VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI – 590018

A Mini Project report on

"Robot Walking Simulator"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Engineering
In
Computer science & Engineering

Submitted by

ABHILASH N	**4KM20CS001**
DEEKSHITH	**4KM20CS011**
DEEKSHITHA K	**4KM20CS012**
PRASHITHA C P	**4KM20CS036**

Under the guidance of

Ms. JAYASHREE M H, M.Tech
Assistant Professor,KIT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

2022-23

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KARAVALI INSTITUTE OF TECHNOLOGY, MANGALORE – 575029

KARAVALI INSTITUTE OF TECHNOLOGY

NEERUMARGA, MANGALORE -575029

(Affiliated to VTU ,Belgaum and recognized by AICTE)

Department of Computer Science & Engineering

CERTIFICATE

It is certified that the mini project entitled **"Robot Walking Simulator"** , is carried out by **ABHILASH N (4KM20CS001) , DEEKSHITH (4KM20CS011) , DEEKSHITHA K (4KM20CS012) , PRASHITHA C P (4KM20CS036)** , the bona-fide students of sixth semester in Department of Computer Science and Engineering of VTU, Belagavi during year 2022-23.

Signature of the Guide

Signature of the H.O.D

Name of Examiners	Signature with Date
1.	1.
2.	2.

ACKNOWLDEGEMENT

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

We are grateful to our institution **KARAVALI INSTITUTE OF TECHNOLOGY** , **MANAGLORE** for providing us with the facilities which has made this project a success.

Our due thanks to **Dr. RAJENDRA PRASAD**, the principal, as we consider ourselves very lucky to have such an excellent computing facilities and their inspiration throughout ourprofessional course.

We express our sincere thanks and wholehearted gratitude to **Prof. MOHAMMED SABIT**, who is our respectable head of Department of Computer Science and guide for giving us constant encouragement, support and valuable guidance throughout the course of the project without whose stable guidance, this project would not have been achieved. We wish the acknowledge his help who made our task easy by providing with his valuable help and encouragement.

Our due thanks to **Asst.Prof.JAYASHREE M H**, as we consider ourself very lucky to have such an excellent computing facilities and their inspiration throughout our professional course.

We also thank the non-teaching staff of Computer science and department ,who guided us at the time of difficulties. Finally ,we thank God and those who are involved directly or indirectlyin completion of our project.

Place: **Mangalore** Date:

ABHILASH N	**DEEKSHITH**	**DEEKSHITHA K**	**PRASHITHA C P**
(4KM20CS001)	**(4KM20CS011)**	**(4KM20CS012)**	**(4KM20CS036)**

ABSTRACT

In this project, we implement both feature-extraction based algorithms and an end-to-end deep reinforcement learning method to learn to control Robot Walking Simulator directly from high-dimensional game screen input. Results show that compared with the pixel feature based algorithms, deep reinforcement learning is more powerful and effective. It leverages the high-dimensional sensory input directly and avoids potential errors in feature extraction. Finally, we propose special training methods to tackle class imbalance problems caused by the increase in game velocity. After training, our Deep-Q AI is able to outperform human experts.

CONTENTS

CHAPTER 1

INTRODUCTION

1. Introduction to Computer Graphics

Computer Graphics is concerned with all the aspects of producing pictures or images using a computer. The field began humbly almost fifty years ago with the display of few lines on a cathode ray tube (CRT), Now we can create images with computer that are indistinguishable from photographs of real objects.

The development of Computer Graphics has been driven by both the needs of the user community and by the advances in hardware and software. The applications of Computer Graphics are many and varied. However, we can divide them into 7 major areas -

- Display of Information.
- Design.
- Simulation and Animation.
- User Interfaces.
- Graphs and Charts, CAD, Data Visualizations.
- Image Processing, Education and Training, Entertainment etc**.**

2. Introduction to OpenGL

It is a software interface to graphics hardware. OpenGL is an industry standard portable 3-D Computer Graphics API. OpenGL is a premiere environment for developing portable interactive 2D or 3D applications.

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. It is also used in video games**,** here it competes with Direct3D on Microsoft Windows platforms (see Direct3D vs. OpenGL). OpenGL is managed by the non-profit technology consortium, the Khronos Group.

Functions in the main GL library have names that begin with the letters gl and are stored in a library usually referred to as GL. The second is the Open GL Utility Library(GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing.

To interface with the window system and to get input from external devices into our programs, we need at least one more library. For each major window system there is a system-specific library that provides the "glut" between the window system and open GL. For the X window system, this library is called as GLX, for Windows, it is wgl, and for the Macintosh, it is agl. Rather than using a different library for each system, we use a readily available library called the Open GL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in modern windowing system.

1. OpenGL for a developer

- OpenGL is not a PL but it contain pre-defined functionality.
- Provides functions to set or get or change the state variables.
- Provides functions to render scene onto a buffer which can then be shown in a window.
- Platform to create simple objects and animate them using various functions.
- Provides functions to develop more artistic options for a certain approach.
- Provides functions to explore multiple graphics related concepts.
- Platform Independent
- Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering.

CHAPTER 2

DESCRIPTION OF TOPIC AND INPUT KEYS

1. Program Definition

A robot walking simulator is a computer program or video game that simulates the walking or locomotion of a robot. It provides a virtual environment where users can control a robot's movements and explore its capabilities related to walking, navigating obstacles, and interacting with the surroundings.

2. Description

The simulator may incorporate realistic physics and mechanics to provide an authentic walking experience for the robot. It could also offer customization options, allowing users to modify the robot's design, behavior, or walking parameters.

3. Rules

- 1. Control Mechanism: The simulator will have a designated control mechanism through the keyboard player can control the robot's walking movements.

- 2. Walking Mechanics: The simulator will incorporate realistic walking mechanics for the robot. This includes factors such as balance, stability, stride length, leg coordination, and weight distribution. The player must use these mechanics effectively to control the robot's movements.

- 3. Customization Options: Many simulators allow players to customize their robot's design, appearance, or walking parameters. This could include selecting the Toggle Wireframe modifications.

- 4. Objectives and Goals: The simulator may have specific objectives or goals for the player to analyze the whole robot model while simulating.

These rules can serve as a starting point for a robot walking simulator, and additional rules can be added based on the desired gameplay experience and objectives of the game.

CHAPTER 3

REQUIREMENT SPECIFICATIONS

1. Hardware Requirements

- Processor : 1.5 GHz or faster processor.
- Processor Speed : Intel Pentium 4, Pentium M, Pentium D processor or better, or AMD K-8 (Athlon) or better.
- RAM : 256 MB internal RAM.
- Graphics Tools : OpenGL 2.0 and later.
- Windows 7, Windows Vista**, Windows XP Professional or Windows XP Home.
- NVIDIA GeForce FX 5200 or better graphics card.
- Monitor Resolution : A color monitor with a minimum resolution of 640*480

2. Software Requirements

- Operating System : MS-DOS based operating system like Windows 98.
- Other Software : C/C++.
- Graphics Tool : OpenGL.
- Alternative Graphics Tools:JAVA 3D,PHIGS,GKS

CHAPTER 4

ADVANTAGES AND DISADVANTAGES

1. Advantages of OpenGL

- It is by nature, not restricted to a single operating system.
- Offers quite a bit of low-level control for graphics.
- Supports a lot of programming languages.
- (Opinionated) People tend to say that it's easier to learn/get started with than other low-level 3D graphics-based APIs.
- Pretty stable.
- Is extensible.
- Graphics based software tend to use OpenGL for their 3D back end.

2. Disadvantages of OpenGL

- It's only a graphics API, meaning that it does not handle anything more than graphics. Audio, controls, logic, etc must be programmed in manually.
- Tends to run slightly slower than say DirectX because of OpenGL's robustness (and relative easiness), but of course, that's arguable. There is no one true benchmark of everything.

CHAPTER 5

OPENGL FUNCTIONS

1. GLUT Functions

- **glutInit()**

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

- **glutInitDisplayMode()**

  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.

- **glutInitWindowSize()**

  The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position.

- **glutCreateWindow()**

  glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name.

- **glutDisplayFunc()**

  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.

- **glutmainLoop()**

  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 call backs that have been registered.

- **glutSolidSphere()**

  glutSolidSphere renders a solid or wireframe sphere respectively.
  void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
  radius: The radius of the sphere.

Slices: The number of subdivisions around the Z axis (similar to lines of longitude).

Stacks: The number of subdivisions along the Z axis (similar to lines of latitude).

- **glutPostRedisplay()**

  glutPostRedisplay requests that the display call back be executed after the current call back returns.

- **glutSwapBuffers()**

  When we have two buffers front and back, the front buffer will always be displayed where as the back buffer in which we draw. Hence the rendering is put into the display callback, to update the back buffer. When the rendering is done glutSwapBuffers is executed and the results will be displayed.

- **glutKeyboardFunc()**

  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. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored.

- **glutIdleFunc()**

  glutIdleFunc 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.

2. GL Functions

- **glEnable()**

  void glEnable(GLenum cap);

  glEnable and glDisable enable and disable various capabilities. Use glIsEnabled or glGet to determine the current setting of any capability. The initial value for each capability with the exception of GL_DITHER is GL_TRUE.

- **glDepthFunc()**

void glDepthFunc(GLenum func);

glDepthFunc specifies the function used to compare each incoming pixel depth value with the depth value present in the depth buffer. The comparison is performed only if depth testing is enabled.

- **GL_PROJECTION()**

  Applies subsequent matrix operations to the projection matrix stack.

- **glLightfv()**

  void glLightfv(GLenum light, GLenum pname, GLfloat param);

  glLight sets the values of individual light source parameters.

  light names the light and is a symbolic name of the form GL_LIGHT i, where i ranges from 0 to the value of GL_MAX_LIGHTS-1. pname specifies one of ten light source parameters, again by symbolic name. params is either a single value or a pointer to an array that contains the new values.

- **glPushMatrix()**

  glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.

- **glClearColor()**

  void glClearColor(GLclampf r,GLclampf g,GLclampf b, GLclamp a);

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

- **glViewport()**

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

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

- **glFlush()**

void glFlush();

forces any buffered openGL commands to execute.

- **glScaled()**

  glScale produces a non-uniform scaling along the x, y, and z axis. The three parameters indicate the desired scale factor along each of the three axes.

- **glBegin()**

  This function specifies the type of the primitive that the vertices define. Every glBegin() ends with a corresponding glEnd(), which ends the list of vertices. The different primitives which we can use are GL_POLYGON, GL_LINE_STRIP, GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP etc.

- **glTranslatef()**

  It alters the current graphic image by post-multiplication. The matrix representing the image is post-multiplied.

CHAPTER 6

IMPLEMENTATION

```
#define SPHERE
#define COLOR
#define LIGHT
#define TORSO
#define HIP
#define SHOULDER
#define UPPER_ARM
#define LOWER_ARM
#define ROCKET_POD
#define UPPER_LEG
#define LOWER_LEG
#define NO_NORM
#define ANIMATION
#define DRAW_MECH
#define DRAW_ENVIRO
#define MOVE_LIGHT
/* end of compilation conditions */

/* start various header files needed */
#include <stdlib.h>
#include <math.h>
#define GLUT
```

```
#define GLUT_KEY
#define GLUT_SPEC
#include <GL/glut.h>
/* end of header files */
#define TEXTID      19
void DrawTextXY(double,double,double,double,char *);
/* start of display list definitions */
#define SOLID_MECH_TORSO        1
#define SOLID_MECH_HIP          2
#define SOLID_MECH_SHOULDER     3
#define SOLID_MECH_UPPER_ARM    4
#define SOLID_MECH_FOREARM  5
#define SOLID_MECH_UPPER_LEG                6
#define SOLID_MECH_FOOT         7
#define SOLID_MECH_ROCKET                   8
#define SOLID_MECH_VULCAN   9
#define SOLID_ENVIRO                        10
/* end of display list definitions */
/* start of motion  variables */
#ifndef M_PI
#define M_PI 3.14
#endif
```

```
GLUquadricObj *qobj;

char leg = 0;

int shoulder1 = 0, shoulder2 = 0, shoulder3 = 0, shoulder4 = 0, lat1 = 20, lat2 = 20,
  elbow1 = 0, elbow2 = 0, pivot = 0, tilt = 10, ankle1 = 0, ankle2 = 0, heel1 = 0,
  heel2 = 0, hip11 = 0, hip12 = 10, hip21 = 0, hip22 = 10, fire = 0, solid_part = 0,
  anim = 0, turn = 0, turn1 = 0, lightturn = 0, lightturn1 = 0;

float elevation = 0.0, distance = 0.0, frame = 3.0
 /* foot1v[] = {} foot2v[] = {} */ ;

/* end of motion variables */

/* start of material definitions */
#ifdef LIGHT        // to change the color of robot box
GLfloat mat_specular[] = {0.0, 0.0, 1.0, 1.0};
GLfloat mat_ambient[] ={0.0, 0.0, 1.0, 1.0};
GLfloat mat_diffuse[] ={0.0, 0.0, 1.0, 1.0};
GLfloat mat_shininess[] ={128.0 * 0.4};

GLfloat mat_specular2[] ={0.508273, 0.508273, 0.508373};
GLfloat mat_ambient2[] ={0.19225, 0.19225, 0.19225};
GLfloat mat_diffuse2[] ={0.50754, 0.50754, 0.50754};
```

```
GLfloat mat_shininess2[] ={128.0 * 0.6};
//to change the wall colorfffffffff
GLfloat mat_specular3[] = {1.0, 1.0, 0.0};
GLfloat mat_ambient3[] ={1.0, 1.0, 0.0};
GLfloat mat_diffuse3[] ={1.0, 1.0, 0.0};
GLfloat mat_shininess3[] ={0.0 * 0.0};
//to change the plateform color
GLfloat mat_specular4[] = {0.633, 0.727811, 0.633};
GLfloat mat_ambient4[] = {0.0215, 0.1745, 0.0215};
GLfloat mat_diffuse4[] = {0.07568, 0.61424, 0.07568};
GLfloat mat_shininess4[] = {128 * 0.6};
GLfloat mat_specular5[] =
{0.60, 0.60, 0.50};
GLfloat mat_ambient5[] =
{0.0, 0.0, 0.0};
GLfloat mat_diffuse5[] =
{0.5, 0.5, 0.0};
GLfloat mat_shininess5[] =
{128.0 * 0.25};
#endif
/* end of material definitions */

/* start of the body motion functions */
void Heel1Add(void)
{
```

```
heel1 = (heel1 - 3) % 360;
}void Heel2Add(void)
{  heel2 = (heel2 + 3) % 360;
}void Heel2Subtract(void)
{  heel2 = (heel2 - 3) % 360;
}void Ankle1Add(void)
{  ankle1 = (ankle1 + 3) % 360;
}void Ankle1Subtract(void)
{ankle1 = (ankle1 - 3) % 360;
}void Ankle2Add(void)
{ ankle2 = (ankle2 + 3) % 360;
}void Ankle2Subtract(void)
{  ankle2 = (ankle2 - 3) % 360;
}void RotateAdd(void)
{  pivot = (pivot + 3) % 360;
}void RotateSubtract(void)
{  pivot = (pivot - 10) % 360;
}void MechTiltSubtract(void)
{  tilt = (tilt - 10) % 360;
}void MechTiltAdd(void)
{  tilt = (tilt + 10) % 360;
}void elbow1Add(void)
{elbow1 = (elbow1 + 2) % 360;
}void elbow1Subtract(void)
{
```

```
{elbow1 = (elbow1 - 2) % 360;
}void elbow2Add(void)
{  elbow2 = (elbow2 + 2) % 360;
}void elbow2Subtract(void)
{  elbow2 = (elbow2 - 2) % 360;
}void shoulder1Add(void)
{  shoulder1 = (shoulder1 + 5) % 360;
}void shoulder1Subtract(void)
{  shoulder1 = (shoulder1 - 5) % 360;
}void shoulder2Add(void)
{  shoulder2 = (shoulder2 + 5) % 360;
}void shoulder2Subtract(void){
 shoulder2 = (shoulder2 - 5) % 360;
}void shoulder3Add(void)
{  shoulder3 = (shoulder3 + 5) % 360;
}void shoulder3Subtract(void)
{  shoulder3 = (shoulder3 - 5) % 360;
}void shoulder4Add(void)
{  shoulder4 = (shoulder4 + 5) % 360;
}void shoulder4Subtract(void)
{  shoulder4 = (shoulder4 - 5) % 360;
}void lat1Raise(void)
{lat1 = (lat1 + 5) % 360;
}void lat1Lower(void){
```

```
lat1 = (lat1 - 5) % 360;
}void lat2Raise(void)
{  lat2 = (lat2 + 5) % 360;
}void lat2Lower(void)
{  lat2 = (lat2 - 5) % 360;
}void FireCannon(void)
{  fire = (fire + 20) % 360;
}void RaiseLeg1Forward(void)
{  hip11 = (hip11 + 3) % 360;
}void LowerLeg1Backwards(void)
{  hip11 = (hip11 - 3) % 360;
}void RaiseLeg1Outwards(void)
{ hip12 = (hip12 + 10) % 360;
}void LowerLeg1Inwards(void)
{  hip12 = (hip12 - 10) % 360;
}void RaiseLeg2Forward(void)
{  hip21 = (hip21 + 3) % 360;
}void LowerLeg2Backwards(void)
{  hip21 = (hip21 - 3) % 360;
}void RaiseLeg2Outwards(void)
{  hip22 = (hip22 + 10) % 360;
}void LowerLeg2Inwards(void)
{  hip22 = (hip22 - 10) % 360;
}/* end of body motion functions */
```

```
/* start of light source position functions */
void TurnRight(void)
{  turn = (turn - 10) % 360;
}void TurnLeft(void)
{  turn = (turn + 10) % 360;
}void TurnForwards(void)
{  turn1 = (turn1 - 10) % 360;
}void TurnBackwards(void)
{  turn1 = (turn1 + 10) % 360;
}void LightTurnRight(void)
{  lightturn = (lightturn + 10) % 360;
}void LightTurnLeft(void){
{  lightturn = (lightturn - 10) % 360;
}void LightForwards(void)
{  lightturn1 = (lightturn1 + 10) % 360;
}void LightBackwards(void)
{lightturn1 = (lightturn1 - 10) % 360;}
void DrawTextXY(double x,double y,double
z,double scale,char *s)
{int i;glPushMatrix();
   glTranslatef(x,y,z);
   glScalef(scale,scale,scale);
   for (i=0;i<strlen(s);i++)

glutStrokeCharacter(GLUT_STROKE_ROMAN,s[
i]);
```

```
glPopMatrix();

}

void Box(float width, float height, float depth, char solid)
void DrawTextXY(double x,double y,double z,double scale,char *s)
{int i;
  glPushMatrix();
  glTranslatef(x,y,z);
  glScalef(scale,scale,scale);
  for (i=0;i<strlen(s);i++)
glutStrokeCharacter(GLUT_STROKE_ROMAN,s[i]);
 glPopMatrix();
}/* start of geometric shape functions */
void Box(float width, float height, float depth, char solid)
{  char i, j = 0;
  float x = width / 2.0, y = height / 2.0, z = depth / 2.0;
 for (i = 0; i < 4; i++) {
   glRotatef(90.0, 0.0, 0.0, 1.0);
   if (j) {
     if (!solid)
       glBegin(GL_LINE_LOOP);
     else
       glBegin(GL_QUADS);
     glNormal3f(-1.0, 0.0, 0.0);
     glVertex3f(-x, y, z);
     glVertex3f(-x, -y, z);
     glVertex3f(-x, -y, -z);
     glVertex3f(-x, y, -z);
     glEnd();
     if (solid) {
       glBegin(GL_TRIANGLES);
       glNormal3f(0.0, 0.0, 1.0);
       glVertex3f(0.0, 0.0, z);
       glVertex3f(-x, y, z);
}
glVertex3f(-x, -y, z);
       glNormal3f(0.0, 0.0, -1.0);
       glVertex3f(0.0, 0.0, -z);
       glVertex3f(-x, -y, -z);
```

```
glVertex3f(-x, y, -z);
      glEnd();
j = 0;
   } else {
     if (!solid)
      glBegin(GL_LINE_LOOP);
     else
      glBegin(GL_QUADS);
     glNormal3f(-1.0, 0.0, 0.0);
     glVertex3f(-y, x, z);
     glVertex3f(-y, -x, z);
     glVertex3f(-y, -x, -z);
     glVertex3f(-y, x, -z);
     glEnd();
     if (solid) {
      glBegin(GL_TRIANGLES);
      glNormal3f(0.0, 0.0, 1.0);
      glVertex3f(0.0, 0.0, z);
      glVertex3f(-y, x, z);
      glVertex3f(-y, -x, z);
      glNormal3f(0.0, 0.0, -1.0);
      glVertex3f(0.0, 0.0, -z);
      glVertex3f(-y, -x, -z);
      glVertex3f(-y, x, -z);
      glEnd();
     }
     j = 1;}}}
 lightturn = (lightturn - 10) % 360;
}void LightForwards(void)
{  lightturn1 = (lightturn1 + 10) % 360;
}void LightBackwards(void)
{lightturn1 = (lightturn1 - 10) % 360;
}/* end of light source position functions */
void DrawTextXY(double x,double y,double z,double scale,char *s)
```

```
{
  int i;
 glPushMatrix();
  glTranslatef(x,y,z);
  glScalef(scale,scale,scale);
  for (i=0;i<strlen(s);i++)
 glutStrokeCharacter(GLUT_STROKE_ROMAN,s[i]);
 glPopMatrix();
}
 void animation_walk(void)


{
  float angle;
  static int step;
if (step == 0 || step == 2) {

   if (frame >= 0.0 && frame <= 21.0) {
    if (frame == 0.0)
     frame = 3.0;
    angle = (180 / M_PI) * (acos(((cos((M_PI / 180) * frame) * 2.043) + 1.1625) / 3.2059));
    if (frame > 0) {
     elevation = -(3.2055 - (cos((M_PI / 180) * angle) * 3.2055));
    } else
     elevation = 0.0;
    if (step == 0) {
     hip11 = -(frame * 1.7);
     if (1.7 * frame > 15)
       heel1 = frame * 1.7;
     heel2 = 0;
     ankle1 = frame * 1.7;
```

```
 if (frame > 0)
        hip21 = angle;
       else
        hip21 = 0;
       ankle2 = -hip21;
 shoulder1 = frame * 1.5;
       shoulder2 = -frame * 1.5;
       elbow1 = frame;
       elbow2 = -frame;
      } else {
       hip21 = -(frame * 1.7);
       if (1.7 * frame > 15)
        heel2 = frame * 1.7;
       heel1 = 0;
       ankle2 = frame * 1.7;
       if (frame > 0)
        hip11 = angle;
       else
        hip11 = 0;
       ankle1 = -hip11;
       shoulder1 = -frame * 1.5;
       shoulder2 = frame * 1.5;
       elbow1 = -frame;
       elbow2 = frame;
      }
      if (frame == 21)
       step++;
      if (frame < 21)frame = frame + 3.0;
    }
   }
   if (step == 1 || step == 3) {

    if (frame <= 21.0 && frame >= 0.0) {
     angle = (180 / M_PI) * (acos(((cos((M_PI / 180) * frame) * 2.043) + 1.1625) / 3.2029));
     if (frame > 0)
      elevation = -(3.2055 - (cos((M_PI / 180) * angle) * 3.2055));
     else
      elevation = 0.0;
     if (step == 1) {
```

```
elbow2 = hip11 = -frame;
      elbow1 = heel1 = frame;
      heel2 = 15;
      ankle1 = frame;
      if (frame > 0)
        hip21 = angle;
      else
        hip21 = 0;
      ankle2 = -hip21;
      shoulder1 = 1.5 * frame;
      shoulder2 = -frame * 1.5;
    } else {
      elbow1 = hip21 = -frame;
      elbow2 = heel2 = frame;
      heel1 = 15;
      ankle2 = frame;
      if (frame > 0)
        hip11 = angle;
      else
        hip11 = 0;
      ankle1 = -hip11;
      shoulder1 = -frame * 1.5;
      shoulder2 = frame * 1.5;
    }
    if (frame == 0.0)
      step++;
    if (frame > 0)
      frame = frame - 3.0;
   }
  }
  if (step == 4)
    step = 0;
  distance += 0.1678;
  glutPostRedisplay();
}

void animation(void)
{
  animation_walk();
}
```

```
 { int i = 0;
if (key == 27) exit (0);
  switch (key) {
    /* start arm control functions */
  case 'q':{shoulder2Subtract();
   i++;++;}
    break;case 'a':{shoulder2Add();i++;
    }break;case 'w':{shoulder1Subtract();
      i++;}break;case 's':{shoulder1Add();i++;
    } break;case '2':{shoulder3Add();i++;
   }break;case '1':{shoulder4Add();
      i++;}  break;
  case '4':{shoulder3Subtract();
      i++;} break;
case 'L':{Ankle1Subtract();
      i++;} break;
    /* end of leg control functions */
/* start of light source position functions */
  case 'p':{
      LightTurnRight();
      i++;}break;case 'i':{LightTurnLeft();
      i++;}break;case 'o':{
      LightForwards();

     glVertex2i(-9 + iy * 6, -9);
```

```
 i++; }break;case '9':{LightBackwards();
     i++;} break; /* end of light source position functions */
  }if (i)
    glutPostRedisplay();
}void special(int key, int x, int y)
    {int i = 0;switch (key) {
       /* start of view position functions */
      case GLUT_KEY_RIGHT:{
         TurnRight();
         i++;}break;case GLUT_KEY_LEFT:{
         TurnLeft();
         i++;}break;case GLUT_KEY_DOWN:{
         TurnForwards();
         i++;}break;
      case GLUT_KEY_UP:{
         TurnBackwards();
        i++;}break;
       /* end of view postions functions */
       /* start of misecllеneous functions */
      case GLUT_KEY_PAGE_UP:{
         FireCannon();
        i++; }break;
       /* end of miscelleneous functions */
      }if (i)glutPostRedisplay();}
```

```
void menu_select(int mode)
{
  switch (mode) {
#ifdef ANIMATION
  case 1:
   glutIdleFunc(animation);
    break;
#endif
  case 2:
    glutIdleFunc(NULL);
    break;
  case 3:
    Toggle();
    glutPostRedisplay();
    break;
  case 4:
    exit(EXIT_SUCCESS);
}
}/* ARGSUSED */
void null_select(int mode)
{
}

void glutMenu(void)
{int glut_menu[13];
  glut_menu[5] = glutCreateMenu(null_select);
  glutAddMenuEntry("forward       : q,w", 0);
  glutAddMenuEntry("backwards     : a,s", 0);
  glutAddMenuEntry("outwards      : z,x", 0);
  glutAddMenuEntry("inwards       : Z,X", 0);

  glut_menu[6] = glutCreateMenu(null_select);
  glutAddMenuEntry("upwards       : Q,W", 0);
  glutAddMenuEntry("downwards     : A,S", 0);
  glutAddMenuEntry("outwards      : 1,2", 0);
  glutAddMenuEntry("inwards       : 3,4", 0);

  glut_menu[1] = glutCreateMenu(null_select);
  glutAddMenuEntry(" : Page_up", 0);
```

```
glut_menu[8] = glutCreateMenu(null_select);
glutAddMenuEntry("forward       : y,u", 0);
glutAddMenuEntry("backwards     : h.j", 0);
glutAddMenuEntry("outwards      : Y,U", 0);
glutAddMenuEntry("inwards       : H,J", 0);

glut_menu[9] = glutCreateMenu(null_select);
glutAddMenuEntry("forward       : n,m", 0);
glutAddMenuEntry("backwards     : N,M", 0);


glut_menu[10] = glutCreateMenu(null_select);
glutAddMenuEntry("toes up       : K,L", 0);
glutAddMenuEntry("toes down     : k,l", 0);

glut_menu[11] = glutCreateMenu(null_select);
glutAddMenuEntry("right         : right arrow", 0);
glutAddMenuEntry("left          : left arrow", 0);
glutAddMenuEntry("down          : up arrow", 0);
glutAddMenuEntry("up            : down arrow", 0);

glut_menu[12] = glutCreateMenu(null_select);
glutAddMenuEntry("right         : p", 0);
glutAddMenuEntry("left          : i", 0);
```

```
glut_menu[7] = glutCreateMenu(NULL);
 glutAddSubMenu("at the bottompart? ", glut_menu[8]);
 glutAddSubMenu("at the knees?", glut_menu[9]);
 glutAddSubMenu("at the ankles? ", glut_menu[10]);
glut_menu[2] = glutCreateMenu(null_select);
 glutAddMenuEntry("turn left    : d", 0);
 glutAddMenuEntry("turn right    : g", 0);
 glutAddMenuEntry("Rocketpod    : v", 0);

 glut_menu[3] = glutCreateMenu(null_select);
 glutAddMenuEntry("tilt backwards : f", 0);
 glutAddMenuEntry("tilt forwards  : r", 0);

 glut_menu[0] = glutCreateMenu(NULL);
 glutAddSubMenu("move the arms.. ", glut_menu[4]);
 glutAddSubMenu("fire the vulcan guns?", glut_menu[1]);
 glutAddSubMenu("move the legs.. ", glut_menu[7]);
 glutAddSubMenu("move the torso?", glut_menu[2]);
 glutAddSubMenu("move the upper portion?", glut_menu[3]);
 glutAddSubMenu("rotate the scene..", glut_menu[11]);
#ifdef MOVE_LIGHT
 glutAddSubMenu("rotate the light source..", glut_menu[12]);
#endif
```

```
 glutCreateMenu(menu_select);
#ifdef ANIMATION
  glutAddMenuEntry("Start Walk", 1);
  glutAddMenuEntry("Stop Walk", 2);
#endif
  glutAddMenuEntry("Toggle Wireframe", 3);
  glutAddSubMenu("How do I ..", glut_menu[0]);
  glutAddMenuEntry("Quit", 4);
  glutAttachMenu(GLUT_LEFT_BUTTON);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

Main program

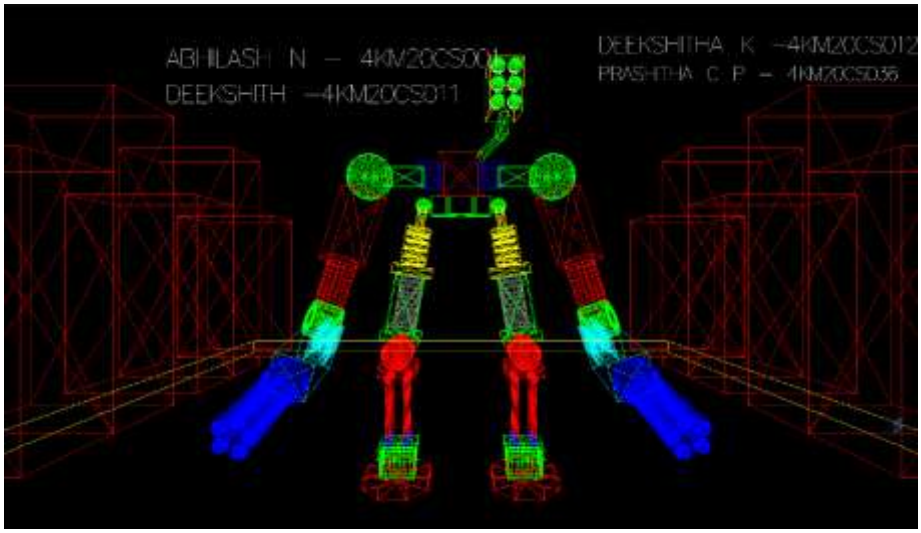
```
int main(int argc, char **argv)
{
#ifdef GLUT
  /* start of glut windowing and control functions */
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
  glutInitWindowSize(1000, 1000);
  glutCreateWindow("glutmech: Vulcan Gunner");
  myinit();
  glutDisplayFunc(display);
  glutReshapeFunc(myReshape);
#ifdef GLUT_KEY
```

```
glutKeyboardFunc(keyboard);

#endif

#ifdef GLUT_SPEC

  glutSpecialFunc(special);

#endif

 glutMenu();

                glPointSize(2.0);

  glutMainLoop();

  /* end of glut windowing and
control functions */

#endif

  return 0;             /* ANSI C requires
main to return int. */

}
```

Simulator

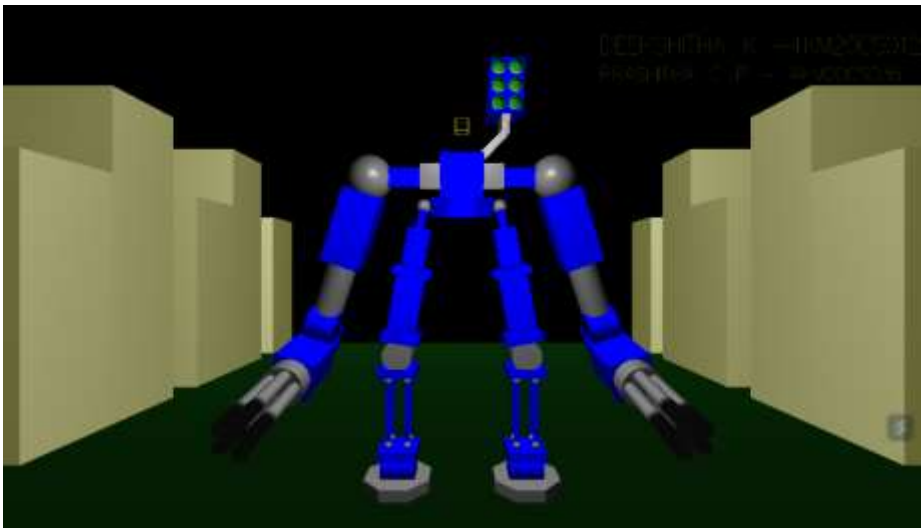
CHAPTER 7

SNAPSHOTS



Snapshot 7.1 Robot Walking Simulator

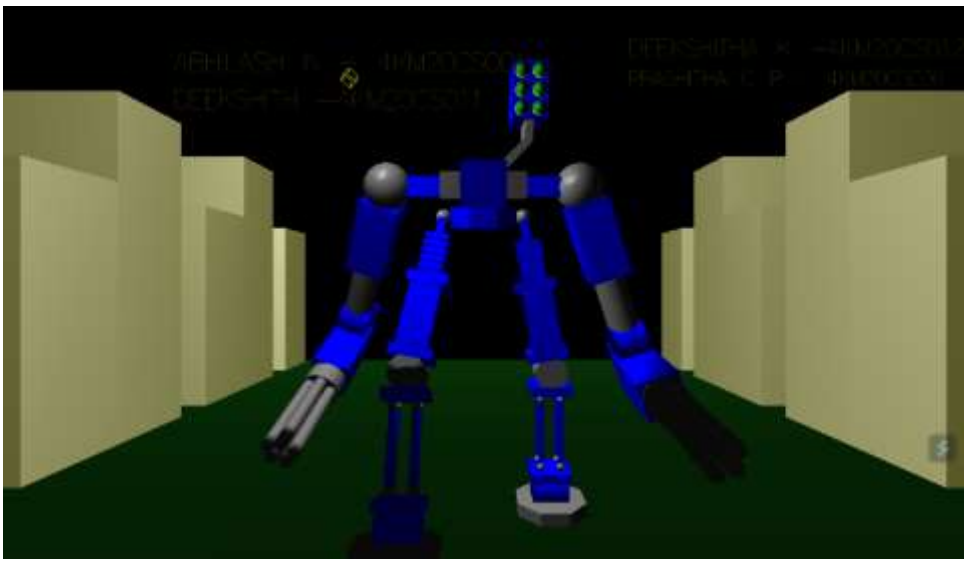
In this snapshot ,we can see the starting page of the game .To start right click and select start game.



Snapshot 7.2 Match won by Player

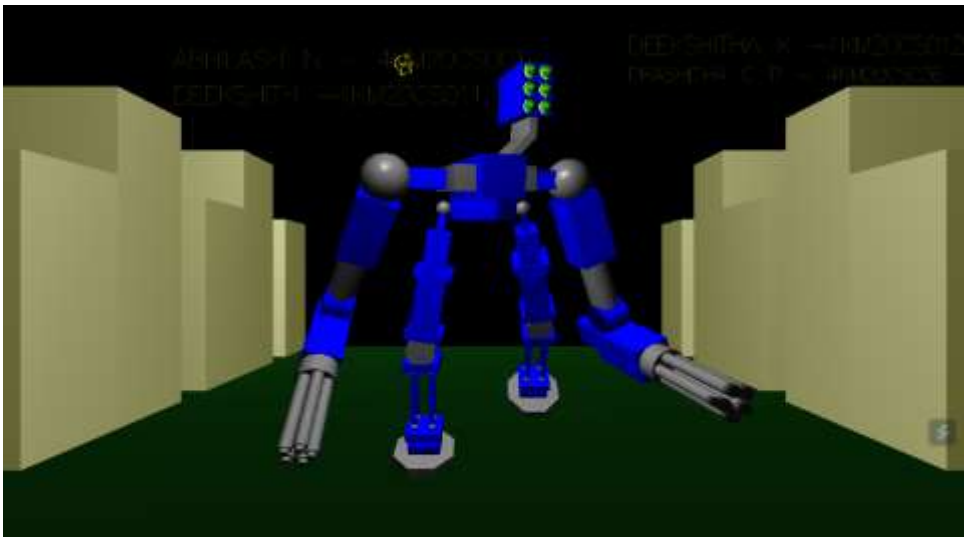
In this snapshot ,the player won the match.It shows Congratulations! You Won

Simulator



Snapshot 7.3 Match won by Computer

In this snapshot,the computer won the match.



Snapshot 7.4 Match is Draw

In this snapshot,the match is draw between the player and the computer .

CONCLUSION

As a part of our academics' mini project we have successfully implemented the concept of ray casting using OpenGL graphics package. We have tried to use the many of the graphics concepts using OpenGL such as translation, rotation, scaling, viewer motions etc. We would like to end by saying that, this project has helped us a lot to learn more about OpenGL ,OpenGL APIs and its functions. It was a very good experience in developing visual of Objects In Multimirror which helped us in improving our knowledge about OpenGL.

REFERENCES

1 Edward Angel, "Interactive Computer Graphics A top-down approach using OpenGL", 5th edition

2 Donald Hearn and Pauline Baker, "Computer Graphics- OpenGL Version", 3rd edition

3 F.S.Hill Jr, "Computer Graphics using OpenGL", 3rd edition

WEBSITES

1 www.OpenGL.org.Redbook

2 www.OpenGL.org.simpleexample

3 http://www.files32.com/Types-Of-Clipping-In-Computer-Graphics.asp

4 http://www.opengl.org/code