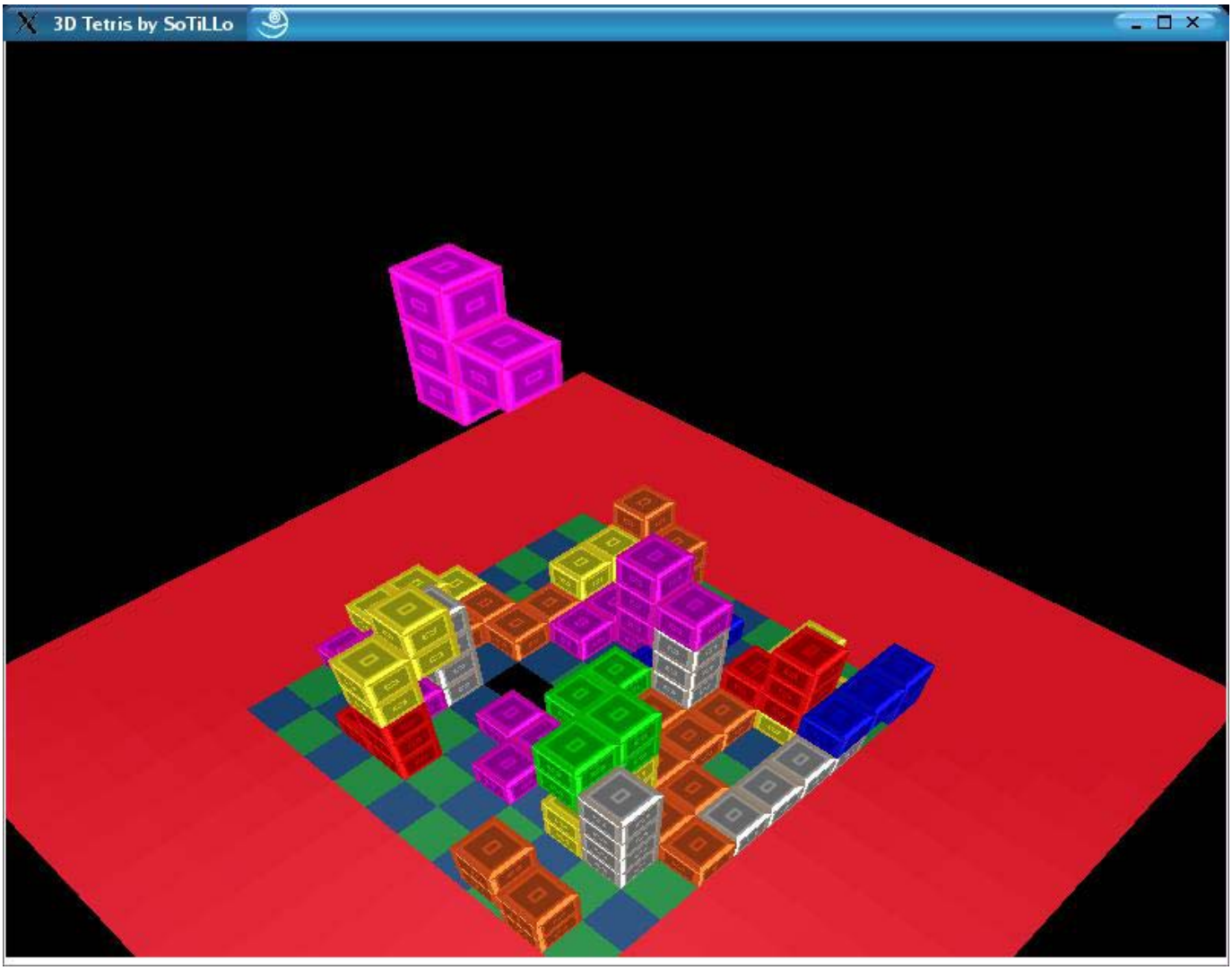
Índice

1. Introducción

El proyecto a realizar no es nada mas que una aplicación del típico juego del tetris en 3D, en la que las piezas tienen 3 dimensiones y no 2 como en el típico juego.

Se ha desarrollado con la librería OpenGL, concretamente con la glut.h y tiene varias opciones de juego, niveles de dificultad, perspectivas, etc.

Esta es una imagen del aspecto de dicho proyecto:



2. Estructura principal del programa

La estructura principal del programa es la siguiente:

●En el procedimiento principal creamos la ventana de juego, con su respectivo tamaño, nombre, posicion etc.

```
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH |
                     GLUT_STENCIL | GLUT_MULTISAMPLE);
glutInitWindowSize (WINDOW_TAM_X, WINDOW_TAM_Y);
glutInitWindowPosition (100, 100);
glutCreateWindow("3D Tetris by SoTiLLo");
```

●Inicializamos mediante nuestro procedimiento inicializar(), con el que asignamos aleatoriamente la pieza actual y ponemos el GL_FLAT.

●Llamamos ahora a las funciones para dibujar y manejar eventos, los cuales los realizamos a través de otras funciones, tales como teclado, raton, etc. Actuando sobre dichos eventos.

```
glutDisplayFunc(display);  /* llamamos para dibujar OpenGL */
glutMouseFunc(raton);
glutMotionFunc(movimiento);
glutKeyboardFunc(teclado); /* entrada de teclado */
glutSpecialFunc(teclado);
```

●Llamamos a la funcion `glutIdleFunc(cayendo)`para actualizar los movimientos ya que dependiendo del nivel de dificultad y de la velocidad las piezas caerán mas rápido. En esta función utilizamos la estructura de la librería time.h, timeval, la cual nos da los tiempos de inicio una vez llamada, sirviéndonos éstos para controlar el intervalo de tiempo a tener en cuenta en la caída de una pieza.

```
struct timeval currtime;
gettimeofday(&currtime,&zone);
tmptime = (currtime.tv_sec-oldtime.tv_sec)*1000000+currtime.tv_usec-
oldtime.tv_usec;
```

●Creamos el menú que se mostrará en pantalla en la posición donde haga clic con el botón derecho del ratón.

●Inicializamos la perspectiva inicial de vista

```
gluPerspective( /*vista en grado */ 55.0,/* proporcion */ 1.0,
              /* Z cerca */ 1.0,/* Z lejos */ 500.0);
```

●Creamos la textura a utilizar

```
glMatrixMode(GL_MODELVIEW);
gluLookAt(0.0, -30.0, -40.0,0.0, -12.0, 0.0, 0.0, -1.0, 0.0);
crearTextura();
```

●Reiniciamos juego, donde ponemos a 0 la variable record, cogemos otra nueva semilla para generar aleatorios, ponemos la variable TotalCapas a 0, reinicializamos los ángulos de la perspectiva inicial y tomamos valores de tiempo para la caída de las piezas

●Activamos el bucle de eventos

```
glutMainLoop();
```

3. Funciones Relevantes

El programa se compone de varias funciones, vamos a detallar las mas relevantes:

3.1 DibujarCuadrado

Función que dibuja un cuadrado con los 4 vertices especificados como argumentos. Para ello utiliza la función normalizar, la cual crea el vector normal a n.

```
/*----------------------------------------------------------------------*
 * dibujar poligono con los 4 vertices especificados
 *----------------------------------------------------------------------*/
void dibujarCuadrado( float p1[], float p2[], float p3[], float p4[])
{
   GLfloat color[4];
   float n[3];
   GLenum drawMode;
   int i;

   glGetFloatv(GL_CURRENT_COLOR, color);
   glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, color);

   for (i = 0; i < 3; i++) color[i] *= 0.5;
   glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, color);
   glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 50.0);

   if (modoMarco || modoSolucionMarco )
           drawMode = GL_LINE_LOOP;
   else
           drawMode = GL_POLYGON;

   glBegin( drawMode );
      for (i = 0; i < 3; i++) n[i] = p1[i];
      normalizarVector(n);
      glNormal3fv(n);
      glVertex3fv( p1 );
      for (i = 0; i < 3; i++) n[i] = p2[i];
      normalizarVector(n);
      glNormal3fv(n);
      glVertex3fv( p2 );
      for (i = 0; i < 3; i++) n[i] = p3[i];
      normalizarVector(n);
      glNormal3fv(n);
      glVertex3fv( p3 );
      for (i = 0; i < 3; i++) n[i] = p4[i];
      normalizarVector(n);
      glNormal3fv(n);
      glVertex3fv( p4 );
   glEnd();
}
```

3.2 DibujarCajaCuadrado

Función que dibuja la figura para formar una pieza, utiliza la función descrita antes (DibujarCuadrado):

```
/*********************************************************************/
/*   Dibuja una caja*/
/********************************************************************* */
void dibujarCajaCuadrado(float tamano)
{
   /* x e y coordenadas como si miraramos del suelo */
        /* cima exterior */
   float cimaDelanteraDer[] = {  1.0, -1.0,  1.0 };
   float cimaDelanteraIzq[] = { -1.0, -1.0,  1.0 };
   float cimaTraseraIzq[] = { -1.0,  1.0,  1.0 };
   float cimaTraseraDer[] = {  1.0,  1.0,  1.0 };
        /* fondo exterior */
   float fondoDelanteroDer[] = {  1.0, -1.0, -1.0 };
   float fondoDelanteroIzq[] = { -1.0, -1.0, -1.0 };
   float fondoTraseroIzq[] = { -1.0,  1.0, -1.0 };
   float fondoTraseroDer[] = {  1.0,  1.0, -1.0 };

        /* alto interior */
   float altoDelanteroDer[] = {  0.8, -0.8,  1.0 };
   float altoDelanteroIzq[] = { -0.8, -0.8,  1.0 };
   float altoTraseroIzq[] = { -0.8,  0.8,  1.0 };
   float altoTraseroDer[] = {  0.8,  0.8,  1.0 };
        /* delantera interior */
   float superiorDelanteroDer[] = {  0.8, -1.0,  0.8 };
   float superiorDelanteroIzq[] = { -0.8, -1.0,  0.8 };
   float inferiorDelanteroIzq[] = { -0.8, -1.0, -0.8 };
   float inferiorDelanteroDer[] = {  0.8, -1.0, -0.8 };
        /* izquierda interior */
   float superiorIzqDelantero[] = { -1.0, -0.8,  0.8 };
   float superiorIzqTrasero[] = { -1.0,  0.8,  0.8 };
   float fondoIzqDelantero[] = { -1.0,  0.8, -0.8 };
   float fondoIzqTrasero[] = { -1.0, -0.8, -0.8 };
        /* derecha interior */
   float superiorDerDelantero[] = {  1.0, -0.8,  0.8 };
   float superiorDerTrasero[] = {  1.0,  0.8,  0.8 };
   float fondoDerDelantero[] = {  1.0, -0.8, -0.8 };
   float fondoDerTrasero[] = {  1.0,  0.8, -0.8 };
        /* trasera interior */
   float superiorDTrasero[] = {  0.8,  1.0,  0.8 };
   float superiorITrasero[] = { -0.8,  1.0,  0.8 };
   float fondoITrasero[] = { -0.8,  1.0, -0.8 };
   float fondoDTrasero[] = {  0.8,  1.0, -0.8 };

    glPushMatrix();

      /* caja al tamaño requerido */
      glScalef(tamano, tamano, tamano);
```

```
/* Todos los polígonos se especifican en sentido contrario a las agujas del reloj */
        dibujarTexturaCuadrado(altoDelanteroDer, altoTraseroDer, altoTraseroIzq,
altoDelanteroIzq);
          dibujarTexturaCuadrado(superiorDelanteroDer, superiorDelanteroIzq,
inferiorDelanteroIzq, inferiorDelanteroDer);
    dibujarTexturaCuadrado(superiorIzqDelantero, superiorIzqTrasero,
fondoIzqDelantero, fondoIzqTrasero);
    dibujarTexturaCuadrado(superiorDerTrasero, superiorDerDelantero,
fondoDerDelantero, fondoDerTrasero);
    dibujarTexturaCuadrado(superiorITrasero, superiorDTrasero, fondoDTrasero,
fondoITrasero);
    dibujarTexturaCuadrado(fondoDelanteroDer, fondoDelanteroIzq, fondoTraseroIzq,
fondoTraseroDer);

 /* dibujo los bordes para cada cuadrado*/
    /* lados de la parte superior */
  dibujarCuadrado(cimaDelanteraIzq, cimaDelanteraDer, altoDelanteroDer,
altoDelanteroIzq);
  dibujarCuadrado(cimaDelanteraDer, cimaTraseraDer, altoTraseroDer,
altoDelanteroDer);
  dibujarCuadrado(cimaTraseraDer, cimaTraseraIzq, altoTraseroIzq, altoTraseroDer);
  dibujarCuadrado(cimaTraseraIzq, cimaDelanteraIzq, altoDelanteroIzq,
altoTraseroIzq);
 /* lados de la parte delantera */
   dibujarCuadrado(fondoDelanteroIzq, fondoDelanteroDer, inferiorDelanteroDer,
inferiorDelanteroIzq);
   dibujarCuadrado(fondoDelanteroDer, cimaDelanteraDer, superiorDelanteroDer,
inferiorDelanteroDer);
   dibujarCuadrado(cimaDelanteraDer, cimaDelanteraIzq, superiorDelanteroIzq,
superiorDelanteroDer);
   dibujarCuadrado(cimaDelanteraIzq, fondoDelanteroIzq, inferiorDelanteroIzq,
superiorDelanteroIzq);
 /* lados de la parte izquierda */
   dibujarCuadrado(cimaDelanteraIzq, cimaTraseraIzq, superiorIzqTrasero,
superiorIzqDelantero);
   dibujarCuadrado(cimaTraseraIzq, fondoTraseroIzq, fondoIzqDelantero,
superiorIzqTrasero);
   dibujarCuadrado(fondoTraseroIzq, fondoDelanteroIzq, fondoIzqTrasero,
fondoIzqDelantero);
   dibujarCuadrado(fondoDelanteroIzq, cimaDelanteraIzq, superiorIzqDelantero,
fondoIzqTrasero);
 /* lados de la parte derecha */
  dibujarCuadrado(cimaTraseraDer, cimaDelanteraDer, superiorDerDelantero,
superiorDerTrasero);
  dibujarCuadrado(cimaDelanteraDer, fondoDelanteroDer, fondoDerDelantero,
superiorDerDelantero);
  dibujarCuadrado(fondoDelanteroDer, fondoTraseroDer, fondoDerTrasero,
fondoDerDelantero);
  dibujarCuadrado(fondoTraseroDer, cimaTraseraDer, superiorDerTrasero,
fondoDerTrasero);
 /* lados de la parte trasera */
   dibujarCuadrado(cimaTraseraIzq, cimaTraseraDer, superiorDTrasero,
superiorITrasero);
   dibujarCuadrado(cimaTraseraDer, fondoTraseroDer, fondoDTrasero, superiorDTrasero);
   dibujarCuadrado(fondoTraseroDer, fondoTraseroIzq, fondoITrasero, fondoDTrasero);
   dibujarCuadrado(fondoTraseroIzq, cimaTraseraIzq, superiorITrasero, fondoITrasero);

   glPopMatrix();
}
```

3.3 Dibujar El mosaico del suelo

Función que dibuja el mosaico del suelo del juego. Para facilitar la colocación de las piezas se diseñó este esquema, en el que la base del juego queda dividida en cuadraditos (locetas).

```
/*-------------------------------------------------------------------------*
 * Dibuja el mosaico del suelo
 *-------------------------------------------------------------------------*/
void dibujarSuelo(void)
{
   float tamanoTotal = 20.0;
   float tamanoCasilla = 2.0;
   float x, y;
   float p1[3], p2[3], p3[3], p4[3];
   int color = 0;
   int colorInicio = 0;   /* usado para dar efecto de locetas */

      /* dibujamos el mosaico del suelo */
      for(x = -tamanoTotal; x < tamanoTotal; x += tamanoCasilla) {
         colorInicio = colorInicio ? 0 : 1;
         color = colorInicio;
         for (y = -tamanoTotal; y < tamanoTotal; y += tamanoCasilla) {
            if (((y>=-tamanoTotal+10) && (y<=tamanoTotal-12)) &&
         ((x>=-tamanoTotal+10) && (x<=tamanoTotal-12)))
     {
            if (color) {
               glColor3f(0.1, 0.3, 0.6);
            }
            else {
               glColor3f(0.1, 0.6, 0.3);
            }
            } else {
               glColor3f(0.9, 0.1, 0.2);
            }

            /* dibujamos una loceta */
            p1[0] = x + 1;
            p1[1] = y + 1;
            p1[2] = -0.99999;
            p2[0] = x + tamanoCasilla + 1;
            p2[1] = y + 1;
            p2[2] = -0.99999;
            p3[0] = x + tamanoCasilla + 1;
            p3[1] = y + tamanoCasilla + 1;
            p3[2] = -0.99999;
            p4[0] = x + 1;
            p4[1] = y + tamanoCasilla + 1;
            p4[2] = -0.99999;
            dibujarCuadrado(p1, p2, p3, p4);

            /* elegimos un color u otro para que no haya 2 locetas iguales */
            color = color ? 0 : 1;
         }
      }
}
```

3.4 Dibujar pieza

Función que dibuja una pieza definida por el índice de piezas establecido, por ejemplo la pieza 0 sería el cubo, etc. El número de pieza se pasa como argumento.

```
/*******************************************/
/*  Dibuja figura dado el indice (nuemro de la figura)*/
/*******************************************/
void dibujarFigura(int numeroFigura)
{
	/*glColor3fv(colorPieza[numeroFigura]);*/
	glColor3fv(colorLt[numeroFigura]);
	glTranslatef(pieza[numeroFigura][0][0],pieza[numeroFigura][0][1],
		pieza[numeroFigura][0][2]);
	dibujarCajaCuadrado( 1.0 );
	glPushMatrix();
	if ( numeroFigura == piezaActual )
	glColor3fv(colorLt[numeroFigura]);
	glTranslatef(pieza[numeroFigura][1][0],pieza[numeroFigura][1][1],
		pieza[numeroFigura][1][2]);
	dibujarCajaCuadrado( 1.0 );
	glPopMatrix();
	glPushMatrix();
	glTranslatef(pieza[numeroFigura][2][0],pieza[numeroFigura][2][1],
		pieza[numeroFigura][2][2]);
	dibujarCajaCuadrado( 1.0 );
	glPopMatrix();
	glPushMatrix();
	glTranslatef(pieza[numeroFigura][3][0],pieza[numeroFigura][3][1],
		pieza[numeroFigura][3][2]);
	dibujarCajaCuadrado( 1.0 );
	glPopMatrix();
}
```

3.5 Dibujar textura

Función que dibuja la textura interior de las piezas asi como la del suelo

```
static void crearTextura(void)
{
  GLubyte TexturaSuelo[16][16][3];
  GLubyte *loc;
  int s, t;
  /* configuracion del color para la textura. */
  loc = (GLubyte*) TexturaSuelo;
  for (t = 0; t < 16; t++) {
    for (s = 0; s < 16; s++) {
      if (circulo[t][s] == 'x') {
        loc[0] = 0xff;
        loc[1] = 0xff;
        loc[2] = 0xff;

      } else
          { /* gris claro. */
        loc[0] = 0xaa;
        loc[1] = 0xaa;
        loc[2] = 0xaa;
      }
      loc += 3;
    }
  }
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexImage2D(GL_TEXTURE_2D, 0, 3, 16, 16, 0, GL_RGB, GL_UNSIGNED_BYTE,
TexturaSuelo);
}
```

3.6 Actualizar las piezas (rotaciones)

Función que actualiza las piezas dependiendo de la rotación ejecutada.

```
/*************************************************************/
/* Actualiza la pieza actual*/
/*************************************************************/
void actualizarPieza( void )
{
         int tmp;

         /* rotamos cuadrados alrededor del eje z: */
         if ( rotacionZ )
        {
                 tmp = pieza[piezaActual][1][0];
                 pieza[piezaActual][1][0] = -pieza[piezaActual][1][1];
                 pieza[piezaActual][1][1] = tmp;

                 tmp = pieza[piezaActual][2][0];
                pieza[piezaActual][2][0] = -pieza[piezaActual][2][1];
                 pieza[piezaActual][2][1] = tmp;

                 tmp = pieza[piezaActual][3][0];
                 pieza[piezaActual][3][0] = -pieza[piezaActual][3][1];
                 pieza[piezaActual][3][1] = tmp;
        }

         /* rotamos cuadrados alrededor del eje X */
         if ( rotacionX )
        {
                 tmp = pieza[piezaActual][1][1];
                 pieza[piezaActual][1][1] = -pieza[piezaActual][1][2];
                 pieza[piezaActual][1][2] = tmp;

                 tmp = pieza[piezaActual][2][1];
                 pieza[piezaActual][2][1] = -pieza[piezaActual][2][2];
                 pieza[piezaActual][2][2] = tmp;

                 tmp = pieza[piezaActual][3][1];
                 pieza[piezaActual][3][1] = -pieza[piezaActual][3][2];
                 pieza[piezaActual][3][2] = tmp;
        }

         if ( rotacionY )
        {
                 tmp = pieza[piezaActual][1][0];
                 pieza[piezaActual][1][0] = pieza[piezaActual][1][2];
                 pieza[piezaActual][1][2] = -tmp;

                 tmp = pieza[piezaActual][2][0];
                 pieza[piezaActual][2][0] = pieza[piezaActual][2][2];
                 pieza[piezaActual][2][2] = -tmp;

                 tmp = pieza[piezaActual][3][0];
                 pieza[piezaActual][3][0] = pieza[piezaActual][3][2];
                 pieza[piezaActual][3][2] = -tmp;
        }
}
```

3.7 Comprobar que no se ha salido del límite de juego

Función que comprueba si la pieza se ha salido del panel de juego.

```
/*****************************************************/
/*Comprobar si esta dentro de los límites del tablero*/
/*****************************************************/
int dentroLimites (int numeroFigura, int movimiento)
{
        int x, y, z, x1, y1, z1, x2, y2, z2, x3, y3, z3;
        int t;
        float s, c, tmp;

        x = pieza[numeroFigura][0][0];
        y = pieza[numeroFigura][0][1];
        z = pieza[numeroFigura][0][2];

        x1 = x + pieza[numeroFigura][1][0];
        y1 = y + pieza[numeroFigura][1][1];
        z1 = z + pieza[numeroFigura][1][2];

        x2 = x + pieza[numeroFigura][2][0];
        y2 = y + pieza[numeroFigura][2][1];
        z2 = z + pieza[numeroFigura][2][2];

        x3 = x + pieza[numeroFigura][3][0];
        y3 = y + pieza[numeroFigura][3][1];
        z3 = z + pieza[numeroFigura][3][2];

        switch(movimiento)
        {
                case 100:  /*izquierda*/
                        return ((y < 10) && (y1 < 10) && (y2 < 10) && (y3 < 10));
                case 101: /* adelante */
                        return ((x < 10) && (x1 < 10) && (x2 < 10) && (x3 < 10));
                case 102: /* derecha */
                        return ((y > -8) && (y1 > -8) && (y2 > -8) && (y3 > -8));
                case 103: /* atras */
                        return ((x > -8) && (x1 > -8) && (x2 > -8) && (x3 > -8));
                case 32:  /* Espacio */
                        return ((z > 0) && (z1 > 0) && (z2 > 0) && (z3 > 0));
        }
}
```

3.8 Comprobar que no hay colisión con otras piezas al realizar el movimiento

Función que comprueba si el movimiento a realizar provoca colisión con alguna pieza.

```
        /*-------------------------------------------------------------------*
 * Detectamos si hay colisión con otras piezas al mover
/*-------------------------------------------------------------------*/
int colision(int numeroFigura, int movimiento)
{
        int x, y, z, x1, y1, z1, x2, y2, z2, x3, y3, z3;
        int t;
        float s, c, tmp;

        if (FinJuego)
                return 1;

        x = XCOORD(numeroFigura);
        y = YCOORD(numeroFigura);
        z = ZCOORD(numeroFigura);

        x1 = x + pieza[numeroFigura][1][0];
        y1 = y + pieza[numeroFigura][1][1];
        z1 = z + pieza[numeroFigura][1][2];

        x2 = x + pieza[numeroFigura][2][0];
        y2 = y + pieza[numeroFigura][2][1];
        z2 = z + pieza[numeroFigura][2][2];

        x3 = x + pieza[numeroFigura][3][0];
        y3 = y + pieza[numeroFigura][3][1];
        z3 = z + pieza[numeroFigura][3][2];

        switch(movimiento)
       {
                case 100: /* movimiento a la izquierdat */
                        if ( Contenedor[x][y+2][z]    || Contenedor[x1][y1+2][z1] ||
                                Contenedor[x2][y2+2][z2] ||
Contenedor[x3][y3+2][z3] )
                                        return TRUE;
                        return FALSE;
                        break;
                case 101: /* movimiento hacia adelante */
                        if ( Contenedor[x+2][y][z]    || Contenedor[x1+2][y1][z1] ||
                                Contenedor[x2+2][y2][z2] ||
Contenedor[x3+2][y3][z3] )
                                return TRUE;
                        return FALSE;
                        break;
                case 102: /* movimiento a la derecha */
                        if ( Contenedor[x][y-2][z]    || Contenedor[x1][y1-2][z1] ||
                                Contenedor[x2][y2-2][z2] || Contenedor[x3][y3-
2][z3] )
                                return TRUE;
                        return FALSE;
                        break;
                case 103: /* movimiento hacia atrás */
                        if ( Contenedor[x-2][y][z]    || Contenedor[x1-2][y1][z1] ||
                                Contenedor[x2-2][y2][z2] || Contenedor[x3-
2][y3][z3] )
                                return TRUE;
                        return FALSE;
```

```
                        break;
                case 32: // av pag
                        if ( Contenedor[x][y][z-2]    || Contenedor[x1][y1][z1-2] ||
                             Contenedor[x2][y2][z2-2] || Contenedor[x3][y3][z3-2] )
                                return TRUE;
                        return FALSE;
                        break;
                case 122: /* rotacion z (tecla z) */
                        for(t=30; t<=90; t+=30)
                       {
                                c = cos(t * M_PI/180.0);
                                s = sin(t * M_PI/180.0);

                                x = pieza[numeroFigura][0][0] + 8;
                                y = pieza[numeroFigura][0][1] + 8;
                                z = pieza[numeroFigura][0][2] - 0;

                                x1 = pieza[numeroFigura][1][0];
                                y1 = pieza[numeroFigura][1][1];

                                rotarX(&x1, &y1, c, s, x, y);


                                x2 = pieza[numeroFigura][2][0];
                                y2 = pieza[numeroFigura][2][1];

                                rotarX(&x2, &y2, c, s, x, y);


                                x3 = pieza[numeroFigura][3][0];
                                y3 = pieza[numeroFigura][3][1];

                                rotarX(&x3, &y3, c, s, x, y);

                                z1 = z + pieza[numeroFigura][1][2];
                                z2 = z + pieza[numeroFigura][2][2];
                                z3 = z + pieza[numeroFigura][3][2];

                                if ((x1 > 18) || (y1 > 18) || (x1 < 0) || (y1 < 0)
|| (z1 < 0) ||
                                        (x2 > 18) || (y2 > 18) || (x2 < 0) || (y2 <
0) || (z2 < 0) ||
                                        (x3 > 18) || (y3 > 18) || (x3 < 0) || (y3 <
0) || (z3 < 0))
                                        return TRUE;

                                if ( Contenedor[x1][y1][z1] ||
Contenedor[x2][y2][z2] ||
                                             Contenedor[x3][y3][z3] )
                                        return TRUE;
                        }
                        return FALSE;
                        break;

                case 120: /* rotacion x (tecla x)*/
                        for(t=30; t<=90; t+=30)
                        {
                                c = cos(t * M_PI/180.0);
                                s = sin(t * M_PI/180.0);

                                x = pieza[numeroFigura][0][0] + 8;
```

```
                                y = pieza[numeroFigura][0][1] + 8;
                                z = pieza[numeroFigura][0][2] - 0;

                                y1 = pieza[numeroFigura][1][1];
                                z1 = pieza[numeroFigura][1][2];

                                rotarX(&y1, &z1, c, s, y, z);

                                y2 = pieza[numeroFigura][2][1];
                                z2 = pieza[numeroFigura][2][2];

                                rotarX(&y2, &z2, c, s, y, z);

                                y3 = pieza[numeroFigura][3][1];
                                z3 = pieza[numeroFigura][3][2];

                                rotarX(&y3, &z3, c, s, y, z);

                                x1 = x + pieza[numeroFigura][1][0];
                                x2 = x + pieza[numeroFigura][2][0];
                                x3 = x + pieza[numeroFigura][3][0];

                                if ((x1 > 18) || (y1 > 18) || (x1 < 0) || (y1 < 0)
|| (z1 < 0) ||
                                        (x2 > 18) || (y2 > 18) || (x2 < 0) || (y2 <
0) || (z2 < 0) ||
                                        (x3 > 18) || (y3 > 18) || (x3 < 0) || (y3 <
0) || (z3 < 0))
                                        return TRUE;

                                if ( Contenedor[x1][y1][z1] ||
Contenedor[x2][y2][z2] ||
                                     Contenedor[x3][y3][z3] )
                                        return TRUE;
                        }
                         return FALSE;
                         break;

                case 99: /* rotancion y (tecla c) */
                         for(t=45; t<=90; t+=45)
                        {
                         c = cos(t * M_PI/180.0);
                         s = sin(t * M_PI/180.0);

                         x = pieza[numeroFigura][0][0] + 8;
                         y = pieza[numeroFigura][0][1] + 8;
                         z = pieza[numeroFigura][0][2] - 0;

                         x1 = pieza[numeroFigura][1][0];
                         z1 = pieza[numeroFigura][1][2];

                         rotarY(&x1, &z1, c, s, x, z);

                         x2 = pieza[numeroFigura][2][0];
                         z2 = pieza[numeroFigura][2][2];

                         rotarY(&x2, &z2, c, s, x, z);

                         x3 = pieza[numeroFigura][3][0];
                         z3 = pieza[numeroFigura][3][2];
```

```
                        rotarY(&x3, &z3, c, s, x, z);

                        y1 = y + pieza[numeroFigura][1][1];
                        y2 = y + pieza[numeroFigura][2][1];
                        y3 = y + pieza[numeroFigura][3][1];

                        if ((x1 > 18) || (y1 > 18) || (x1 < 0) || (y1 < 0) || (z1 <
0) ||
                                (x2 > 18) || (y2 > 18) || (x2 < 0) || (y2 < 0) ||
(z2 < 0) ||
                                (x3 > 18) || (y3 > 18) || (x3 < 0) || (y3 < 0) ||
(z3 < 0))
                        return TRUE;

                        if ( Contenedor[x1][y1][z1] || Contenedor[x2][y2][z2] ||
                                    Contenedor[x3][y3][z3] )
                            return TRUE;
                }
                 return FALSE;
                 break;

                 default:
                 break;
        }
}
```

3.9 Funciones que nos devuelven los cubos a comprobar tras rotar ejes

3.9.1 RotarY

```
/*************************************************************
      Esta función da los cubos as chequear cuando rotamos en el eje Y
*************************************************************/
void rotarY(int *a, int *b, float c, float s, int d, int f)
{
 float tmp1, tmp2;
 float aa = *a * 100;
 float bb = *b * 100;

 tmp1 = (aa) * c + (bb) * s;

 tmp2 = -(aa) * s + (bb) * c;

 *a =(int)(tmp1 / 100) + d;
 *b = (int)(tmp2 /100) + f;

 if (((*a) % 2) == 1) (*a)--;
 if (((*b) % 2) == 1) (*b)--;
}
```

3.9.2 RotarX

```
/*************************************************************
      Esta función da los cubos as chequear cuando rotamos en el eje x
 *************************************************************/
void rotarX(int *a, int *b, float c, float s, int d, int f)
{
         float tmp2, tmp1;
         float aa = *a * 100;
         float bb = *b * 100;

         tmp1 = (aa) * c - (bb) * s;

         tmp2 = (aa) * s + (bb) * c;

         *a =(int)(tmp1 / 100) + d;
         *b = (int)(tmp2 /100) + f;

         if (((*a) % 2) == 1) (*a)--;
         if (((*b) % 2) == 1) (*b)--;
}
```

3.10 Eventos de teclado

Función que actúa dependiendo del evento recibido de teclado (mover o rotar piezas, acelerar caída, pausar juego o salir).

```
/*----------------------------------------------------------------------*
 * Funcion llamada cuando se pulsa una tecla del teclado
 *----------------------------------------------------------------------*/
static void teclado(unsigned char tecla, int x, int y)
{
   int i;
   //printf("teclaaaa: %i\n",tecla);
   switch (tecla)
   {
                case 'p':
                        Pausa = (Pausa)? 0 : 1;
                        break;

               case 27:       /* tecla ESC */
                       exit(0);
                        break;
                case 100: /* tecla cursor izquierda */
                        if(dentroLimites (piezaActual, 100))
                        {
                                if ( !colision(piezaActual, 100) )
                               {
                                        for(i=0;i<2;i += 1)
                                        {
                                                pieza[piezaActual][0][1] +=1;
                                                display();
                                        }
                               }
                       }
                        break;

                case 101: /* tecla cursor adelante */
                        if (dentroLimites (piezaActual, 101))
                        {
                                if ( !colision(piezaActual, 101) )
                               {
                                        for(i=0;i<2;i += 1)
                                        {
                                                pieza[piezaActual][0][0] += 1;
                                                display();
                                        }
                               }
                       }
                        break;
                case 102: /* tecla cursor derecha */
                        if(dentroLimites (piezaActual, 102))
                        {
                                if ( !colision(piezaActual, 102) )
                               {
                                        for(i=0;i<2;i += 1)
                                        {
                                                pieza[piezaActual][0][1] -=1;
                                                display();
                                        }
                               }
                       }
```

```
                        break;
                case 103: /* tecla cursor atrá */
                        if(dentroLimites (piezaActual, 103))
                        {
                                if ( !colision(piezaActual, 103) )
                                {
                                        for(i=0;i<2;i += 1)
                                        {
                                                pieza[piezaActual][0][0] -=1;
                                                display();
                                        }
                                }
                        }
                        break;
                case 32: //tecla espaciador
                        if(dentroLimites (piezaActual, 32))
                        {
                                if ( !colision(piezaActual, 32) )
                                {
                                    for(i=0;i<2;i += 1)
                                    {
                                                pieza[piezaActual][0][2] -=1;
                                                display();
                                    }
                                }
                        }
                        break;
                case 122: /* tecla z */
                        rotacionZ = 1;
                        if ( !colision(piezaActual, 122) )
                        {
                                  for(i=PASOS_ROT;i<=90;i+=PASOS_ROT)
                                          display();
                          }
                        else
                                rotacionZ = 0;
                        break;

                case 120: /* tecla x */
                        rotacionX = 1;
                        if ( !colision(piezaActual, 120) )
                        {
                                  for(i=PASOS_ROT;i<=90;i+=PASOS_ROT)
                                 display();
                        }
                        else
                                rotacionX = 0;
                        break;
                case 99: /* tecla c */
                        rotacionY = 1;
                        if ( !colision(piezaActual, 99) )
                        {
                                  for(i=PASOS_ROT;i<=90;i+=PASOS_ROT)
                                        display();
                          }
                          else
                                rotacionY = 0;
                        break;
        }
}
```

3.11 Eventos de ratón

Función que actúa dependiendo del evento recibido del raton. (Mover perspectiva o ver menu)

```
/*-----------------------------------------------------------------------*
 * Callback del raton
 *-----------------------------------------------------------------------*/
void raton(int boton, int estado, int x, int y)
{
          if (boton == GLUT_LEFT_BUTTON)
          {
            if (estado == GLUT_DOWN)
            {
                      traslado = 1;
                      comienzoX = x;
                      comienzoY = y;
            }
            if (estado == GLUT_UP)
            {
                      traslado = 0;
            }
          }
          if (boton == GLUT_MIDDLE_BUTTON)
          {
                  if (estado == GLUT_DOWN)
                  {
                      perspectivaTraslado = 1;
                      perspectivaComienzoX = x;
                      perspectivaComienzoY = y;
            }
            if (estado == GLUT_UP)
                    perspectivaTraslado = 0;
          }
}
```

3.12 Acciones del menú principal

Función que actúa dependiendo de la elección del menú principal

```
/*-------------------------------------------------------------------------*
 *  Acciones del menu principal
 *-------------------------------------------------------------------------*/
static void seleccionMenuPral(int seleccion)
{
   switch (seleccion)
   {
              case MENU_PRAL_AYUDA:
                      printf("\n\n\n");
                      printf("\n***************** Ayuda Tetris3D***********");
                      printf("\n");
                      printf("\n             ^ mueve pieza adelante");
                      printf("\n          <- | ->  mueve pieza izquierda derecha");
                      printf("\n             v mueve pieza atras");
                      printf("\n");
                      printf("\n");
                      printf("\n       z rota pieza sobre eje Z");
                      printf("\n       x rota pieza sobre eje X");
                      printf("\n       c rota pieza sobre eje Y");
                      printf("\n");
                      printf("\n Con la tecla espaciador aceleramos la caida");
                      printf("\n*****************************************\n\n");
              break;

        case MENU_PRAL_REINICIAR:
                reiniciarJuego();
                display();
        break;

        case MENU_PRAL_SIGUIENTE:
                MostrarSiguiente = (MostrarSiguiente)? 0 : 1;
                display ();
         break;

        case MENU_PRAL_PAUSA:
                Pausa = (Pausa)? 0 : 1;
        break;

         case MENU_PRAL_RECORD:
                 printf ("\nTu record actual es: %d.\n", Record);
         break;

         case MENU_PRAL_SALIR:
                 exit(0);
        break;
   }
}
```

3.13 Operaciones con el contenedor

Estas funciones se encargan de mantener la información de la colocación de las piezas para así poder realizar limpiadas de capas y obtener puntos.

3.13.1 Actualizar contenedor

Actualizamos contenedor cada vez que se coloca una pieza, guardando las características de dicha pieza, posición, capa, etc.

```
/*actualizacion del contenedor*/
void actualizarContenedor (void)
{
         int i, x, y, z;
         int x0 = pieza [piezaActual][0][0];
         int y0 = pieza [piezaActual][0][1];
         int z0 = pieza [piezaActual][0][2];
         anadirContenedor (x0, y0, z0);

         for (i = 1; i <= 3; i++)
         {
                 x = x0 + pieza [piezaActual][i][0];
                 y = y0 + pieza [piezaActual][i][1];
                 z = z0 + pieza [piezaActual][i][2];
                 anadirContenedor (x, y, z);
        }

         resolver (z0);

         piezaNueva ();
}
```

3.13.2 Añadir contenedor

Añadimos un contenedor para la pieza actual

```
void anadirContenedor (int x, int y, int z)
{
         if ((x < -8) || (x > 10)) return;
         if ((y < -8) || (y > 10)) return;
         if ((z < 0) || (z > 28)) return;

         Contenedor [x+8][y+8][z] = 1;
         ContenedorColor [x+8][y+8][z] = piezaActual;
}
```

3.13.3 Inicializar contenedor

Inicializamos el contenedor a 0 indicando que no hay piezas

```
void inicializarContenedor (void)
{
        int x, y, z;

        for(x=0; x<=18; x++)
                for(y=0; y<=18; y++)
                        for(z=0; z<=28; z++)
                                Contenedor [x][y][z] = 0;
}
```

3.14 Comprobar si tenemos alguna capa completa

Función que mira si tenemos una fila o columna completa por cubos de fichas, y limpiamos dicha capa y bajamos los restantes a la capa anterior. Utilizamos las funciones comprobar fila y comprobar columna.

```
/*mirar si tenemos filas o columnas completas en una capa*/
int Comprobar (int capa)
{
        int x, y;
         int count_x = 0, count_y = 0;

         if ((capa < 0) || (capa > 28))
                 return 0;

         for(x=0; x<=18; x+=2)
            if (ComprobarFila (x, capa)) count_x++;

         for(y=0; y<=18; y+=2)
                 if (ComprobarColumna (y, capa)) count_y++;

         switch (Dificultad)
         {
                 case 0:
                         if ((count_x + count_y) > 0)
                         return 1;
                 case 1:
                         if ((count_x > 0) && (count_y > 0))
                         return 1;
                 case 2:
                         if (count_x == 10)
                         return 1;
        }
         return 0;
}
```

3.14.1 Comprobar filas y columnas

```
/*mirar si tenemos una fila completa*/
int ComprobarFila (int fila, int capa)
{
         int y;

         for (y=0; y<=18; y+=2)
                 if (!Contenedor [fila][y][capa])
                         return 0;
                 return 1;
}

/*mirar si tenemos una columna completa*/
int ComprobarColumna (int columna, int capa)
{
         int x;

         for (x=0; x<=18; x+=2)
                 if (!Contenedor [x][columna][capa])
                         return 0;
                 return 1;
}
```

3.15 Implementación de la caída de piezas

Función que implementa la caída de piezas mediante la estructura timeval. Pudiendo así ejecutar distintas velocidades de caida, etc. También actualizamos las piezas en su caida.

```
void cayendo (void)
{
         int i,tmp;
         long tmptime;

         if (Pausa || FinJuego) return;

         struct timeval currtime;
         gettimeofday(&currtime,&zone);
         tmptime = (currtime.tv_sec-oldtime.tv_sec)*1000000+currtime.tv_usec-
oldtime.tv_usec;

         if(tmptime>Intervalo)
        {
                 oldtime.tv_sec = currtime.tv_sec;
                 oldtime.tv_usec = currtime.tv_usec;

                 if (!colision (piezaActual, 32) && dentroLimites (piezaActual,
32))
                 {
                    for(i=0;i<2;i += 1)
                    {
                                pieza[piezaActual][0][2] -=1;
                                SiempreCallendo = 1;
                                display();
                    }
                            for (i=0;i<500000 + (9 - Velocidad) * 30000; i++)
                                     tmp = sqrt (i);
                 }
                 else
                 {
                          if (SiempreCallendo)
                          {
                                  actualizarContenedor ();
                                  display ();
                                  SiempreCallendo = 0;
                         }
                         else
                         {
                                  PiezaPrevia = -1;
                                  FinJuego = 1;
                                 printf("\n\n\nFin de juego. Su Record es:
%d\n\n", Record);
                         }
                 }
        }
}
```

3.16 Implementación de generación de número aleatorios

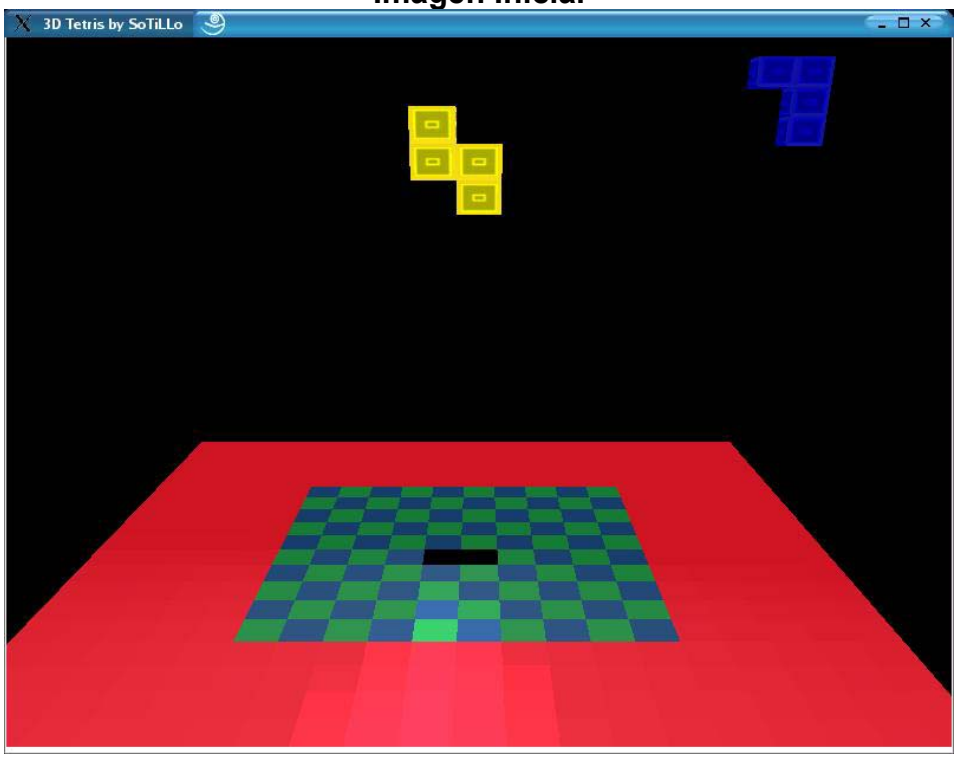
Para generar número aleatorios hemos utilizado un generador de números seudoaleatorio y se ha incluido en el fichero random.h. Y es el siguiente

```
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>


/***** GENERADOR DE NUMEROS PSEUDOALETORIOS *****/

/* used in random number generator below */
#define MASK 2147483647
#define PRIME 65539
#define SCALE 0.4656612875e-9

/*************************************************************/
/*  Rand genera un numero real pseudoaleatorio entre 0 y 1,        */
/*  excluyendo el 1.                                          */
/*  Randint genera un numero entero entre low y high, ambos         */
/*  incluidos.                                                */
/*  Randfloat genera un numero real entre low y high, incluido low */
/*  y no incluido high                                             */
/*************************************************************/

#define Rand()  (( Seed = ( (Seed * PRIME) & MASK) ) * SCALE )

#define Randint(low,high) ( (int) (low + (high-(low)+1) * Rand()))

#define Randfloat(low,high) ( (low + (high-(low))*Rand()))
```

4. Ejecuciones del programa

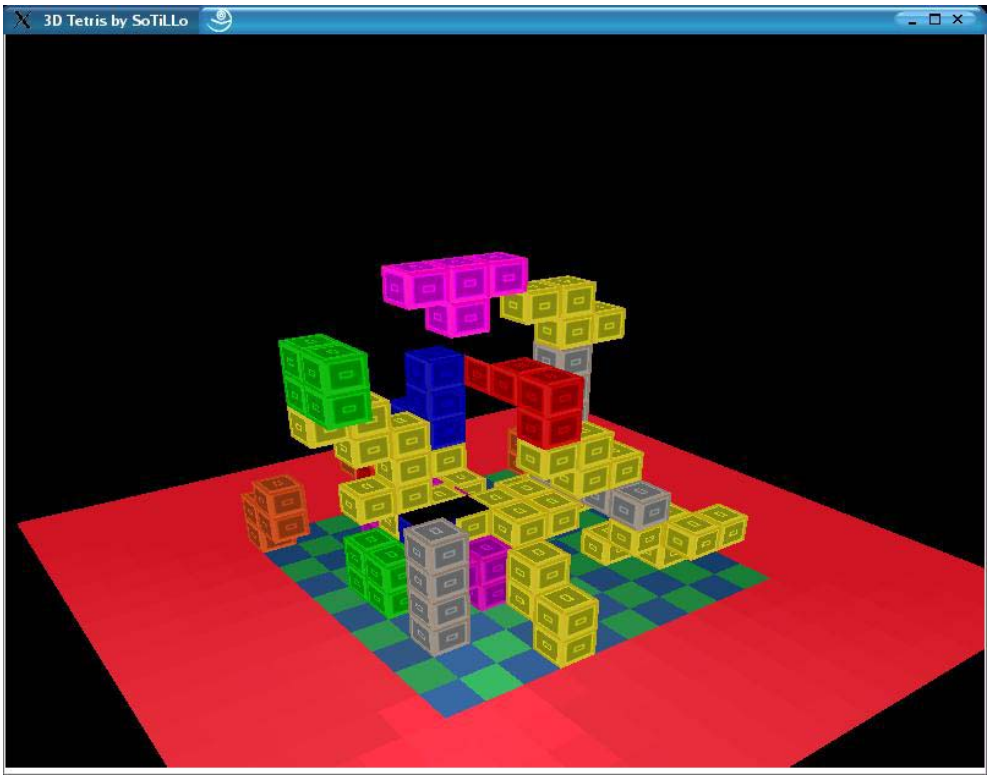
Estas son distintas ejecuciones del programa.

Imagen Inicial

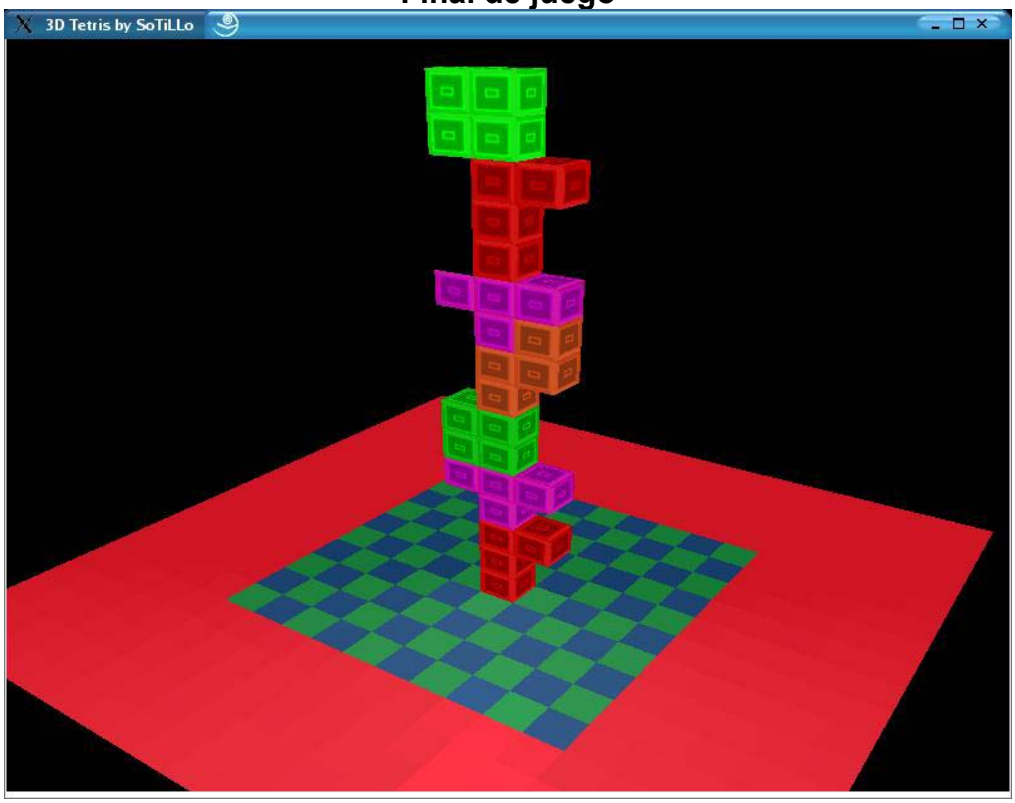


Perfil Perspectiva inicial

Perfil



Final de juego



5. Manual de juego

Para jugar al maravilloso juego, hay que lanzar el ejecutable, bien desde una consola o bien haciendo clic sobre éste.

CONSOLA: $: `./tetris3d`

Para mover las piezas se utilizan los cursores, los cuales mueven la pieza actual en la dirección indicada. Se tiene asignado el movimiento por defecto a la perspectiva inicial, por lo que si se cambia de perspectiva el movimiento será el mismo que el usado para la perspectiva inicial.

- Cursor ↑ : Mueve la pieza en dirección atrás en la perspectiva (aleja)
- Cursor ↓ : Mueve la pieza en dirección adelante en la perspectiva (acerca)
- Cursor ← : Mueve la pieza en dirección izquierda en la perspectiva (izquierda)
- Cursor → : Mueve la pieza en dirección derecha en la perspectiva (derecha)

Para realizar rotaciones se han definido las siguientes teclas:

- Tecla **z** : Rota alrededor del eje Z (siempre que la rotación no implique que la pieza se salga fuera del panel de juego)
- Tecla **x** : Rota alrededor del eje X(siempre que la rotación no implique que la pieza se salga fuera del panel de juego)
- Tecla **c** : Rota alrededor del eje Y (siempre que la rotación no implique que la pieza se salga fuera del panel de juego)

Podemos asegurar la caída de las piezas manteniendo pulsada la **tecla Espacio**

Para visualizar el menú basta con hacer clic con el botón derecho y sin soltar el botón desplazarse por el menú deseado. Tenemos los siguientes menús:

Menú Dificultad: Ponemos la dificultad del juego (Principiante, Intermedio, Experto)
Menú Velocidad: Ponemos mayor o menor velocidad
Menú Mostrar Siguiente Pieza: Vemos o no la siguiente pieza
Menú Reiniciar: Reiniciamos el juego
Menú Pausa: Hacemos una pausa en el juego (también se puede hacer con la **tecla p**)
Menú Mostrar Record: Mostramos los puntos obtenidos (consola)
Menú Ayuda: Mostramos una ayuda de cómo usar el juego (consola)
Menú Salir: Salimos del juego

Haciendo clic con el botón izquierdo del ratón en cualquier zona del juego y manteniéndolo pulsado podemos cambiar la perspectiva del juego.