

```

#include<GL/glut.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>

//Declaramos constantes
#define MUNDO 5.0f
#define PARTICULAS 250
#define DELTA 0.01f
#define PI 3.141592654f

//Definimos un nuevo tipo de dato
typedef struct {
    float x;
    float y;
    float radio;
    float color[3];
}Pelota;

//Declaracion de funciones
void dibujaCirculo(GLenum primitiva, float radio, float cx, float cy);
void inicializar (void);
void dibuja (void);
void mover (void);
void redimensiona (int ancho, int alto);

//Variables globales
float maxX;
float maxY;
float incX;
float incY;
Pelota particulas[PARTICULAS];

int main(int argc, char **argv) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("Movimiento Browniano");
    inicializar();
    //Registro de funciones de callback
    glutDisplayFunc(dibuja);
    glutReshapeFunc(redimensiona);
    glutIdleFunc(mover);
    glutMainLoop();

    return 0;
}

void dibujaCirculo(GLenum primitiva, float radio, float cx, float cy) {
    float angulo;
    int i;
    glBegin(primitiva);
        for (i = 0; i < 20; i++) {
            angulo = 2.0f * PI / 20.0f * (float)i;
            glVertex3f(radio * cos(angulo) + cx, radio * sin(angulo) + cy, 0.0f);
        }
    glEnd();
}

void inicializar (void) {

    glClearColor(0.33333333f, 0.77254902f, 0.784313725f, 1.0);

    int i;
    //Plantamos semilla del generador de numeros aleatorios
    srand(time(NULL));

    //Iniciamos el incremento
    incX = DELTA;
    incY = DELTA;

    //Iniciamos las bolitas
    for (i = 0; i < PARTICULAS; i++)
    {
        particulas[i].color[0] = (float)(rand() % 2);
        particulas[i].color[1] = (float)(rand() % 2);
        particulas[i].color[2] = (float)(rand() % 2);
    }
}

```

```

        particulas[i].x = (float)(rand() % 11 - 5) * MUNDO / 5.0;
        particulas[i].y = (float)(rand() % 11 - 5) * MUNDO / 5.0;

        particulas[i].radio = (float)((rand() % 10 + 1) * MUNDO / 100.0);
    }

}

void dibuja (void) {
    int i;
    glClear(GL_COLOR_BUFFER_BIT);

    for (i = 0; i < PARTICULAS; i++)
    {
        glColor3fv(particulas[i].color);
        dibujaCirculo(GL_POLYGON, particulas[i].radio, particulas[i].x, particulas[i].y);
    }

    glutSwapBuffers();
}

void redimensiona (int ancho, int alto) {

    float aspectRadio;

    if (alto == 0) {
        alto = 1;
    }

    glViewport(0, 0, ancho, alto);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRadio = (float) ancho / (float) alto;

    if (ancho <= alto) {
        glOrtho(-MUNDO, MUNDO, -MUNDO / aspectRadio, MUNDO / aspectRadio, -MUNDO, MUNDO);
        maxX = MUNDO;
        maxY = MUNDO / aspectRadio;
    } else {
        glOrtho(-MUNDO * aspectRadio, MUNDO * aspectRadio, -MUNDO, MUNDO, -MUNDO, MUNDO);
        maxX = MUNDO * aspectRadio;
        maxY = MUNDO;
    }

}

void mover (void) {
    float angulo;
    int i;

    for (i = 0; i < PARTICULAS; i++)
    {
        angulo = (float)(rand() % 360);

        incX = DELTA * cos(angulo * 0.017453293);
        incY = DELTA * sin(angulo * 0.017453293);

        //Detección y respuesta de colisiones simple
        if ((fabs(particulas[i].y) + particulas[i].radio) >= maxY) {
            incY *= -1.0;
        }

        if ((fabs(particulas[i].x) + particulas[i].radio) >= maxX) {
            incX *= -1.0;
        }

        particulas[i].x += incX;
        particulas[i].y += incY;
    }
    glutPostRedisplay();
}

```