METODOLOGÍA PARA LA PROGRAMACIÓN DE VIDEOJUEGOS PARA PC EN LA PLATAFORMA XNA GAME STUDIO 3.1 Y VISUAL STUDIO 2008

Integrantes

EDWIN ANDRÉS BETANCUR ÁLVAREZ

JHON FREDDY VELÁSQUEZ RESTREPO

UNIVERSIDAD SAN BUENAVENTURA

Facultad de Ingenierías

Seccional Medellín

Año 2012

TABLA DE CONTENIDO

1. INTRODUCCIÓN

Principalmente en una metodología lo que se pretende es dar una guía para realizar unas acciones, que nos van indicando que hacer y nos van orientando para desempeñar una función a cabalidad.

Partimos en que en el campo de los videojuegos podemos observar que todo tiene un proceso de construcción y elaboración empezando desde lo mas mínimo en una estructura hasta lo más complejo que es un avatar en movimiento con texturas, iluminación, colisión frente a otras estructuras y los efectos especiales que pueden surgir de éste.

En esta ocasión se aportará un espacio para que los estudiantes del semillero de investigación llamado SISUSBMED (Semillero de Investigación y Desarrollo del Software) del cual es parte el Microsoft Student Tech Club, pueda tener pautas para empezar y puedan hacer el proceso de futura participación para el Imagine Cup .

2. OBJETIVOS DE LA METODOLOGÍA

2.1 **Objetivo General**

Elaborar una guía de funcionamiento elemental de usuario sobre la plataforma de desarrollo Microsoft XNA que permita, en modo principiante una interacción con este sistema de desarrollo de videojuegos.

2.2 **Objetivos Específicos**

- Presentar una muestra práctica y concisa de un mundo virtual, elaborada en esta plataforma para desarrollo de juegos en XNA, la cual servirá de ejemplo ilustrado para futuros estudiantes que se interesen por el tema.
- Elaborar un seguimiento de código de XNA para entender el funcionamiento de cada línea para saber la lógica de construcción del mundo.
- Identificar la lógica de programación en el entorno de XNA para la creación de un mundo virtual.

METODOLOGÍA EN 3D

3. INICIACIÓN DE SOFTWARE

Inicialmente abrimos nuestro asistente de Visual Studio 2008

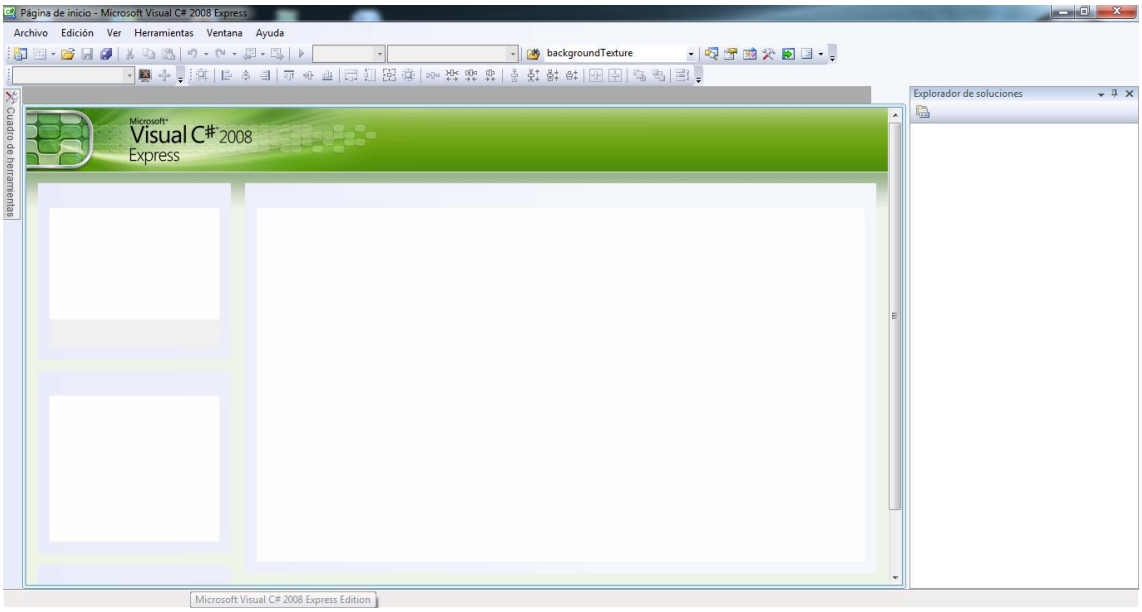


Figura 1: Iniciación de software

Luego que tenemos nuestro asistente abierto vamos al menú Archivo-Nuevo Proyecto

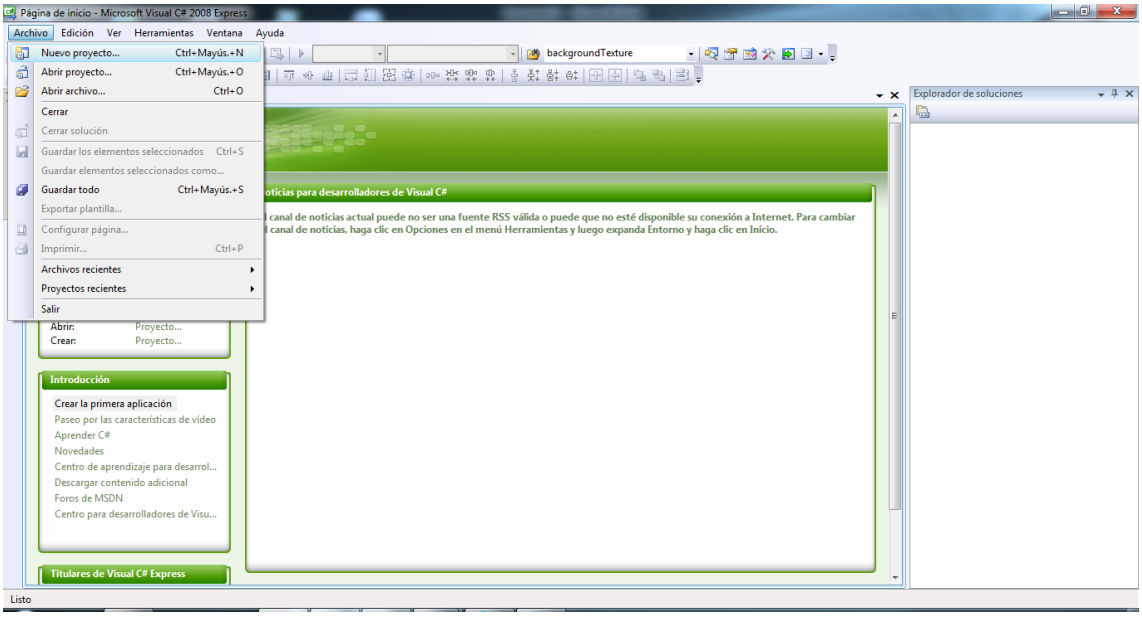


Figura 2: Creación de nuevo proyecto

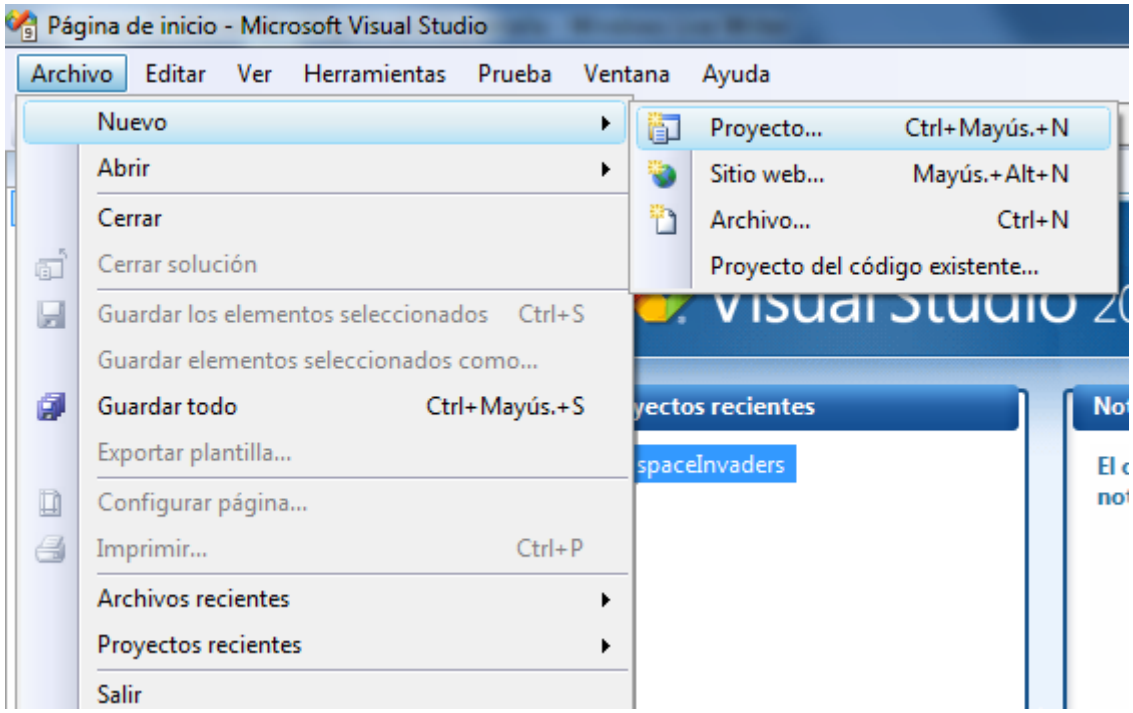


Figura 1: Nuevo proyecto visual c#

Al darle nuevo proyecto nos abre una ventana para poder seleccionar que tipo de proyecto queremos desarrollar en nuestro caso vamos a hacer un desarrollo en Visual Studio 2008 con XNA 3.1 y en esta ventana de Tipo de proyecto seleccionamos la opción de XNA 3.1 y en las parte de las plantillas escogemos la plantilla de Windows Game 3.1 y a continuación le damos el nombre del proyecto en este caso lo llamaremos “Mundo_Virtual”

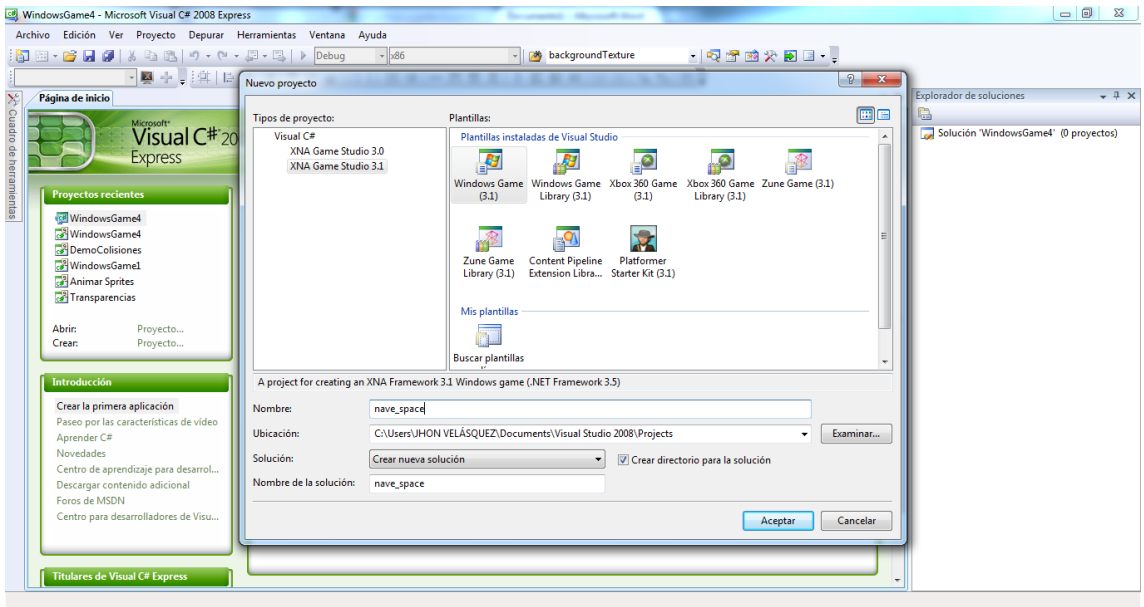


Figura 4: Creación de proyecto con XNA 3.1

Al dar doble clic en esta opción ya nos abre el gestor de programación donde vamos a ver que hay unas clases ya creadas que por defecto al darle nosotros en la plantilla de Windows Game 3.1 esta nos crea las clases necesarias para el desarrollo del juego.

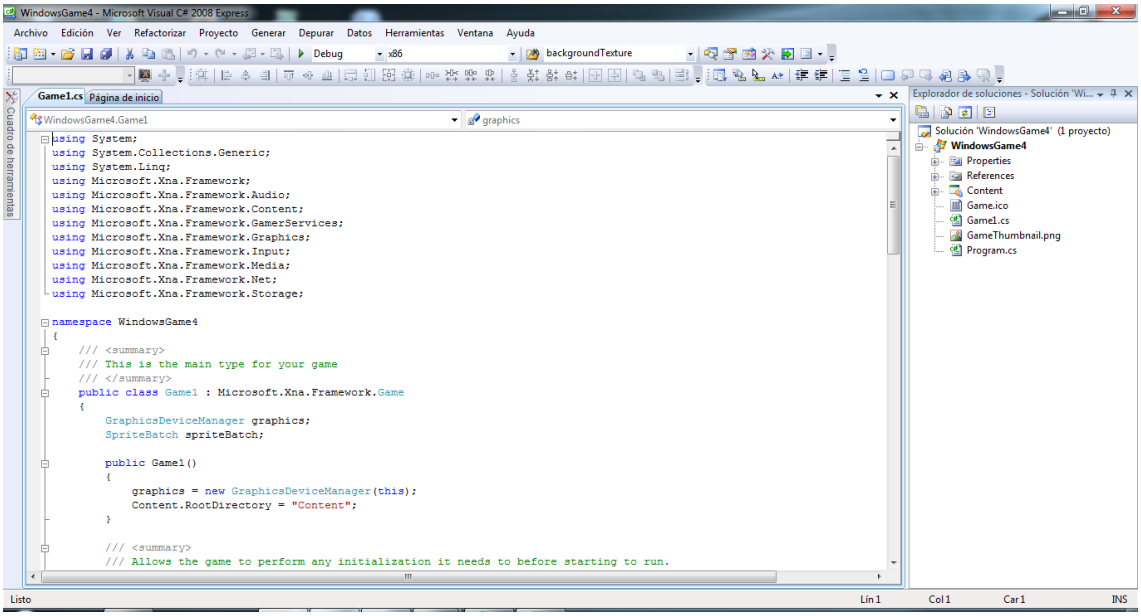


Figura 5: Vista de pantallazo inicial de codigo

Veremos que se creará un proyecto nuevo, con varios ficheros (Game1.cs, Program.cs). Vemos que ya hay código escrito, y si le damos a F5 nos saldrá una pantalla azul. Esa pantalla es nuestro futuro juego. El fichero Program.cs no debemos tocarlo.

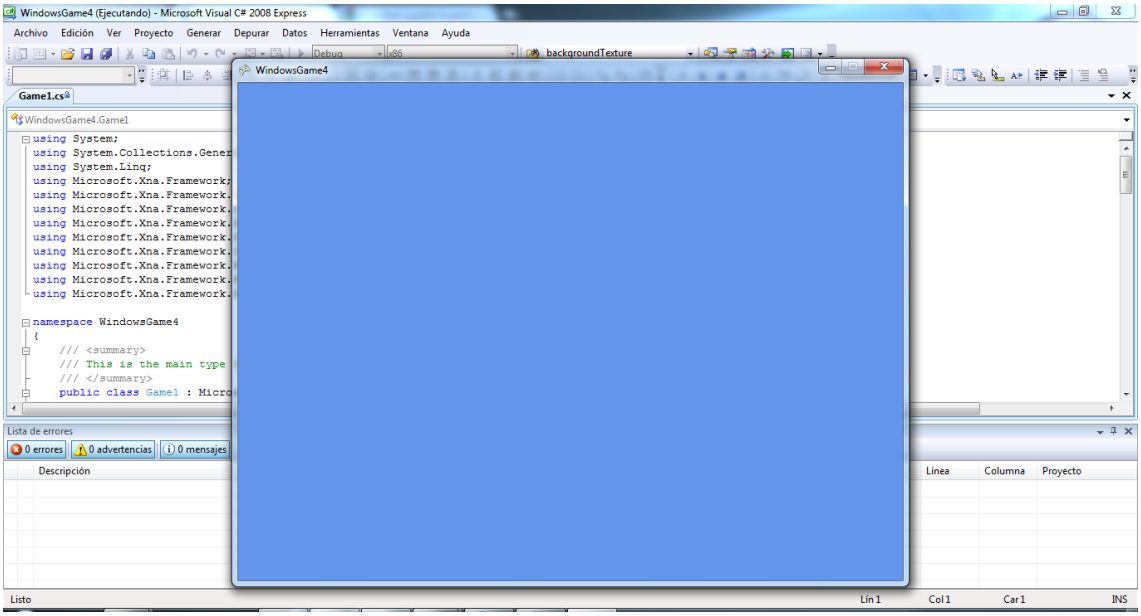


Figura 6: Vista de panel de juego

Al crear el juego se inicializan varios métodos por defecto los cuales son necesarios para la creación del juego estos métodos son:

4. CODIGO POR DEFECTO DE JUEGO

```
namespace Mundo_virtual
{
   public class Game1 : Microsoft.Xna.Framework.Game
   {
      GraphicsDeviceManager graphics;
      SpriteBatch spriteBatch;
      public Game1()
      {
         graphics = new GraphicsDeviceManager(this);
         Content.RootDirectory = "Content";
      }

      protected override void Initialize()
      {
         base.Initialize();
      }


      protected override void LoadContent()
      {
                  spriteBatch = new SpriteBatch(GraphicsDevice);

               }


      protected override void UnloadContent()
      {
               }


      protected override void Update(GameTime gameTime)
      {

         if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
            this.Exit();

               base.Update(gameTime);
      }


      protected override void Draw(GameTime gameTime)
      {
         GraphicsDevice.Clear(Color.CornflowerBlue);

                  base.Draw(gameTime);
      }
   }
}
```

5. PROGRAMACIÓN DE CADA MÉTODO

Después de que el juego crea estos métodos venimos a la programación de cada uno de ellos para la construcción de un mundo 3D.

// Con las dos barras se denotan los comentarios

Dentro de la clase public class Game1 : Microsoft.Xna.Framework.Game empezamos con la inicializacion y declaracion de las variables que necesitamos para la contruccion del mundo. Para esto vamos a inicializar las siguientes variables:

5.1 Declaración de variables

```
public class Game1 : Microsoft.Xna.Framework.Game
   {
      float angle = 0f; // declaracion de variable flotante para visualizar los angulos del mundo
virtual

      //GraphicsDeviceManager g;
      GraphicsDeviceManager graphics;// declaracion que viene por defecto al inicializar el
juego

      //BasicEffect r;
      BasicEffect cubeEffect;// declaracion tipo BasicEffect  para la creacion de cubos en el
mundo



      public Vector3 shapeSize; //  declaracion de variable tipo vector para la forma y tamaño
      public Vector3 shapePosition; // declaracion de variable tipo vector para la forma y
posicion
      private VertexPositionNormalTexture[] shapeVertices; // declaracion de variables tipo
vector posicion para declaracion de los vertices para crear la estructura
      private int shapeTriangles; // declaracion tipo entero de forma de triangulo
      private VertexBuffer shapeBuffer; // declaracion de variable para la forma del buffer

      public Texture2D shapeTexturePiso;// continuamos con la creacion de variables 2D para la
textura del piso
      public Texture2D shapeTextureLimite;//el limite
      public Texture2D shapeTextureEdificio;// el edificio


      Matrix worldMatrix; // De tipo matrix para el mundo virtual
      Matrix projectionMatrix;// y para el proyecto

      // Posicion del Avatar y rotacion de variables
      Vector3 avatarPosition = new Vector3(0, 30, -300);  // Y altura, X y Z en el plano 2d
      float avatarYaw;
      // Rotacion
      float rotationSpeed = 1f / 60f;  //velocidad horizontal y vertical--un fotograma cada 60
segundos
      float forwardSpeed = 100f / 60f; //velocidad profundidad
      // Direccion de la camara
```

```
Vector3 cameraReference = new Vector3(0, 0, 1); //desplazamiento en eje Z

MouseState originalMouseState; //estado del mouse

SpriteBatch spriteBatch;//variable por defecto de la estructura del mundo
Texture2D backgroundTexture; //de tipo 2D la declaracion de textura
```

5.2 **Declaración de clase principal**

El public game es la clase princpal y dentro de esta clase de crean todos los metodos que programamos.

Lo siguiente que debemos de programar sera

```
public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            this.graphics.PreferredBackBufferWidth = 1000;
// dimension      de la ventana de juego
            this.graphics.PreferredBackBufferHeight = 600;
// en ancho por alto
        }
```

Que es el metodo que hace la carga del entorno del juego, es decir el pantallazo principal.

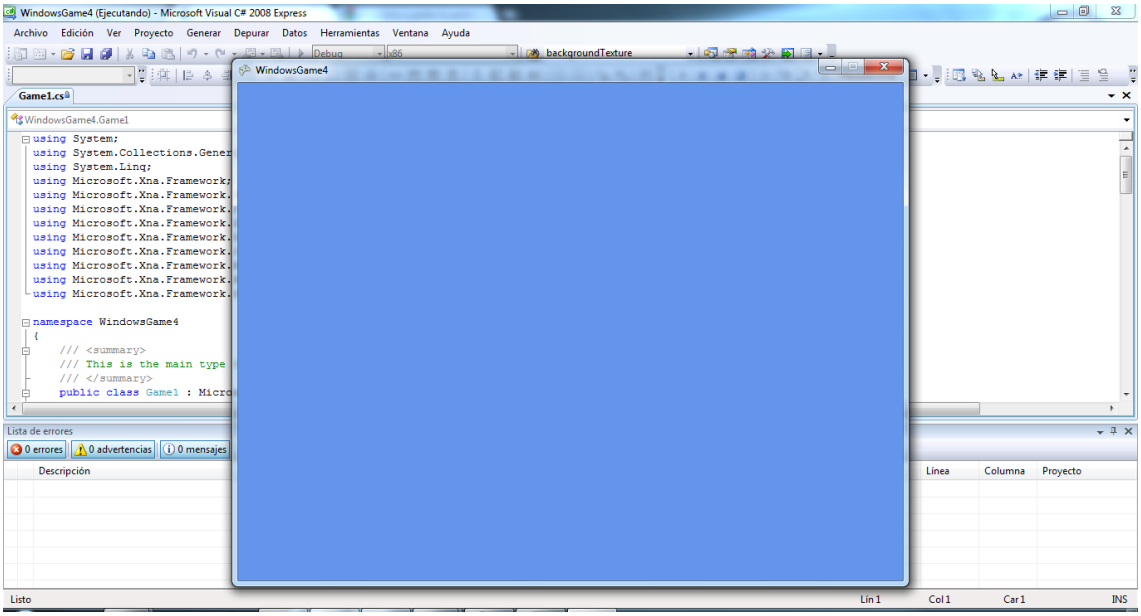


Figura 7: Entorno de juego

5.3 **Carga de texturas**

Al tener claro cuales son las imágenes con los nombres que debiamos agregar ya pasamos a agregar las imágenes fisicas al la capeta CONTENT en el explorador de soluciones dando clic derecho sobre la carpetica

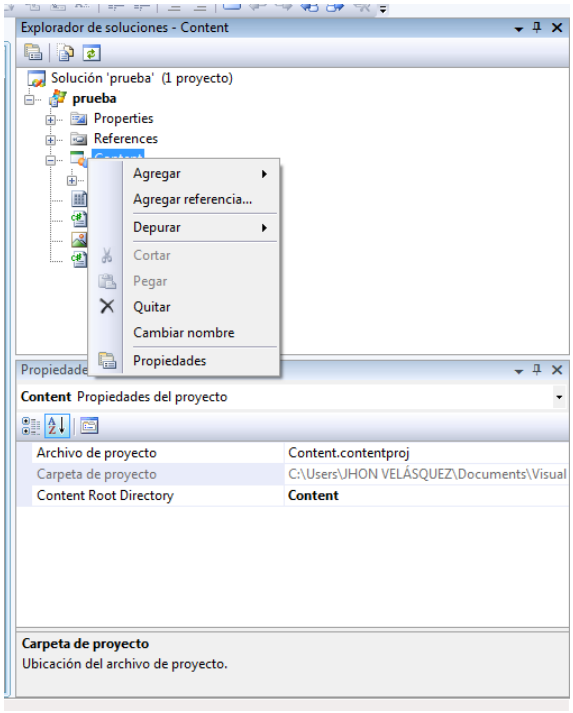


Figura 8: Adición de texturas

De aca procesimos a buscar las imágenes que necesitamos agregar al proyecto para nos puedan aparecer en el solucionario.

Le damos agregar elemento existente y buscamos en donde se encuentre el fichero o la imagen.

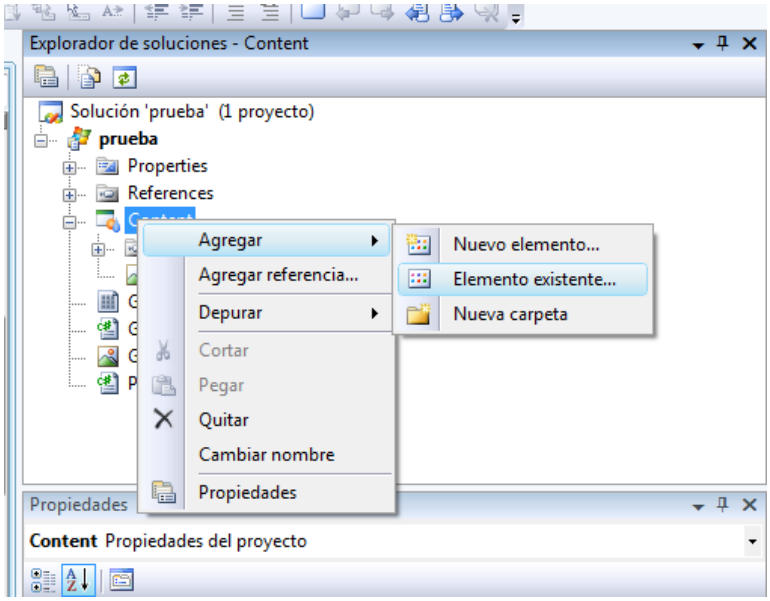


Figura 9: Adición de texturas 2

Cuando las estructuras de imagen este cargadas continuamos con la programacion del codigo.

5.4 **Programacion de método Update**

Despues de crear la estructura de carga de juego pasamos a agregar los contenidos al juego por medio del metodo

```
protected override void LoadContent()
      {
          cubeEffect = new BasicEffect(GraphicsDevice, null); // se hace carga del cubeEffect desde el load para la carga de la imagen

          cubeEffect.VertexColorEnabled = true;// declara la variable veradadera y con color para poder mostrarlo

          shapeTexturePiso = Content.Load<Texture2D>("Texture\\Roads0121_S"); // llamado de la imagen textura que se carga en el content

          shapeTextureLimite = Content.Load<Texture2D>("Texture\\Space1");// llamado de la imagen de limite

          shapeTextureEdificio = Content.Load<Texture2D>("Texture\\Edificio3");// llamado de la imagen de edificio


          spriteBatch = new SpriteBatch(GraphicsDevice);// variable que por defecto trae el juego para poder hacer la ejecucion del grafico
      }
```

El metodo siguiente es el metodo de

```
protected override void Initialize()
      {
         base.Initialize();
         initializeWorld();
         cubeEffect=new BasicEffect(GraphicsDevice,null);
      }
```

Que es un metodo para la carga inicial del juego para poder hacer la ejecucion.

Seguimos con el metodo para actualizar el juego que se programa de la siguiente manera.

5.4.1 Programación de perifericos en el metodo update

```
protected override void Update(GameTime gameTime)// metodo para la actualizacion del
mundo virtual
      {
         if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)// este
es un condicional para las presiones de teclas
            this.Exit();

         angle += 0.5f; // numero de fotogramas por segundo actualiza en la ejecucion

         UpdateAvatarPosition();// la actualizacion del avatar o el visor principal

         UpdateCamera(); // actualizacion de la camara

         base.Update(gameTime);// actulizacion de tiempo de juego
      }
```

Dentro del metodo update tambien se actualizan todos los elementos del mundo virtual a continuacion vamos a ver el metodo para los movimientos de teclado y la actualizacion del avatar.

5.4.2 Programación de posicion de avatar y movimiento

```
void UpdateAvatarPosition()
      {
         KeyboardState keyboardState = Keyboard.GetState();// se hace la cactura del teclado
         GamePadState currentState = GamePad.GetState(PlayerIndex.One);// se verifica el
estado

         if (keyboardState.IsKeyDown(Keys.Left) || (currentState.DPad.Left ==
ButtonState.Pressed))
         {
            // la programacion del condiciona para Rotate izquierda.
            avatarYaw += rotationSpeed;
```

```
        }
        if (keyboardState.IsKeyDown(Keys.Right) || (currentState.DPad.Right ==
ButtonState.Pressed))
        {
            // la programacion del condiciona paraRotate derecha.
            avatarYaw -= rotationSpeed;
        }
        if (keyboardState.IsKeyDown(Keys.Up) || (currentState.DPad.Up ==
ButtonState.Pressed))
        {
            Matrix forwardMovement = Matrix.CreateRotationY(avatarYaw);
            Vector3 v = new Vector3(0, 0, forwardSpeed);
            v = Vector3.Transform(v, forwardMovement);
            avatarPosition.Z += v.Z;
            avatarPosition.X += v.X;
            // la programacion del condiciona para moverse arriba
        }
        if (keyboardState.IsKeyDown(Keys.Down) || (currentState.DPad.Down ==
ButtonState.Pressed))
        {
            Matrix forwardMovement = Matrix.CreateRotationY(avatarYaw);
            Vector3 v = new Vector3(0, 0, -forwardSpeed);
            v = Vector3.Transform(v, forwardMovement);
            avatarPosition.Z += v.Z;
            avatarPosition.X += v.X;
            // la programacion del condiciona para moverse abajo
        }

        MouseState currentMouseState = Mouse.GetState();// programacion para la navegacion
por medio del mouse de avatar
        if (currentMouseState != originalMouseState)
        {
            float xDifference = currentMouseState.X - originalMouseState.X; // mover entre x
            float yDifference = currentMouseState.Y - originalMouseState.Y;// mover entre y
            avatarPosition.X -= rotationSpeed * xDifference;
            avatarPosition.Y -= rotationSpeed * yDifference;
            Mouse.SetPosition(graphics.GraphicsDevice.Viewport.Width / 2,
graphics.GraphicsDevice.Viewport.Height / 2);
        }
    }
```

Luego pasamos al metodo para actulizar el visor a la camara con lo que podemos observar los movimientos del mundo virtual.

5.4.3 **Programación de actualizacion de camara**

```
void UpdateCamera()
    {
        // Posicion Camara
        Vector3 cameraPosition = avatarPosition; // declaracion de varible posicion avatar de
tipo vector
```

```
        Matrix rotationMatrix = Matrix.CreateRotationY(avatarYaw); // creacion de variable tipo matrix para la rotacion del avatar

        // Direccion donde la camara apunta
        Vector3 transformedReference = Vector3.Transform(cameraReference, rotationMatrix);

        // Posicion donde la camara apunta
        Vector3 cameraLookat = cameraPosition + transformedReference;

        cubeEffect.View = Matrix.CreateLookAt(cameraPosition, cameraLookat, new Vector3(0.0f, 1.0f, 0.0f));//posicion de la camara con respecto al edificio

        Viewport viewport = graphics.GraphicsDevice.Viewport;// declaracion para visualizacion de objeto

        Mouse.SetPosition(viewport.Width / 2, viewport.Height / 2); // posicionamiento del mouse con respecto al avatar

        originalMouseState = Mouse.GetState();// estado del mouse
    }
```

6.1.1. **Programacion de metodo Draw**

```
protected override void Draw(GameTime gameTime)// el metodo para la generacion de las imagenes o dibujo de èstas
    {
        GraphicsDevice.Clear(Color.CornflowerBlue); // Color que traer por defecto el mundo virtual

        //--------------------------------------------

        backgroundTexture = Content.Load<Texture2D>(@"Texture\\Space"); // asignacion de una textura de fondo al mundo

        spriteBatch.Begin(SpriteBlendMode.AlphaBlend);// inicializacion de dibujado de la textura

        Vector2 pos = new Vector2(0, 0); // creacion de variable para la textura

        spriteBatch.Draw(this.backgroundTexture, pos, Color.White); // generacion de textura general, con posicionamiento y con color por defecto

        spriteBatch.End(); // finalizacion del dibujo

        //--------------------------------------------

        cubeEffect.TextureEnabled = true; // se hace la declaracion de la varible para que se vuelva verdadera y se mostrada en pantalla

        GraphicsDevice.RenderState.CullMode = CullMode.CullCounterClockwiseFace; // declaracion para mostrar imagen en tiempo exacta

        cubeEffect.Projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(45.0f), 1.0f, 1.0f, 1000.0f);// mostrar los radianes de la textura a inicializar
```

```
            cubeEffect.GraphicsDevice.VertexDeclaration = new VertexDeclaration(GraphicsDevice,
VertexPositionColor.VertexElements); // entrada de variables para funcionamiento de la textura

            //---------------------------------------------

// continuamos aca con la continuacion de pintar las otras estructuras en este caso se pinta la
estructura de piso de mundo virtual que tenemos que sera un cuadro en 2D
            cubeEffect.Begin();

            worldMatrix = Matrix.CreateRotationY(MathHelper.ToRadians(angle)) *
Matrix.CreateRotationX(MathHelper.ToRadians(angle));

            foreach (EffectPass pass in cubeEffect.CurrentTechnique.Passes)
            {
                pass.Begin();

                cubeEffect.Texture = shapeTextureLimite;

                shapeTriangles = 2; // se inicializa con dos triangulos porque el piso o mas bien un
cuadrado esta compuesto por dos de ellos.

                Piso();

                shapeBuffer = new VertexBuffer(GraphicsDevice,
VertexPositionNormalTexture.SizeInBytes * shapeVertices.Length, BufferUsage.WriteOnly);
                shapeBuffer.SetData(shapeVertices);

                GraphicsDevice.Vertices[0].SetSource(shapeBuffer, 0,
VertexPositionNormalTexture.SizeInBytes);
                GraphicsDevice.VertexDeclaration = new VertexDeclaration(GraphicsDevice,
VertexPositionNormalTexture.VertexElements);

                GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, shapeTriangles);
                pass.End();
                ////---------------------------
```

Despues de crear la textura de piso podemos observar lo siguiente.

Figura 10: Creación de textura piso

```
            pass.Begin();

            cubeEffect.Texture = shapeTextureEdificio;

            shapeTriangles = 8; // continuamos con la dibujada de los triangulos que componen un limite que son 4 cuadrados que son en total 8

            Limite();

            shapeBuffer = new VertexBuffer(GraphicsDevice, VertexPositionNormalTexture.SizeInBytes * shapeVertices.Length, BufferUsage.WriteOnly);
            shapeBuffer.SetData(shapeVertices);

            GraphicsDevice.Vertices[0].SetSource(shapeBuffer, 0, VertexPositionNormalTexture.SizeInBytes);
            GraphicsDevice.VertexDeclaration = new VertexDeclaration(GraphicsDevice, VertexPositionNormalTexture.VertexElements);

            GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, shapeTriangles);

            pass.End();

            ////----------------------------
```

Al hacer la creacion de la textura de limite podemos observar la construccion del limite

Figura 11: Creacion de textura limite

```
            pass.Begin();

            //texture
            cubeEffect.Texture = shapeTexturePiso;

            shapeTriangles = 12;// el numero de triangulos que compone el cubo o edificio

            Cubo();
```

```
            shapeBuffer = new VertexBuffer(GraphicsDevice,
VertexPositionNormalTexture.SizeInBytes * shapeVertices.Length, BufferUsage.WriteOnly);
            shapeBuffer.SetData(shapeVertices);

            GraphicsDevice.Vertices[0].SetSource(shapeBuffer, 0,
VertexPositionNormalTexture.SizeInBytes);
            GraphicsDevice.VertexDeclaration = new VertexDeclaration(GraphicsDevice,
VertexPositionNormalTexture.VertexElements);

            GraphicsDevice.DrawPrimitives(PrimitiveType.TriangleList, 0, shapeTriangles); //
Llamado del metodo primitivas para la construccion de las estructuras en 3D

            pass.End();

        }

        cubeEffect.End();

        base.Draw(gameTime);
    }
```

Y ya pasamos a la creacion de las clases privadas para las estructuras en este caso explicaremos la estructura de un cubo en 3D

```
private void Cubo()
    {
      shapeSize = new Vector3(3, 3, 3);
       shapePosition = new Vector3(-100, 0, 0);

       shapeVertices = new VertexPositionNormalTexture[36];// el numero de vertices que
posee el edificio

       // en este espacio es donde se hacen las caras que contiene un cuadrado
       // se dimensiona el cuadrado o el cubo de la forma como lo queramos poner, `para
poder dimensionar el edificio hay que poner todos los vertices el mismo valor
      // Vector3 topLeftFront = new Vector3(-x.0f, y.0f, -z.0f);
       Vector3 topLeftFront = new Vector3(-100.0f, 100.0f, -30.0f);
       Vector3 bottomLeftFront = new Vector3(-100.0f, 0.0f, -30.0f);

       Vector3 topRightFront = new Vector3(100.0f, 100.0f, -30.0f);
       Vector3 bottomRightFront = new Vector3(100.0f, 0.0f, -30.0f);

       Vector3 topLeftBack = new Vector3(-100.0f, 100.0f, 30.0f);
       Vector3 bottomLeftBack = new Vector3(-100.0f, 0.0f, 30.0f);

       Vector3 topRightBack = new Vector3(100.0f, 100.0f, 30.0f);
       Vector3 bottomRightBack = new Vector3(100.0f, 0.0f, 30.0f);

       //Normal
       Vector3 frontNormal =  new Vector3(0.0f, 0.0f, 10.0f); // frente en z
       Vector3 backNormal =  new Vector3(0.0f, 0.0f, -10.0f);// atras en z
       Vector3 topNormal = new Vector3(0.0f, 100.0f, 0.0f); //arriba en y
       Vector3 bottomNormal =  new Vector3(0.0f, 0.0f, 0.0f);//centro en y
       Vector3 leftNormal = new Vector3(-10.0f, 0.0f, 0.0f);// izquierda en x
       Vector3 rightNormal =   new Vector3(10.0f, 0.0f, 0.0f);//derecha en x
```

```
        //Texture
        //aplicar la textura deacuerdo a los vectores
        Vector2 textureTopLeft     = new Vector2(0.0f, 0.0f);
        Vector2 textureBottomLeft  = new Vector2(0.0f, 1.0f);
        Vector2 textureTopRight    = new Vector2(1.0f, 0.0f);
        Vector2 textureBottomRight = new Vector2(1.0f, 1.0f);

        //se hace la construccion del edificio tenniendo en cuenta la ubicacion de arriba.
        // Front face.
        shapeVertices[0] = new VertexPositionNormalTexture( topLeftFront,     frontNormal,
textureTopLeft);
        shapeVertices[1] = new VertexPositionNormalTexture( bottomLeftFront,  frontNormal,
textureBottomLeft);
        shapeVertices[2] = new VertexPositionNormalTexture( topRightFront,    frontNormal,
textureTopRight);
        shapeVertices[3] = new VertexPositionNormalTexture( bottomLeftFront,  frontNormal,
textureBottomLeft);
        shapeVertices[4] = new VertexPositionNormalTexture( bottomRightFront, frontNormal,
textureBottomRight);
        shapeVertices[5] = new VertexPositionNormalTexture( topRightFront,    frontNormal,
textureTopRight);

        // Back face.
        shapeVertices[6] = new VertexPositionNormalTexture( topLeftBack, backNormal,
textureTopRight);
        shapeVertices[7] = new VertexPositionNormalTexture( topRightBack, backNormal,
textureTopLeft);
        shapeVertices[8] = new VertexPositionNormalTexture( bottomLeftBack, backNormal,
textureBottomRight);
        shapeVertices[9] = new VertexPositionNormalTexture( bottomLeftBack, backNormal,
textureBottomRight);
        shapeVertices[10] = new VertexPositionNormalTexture( topRightBack, backNormal,
textureTopLeft);
        shapeVertices[11] = new VertexPositionNormalTexture( bottomRightBack, backNormal,
textureBottomLeft);

        // Top face.
        shapeVertices[12] = new VertexPositionNormalTexture( topLeftFront, topNormal,
textureBottomLeft);
        shapeVertices[13] =  new VertexPositionNormalTexture( topRightBack, topNormal,
textureTopRight);
        shapeVertices[14] = new VertexPositionNormalTexture( topLeftBack, topNormal,
textureTopLeft);
        shapeVertices[15] =  new VertexPositionNormalTexture( topLeftFront, topNormal,
textureBottomLeft);
        shapeVertices[16] =  new VertexPositionNormalTexture( topRightFront, topNormal,
textureBottomRight);
        shapeVertices[17] = new VertexPositionNormalTexture( topRightBack, topNormal,
textureTopRight);

        // Bottom face.
        shapeVertices[18] =  new VertexPositionNormalTexture( bottomLeftFront,
bottomNormal, textureTopLeft);
        shapeVertices[19] = new VertexPositionNormalTexture( bottomLeftBack, bottomNormal,
textureBottomLeft);
        shapeVertices[20] = new VertexPositionNormalTexture( bottomRightBack,
bottomNormal, textureBottomRight);
```

```
            shapeVertices[21] = new VertexPositionNormalTexture( bottomLeftFront, bottomNormal,
textureTopLeft);
            shapeVertices[22] = new VertexPositionNormalTexture( bottomRightBack,
bottomNormal, textureBottomRight);
            shapeVertices[23] = new VertexPositionNormalTexture( bottomRightFront,
bottomNormal, textureTopRight);

            // Left face.
            shapeVertices[24] = new VertexPositionNormalTexture( topLeftFront, leftNormal,
textureTopRight);
            shapeVertices[25] = new VertexPositionNormalTexture( bottomLeftBack, leftNormal,
textureBottomLeft);
            shapeVertices[26] = new VertexPositionNormalTexture( bottomLeftFront, leftNormal,
textureBottomRight);
            shapeVertices[27] = new VertexPositionNormalTexture( topLeftBack, leftNormal,
textureTopLeft);
            shapeVertices[28] = new VertexPositionNormalTexture( bottomLeftBack, leftNormal,
textureBottomLeft);
            shapeVertices[29] = new VertexPositionNormalTexture( topLeftFront, leftNormal,
textureTopRight);

            // Right face.
            shapeVertices[30] = new VertexPositionNormalTexture( topRightFront, rightNormal,
textureTopLeft);
            shapeVertices[31] = new VertexPositionNormalTexture( bottomRightFront, rightNormal,
textureBottomLeft);
            shapeVertices[32] = new VertexPositionNormalTexture( bottomRightBack, rightNormal,
textureBottomRight);
            shapeVertices[33] = new VertexPositionNormalTexture( topRightBack, rightNormal,
textureTopRight);
            shapeVertices[34] = new VertexPositionNormalTexture( topRightFront, rightNormal,
textureTopLeft);
            shapeVertices[35] = new VertexPositionNormalTexture( bottomRightBack, rightNormal,
textureBottomRight);
        }
```

Y por último observamos que después de hacer la creación y programación de cada método podemos observar por terminado un edificio en un mundo 3D con la utilización de Microsoft Visual Studio 2008 y la herramienta de XNA 3.1 y este es el resultado.

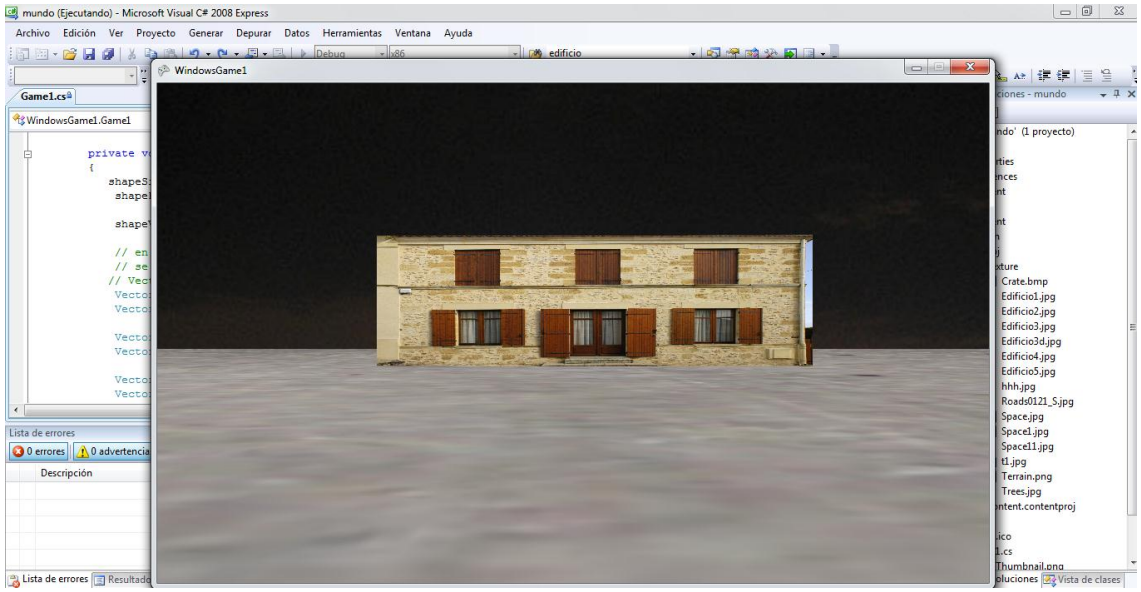


Figura 12: Estructura creada en XNA

La navegación en este mundo virtual se hace mediante el mouse que es el visualizador de cámara es decir navegación en el entorno en la perspectiva desde arriba y desde abajo del mundo virtual, en la imagen podemos visualizar la perspectiva del mundo en la cámara superior.

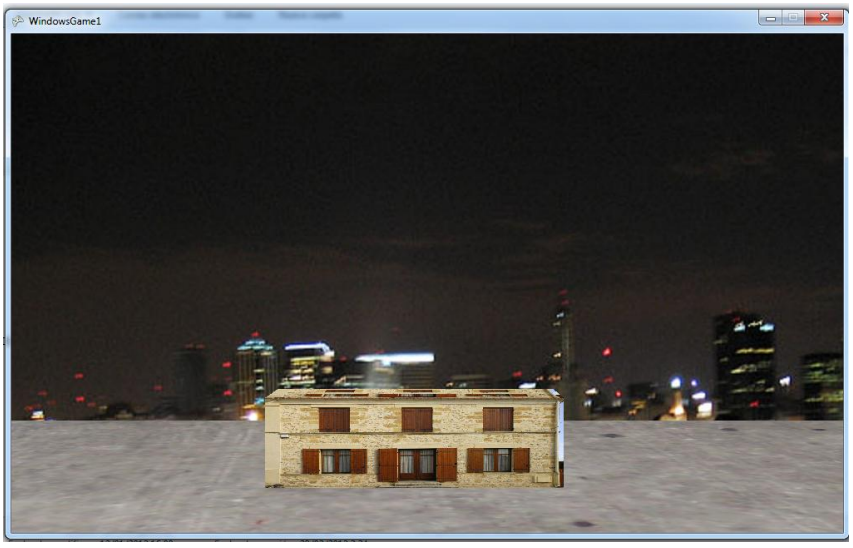
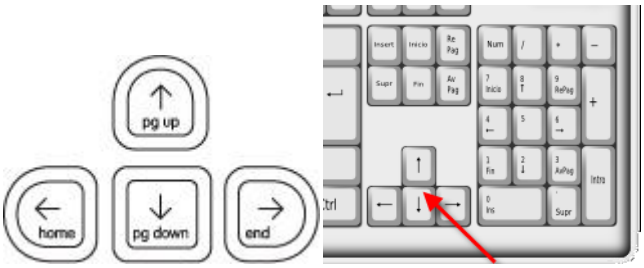


Figura 12: Navegacion mediante mouse

La otra navegación que se hace es mediante las teclas de navegación en cruz del teclado.



[1][2]Figura 13: Teclas de navegación

Estas son las que nos permiten que el avatar que está en primera persona pueda recorrer el mundo virtual o puedan hacer movimiento de rotación y la navegación se puede observar de la siguiente manera.

Figura 14: Observacion mediante teclas de navegación

En la imagen anterior podemos observar como el avatar que navega el mundo virtual esta posicionado en diferentes lugares del mundo visualizando desde diferentes perspectivas la estructura creada y el limitante del mundo que es una ciudad de fondo.

6. REFERENCIAS

[1]http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines/The_Sugar_Interface/Input_Systems/lang-es

[2]http://hernandezadrianc.blogspot.com/2010/09/las-herramientas-que-facilitan-tu.html