VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAVI-590018

Computer Graphics Laboratory
A Mini-Project Report
On

“Story Simulation – Thirsty Crow”

Submitted in partial fulfillment of the requirements for the 6th semester of ***Bachelor of Engineering in Computer Science and Engineering*** *of Visvesvaraya Technological University, Belagavi*

Submitted by:

Rachanaa Raghuthama	**1RN18CS078**
Ridha P	**1RN18CS086**
Varnika Bagaria	**1RN18CS115**

Under the Guidance of:

Mrs. S Mamatha Jajur
Asst. Professor
Dept. of CSE

Mrs. Swathi G
Asst. Professor
Dept. of CSE

Department of Computer Science and Engineering

RNS Institute of Technology

Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560 098

2020-2021

RNS Institute of Technology
Channasandra, Dr.Vishnuvardhan Road,
Bengaluru-560 098

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that the mini project work entitled **"Story Simulation – Thirsty Crow"** has been successfully carried out by **Rachanaa Raghuthama** bearing USN **1RN18CS078, Ridha P** bearing USN **1RN18CS086** and **Varnika Bagaria** bearing USN **1RN18CS115,** bonafide students of **RNS Institute of Technology** in partial fulfillment of the requirements for the **6th semester** of **Bachelor of Engineering** in **Computer Science and Engineering** of **Visvesvaraya Technological University**, Belagavi, during the academic year 2020-21. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the Computer Graphics laboratory requirements of 6th semester BE, CSE.

Mrs. S Mamatha Jajur	**Dr. Kiran P**
Asst Professor	**Prof. and Head**
Dept. of CSE	**Dept of CSE**

External Viva:

Name of the Examiners	**Signature with Date:**
1.	
2.	

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

We are grateful to **Dr. M K Venkatesha,** Principal, RNSIT, Bangalore, for his support towards completing our project.

We would like to thank **Dr.Kiran P**, Prof & Head, Department of Computer Science & Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice.

We deeply express our sincere gratitude to our guides **Mrs. S Mamatha Jajur, Asst Prof, Department of CSE, RNSIT, Bangalore** and **Mrs. Swathi G**, **Asst Prof, Department of CSE, RNSIT, Bangalore,** for their able guidance, regular source of encouragement and assistance throughout this project.

We would like to thank all the teaching and non-teaching staff of Department of Computer Science & Engineering RNSIT, Bangalore for their constant support and encouragement.

Date:	**Rachanaa Raghuthama**	**1RN15CS078**
Place: Bengaluru	**Ridha P**	**1RN15CS086**
	Varnika Bagaria	**1RN18CS115**

ABSTRACT

This Project is on “Story Simulation” Computer Graphics using OpenGL Functions. It is a User interactive program where in the User can view the required display by making use of the input devices like Keyboard and Mouse. This project mainly consists of the thirsty crow. Various options such as reset, toggle water level is provided. The user can choose the desired option. It has a very simple and effective user interface.

OpenGL(open graphics library) is a standard specification defining a cross language cross platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives. OpenGL was developed by silicon graphics Inc.(SGI) in 1992 and is widely used in CAD ,virtual reality , scientific visualization , information visualization and flight simulation. It is also used in video games, where it competes with direct 3D on Microsoft Windows platforms.

The “Story Simulation – Thirsty Crow” is a graphical mini project which is used to simulate the story where a thirsty crow on a sunny day uses its clever mind to quench its thirst. To implement the Scientific computer simulation of the story with the computational model we are using OpenGL framework, different primitives and various functions available in OpenGL.

CONTENTS

Chapter 1

INTRODUCTION

1.1 OVERVIEW OF COMPUTER GRAPHICS

The term computer graphics has been used in a broad sense to describe almost everything on computers that is not text or sound. Typically, the term *computer graphics* refers to several different things:

- The representation and manipulation of image data by a computer
- The various technologies used to create and manipulate images
- The images so produced, and
- The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Today, computers and computer-generated images touch many aspects of daily life. Computer images are found on television, in newspapers, for example in weather reports, in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, thesis, and other presentation material. Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 4D, 7D, and animated graphics.

As technology has improved, 3D computer graphics have become more common. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic component.

1.1.1 HISTORY

In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software.[4] Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location. Also in 1961 another student at MIT, Steve Russell, created the first video game, E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963.

During 1970s, the first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

In the 1980s, artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use graphical user interfaces (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimedia technology.

3D graphics became more popular in the 1990s in gaming, multimedia and animation. In 1996, Quake, one of the first fully 3D games, was released. In 1995, Toy Story, the first full-length computer-generated animation film, was released in cinemas worldwide. Since then, computer graphics have only become more detailed and realistic, due to more powerful graphics hardware and 3D modeling software.

1.1.2 APPLICATIONS

The applications of computer graphics can be divided into four major areas:

- **Display of information:** Computer graphics has enabled architects, researchers and designers to pictorially interpret the vast quantity of data. Cartographers have developed maps to display the celestial and geographical information. Medical imaging technologies like Computerized Tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound, Positron Emission Tomography (PET) and many others make use of computer graphics.
- **Design:** Professions such as engineering and architecture are concerned with design. They start with a set of specification; seek cost-effective solutions that satisfy the specification. Designing is an iterative process. Designer generates a possible design, tests it and then uses the results as the basis for exploring other solutions. The use of interactive graphical tools in Computer Aided Design (CAD) pervades the fields including architecture, mechanical engineering, and the design of very-large-scale integrated (VLSI) circuits and creation of characters for animation.
- **Simulation and animation:** Once the graphics system evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators. Graphical flight simulators have proved to increase the safety and to reduce the training expenses. The field of virtual reality (VR) has opened many new horizons. A human viewer can be equipped with a display headset that allow him/her to see the images with left eye and right eye which gives the effect of stereoscopic vision. This has further led to motion pictures and interactive video games.
- **User interfaces:** Computer graphics has led to the creation of graphical user interfaces (GUI) using which even naive users are able to interact with a computer. Interaction with the computer has been dominated by a visual paradigm that includes windows, icons, menus and a pointing device such as mouse. Millions of people are internet users; they access the internet through the graphical network browsers such as Microsoft internet explorer and Mozilla Firefox.

1.2 DESCRIPTION OF PROJECT

- This project is the creative implementation of 'Story Simulation – Thirsty Crow' using a set of OpenGL functions. This consists of the following scenes:
  - The first scene is of a thirsty crow flying on a sunny day.
  - This will be followed by him finding a pot near the house which has a very low level of water, and him striking upon an idea.
  - This will be followed by him picking up some stones. This will be concluded with moral of story
- User interaction is provided by allowing the user to go to the next scene as well as picking up stones.
- Water level in the pot can be seen by using the right click button.

The project was organized in a systematic way. First, we analyzed what are the basic features to be included in the project to make it acceptable. As it is a graphics-oriented project, we made the sketches prior, so as to have an idea about how our output must look like. After all these, the source code was formulated as a paper work. All the required software were downloaded. Finally, the project was successfully implemented

1.3 OVERVIEW TO OpenGL

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation.

OpenGL provides a set of commands to render a three-dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL-applications without licensing. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation. Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

1.4 OpenGL LIBRARIES

Computer Graphics are created using OpenGL, which became a widely accepted standard software system for developing graphics applications. As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL stands for 'open graphics library' graphics library is a collection of API's (Applications Programming Interface). Graphics library functions are:

1. **GL library** (OpenGL in windows) – Main functions for windows.
2. **GLU** (OpenGL utility library) - Creating and viewing objects.
3. **GLUT** (OpenGL utility toolkit)- Functions that help in creating interface of windows

OpenGL draws *primitives*—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. These libraries are included in the application program using preprocessor directives

#include <GL/glut.h>

OpenGL User Interface Library (**GLUI**) is a C++ user interface library based on the OpenGL Utility Toolkit (GLUT) which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window and operating system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management.
The **OpenGL Utility Library** (**GLU**) is a computer graphics library. It consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides. It is usually distributed with the base OpenGL package.

Chapter 2

REQUIREMENT ANALYSIS

2.1 HARDWARE REQUIREMENTS

The Hardware Requirements are very minimal and the program can be run on most of the machines.

Processor: Intel Core i3 processor

Processor Speed: 1.70 GHz

RAM: 4 GB

Storage Space: 40 GB

Monitor Resolution: 1024x768 or 1336x768 or 1280x1024

2.2 SOFTWARE REQUIREMENTS

Operating System: Windows

IDE: Microsoft Visual Studio with C++ (version 6)

OpenGL libraries, Header Files which includes GL/glut.h, Object File Libraries, glu32.lib,opengl32.lib,glut32.lib,DLLfiles,glu32.dll,glut32.dll,opengl32.dll.

Chapter 3

SYSTEM DESIGN

3.1 MODULES

The description of all the functions used in the program is given below:

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

This function requests a display with the properties in 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 glutInitWindowPosition (int x, int y);**

This specifies the initial position of top-left corner of the windows in pixels.

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

This function specifies the initial height and width of the window in pixels.

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

This function creates a window on the 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 glutDisplayFunc (void (*func) (void));**

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

- **void glClearColor(GLclampfr,GLclampf g, GLclampfb,GLclampf a);**

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

- **void glBlindTexture(GLenum target, GLuint texture);**

Allows to create or use a named texture

- **void glTextCord2f(GLfloat s, GLfloat r);**

Specifies texture coordinates in one, two, three, or four dimensions.A call to glTexCoord2 sets them to s t 0 1.

- **void glutSwapBuffers(void);**

Performs a buffer swap on the layer in use for the current window.

- **void glClear(GLbitfield mask);**

It clear buffers to present values. The value of mask is determined by the bitwise OR of options GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT

- **void glutTimerFunc(unsigned int msecs,void (*func)(int value), value)**

glutTimerFunc registers the timer callback func to be triggered in at least milliseconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc

- **void glutBitmapCharacter(void *font, int character);**

Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font.

- **void glutIdleFunc(void (*func)(void));**

Sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received

- **glTranslate(GLdouble x, GLdouble y, GLdouble z);**

Produces a translation by x y z . The current matrix is multiplied by this translation matrix, with the product replacing the current matrix.

- **glRotate(GLdouble angle,GLdouble x, GLdouble y, GLdouble z);**

Produces a rotation of angle degrees around the vector x y z . The current matrix is multiplied by a rotation matrix with the product replacing the current matrix.

- **void glRasterPos2f(GLfloat x,GLfloat y)**

It is used to position pixel and bitmap write operations.

- **glTexImage2D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const void * data);**

Used to define texture of image. The arguments describe the parameters of the texture image, such as height, width, width of the border, level-of-detail number and number of color components provided. The last three arguments describe how the image is represented in memory.

- **glTexParameteri(GLenum target, GLenum pname,GLint param);**

Assigns the value or values in params to the texture parameter specified as pname. For glTexParameter, target defines the target texture.

- **glGenTextures(GLsizei n, GLuint * textures);**

It returns texture names in textures. There is no guarantee that the names form a contiguous set of integers; however, it is guaranteed that none of the returned names was in use immediately before the call to glGenTextures.

- **void glFlush(void);**

Empties all the buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine

- **void glClearColor(GLclampfr,GLclampf g, GLclampfb,GLclampf a)**

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

- **void glClear(GLbitfield mask)**

It clears buffers to present values. The value of mask is determined by the bitwise OR of options GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT

- **void glutPostRedisplay ();**

This function requests that the display callback be executed after the current callback returns.

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

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

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

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

- **void glMatrixMode (GLenum mode)**

This function specifies which matrix will be affected by subsequent transformations. Mode can be GL_MODEL_VIEW, GL_PROJECTION, GL_TEXTURE.

- **void glLoadIdentity ()**

This function sets the current transformation matrix to an identity matrix.

- **void gluOrtho2D(GLdoubleleft, GLdouble right, GLdouble bottom, GLdouble top)**

This function defines a two-dimensional viewing rectangle in the plane z=0.

- **void glutMouseFunc (void *f (int button, int state, int x, int y)**

This function registers the mouse callback function f. The callback function returns the button(GLUT_LEFT_BUTTON,GLUT_MIDDLE_BUTTON,GLUT_RIGHT_BUTTON),
the state of the button after the event (GLUT_UP, GLUT_DOWN), and the position of the mouse relative to the top-left corner of the window.

- **void glVertex3f(TYPE xcoordinate, TYPE ycoordinate, TYPE zcoordinate)**

void glVertex3fv(TYPE *coordinates)
This specifies the position of a vertex in 3 dimensions. If v is present, the argument is a pointer to an array containing the coordinates.

- **void glBegin(glEnum mode)**

This function initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES and GL_POLYGON.

- **void glEnd()**

This function terminates a list of vertices.

- **void glutMainLoop()**

This function causes the program to enter an event processing loop. It should be the last statement in main.

- **void glPushMatrix(void);**

Pushes all matrices in the current stack down one level. The current stack is determined by glMatrixMode(). The topmost matrix is copied, so its contents are duplicated in both the top and second-from-the-top matrix. If too many matrices are pushed, an error is generated.
Void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z);
voidglTranslate{fd} (TYPE x, TYPE y, TYPE z);
Void glScale {fd} (TYPE x, TYPE y, TYPE z);

- **void glPopMatrix (void);**

Pops the top matrix off the stack, destroying the contents of the popped matrix. What was the second-from-the-top matrix becomes the top matrix. The current stack is determined byglMatrixMode(). If the stack contains a single matrix, calling glPopMatrix() generates an error.

- **glLoadName(ID) :**

To replace the top of the integer-ID name stack for picking operations

- **glSelectBuffer(size,pickbuff)** : To set up a pick-buffer array

- **glRenderMode(GL_SELECT) :** To activate OpenGL picking operations

- **glInitNames() :** To activate the integer-ID name stack for the picking operations

- **glPushName(ID) :** To place an unsigned interger value on the stack

- **glGet** :** Query function to copy specified state values into an array(requires specification of data type,symbolic name of a state parameter and an array pointer)

- **gluPickMatrix(xPick,yPick,widthPick,heightPick,vpArray) :** To define a pick window within a selected viewport

- **glPopMatrix()** : To destroy the matix on top of the stack and to make the second matrix on the stack become the current matrix

- **glutMouseFunc(mousefcn)** : Specify a mouse callback function that is to be invoked when a mouse button is pressed

- **glutMotionFunc(motionfcn) :** Specify a mouse callback function that is to be invoked when the mouse cursor is moved while a button is pressed

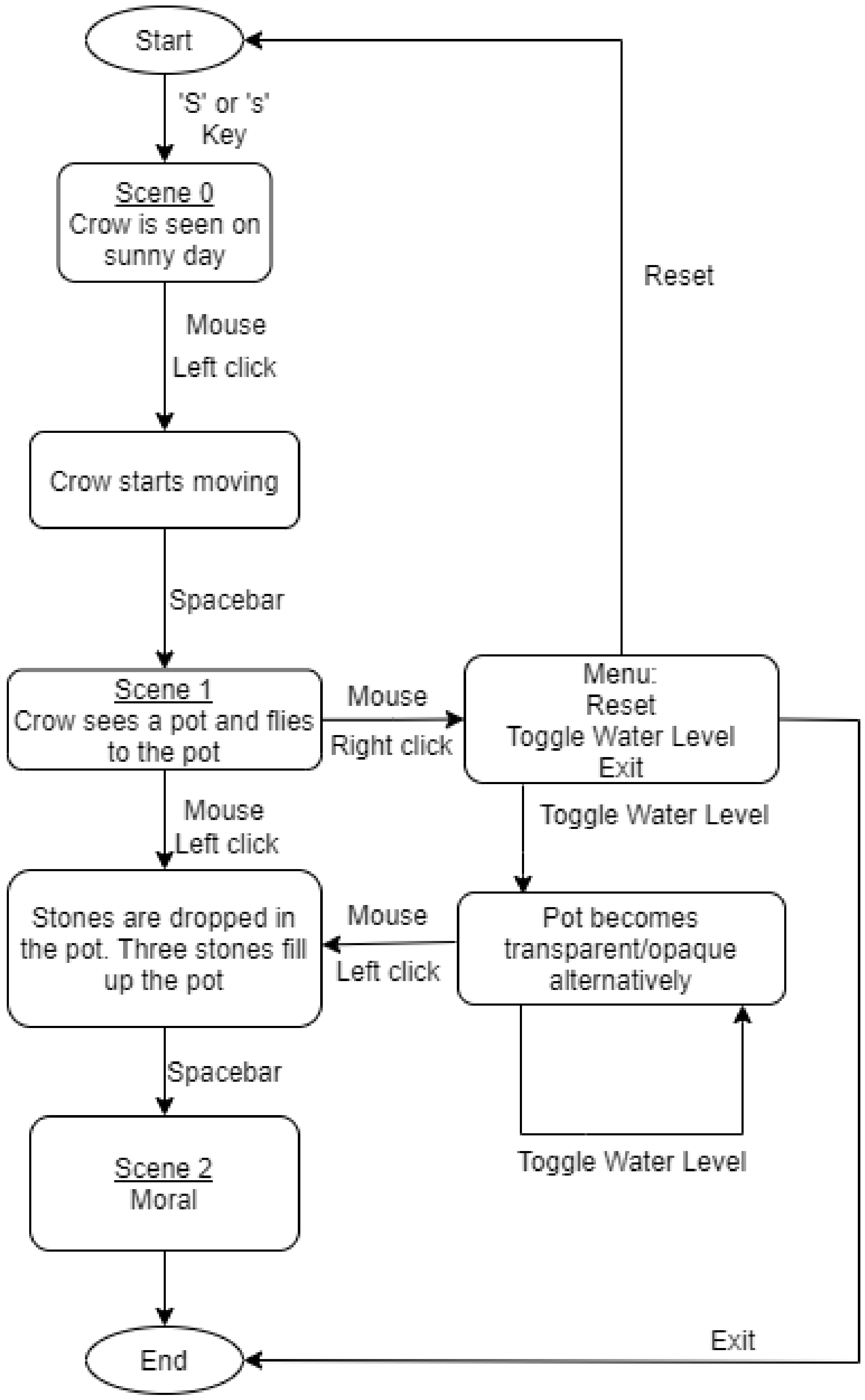
- **glutKeyboardFunc(keyboardfcn)** : Specify a keyboard callback function that is to be invoked when a standard key is pressed

- **glutCreateMenu(menuFunc) :** To create a popup menu and specify the procedure to be invoked when a menu item is selected

- **glutAddMenuEntry(charString,menuItemNumber) :** To list the name and position for each option

- **glutAttachMenu(button)** : To specify a mouse button that is to be used to select a menu option

3.2 PSEUDOCODE/FLOWCHART/FLOW DIAGRAMS



CHAPTER 4

IMPLEMENTATION

line.h

```
#include <GL/glut.h>
class Line
{

private:
    float color[3];

public:
    void setColor(float r, float g, float b)
    {
        color[0] = r;
        color[1] = g;
        color[2] = b;
    }
    void draw(int x1, int y1, int x2, int y2, int depthIndex)
    {
        glColor3fv(color);
        glLineWidth(5);
        glBegin(GL_LINES);
        glVertex3f(x1, y1, depthIndex);
        glVertex3f(x2, y2, depthIndex);
        glEnd();
    }
};
```

ellipse.h

```
#ifndef ELLIPSE_H
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <math.h>
class Ellipse
{
```

```
private:
    float color[3];
    int depthIndex;

public:
    void setColor(float r, float g, float b)
    {
        color[0] = r;
        color[1] = g;
        color[2] = b;
    }
    void draw(int xCenter, int yCenter, int Rx, int Ry, int depthIndex, bool hollow, int
startAngle, int stopAngle)
    {
        glPushMatrix();
        glTranslatef(xCenter, yCenter, 0);
        glColor3fv(color);
        if (hollow)
            glBegin(GL_POINTS);
        else if (!hollow)
            glBegin(GL_POLYGON);
        for (float i = startAngle; i <= stopAngle; i++)
        {
            float x = Rx * cos((i * 3.142) / 180);
            float y = Ry * sin((i * 3.142) / 180);
            glVertex3f(x, y, depthIndex);
        }
        glEnd();
        glPopMatrix();
    }
    void draw(int xCenter, int yCenter, int Rx, int Ry, int depthIndex, bool hollow, int
startAngle, int stopAngle, int lineWidth)
    {
        glPushMatrix();
        glTranslatef(xCenter, yCenter, 0);
        glColor3fv(color);
        glPointSize(lineWidth);
        if (hollow)
            glBegin(GL_POINTS);
        else if (!hollow)
            glBegin(GL_POLYGON);
        for (float i = startAngle; i <= stopAngle; i++)
        {
            float x = Rx * cos((i * 3.142) / 180);
            float y = Ry * sin((i * 3.142) / 180);
            glVertex3f(x, y, depthIndex);
        }
        glEnd();
        glPopMatrix();
    }
};
#endif
```

circle.h

```
#include <GL/glut.h>
#include <math.h>
class Circle
{
private:
    int depthIndex;
    float color[3];

public:
    void setColor(float r, float g, float b)
    {
        color[0] = r;
        color[1] = g;
        color[2] = b;
    }
    void draw(int r, float xpos, float ypos, int depthIndex, bool hollow)
    {
        this->depthIndex = depthIndex;
        glPushMatrix();
        glTranslatef(xpos, ypos, 0);
        glColor3fv(color);
        if (!hollow)
            glBegin(GL_POLYGON);
        else
            glBegin(GL_POINTS);
        for (int i = 0; i <= 360; i++)
        {
            float x = r * cos((i * 3.142) / 180);
            float y = r * sin((i * 3.142) / 180);
            glVertex3f(x, y, depthIndex);
        }
        glEnd();
        glPopMatrix();
    }
};
```

Triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
class Triangle
{
private:
    float color[3];

public:
    void setColor(float r, float g, float b)
    {
        color[0] = r;
        color[1] = g;
        color[2] = b;
    }
    void draw(int x1, int y1, int x2, int y2, int x3, int y3, int depthIndex)
    {
        glColor3fv(color);
        glBegin(GL_TRIANGLES);
        glVertex3f(x1, y1, depthIndex);
        glVertex3f(x2, y2, depthIndex);
        glVertex3f(x3, y3, depthIndex);
        glEnd();
    }
};
#endif
```

sector.h

```
#include <GL/glut.h>
#include <math.h>
#include <iostream>
#include <string>
class Sector
{
private:
    float color[3];
public:
    void setColor(float r, float g, float b)
    {
        color[0] = r;
        color[1] = g;
        color[2] = b;
    }
    void draw(int r, int startAngle, int stopAngle, int depthIndex)
    {
        float x = 0;
        float y = 0;
        glPushMatrix();
        glColor3fv(color);
        x = r * cos((startAngle * 3.142) / 180);
        y = r * sin((startAngle * 3.142) / 180);
        float t1x = x;
        float t1y = y;
        glBegin(GL_POLYGON);
        for (float t = startAngle; t <= stopAngle; t++)
        {
            float angle = (t * 3.142) / 180;
            x = r * cos(angle);
            y = r * sin(angle);
            glVertex3f(x, y, depthIndex);
        }
        glEnd();
        glBegin(GL_TRIANGLES);
        glVertex3f(x, y, depthIndex);
        glVertex3f(t1x, t1y, depthIndex);
        glVertex3f(0, 0, depthIndex);
        glEnd();
        glPopMatrix();
    }
    void drawWing(int state, int depthIndex, int xpos, int ypos)
    {
        glPushMatrix();
        glTranslatef(xpos, ypos, 0);
        glRotatef(state, 0, 0, 1);
        draw(400, 45, 135, depthIndex);
        glPopMatrix();
    }
};
//#endif /* SECTOR_H */
#pragma once
```

pot.h

```
#include <GL/glut.h>
#include "ellipse.h"
#include "stone.h"
Ellipse ellipse;
class Pot
{
public:
    void draw(int xpos, int ypos, bool hollow, int stones, Stone stone)
    {
        int lw;
        if (hollow)
            lw = 5;
        else
            lw = 2;
        glPushMatrix();
        glTranslatef(xpos, ypos, 0);
        glLineWidth(lw);
        ellipse.setColor(0, 0, 0);
        ellipse.draw(400, 300, 400, 300, 9, true, 117, 420, lw);
        if (hollow)
        {
            ellipse.setColor(0, 0, 1);

            switch (stones)
            {
            case 0:
                ellipse.draw(400, 300, 400, 300, 9, false, 200, 350, lw);
                break;
            case 1:
                ellipse.draw(400, 300, 400, 300, 9, false, 177, 383, lw);
                stone.draw(400, 70);
                break;
            case 2:
                ellipse.draw(400, 300, 400, 300, 9, false, 117, 420, lw);
                stone.draw(400, 70);
                stone.draw(250, 70);
                break;
            case 3:
                ellipse.draw(400, 300, 400, 300, 9, false, 117, 420, lw);
                glColor3f(0, 0, 1);
                ellipse.draw(400, 980, 200, 50, 7, false, 0, 360, lw);
                glBegin(GL_POLYGON);
                glVertex3f(220, 560, 8);
                glVertex3f(600, 560, 8);
                glVertex3f(540, 900, 8);
                glVertex3f(270, 900, 8);
                glEnd();
                stone.draw(400, 70);
                stone.draw(250, 70);
                stone.draw(550, 70);
                break;
            }
        }
        glColor3f(0, 0, 0);
        glBegin(GL_LINES);
        glVertex3f(220, 560, 8);
        glVertex3f(270, 900, 8);
        glEnd();
        glBegin(GL_LINES);
        glVertex3f(600, 560, 8);
        glVertex3f(540, 900, 8);
        glEnd();
        ellipse.setColor(0, 0, 0);
        ellipse.draw(400, 980, 250, 100, 7, true, 0, 360, lw);
        ellipse.draw(400, 980, 200, 50, 7, true, 0, 360, lw);
        glPopMatrix();
        if (!hollow)
        {
            glDisable(GL_TEXTURE_2D);
            glPushMatrix();
            glTranslatef(xpos, ypos, 0);
            glLineWidth(lw);
            ellipse.setColor(0.43, 0.32, 0.24);
            ellipse.draw(400, 300, 400, 300, 9, false, 0, 360, lw);
            glColor3f(0.43, 0.32, 0.24);
            glBegin(GL_POLYGON);
            glVertex3f(220, 560, 8);
            glVertex3f(600, 560, 8);
            glVertex3f(540, 900, 8);
            glVertex3f(270, 900, 8);
            glEnd();
            ellipse.draw(400, 980, 250, 100, 8, false, 0, 360, lw);
            if (stones == 3)
                ellipse.setColor(0, 0, 1);
            else
                ellipse.setColor(0, 0, 0);
            ellipse.draw(400, 980, 200, 50, 7, false, 0, 360, lw);
            glPopMatrix();
        }
    }
```

```
};

//#endif /* POT_H */
#pragma once
```

bird.h

```
#include <GL/glut.h>

#include "line.h"
#include "ellipse.h"
#include "sector.h"
#include "circle.h"
#include "Triangle.h"
#include<string>
using namespace std;
class Bird
{
private:
    void tail(void)
    {
        Line line;
        line.setColor(0.128, 0.128, 0.128);
        line.draw(200, 400, 0, 350, 1);
        line.draw(200, 400, 30, 300, 1);
        line.draw(200, 400, 50, 240, 1);
        line.draw(200, 400, 110, 245, 1);
    }
    void body(void)
    {
        Ellipse ellipse;
        ellipse.setColor(0.128, 0.128, 0.128);
        ellipse.draw(500, 400, 300, 200, 2, false, 0, 360);
    }
    void wing(int flapState)
    {
        Sector sector;
        glPushMatrix();
        glTranslatef(500, 400, 0);
        glRotatef(flapState, 0, 0, 1);
        sector.setColor(0, 0, 0);
        sector.draw(400, 55, 125, 1);
        glPopMatrix();
    }
    void face(void)
```

```
{
    Circle circle;
    circle.setColor(0.128, 0.128, 0.128);
    circle.draw(170, 900, 400, 2, false);
}
void eye(void)
{
    Circle circle;
    circle.setColor(1, 1, 1);
    circle.draw(20, 1000, 400, 1, false);
}
void beak(void)
{
    Triangle triangle;
    triangle.setColor(0, 0, 0);
    triangle.draw(1000, 300, 1000, 500, 1150, 400, 2);
}
void drawtext(float x, float y, string s)
{
    glColor3f(0, 0, 0);
    glRasterPos2f(x, y);
    for (int i = 0; s[i] != '\0'; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, s[i]);
    glFlush();
}
void drawtext(float x, float y, string s, int t)
{
    glColor3f(1, 1, 1);
    glRasterPos2f(x, y);
    for (int i = 0; s[i] != '\0'; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, s[i]);
    glFlush();
}
void legs(void)
{
    Line line;
    line.setColor(0, 0, 0);
    line.draw(600, 250, 500, 100, 2);
    line.draw(500, 100, 550, 60, 2);
    line.draw(500, 250, 400, 100, 2);
    line.draw(400, 100, 460, 10, 2);
    line.draw(550, 60, 620, 30, 1);
    line.draw(550, 60, 530, 15, 1);
    line.draw(460, 10, 510, 0, 1);
}
```

```
public:
    void drawBird(int flapState, int xpos, int ypos)
    {
        glDisable(GL_TEXTURE_2D);
        glPushMatrix();
        glTranslatef(xpos, ypos, 0);
        tail();
        body();
        wing(flapState);
        face();
        eye();
        beak();
        glPopMatrix();
        glEnable(GL_TEXTURE_2D);
    }
    void drawBird(int flapState, int xpos, int ypos,int t)
    {
        glDisable(GL_TEXTURE_2D);
        glPushMatrix();
        glTranslatef(xpos, ypos, 0);
        tail();
        body();
        Ellipse ellipse;
        ellipse.setColor(0, 0, 0);
        ellipse.draw(500, 400, 200, 100, 1, false, 0, 360);
        legs();
        face();
        eye();
        beak();
        glPopMatrix();
    }
    void cloud(int xpos, int ypos, string line1,  string line2)
    {
        glPushMatrix();
        glTranslatef(xpos, ypos, 0);
        Circle circle;
        circle.setColor(1, 1, 1);
        circle.draw(80, 80, 80, 1, false);
        circle.draw(120, 270, 200, 1, false);
        Ellipse ellipse;
        ellipse.setColor(1, 1, 1);
        ellipse.setColor(1, 1, 1);
        ellipse.draw(680, 400, 300, 200, 1, false, 0, 360);
        drawtext(500, 420, line1);
        drawtext(500, 330, line2);
        glPopMatrix();
    }
    void cloud(int xpos, int ypos,  string line1, string line2, int t)
    {
        glPushMatrix();
        glTranslatef(xpos, ypos, 0);
        Circle circle;
        circle.setColor(0, 0, 0);
        circle.draw(80, 80, 80, 1, false);
        circle.draw(120, 270, 200, 1, false);
        Ellipse ellipse;
        ellipse.setColor(0, 0, 0);
        ellipse.draw(680, 400, 300, 200, 1, false, 0, 360);
        drawtext(500, 420, line1, 1);
        drawtext(500, 330, line2, 1);
        glPopMatrix();
    }
};
```

stone.h

```
#ifndef STONE_H
#define STONE_H
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include "ellipse.h"
class Stone
{
public:
    void draw(int xpos, int ypos)
    {
        glDisable(GL_TEXTURE_2D);
        Ellipse ellipse;
        ellipse.setColor(0, 0, 0);
        ellipse.draw(xpos, ypos, 50, 30, 3, false, 0, 360);
    }
};
#endif
```

scene2.h

```
#pragma once
```

main.cpp

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
#include "stone.h"
#include <fstream>
#include<iostream>
#include <string>
#include "bird.h"
#include "pot.h"
#define STB_IMAGE_IMPLEMENTATION
#define SPACEBAR 32
#include "stb_image.h"
#define FLAP_DOWN -180
#define FLAP_UP 0
float i = 0.0, m = 0.0, n = 0.0, o = 0.0, c = 0.0, b = 0.0;
float p = 0.75, q = 0.47, r = 0.14;
float e = 0.90, f = 0.91, g = 0.98;

void drawtext(float, float, string);
std::string getFileContents(std::ifstream&);
void menuS2(void);
using namespace std;
unsigned int bg1, intro, bg2, moral;
/*char line1[100] = "I cannot reach";
char line2[100] = "the water";
char line3[100] = " ";
char line4[100] = " ";*/
string line1 = "I cannot reach";
string line2 = "the water";
string line3 = "";
string line4 = "";
Bird bird;
Pot pot;
Stone stone;

class State
{
public:
    static int flap;
    static int birdXpos;
    static int birdYpos;
    static int scene;
    static bool displayCloudS1;
    static bool displayCloudS2;
    static bool moveBird;
    static bool water;
    static bool yMove;
    static int stones;
    static bool mouthStone;
    static string line;
};
int State::flap = FLAP_DOWN;
int State::birdXpos = 0;
int State::birdYpos = 1400;
int State::scene = -1;
bool State::displayCloudS1 = true;
bool State::displayCloudS2 = true;
bool State::moveBird = false;
bool State::water = false;
bool State::yMove = false;
bool State::mouthStone = false;
int State::stones = 0;
string State::line = "Click to Continue";
void init(void)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 5000, 0, 5000, 0, -500);
    glMatrixMode(GL_MODELVIEW);
    glClearColor(1, 1, 1, 1);
}
void doExit(void)
{
    std::ifstream Reader("ART.txt"); //Open file

    std::string Art = getFileContents(Reader); //Get file

    std::cout << Art << std::endl; //Print it to the screen

    Reader.close(); //Close file

    exit(0);
}
void changeCloud(int value)
{
    if (State::stones == 0)
    {
        line1 = "Let me use";
        line2 = "Some Stones";
        line3 = "Use Right Click Menu to toggle Water Level";
        line4 = "Click the Stone to drop it in Pot";
        glutPostRedisplay();
    }
}
void displayIntro(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_TEXTURE_2D);
    glColor3f(1, 1, 1);
    glBindTexture(GL_TEXTURE_2D, intro);
    glBegin(GL_QUADS);
    glVertex3f(0, 0, 10);
    glTexCoord2f(0, 0);
    glVertex3f(0, 5000, 10);
    glTexCoord2f(0, 1);
    glVertex3f(5000, 5000, 10);
    glTexCoord2f(1, 1);
    glVertex3f(5000, 0, 10);
    glTexCoord2f(1, 0);
    glEnd();
    glFlush();
    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
}
void displayScene1(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    bird.drawBird(State::flap, State::birdXpos, 2000);
    glDisable(GL_TEXTURE_2D);
    if (State::displayCloudS1)
    {
        bird.cloud(3200, 2500, "Its a Sunny day", "I'm Thirsty", 1);
    }

    glColor3f(0, 0, 0);
    drawtext(2100, 430, State::line);
    glEnable(GL_TEXTURE_2D);
    glColor3f(1, 1, 1);
    glBindTexture(GL_TEXTURE_2D, bg1);
    glBegin(GL_QUADS);
    glVertex3f(0, 0, 10);
    glTexCoord2f(0, 0);
    glVertex3f(0, 5000, 10);
    glTexCoord2f(0, 1);
    glVertex3f(5000, 5000, 10);
    glTexCoord2f(1, 1);
    glVertex3f(5000, 0, 10);
    glTexCoord2f(1, 0);
    glEnd();
    glFlush();
    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
}
void displayScene2(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glDisable(GL_TEXTURE_2D);
    pot.draw(3500, 400, State::water, State::stones, stone);
    if (State::displayCloudS2)
        bird.cloud(4000, 2000, line1, line2);
    if (!State::yMove)
        bird.drawBird(FLAP_DOWN, 3200, 1400, 1);
    else if (State::yMove)
        bird.drawBird(State::flap, 3200, State::birdYpos);
    if (State::mouthStone)
        stone.draw(4350, 380 + State::birdYpos);
    stone.draw(4000, 300);
    stone.draw(4200, 200);
    stone.draw(3900, 150);
    stone.draw(4300, 200);
    stone.draw(4500, 100);
    glColor3f(0, 0, 0);
    drawtext(1800, 4800, line3);
    drawtext(2000, 4600, line4);
    drawtext(2100, 4400, State::line);
    glEnable(GL_TEXTURE_2D);
    glColor3f(1, 1, 1);
    glBindTexture(GL_TEXTURE_2D, bg2);
    glBegin(GL_QUADS);
    glutTimerFunc(3000, changeCloud, 0);
    glVertex3f(0, 0, 10);
    glTexCoord2f(0, 0);
    glVertex3f(0, 5000, 10);
    glTexCoord2f(0, 1);
    glVertex3f(5000, 5000, 10);
    glTexCoord2f(1, 1);
    glVertex3f(5000, 0, 10);
    glTexCoord2f(1, 0);
    glEnd();
    glDisable(GL_TEXTURE_2D);

    glutSwapBuffers();
}
void displayMoral(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_TEXTURE_2D);
    glColor3f(1, 1, 1);
    glBindTexture(GL_TEXTURE_2D, moral);
    glBegin(GL_QUADS);
    glVertex3f(0, 0, 10);
    glTexCoord2f(0, 0);
    glVertex3f(0, 5000, 10);
    glTexCoord2f(0, 1);
    glVertex3f(5000, 5000, 10);
    glTexCoord2f(1, 1);
    glVertex3f(5000, 0, 10);
    glTexCoord2f(1, 0);
    glEnd();
    glFlush();
    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
}
void drawtext(float x, float y, string s)
{
    glRasterPos2f(x, y);
    for (int i = 0; s[i] != '\0'; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, s[i]);
}
void idle()
{
    if (State::scene == 0)
    {
        if (State::birdXpos <= 2000)
        {
            State::birdXpos += 30;
            glutPostRedisplay();
        }
        else
        {
            State::line = "Press SPACEBAR to continue";
            State::displayCloudS1 = true;
            glutIdleFunc(NULL);
        }
    }
}
void moveBird(void)
{
    if (State::scene == 0)
    {
        if (State::birdXpos <= 5000)
        {
            State::birdXpos += 30;
            glutPostRedisplay();
        }
        else
        {
            glutIdleFunc(NULL);
            State::scene = 1;
            State::line = "";
            glutDisplayFunc(displayScene2);
            glutPostRedisplay();
            menuS2();
        }
    }
}
void birdUp(void)
{
    if (State::birdYpos <= 1300)
    {
        State::birdYpos += 30;
        glutPostRedisplay();
    }
    else
    {
        glutIdleFunc(NULL);
        State::yMove = false;
        State::mouthStone = false;
        if (State::stones < 3)
        {
            State::stones++;
            line1 = "I Need more";
            line2 = "Stones";
        }
        if (State::stones == 3)
        {
            line1 = "I can now";
            line2 = "Drink Water";
            State::line = "Press SPACEBAR to continue";
        }
        State::displayCloudS2 = true;
        glutPostRedisplay();
    }
}
void birdDown(void)
{
    if (State::birdYpos >= 10 && State::stones < 3)
    {
        State::birdYpos -= 30;
        glutPostRedisplay();
    }
    else
    {
        State::mouthStone = true;
        glutIdleFunc(NULL);
        glutIdleFunc(birdUp);
    }
}
void onClick(int btn, int state, int x, int y)
{
    if (State::scene == 0)
    {
        if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
            glutIdleFunc(idle);
    }
    else if (State::scene == 1)
    {
        if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        {
            {
                State::yMove = true;
                State::displayCloudS2 = false;
                glutIdleFunc(birdDown);
                glutPostRedisplay();
            }
        }
    }
}
void timer(int value)
{
    if (State::flap == FLAP_DOWN)
        State::flap = FLAP_UP;
    else if (State::flap == FLAP_UP)
        State::flap = FLAP_DOWN;
    glutTimerFunc(500, timer, 0);
    glutPostRedisplay();
}
void keyboard(unsigned char key, int x, int y)
{
    if (State::scene == -1)
    {
        if (key == 's' || key == 'S')
        {
            State::scene = 0;
            glutDisplayFunc(displayScene1);
            glutPostRedisplay();
        }
    }
    else if (State::scene == 0)
    {
        if (State::displayCloudS1)
        {
            if (key == SPACEBAR)
            {
                State::displayCloudS1 = false;
                glutIdleFunc(moveBird);
            }
        }
    }
    else if (State::scene == 1)
    {
        if (line1 == "I can now")
        {
            if (key == SPACEBAR)
            {
                State::scene = 2;
                glutDisplayFunc(displayMoral);
                glutPostRedisplay();
            }
        }
    }
    else if (State::scene == 2)
    {

        if (key == 27)
        {
            doExit();
        }
    }
}
void processMenuS2(int option)
```

```
{
    switch (option)
    {
    case 1:
        State::flap = FLAP_DOWN;
        State::birdXpos = 0;
        State::birdYpos = 1400;
        State::scene = -1;
        State::displayCloudS1 = false;
        State::displayCloudS2 = true;
        State::moveBird = false;
        State::water = false;
        State::yMove = false;
        State::mouthStone = false;
        State::stones = 0;
        glutDisplayFunc(displayIntro);
        glutPostRedisplay();
        break;
    case 2:
        if (State::water == false)
            State::water = true;
        else if (State::water == true)
            State::water = false;
        glutPostRedisplay();
        break;
    case 3:
        doExit();
    }
}
void menuS2(void)
{

    int menu;

    // create the menu and
    // tell glut that "processMenuEvents" will
    // handle the events
    menu = glutCreateMenu(processMenuS2);

    //add entries to our menu
    glutAddMenuEntry("Reset", 1);
    if (State::scene == 1)
        glutAddMenuEntry("Toggle Water Level", 2);
    glutAddMenuEntry("Exit", 3);

    // attach the menu to the right button
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void loadBackground(void)
{
    glGenTextures(1, &bg1);
    glBindTexture(GL_TEXTURE_2D, bg1);
    // set the bg1 wrapping/filtering options (on the currently bound bg1 object)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    // load and generate the bg1
    int width, height, nrChannels;
    unsigned char* data = stbi_load("S1BG.psd", &width, &height, &nrChannels, 0);
    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, data);
        //glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load bg1" << std::endl;
    }
    stbi_image_free(data);
}
void loadIntro(void)
{
    glGenTextures(1, &intro);
    glBindTexture(GL_TEXTURE_2D, intro);
    // set the bg1 wrapping/filtering options (on the currently bound bg1 object)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    // load and generate the bg1
    int width, height, nrChannels;
    unsigned char* data = stbi_load("INTRO.psd", &width, &height, &nrChannels, 0);
    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, data);
        //glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load intro" << std::endl;
    }
    stbi_image_free(data);
}
void loadMoral(void)
{
    glGenTextures(1, &moral);
    glBindTexture(GL_TEXTURE_2D, moral);
    // set the bg1 wrapping/filtering options (on the currently bound bg1 object)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    // load and generate the bg1
    int width, height, nrChannels;
    unsigned char* data = stbi_load("MORAL.psd", &width, &height, &nrChannels, 0);
    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, data);
        //glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load intro" << std::endl;
    }
    stbi_image_free(data);
}
void loadBackground2(void)
{
    glGenTextures(1, &bg2);
    glBindTexture(GL_TEXTURE_2D, bg2);
    // set the bg1 wrapping/filtering options (on the currently bound bg1 object)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    // load and generate the bg1
    int width, height, nrChannels;
    unsigned char* data = stbi_load("BACKGROUND.psd", &width, &height,
&nrChannels, 0);
```

```
    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, data);
        //glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load bg1" << std::endl;
    }
    stbi_image_free(data);
}

void draw_pixel(GLint cx, GLint cy)
{

    glBegin(GL_POINTS);
    glVertex2i(cx, cy);
    glEnd();
}

void plotpixels(GLint h, GLint k, GLint x, GLint y)
{
    draw_pixel(x + h, y + k);
    draw_pixel(-x + h, y + k);
    draw_pixel(x + h, -y + k);
    draw_pixel(-x + h, -y + k);
    draw_pixel(y + h, x + k);
    draw_pixel(-y + h, x + k);
    draw_pixel(y + h, -x + k);
    draw_pixel(-y + h, -x + k);
}

void draw_circle(GLint h, GLint k, GLint r)
{
    GLint d = 1 - r, x = 0, y = r;
    while (y > x)
    {
        plotpixels(h, k, x, y);
        if (d < 0)
            d += 2 * x + 3;
        else
        {
            d += 2 * (x - y) + 5;
            --y;
        }
        ++x;
    }
    plotpixels(h, k, x, y);
}


void draw_object()
{
    int l;

    glColor3f(0.0, 0.9, 0.9);
    glBegin(GL_POLYGON);
    glVertex2f(0, 380);
    glVertex2f(0, 700);
    glVertex2f(1100, 700);
    glVertex2f(1100, 380);
    glEnd();
    for (l = 0; l <= 35; l++)
    {
        glColor3f(1.0, 0.9, 0.0);
        draw_circle(100, 625, l);
    }
    for (l = 0; l <= 20; l++)
    {
        glColor3f(1.0, 1.0, 1.0);
        draw_circle(160 + m, 625, l);
    }
    for (l = 0; l <= 35; l++)
    {
        glColor3f(1.0, 1.0, 1.0);
        draw_circle(200 + m, 625, l);
        draw_circle(225 + m, 625, l);
    }
    for (l = 0; l <= 20; l++)
    {
        glColor3f(1.0, 1.0, 1.0);
        draw_circle(265 + m, 625, l);
    }
    for (l = 0; l <= 20; l++)
    {
        glColor3f(1.0, 1.0, 1.0);
        draw_circle(370 + m, 615, l);
    }
    for (l = 0; l <= 35; l++)
    {
        glColor3f(1.0, 1.0, 1.0);
        draw_circle(410 + m, 615, l);
        draw_circle(435 + m, 615, l);
        draw_circle(470 + m, 615, l);
    }
    for (l = 0; l <= 20; l++)
    {
        glColor3f(1.0, 1.0, 1.0);
        draw_circle(500 + m, 615, l);
    }
    glColor3f(0.6, 0.8, 0.196078);
    glBegin(GL_POLYGON);
    glVertex2f(0, 160);
    glVertex2f(0, 380);
    glVertex2f(1100, 380);
    glVertex2f(1100, 160);
    glEnd();
    glColor3f(0.0, 0.3, 0.0);
    glBegin(GL_POLYGON);
    glVertex2f(-600, 0);
    glVertex2f(-600, 185);
    glVertex2f(1100, 185);
    glVertex2f(1100, 0);
    glEnd();
    glColor3f(0.9, 0.2, 0.0);
    glBegin(GL_POLYGON);
    glVertex2f(280, 185);
    glVertex2f(280, 255);
    glVertex2f(295, 255);
    glVertex2f(295, 185);
    glEnd();


    for (l = 0; l <= 30; l++)
    {
        glColor3f(0.0, 0.5, 0.0);
        draw_circle(270, 250, l);
        draw_circle(310, 250, l);
    }

    for (l = 0; l <= 25; l++)
    {
        glColor3f(0.0, 0.5, 0.0);
        draw_circle(280, 290, l);
        draw_circle(300, 290, l);
    }
```

```
for (l = 0; l <= 20; l++)
{
    glColor3f(0.0, 0.5, 0.0);
    draw_circle(290, 315, l);
}
glColor3f(0.9, 0.2, 0.0);
glBegin(GL_POLYGON);
glVertex2f(100, 135);
glVertex2f(100, 285);
glVertex2f(140, 285);
glVertex2f(140, 135);
glEnd();


for (l = 0; l <= 40; l++)
{
    glColor3f(0.0, 0.5, 0.0);
    draw_circle(40, 280, l);
    draw_circle(90, 280, l);
    draw_circle(150, 280, l);
    draw_circle(210, 280, l);
    draw_circle(65, 340, l);
    draw_circle(115, 340, l);
    draw_circle(175, 340, l);

}

for (l = 0; l <= 55; l++)
{
    glColor3f(0.0, 0.5, 0.0);
    draw_circle(115, 360, l);


}
glColor3f(0.35, 0.0, 0.0);
glBegin(GL_TRIANGLES);

glVertex2f(540, 300);
glVertex2f(650, 410);
glVertex2f(760, 300);

glEnd();

//Creating House
glColor3f(p, q, r);
glBegin(GL_POLYGON);
```

```
glVertex2f(550, 150);
glVertex2f(550, 300);
glVertex2f(750, 300);
glVertex2f(750, 150);

glEnd();
glBegin(GL_POLYGON);

glVertex2f(600, 150);
glVertex2f(600, 250);
glVertex2f(700, 225);
glVertex2f(700, 175);

glEnd();
//window
glColor3f(1, 1, 1);
glBegin(GL_POLYGON);

glVertex2f(570, 240);
glVertex2f(570, 270);
glVertex2f(600, 270);
glVertex2f(600, 240);

glEnd();

glColor3f(1, 1, 1);
glBegin(GL_POLYGON);

glVertex2f(700, 240);
glVertex2f(700, 270);
glVertex2f(730, 270);
glVertex2f(730, 240);

glEnd();
glColor3f(0.35, 0.0, 0.0);
glBegin(GL_LINES);

glVertex2f(570, 255);
glVertex2f(600, 255);
glVertex2f(700, 255);
glVertex2f(730, 255);

glVertex2f(715, 240);
glVertex2f(715, 270);
glVertex2f(585, 240);
```

```
    glVertex2f(585, 270);

    glVertex2f(570, 240);
    glVertex2f(570, 270);
    glVertex2f(570, 270);
    glVertex2f(600, 270);
    glVertex2f(600, 270);
    glVertex2f(600, 240);
    glVertex2f(600, 240);
    glVertex2f(570, 240);

    glVertex2f(700, 240);
    glVertex2f(700, 270);
    glVertex2f(700, 270);
    glVertex2f(730, 270);
    glVertex2f(730, 270);
    glVertex2f(730, 240);
    glVertex2f(730, 240);
    glVertex2f(700, 240);
    glEnd();


    //door
    glColor3f(e, f, g);
    glBegin(GL_POLYGON);

    glVertex2f(630, 150);
    glVertex2f(630, 200);
    glVertex2f(670, 200);
    glVertex2f(670, 150);

    glEnd();
    glColor3f(0.35, 0.0, 0.0);
    glBegin(GL_POLYGON);

    glVertex2f(630, 150);
    glVertex2f(630, 200);
    glVertex2f(650, 190);
    glVertex2f(650, 160);

    glEnd();

    glFlush();
 }
 int main(int argc, char** argv)
 {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutCreateWindow("A Smart Crow");
    glutFullScreen();
    glutDisplayFunc(displayIntro);
    glutMouseFunc(onClick);
    glutKeyboardFunc(keyboard);
    glutTimerFunc(1000, timer, 0);
    glEnable(GL_DEPTH_TEST);
    loadBackground();
    loadIntro();
    loadBackground2();
   // glScalef(1.5f, 1.5f, 1.0f);
    //draw_object;
    loadMoral();
    init();
    menuS2();
    glutMainLoop();
}

string getFileContents(std::ifstream& File)
{
    std::string Lines = ""; //All lines

    if (File) //Check if everything is good
    {
        while (File.good())
        {
            std::string TempLine;         //Temp line
            std::getline(File, TempLine); //Get temp line
            TempLine += "\n";             //Add newline character

            Lines += TempLine; //Add newline
        }
        return Lines;
    }
    else //Return error
    {
        return "ERROR File does not exist.";
    }
}
```

CHAPTER 5

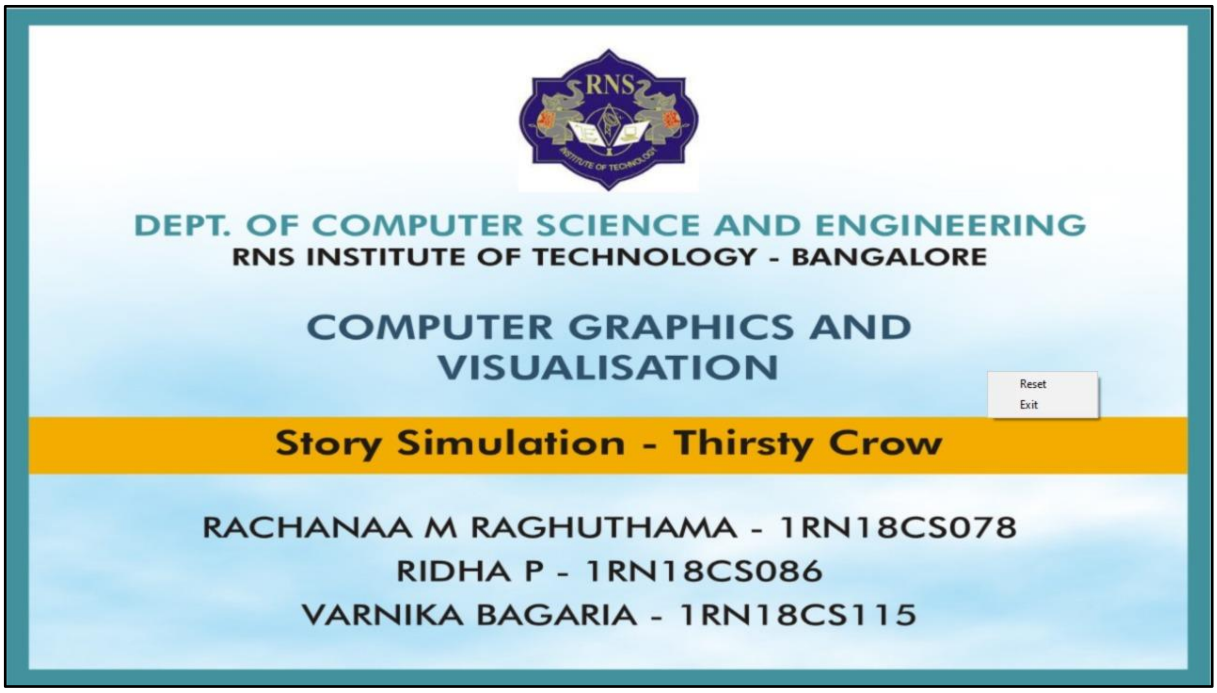
TESTING AND RESULTS

5.1 TEST CASES

In unit testing, the program modules that make up the system are tested individually. Unit testing focuses to locate errors in the working modules that are independent to each other. This enables to detect errors in coding and the logic within the module alone. This testing is also used to ensure the integrity of the data stored. The various routines were checked by passing the inputs and the corresponding output is tested. Test cases used in the project as follows:

Serial No	**Metric**	**Description**	**Observation**
1	**Keyboard Function**	Key s/S to start(Move from Scene 0 to 1). Spacebar to move to next scene. Esc to exit at the end of last scene	Results obtained as expected
2	**Display Function**	Scenes, pot and stones are displayed.	Results obtained as expected
3	**Animation Effect**	Motion of crow. Flapping of wings.	Results obtained as expected

5.2 SCREENSHOTS/SNAPSHOTS

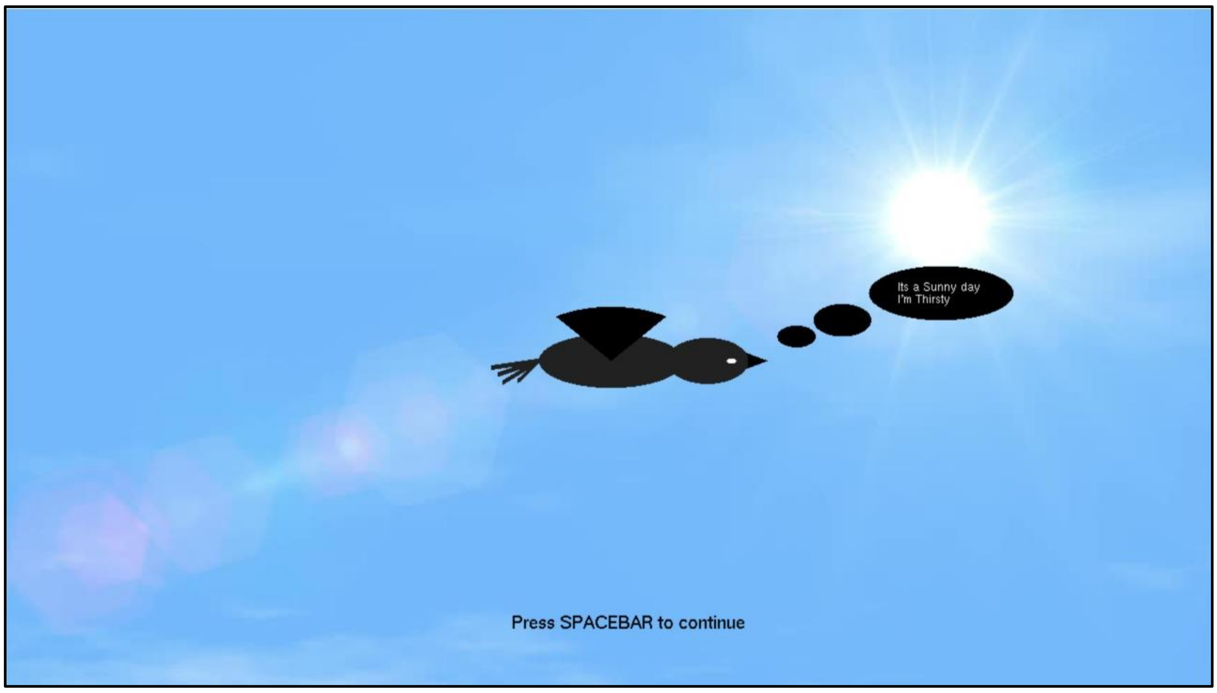

DEPT. OF COMPUTER SCIENCE AND ENGINEERING
RNS INSTITUTE OF TECHNOLOGY - BANGALORE

COMPUTER GRAPHICS AND VISUALISATION

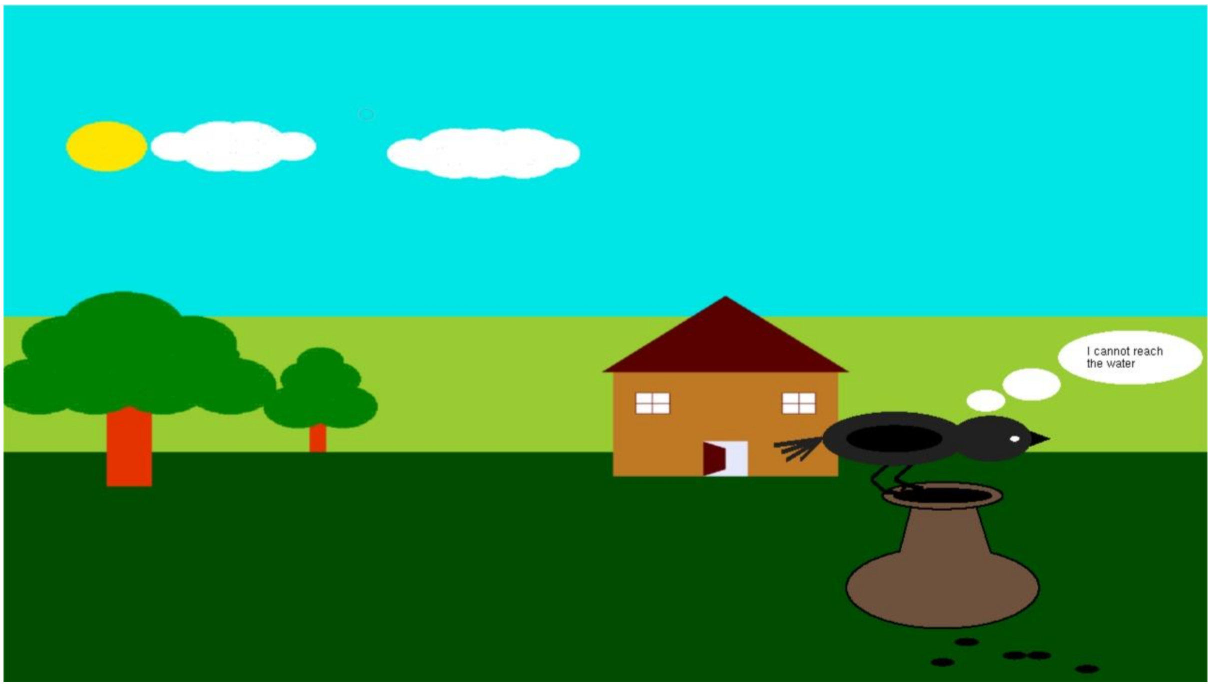
Story Simulation - Thirsty Crow

RACHANAA M RAGHUTHAMA - 1RN18CS078
RIDHA P - 1RN18CS086
VARNIKA BAGARIA - 1RN18CS115

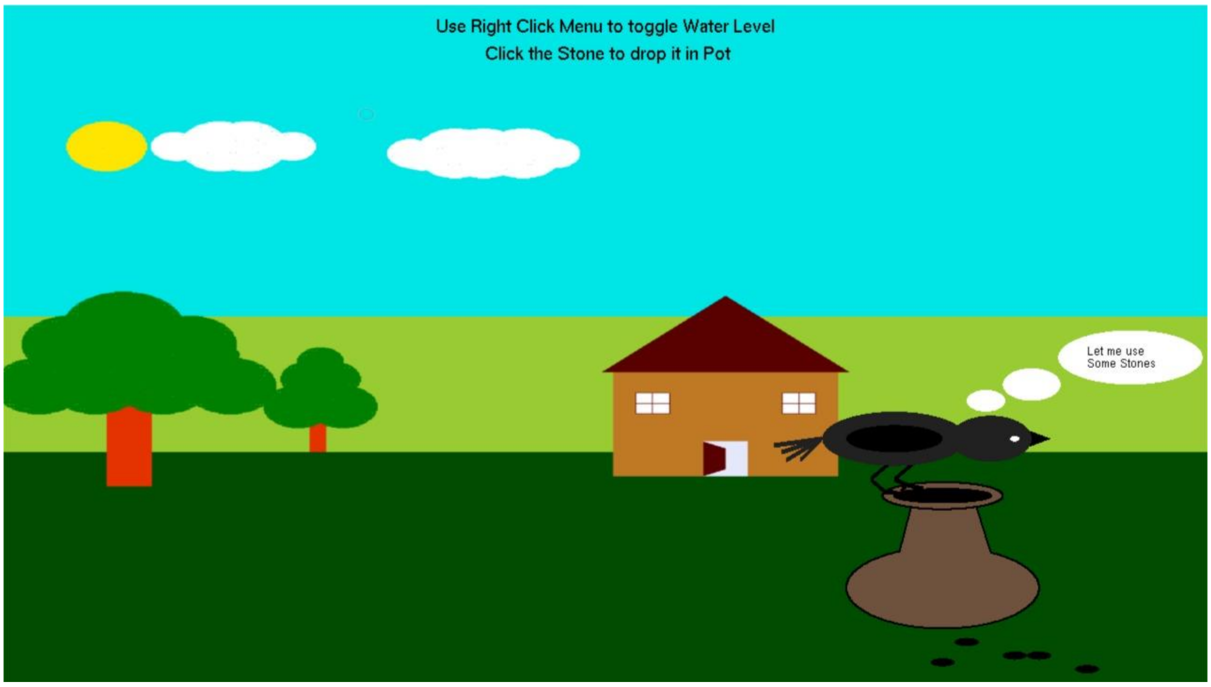
Introduction Page with information on Project Idea and Team Members



Scene 1: A thirsty crow flying on a sunny day. Press SPACEBAR to continue.

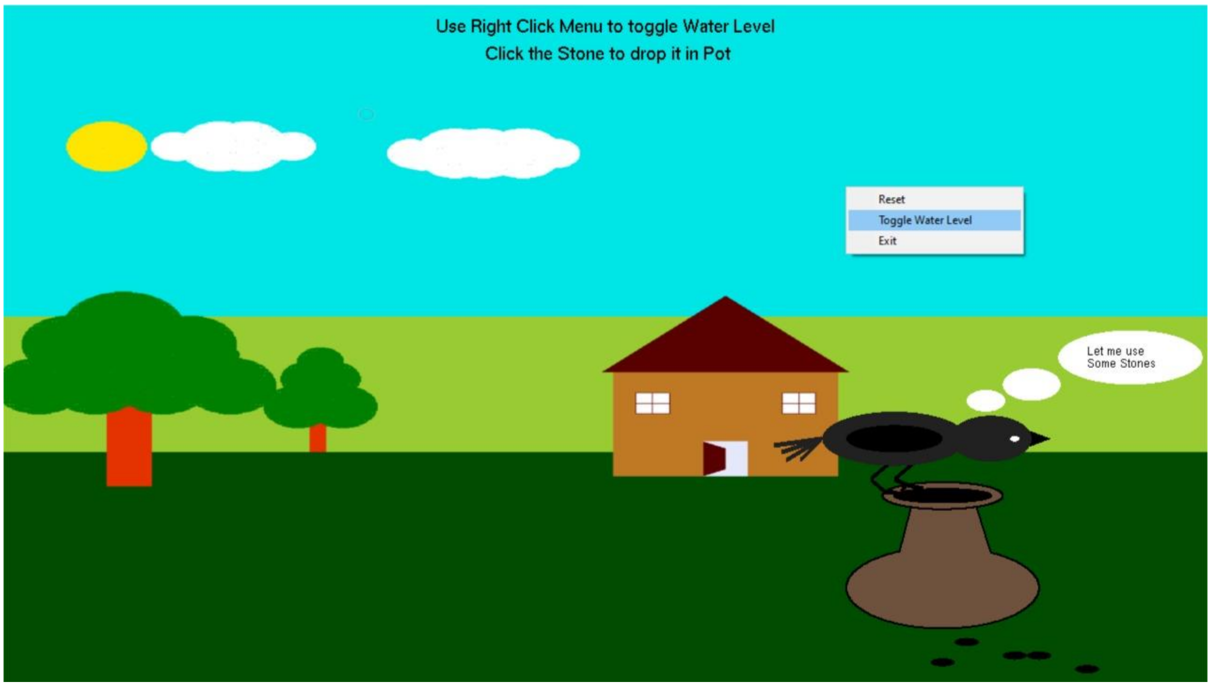


Scene 2: The crow finds a pot with water at the bottom.

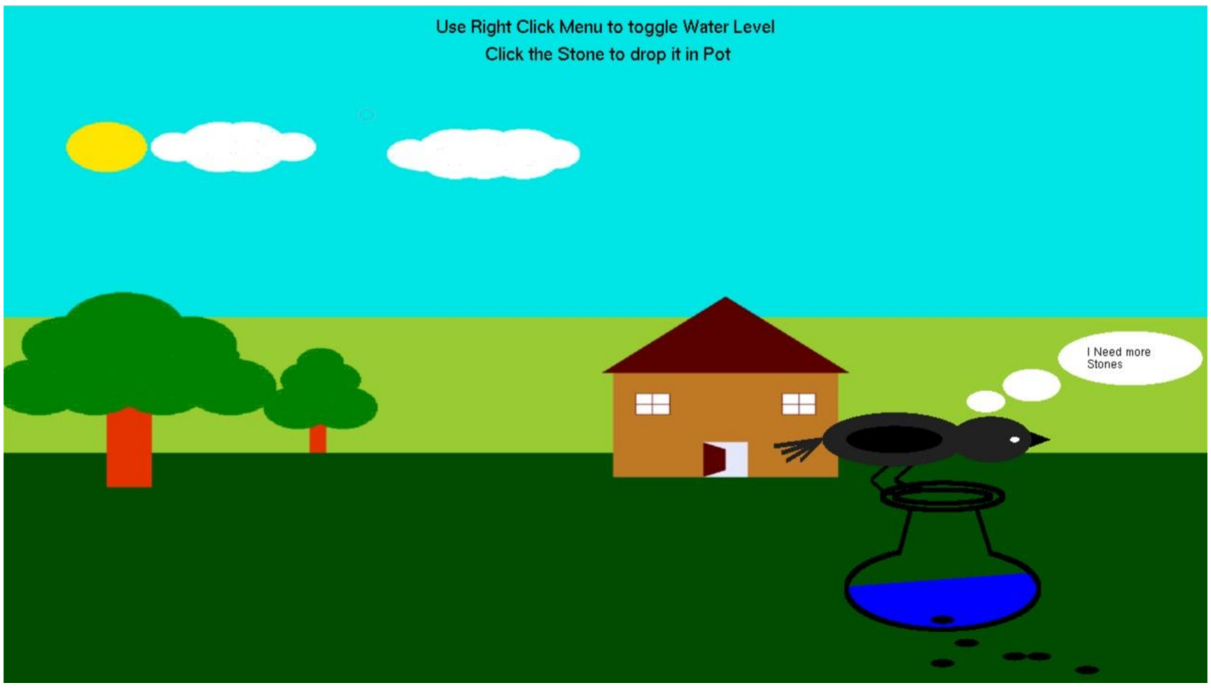

Use Right Click Menu to toggle Water Level

Click the Stone to drop it in Pot

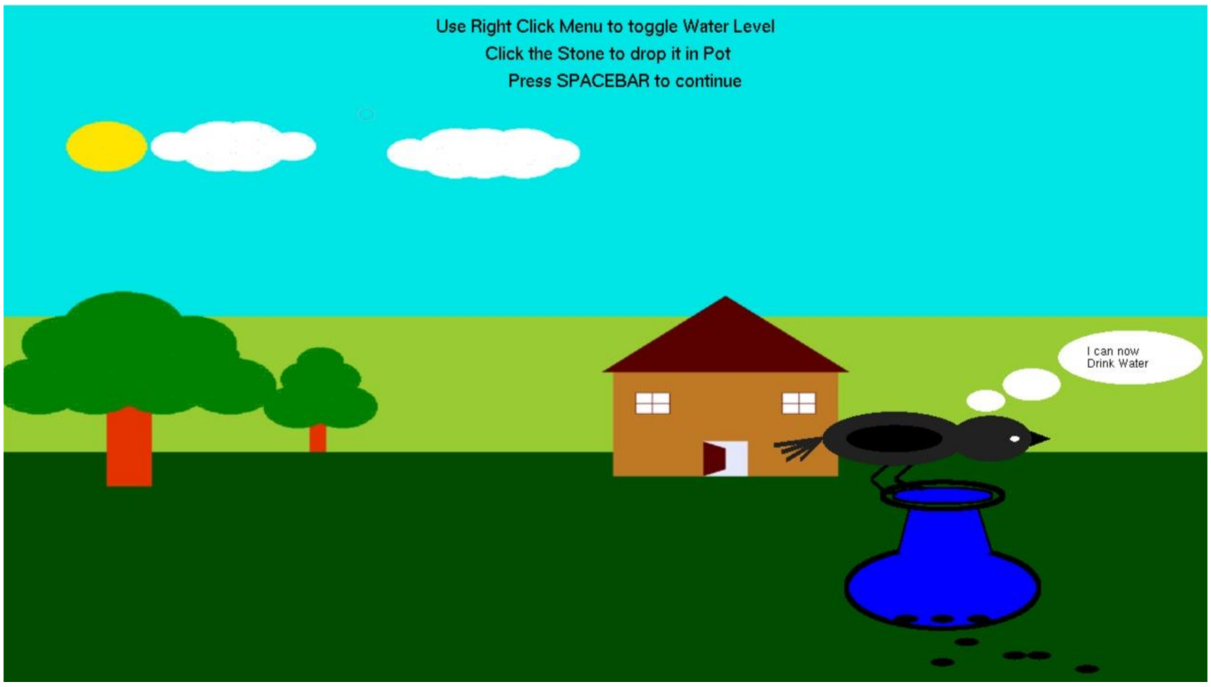
He finds stones on ground and decides to throw it in the pot


Use Right Click Menu to toggle Water Level
Click the Stone to drop it in Pot
Reset
Toggle Water Level
Exit
Let me use Some Stones

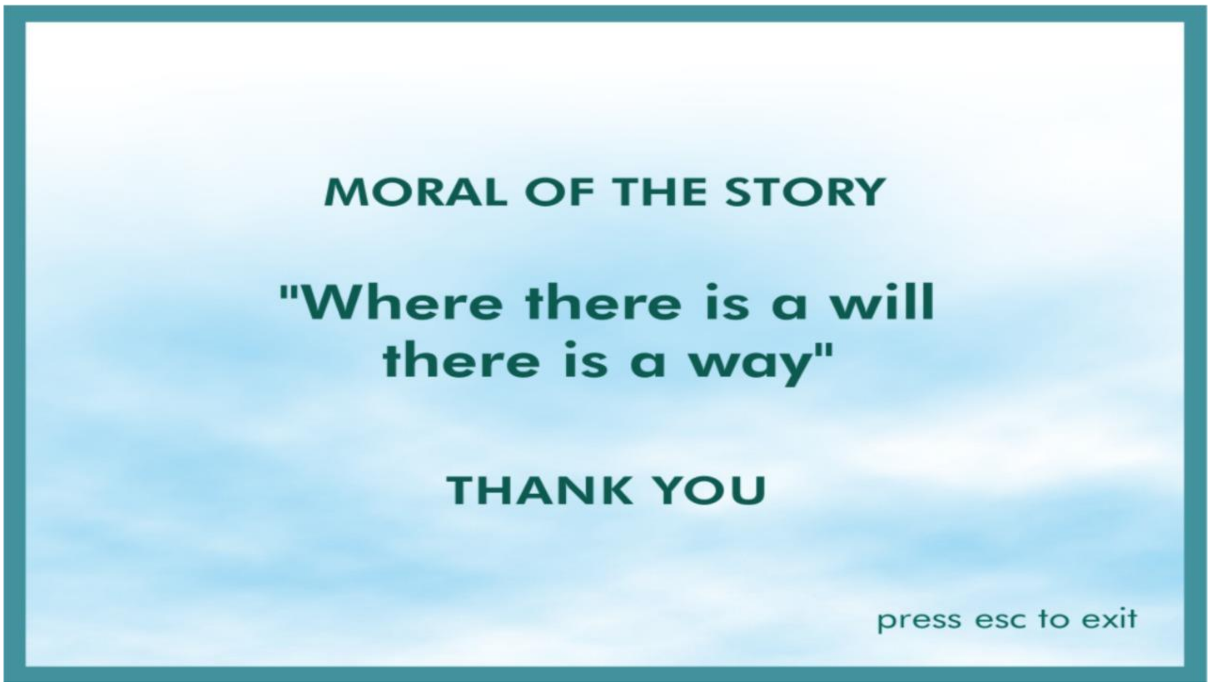
On Right click we get a menu to see the water level in it.


Use Right Click Menu to toggle Water Level
Click the Stone to drop it in Pot
I Need more Stones

Now we are able to see the water level in the pot


Use Right Click Menu to toggle Water Level
Click the Stone to drop it in Pot
Press SPACEBAR to continue
I can now Drink Water

Now the crow can drink water from the pot. Press SPACEBAR to go to next scene.



MORAL OF THE STORY

"Where there is a will there is a way"

THANK YOU

press esc to exit

Scene 3: We can see the moral of story. Press Esc to exit.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

The “Story Simulation – Thirsty Crow” is a graphical mini project which is used to simulate the story where a thirsty crow on a sunny day uses its clever mind to quench its thirst. The project has been tested under Visual Studio , and has been found to provide ease of use and manipulation to the user. It has a very simple and effective user interface.

We found designing and developing this “Story Simulation – Thirsty Crow” as a very interesting and learning experience. It helped us to learn about computer graphics, design of Graphical Use interfaces, interface to the user, user interaction handling and screen management. The graphics editor provides all and more than the features that have been detailed in the university syllabus. Through this project , we also have acquired a much deeper knowledge of the OpenGL constraints and computer graphics.

In future we plan to enhance the project in such a way that:

- We can interact more with the crow .
- The project can be implemented in 3D Space.
- We plan to add a Day and Night scene for which the crow also should simulate accordingly.
- Furthermore, we plan to make the realistic model of the story in which the scene has overflowing water and conditions to tackle the problem.
- A vast amount of future work can be possible by following investigations and strategies.
- More features can be included and can be modified in a more versatile way.
- Additionally, we would like to improve the looks of the project

We thus would like to emphasize the importance of this project to many other perspectives of Technical, mathematical, graphical and software concepts which we were unaware of.

REFERENCES

1. Edward Angel, “Interactive Computer Graphics A Top-Down Approach With OpenGL” 5th Edition, Addison-Wesley, 2008.

2. F.S. Hill, Jr., “Computer Graphics Using OpenGL”, 2nd Edition, Pearson Education, 2001.

3. JamesD.Foley, AndriesVan Dam, StevenK.Feiner, JohnF.Hughes, ”ComputerGraphes”, Second Edition, Addison-Wesley Proffesional, August 14,1995

4. www.opengl.org.

5. www.cs.uiowo.edu/~cwyman/classes/common

6. www.graphicsonline.com

7. www.stackoverflow.com