VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“Jnana Sangama”, Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K. R. Road, V. V. Puram, Bengaluru-560 004

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Computer Graphics Laboratory With Mini Project Report-18CSL67
On

“DINO GAME”

Submitted By

1BI20CS048	**CHIRAYU V GANGADKAR**
1BI20CS055	**DHANUSH C**

for the academic year 2022-23

Under the guidance of

Prof. Bhanushree K J	**Dr. J Girija**
Associate Professor	Head of Dept, CSE

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“Jnana Sangama”, Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K. R. Road, V. V. Puram, Bengaluru-560 004

Department of Computer Science & Engineering

Certificate

This is to certify that the implementation of **Computer Graphics Laboratory With Mini Project (18CSL68)** entitled **“DINO GAME”** has been successfully completed by

1BI20CS048	**CHIRAYU V GANGADKAR**
1BI20CS055	**DHANUSH C**

of VI semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in **Computer Science & Engineering** of the **Visvesvaraya Technological University** during the academic year **2022-2023**.

Lab In charges:

Prof. Mahalakshmi C V	**Prof. Mamatha V**	**Dr. Girija J.**
Assistant Professor	Assistant Professor	Professor and Head
Dept. of CSE, BIT	Dept. of CSE, BIT	Dept. of CSE, BIT

Examiners: 1) 2)

ACKNOWLEDGEMENT

The knowledge & satisfaction that accompany the successful completion of any task would be incomplete without mention of people who made it possible, whose guidance and encouragement crowned my effort with success. I would like to thank all and acknowledge the help I have received to carry out this Mini Project.

I would like to convey my sincere thanks to **Dr. M. U. Aswath**, Principal, BIT and **Dr.Girija J .**, HOD, Department of CS&E, BIT for being kind enough to provide the necessary support to carry out the mini project.

I am most humbled to mention the enthusiastic influence provided by the lab in-charges **Prof. Mamatha V** and **Prof. Mahalakshmi C V,** on the project for their ideas, time to time suggestions for being a constant guide and co-operation showed during the venture and making this project a great success.

I would also take this opportunity to thank my friends and family for their constant support and help. I'm very much pleasured to express my sincere gratitude to the friendly co-operation showed by all the **staff members** of Computer Science Department, BIT.

CHIRAYU V GANGADKAR
1BI20CS048
DHANUSH C
1BI20CS055

Table of contents

CHAPTER 1
INTRODUCTION

1.1 Computer Graphics

Computer graphics is an art of drawing pictures, lines, charts, using computers with the help of programming. Computer graphics is made up of number of pixels. Pixel is the smallest graphical picture or unit represented on the computer screen. Basically, there are 2 types of computer graphics namely,

Interactive Computer Graphics involves a two-way communication between computer and user. The observer is given some control over the image by providing him with an input device. This helps him to signal his request to the computer.

Non-Interactive Computer Graphics otherwise known as passive computer graphics it is the computer graphics in which user does not have any kind of control over the image. Image is merely the product of static stored program and will work according to the instructions given in the program linearly. The image is totally under the control of program instructions not under the user. Example: screen savers.

1.2 Applications of Computer Graphics

Scientific Visualization

Scientific visualization is a branch of science, concerned with the visualization of three-dimensional phenomena, such as architectural, meteorological, medical, biological systems.

Graphic Design

The term graphic design can refer to a number of artistic and professional disciplines which focus on visual communication and presentation

Computer-aided Design

Computer-aided design (CAD) is the use of computer technology for the design of objects, real or virtual. The design of geometric models for object shapes, in particular, is often called computer-aided geometric design (CAGD). The manufacturing process is tied in to the computer description of the designed objects so that the fabrication of a product can be automated using methods that are referred to as CAM, computer-aided manufacturing.

Web Design

Web design is the skill of designing presentations of content usually hypertext or hypermedia that is delivered to an end-user through the World Wide Web, by way of a Web browser.

Digital Art

Digital art most commonly refers to art created on a computer in digital form.

Video Games

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a raster display device.

Virtual Reality

Virtual reality (VR) is a technology which allows a user to interact with a computer simulated environment. The simulated environment can be similar to the real world. This allows the designer to explore various positions of an object. Animations in virtual reality environments are used to train heavy equipment operators or to analyse the effectiveness of various cabin configurations and control placements.

Computer-aided Design

Computer-aided design (CAD) is the use of computer technology for the design of objects, real or virtual. The design of geometric models for object shapes, in particular, is often called computer-aided geometric design (CAGD). The manufacturing process is tied in to the computer description of the designed objects so that the fabrication of a product can be automated using methods that are referred to as CAM, computer-aided manufacturing.

Web Design

Web design is the skill of designing presentations of content usually hypertext or hypermedia that is delivered to an end-user through the World Wide Web, by way of a Web browser.

Digital Art

Digital art most commonly refers to art created on a computer in digital form.

Video Games

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a raster display device.

Virtual Reality

Virtual reality (VR) is a technology which allows a user to interact with a computer simulated environment. The simulated environment can be similar to the real world. This allows the designer to explore various positions of an object. Animations in virtual reality environments are used to train heavy equipment operators or to analyse the effectiveness of various cabin configurations and control placements.

1.3 OpenGL

OpenGL has become a widely accepted standard for developing graphics applications. Most of our applications will be designed to access OpenGL directly through functions in the three libraries. Functions in main GL libraries have names that begin with the letters gl and are stored in a library usually referred to as GL.

The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All function in GLU can be created from the core GL library. The GLU library is available in all OpenGL implementations. Functions in the GLU library starts with the letters glu.

The third is the OpenGL Utility Toolkit (GLUT). It provides the minimum functionality that should be formulated in modern windowing systems.

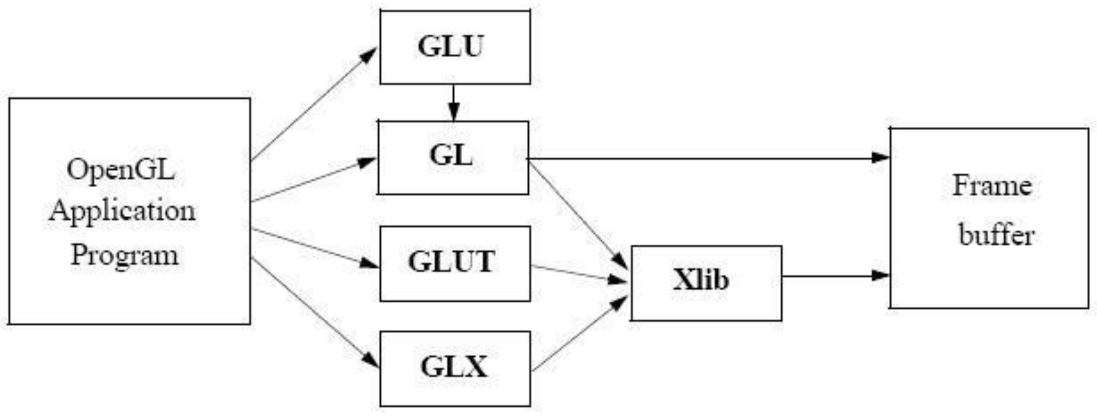


Figure 1.1 Basic block diagram of OpenGL

1.4 Problem Statement

“To implement a simple 2D game called "Dino Game" using OpenGL and GLUT library in C. The game will feature a dinosaur character that needs to jump over cacti obstacles to score points. The objective of the game is to achieve the highest score possible by avoiding collisions with the cacti..”

1.5 Objectives of the Project

- Implement smooth and responsive controls for the dinosaur character, allowing players to make it jump over the cacti.
- Generate random cacti obstacles at varying positions on the screen and make them move towards the dinosaur.
- Detect collisions between the dinosaur and the cacti to determine if the game should end.
- Track and display the player's score based on the number of cacti successfully avoided.
- Store and display the highest score achieved by the player.
- Provide an enjoyable and challenging gaming experience that encourages players to achieve higher scores and try again.

1.6 Organisation of the Project

The project was organised in a systematic way. First we analysed 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 like 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 successful implementation of the project.

Chapter -2

SYSTEM SPECIFICATION

2.1 Hardware Requirements

- Main Processor : PENTIUM III
- Processor Speed: 800 MHz
- RAM Size : 128 MB DDR
- Keyboard : Standard qwerty serial or PS/2 keyboard
- Mouse : Standard serial or PS/2 mouse
- Compatibility : AT/T Compatible
- Cache memory : 256 KB
- Diskette drive : 1,44MB,3.5 inches

2.2 Software Requirements

- Operating System: Windows 10 or Linux (Fedora) or macOS
- Hypervisor used : Docker
- Compiler used : g++
- Language used : C++ language
- Editor : Visual Studio Code
- Toolkit : GLUT Toolkit

Chapter 3

DESIGN

3.1 Flow Diagram

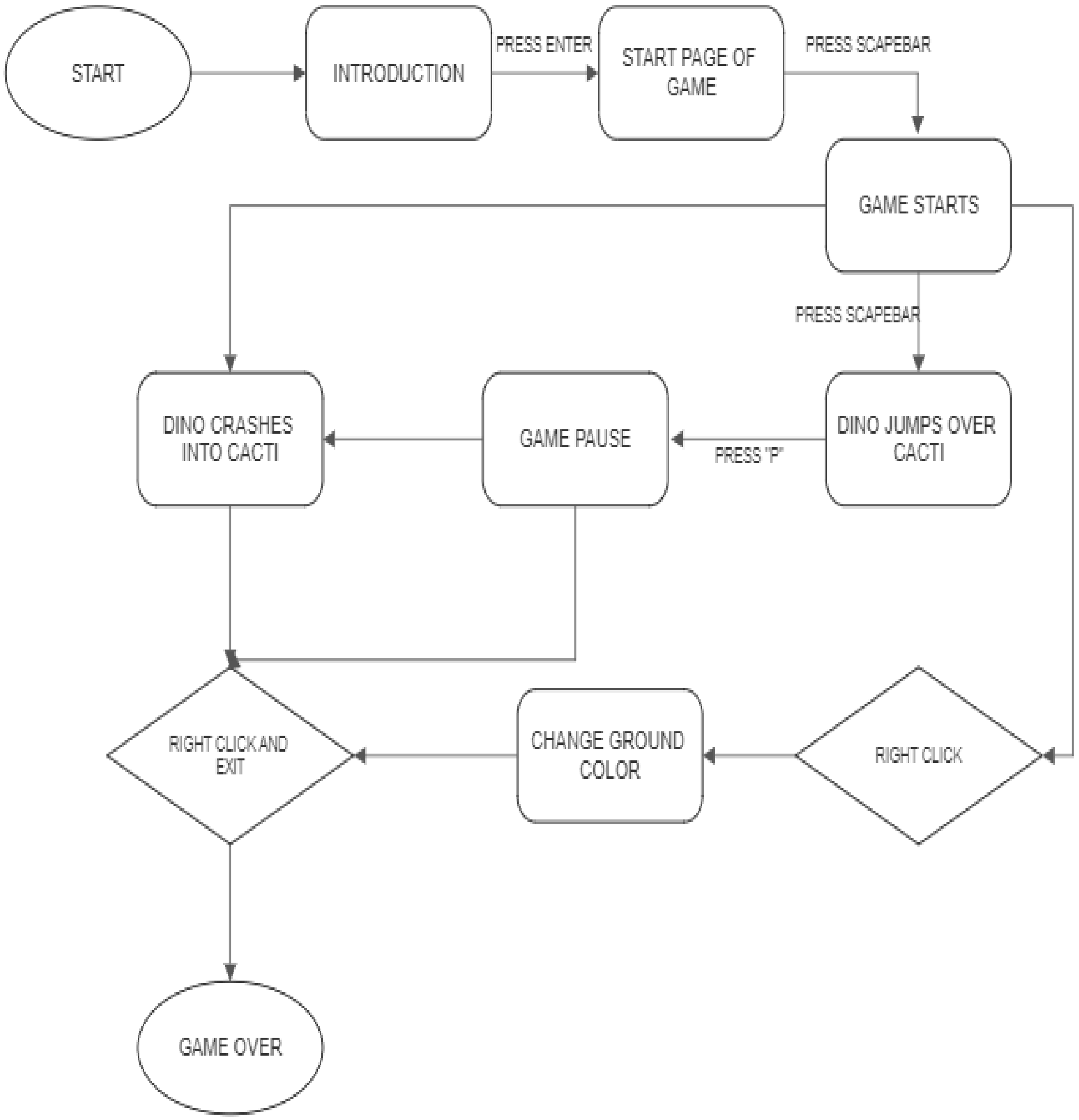


Figure 3.1 : Flow diagram of scene Game Play

3.2 Description of Flow Diagram

The description of the flow diagram is as follows:

Step 1: Start

Step 2: The User is presented with an introduction page and press ENTER to proceed to the next screen.

Step 3: The User will view the start screen of the game. On pressing SPACEBAR the game starts and the dino starts running.

Step 4: When the Dino encounter a cacti the user has to press SPACEBAR inorder to jump and avoid the cacti.

Step 5: If the Dino carshes into the cacti the user can press SPACEBAR to continue to play another round of the Game. Each rounds score is recorded and the hiscore is stored.

Step 6: The User has an option to pause the game at any poit by pressing P button, and can continue by pressing the same button.

Step 7: The User can also change the color of the ground by RIGHT CLICKING and choosing the preferred color.

Step 8: If the user wishes to Exit he/she can do so by RIGHT CLICKING the mouse and pressing the exit button.

Chapter 4

IMPLEMENTATION

4.1 Built in Functions

1. glutInit()

glutInit is used to initialize the GLUT library.

Usage: void glutInit (int *argc, char **argv);

Description: glutInit will initialize the GLUT library and negotiate a session with the window system.

2. glutInitDisplayMode()

glutInitDisplayMode sets the initial display mode.

Usage: void glutInitDisplayMode (unsigned int mode);

Mode-Display mode, normally the bitwise OR-ing GLUT display mode bit masks.

Description: The initial display mode is used when creating top-level windows, sub-windows, and overlays to determine the OpenGL display mode for the to-be created window or overlay.

3. glutCreateWindow()

glutCreateWindow creates a top-level window.

Usage: intglutCreateWindow (char *name); Name-ASCII character string for use as window name

Description: glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name. Implicitly, the current window is set to the newly created window.

4. glutDisplayFunc()

glutDisplayFunc sets the display callback for the current window.

Usage:void glutDisplayFunc (void(*func)(void));

Func: The new display callback function.

Description: glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback.

5. glutMainLoop()

glutMainLoop enters the GLUT event processing loop.

Usage: void glutMainLoop(void);

Description: glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

6. glMatrixMode()

The two most important matrices are the model-view and projection matrix. At many times, the state includes values for both of these matrices, which are initially set to identity matrices. There is only a single set of functions that can be applied to any type of matrix. Select the matrix to which the operations apply by first set in the matrix mode, a variable that is set to one type of matrix and is also part of the state.

7. glTranslate(GLfloat X, GLfloat Y, GLfloat Z)

glTranslate produces a translation by x y z. If the matrix mode is either GL_MODEL_VIEW or GL_PROJECTION, all objects drawn after a call to glTranslate are translated.

8. glRotatef(GLdouble angle, GLdouble X, GLdouble Y, GLdouble Z)

glRotatef produces a rotation of angle degrees around the vector x y z. If the matrix mode is either GL_MODEL_VIEW or GL_PROJECTION, all objects drawn after glRotatef is called are rotated. Use glPushMatrix() and glPopmatrix() to save and restore the unrotated coordinate system.

9. glPushMatrix()

There is a stack of matrices for each of the matrix mode. In GL_MODELVIEW mode, the stack depth is atleast 32. In other modes, GL_COLOR, GL_PROJECTION, and GL_TEXTURE, the depth is atleast 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

10. glPopMatrix()

glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack. Initially, each of the stack contains one matrix, an identity matrix. It is an error to push a full matrix stack or pop a matrix stack that contains only a single matrix. In either case, the error flag is set and no other change is made to GL state.

11. glutSwapBuffers()

Usage: void glutSwapBuffers(void);

Description: Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the front buffer. The contents of the back buffer then become undefined.

12. glPointSize(GLfloat size)

glPointSize specifies the rasterized diameter of points. This value will be used rasterize points. Otherwise, the value written to the shading language built-in variable gl-PointSize will be used. The point size specified by glPointSize is always returned when GL_POINT_SIZE is queried.

13. glutKeyboardFunc()

Usage: void glutKeyboardFunc(void(*func)(unsigned char key, int x, int y)

Func: The new keyboard callback function

Description: glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character.

14. glLineWidth(GLfloat width)

Parameters: width- Specifies the width of rasterized lines. The initial value is 1. **Description:** glLineWidth specifies the rasterized width of lines. The actual width is determined by rounding the supplied width to the nearest integer. i pixels are filled in each column that is rasterized, where I is the rounded value of width.

15. glLoadIdentity(void)

glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix.

4.2 User Defined Functions

1. void init():

Initializes various variables and settings for the game.

2. void restart():

Performs actions needed to reset the game state.

3. void start():

Performs actions needed when the game is started.

4. void renderScene():

Renders the introductory scene with text and credits.

5. void convColors():

Converts color values from the RGB scale (0-255) to OpenGL's scale (0-1).

6. void loadImages():

Loads image files and creates figure objects.

7. loadArray(FILE *fp, int w, int h):

Loads pixel data from a file into a figure object.8. void displayScene4().

8. void keyPress(unsigned char key, int x, int y):

Handles keyboard input events.

9. void reset():

Resets the game state to its initial values.

10. void loop(int val):

The main game loop callback function that updates the game state and checks for collisions.

11. void checkCollision():

Checks for collisions between the dinosaur and cacti.

12. void updateDino():

Updates the position and appearance of the dinosaur.

13. void updateCacti():

Updates the position of the cacti.

14. void drawScene():

Draws the game scene including the dinosaur, cacti, and clouds.

15. void drawFigure(int x, int y, int h, int w, int imgH, int imgW):

Draws a figure at the specified position and size.

16. void drawLine(int x1, int y1, int x2, int y2):

Draws a line between two points.

17. void drawRect(int x, int y, int w, int h):

Draws a rectangle at the specified position and size.

18. void drawText():

Draws text displaying the score and high score.

19. void intToStr(int n):

Converts an integer to a string representation.

20. void void frontscreen():

Renders the text on th front screen.

4.3 PSUEDOCODE

dino.c

```
#include <GL/freeglut.h> // basic GLUT library
#include <stdlib.h>      // for dynamic memory allocation
#include <stdbool.h>     // for boolean variables
#include <time.h>        // for seed in random generation
#include <string.h>      // for text operation
#include <math.h>

#include <stdio.h> // generic terminal printing (for debugging)

#define WW 1200 // Window   Width
#define WH 800  // Window   Height
#define DW 150  // Dinosaur Width
#define DH 100  // Dinosaur Height
#define CW 60   // Cactus   Width
#define CH 80   // Cactus   Height
#define LW 252  // Cloud    Width
#define LH 140  // Cloud    Height

// ---------------------------------------------------------------------------

// Struct for Image Objects

typedef struct figureObjects
{
   unsigned char **r, **g, **b;
} figure;

// ---------------------------------------------------------------------------

// Function Declarations

/* -- Initialization Functions -- */

void init(void);
void convColors(void);
void loadImages(void);
figure loadArray(FILE *, int, int);

/* -- Event Trigger Functions -- */

void keyPress(unsigned char, int, int);
void reset(void);

/* -- Action Functions -- */

void loop(int); // Callback Loop (timed)

void checkCollision(void);
void eventCollision(void);

void updateDino(void);
void updateCacti(void);

void placeCacti(void);

void disp(void); // Display Function

/* -- Drawing Functions -- */

void drawScene(void);
void drawFigure(int, int, int, int, int, int);

// Drawing Utilities

void drawLine(int, int, int, int);
void drawRect(int, int, int, int);
void drawText(void);
char *intToStr(int);

// ---------------------------------------------------------------------------

// Global Variables

/* -- Environment Variables -- */

int refreshPeriod = 5, runtime = 0, score = 0, hiscore;
bool halt = true, started = false;

/* -- Colour Definitions (RGB) -- */

GLfloat cols[7][3] = {
   {0, 0, 0},       // Black
   {230, 230, 230}, // Grey Light
   {85, 85, 85},    // Grey Dark
   {25, 50, 128},   // Blue
   {25, 75, 25},    // Green
   {76, 38, 10},    // Brown
```

```
    {255, 255, 255}  // White
};

/* -- Image Objects -- */

figure dino1, dino2, dino3, dino4, dino5, dino6; // Dinosaur
figure cacti, cloud;                             // Scenery

/* -- Control Variables for Dinosaur -- */

// Figure

int dinoState = 0, dinoX = 100, dinoY = WH / 4, dinoHS = 0;
int dinoHeights[20] = {
    0, 6, 12, 17, 21, 25, 28, 30, 32, 33,
    34, 33, 32, 30, 28, 25, 21, 17, 12, 6};
int dinoLeftEdge, dinoRightEdge;

// Motion

bool dinoJumpEnable = false;
int dinoPeriod = 20;

/* -- Control Variables for Cactus -- */

// Figure

int cactusOffset = WH / 4;
int cactiPos[5] = {WW * 2, WW * 2, WW * 2, WW * 2, WW * 2}, cactiLastPushed = -
999;
int winLeftEdge, winRightEdge;

// Motion

int cactiPeriod = 5, cactiShift = 5;
int gapPeriodOrig = 800, gapDelta = 0, gapPeriod = 800;

/* -- Control Variables for Cloud -- */

// Figure

int cloudOffset = 13 * WH / 20, cloudOffsetDelta = 0;
int cloudPos = WW / 3, cloudLastPushed = -999;

// Motion
```

```
int cloudPeriod = 20, cloudShift = 2;

// ---------------------------------------------------------------------------

// Function Definitions
int selectedColor = 3;
int flag = 0;
void *currentfont;

void setFont(void *font)
{
   currentfont = font;
}

void drawstring(float x, float y, float z, char *string)
{
   char *c;
   int len = (int)strlen(string);
   int i;
   glRasterPos3f(x, y, z);
   /*for(c=string;*c!='\0';c++)
   {
      glColor3f(0.0,0.0,0.0);
      glutBitmapCharacter(currentfont,*c);
   }*/
   for (i = 0; i < len; i++)
   {
      glColor3f(0.0, 0.0, 0.0);
      glutBitmapCharacter(currentfont, string[i]);
   }
}

void frontscreen(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   setFont(GLUT_BITMAP_TIMES_ROMAN_24);
   glColor3f(0, 0, 1);
   drawstring(250, 500.0, 0.0, "BANGALORE INSTITUTE OF TECHNOLOGY");
   glColor3f(0.7, 0, 1);
   drawstring(250, 450, 0.0, "DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING");
   glColor3f(1, 0.5, 0);
   drawstring(250, 400, 0.0, "A MINI PROJECT ON");
   glColor3f(1, 0, 0);
   drawstring(600, 400, 0.0, "DINO GAME");
   glColor3f(1, 0.5, 0);
```

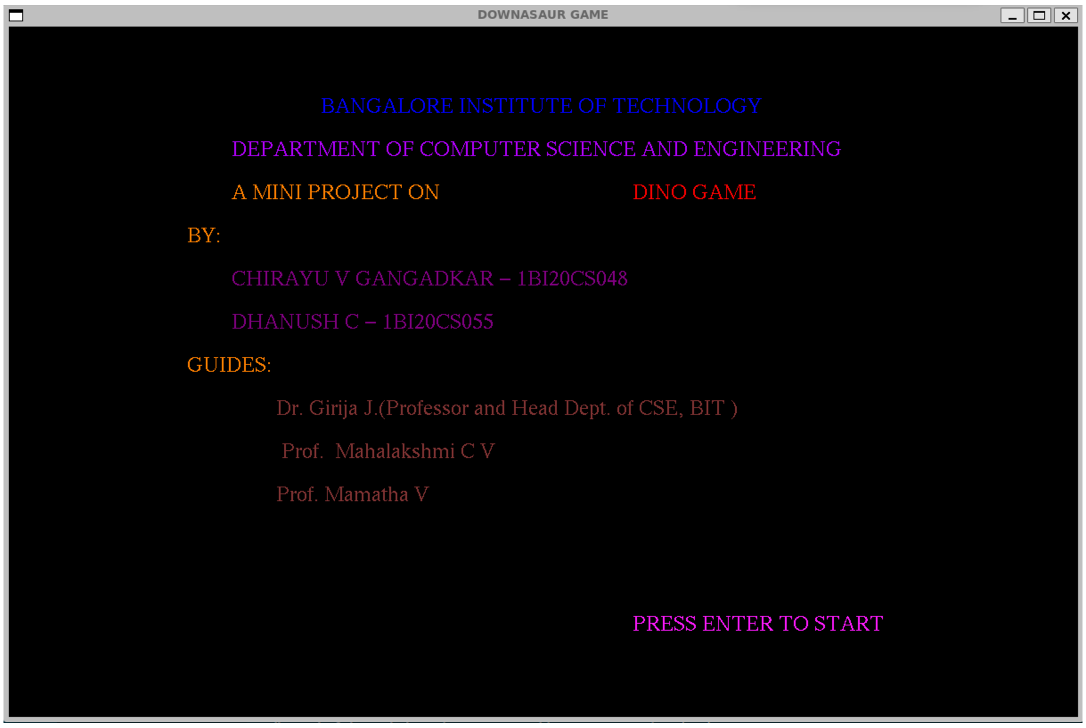
```
    drawstring(200, 350, 0.0, "BY:");
    glColor3f(0.5, 0, 0.5);
    drawstring(250, 300, 0.0, "CHIRAYU V GANGADKAR - 1BI20CS048");
    glColor3f(0.5, 0, 0.5);
    drawstring(700, 300, 0.0, "DHANUSH C - 1BI20CS055");
    glColor3f(1, 0.5, 0);
    drawstring(250, 250, 0.0, "GUIDES:");
    glColor3f(0.5, 0.2, 0.2);
    drawstring(300, 200, 0.0, "GUIDE NAME FIRST");
    drawstring(400, 200, 0.0, "GUIDE NAME SECOND");
    glColor3f(1, 0.1, 1);
    drawstring(550, 100, 0.0, "PRESS ENTER TO START");
    glutSwapBuffers();
    glFlush();
}
void mydisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    if (flag == 0)
        frontscreen();
    if (flag == 1)
        disp();
}
// Reshape function
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-1.0, 1.0, -1.0 * ((GLfloat)h / (GLfloat)w), 1.0 * ((GLfloat)h / (GLfloat)w),
-1.0, 1.0);
    else
        glOrtho(-1.0 * ((GLfloat)w / (GLfloat)h), 1.0 * ((GLfloat)w / (GLfloat)h), -1.0, 1.0,
-1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();
}

// Menu Functions
void menu(int value)
{
    switch (value)
    {
    case 1:
        selectedColor = 1; // Black
        break;
    case 2:
        selectedColor = 2; // Green
        break;
    case 3:
        selectedColor = 3; // Green
        break;
    case 4:
        exit(0); // Exit
        break;
    default:
        break;
    }
    glutPostRedisplay();
}
/* -- Main Function -- */

int main(int argc, char *argv[])
{
    int sub1;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

    glutInitWindowSize(WW, WH);
    glutReshapeFunc(reshape);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("DOWNASAUR GAME");
    init();
    sub1 = glutCreateMenu(menu);
    glutAddMenuEntry("Black", 1);
    glutAddMenuEntry("Green", 2);
    glutAddMenuEntry("white", 3);
    glutAddMenuEntry("Exit", 4);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutKeyboardFunc(keyPress);
    glutDisplayFunc(mydisplay);
    glutTimerFunc(refreshPeriod, loop, 0);
    glutMainLoop();

    return 0;
}

/* -- Initialization Functions  -- */

void init(void)
{
    gluOrtho2D(0.0, WW, 0.0, WH);

    dinoLeftEdge = 0 - (CW / 2) + dinoX;
    dinoRightEdge = DW + (CW / 2) + dinoX;
    winLeftEdge = 0 - (CW / 2);
    winRightEdge = WW + (CW / 2);
    gapPeriod = cactiPeriod * 100;

    convColors();
    loadImages();

    FILE *fp = fopen("res/hiscore", "r");
    fscanf(fp, "%d", &hiscore);
}

void convColors(void)
{
    for (int i = 0; i < (sizeof(cols) / (sizeof(GLfloat) * 3)); i++)
        for (int j = 0; j < 3; j++)
            cols[i][j] /= 255;
}

void loadImages(void)
{
    char **fn = (char **)malloc(sizeof(char *) * 8);
    fn[0] = "res/dino1.pbm";
    fn[1] = "res/dino2.pbm";
    fn[2] = "res/dino3.pbm";
    fn[3] = "res/dino4.pbm";
    fn[4] = "res/dino5.pbm";
    fn[5] = "res/dino6.pbm";
    fn[6] = "res/cacti.pbm";
    fn[7] = "res/cloud.pbm";

    for (int i = 0; i < 8; i++)
    {
        FILE *fp = fopen(fn[i], "r");
        if (fp == NULL)
        {
            printf(" -- file \"%s\" missing\n", fn[i]);
            break;
            exit(0);
        }
        else
        {
            switch (i)
            {
            case 0:
                dino1 = loadArray(fp, DW, DH);
                break;
            case 1:
                dino2 = loadArray(fp, DW, DH);
                break;
            case 2:
                dino3 = loadArray(fp, DW, DH);
                break;
            case 3:
                dino4 = loadArray(fp, DW, DH);
                break;
            case 4:
                dino5 = loadArray(fp, DW, DH);
                break;
            case 5:
                dino6 = loadArray(fp, DW, DH);
                break;
            case 6:
                cacti = loadArray(fp, CW, CH);
                break;
            case 7:
                cloud = loadArray(fp, LW, LH);
                break;
            }
        }
    }
}

figure loadArray(FILE *fp, int w, int h)
{
    unsigned char **r, **g, **b;
    r = malloc(sizeof(unsigned char *) * h);
    g = malloc(sizeof(unsigned char *) * h);
    b = malloc(sizeof(unsigned char *) * h);
    for (int i = 0; i < h; i++)
    {
        r[i] = malloc(sizeof(unsigned char) * w);
        g[i] = malloc(sizeof(unsigned char) * w);
        b[i] = malloc(sizeof(unsigned char) * w);
    }

    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            r[h - i - 1][w - j - 1] = fgetc(fp);
            g[h - i - 1][w - j - 1] = fgetc(fp);
            b[h - i - 1][w - j - 1] = fgetc(fp);
        }
    }

    figure img;
    img.r = r;
    img.g = g;
    img.b = b;
    return img;
}

/* -- Event Trigger Functions -- */

void keyPress(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 13:
        if (flag == 0) // Ascii of 'enter' key is 13
            flag = 1;
        break;
    case 27: // Key Esc
        exit(0);
        break;
    case 32: // Key Space
        if (halt == true)
            reset();
        else
            dinoJumpEnable = true;
        break;
    case 112: // Key 'p'
        halt = !halt;
        break;
    }
    mydisplay();
}

void reset(void)
{
    runtime = 0;
    halt = false;
    started = true;
    score = 0;
    dinoState = 0;
    dinoHS = 0;
    cactiPos[0] = cactiPos[1] = cactiPos[2] = cactiPos[3] = cactiPos[4] = WW * 2;
    cactiLastPushed = -999;
}

/* -- Action Functions -- */

// Callback Loop (timed)

void loop(int val)
{
    runtime += refreshPeriod;

    if (halt == true)
        goto skip;

    if (runtime % dinoPeriod == 0)
        updateDino();
    if (runtime % cactiPeriod == 0)
        updateCacti();
    checkCollision();

    if (runtime % (10 * cactiPeriod) == 0)
        score++;

    glutPostRedisplay();

skip:
    glutTimerFunc(refreshPeriod, loop, 0);
}

void checkCollision(void)
{
    for (int i = 0; i < 5; i++)
        if (cactiPos[i] > dinoLeftEdge && cactiPos[i] < dinoRightEdge)
        {
            if (cactiPos[i] > dinoX + 2 * DW / 5 && cactiPos[i] < dinoX + 3 * DW / 5)
            {
                if (dinoHeights[dinoHS] * 4 < CH)
                {
                    eventCollision();
                    break;
                }
            }
            else
            {
                if (dinoHeights[dinoHS] * 4 < CH - (13 * DH / 20))
                {
                    eventCollision();
                    break;
                }
            }
        }
}

void eventCollision(void)
{
    halt = true;
    if (score <= hiscore)
        return;
    FILE *fp = fopen("res/hiscore", "w");
    fprintf(fp, "%d", score);
    fclose(fp);
}

void updateDino(void)
{
    dinoState = ((dinoState + 1) % 6);
    if (dinoJumpEnable == true)
    {
        dinoHS = ((dinoHS + 1) % 19);
        if (dinoHS == 0)
            dinoJumpEnable = false;
    }
}

void updateCacti(void)
{
    if (cactiPos[4] == winRightEdge)
        cactiLastPushed = runtime;

    for (int i = 0; i < 5; i++)
        cactiPos[i] -= cactiShift;

    if (runtime == (cactiLastPushed + gapPeriod))
    {
        for (int i = 0; i < 4; i++)
            cactiPos[i] = cactiPos[i + 1];
        cactiPos[4] = winRightEdge;
        srand(time(0));
        gapDelta = rand() % 70;
        gapPeriod = gapPeriodOrig + gapDelta * 10;
    }
}

// Display Function

void disp(void)
{
    drawScene();

    // Draw Dinosaur

    drawFigure(dinoState, dinoX, dinoY, dinoHeights[dinoHS] * 4, DW, DH);

    // Draw Cacti

    placeCacti();

    // Draw Vvisible Text Content

    drawText();

    glutSwapBuffers();
}

/* -- Drawing Functions -- */

void drawScene(void)
{
    /* -- Background -- */
    /* -- Background -- */
    glColor3fv(cols[1]);
    drawRect(0, 0, WW, WH);

    /* -- Clouds -- */
    if (runtime % cloudPeriod == 0)
    {
        if (cloudPos >= -(LW / 2) && cloudPos <= WW + (LW / 2))
            cloudPos -= cloudShift;
        else
        {
            cloudPos = WW + (LW / 2);
            srand(time(0));
            cloudOffsetDelta = (rand() % (WH / 40)) * 10;
        }
    }
```

```
drawFigure(7, cloudPos - (LW / 2), cloudOffset + cloudOffsetDelta, 0, LW, LH);

/* -- Ground -- */
switch (selectedColor)
{
case 1:
    glColor3f(0.0, 0.0, 0.0); // Black
    break;
case 2:
    glColor3f(0.0, 0.601, 0.0089); // Green
    break;
case 3:
    glColor3f(0.9, 0.9, 0.9); // white
    break;
default:
    break;
}
drawRect(0, 0, WW, WH / 4);

/* -- Sun -- */
// Define the position and size of the sun
int sunRadius = 50;
int sunX = WW / 2;
int sunY = WH - (WH / 4) + 100;

// Calculate the rotation angle
float rotation_angle = runtime % 360;
float rotation_rad = rotation_angle * 3.14159 / 180;

// Draw the rotating sun
glColor3f(1.0, 1.0, 0.0); // Yellow color
glBegin(GL_POLYGON);
for (int i = 0; i < 360; i++)
{
    float degInRad = i * 3.14159 / 180;
    float x = sunX + cos(degInRad + rotation_rad) * sunRadius;
    float y = sunY + sin(degInRad + rotation_rad) * sunRadius;
    glVertex2f(x, y);
}
glEnd();

// Draw the rays of the sun
glColor3f(1.0, 1.0, 0.0); // Yellow color
glBegin(GL_LINES);
for (int i = 0; i < 360; i += 10)
{
        float degInRad = i * 3.14159 / 180;
        float x1 = sunX + cos(degInRad + rotation_rad) * sunRadius;
        float y1 = sunY + sin(degInRad + rotation_rad) * sunRadius;
        float x2 = sunX + cos(degInRad + rotation_rad) * (sunRadius + 20);
        float y2 = sunY + sin(degInRad + rotation_rad) * (sunRadius + 20);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
    }
    glEnd();
}

void drawFigure(int f, int xpos, int ypos, int yoff, int w, int h)
{
    figure temp;
    if (yoff > 0)
    {
        temp = dino4;
        goto skip;
    }
    switch (f)
    {
    case 0:
        temp = dino1;
        break;
    case 1:
        temp = dino2;
        break;
    case 2:
        temp = dino3;
        break;
    case 3:
        temp = dino4;
        break;
    case 4:
        temp = dino5;
        break;
    case 5:
        temp = dino6;
        break;
    case 6:
        temp = cacti;
        break;
    case 7:
        temp = cloud;
        break;
    }
```

```
skip:
    glBegin(GL_POINTS);
    for (int i = 0; i < h; i++)
    {
        for (int j = 0; j < w; j++)
        {
            glColor3ub(temp.r[i][j], temp.g[i][j], temp.b[i][j]);
            glVertex2f(xpos + j, ypos + yoff + i);
        }
    }
    glEnd();
}

void placeCacti(void)
{
    for (int i = 0; i < 5; i++)
    {
        if (cactiPos[i] > winLeftEdge && cactiPos[i] < winRightEdge)
            drawFigure(6, cactiPos[i] - (CW / 2), cactusOffset, 0, CW, CH);
    }
}

/* -- Drawing Utilities -- */

void drawLine(int xa, int ya, int xb, int yb)
{
    glBegin(GL_LINES);
    glVertex2i(xa, ya);
    glVertex2i(xb, yb);
    glEnd();
}

void drawRect(int xa, int ya, int xb, int yb)
{
    glBegin(GL_POLYGON);
    glVertex2i(xa, ya);
    glVertex2i(xa, yb);
    glVertex2i(xb, yb);
    glVertex2i(xb, ya);
    glEnd();
}

void drawText(void)
{
    glColor3fv(cols[3]);
    glRasterPos2f(3 * WW / 4, 7 * WH / 8);
    unsigned char points[] = "Score : ";
    strcat(points, intToStr(score));
    glutBitmapString(GLUT_BITMAP_HELVETICA_18, points);

    if (score > hiscore)
        hiscore = score;
    glRasterPos2f(3 * WW / 4, 13 * WH / 16);
    unsigned char hipoint[] = "Hi-Score : ";
    strcat(hipoint, intToStr(hiscore));
    glutBitmapString(GLUT_BITMAP_HELVETICA_18, hipoint);

    glColor3fv(cols[1]);
    glRasterPos2f(11 * WW / 28, WH / 10);
    unsigned char pause[] = "Press P for Play/Pause";
    glutBitmapString(GLUT_BITMAP_HELVETICA_18, pause);

    if (halt == false)
        return;

    glColor3fv(cols[4]);
    glRasterPos2f(23 * WW / 56, WH / 2);
    unsigned char success[] = "Press Spacebar";
    glutBitmapString(GLUT_BITMAP_HELVETICA_18, success);
}

char *intToStr(int n)
{
    if (n == 0)
        return "0\0";

    char s[8];
    int k = 0;
    while (n)
    {
        s[k++] = (n % 10) + '0';
        n /= 10;
    }
    char *tem = (char *)malloc(sizeof(char) * k + 1);
    for (int i = 0; i < k; i++)
        tem[i] = s[k - i - 1];
    tem[k] = '\0';

    return tem;
```

Chapter 5

SNAPSHOTS



DOWNASAUR GAME

BANGALORE INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A MINI PROJECT ON DINO GAME

BY:

CHIRAYU V GANGADKAR – 1BI20CS048

DHANUSH C – 1BI20CS055

GUIDES:

Dr. Girija J.(Professor and Head Dept. of CSE, BIT)

Prof. Mahalakshmi C V

Prof. Mamatha V

PRESS ENTER TO START

Figure 5.1 : Introduction Page

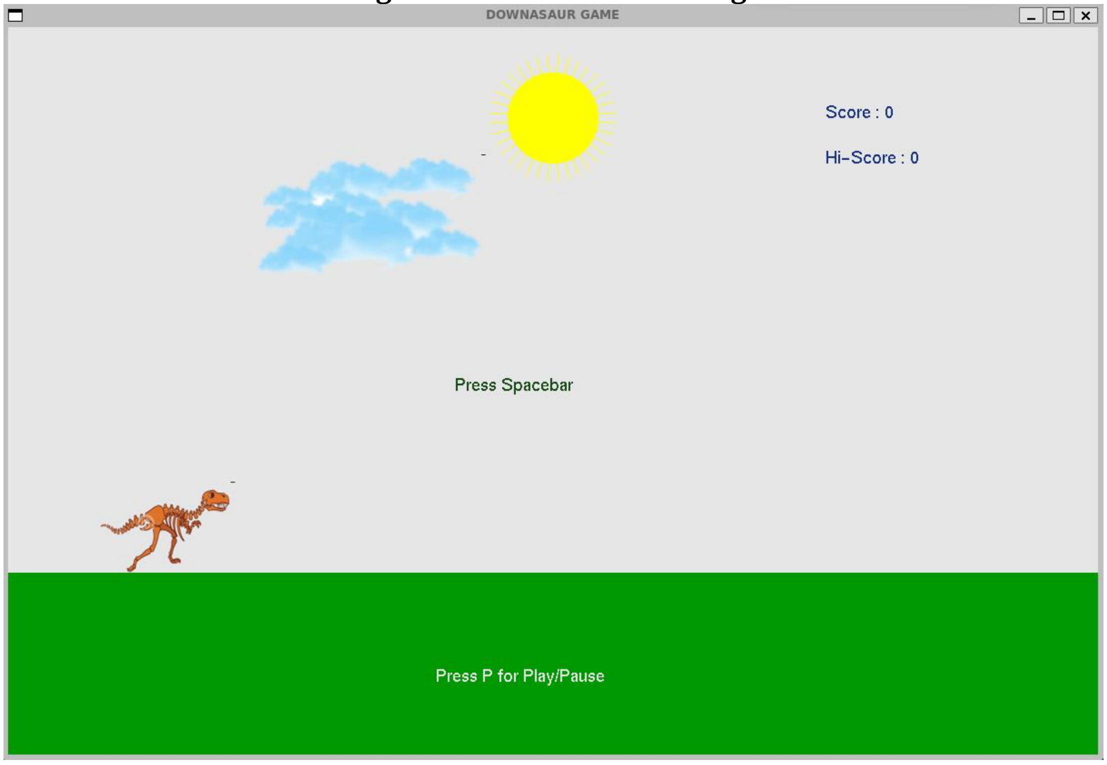


Figure 5.2 : Start Screen of Game

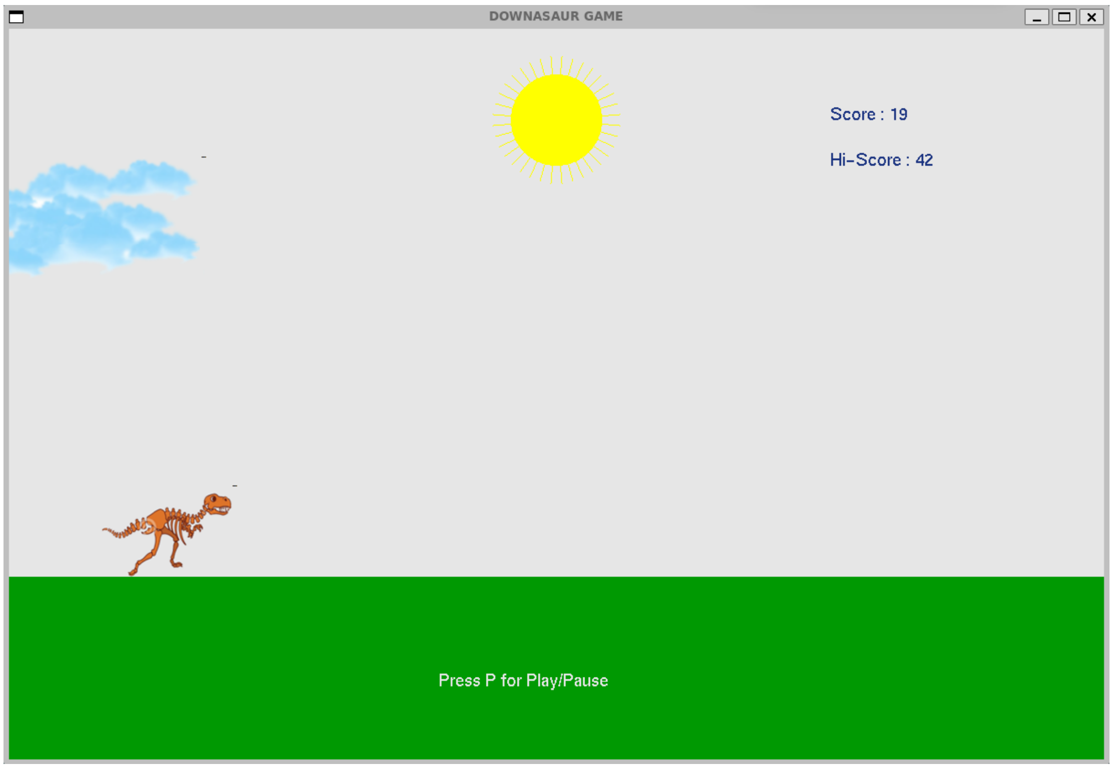


Figure 5.3 : Game starts (Dino Running)

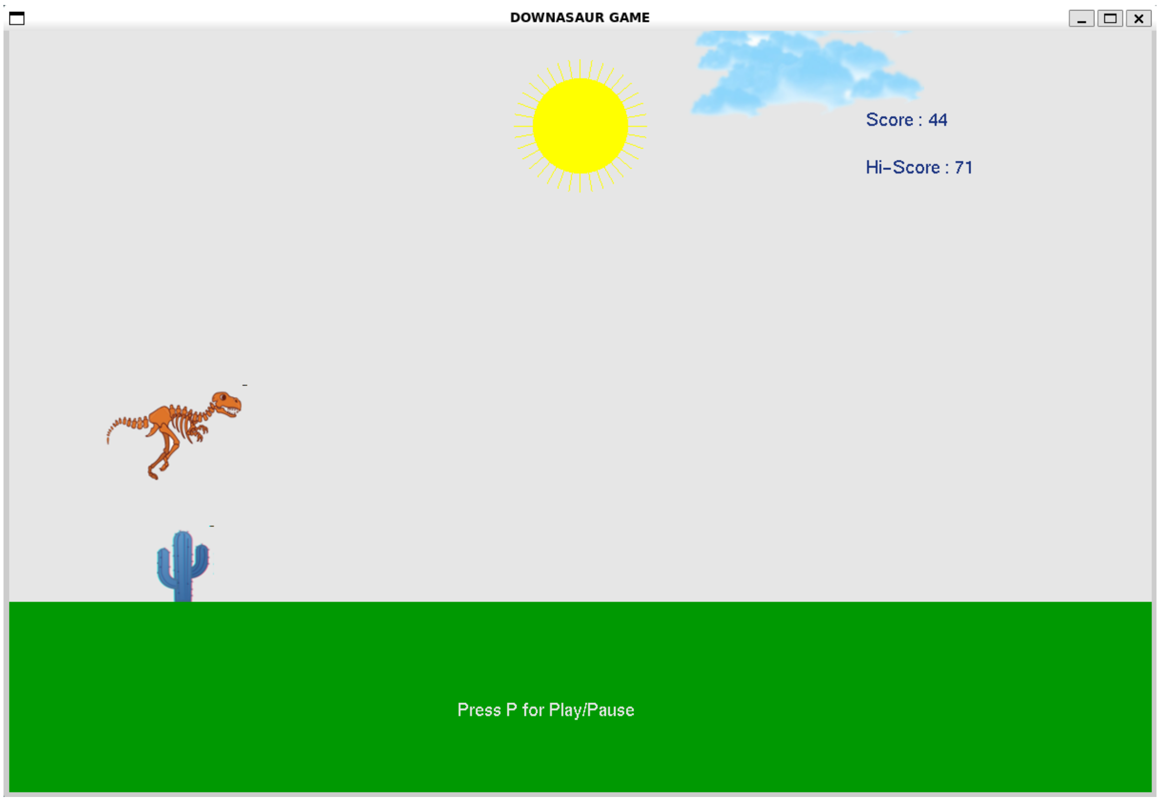


Figure 5.4 : Game Play (Dino Jumping over cacti)

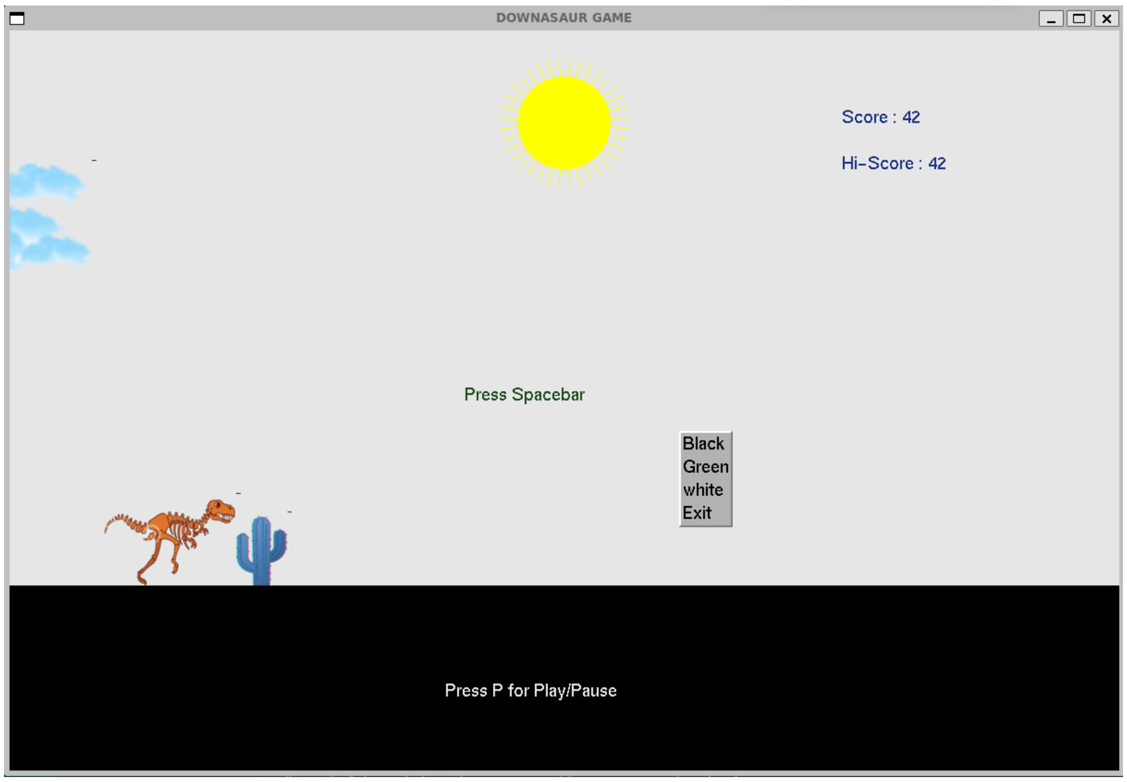


Figure 5.5 : Menu Function(Changing of Ground Color to Black)

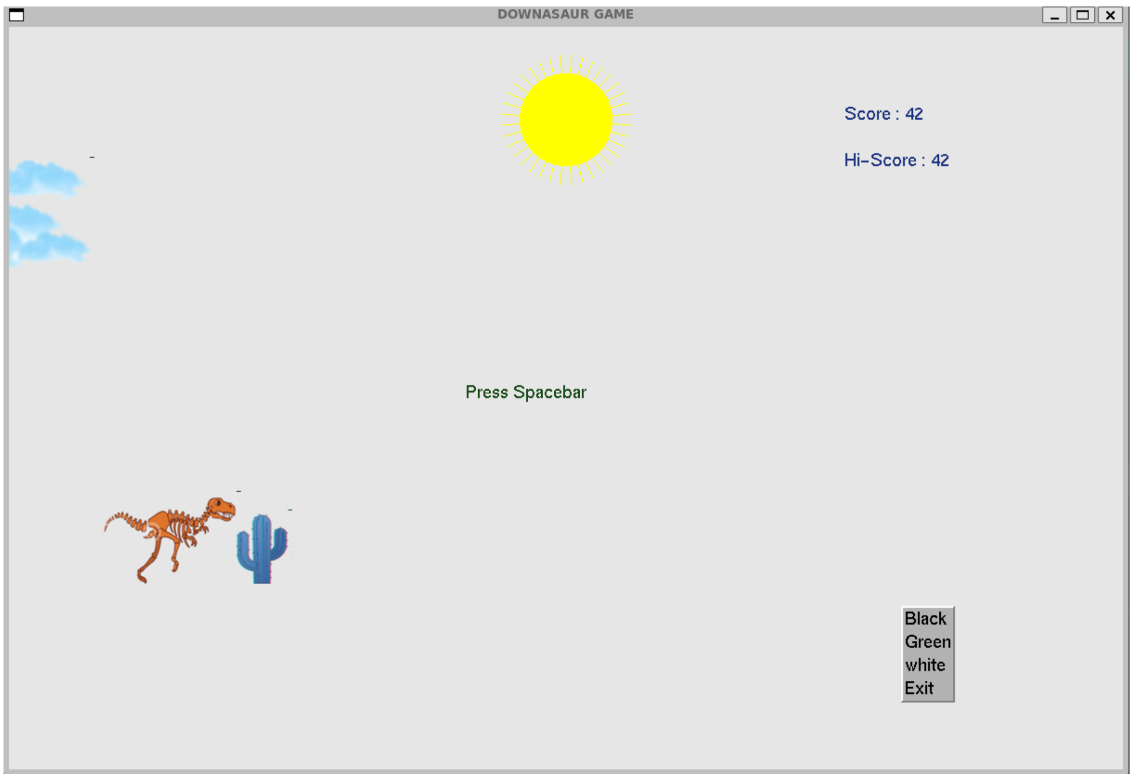


Figure 5.6 : Menu Function(Changing of Ground Color to White)

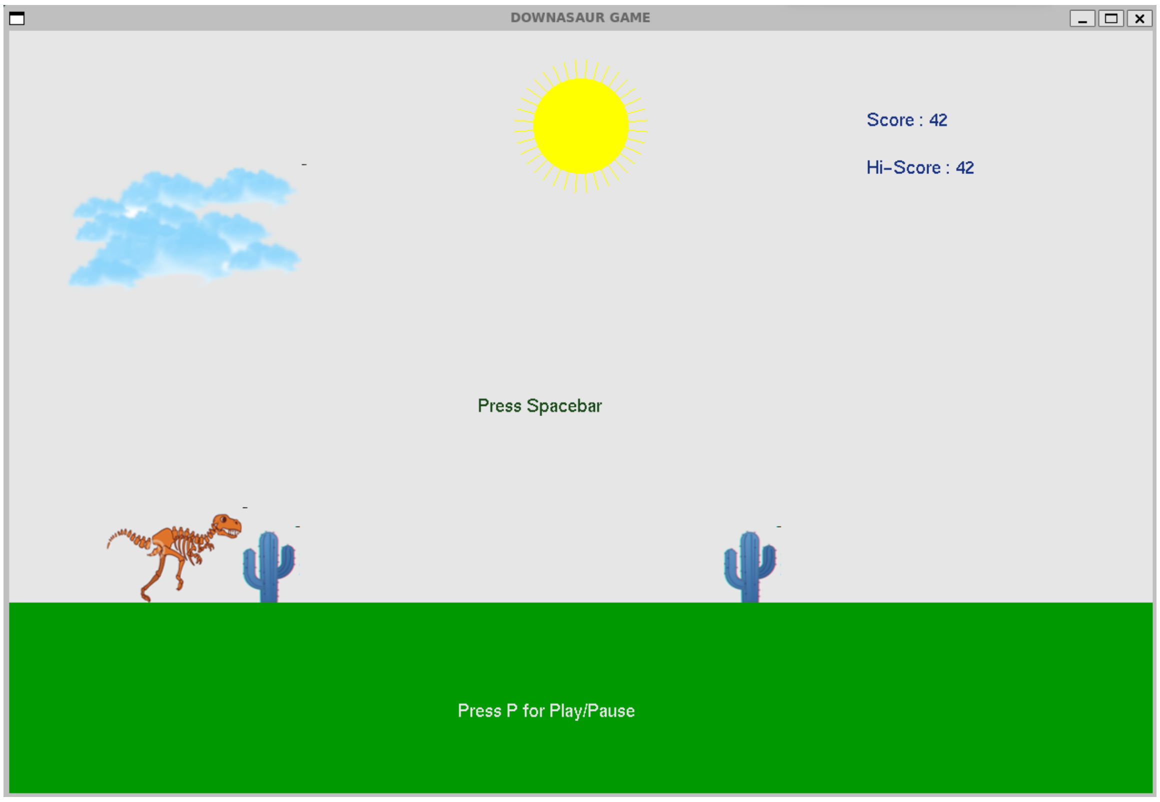


Figure 5.7 : Game Over (Dino Crashes into cacti)

Chapter 6

CONCLUSION

The game involves controlling a dinosaur character to avoid collision with incoming cactus obstacles. Here is a brief conclusion for the project code:

The code consists of several functions, including initialization functions, event trigger functions, and action functions. It also defines a struct for image objects and global variables to control various aspects of the game.

The initialization functions set up the initial state of the game, including loading images, converting colors, and initializing variables. The event trigger functions handle user input, such as key presses, to control the game. The action functions perform the main gameplay logic, including updating the dinosaur's position, updating the cacti positions, checking for collisions, and updating the score.

The game loop function, loop(), is a callback function that is executed at regular intervals to update the game state, handle user input, and check for collisions. It uses the GLUT library for rendering the game window and displaying the game scene.

Overall, the code provides the basic structure and functionality for a Dino Game using OpenGL. Further customization and enhancements can be made to the game, such as adding sound effects, improving graphics, and implementing additional features to make the gameplay more engaging.

6.1 Future Enhancements

- Implement Scoring: Add a scoring system that keeps track of the player's score based on the distance covered by the dinosaur. Display the score on the screen and update it as the game progresses.
- Add Power-Ups: Introduce power-ups or bonuses that the dinosaur can collect during the game. These power-ups could provide temporary advantages such as increased speed, invincibility, or the ability to jump higher.

- Include Different Obstacles: Introduce a variety of obstacles with different shapes, sizes, and behaviors. This variation will make the game more challenging and engaging for players.
- Implement Levels or Difficulty Modes: Create different levels or difficulty modes with increasing obstacles, faster gameplay, or additional challenges. This will provide a sense of progression and encourage players to continue playing.
- Integrate Sound Effects and Music: Add sound effects for actions such as jumping, colliding with obstacles, and collecting power-ups. Additionally, include background music to enhance the overall gaming experience.
- Implement High Score Tracking: Include a feature that keeps track of the highest scores achieved by players. Store and display these high scores to motivate players to beat their own records or compete with others.
- Design a Game Over Screen: Create a visually appealing and informative game over screen that displays the player's final score, high scores, and provides options to restart the game or quit.
- Optimize Performance: Improve the efficiency of the code by implementing optimizations such as object pooling to reuse game objects, reducing unnecessary calculations, or implementing efficient collision detection algorithms.
- Implement Additional Gameplay Elements: Consider adding new gameplay elements such as bonus stages, boss battles, or different game modes to add depth and variety to the game.

BIBLIOGRAPHY

Reference Books

[1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version, 3rd/4th Edition, Pearson Education, 2011

[2] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5th Edition, Pearson Education

[3] James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer Graphics with OpenGL, Pearson Education

[4] Macro Cantu: Mastering Delphi

Websites

[1] https://www.opengl.org/

[2] https://learnopengl.com/