VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAUM



**A**

**MINI PROJECT REPORT**

**On**

**“SNAKE GAME”**

Submitted in partial fulfillment of the Bachelor Degree

**In**

**COMPUTER SCIENCE AND ENGINEERING**

**VI SEMESTER Mini Project Work (10CSL67)**

**By**

**Bhatt Mehul B.**

**(1RI12CS401)**

**Anupam Kumar**

**(1RI12CS005)**

Under the guidance of

**Mrs. Poornima U.S** (Asst. professor)

**&**

**Mrs. Rashmi** (Asst. professor)

Department of Computer Science and Engineering

**2013-2014**



**RR INSTITUTE OF TECHNOLOGY**

R R Layout, Heseraghatta Main Road, Chikkabanavara,   
Bangalore 560 090

**RR INSTITUTE OF TECHNOLOGY**

R R Layout, Heseraghatta Main Road, Chikkabanavara,   
Bangalore 560 090



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

Certified that the project work entitled **“*SNAKE GAME*”** is a bonafide work carried out by **Bhatt Mehul B. (1RI12CS401)** and **Anupam Kumar. (1RI12CS005)** in partial fulfillment for the award of Degree of Bachelor of Engineering in **Computer** **Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2013-2014. 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 academic requirements in respect of Project work prescribed for the Bachelor of Engineering Degree.

--------------------------- --------------------------- ---------------------------

Signature of the Guide Signature of the HOD Signature of the Principal

**Name of the Examiners**  **Signature with date**

1. ----------------------------

2. ----------------------------

ABSTRACT

The project entitled “Snake Game” demonstrates the animation game used in OpenGL.This project aims to bring the fun and simplicity of snake game with some new features

This project explores a new dimension in the traditional snake game to make it more interesting. The simplicity of this game makes it an ideal candidate for a minor project as we can focus on advanced topics like implementation of computer controlled intelligent opponents. The snake is implementing by using of Sphere, Cone and its food is made from Sphere function, all this function is inbuilt function in the OpenGL library.

Snake will run with in the cube, it cannot go outside of cube if it reaches to the boundary of cube it will reappear from the opposite side of that boundary. This cube is drawn in such a manner that the user feels that it is one type of surface where snake is running. In this project I have implement function which shows the position of each every coordinate

Snake Food contains different points and after getting some points, the level will increase and speed of the snake will be increase. This game will continue until snake eats itself only, once it eat itself the game is over then again the game will restart with initial level of game 1 and score 0 and same points of food.

ACKNOWLEDGEMENT

I consider it a privilege to whole-heartedly express our gratitude and respect to each and everyone who guided and helped us in the successful completion of this project.

I would greatly mention the enthusiastic influence provided by **Prof. Mrs. Poornima U.S**, Department of Computer Science, for his ideas and co-operation showed on me during my venture and making this project a great success.

I am also thankful to **Mr.** HOD, Department of Computer Science, for his co-operation and encouragement at all moments of my approach.

I am very thankful to the Principal, **Dr. M.K. Muralidhara,** RRIT, Bangalore, for being kind enough to provide me an opportunity to work on a project in this institution.

Finally, it is a pleasure and happiness to the friendly co-operation showed by all the staff members of computer science department, RRIT

Bhatt Mehul B.

(1RI12CS401)

Anupam Kumar

(1RI12CS005)

CONTENTS

Page No.

1. Introduction 6
   1. Introduction to OpenGL
      1. Setting Up Compiler
   2. Introduction to Project 10
2. Requirement Specification 12
   1. Hardware Requirements
   2. Software Requirements
   3. Platform Used 13
3. Design
   1. Function Design of Major Modules
   2. Flow Chart
4. Implementation 17
5. Source Code 22
6. Snap Shots 39
7. Future Enhancement 45
8. Conclusion 46
9. Bibliography 47

1. INTRODUCTION

1.1 INTRODUCTION OF OPENGL

OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives – points, lines, polygons, image and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn).

Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines and polygons), the OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated three-dimension object such as a sphere, a cube, a cone etc. GLUT may not be satisfactory for full-featured of OpenGL application, but it is a useful starting point for learning OpenGL.

GLUT is designed to fill to need for windows system independent programming interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Removing windows system operation form OpenGL is a sound decision because it allows the OpenGL graphic system to be retargeted to various system including powerful but expensive graphic workstations as well as mass-production graphics system like video games, set-top boxes for interactive television and PCs.

GLUT simplifies the implementation of programs using OpenGL rendering. The GLUT Application Programming Interface (API) requires very few routines to display a graphic scene render using OpenGL. The GLUT routines also take relatively few parameters.

Rendering Pipeline:

Most implementation of OpenGL has a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. Although this is not a strict rule of how OpenGL is implemented, it provides a reliable guide for predicting what OpenGL will do. Geometric data (vertices, line and polygons) follow a path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images and bitmaps) are treated differently for part of process. Both types of data undergo the same final step (rasterization) before the final pixel data is written to the frame buffer.

Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line width, point’s size, shading model and coverage calculation to support ant aliasing is taken into consideration as vertices are connected into lines or the interior pixels are calculated for filled polygon. Color and depth values are assigned for each fragment square. The processed fragment is the drawn into the appropriate buffer, where it has finally advanced to be a pixel and achieved its final resting place.

Libraries:

OpenGL provides a powerful but primitives set of rendering commands and all higher-level drawing must be done in terms of these commands. There are several libraries that allow you to simplify your programming tasks, including the following:

* OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up vertexes for specific viewing orientations and projection and rendering surface.
* OpenGL Utility Toolkit (GLUT) is a windows system independent toolkit, written by MARK KILGARD, to hide complexities of differing windows APIs.

Include Files:

For all OpenGL application, you want to include ‘GL.H’ header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which also requires inclusion of the ‘GLU.H’ header file. So almost every OpenGL source file….

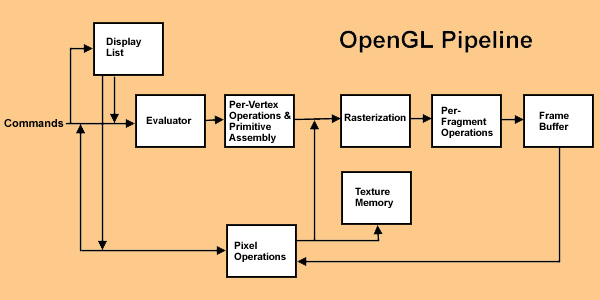
Begins with: #include<GL/gl.h>

#include<GL/glu.h>

If you are using the OpenGL Utility Toolkit (GLUT) for managing your windows manager tasks, you should include:

#include<GL/glut.h>

Note that ‘GLUT.H’ guarantees that ‘GL.H’ and ‘GLU.H’ are properly included for you so including these three files is redundant. To make your GLUT program portable, include ‘GLUT.H’ and do not include ‘GL.H’ or ‘GLU.H’ explicitly. OpenGL has historically been influential on the development of 3D accelerators, promoting a base level of functionality that is now common in consumer-level hardware:



1.1.1 SETTING UP COMPILER

* Windows Using MS Visual C++

Installing GLUT

1. Most of the following files (i.e. OpenGL and GLU) will already be present if you have installed MS Visual C++ v5.0 or later. The following GLUT files will need to be copied into the specified directories.
2. To install:
   1. Right-click each link
   2. Choose Save Link As…
   3. Accept the default name (just click Save)
   4. Libraries (place in the lib\subdirectory of Visual C++)
      1. Opengl32.lib
      2. Glu32.lib
      3. Glut32.lib
3. Include files (place in the include\GL\subdirectory of Visual C++)
   1. gl.h
   2. glu.h
   3. glut.h
4. Dynamically–linked libraries (place in the \Windows\System subdirectory)
   1. opengl32.dll
   2. glu32.dll
   3. glut32.dll

1.2 INTRODUSTION OF PROJECT

A game is structured playing, usually undertaken for enjoyment and sometimes used as an educational tool. Games are distinct from work, which is usually carried out for remuneration, and from art, which is more often an expression of aesthetic or ideological elements.

The project entitled “Snake Game” demonstrates the animation game used in OpenGL. The first basic think is to change background and put some character for introduce itself in very short way. It has been done by “Raster Text” and “Lights”.

Raster text is simple and fast. Characters are defining as rectangular of bits call bit-blocks. Each block defines a single character by the pattern of 0 and 1 bits in the block.

Light function is use to change the color of back gourd. In this function there are 3 type of light.

1. Ambient Light:-

Ambient lighting is a general illumination that comes from all direction in a room that has no visible source. This type of lighting is in contrast to directional lighting.

2. Positioned Light:-

Positioned light is declared for the contrast. Contrast is also important and poorly positioned light source may cause contrast reduction, resulting in loss of visibility.

3. Directed Light:-

Directional lights are different from the previous two by the fact that all the light rays are parallel to each other. In reality this never happens, however, when a light source is very distant (e.g. sun, moon) the difference in angle of the incoming rays is so small that you could say they are parallel.

Now make part of the project is Snake and its Food. The shape of Snake is making with help of Solid Sphere and Cone and colored with the Color function. The Snake’s Food also make by the Solid Sphere and Color function.

When snake will eat the food it should increase the size of snake. For each food it has different points. These points will calculate and after some points the level of the game should be increase. Each and every level the speed of snake will increase. If in case Snake only eat itself then the game over and again it will start from beginning.

Snake will run with in the cube, it cannot go outside of cube if it reaches to the boundary of cube it will reappear from the opposite side of that boundary. These cube you can rotate at any angel of X-axis, Y-axis, Z-axis and even you have zoom in and zoom out facilities.

The Frame per Second (FPS) is introduced in this project. FPS means to increase Frames per Seconds you can lower the Graphics the FPS will increase. The human eye can see about 30 FPS, but for things like game 60+ is ideal. Along with this it mention the position of the snake each and every point of X-axis and Z-axis.

These all function are performed with help of KEYBOARD function. From Keyboard function you can move Snake in all direction and Cube also in of the axis.

2. REQUIREMENT SPECIFICATION

2.1 Hardware Specification:

Processor : Intel Pentium 4 onwards Compatible Hardware

RAM : 256MB RAM

Hard Disk : 80 GB

2.2 Software Specification:

Operating System : Window XP onwards

Language Tool : C

Compiler : Visual C++ Libraries : Supporting glut32.h & 32 bit color resolution Documentation Tool : Ms-Word

2.3 Platform used:

We have chosen the Windows platform because it is the commonly used in most of Systems and Microsoft Visual C++, which is simple and user friendly and easy to debug errors, if any, also it offers support to graphics functions.

3. DESIGN

3.1Function Design of Major Modules:

Project design first requires gathering, synthesizing, and analyzing information with enough objectivity and detail to support a program decision that makes optimum use of resources to achieve desired results.

In Snake Game project Snake, Food, Surface, Text, Fps and Viewing Angels this all are the major function which are the use to design the Snake Game project. After completion of designing, this all major function can implement by writing the code.

Snake and Food:-

The Snake can draw by many ways like we can make by square, circle etc but using of circle is good for viewing the snake. This project is done in 3D so with the help of OpenGL, we can easily built make circle. In OpenGL to draw the circle in 3D we have function name “glutSolidsphere ()”. In function pass argument for how much size of sphere is need for project.

To define a specific position of Snake we can use the Translate function. These function is declare as “glTranslatef ()”, it will take only 3 arguments. These 3 arguments which help to object where to place, this argument defines the position of X-axis, Y-axis and Z-axis.

According to this project the Snake has to run in vertical and horizontal ways only. It could be in negative or positive values for run the snake in vertical and horizontal axis. For rotating the Snake head in different direction we are using the Rotation function. To declare this function “glRotate()”, by passing the arguments , it define the rotation around X-axis, Y-axis, Z-axis.

To move the Snake at any direction with help of keyboard, one User Define Function has to declare name as “Run“. In these function we have to make check the position of X-axis and Z-axis. Here Y-axis will not come for checking because it’s vertical according to surface of snake where it’s run. Here each and every position are storing in some variable to check the condition for moving the snake.

The Food also are drawn by these two function only, but differences only that after eating the Food by Snake, it will reappear at different position on surface. For reappearing we have to make one User Define Function, which will call after each Food has been eaten by Snake. While eating Food by Snake, it causes collision between Head of Snake and Food. According to the project whenever the Snake will eat Food, the size of Snake has to be increase by 1.For this type of functionality the logic has declare in “Run” function.

Surface and Viewing Angels:-

The Surface or Plane is main module of this project. The Snake will run within this surface only. Surface is made by help of the OpenGL inbuilt function name Solid Cube. To declare this function we use “glutSolidCube()”, in this project we need 3D cube so by this function we can easily implement surface for the snake.

Viewing surface from different angle we have to design one function with the help of OpenGL function called “glRotatef()”, we have give argument in a such way that it will rotate the Surface in every degree of X-axis, Y-axis and Z-axis. It can be rotate in negative and positive degree also and this could be done by help of keyboard.

Even we can do Zoom in and Zoom out by using the zoom function with help of keyboard. In OpenGL they have given facility for interacted with the physical devices such as keyboard, mouse etc. After rotation has been completed is should be redisplay again so we have to use OpenGL function name “glutPostRedispaly ()”.

The main requirement of this project is that the Snake should be reappearing again from opposite side if it reaches to any end of the side of cube. To design this logic we have put the condition on body position of snake and the X-axis, Z-axis. This logic is defined in function “Run”.

Text:-

Text feature in OpenGL is very use to show character, number or any special symbols in the graphical window. In this project the text feature is to display Welcome Screen, Position of Snake, Level, Points, Fps value.

OpenGL have function to display the text on graphical window screen, function called “glutBitmapCharracter ()”. By using this function we can easily display text on screen. Even we can give specific position for that particular text. In OpenGL function called “glRasterPos ()”, this will help to give position of the particular text. By passing two arguments of X-axis and Y-axis, it will place text at that position.

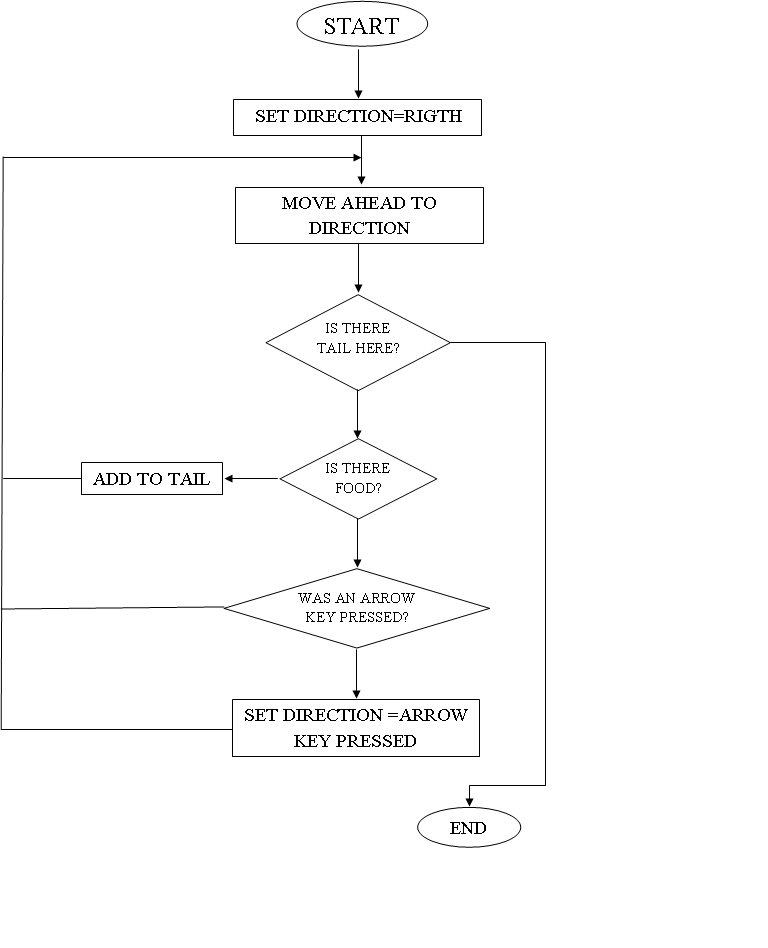
Frame per Second and Position of Snake:-

Frame rate (also known as frame frequency) is the frequency (rate) at which an imaging device produces unique consecutive images called frames. The term applies equally well to film and video cameras, computer graphics, and motion capture systems. Frame rate is most often expressed in frames per second (FPS) and is also expressed in progressive scan monitors as hertz (Hz).

The Fps can count by help of OpenGL function called “GetTickCount ()”. GetTickCount () in windows.h returns ticks (miliseconds) elapsed. When your app start, call this function and store its value, then whenever you need to know elapsed time since your program start, call this method again and subtract its value from start value.

To count position of the Snake for each and every point of coordinate system, which variable has used in user function “Run” for checking condition that variable only we have print here to get snake position.

3.2 Flow Chart:-



4. Implementation

In computer science, an implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through computer programming and deployment.

1. Welcome Screen:-

* + glLightModelfv();
    - The glLightModelfv function sets lighting model parameter.
  + glLight();
    - The glLight Function will return the light source parameter values.
* glEnable();
  + This function will enable various capabilities. It takes only one argument as a parameter.
* glViewPort();
  + GlViewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let x nd y nd be normalized device coordinates. Then the window coordinates x w y w are computed as follows:
    - x w = x nd + 1 ⁢ width 2 + x
    - y w = y nd + 1 ⁢ height 2 + y
  + Viewport width and height are silently clamped to a range that depends on the implementation.
* glutBitmapCharacter();
  + Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font. The available fonts are:
    - GLUT\_BITMAP\_8\_BY\_13
      * A fixed width font with every character fitting in an 8 by 13 pixel rectangle.
* glRasterPos2f();
  + The raster position is transformed by the modelview matrix. If you are applying some matrix to your modelview while panning and zooming, then it will affect glRasterPos2f. You might want to load an identity matrix, then call glRasterPos2f.
* glMatrixMode();
  + GlMatrixMode sets the current matrix mode.
    - GL\_PROJECTION
      * Applies subsequent matrix operations to the projection matrix stack.
* glPopMatrix(); and glPushMatrix();
  + The glPushMatrix function pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on the top of the stack is identical to the one below it. The glPopMatrix function pops the current matrix stack, replacing the current matrix with the one below it on the stack. Initially, each of the stacks contains one matrix, an identity matrix.
* glClear();
  + GlClear sets the bitplane area of the window to values previously selected by glClearColor, glClearDepth, and glClearStencil.
  + The pixel ownership test, the scissor test, dithering, and the buffer writemasks affect the operation of glClear. The scissor box bounds the cleared region. Alpha function, blend function, logical operation, stenciling, texture mapping, and depth-buffering are ignored by glClear.
* glClearColor();
  + GlClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range 0 1.

2. Snake, Food and Surface:-

* gLLoadIdentity();
* Identity matrix is the equivalent of 1 for number. As you know any number that multiplies with 1 is itself (e.g. A x 1 = A), the same thing goes for matrix (MatrixA x IdentityMatrix = MatrixA). So loading an identity matrix is a way to initialize your matrix to the right state before you multiply further matrices into the matrix stack.
* glTranslatef();
* GLTranslate produces a translation by (*x*, *y*, *z*). The current matrix (see GLMatrixMode) is multiplied by this translation matrix, with the product replacing the current matrix.
* glRotatef();
* GLRotate produces a rotation of *angle* degrees around the vector (*x*, *y*, *z*). The current matrix (see GLMatrixMode) is multiplied by a rotation matrix with the product replacing the current matrix. If the matrix mode is either GL\_ModelView or GL\_Projection, all objects drawn after glRotate is called are rotated. Use glPushMatrix and glPopMatrix to save and restore the unrotated coordinate system.
* glScalef();
* GLScale produces a nonuniform scaling along the *x*, *y*, and *z* axes. The three parameters indicate the desired scale factor along each of the three axes. If the matrix mode is either GL\_ModelView or GL\_Projection, all objects drawn after glScale is called are scaled. Use glPushMatrix and glPopMatrix to save and restore the unscaled coordinate system.
* glPerspective();
* GluPerspective specifies a viewing frustum into the world coordinate system. In general, the aspect ratio in gluPerspective should match the aspect ratio of the associated viewport. For example, aspect = 2.0 means the viewer's angle of view is twice as wide in x as it is in y. If the viewport is twice as wide as it is tall, it displays the image without distortion.
* The matrix generated by gluPerspective is multiplied by the current matrix, just as if glMultMatrix were called with the generated matrix. To load the perspective matrix onto the current matrix stack instead, precede the call to gluPerspective with a call to glLoadIdentity.
* glColor3f();
  + The GL stores both a current single-valued color index and a current four-valued RGBA color. GLColor sets a new four-valued RGBA color**.** GLColor has two major variants: GLColor3 and GLColor4. GLColor3 variants specify new red, green, and blue values explicitly and set the current alpha value to 1.0 (full intensity) implicitly. GLColor4 variants specify all four color components explicitly.
  + GLColor3b, GLColor4b, GLColor3s, GLColor4s, GLColor3i, and GLColor4i take three or four signed byte, short, or long integers as arguments. When v is appended to the name, the color commands can take a pointer to an array of such values.
* glutSolidSphere();
  + Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.
* glutSolidCube();
  + GlutSolidCube and GlutWireCube render a solid or wireframe cube respectively. The cube is centered at the modeling coordinate’s origin with sides of length size.

3. Speedup the Snake and FPS:-

* glutTimerFunc();
  + GlutTimerFunc registers the timer callback func to be triggered in at least msecs milliseconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc. Multiple timer callbacks at same or differing times may be registered simultaneously.
  + The number of milliseconds is a lower bound on the time before the callback is generated. GLUT attempts to deliver the timer callback as soon as possible after the expiration of the callback's time interval.
  + There is no support for canceling a registered callback. Instead, ignore a callback based on its value parameter when it is triggered.
* getTickCount();
  + The resolution of the GetTickCount function is limited to the resolution of the system timer, which is typically in the range of 10 milliseconds to 16 milliseconds. The resolution of the GetTickCount function is not affected by adjustments made by the GetSystemTimeAdjustment function.
* GetTickCount() in windows.h returns ticks(miliseconds) elapsed. When your app start, call this function and store its value, then whenever you need to know elapsed time since your program start, call this method again and subtract its value from start value.

5. Source Code:-

#include<stdafx.h>

#include<stdio.h>

#include<time.h>

#include<windows.h>

#include<glut.h>

#define UP 1

#define Down 2

#define LEFT 3

#define RIGHT 4

// Status Variables

GLint lvl = 1;

GLint points =0;

GLint points1 = 0;

GLint points2 = 0;

GLint points3 = 0;

GLint points4 = 0;

GLint size = 0;

GLbyte gameOver = true;

GLbyte EnableLight = true;

// Snake Variables

GLint bodyPos[2][100] = {{}};

GLint \_x = 5;

GLint \_z = 10;

GLint \_oldX[2] = {};

GLint \_oldZ[2] = {};

GLbyte direction = 0;

// Food Variables

GLint \_ax = 0;

GLint \_az = 0;

GLint \_bx = 0;

GLint \_bz = 0;

GLint \_cx = 0;

GLint \_cz = 0;

GLint \_dx = 0;

GLint \_dz = 0;

// Screen variables

GLint \_w = 800;

GLint \_h = 550;

GLint \_Giw = 0;

GLint \_Gih = 0;

GLint \_Gfw = 150;

GLint \_Gfh = 150;

//Variables for the Camera Angle

static GLfloat view\_rotx=45.0F ;

static GLfloat view\_roty=0.0F ;

static GLfloat view\_rotz=0.0F ;

//Variable for the Zoom nad Rotation

static GLfloat headRotation=90.0F ;

static GLfloat zoom=-300.0F ;

// Variables to check the FPS

DWORD g\_dwLastFPS = 0;

int g\_nFPS = 0, g\_nFrames = 0;

//Configure the lightning

void initLight()

{ //Add ambient light

GLfloat ambientColor[] = {0.3f, 0.4f, 0.8f, 1.0f};

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambientColor);

//Add positioned light

GLfloat lightColor0[] = {0.5f, 0.5f, 0.5f, 1.0f};

GLfloat lightPos0[] = {4.0f, 0.0f, 8.0f, 1.0f};

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightColor0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos0);

//Add directed light

GLfloat lightColor1[] = {0.5f, 0.2f, 0.2f, 1.0f};

//Coming from the direction (-1, 0.5, 0.5)

GLfloat lightPos1[] = {-1.0f, 0.5f, 0.5f, 0.0f};

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, lightColor1);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos1);

}

//initialize the first configurations

void Initialize(void)

{

glEnable(GL\_DEPTH\_TEST);

glClearColor(0.7f, 0.9f, 1.0f, 1.0f); //Change the background to sky blue

if(EnableLight)

{

glEnable(GL\_COLOR\_MATERIAL); //Enable color

glEnable(GL\_LIGHTING); //Enable lighting

glEnable(GL\_LIGHT0); //Enable light #0

glEnable(GL\_LIGHT1); //Enable light #1

glEnable(GL\_NORMALIZE); //Automatically normalize normals

}

}

//Configure window resize

void resize (int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0, (double)w / (double)h, 1, 800.0);

}

void Write(char \*string)

{//Write string on the screen

while(\*string)

glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, \*string++);

}

//This Function will rotate the object according to the Angles

void ManipulateViewAngle() {

glRotatef(view\_rotx,1.0,0.0,0.0);//Rotate Arround X axis

glRotatef(view\_roty,0.0,1.0,0.0);//Rotate Arround Y axis

glRotatef(view\_rotz,0.0,0.0,1.0);//Rotate Arround Z axis

}

//This Function will reset the snake size and location...

void Reset(){

\_x = 5;

\_z = 10;

direction = 0;

lvl = 1;

points1 = 0;

points2 = 0;

points3 = 0;

points4 = 0;

size = 0;

gameOver = false;

view\_rotx=45.0F ;

view\_roty=0.0F ;

view\_rotz=0.0F ;

headRotation=90.0F ;

zoom=-300.0F ;

}

//Display a welcome screen

void WelcomeScreen()

{

char tmp\_str[40];

glColor3f(1, 0, 0);

glRasterPos2f(0, 25);

Write("Welcome To Snake 3D Game.");

glColor3f(1, 0, 1);

glRasterPos2f(0, 15);

Write("Prepared By Bhatt Mehul B. & Anupam Kumar.");

glColor3f(1, 0, 1);

glRasterPos2f(0, 8);

Write("USN:-1RI12CS401 & USN:-1RI12CS005");

glColor3f(0, 0, 1);

glRasterPos2f(0, 0);

Write("To Start Playing please press 'n'. Enjoy");

}

void DrawSnake()

{

int i;

//Drawing the head & the plane

glPushMatrix();

ManipulateViewAngle();

//This will draw the plane that the snake will run on.

glPushMatrix();

glColor3f(0.6f, 0.7f, 0.8f);

glTranslatef(75.0, -16.0, 75.0);

glScalef(155,5.0,155);

glutSolidCube(1);

glPopMatrix();

//Here we will draw the Head of the snake

glColor3f(0.0,0.0,0.0);//Color it blue

glTranslatef(\_x,-10.0,\_z);//Give it the location according to \_x & \_z

glScalef(0.5,0.5,0.5);

glutSolidSphere(10,20,20);//Draw the head as a sphere a litle bit bigger than the body spheres

glRotatef(headRotation, 0.0, 1.0, 0.0);

glColor3f(1.0,1.0,0.0);

glTranslatef(0,0.0,6.0);

glScalef(0.8,1.0,1.0);

glutSolidCone(10,10,20,20);

glColor3f(0,0,0);

glTranslatef(-4.0,10.0,0.0);

glScalef(0.3,0.3,0.3);

glutSolidSphere(10,20,20);

glTranslatef(26.0,0.0,0.0);

glutSolidSphere(10,20,20);

glPopMatrix();

//Drawing the body

for(i=0; i<size; i++)

{//Loop throw the size and draw spheres representing the body

glPushMatrix();

ManipulateViewAngle();

glTranslatef(bodyPos[0][i],-10.0,bodyPos[1][i]);//this will locate the spheres

glColor3f(1.0,1.0,0.6);//Color Red

glScalef(0.5,0.5,0.5);

glutSolidSphere(7,20,20);

glPopMatrix();

}

}

bool collision()//This Function Will Check for Collision

{

int i;

for(i=0; i<size; i++)

{

if((bodyPos[0][i] == \_x && bodyPos[1][i] == \_z) ||

((bodyPos[0][i] >= \_x) && (bodyPos[0][i] <= \_x + 5) && (bodyPos[1][i] >= \_z) && (bodyPos[1][i] <= \_z + 5)) ||

((bodyPos[0][i] <= \_x) && (bodyPos[0][i] >= \_x - 5) && (bodyPos[1][i] <= \_z) && (bodyPos[1][i] >= \_z - 5)) ||

((bodyPos[0][i] <= \_x) && (bodyPos[0][i] >= \_x - 5) && (bodyPos[1][i] >= \_z) && (bodyPos[1][i] <= \_z + 5)) ||

((bodyPos[0][i] >= \_x) && (bodyPos[0][i] <= \_x + 5) && (bodyPos[1][i] <= \_z) && (bodyPos[1][i] >= \_z - 5)))

return true;

}

return false;

}

void DrawFood()

{

char tmp\_str[100];

//Draw the Sphere representing the Food for the snake

glPushMatrix();

ManipulateViewAngle();

glTranslatef(\_bx,-10.0,\_bz);

glColor3f(0.8, 0.4, 0.4);

glScalef(0.5,0.5,0.5);

glutSolidSphere(7,20,20);

glPopMatrix();

}

void DrawFood2()

{

char tmp\_str[100];

glPushMatrix();

ManipulateViewAngle();

glTranslatef(\_ax,-10.0,\_az);

glColor3f(0.4, 0.6, 0.4);

glScalef(0.5,0.5,0.5);

glutSolidSphere(7,10,20);

glPopMatrix();

}

void DrawFood3()

{

char tmp\_str[100];

glPushMatrix();

ManipulateViewAngle();

glTranslatef(\_cx,-10.0,\_cz);

glColor3f(0.3, 0.4, 0.3);

glScalef(0.5,0.5,0.5);

glutSolidSphere(7,10,20);

glPopMatrix();

}

void DrawFood4()

{

char tmp\_str[100];

glPushMatrix();

ManipulateViewAngle();

glTranslatef(\_dx,-10.0,\_dz);

glColor3f(0.3, 0.9, 0.3);

glScalef(0.5,0.5,0.5);

glutSolidSphere(7,10,20);

glPopMatrix();

}

//Generates Random Numbers for the location of the food that the snake will eat

int RandomNumber(int high, int low)

{

return (rand() % (high-low))+low;

}

//Generate the New Food that the snake will eat

void newFood()

{

time\_t seconds;

time(&seconds);

srand((unsigned int) seconds);

\_bx = RandomNumber(\_Gfw-\_Giw, \_Giw+10);

\_bz = RandomNumber(\_Gfh-\_Gih, \_Gih+10);

}

int RandomNumber2(int high, int low)

{

return (rand() % (high+low))-low;

}

void newFood2()

{

time\_t seconds;

time(&seconds);

srand((unsigned int) seconds);

\_ax = RandomNumber2(\_Gfw-\_Giw, \_Giw+10);

\_az = RandomNumber2(\_Gfh-\_Gih, \_Gih+10);

}

int RandomNumber3(int high, int low)

{

return (rand() % (high+low));

}

void newFood3()

{

time\_t seconds;

time(&seconds);

srand((unsigned int) seconds);

\_cx = RandomNumber3(\_Gfw-\_Giw, \_Giw+10);

\_cz = RandomNumber3(\_Gfh-\_Gih, \_Gih+10);

}

int RandomNumber4(int high, int low)

{

return (rand() % (high-low));

}

void newFood4()

{

time\_t seconds;

time(&seconds);

srand((unsigned int) seconds);

\_dx = RandomNumber4(\_Gfw-\_Giw, \_Giw+10);

\_dz = RandomNumber4(\_Gfh-\_Gih, \_Gih+10);

}

//This Function will calculate the frame per second to display on the screen

void getFPS()

{

char tmp\_str[40];

if( GetTickCount() - g\_dwLastFPS >= 1000 ) // When A Second Has Passed...

{

g\_dwLastFPS = GetTickCount(); // Update Our Time Variable

g\_nFPS = g\_nFrames; // Save The FPS

g\_nFrames = 0; // Reset The FPS Counter

}

g\_nFrames++;

glRasterPos2f(75, 50);

sprintf(tmp\_str, "FPS: %d", g\_nFPS);

Write(tmp\_str);

glRasterPos2f(50, 60);

sprintf(tmp\_str, "Pos X: %d Pos Z: %d", \_x, \_z);

Write(tmp\_str);

}

void pnts() // sum of all points

{

points=points1+points2+points3+points4;

}

void level() // level increase according to points

{

pnts();

if((points1+points2+points3+points4)%20==0 && lvl<15)

{ lvl++; }

}

void Run2()

{

if((\_x == \_ax && \_z == \_az) ||

((\_x >= \_ax) && (\_x <= \_ax + 4) && (\_z >= \_az) && (\_z <= \_az + 4))||

((\_x <= \_ax) && (\_x >= \_ax - 4) && (\_z <= \_az) && (\_z >= \_az - 4))||

((\_x <= \_ax) && (\_x >= \_ax - 4) && (\_z >= \_az) && (\_z <= \_az + 4)) ||

((\_x >= \_ax) && (\_x <= \_ax + 4) && (\_z <= \_az) && (\_z >= \_az - 4)))

{

points2=points2+4;

if(points2 < 500) size++;

level();

newFood2();

}

}

void Run3()

{

if((\_x == \_cx && \_z == \_cz) ||

((\_x >= \_cx) && (\_x <= \_cx + 4) && (\_z >= \_cz) && (\_z <= \_cz + 4))||

((\_x <= \_cx) && (\_x >= \_cx - 4) && (\_z <= \_cz) && (\_z >= \_cz - 4))||

((\_x <= \_cx) && (\_x >= \_cx - 4) && (\_z >= \_cz) && (\_z <= \_cz + 4)) ||

((\_x >= \_cx) && (\_x <= \_cx + 4) && (\_z <= \_cz) && (\_z >= \_cz - 4)))

{

points3=points3+6;

if(points3 < 500) size++;

level();

newFood3();

}

}

void Run4()

{

if((\_x == \_dx && \_z == \_dz) ||

((\_x >= \_dx) && (\_x <= \_dx + 4) && (\_z >= \_dz) && (\_z <= \_dz + 4))||

((\_x <= \_dx) && (\_x >= \_dx - 4) && (\_z <= \_dz) && (\_z >= \_dz - 4))||

((\_x <= \_dx) && (\_x >= \_dx - 4) && (\_z >= \_dz) && (\_z <= \_dz + 4)) ||

((\_x >= \_dx) && (\_x <= \_dx + 4) && (\_z <= \_dz) && (\_z >= \_dz - 4)))

{

points4=points4+8;

if(points4 < 500) size++;

level();

newFood4();

}

}

//This Function will move the snake according to the directions

//Taken from the Keyboard keys

void Run(int value)

{

int i;

\_oldX[1] = \_x;

\_oldZ[1] = \_z;

switch(direction){

case RIGHT:

headRotation =90;

\_x += 6;

if(\_x > \_Gfw-2) \_x = \_Giw-1;//This will check if the snake is going into the border so it will appear on the other side

break;

case LEFT:

headRotation =-90;

\_x -= 6;

if(\_x < 0) \_x = \_Gfw-2;//This will check if the snake is going into the border so it will appear on the other side

break;

case UP:

headRotation =0;

\_z += 6;

if(\_z > \_Gfh-2) \_z = \_Gih-1;//This will check if the snake is going into the border so it will appear on the other side

break;

case Down:

headRotation =180;

\_z -= 6;

if(\_z < 2) \_z = \_Gfh-2;//This will check if the snake is going into the border so it will appear on the other side

break;

}

//Checks for Collisoin if yes Game Over

if(collision()) gameOver = true;

//Checks if the snake ate the food (check the X and Y)

// If yes it will increase the points & the size of the snake & create a new food

if((\_x == \_bx && \_z == \_bz) ||

((\_x >= \_bx) && (\_x <= \_bx + 4) && (\_z >= \_bz) && (\_z <= \_bz + 4)) ||

((\_x <= \_bx) && (\_x >= \_bx - 4) && (\_z <= \_bz) && (\_z >= \_bz - 4)) ||

((\_x <= \_bx) && (\_x >= \_bx - 4) && (\_z >= \_bz) && (\_z <= \_bz + 4)) ||

((\_x >= \_bx) && (\_x <= \_bx + 4) && (\_z <= \_bz) && (\_z >= \_bz - 4)))

{

points1=points1+2;

if(points1 < 500) size++;

level();

newFood();

}

Run2();

Run3();

Run4();

for(i = 0; i<size; i++){// Save the positions of the body parts

\_oldX[0] = \_oldX[1];

\_oldZ[0] = \_oldZ[1];

\_oldX[1] = bodyPos[0][i];

\_oldZ[1] = bodyPos[1][i];

bodyPos[0][i] = \_oldX[0];

bodyPos[1][i] = \_oldZ[0];

}

//Set the Timer

glutTimerFunc(130-lvl\*6, Run, 0);

}

void GameStatus()

{

char tmp\_str[40];// Print the status of the game on the screen

glColor3f(0, 0, 0);

glRasterPos2f(5, 50);

pnts();

sprintf(tmp\_str, "Level: %d Points: %d", lvl, points);

Write(tmp\_str);

}

void Display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);//Draw Function and Clear screen

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

if(EnableLight) initLight();

glTranslatef (-60.0, 40.0, zoom);

if(!gameOver)

//Check if the value of the Flag “Game Over is not True it will continue the game”

{

GameStatus();

DrawFood();

DrawFood2();

DrawFood3();

DrawFood4();

DrawSnake();

}

else

WelcomeScreen();

getFPS();

glutPostRedisplay();// Updates the screen

glutSwapBuffers();

}

void Special(int key, int x, int y) // for snake direction

{

switch(key)

{

case GLUT\_KEY\_RIGHT :

if(direction != LEFT)

direction = RIGHT;

break;

case GLUT\_KEY\_LEFT :

if(direction != RIGHT)

direction = LEFT;

break;

case GLUT\_KEY\_UP :

if(direction != UP)

direction = Down;

break;

case GLUT\_KEY\_DOWN :

if(direction != Down)

direction = UP;

break;

}

}

void keyboard (unsigned char key, int x, int y)

{

switch (key)//All Snake Rotation Over X, Y, Z angles

{

case 'x' : view\_rotx +=2;

glutPostRedisplay();

break;

case 'X' : view\_rotx -=2 ;

glutPostRedisplay();

break;

case 'y' : view\_roty +=2 ;

glutPostRedisplay();

break;

case 'Y' : view\_roty -=2 ;

glutPostRedisplay();

break;

case 'z' : view\_rotz +=2 ;

glutPostRedisplay();

break;

case 'Z' : view\_rotz -=2 ;

glutPostRedisplay();

break;

case 'a' : size++ ;

glutPostRedisplay();

break;

case 'A' : size-- ;

glutPostRedisplay();

break;

case '+' : zoom++ ;

glutPostRedisplay();

break;

case '-' : zoom-- ;

glutPostRedisplay();

break;

case 'n' : Reset() ;

glutPostRedisplay();

break;

case 27://ESC to Exit

exit(0);

break;

default:

break;

}

}

int main(void)

{

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(\_w,\_h);

glutInitWindowPosition(80,80);

glutCreateWindow("Snake Game - (DD)");

glutSpecialFunc(Special);

glutKeyboardFunc(keyboard);

glutDisplayFunc(Display);

glutReshapeFunc(resize);

newFood();

newFood2();

newFood3();

newFood4();

Run(0);

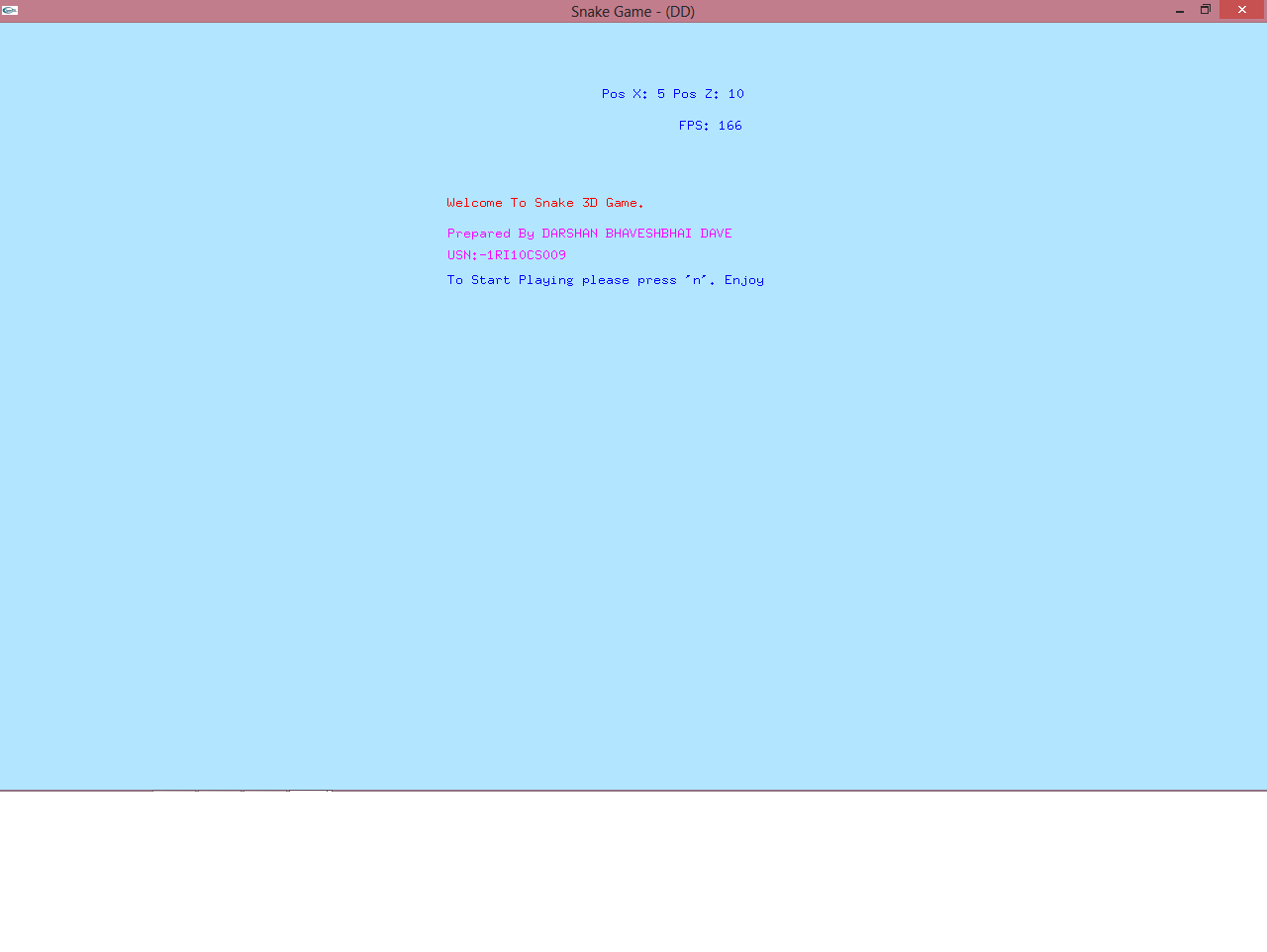
Initialize();

glutMainLoop();

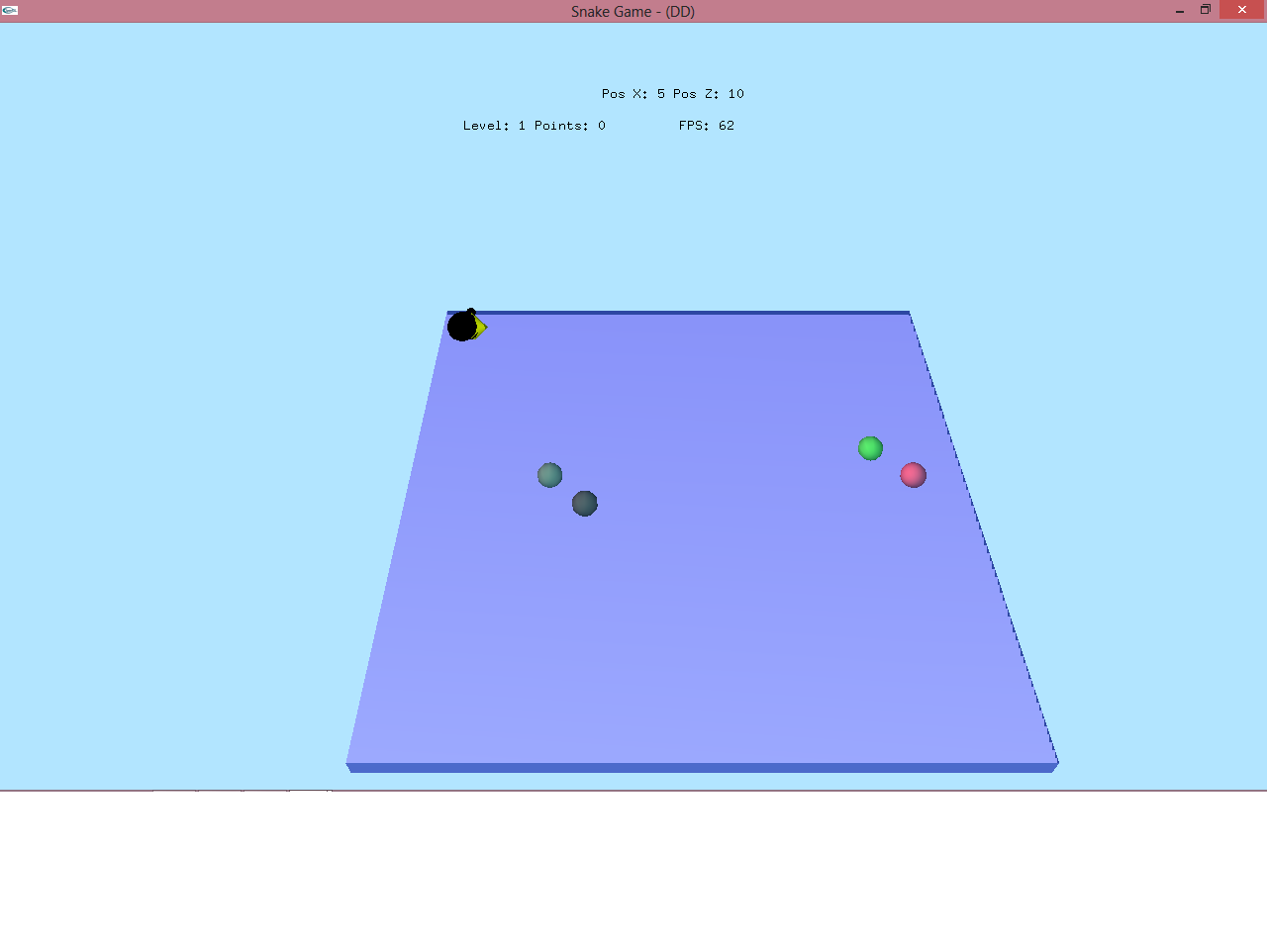
}

6. Snap Shots:-

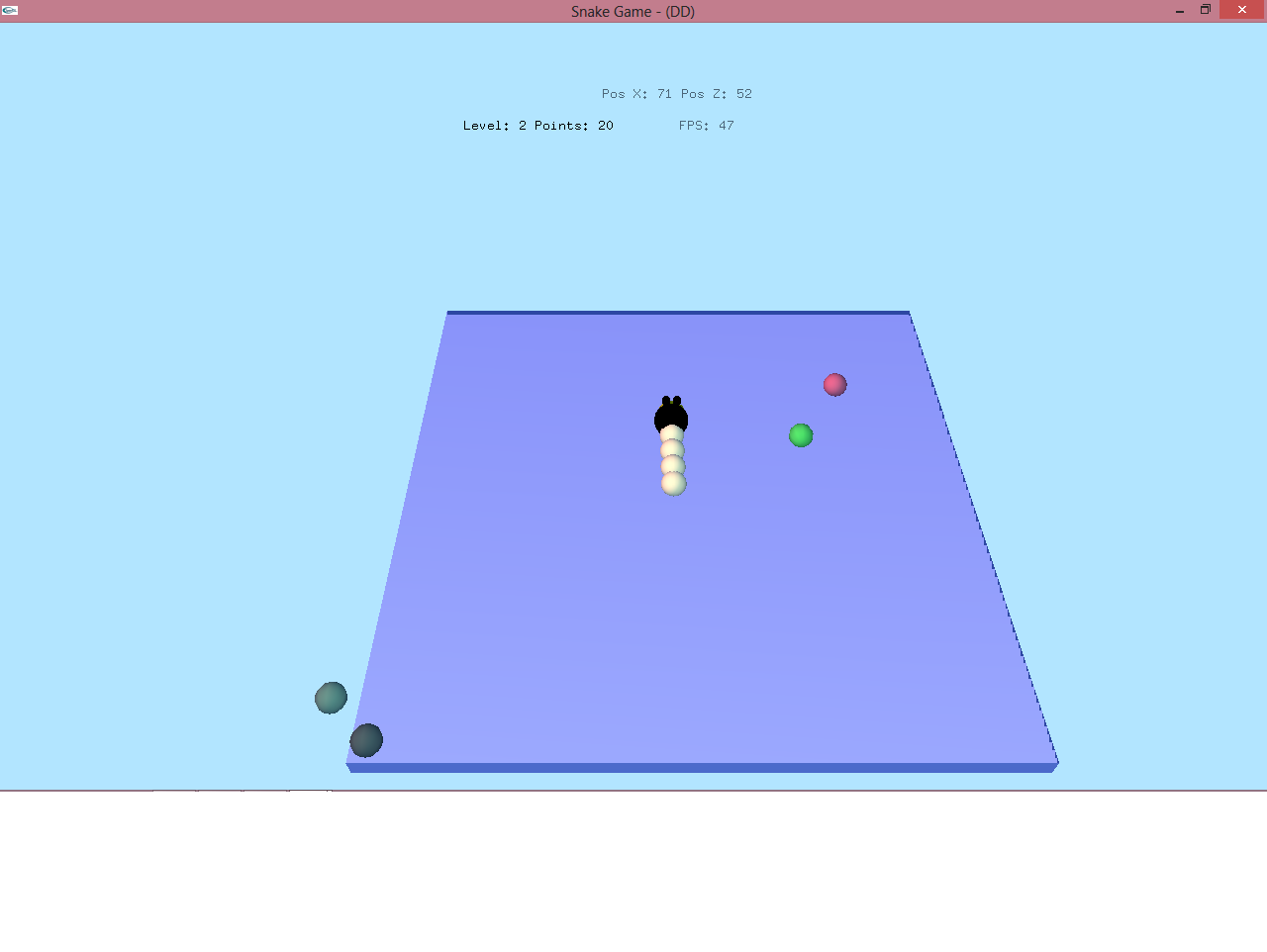
1. *Welcome* Screen:-



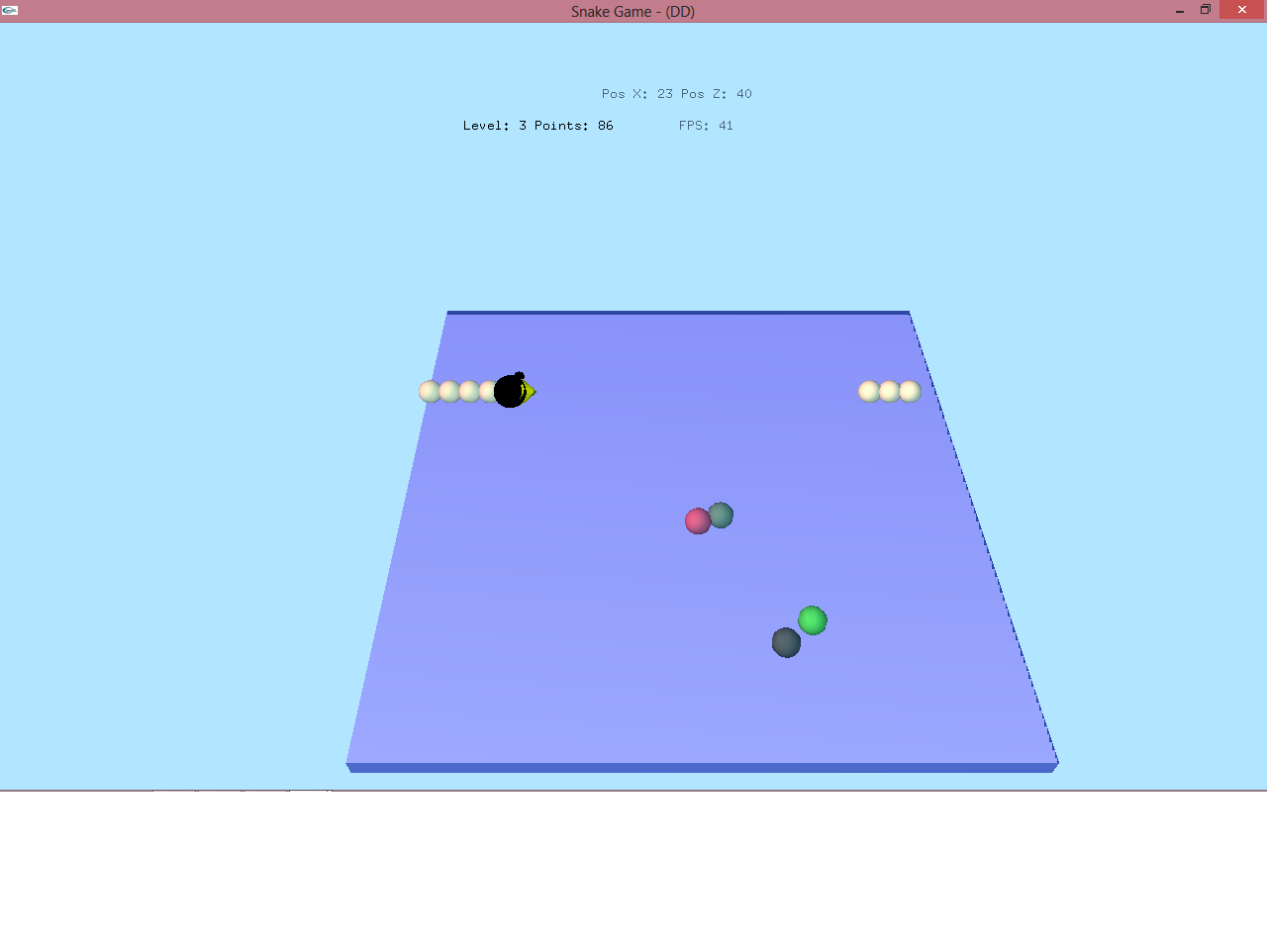
2.Initial Start:-

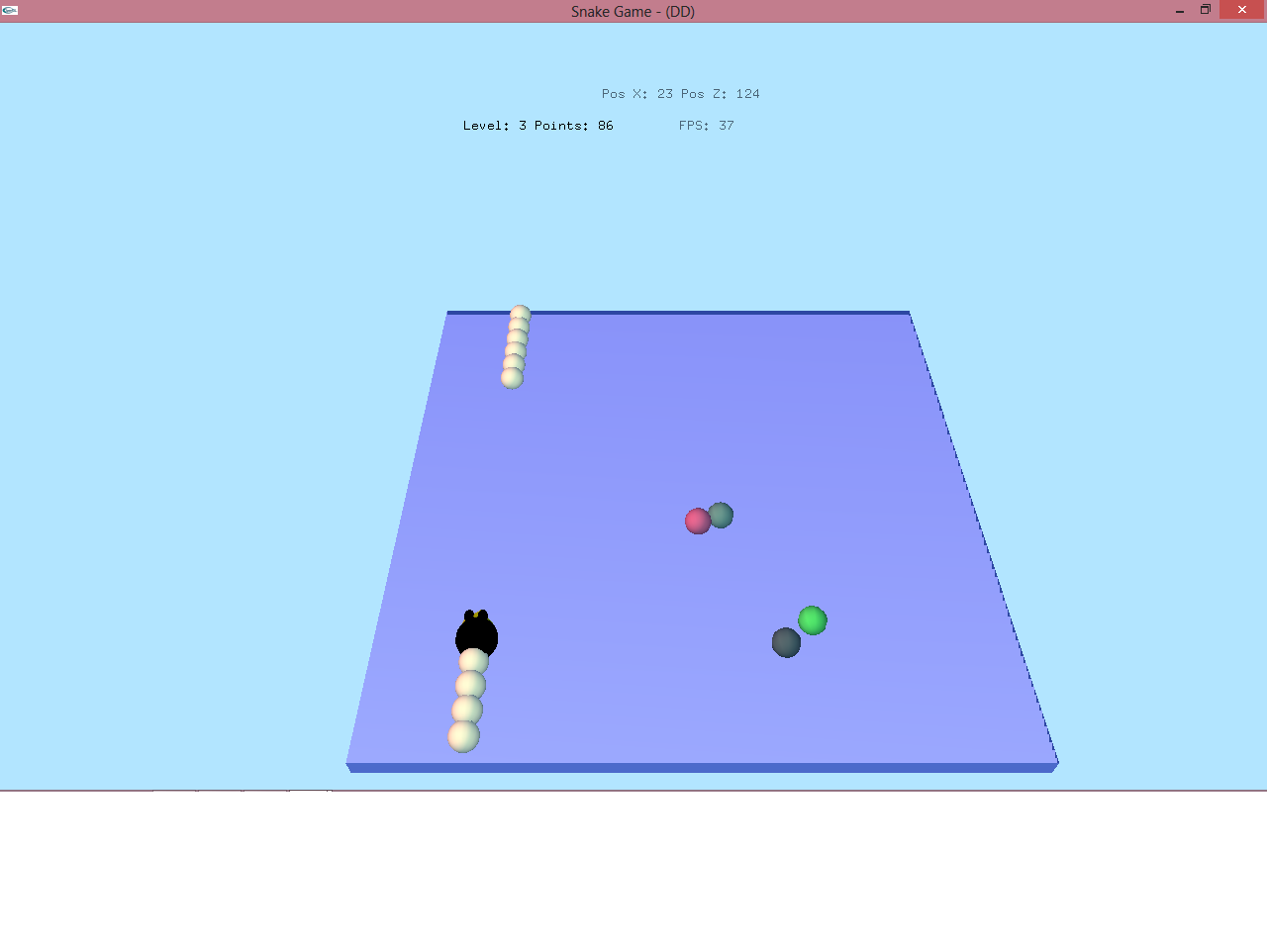


3.Level Up:-



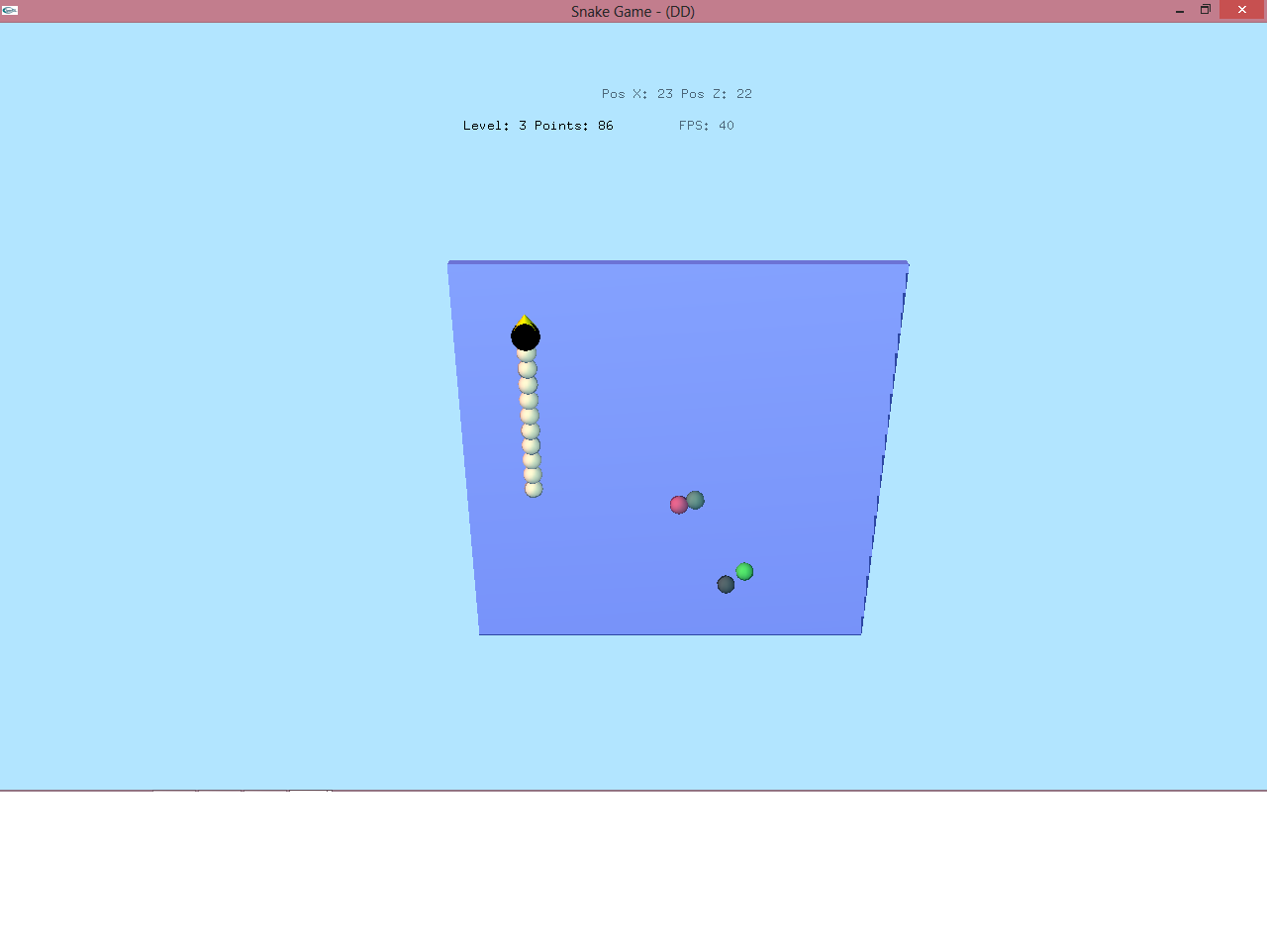
4. Different Postion of Snake:-



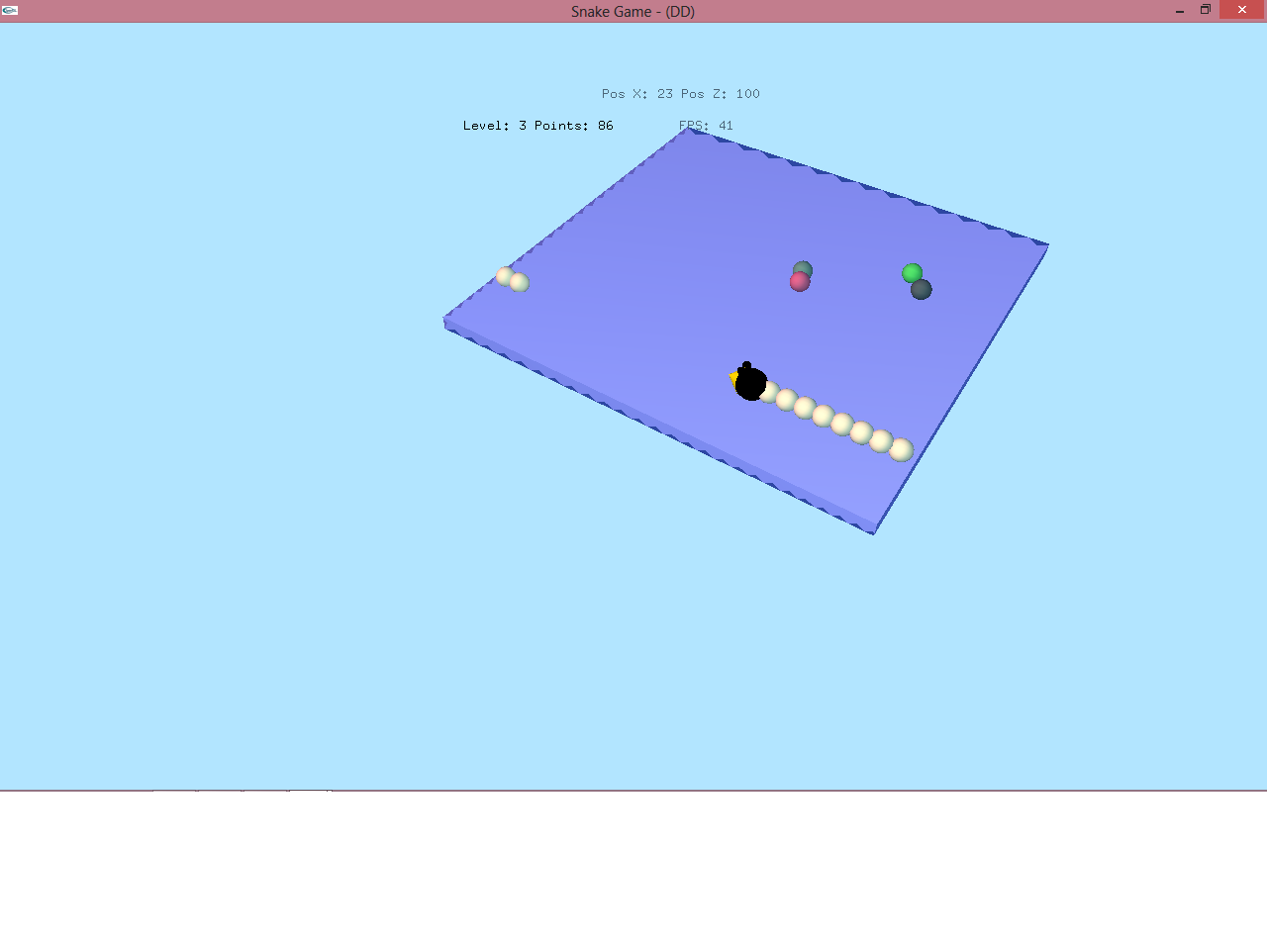


5. Different Angles:-

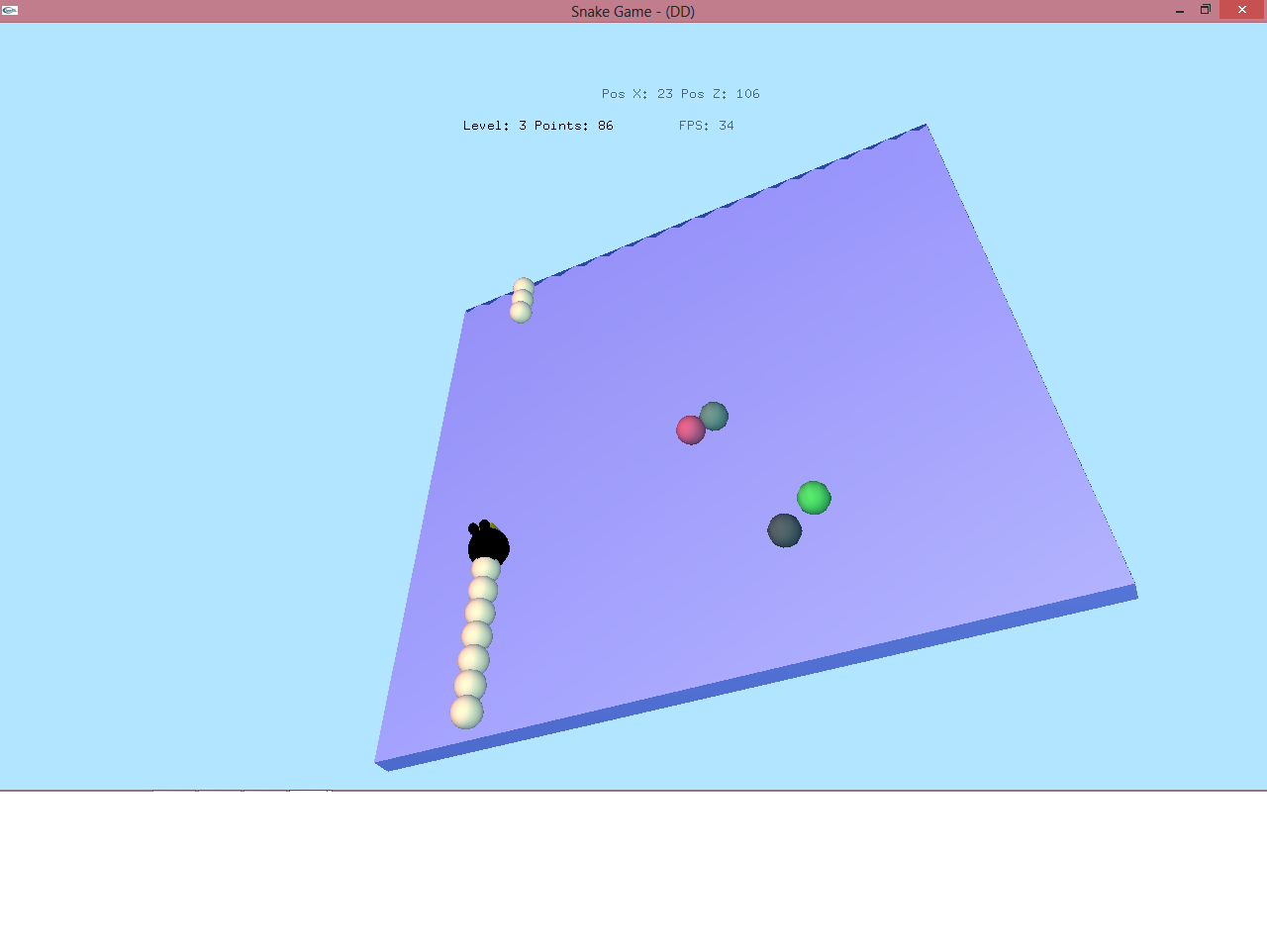
5.1 X -Axis:-



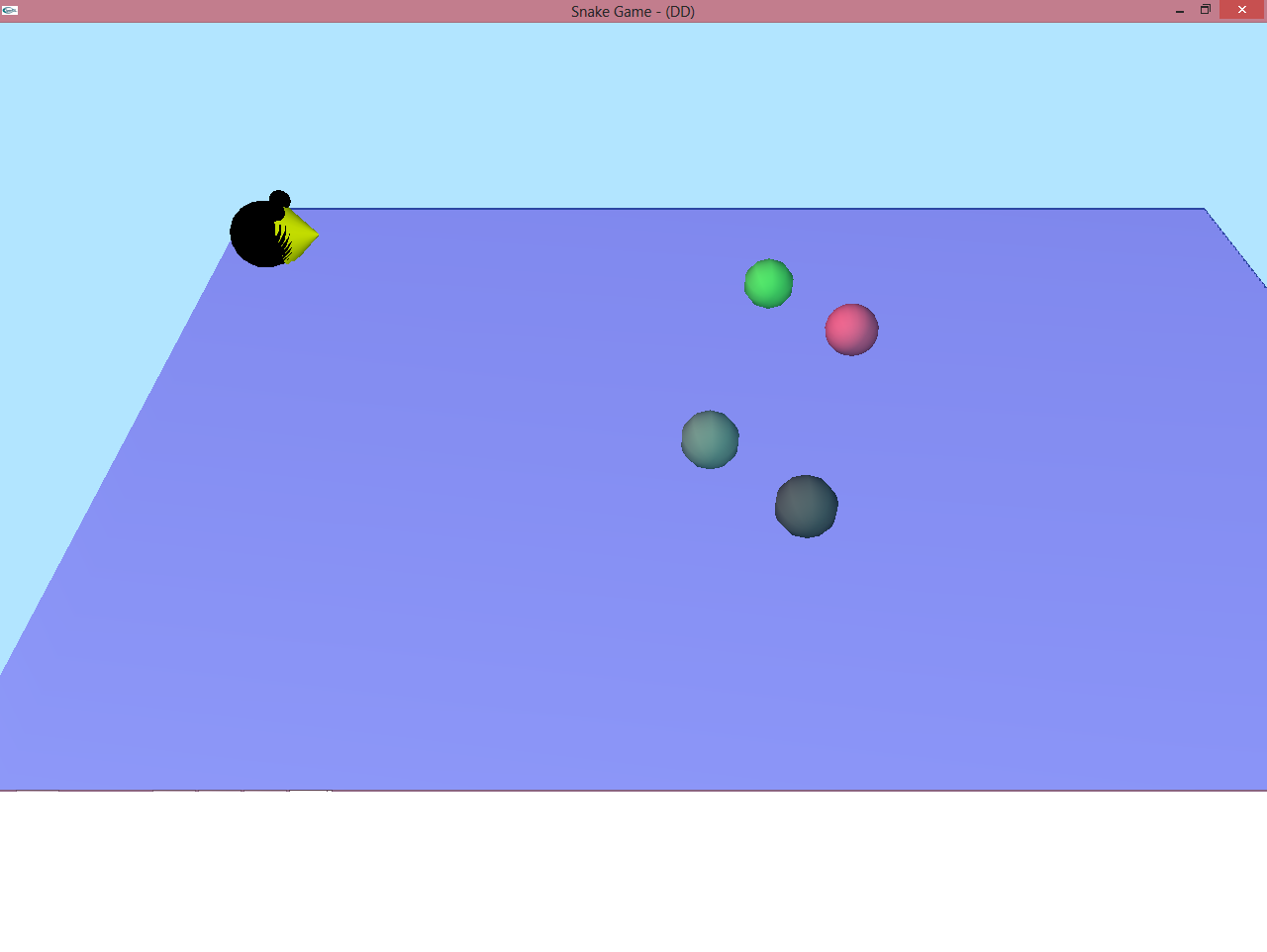
5.2 Y-Axis:-

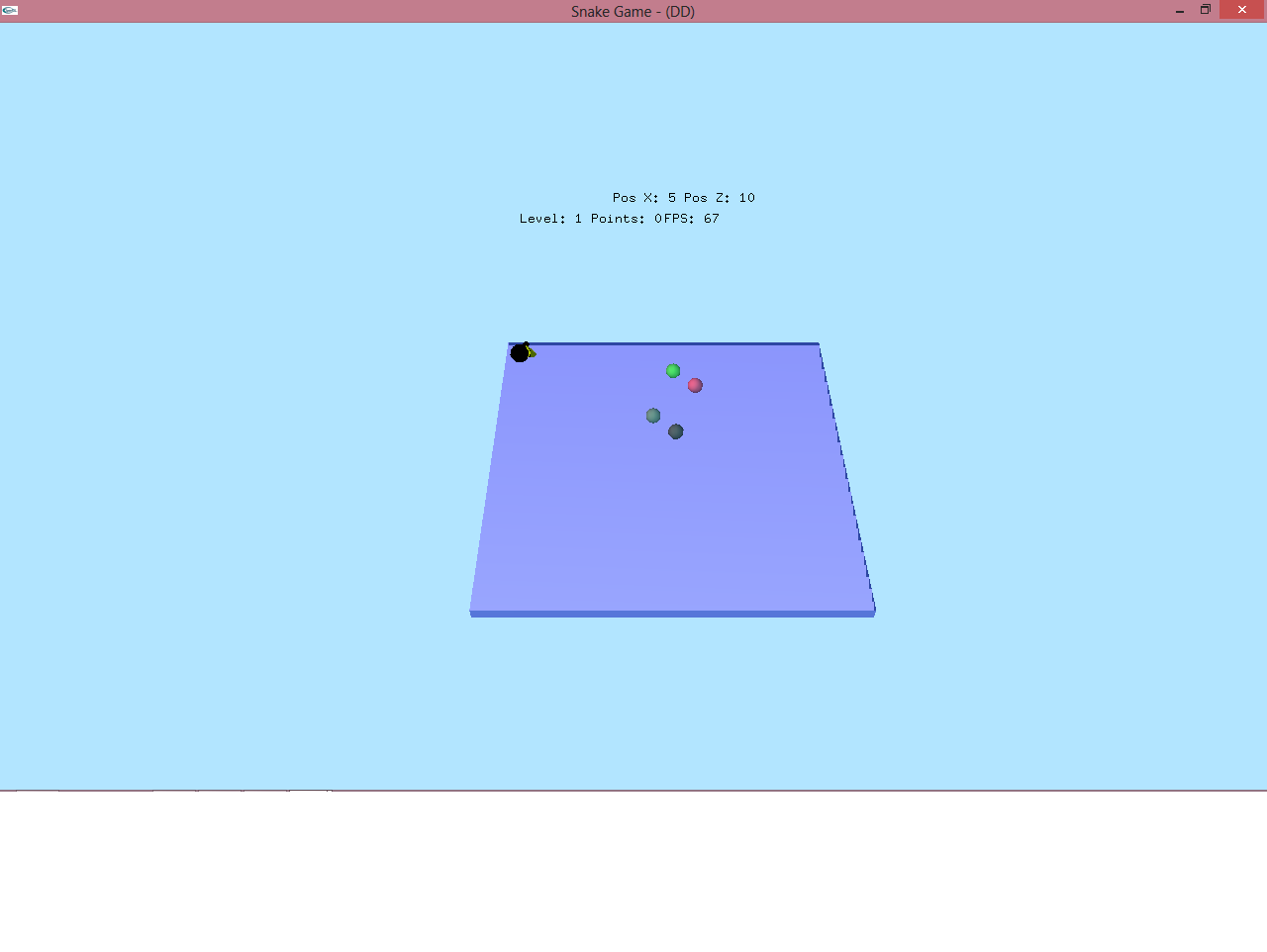


5.3 Z-Aixs:-

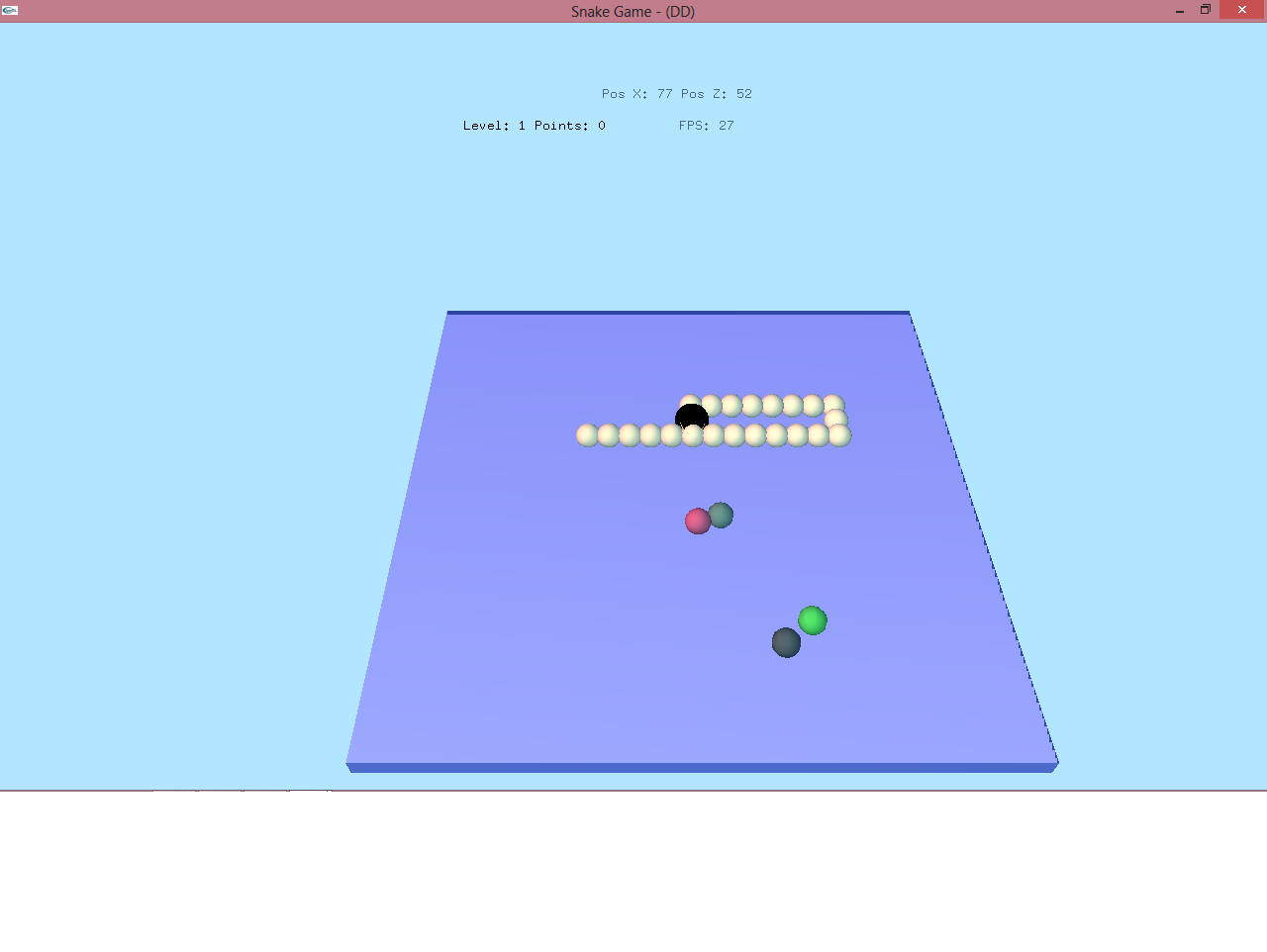


7. Zoom-IN and Zoom-Out:-





6. Game Over:-



7. Future Enhancement

* To get good knowledge about OpenGL for gaming, this project will help in future to get the logic for design and implementation for game.
* By modifying this project in future we can make good graphic and make more interactive with user and other devices.
* Game making is to develop such kind of software that use only for only entertainment purpose.
* The game you can use in mobile or computer and its very user friendly for both devices.
* Games are developed as a creative outlet and to generate profit. Development is normally funded by a publisher.

8. Conclusion

* This project “Snake Game” was designed and implemented by solely by its creators as the exercise of “Computer Graphics and Visualization” Laboratory.
* The concept used in the design are that our own, but certain referrals were made to my lectures and others regarding some technical issues. More features to make it a competitive.
* Further work on project like this would enable greater knowledge an prowess in OpenGL.
* This project is only for entertainment for children and it can be implement on Mobile devise also for entertainment.

9. Bibliography

The book and website that has helped us in implementing this project are as follows:

* Interactive Computer graphics -Edward angel.
* An Introduction to graphics programming with OpenGL -Toby Howard.
* Notes for a Computer Graphics Programming Course -Dr. Steve Cunningham.
* A Mathematical Introduction with OpenGL-S. Buss.
* [www.opengl.org](http://www.opengl.org)
* En.wikipedia.org/wiki/OpenGL
* msdn.microsoft.com