Course Project

Name : Abdelaziz Mohamed Abdelaziz.

I.D : 19015941.

Code flow :

- The code defines several functions, including randomFloat, checkSpheresIntersection, CraftCollision, Start_Stars, animate, setup, writeBitmapString, drawScene.

- The randomFloat function generates a random float value within a given range.

- The checkSpheresIntersection function checks for collision between two spheres using their positions and radii.

- The CraftCollision function checks for collision between a spacecraft and planets or the sun by iterating over the planets and calling the checkSpheresIntersection function.

- The Start_Stars function initializes the positions of stars by generating random coordinates within a specified range.

- The animate function is used to animate the rotation of planets around the sun and the moon around the Earth.

- The setup function is called to initialize the star positions, generate display lists for the spacecraft, sun, planets, and moon, and set up lighting.

- The writeBitmapString function is used to render text on the screen.

- The drawScene function is the main drawing routine. It sets up lighting, clears the buffers, and renders the stars, sun, planets, and spacecraft. It also handles viewport and camera settings.

- The code then calls the setup function to initialize the scene and starts the animation using animate.

- The drawScene function is called periodically to update the display and render the scene.

Screenshots of Code:

```
#define _USE_MATH_DEFINES
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <GL/glut.h>
#include <cmath>
#include <cstdint>

// Globals.
static int width, height; // Size of the OpenGL window.
static float angle = 270.0; // Angle of the spacecraft.
static float xVal = -100, zVal = 0; // Co-ordinates of the spacecraft.
static int isCollision = 0; // Is there collision between the spacecraft and an asteroid?
static unsigned int spacecraft; // Display lists base index.
static unsigned int sun;


static unsigned int planets[8]; // Array of display list indices for the eight planets.
static float planetDistances[9] = {10.0, 20.0, 30.0, 40.0, 50.0, 65.0, 78.0, 89.0,0.0};// Distances of each planet from the sun in arbit
static float planetSizes[9] = {2.382, 2.949, 3.0, 2.532, 6.209, 5.449, 4.007, 3.883,7.0}; // Sizes of each planet relative to the sun.
float planetAngles[9] = { 0 };
static float planetSpeeds[8] = { 2.15, 1.82, 1.50, 1.33, 1.1, 0.87, 0.75, 0.69 };
```

```
static float d = 1.0; // Diffuse and specular white light intensity.
static float m = 0.2;
static float p = 1.0; // Positional light?
static float t = 0.0;

static const float moonDistance = 5.0; // Distance between the Moon and the Earth in arbitrary units.
static const float moonSize = 0.57;
float moonAngle = 0.0;
static unsigned int moon;


const int numStars = 50;       // Number of stars
const float minCoord = -100.0; // Minimum coordinate value
const float maxCoord = 100.0;  // Maximum coordinate value
float starPositions[numStars][3];

static void * font = GLUT_BITMAP_8_BY_13; // Font selection
int animationDelay = 20000;
bool animates=1;
```

```
// this function is used to generate random places for stars.
float randomFloat(float minVal, float maxVal) {
    return minVal + static_cast <float> (rand()) / (static_cast <float> (RAND_MAX / (maxVal - minVal)));
}


//this function check collision between space craft and (planet or sun)
int checkSpheresIntersection(float x1, float y1, float z1, float r1,float x2, float y2, float z2, float r2)
{
    return ((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2) + (z1 - z2)*(z1 - z2) <= (r1 + r2)*(r1 + r2));
}
```

```
// this function we loop on planets and sun to check collision.
int CraftCollision(float x, float z, float a)
{
    int i, j;
    for (i = 0; i<9; i++)
        if (checkSpheresIntersection(x - 5 * sin((M_PI / 180.0) * a), 0.0,z - 5 * cos((M_PI / 180.0) * a), 7.072,
                                     planetDistances[i]*cos(planetAngles[i]*(M_PI / 180.0)), 0,
                                    planetDistances[i]*sin(planetAngles[i]*(M_PI / 180.0))*-1, planetSizes[i]))
        {

            return 1;
        }

    return 0;
}
```

```
// Function to initialize the star positions
void Start_Stars() {
    unsigned int seed = static_cast<unsigned>(time(0));
    for (int i = 0; i < numStars; i++) {
        starPositions[i][0] = randomFloat(minCoord, maxCoord); // X-coordinate
        starPositions[i][1] = randomFloat(minCoord, maxCoord); // Y-coordinate
        starPositions[i][2] = randomFloat(minCoord, maxCoord); // Z-coordinate
    }
}
```

```
// this function is used to animate in which planets rotate around sun.
void animate(int value)
{
        int i;
        if(animates)
        {
            for (i = 0; i < 8; i++) {
            planetAngles[i] += planetSpeeds[i]; // Update the angle for each planet based on its speed.
            if (planetAngles[i] > 360.0) {
                planetAngles[i] -= 360.0; // Keep the angle within the range [0, 360).
            }
        }// Update the angle for each planet based on its speed.
        moonAngle += 20;
        if (moonAngle > 360.0) {
            moonAngle -= 360.0;
        }
        glutPostRedisplay(); // Mark the current window as needing to be redisplayed.
        }
         glutTimerFunc(200, animate, 1);
}
```

```
void setup(void)
{

    int i, j;
    Start_Stars();
    spacecraft = glGenLists(1);
    sun = glGenLists(1);
    for (i = 0; i < 8; i++) {
        planets[i] = glGenLists(1);
    }

    //intilaze space craft.
    glNewList(spacecraft, GL_COMPILE);
    glPushMatrix();
    glRotatef(180.0, 0.0, 1.0, 0.0); // To make the spacecraft point down the $z$-axis initially.
    glColor3f(1.0, 1.0, 1.0);
    glutWireCone(5.0, 10.0, 10, 10);
    glPopMatrix();
    glEndList();


    //initilze the sun.
    glNewList(sun, GL_COMPILE);
    glColor3f(1.0, 1.0, 0.0); // Yellow color for the sun.
    glutSolidSphere(7.0, 20, 20); // Sphere with radius 5.0 and 20 slices/stacks.
    glEndList();
```

```
    //initilze planets .
    for (i = 0; i < 8; i++) {
        glNewList(planets[i], GL_COMPILE);
        glutSolidSphere(planetSizes[i], 20, 20); // Sphere with radius determined by planetSizes array.
        glEndList();
    }

    //initilze the moon.
    moon = glGenLists(1);
    glNewList(moon, GL_COMPILE);
    glutSolidSphere(moonSize, 20, 20); // Sphere with radius determined by moonSize constant.
    glEndList();


    //lighting.
    glEnable(GL_DEPTH_TEST);
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glEnable(GL_LIGHTING);
    float matAmbAndDif[] = { 0.0, 0.0, 1.0, 1.0 };
    float matSpec[] = { 1.0, 1.0, 1,0, 1.0 };
    float matShine[] = { 50.0 };
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, matAmbAndDif);
    glMaterialfv(GL_FRONT, GL_SHININESS, matShine);


    animate(1);
}
```

```
// this function is used to write text when space craft crashed.
void writeBitmapString(void *font, char *string)
{
    char *c;
    for (c = string; *c != '\0'; c++) glutBitmapCharacter(font, *c);
}
```

```
// Drawing routine.
void drawScene(void)
{
    // lighting
    int i, j;
    float lightAmb[] = { 0.0, 0.0, 0.0, 1.0 };
    float lightDifAndSpec0[] = { d, d, d, 1.0 };
    float lightPos0[] = { 0.0, 0.0, 0.0, p };
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDifAndSpec0);
    glEnable(GL_LIGHT0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


    // Beg  in left viewport.
    glViewport(0, 0, width , height);//demo
    glLoadIdentity();
    // Write text in isolated (i.e., before gluLookAt) translate block.
    gluLookAt(xVal - 7 * sin((M_PI / 180.0) * angle),
              0.0,
              zVal - 7 * cos((M_PI / 180.0) * angle),
              xVal - 11 * sin((M_PI / 180.0) * angle),
              0.0,
              zVal - 11 * cos((M_PI / 180.0) * angle),
              0.0,
              1.0,
              0.0);
```

```
glDisable(GL_LIGHTING);
// stars.
glColor3f(1.0, 1.0, 1.0);
glPointSize(2.0);
glBegin(GL_POINTS);
for (int i = 0; i < 11000; i++) {
    glVertex3fv(starPositions[i]);
}
glEnd();

// Draw the sun using its display list.
glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);
glPushMatrix();
glTranslatef(0.0, 0.0, 0.0);
glCallList(sun);
glPopMatrix();
glEnable(GL_LIGHTING);


GLfloat planetColors[9][4] = {
        { 0.545, 0.494, 0.462 ,1.0 }, // Mercury
        { 0.871, 0.788, 0.604 ,1.0}, // Venus
        { 0.196, 0.388, 0.686 , 1.0 }, // Earth
        { 0.706, 0.365, 0.161 ,1.0}, // Mars
        { 0.718, 0.631, 0.545 ,1.0}, // Jupiter
        { 0.659, 0.592, 0.533,1.0 }, // Saturn
        { 0.494, 0.733, 0.816 ,1.0}, // Uranus
```

```
// draw planets .
for (i = 0; i < 8; i++) {
    if(i==2)
    {
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, planetColors[i]);
        glPushMatrix();
        glRotatef(planetAngles[i], 0.0, 1.0, 0.0);  // Rotate the planet around the sun.
        glTranslatef(planetDistances[i], 0.0, 0.0); // Translate along the x-axis to position the planet at the correct distance from t
        glCallList(planets[i]);
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, planetColors[8]);
        glPushMatrix();
        glRotatef(moonAngle, 0.0, 1.0, 0.0); // Rotate the Moon around the Earth.
        glTranslatef(moonDistance, 0.0, 0.0); // Translate along the x-axis to position the Moon at the correct distance from the Earth
        glCallList(moon);
        glPopMatrix();
        glPopMatrix();
    }
    else
    {
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, planetColors[i]);
        glPushMatrix();
        glRotatef(planetAngles[i], 0.0, 1.0, 0.0);  // Rotate the planet around the sun.
        glTranslatef(planetDistances[i], 0.0, 0.0); // Translate along the x-axis to position the planet at the correct distance from t
        if(i==5)
        {
            glRotatef(70, 1.0, 0.0, 0.0);
            glutSolidTorus(0.5,7.0,3,30);

        }
```

```
        ,
        else
        {
            glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, planetColors[i]);
            glPushMatrix();
            glRotatef(planetAngles[i], 0.0, 1.0, 0.0);  // Rotate the planet around the sun.
            glTranslatef(planetDistances[i], 0.0, 0.0); // Translate along the x-axis to position the planet at the correct distar
            if(i==5)
            {
                glRotatef(70, 1.0, 0.0, 0.0);
                glutSolidTorus(0.5,7.0,3,30);

            }
            glCallList(planets[i]);
            glPopMatrix();
        }
    }


    // draw space craft.
    glPushMatrix();
    glTranslatef(xVal, 0.0, zVal);
    glRotatef(angle, 0.0, 1.0, 0.0);
    glCallList(spacecraft);
    glPopMatrix();
    // Fixed camera.
    glPushMatrix();
    glColor3f(1.0, 0.0, 0.0);
```

```
    if(isCollision == 1)
    {
        writeBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, "Game Over - Space Craft Crashed - would you like to contuine??");
    }
    else
    {
         animates = 1;
    }
   glPopMatrix();

    // Draw spacecraft.

    // End left viewport.

    // Begin right viewport.
    glViewport(width*0.75, 0, width/3, height/3);
    glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, t);
    glLoadIdentity();

    // Write text in isolated (i.e., before gluLookAt) translate block.
    glPushMatrix();
    glColor3f(1.0, 0.0, 0.0);
    glRasterPos3f(-28.0, 25.0, -30.0);
    glPopMatrix();

    // Draw a vertical line on the left of the viewport to separate the two viewports
    glColor3f(1.0, 1.0, 1.0);
    glLineWidth(2.0);
    glBegin(GL_LINES);
```

```
glColor3f(1.0, 1.0, 1.0);
glLineWidth(2.0);
glBegin(GL_LINES);
glVertex3f(-5.0, 5.0, -5.0);
glVertex3f(20.0, 5.0, -5.0);
glEnd();
glLineWidth(1.0);

// Locate the camera at the tip of the cone and pointing in the direction of the cone.
gluLookAt(0.0, 100.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0);

glDisable(GL_LIGHTING);
glColor3f(1.0, 1.0, 1.0);
glPointSize(2.0);
glBegin(GL_POINTS);
for (int i = 0; i < 11000; i++) {
    glVertex3fv(starPositions[i]);
}
glEnd();
glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);

// End right viewport.
glPushMatrix();
glTranslatef(0.0, 0.0, 0.0);
glCallList(sun);
glPopMatrix();
glEnable(GL_LIGHTING);
```

```
for (i = 0; i < 8; i++) {
    if(i==2)
    {

        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, planetColors[i]);
        glPushMatrix();
        glRotatef(planetAngles[i], 0.0, 1.0, 0.0);  // Rotate the planet around the sun.
        glTranslatef(planetDistances[i], 0.0, 0.0); // Translate along the x-axis to position the planet at the correct distance from
        glCallList(planets[i]);
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, planetColors[8]);
        glPushMatrix();
        glRotatef(moonAngle, 0.0, 1.0, 0.0); // Rotate the Moon around the Earth.
        glTranslatef(moonDistance, 0.0, 0.0); // Translate along the x-axis to position the Moon at the correct distance from the Ear
        glCallList(moon);

        glPopMatrix();
        glPopMatrix();

    }
    else
    {
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, planetColors[i]);
        glPushMatrix();
        glRotatef(planetAngles[i], 0.0, 1.0, 0.0);  // Rotate the planet around the sun.
        glTranslatef(planetDistances[i], 0.0, 0.0); // Translate along the x-axis to position the planet at the correct distance from
        if(i==5)
        {

            glRotatef(70, 1.0, 0.0, 0.0);
```

```
    else
    {
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, planetColors[i]);
        glPushMatrix();
        glRotatef(planetAngles[i], 0.0, 1.0, 0.0);  // Rotate the planet around the sun.
        glTranslatef(planetDistances[i], 0.0, 0.0); // Translate along the x-axis to position the planet at tl
        if(i==5)
        {

            glRotatef(70, 1.0, 0.0, 0.0);
            glutSolidTorus(0.5,7.0,3,30);

        }
        glCallList(planets[i]);
        glPopMatrix();
    }
}
// draw space craft.
glPushMatrix();
glTranslatef(xVal, 0.0, zVal);
glRotatef(angle, 0.0, 1.0, 0.0);
glCallList(spacecraft);
glPopMatrix();

// draw the text.
glPushMatrix();
glColor3f(1.0, 1.0, 1.0);
glRasterPos3f(10.0, 65.0, -30.0);
```

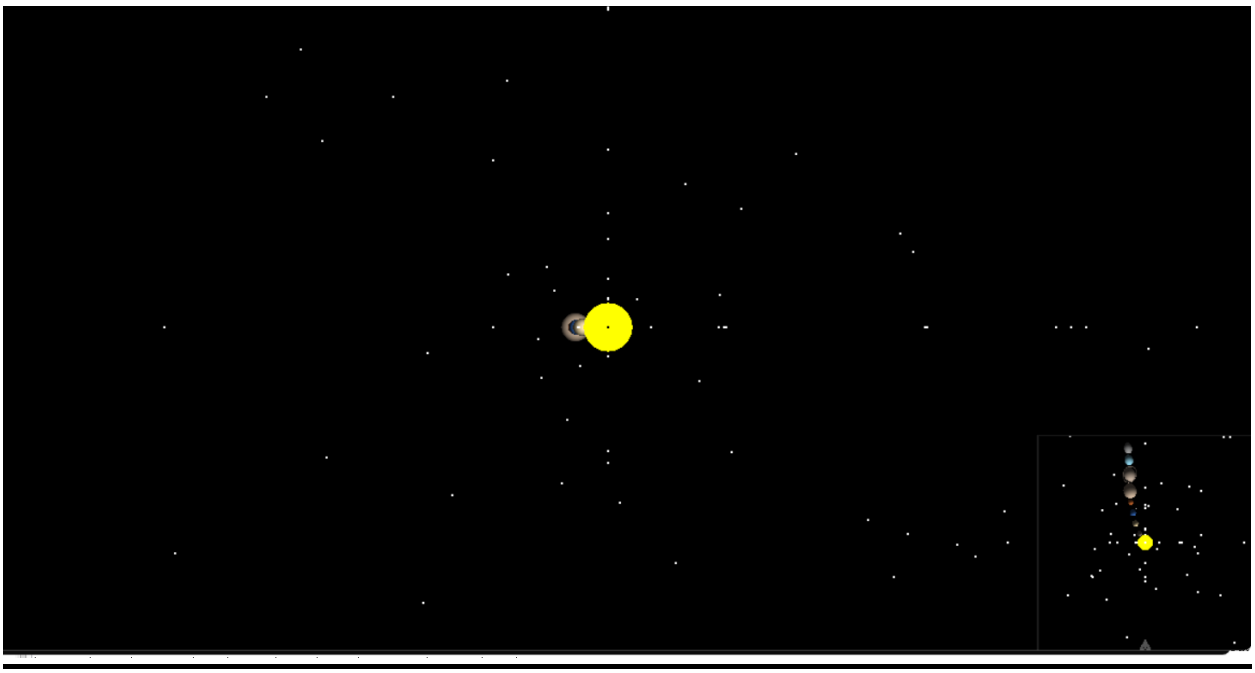
```
// OpenGL window reshape routine.
void resize(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-10.0, 10.0, -5.0, 5.0, 5.0, 250.0);
    glMatrixMode(GL_MODELVIEW);
    // Pass the size of the OpenGL window.
    width = w;
    height = h;
}

// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 27:
            exit(0);
            break;
        default:
            break;
    }
}
```

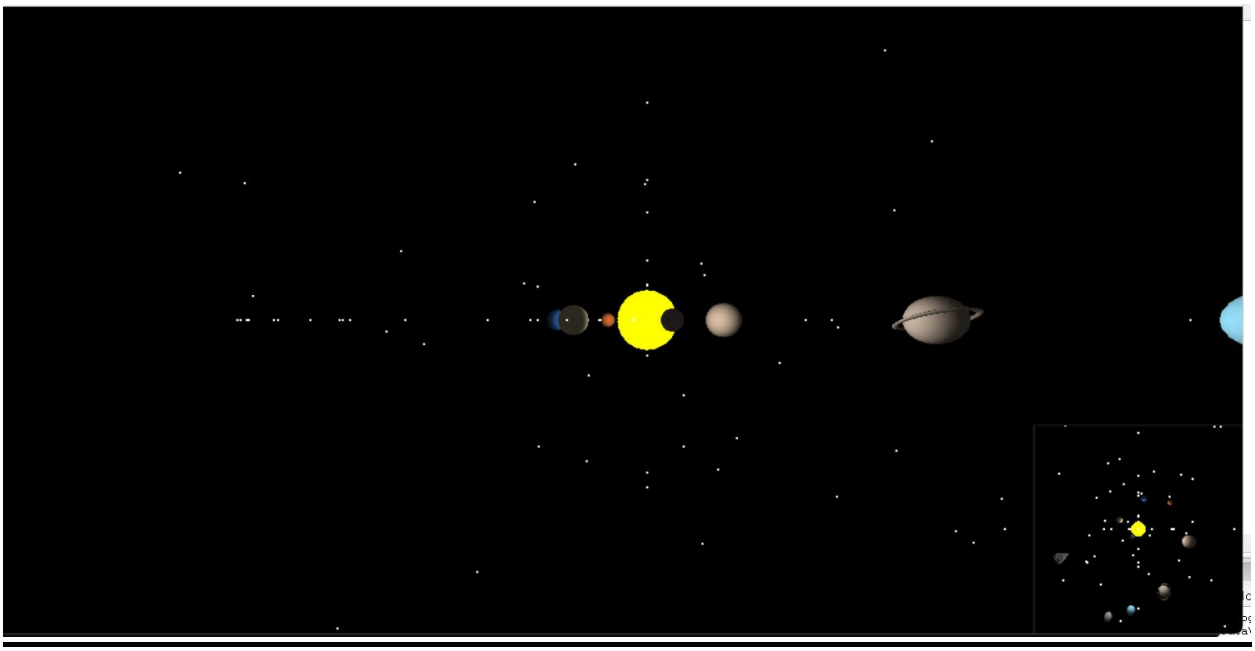
```
void specialKeyInput(int key, int x, int y)
{
    float tempxVal = xVal, tempzVal = zVal, tempAngle = angle;

    // Compute next position.
    if (key == GLUT_KEY_LEFT)
    {
        tempAngle = angle + 5.0;

    }
    if (key == GLUT_KEY_RIGHT)
    {
        tempAngle = angle - 5.0;

    }
    if (key == GLUT_KEY_UP)
    {
        tempxVal = xVal - sin(angle * M_PI / 180.0);
        tempzVal = zVal - cos(angle * M_PI / 180.0);

    }
    if (key == GLUT_KEY_DOWN)
    {
        tempxVal = xVal + sin(angle * M_PI / 180.0);
        tempzVal = zVal + cos(angle * M_PI / 180.0);

    }
```

```
    // Angle correction.
    if (tempAngle > 360.0) tempAngle -= 360.0;
    if (tempAngle < 0.0) tempAngle += 360.0;


    if (!CraftCollision(tempxVal, tempzVal, tempAngle))
    {
        isCollision = 0;
        xVal = tempxVal;
        zVal = tempzVal;
        angle = tempAngle;
        animates = 1;
    }
    else
     {
         // when space craft crahsed it return to its start posisition and planets stop rotating until user press on of the
          angle = 270.0;
          xVal = -100, zVal = 0;
          xVal= xVal;
          zVal= zVal;
          angle = angle;
          isCollision = 1;
          animates = 0;
     }
    glutPostRedisplay();
}
```

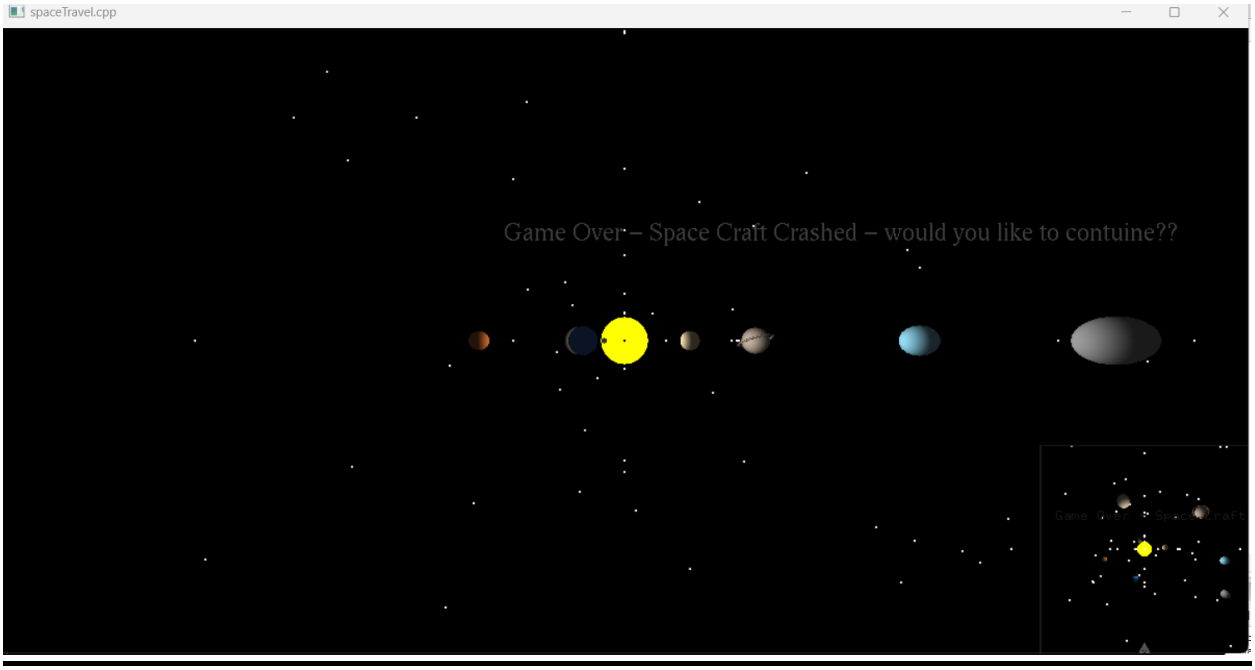
```cpp
// Routine to output interaction instructions to the C++ window.
void printInteraction(void)
{
    std::cout << "Interaction:" << std::endl;
    std::cout << "Press the left/right arrow keys to turn the craft." << std::endl
              << "Press the up/down arrow keys to move the craft." << std::endl;
    std::cout << "when you it appear Game over press up/down/left/right to contuine playing" << std::endl;
}


// Main routine.
int main(int argc, char **argv)
{
    printInteraction();
    glutInit(&argc, argv);
    glutInitContextVersion(4, 3);
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(1200, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("spaceTravel.cpp");
    glutDisplayFunc(drawScene);
    glutReshapeFunc(resize);
    glutKeyboardFunc(keyInput);
    glutSpecialFunc(specialKeyInput);
    glewExperimental = GL_TRUE;
    glewInit();
    setup();
    glutMainLoop();
}
```

Sample runs:





When space craft crash we stop rotate of planets and ask user if he want to continue and he should move the space craft using arrows to continue playing.



Video link:

https://drive.google.com/file/d/1GBORQZkpmicplZYqBpUPODGQ0McSj3m8/view?usp=sharing