
Start the simulation
Display the entire simulation or just the solutions, as specified by the user
Stop

Figure 4.2 Flowchart of the Proposed System

Chapter 5

IMPLEMENTATION

5.1 Module Description

- **void drawSquare(GLint x1, GLint y1, GLint x2, GLint y2, GLint x3, GLint y3, GLint x4, GLint y4)**

  This function draws the squares for the chessboard at the points (x1,y1), (x2,y2), (x3,y3), (x4,y4).

- **void put_queen(int x1, int y1)**

  This function draws the queen on the square whose bottom left corner's coordinates are (x1,y1).

- **void queen(int row, int n)**

  This is a recursive function which backtracks the various positions of the queen until the right combination is found. It takes two parameters, current row that is being checked and the number of queens. The queens are placed on the squares with no conflicts and are marked green.

- **int place(int row, int column)**

  This function checks if there are any conflicts if a queen is placed in the square of the specified row and column, which are taken as arguments. It checks for the diagonal conflicts and column/row conflicts. It returns 0 if there is a conflict, else it returns 1.

- **void mark(int column, int row)**

  This function is used to color the squares which are conflicted red.

- **void putrect(int column, int row)**

  This function is used to re-draw the squares of appropriate colors in the specified row and column on the chessboard in the event of the queen that was placed on that square having a conflict.

- **void displaycount()**

  This function is used to display the count of the total solutions of N-Queens for the chosen value of N.

- **void displayheader()**

  This function is used to display the heading of the problem.

- **void nvalue(int choice)**

  This function is used to create a menu in the first page, which is attached to the right mouse button. It contains a list of N values for the users to select. When a choice is selected from the menu, the code under that choice is executed.

- **void mouse(int choice)**

  This function is used to create a menu in the second page, which is attached to the right mouse button. It contains the options to either view the simulation of the problem or just the solutions. When a choice is selected from the menu, the code under that choice is executed.

5.2 High Level Code

5.2.1 Built-In Functions

- **void glClear(glEnum mode);**

  Clears the buffers namely color buffer and depth buffer. mode refers to GL_COLOR_BUFFER_BIT or DEPTH_BUFFER_BIT.

- **void glLoadIdentity( );**
  Sets the current transformation matrix to identity matrix.

- **void glutBitmapCharacter(void *font, int character);**
  Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font. The fonts used are

GLUT_BITMAP_TIMES_ROMAN_24.

- **void glRasterPos[234][ifd](GLfloat x, GLfloat y, GLfloat z);**

  This position, called the raster position, is used to position pixel and bitmap write operations

- **void glutInit(int *argc, char **argv);**

  Initializes GLUT; the arguments from main are passed in and can be used by the application.

- **void glutInitDisplayMode(unsigned int mode);**

  Requests a display with the properties in the mode; the value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **void glutCreateWindow(char *title);**

  Creates a window on display; the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

- **void glutMainLoop();**

  Causes the program to enter an event-processing loop.

- **void glutDisplayFunc(void (*func)(void))**

  Registers the display function func that is executed when the window needs to be redrawn.

- **void glClearColor(GLclampf r, GLclampf g, GLclamp b, Glclamp a)**

  Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

- **void glViewport(int x ,int y, GLsizei width, GLsizei height)**

  Specifies the width*height viewport in pixels whose lower left corner is at (x,y) measured from the origin of the window.

- **void glColor3[b I f d ub us ui](TYPE r, TYPE g, TYPE b)**

  Sets the present RGB colors. Valid types are bytes(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui). The maximum and minimum value for floating point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are different, for eg, 255 and 0 for unsigned bytes.

- **void glutInitWindowSize(int width, int height);**

  Specifies the initial height and width of the window in pixels.

- **void glutReshapeFunc(void *f(int width, int height));**

  It registers the reshape callback function. The callback function returns the height and width of the new window. The reshape callback invokes the display callback.

- **void createMenu(void);**

  This function is used to create menus which are used as options in program.

- **void glutAttachMenu(button);**

  The function glutAttachMenu attaches a mouse button for the current window to the identifier of the current menu. The arguments to be given are GLUT_LEFT_BUTTOM, GLUT_RIGHT_BUTTON, etc.

- **void glutAddMenuEntry(char *name, int value);**

  The function glutAddMenuEntry adds a menu entry to the bottom of the current menu. The string name will be displayed for the newly added menu entry. If the menu entry is selected by the user, the menu's callback will be called passing value as the callback's parameter.

- **void glRecti(GLint *x1*, GLint *y1*, GLint *x2*, GLint *y2*);**

  This is used to draw a rectangle. (x1,y1) specifies oner vertex, and (x2,y2) specifies the opposite vertex. The parameters can take integer, floating point or double values.

- **void glutPostRedisplay();**

  The glutPostRedisplay function marks the current window as needing to be redisplayed. The next iteratio through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane.

5.2.2 User Implementation

- **Reshape Function**

```
void reshape(int w, int h) {
   glViewport(100, 100, w, h);
}
```

- **Display Function**

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   if (page == 1) { //Displaying the contents of first page
            y = 600;
            char string1[30] = "N-Queens Simulation!";
            BitMapDisplay(string1);
            char string2[100] = "The N-Queens problem is a popular
            classic puzzle where";
            BitMapDisplay(string2);
            char string3[80] = "n number of queens are to be placed on an
            n x n";
            BitMapDisplay(string3);
            char string3a[80] = "chessboard such that no queen can attack
            any other queen.";
            BitMapDisplay(string3a);
            char string4[60] = "The queens can move diagonally or in a
            straight line. ";
            BitMapDisplay(string4);
            char string5[100] = "Therefore, to arrive at the solutions for
            this problem, no two";
```

```
                BitMapDisplay(string5);
                char string6[100] = "queens should be placed in the same
                row/column/diagonal.";
                BitMapDisplay(string6);
                char string7[80] = "A lot of algorithms used for real word
                problems are tested";
                BitMapDisplay(string7);
                char string7a[80] = "out on this puzzle.";
                BitMapDisplay(string7a);
                char string8[70] = "To view the simlulation, Right click and
                select the number";
                BitMapDisplay(string8);
                char string8a[70] = "of queens, ranging from 4 to 8.";
                BitMapDisplay(string8a);
                glutCreateMenu(nvalue);
                glutAddMenuEntry("4", 1);
                glutAddMenuEntry("5", 2);
                glutAddMenuEntry("6", 3);
                glutAddMenuEntry("7", 4);
                glutAddMenuEntry("8", 5);
                glutAttachMenu(GLUT_RIGHT_BUTTON);
        }
        if (page == 2) { //Displaying the contents of the second page
                col = 1;
                glClear(GL_COLOR_BUFFER_BIT);
                glClearColor(0.0, 0.0, 1.0, 1.0);
                glColor3f(0, 0, 1);
                glRectf(0, n * SIDE_LENGTH, 1000, n * SIDE_LENGTH +
                35);
                GLint x, y;
                for (x = 0; x < n * SIDE_LENGTH; x += SIDE_LENGTH)
                {
                        if (n % 2 == 0) {
```

```
                        if (col == 0)
                            col = 1;
                        else
                            col = 0;
                    }
                    for (y = 0; y < n * SIDE_LENGTH; y +=
                    SIDE_LENGTH)
                    {
                        drawSquare(x, y, x, y + SIDE_LENGTH, x +
                        SIDE_LENGTH, y + SIDE_LENGTH, x +
                        SIDE_LENGTH, y);
                    }
                }
                char string[40] = "Right click for more options ";
                glColor3f(1.0, 1.0, 1.0);
                glRasterPos2f(0, n * (SIDE_LENGTH+2));
                int len = (int)strlen(string);
                for (int i = 0; i < len; i++) {
                    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROM
                    AN_24, string[i]);
                }
                glutCreateMenu(mouse);
                glutAddMenuEntry("Full Simulation", 1);
                glutAddMenuEntry("Possible Solutions", 2);
                glutAttachMenu(GLUT_RIGHT_BUTTON);
        }
        glFlush();
    }

    void BitMapDisplay(char str[])
    {
        /*To print the characters on the first page of the screen*/
        y -= 40;
    glColor3f(1, 1, 1);
    glRasterPos2f(0, y);
    int len = (int)strlen(str);
    for (int i = 0; i < len; i++) {
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
            str[i]);
    }
}

void drawSquare(GLint x1, GLint y1, GLint x2, GLint y2, GLint x3,
GLint   y3, GLint x4, GLint y4){
    /* Function to draw squares for the chessboard*/
    if (n % 2 == 0) {
            if (col == 0)
            {
                    glColor3f(1.0, 1.0, 1.0);
                    col = 1;
            }
            else
            {
                    glColor3f(0, 0, 0);
                    col = 0;
            }
    }
    else {
            if (col == 0)
            {
                    glColor3f(0, 0, 0);
                    col = 1;
            }
            else
            {
                    glColor3f(1, 1, 1);
                    col = 0;
              }
        }
        glBegin(GL_POLYGON);
              glVertex2i(x1, y1);
              glVertex2i(x2, y2);
              glVertex2i(x3, y3);
              glVertex2i(x4, y4);
        glEnd();
  }
```

- **Menu and Mouse Functions:**

```
void nvalue(int choice)
{
  /*Displays the different choices for N on the first page*/
  if (choice == 1)
        n = 4;
  if (choice == 2)
        n = 5;
  if (choice == 3)
        n = 6;
  if (choice == 4)
        n = 7;
  if (choice == 5)
        n = 8;
  page = 2;
  glutPostRedisplay();
}

void mouse(int choice)
{
  /*Displays the different choices for simulating the problem on the
  second page*/
```

```
        glColor3f(0, 0, 1);
        glRectf(0, n * SIDE_LENGTH, 1000, n * SIDE_LENGTH + 50);
        count = 1;  /*Total Solutions*/
        if (choice == 1) {
                sol_only = 0;
                sleeptime = 275;
                col = 1;
        }
        if (choice == 2) {
                sol_only = 1; /*Displays only solutions*/
                sleeptime = 0;
                col = 1;
        }
        displayheader();
        queen(0, n);
        glutPostRedisplay();
    }
```

- **Function to draw the queen:**

```
    void put_queen(int x1, int y1)
    {
                glColor3f(0.0, 0.0, 1.0);
                int mid = SIDE_LENGTH / 2;
                glBegin(GL_POINTS);
                        glVertex2i(x1 + mid + 12, y1 + SIDE_LENGTH - 16);
                        glVertex2i(x1 + mid - 12, y1 + SIDE_LENGTH - 16);
                        glVertex2i(x1 + mid + 24, y1 + SIDE_LENGTH - 24);
                        glVertex2i(x1 + mid - 24, y1 + SIDE_LENGTH - 24);
                glEnd();
                glBegin(GL_LINE_LOOP);
                        glVertex2i(x1 + mid - 24, y1 + SIDE_LENGTH - 24);
                        glVertex2i(x1 + mid - 16, y1 + 16);
                        glVertex2i(x1 + mid + 16, y1 + 16);
                glVertex2i(x1 + mid + 24, y1 + SIDE_LENGTH - 24);
                glVertex2i(x1 + mid + 12, y1 + SIDE_LENGTH - 32);
                glVertex2i(x1 + mid + 12, y1 + SIDE_LENGTH - 16);
                glVertex2i(x1 + mid, y1 + SIDE_LENGTH - 32);
                glVertex2i(x1 + mid - 12, y1 + SIDE_LENGTH - 16);
                glVertex2i(x1 + mid - 12, y1 + SIDE_LENGTH - 32);
        glEnd();
        glFlush();
}
```

- **Recursive function to solve N-Queens problem:**

```
void queen(int row, int n)
{
        int column;
        for (column = 0; column < n; column++)
        {
                if (place(row, column))
                {
                        board[row] = column;
                        Sleep(sleeptime);
                        glColor3f(0, 1, 0);
                        glRecti(column * SIDE_LENGTH, row *
                        SIDE_LENGTH, column * SIDE_LENGTH +
                        SIDE_LENGTH, row * SIDE_LENGTH +
                        SIDE_LENGTH);
                                put_queen(column * SIDE_LENGTH,
                                row * SIDE_LENGTH);
                        if(sol_only == 1)
                                displaycount();
                        Sleep(sleeptime);
                        if (row == n - 1) {
                                /* If all the rows are covered */
                                if (sol_only != 1)
```

```
                                        displaycount();
                                    count++;
                                    Sleep(2000);
                                    glFlush();
                                    putrect(column, row);
                                    return;
                                }
                                else
                                {
                                    /*Move on the next row*/
                                    queen(row + 1, n);
                                    putrect(column, row);
                                }
                            }
                            else {
                                putrect(column, row);
                            }
                    }
            }
```

- **Function to check conflicts:**

```
int place(int row, int column)
{
   int i;
   /* checking diagonal and column conflicts*/
   for (i = 0; i <= row - 1; ++i)
   {
           if (board[i] == column)
           {
                   if (sol_only != 1)
                   {
                           Sleep(sleeptime);
                           mark(column, row);
```

```
                        put_queen(column * SIDE_LENGTH, row *
                        SIDE_LENGTH);
                        Sleep(sleeptime);
                    }
                    return 0;
                }
                else
                {
                    if (abs(board[i] - column) == abs(i - row))
                    {
                        if (sol_only != 1) {
                            Sleep(sleeptime);
                            mark(column, row);
                            put_queen(column * SIDE_LENGTH,
                            row * SIDE_LENGTH);
                            Sleep(sleeptime);
                        }
                    return 0;
                    }
                }
            }
        return 1;
    }
```

- **Function to mark the squares with conflicts red:**

```
void mark(int column, int row)
{
        glColor3f(255, 0, 0);
        glRectf(column * SIDE_LENGTH, row * SIDE_LENGTH,
        column * SIDE_LENGTH + SIDE_LENGTH, row *
        SIDE_LENGTH + SIDE_LENGTH);
}
```

- **Function to replace the queens that were conflicted with the appropriate chessboard squares:**

```
void putrect(int column, int row)
{
   if (column % 2 == 0 && row % 2 == 0)
            glColor3f(1.0, 1.0, 1.0);
   if (column % 2 != 0 && row % 2 != 0)
            glColor3f(1.0, 1.0, 1.0);
   if (column % 2 == 0 && row % 2 != 0)
            glColor3f(0.0, 0.0, 0.0);
   if (column % 2 != 0 && row % 2 == 0)
            glColor3f(0.0, 0.0, 0.0);
   glRectf(column * SIDE_LENGTH, row * SIDE_LENGTH, column *
   SIDE_LENGTH + SIDE_LENGTH, row * SIDE_LENGTH +
   SIDE_LENGTH);
}
```

- **Function to display the text on the screen:**

```
void displayheader()
{
            /*To display the heading*/
            char string[10] = {0};
            _itoa_s(n, string, 10, 10);
            glColor3f(0, 0, 0);
            glRasterPos2f(0, n * (SIDE_LENGTH + 2));
            int i;
            int len = (int)strlen(string);
            for (i = 0; i < len; i++) {
                     glutBitmapCharacter(GLUT_BITMAP_TIMES_ROM
                     AN_24, string[i]);
            }
            char string2[40] = "-Queens simulation! Total Solutions: ";
```

```
                glRasterPos2f(15, n *(SIDE_LENGTH + 2));
                len = (int)strlen(string2);
                for (i = 0; i < len; i++) {
                        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROM
                        AN_24, string2[i]);
                }
        }
        void displaycount()
        {
                /*To display the count of the solutions shown*/
                glColor3d(0, 0, 1);
                glRectf(400, n * (SIDE_LENGTH + 2), 400 + 24, n *
                (SIDE_LENGTH + 2) + 24);
                char c[10] = { 0 };
                _itoa_s(count, c, 10, 10);
                glColor3f(0, 0, 0);
                glRasterPos2f(400, n * (SIDE_LENGTH + 2));
                int len = (int)strlen(c);
                for (int i = 0; i < len; i++) {
                        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROM
                        AN_24, c[i]);
                }
        }
```

- **Initialization function:**

```
        void init() {
                glClearColor(0.0, 0.0, 1.0, 1.0);
                glClear(GL_COLOR_BUFFER_BIT);
                glPointSize(4.0);
                glMatrixMode(GL_PROJECTION);
                gluOrtho2D(0, 700, 0, 700);
                glMatrixMode(GL_MODELVIEW);
                glLoadIdentity();
        }
```

- **Main Function:**

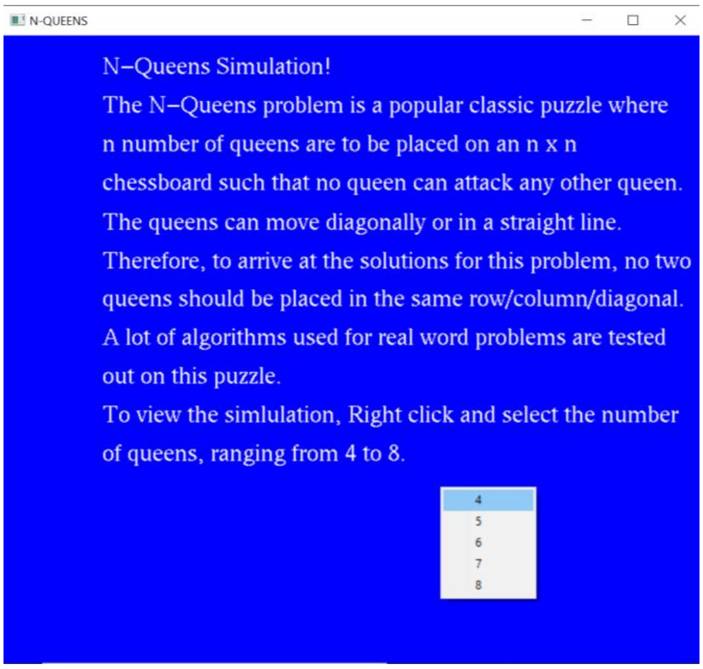
```
        int main(int argc, char** argv)
        {
                glutInit(&argc, argv);
                glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
                glutInitWindowSize(700, 700);
                glutInitWindowPosition(0, 0);
                glutCreateWindow("N-QUEENS");
                init();
                glutReshapeWindow(700,700);
                glutReshapeFunc(reshape);
                glutDisplayFunc(display);
                glutMainLoop();
                return 0;
        }
```

Chapter 6

RESULTS

6.1 Screen 1:

The first screen of the application gives a brief description about N-Queens, and allows user to select the value of N on clicking the right mouse button.



N-QUEENS

N–Queens Simulation!
The N–Queens problem is a popular classic puzzle where n number of queens are to be placed on an n x n chessboard such that no queen can attack any other queen.
The queens can move diagonally or in a straight line.
Therefore, to arrive at the solutions for this problem, no two queens should be placed in the same row/column/diagonal.
A lot of algorithms used for real word problems are tested out on this puzzle.
To view the simlulation, Right click and select the number of queens, ranging from 4 to 8.

Figure 6.1 First Screen of the Application, where the value of N is selected as 4

6.2 Screen 2:

This screen of the application displays an N x N chessboard, and the user can select two options from the menu that is attached to the right button of the mouse. They are:

1 To view the complete simulation of the N-Queens problem
2 To view only solutions of the N-Queens problem.

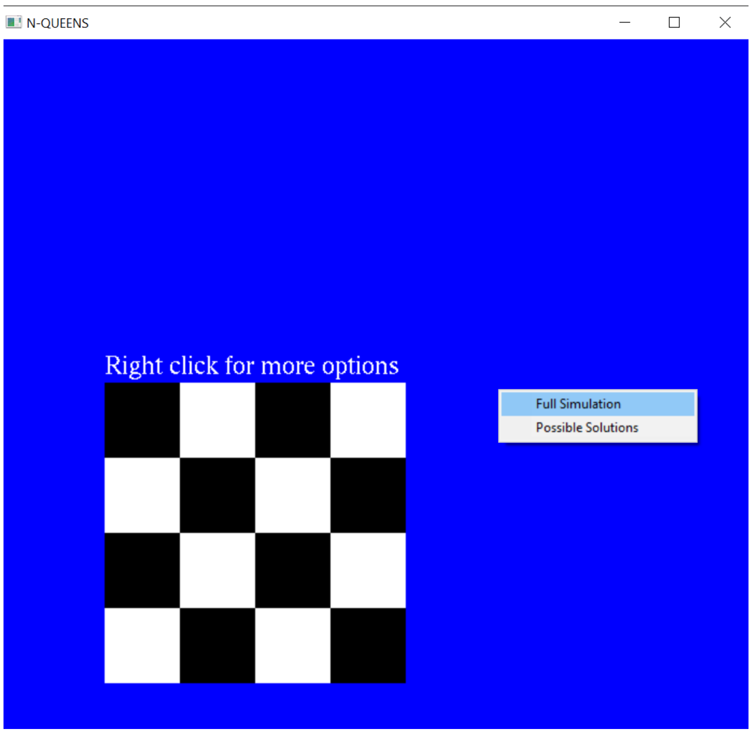


Figure 6.2 Second Screen of the Application, where N x N chessboard is displayed

6.2.1 When Full Simulation is selected:

When full simulation is selected, the display shows the working of the algorithm used to solve the N-Queens problem. The queen tries out various positions. If there are conflicts for that position, the square is marked red and the queen moves on to the next position. If there are no conflicts, the square is marked green and the queen is placed there.

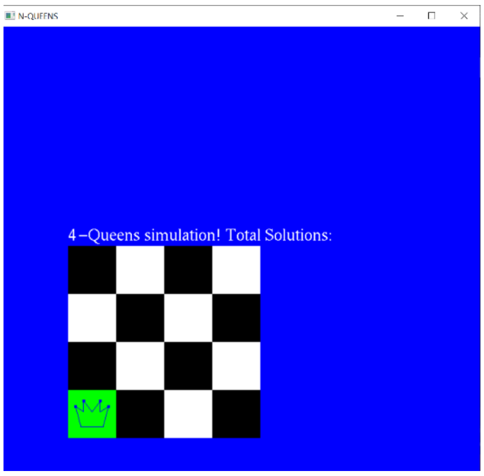


Figure 6.2.1.1 Start of the simulation

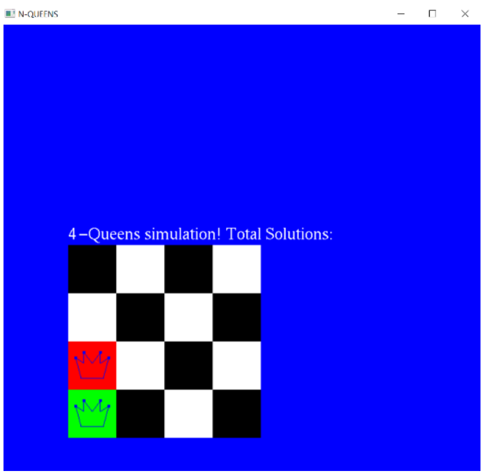


Figure 6.2.1.2 First position tried

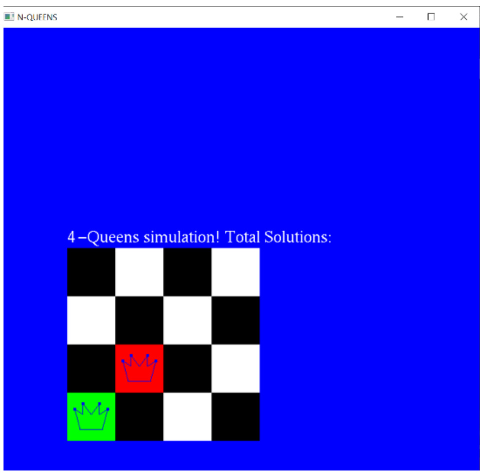


Figure 6.2.1.3 Second position tried

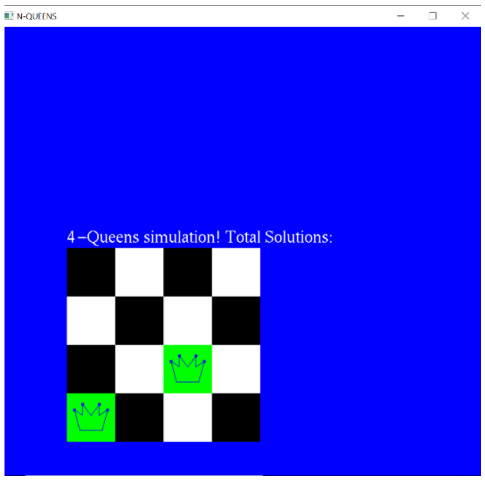


Figure 6.2.1.4 Queen is placed

For each Soliton encountered in the simulation, the count value increments by 1. The simulation runs until the queen has tried all possible places and combinations.

6.2.2 When Possible Solutions is selected:

When possible solutions is selected, the display shows all the possible solutions of the problem one after the other. Every time a solution is displayed the count value is incremented by 1.

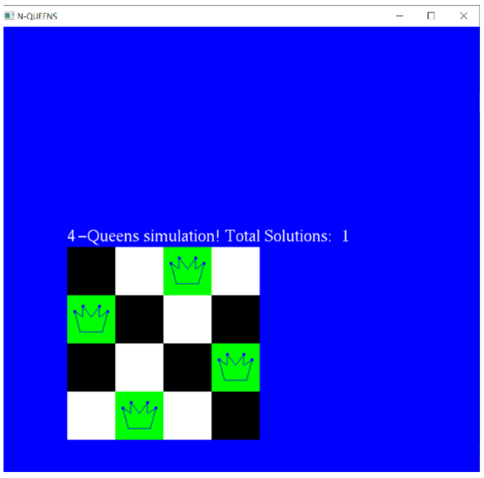


Figure 6.2.2.1 Solution 1

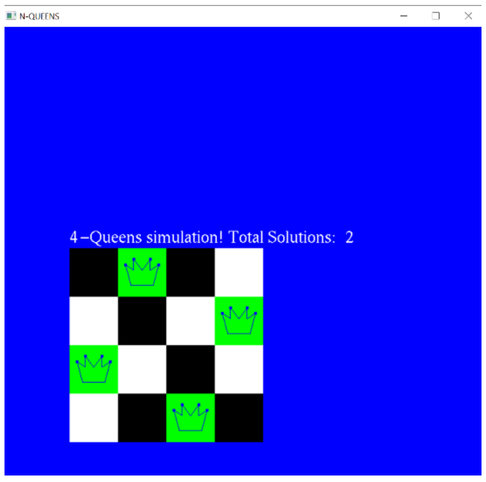


Figure 6.2.2.2 Solution 2

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

N-Queens problem is the most commonly used concept to test out the working of various algorithms in the software industry. This application visualizes this concept, which makes it easier to understand it and perform experiment on it. It plays a vital role to understand the working and efficiency of many algorithms. During the course of building this application various OpenGL API Functions and variables were utilized that made it easier to build the application.

In the future this application can be further enhanced by using different algorithms to solve the problem, not just backtracking algorithm. The model can also be implemented in 3D.

BIBLIOGRAPHY

[1] Edward Angel: Interactive Computer Graphics: A Top Down Approach 5th Edition, Addison – Wesley, 2008

[2] Donald Hearn and Pauline Baker: OpenGL, 3rd Edition, Pearson Education, 2004

[3] Wikipedia: N Queens - https://en.wikipedia.org/wiki/Eight_queens_puzzle

[4] Wikipedia: Computer Graphics – https://en.wikipedia.org/wiki/ComputerGraphics

[5] GeeksForGeeks: N-Queens using backtracking - https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/