Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación

GRAFICACIÓN

PROYECTO 1: GRAFICADO EN ENTORNO 2D

Docente:
Prof. Iván Olmos Pineda

Alumno:
Jesús Huerta Aguilar

Matricula:
202041509

NRC: 10592
Sección: 001

CUARTO SEMESTRE

Puebla, Pue. Fecha de entrega: 27/09/2022

INDICE

INTRODUCCIÓN

OpenGL es una API de programación que a partir de primitivas geométricas simples, tales como puntos, líneas y polígonos, permite dibujar escenas tridimensionales complejas. Para ello oculta al programador las características de cada tarjeta gráfica, presentando una interfaz común, pudiendo emular hardware mediante software en el caso de que este no soporte algún tipo de función.

En esta practica se llevo a cabo lo aprendido en las ultimas clases del profesor, aplicando toda la teoría y desarrollar conceptos matriciales para poder hacer que un objeto básico o personalizado tenga traslación, rotación y/o escalamiento dentro del entorno de graficado de OpenGL, además de aplicar conocimientos y teoría matemática para las operaciones matriciales que hay para poder crear la matriz de modelado, la cual nos ayudara con el movimiento general de cualquier objeto creado.

CONCEPTOS DESARROLLADOS

Las transformaciones de traslación y rotación se conocen como transformaciones de cuerpo rígido. Estas transformaciones preservan las distancias y los ángulos.

Si a las transformaciones de cuerpo rígido les adicionamos las transformaciones de reflexión y escalado uniforme, tenemos las transformaciones de similaridad. Éstas preservan los ángulos, las distancias entre puntos cambian en una proporción fija y se mantiene una forma similar (triángulos similares, círculos mapean a círculos, …).

- ## Traslación

  Esta operación se usa para mover un objeto o grupo de objetos de manera lineal a una nueva ubicación en el espacio bidimensional.

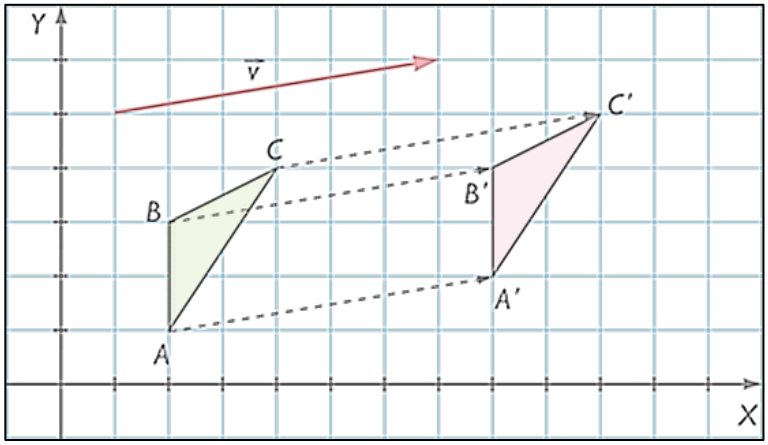


Ilustración 1:Traslación de todos los puntos de un polígono.

Trasladar un objeto una distancia t_x en x y una distancia t_y en y se expresa como:

$$\begin{cases} x = x + t_x \\ y = y + t_y \end{cases} \Longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} [OP] \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Escalado

Es una transformación que permite cambiar el tamaño o la proporción de un objeto o grupo de objetos. Hay escalados proporcionales y no proporcionales.

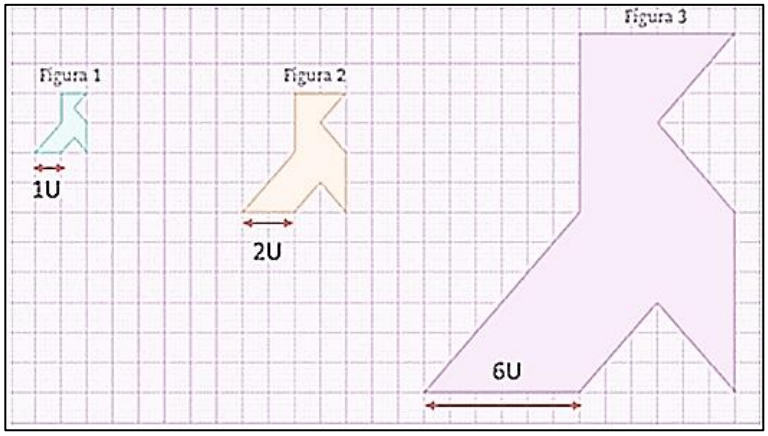


Ilustración 2: Escalado de un polígono

Escalar un objeto en s_x según x y en s_y según y se expresa como:

$$\begin{cases} x = x + s_x \\ y = y + s_y \end{cases} \Longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Rotación

Esta transformación geométrica se usa para mover un objeto o grupo de objetos alrededor de un punto.

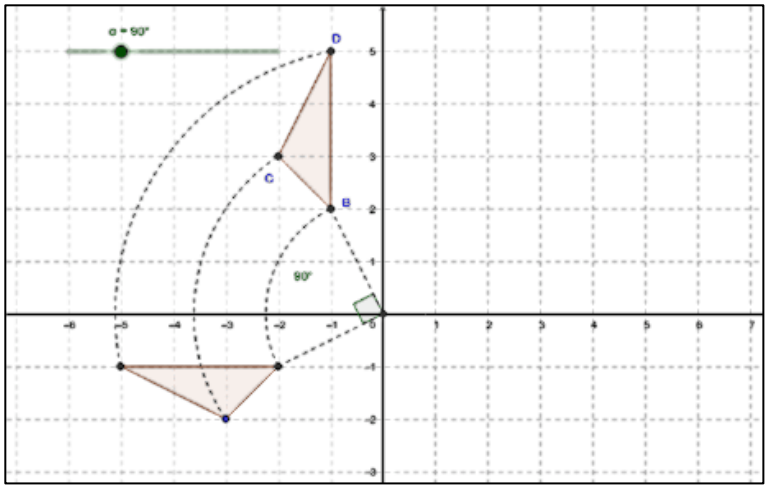


Ilustración 3:Rotación con respecto al origen.

Rotar un objeto un ángulo α en sentido horario se expresa como:

$$\begin{cases} x = x\cos\theta - y\sin\theta \\ y = x\sin\theta + y\cos\theta \end{cases} \Longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

ANALISIS EMPIRICO

En este análisis empírico solo se analizarán las clases más importantes para el funcionamiento del programa, las cuales son: main, OMatrices, RPunto, OLinea, Cuadrado, Triangulo, Circulo, Figuras, OMModelado, ModeladoC, ControlV, Stage y Background.

- Main

En la función `dibujaGrafica` creamos una variable de tipo Stage la cual tendrá acceso a la función para mostrar la clase que tiene parte de los polígonos a dibujar (la clase Stage), además, cada vez que se ejecute esa función (con ayuda de `glutIdleFunc();`) aumentaremos en uno una variable global llamada `supercont` la cual esta inicializada en cero y es de tipo entero, esta variable nos ayudara para determinar el cambio de escenas y resetear nuestras banderas usadas, ya que también el arreglo de banderas es global y lo podemos ejecutar en cualquier clase del proyecto.

Nuestros arreglos para las variables de control (`CV[20] y CVBG[20]`) son también globales y son de tipo `ControlV` ya que esto nos permitirá controlar las variables de control por cada objeto.

Código:

```
1. //BENEMERITA UNIVERSIDAD AUTONOMA DE PUEBLA -FACULTAD DE CIENCIAS DE LA
   COMPUTACION
2. //GRAFICACION - JESUS HUERTA AGUILAR "PLANISPHERE"
3. #ifdef __APPLE__
4. #include <GLUT/glut.h>
5. #else
6. #include <GL/glut.h>
7. #endif
8.
9. #include <iomanip>
10.       #include <stdlib.h>
11.
12.       #include "ControlV.h"
13.       #include "Stage.h"
14.
15.       GLfloat sizep;
16.       ///GLOBALES
17.       float angulo = 0,rad = 0,red = 0;
18.       int first = 1;
19.       int band[50];
20.       int supercont = 0;
21.       ControlV CV[20];
22.       ControlV CVBG[20];
23.
24.       //INIT
25.       void init(void){
26.           glColor3f(1.0, 1.0, 1.0);
27.           glPointSize(1.0);
28.           glClearColor(0.0, 0.0, 0.0, 1.0);
29.           glMatrixMode(GL_PROJECTION);
30.           glLoadIdentity();
31.           gluOrtho2D(-500, 500, -240, 240);
32.           glMatrixMode(GL_MODELVIEW);
33.           glLoadIdentity();
34.       }
```

```
35.       //Todos los elementos de la bandera en ceros
36.       void BanderaCero(int used){
37.           for(int i = 0; i < used;i++){
38.               band[i] = 0;
39.           }
40.       }
41.
42.       //Dibuja grafica
43.       void dibujaGrafica(){
44.           glClear(GL_COLOR_BUFFER_BIT);//borra la ventana de visualizacion
45.
46.           Stage show;
47.           show.Theater();
48.           supercont++;
49.
50.           glFlush();
51.           Sleep(30);
52.       }
53.       //Main
54.       int main(int argc, char** argv){
55.           glutInit(&argc, argv);
56.           glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
57.           //Posicion de la ventana
58.           glutInitWindowPosition(150,100);
59.           //se establece el ancho y la altura de la ventana de
   visualizacion
60.           glutInitWindowSize(1000,480);
61.           //se crea la ventana de visualizacion
62.           glutCreateWindow(":: PLANISPHERE ::");
63.           init();
64.           BanderaCero(50);
65.           glutDisplayFunc(dibujaGrafica);
66.           //Animation
67.           glutIdleFunc(dibujaGrafica);
68.           glutMainLoop();
69.           return 0;
70. }
```

- OMatrices

El propósito principal de la clase `OMatrices` es instanciar todos los componentes de todas las matrices a crear con un valor 0 flotante o también cualquier valor flotante que ingrese el usuario, `OMatrices` se utiliza en todas las demás clases a excepción de RPunto y OLinea. En la clase están definidos los *setters*, *getters* y su respectivo constructor para el único valor privado que posee.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3.
4. #ifndef OMATRICES_H
5. #define OMATRICES_H
6.
7. class OMatrices{
8.     public:
9.         //Constructores
```

```
10.             OMatrices();
11.             OMatrices(float x);
12.             virtual ~OMatrices(); //Destructor
13.             //Getter
14.             float getVal();
15.             //Setter
16.             void setVal(float);
17.         private:
18.             float val;
19.     };
20.
21. #endif // OMATRICES_H
```

Código CPP:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include "OMatrices.h"
4.
5. ///Constructores
6. OMatrices::OMatrices(){
7.     val = 0.0;
8. }
9. OMatrices::OMatrices(float x){
10.         val = x;
11.     }
12.     ///DESTRUCTOR
13.     OMatrices::~OMatrices(){}
14.
15.     //Getter
16.     float OMatrices::getVal(){
17.         return val;
18.     }
19.     //Setter
20.     void OMatrices::setVal(float x){
21.         val = x;
22. }
```

- RPunto

La clase `RPunto` asigna los valores X y Y en un punto con *setters* con *getters* obtenemos independientemente los valores del punto.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #ifndef RPUNTO_H
4. #define RPUNTO_H
5. #include "OMatrices.h"
6.
7. class RPunto{
8.     public:
9.         RPunto(); //Contructor
10.             ~RPunto(); //Destructor
11.             //Setter
12.             void setXY(OMatrices,OMatrices);
```

```
13.             void setX(OMatrices);
14.             void setY(OMatrices);
15.             //Getter
16.             OMatrices getX();
17.             OMatrices getY();
18.         private:
19.             OMatrices x,y;
20.     };
21. #endif // RPUNTO_H
```

Código CPP:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include "RPunto.h"
4. ///CONSTRUCTOR
5. RPunto::RPunto(){
6.     x = y = 0;
7. }
8. ///DESTRUCTOR
9. RPunto::~RPunto(){}
10.
11.     ///SETTERS
12.     void RPunto::setXY(OMatrices in,OMatrices in2){
13.         x = in;
14.         y = in2;
15.     }
16.     void RPunto::setX(OMatrices in){
17.         x = in;
18.     }
19.     void RPunto::setY(OMatrices in){
20.         y = in;
21.     }
22.
23.     ///GETTERS
24.     OMatrices RPunto::getX(){
25.         return x;
26.     }
27.     OMatrices RPunto::getY(){
28.         return y;
```

- OLinea

La clase `OLinea` tiene una única función para establecer una línea, la cual será parámetro de entrada (como objeto) en la clase Bresenham, `setLinea` recibe como parámetro 4 valores de tipo `OMatrices`, los cuales se guardaran respectivamente en los puntos 1 y 2.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #ifndef OLINEA_H
4. #define OLINEA_H
5. #include "RPunto.h"
6.
7. class OLinea{
8.     public:
```

```
9.          OLinea(); //Constructor
10.             ~OLinea(); //Destructor
11.             void
    setLinea(OMatrices,OMatrices,OMatrices,OMatrices);
12.             RPunto getP1();
13.             RPunto getP2();
14.           private:
15.             RPunto p1,p2;
16.       };
17. #endif // OLINEA_H
```

Código CPP:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include "OLinea.h"
4.
5. ///CONSTRUCTOR
6. OLinea::OLinea(){}
7.
8. ///DESTRUCTOR
9. OLinea::~OLinea(){}
10.
11.       ///DIVERSOS
12.       //SETLINEA
13.       //Determinamos el inicio y fin de una linea
14.       void OLinea::setLinea(OMatrices x1,OMatrices y1,OMatrices
    x2,OMatrices y2){
15.           //Asignacion de valores...
16.           //Para punto 1 (p1)
17.           p1.setX(x1);
18.           p1.setY(y1);
19.           //Para punto 2 (p2)
20.           p2.setX(x2);
21.           p2.setY(y2);
22.       }
23.       ///GETTERS PUNTOS
24.       //Punto 1
25.       RPunto OLinea::getP1(){
26.           return p1;
27.       }
28.       //Punto 2
29.       RPunto OLinea::getP2(){
30.           return p2;
31. }
```

- OMModelado

La clase mas importante de todo el proyecto, en la clase `OMModelado` están todos los operadores matriciales que dan soporte a las operaciones de traslación, rotación, escalamiento y la traslación inversa para que con el resultado final se pueda crear la matriz de modelado (la cual no se crea en esta clase en sí).

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include <vector>
4. #ifndef OMMODELADO_H
5. #define OMMODELADO_H
6. #include "OMatrices.h"
7. #include "OPuntox.h"
8.
9. class OMModelado{
10.          public:
11.              OMModelado(); //Constructor
12.              virtual ~OMModelado(); //Destructor
13.              //Diversos ([T'OPT])
14.              OPuntox traslation(float,float);
15.              OPuntox rotation(float, OPuntox);
16.              OPuntox scaling(float,float,OPuntox);
17.              OPuntox traslationInversa(float,float,OPuntox);
18.      };
19. #endif // OMMODELADO_H
```

Código CPP:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include <vector>
4. #include <cmath> //para las operaciones de rotación
5. #include "OMModelado.h"
6.
7. ///CONSTRUCTOR
8. OMModelado::OMModelado(){}
9. ///DESTRUCTOR
10.      OMModelado::~OMModelado(){}
11.
12.      ///TRASLACION
13.      //Devuelve la matriz de traslacion (OPuntox)
14.      OPuntox OMModelado::traslation(float tx,float ty){
15.
16.      ///                  [ 1 ][ 0 ][-tx]
17.      ///   Traslación =   [ 0 ][ 1 ][-ty]
18.      ///                  [ 0 ][ 0 ][ 1 ]
19.
20.          //Matriz de rotacion (de retorno)
21.          OPuntox res = OPuntox(3,3);
22.          //Diagonal de unos
23.          for(int i = 0; i < 3; i++){
24.              for(int j = 0;j < 3;j++){
25.                  if(i == j){
26.                      res.setComp(i,j,1);
27.                  }
28.              }
29.          }
30.          //Asignacion de componentes
31.          res.setComp(0,2,-1*tx);
32.          res.setComp(1,2,-1*ty);
33.          return res;
34.      }
35.
```

```
36.       ///ROTACION
37.       //Devuelve la matriz de rotacion
38.       OPuntox OMModelado::rotation(float theta, OPuntox mod){
39.
40.       ///                [Cos(θ)][-Sen(θ)][   0  ]
41.       ///   Rotación =   [Sen(θ)][ Cos(θ)][   0  ]
42.       ///                [   0  ][   0   ][   1  ]
43.
44.           //Matriz de rotacion
45.           OPuntox rota = OPuntox(3,3);
46.           //Matriz auxiliar (de retorno)
47.           OPuntox res = OPuntox(3,3);
48.           //Asignacion de componentes
49.           rota.setComp(0,0,cos(theta));
50.           rota.setComp(0,1,-1*sin(theta));
51.           rota.setComp(1,0,sin(theta));
52.           rota.setComp(1,1,cos(theta));
53.           rota.setComp(2,2,1);
54.           //Multiplicacion de matrices
55.           res = res.multMatriz(mod,rota);
56.           return res;
57.       }
58.
59.       ///ESCALAMIENTO
60.       //Devuelve la matriz de escalamiento(OPuntox)
61.       //Si sx y sy son iguales a 1 la figura no muestra
   escalamiento alguno
62.       OPuntox OMModelado::scaling(float sx,float sy,OPuntox mod){
63.
64.       ///                [sx][ 0][ 0]
65.       ///   Escalado =   [ 0][sy][ 0]
66.       ///                [ 0][ 0][ 1]
67.
68.           //Matriz de escalamiento
69.           OPuntox scal = OPuntox(3,3);
70.           //Matriz auxiliar (de retorno)
71.           OPuntox res = OPuntox(3,3);
72.           //Asignacion de componentes
73.           scal.setComp(0,0,sx);
74.           scal.setComp(1,1,sy);
75.           scal.setComp(2,2,1);
76.           //Multiplicacion de matrices
77.           res = res.multMatriz(mod,scal);
78.           return res;
79.       }
80.
81.       ///TRASLACION INVERSA (REGRESA POSICION ORIGINAL)
82.       //Devuelve la matriz de traslacion inversa (OPuntox)
83.       OPuntox OMModelado::traslationInversa(float tx,float
   ty,OPuntox mod){
84.
85.       ///                [ 1][ 0][tx]
86.       ///   Tras Inv =   [ 0][ 1][ty]
87.       ///                [ 0][ 0][ 1]
88.
89.           //Matriz de traslacion inversa
90.           OPuntox trasInv = OPuntox(3,3);
```

```
91.          //Matriz auxiliar (de retorno)
92.          OPuntox res = OPuntox(3,3);
93.          //Asignacion de componentes
94.          trasInv.setComp(0,0,1);
95.          trasInv.setComp(1,1,1);
96.          trasInv.setComp(2,2,1);
97.          trasInv.setComp(0,2,tx);
98.          trasInv.setComp(1,2,ty);
99.          //Multiplicacion de matrices
100.         res = res.multMatriz(mod,trasInv);
101.         return res;
102.}
```

- ModeladoC

En la clase `ModeladoC` creamos una matriz de 3x3 de tipo `OMatrices` la cual será la que se modificara (con las operaciones matriciales necesarias)para ser retornada como la matriz de modelado.

Código Header:

```
1. #ifndef MODELADOC_H
2. #define MODELADOC_H
3. #include "OMatrices.h"
4. #include "OPuntox.h"
5. #include "OMModelado.h"
6.
7. class ModeladoC{
8.     public:
9.         ModeladoC(); //Constructor
10.              virtual ~ModeladoC(); //Destructor
11.              OPuntox
   modeladoCreation(float,float,float,float,float,int); //Creacion de
   matriz de modelado
12.      };
13. #endif // MODELADOC_H
```

Código CPP:

```
1. #include "ModeladoC.h"
2.
3. ///CONSTRUCTOR
4. ModeladoC::ModeladoC(){}
5. ///DESTRUCTOR
6. ModeladoC::~ModeladoC(){}
7.
8. ///CREACION MATRIZ DE MODELADO
9. //Matriz donde se guardara el resultado de la multiplicacion de
   todas las matrices de [T'OPT]
10.      OPuntox ModeladoC::modeladoCreation(float tx,float ty,float
   sx,float sy,float theta,int on_off){
11.          //Llamado a operadores de modelado
12.          OMModelado op = OMModelado();
13.          //Matriz de modelado (de retorno)
14.          OPuntox model = OPuntox(3,3);
15.
```

```
16.          ///[T'OPT]
17.          // T
18.          model = op.traslation(tx,ty);
19.          // OP
20.          model = op.rotation(theta,model);
21.          model = op.scaling(sx,sy,model);
22.          // T'
23.          if(on_off == 1){
24.            model = op.traslationInversa(tx,ty,model);
25.          }
26.          return model;
27. }
```

- ControlV

Con la clase `ControlV` asignamos todas las variables de control necesarias para traslación, rotación, escalamiento y activación de la matriz de traslación, con settters y getters podemos acceder a estas y poder operarlas con un objeto de tipo `ControlV`.

Código Header:

```
1. #ifndef CONTROLV_H
2. #define CONTROLV_H
3. #include "OMatrices.h"
4. #include "OPuntox.h"
5. #include "OMModelado.h"
6.
7. class ControlV{
8.     public:
9.         ControlV(); //Constructor
10.             ~ControlV(); //Destructor
11.             //SETTER
12.             void setDT(float,float);
13.             void setDS(float,float);
14.             void setDtheta(float);
15.             void setTrasInvStatus(int);
16.             //GETTER
17.             float getDtx();
18.             float getDty();
19.             float getDsx();
20.             float getDsy();
21.             float getDtheta();
22.             int getTrasInvStatus();
23.         private:
24.             float Dtx,Dty,Dsx,Dsy,Dtheta;
25.             int on_off;
26.     };
27. #endif // CONTROLV_H
28.
```

Código CPP:

```
1. #include "ModeladoC.h"
2. #include "ControlV.h"
3.
4. ///CONSTRUCTOR
5. ControlV::ControlV(){
6.     Dtx = Dty = 0;
```

```
7.      Dsx = Dsy = 0;
8.      Dtheta = 0;
9.      on_off = 0;
10.       }
11.
12.       ///DESTRUCTOR
13.       ControlV::~ControlV(){}
14.
15.       ///SETTERS
16.       void ControlV::setDT(float in1,float in2){
17.           Dtx = in1;
18.           Dty = in2;
19.       }
20.
21.       void ControlV::setDS(float in1,float in2){
22.           Dsx = in1;
23.           Dsy = in2;
24.       }
25.
26.       void ControlV::setDtheta(float in){
27.           Dtheta = in;
28.       }
29.
30.       void ControlV::setTrasInvStatus(int in){
31.           on_off = in;
32.       }
33.
34.       ///GETTERS
35.       float ControlV::getDtx(){
36.           return Dtx;
37.       }
38.       float ControlV::getDty(){
39.           return Dty;
40.       }
41.       float ControlV::getDsx(){
42.           return Dsx;
43.       }
44.       float ControlV::getDsy(){
45.           return Dsy;
46.       }
47.       float ControlV::getDtheta(){
48.           return Dtheta;
49.       }
50.       int ControlV::getTrasInvStatus(){
51.           return on_off;
52.       }
```

- Cuadrado

Definimos 4 puntos estándar para marcar al cuadrado, los cuales se podrán modificar con las variables de control de escalamiento, creamos una arista de tipo `OLinea`, llamamos al algoritmo de Bresenham, creamos 4 variables temporales (TPX) de tipo `OMatrices` las cuales servirán para almacenar los valores de los puntos que tenga el objeto Points de tipo `OPuntox`. También se crea la matriz de modelado para poder multiplicarla con cada punto de la figura.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include <vector>
4. #ifndef CUADRADO_H
5. #define CUADRADO_H
6. #include "Bresenham.h"
7. #include "ControlV.h"
8. class Cuadrado{
9.     public:
10.             Cuadrado(); //Constructor
11.             ~Cuadrado(); //Destructor
12.             //Figuras
13.             void drawCuadrado(ControlV CV);
14.     };
15. #endif // CUADRADO_H
16.
```

Código CPP:

```
1. #ifdef __APPLE__
2. #include <GLUT/glut.h>
3. #else
4. #include <GL/glut.h>
5. #endif
6.
7. #include <iomanip>
8. #include <stdlib.h>
9. #include <vector>
10.
11.     #include "OMatrices.h"
12.     #include "OPuntox.h"
13.     #include "OMModelado.h"
14.     #include "ModeladoC.h"
15.     #include "Bresenham.h"
16.     #include "ControlV.h"
17.     #include "Cuadrado.h"
18.
19.     ///CONSTRUCTOR
20.     Cuadrado::Cuadrado(){}
21.     ///DESTRUCTOR
22.     Cuadrado::~Cuadrado(){}
23.
24.     ///DIBUJAR POLIGONO: CUADRADO
25.     void Cuadrado::drawCuadrado(ControlV CV){
26.         //Puntos a usar
27.         RPunto p[4];
28.         p[0].setX(-20); p[0].setY(-20);
29.         p[1].setX(20); p[1].setY(-20);
30.         p[2].setX(20); p[2].setY(20);
31.         p[3].setX(-20); p[3].setY(20);
32.
33.         //Linea bresenham
34.         OLinea arista;
35.         //Variables donde se guardaran los puntos
36.         OMatrices TP1,TP2,TP3,TP4;
37.         //Llamado a la creacion de matriz de modelado
38.         ModeladoC op = ModeladoC();
```

```
39.          //Llamado al algoritmo de Bresenham
40.          Bresenham line = Bresenham();
41.          //Matriz de puntos
42.          OPuntox points = OPuntox(3,1);
43.          //Matriz de modelado
44.          OPuntox MModelado = OPuntox(3,3);
45.
46.          //Creacion de matriz de modelado
47.         MModelado =
    op.modeladoCreation(0+CV.getDtx(),0+CV.getDty(),1+CV.getDsx(),1+CV.
    getDsy(),0+CV.getDtheta(),CV.getTrasInvStatus());
48.
49.          points = points.multMatrizPoints(p[0],MModelado); //Mult.
    la matriz de modelado con el punto (ax,ay)
50.          for(int i = 0; i < 4;i++){
51.              TP1 = points.getComp(0,0);  //TP1 = ax
52.              TP2 = points.getComp(1,0);  //TP2 = ay
53.              if(i == 3){
54.                  points = points.multMatrizPoints(p[0],MModelado);
    //Mult. la matriz de modelado con el punto (bx,by)
55.              }
56.              else{
57.                  points =
    points.multMatrizPoints(p[i+1],MModelado); //Mult. la matriz de
    modelado con el punto (bx,by)
58.              }
59.              TP3 = points.getComp(0,0);  //TP3 = bx
60.              TP4 = points.getComp(1,0);  //TP4 = by
61.
    arista.setLinea(TP1.getVal(),TP2.getVal(),TP3.getVal(),TP4.getVal()
    ); //Creacion de la linea
62.              line.Bresenbased(arista); //Bresenham (A a B)
63.          }
64. }
65.
```

- Triangulo

Definimos 3 puntos estándar para marcar al triangulo, los cuales se podrán modificar con las variables de control de escalamiento, creamos una arista de tipo OLinea, llamamos al algoritmo de Bresenham, creamos 4 variables temporales (TPX) de tipo OMatrices las cuales servirán para almacenar los valores de los puntos que tenga el objeto Points de tipo OPuntox. También se crea la matriz de modelado para poder multiplicarla con cada punto de la figura.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include <vector>
4. #ifndef TRIANGULO_H
5. #define TRIANGULO_H
6. #include "Bresenham.h"
7. #include "ControlV.h"
8.
9. class Triangulo{
10.          public:
11.              Triangulo(); //Constructor
```

```
12.             ~Triangulo(); //Destructor
13.             void drawTriangulo(ControlV);
14.     };
15.     #endif // TRIANGULO_H
```

Código CPP:

```
1. #ifdef __APPLE__
2. #include <GLUT/glut.h>
3. #else
4. #include <GL/glut.h>
5. #endif
6.
7. #include <iomanip>
8. #include <stdlib.h>
9. #include <vector>
10.
11.     #include "OMatrices.h"
12.     #include "OPuntox.h"
13.     #include "OMModelado.h"
14.     #include "ModeladoC.h"
15.     #include "Bresenham.h"
16.     #include "ControlV.h"
17.     #include "Triangulo.h"
18.
19.     ///CONSTRUCTOR
20.     Triangulo::Triangulo(){}
21.     ///DESTRUCTOR
22.     Triangulo::~Triangulo(){}
23.
24.     ///DIBUJAR POLIGONO: TRIANGULO
25.     //void Triangulo::drawTriangulo(float Dtx,float Dty,float
   Dsx,float Dsy,float Dtheta){
26.     void Triangulo::drawTriangulo(ControlV CV){
27.
28.         //Puntos a usar
29.         RPunto p[3];
30.         p[0].setX(-20); p[0].setY(-20);
31.         p[1].setX(20); p[1].setY(-20);
32.         p[2].setX(0); p[2].setY(20);
33.
34.         //Linea bresenham
35.         OLinea arista;
36.         //Variables donde se guardaran los puntos
37.         OMatrices TP1,TP2,TP3,TP4;
38.         //Llamado a la creacion de matriz de modelado
39.         ModeladoC op = ModeladoC();
40.         //Llamado al algoritmo de Bresenham
41.         Bresenham line = Bresenham();
42.         //Matriz de puntos
43.         OPuntox points = OPuntox(3,1);
44.         //Matriz de modelado
45.         OPuntox MModelado = OPuntox(3,3);
46.
47.         //Creacion de matriz de modelado
```

```
48.          MModelado =
   op.modeladoCreation(0+CV.getDtx(),0+CV.getDty(),1+CV.getDsx(),1+CV.
   getDsy(),0+CV.getDtheta(),CV.getTrasInvStatus());
49.
50.          points = points.multMatrizPoints(p[0],MModelado); //Mult.
   la matriz de modelado con el punto (ax,ay)
51.          for(int i = 0; i < 3;i++){
52.              TP1 = points.getComp(0,0);  //TP1 = ax
53.              TP2 = points.getComp(1,0);  //TP2 = ay
54.              if(i == 2){
55.                  points = points.multMatrizPoints(p[0],MModelado);
   //Mult. la matriz de modelado con el punto (bx,by)
56.              }
57.              else{
58.                  points =
   points.multMatrizPoints(p[i+1],MModelado); //Mult. la matriz de
   modelado con el punto (bx,by)
59.              }
60.              TP3 = points.getComp(0,0);  //TP3 = bx
61.              TP4 = points.getComp(1,0);  //TP4 = by
62.
   arista.setLinea(TP1.getVal(),TP2.getVal(),TP3.getVal(),TP4.getVal()
   ); //Creacion de la linea
63.              line.Bresenbased(arista); //Bresenham (A a B)
64.          }
65. }
```

- Circulo

Con operaciones polares creamos un circulo, el cual por cada valor X,Y generado por la transformación polar a cartesiana, se introducen como valores de entrada en la matriz de puntos, la cual se multiplicara con la matriz de modelado y ajustar sus nuevos valores.
Creamos una arista de tipo OLinea, llamamos al algoritmo de Bresenham, creamos 4 variables temporales (TPX) de tipo OMatrices las cuales servirán para almacenar los valores de los puntos que tenga el objeto Points de tipo OPuntox. También se crea la matriz de modelado para poder multiplicarla con cada punto de la figura.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include <vector>
4. #ifndef CIRCULO_H
5. #define CIRCULO_H
6. #include "Bresenham.h"
7. #include "ControlV.h"
8.
9. class Circulo{
10.          public:
11.              Circulo(); //Constructor
12.              ~Circulo(); //Destructor
13.              void drawCirculo(ControlV);
14.      };
15. #endif // CIRCULO_H
```

Código CPP:

```
1. #ifdef __APPLE__
2. #include <GLUT/glut.h>
3. #else
4. #include <GL/glut.h>
5. #endif
6.
7. #include <iomanip>
8. #include <stdlib.h>
9. #include <vector>
10.      #include <cmath> //Operaciones circulo
11.
12.      #include "OMatrices.h"
13.      #include "OPuntox.h"
14.      #include "OMModelado.h"
15.      #include "ModeladoC.h"
16.      #include "Bresenham.h"
17.      #include "Circulo.h"
18.      #include "ControlV.h"
19.
20.      ///Variables globales
21.      extern int first;
22.      extern float angulo,rad;
23.
24.      ///CONSTRUCTOR
25.      Circulo::Circulo(){}
26.      ///DESTRUCTOR
27.      Circulo::~Circulo(){}
28.
29.      ///DIBUJAR POLIGONO: CIRCULO
30.      void Circulo::drawCirculo(ControlV CV){
31.          //Puntos a usar
32.          RPunto a,b,temp;
33.          float rpolar,diff;
34.          //Linea bresenham
35.          OLinea arista;
36.          //Variables donde se guardaran los puntos
37.          OMatrices TP1,TP2,TP3,TP4;
38.           //Llamado a la creacion de matriz de modelado
39.          ModeladoC op = ModeladoC();
40.          //Llamado al algoritmo de Bresenham
41.          Bresenham line = Bresenham();
42.          //Matriz de puntos
43.          OPuntox points = OPuntox(3,1);
44.          //Matriz de modelado
45.          OPuntox MModelado = OPuntox(3,3);
46.
47.          //Creacion de matriz de modelado
48.          MModelado =
   op.modeladoCreation(0+CV.getDtx(),0+CV.getDty(),1+CV.getDsx(),1+CV.
   getDsy(),0+CV.getDtheta(),CV.getTrasInvStatus());
49.
50.          //MUESTRA GRAFICA
51.          diff = 360/90;
52.          for (int i=0;i<90;i++){
53.              rpolar = 30;
54.              temp.setX(rpolar*cos(rad));
55.              temp.setY(rpolar*sin(rad));
```

```
56.
57.             //IMPRESION DE RECTAS
58.             if(first == 1){//PRIMER PUNTO
59.                 a.setX(temp.getX());
60.                 a.setY(temp.getY());
61.                 b.setX(temp.getX());
62.                 b.setY(temp.getY());
63.                 first = 0;
64.             }
65.             else{//DEMAS PUNTOS
66.                 a.setX(b.getX());
67.                 a.setY(b.getY());
68.                 b.setX(temp.getX());
69.                 b.setY(temp.getY());
70.                 if(i > 0){
71.                     points =
   points.multMatrizPoints(a,MModelado); //Mult. la matriz de modelado
   con el punto (ax,ay)
72.                     TP1 = points.getComp(0,0);  //TP1 = x0
73.                     TP2 = points.getComp(1,0);  //TP2 = y0
74.                     points =
   points.multMatrizPoints(b,MModelado); //Mult. la matriz de modelado
   con el punto (bx,by)
75.                     TP3 = points.getComp(0,0);  //TP3 = x1
76.                     TP4 = points.getComp(1,0);  //TP2 = y1
77.
   arista.setLinea(TP1.getVal(),TP2.getVal(),TP3.getVal(),TP4.getVal()
   ); //Creacion de la linea
78.                     line.Bresenbased(arista); //Bresenham (A a B)
79.                 }
80.             }
81.             //Grados a radianes
82.             angulo = angulo + diff;
83.             rad = angulo*(M_PI/180);
84.         }
85. }
```

- Figuras

En esta clase se almacenan todos los objetos personalizados, siguiendo la misma lógica que los objetos: cuadrado, triangulo o circulo, solo que, con algunas ligeras modificaciones, por ejemplo, el cambio de color de impresión de puntos de Bresenham en un determinado rango de puntos pivote.
Las figuras están dibujadas punto por punto, como si se dibujaran sin despegar el lápiz.
Algunas funciones reciben parámetros extra, pero solo modifican el color de los puntos.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include <vector>
4. #ifndef FIGURAS_H
5. #define FIGURAS_H
6. #include "Bresenham.h"
7. #include "ControlV.h"
8.
```

```
9. class Figuras{
10.          public:
11.              Figuras(); //Constructor
12.              ~Figuras(); //Destructor
13.              //FIGURAS
14.              void drawAmongUsMuerto(ControlV,float,float,float);
15.              void drawAmongUs(ControlV,float,float,float);
16.              void drawVaso(ControlV);
17.              void drawCaja(ControlV);
18.      };
19. #endif // FIGURAS_H
20.
```

Código CPP:

```
1. #ifdef __APPLE__
2. #include <GLUT/glut.h>
3. #else
4. #include <GL/glut.h>
5. #endif
6.
7. #include <iomanip>
8. #include <stdlib.h>
9. #include <vector>
10.
11.      #include "OMatrices.h"
12.      #include "OPuntox.h"
13.      #include "OMModelado.h"
14.      #include "ModeladoC.h"
15.      #include "Bresenham.h"
16.      #include "ControlV.h"
17.
18.      #include "Cuadrado.h"
19.      #include "Triangulo.h"
20.      #include "Circulo.h"
21.      #include "Figuras.h"
22.
23.      ///CONSTRUCTOR
24.      Figuras::Figuras(){}
25.      ///DESTRUCTOR
26.      Figuras::~Figuras(){}
27.
28.      ///:::::::::::::::::::::::::::::::::::::::::::::::::::::::::///
29.      ///DIBUJAR POLIGONO: AMONG US MUERTO
30.      void Figuras::drawAmongUsMuerto(ControlV CV,float r,float
   g,float b){
31.          //Puntos a usar
32.          RPunto p[21];
33.          p[0].setXY(-2,-2);
34.          p[1].setXY(-1,-2);
35.          p[2].setXY(-1,-1);
36.          p[3].setXY(0,-1);
37.          p[4].setXY(0,-2);
38.          p[5].setXY(1,-2);
39.          p[6].setXY(1,0);
40.          p[7].setXY(0,0);
41.          p[8].setXY(0,1);
```

```
42.          p[9].setXY(0.5,1);
43.          p[10].setXY(1,1.5);
44.          p[11].setXY(0.5,2);
45.          p[12].setXY(0,1.5);
46.          p[13].setXY(-1,1.5);
47.          p[14].setXY(-1.5,2);
48.          p[15].setXY(-2,1.5);
49.          p[16].setXY(-1.5,1);
50.          p[17].setXY(-1,1);
51.          p[18].setXY(-1,0);
52.          p[19].setXY(0,0);
53.          p[20].setXY(-2,0);
54.          //Linea bresenham
55.          OLinea arista;
56.          //Variables donde se guardaran los puntos
57.          OMatrices TP1,TP2,TP3,TP4;
58.          //Llamado a la creacion de matriz de modelado
59.          ModeladoC op = ModeladoC();
60.          //Llamado al algoritmo de Bresenham
61.          Bresenham line = Bresenham();
62.          //Matriz de puntos
63.          OPuntox points = OPuntox(3,1);
64.          //Matriz de modelado
65.          OPuntox MModelado = OPuntox(3,3);
66.
67.          //Creacion de matriz de modelado
68.          MModelado =
   op.modeladoCreation(0+CV.getDtx(),0+CV.getDty(),1+CV.getDsx(),1+CV.
   getDsy(),0+CV.getDtheta(),CV.getTrasInvStatus());
69.
70.          points = points.multMatrizPoints(p[0],MModelado); //Mult.
   la matriz de modelado con el punto (ax,ay)
71.          for(int i = 0; i < 21;i++){
72.              TP1 = points.getComp(0,0);  //TP1 = ax
73.              TP2 = points.getComp(1,0);  //TP2 = ay
74.              if(i == 20){
75.                  points = points.multMatrizPoints(p[0],MModelado);
   //Mult. la matriz de modelado con el punto (bx,by)
76.              }
77.              else{
78.                  points =
   points.multMatrizPoints(p[i+1],MModelado); //Mult. la matriz de
   modelado con el punto (bx,by)
79.              }
80.              //HUESO
81.              if(i >= 7 && i < 18){
82.                  glColor3f(1.0, 1.0, 1.0);
83.              }
84.              else{
85.                  glColor3f(r,g,b);
86.              }
87.              TP3 = points.getComp(0,0);  //TP3 = bx
88.              TP4 = points.getComp(1,0);  //TP4 = by
89.
   arista.setLinea(TP1.getVal(),TP2.getVal(),TP3.getVal(),TP4.getVal()
   ); //Creacion de la linea
90.              line.Bresenbased(arista); //Bresenham (A a B)
```

```
91.           }
92.
93.       }
94.       ///:::::::::::::::::::::::::::::::::::::::::::::::::///
95.       ///DIBUJAR POLIGONO: AMONG US
96.       void Figuras::drawAmongUs(ControlV CV,float r,float g,float
   b){
97.           //Puntos a usar
98.           RPunto p[19];
99.           p[0].setXY(-2,-2);
100.          p[1].setXY(-1,-2);
101.          p[2].setXY(-1,-1);
102.          p[3].setXY(0,-1);
103.          p[4].setXY(0,-2);
104.          p[5].setXY(1,-2);
105.          p[6].setXY(1,2);
106.          p[7].setXY(-1,2);
107.          p[8].setXY(-1,1);
108.          p[9].setXY(1,1);
109.          p[10].setXY(1,2);
110.          p[11].setXY(0,3);
111.          p[12].setXY(-1,3);
112.          p[13].setXY(-2,2);
113.          p[14].setXY(-2,1.5);
114.          p[15].setXY(-2.5,1.5);
115.          p[16].setXY(-2.5,-0.5);
116.          p[17].setXY(-2,-0.5);
117.          p[18].setXY(-2,1.5);
118.          //Linea bresenham
119.          OLinea arista;
120.          //Variables donde se guardaran los puntos
121.          OMatrices TP1,TP2,TP3,TP4;
122.          //Llamado a la creacion de matriz de modelado
123.          ModeladoC op = ModeladoC();
124.          //Llamado al algoritmo de Bresenham
125.          Bresenham line = Bresenham();
126.          //Matriz de puntos
127.          OPuntox points = OPuntox(3,1);
128.          //Matriz de modelado
129.          OPuntox MModelado = OPuntox(3,3);
130.
131.          //Creacion de matriz de modelado
132.          MModelado =
   op.modeladoCreation(0+CV.getDtx(),0+CV.getDty(),1+CV.getDsx(),1+CV.
   getDsy(),0+CV.getDtheta(),CV.getTrasInvStatus());
133.
134.          points = points.multMatrizPoints(p[0],MModelado); //Mult.
   la matriz de modelado con el punto (ax,ay)
135.          for(int i = 0; i < 19;i++){
136.              TP1 = points.getComp(0,0);  //TP1 = ax
137.              TP2 = points.getComp(1,0);  //TP2 = ay
138.              if(i == 18){
139.                  points = points.multMatrizPoints(p[0],MModelado);
   //Mult. la matriz de modelado con el punto (bx,by)
140.              }
141.              else{
```

```
142.                 points =
   points.multMatrizPoints(p[i+1],MModelado); //Mult. la matriz de
   modelado con el punto (bx,by)
143.             }
144.             //VISOR
145.             if(i >= 6 && i < 10){
146.                 glColor3f(0.0, 0.0, 1.0);
147.             }
148.             else{
149.                 glColor3f(r,g,b);
150.             }
151.             TP3 = points.getComp(0,0);  //TP3 = bx
152.             TP4 = points.getComp(1,0);  //TP4 = by
153.
   arista.setLinea(TP1.getVal(),TP2.getVal(),TP3.getVal(),TP4.getVal()
   ); //Creacion de la linea
154.             line.Bresenbased(arista); //Bresenham (A a B)
155.         }
156.     }
157.     ///:::::::::::::::::::::::::::::::::::::::::::::::::::::::::///
158.     ///DIBUJAR POLIGONO: VASO
159.     void Figuras::drawVaso(ControlV CV){
160.         //Puntos a usar
161.         RPunto p[4];
162.         p[0].setXY(-10,15);
163.         p[1].setXY(10,15);
164.         p[2].setXY(5,-10);
165.         p[3].setXY(-5,-10);
166.
167.         //Linea bresenham
168.         OLinea arista;
169.         //Variables donde se guardaran los puntos
170.         OMatrices TP1,TP2,TP3,TP4;
171.         //Llamado a la creacion de matriz de modelado
172.         ModeladoC op = ModeladoC();
173.         //Llamado al algoritmo de Bresenham
174.         Bresenham line = Bresenham();
175.         //Matriz de puntos
176.         OPuntox points = OPuntox(3,1);
177.         //Matriz de modelado
178.         OPuntox MModelado = OPuntox(3,3);
179.
180.         //Creacion de matriz de modelado
181.         MModelado =
   op.modeladoCreation(0+CV.getDtx(),0+CV.getDty(),1+CV.getDsx(),1+CV.
   getDsy(),0+CV.getDtheta(),CV.getTrasInvStatus());
182.
183.         glColor3f(1,0,0);
184.         points = points.multMatrizPoints(p[0],MModelado); //Mult.
   la matriz de modelado con el punto (ax,ay)
185.         for(int i = 0; i < 4;i++){
186.             TP1 = points.getComp(0,0);  //TP1 = ax
187.             TP2 = points.getComp(1,0);  //TP2 = ay
188.             if(i == 3){
189.                 points = points.multMatrizPoints(p[0],MModelado);
   //Mult. la matriz de modelado con el punto (bx,by)
190.             }
```

```
191.             else{
192.                 points =
   points.multMatrizPoints(p[i+1],MModelado); //Mult. la matriz de
   modelado con el punto (bx,by)
193.             }
194.             //Linea blanca
195.             if(i >= 0 && i < 1){
196.                 glColor3f(1.0, 1.0, 1.0);
197.                 glPointSize(3.0);
198.             }
199.             else{
200.                 glColor3f(1,0,0);
201.                 glPointSize(1.0);
202.             }
203.             TP3 = points.getComp(0,0);  //TP3 = bx
204.             TP4 = points.getComp(1,0);  //TP4 = by
205.
   arista.setLinea(TP1.getVal(),TP2.getVal(),TP3.getVal(),TP4.getVal()
   ); //Creacion de la linea
206.             line.Bresenbased(arista); //Bresenham (A a B)
207.         }
208.     }
209.     ///:::::::::::::::::::::::::::::::::::::::::::::::::::::::::///
210.     ///DIBUJAR POLIGONO: CAJA
211.     void Figuras::drawCaja(ControlV CV){
212.         //Puntos a usar
213.         RPunto p[12];
214.         p[0].setXY(-2,1);
215.         p[1].setXY(-1,2);
216.         p[2].setXY(1,1);
217.         p[3].setXY(0,0);
218.         p[4].setXY(0,-2);
219.         p[5].setXY(1,-1);
220.         p[6].setXY(1,1);
221.         p[7].setXY(0,0);
222.         p[8].setXY(-2,1);
223.         p[9].setXY(0,0);
224.         p[10].setXY(0,-2);
225.         p[11].setXY(-2,-1);
226.         //Linea bresenham
227.         OLinea arista;
228.         //Variables donde se guardaran los puntos
229.         OMatrices TP1,TP2,TP3,TP4;
230.         //Llamado a la creacion de matriz de modelado
231.         ModeladoC op = ModeladoC();
232.         //Llamado al algoritmo de Bresenham
233.         Bresenham line = Bresenham();
234.         //Matriz de puntos
235.         OPuntox points = OPuntox(3,1);
236.         //Matriz de modelado
237.         OPuntox MModelado = OPuntox(3,3);
238.
239.         //Creacion de matriz de modelado
240.         MModelado =
   op.modeladoCreation(0+CV.getDtx(),0+CV.getDty(),1+CV.getDsx(),1+CV.
   getDsy(),0+CV.getDtheta(),CV.getTrasInvStatus());
241.
```

```
242.         points = points.multMatrizPoints(p[0],MModelado); //Mult.
   la matriz de modelado con el punto (ax,ay)
243.         for(int i = 0; i < 12;i++){
244.             TP1 = points.getComp(0,0);  //TP1 = ax
245.             TP2 = points.getComp(1,0);  //TP2 = ay
246.             if(i == 11){
247.                 points = points.multMatrizPoints(p[0],MModelado);
   //Mult. la matriz de modelado con el punto (bx,by)
248.             }
249.             else{
250.                 points =
   points.multMatrizPoints(p[i+1],MModelado); //Mult. la matriz de
   modelado con el punto (bx,by)
251.             }
252.             TP3 = points.getComp(0,0);  //TP3 = bx
253.             TP4 = points.getComp(1,0);  //TP4 = by
254.
   arista.setLinea(TP1.getVal(),TP2.getVal(),TP3.getVal(),TP4.getVal()
   ); //Creacion de la linea
255.             line.Bresenbased(arista); //Bresenham (A a B)
256.         }
257.     }
258.
```

- Background

Con esta clase se dibujan (mayormente con líneas) los objetos del paisaje que decoraran el fondo, ósea, los objetos que no son principales en la escena pero aun así siguen contando como objetos a tomar en cuenta en el proyecto.
Dentro de esta clase están elaborados objetos (no de clases) para adornar el ambiente grafico, tales como mesas, vasos de refresco, cajas, botones, ventanas y demás objetos del juego Among Us.

Con ayuda de un Switch (que funciona con el único parámetro entero de entrada de la función `BG1(int)`), determinamos que escena se mostrara en pantalla, la escena 1 es dentro de la nave y la escena 2 es cuando eyectan al tripulante impostor.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include <vector>
4. #ifndef BACKGROUND_H
5. #define BACKGROUND_H
6. #include "Bresenham.h"
7. #include "ControlV.h"
8.
9. class Background{
10.         public:
11.             Background(); //Constructor
12.             ~Background(); //Destructor
13.             void BG1(int);
14.     };
15.
16. #endif // BACKGROUND_H
17.
```

Código CPP:

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include <iomanip>
#include <stdlib.h>
#include <vector>

      #include "OMatrices.h"
      #include "OPuntox.h"
      #include "OMModelado.h"
      #include "ModeladoC.h"
      #include "Bresenham.h"
      #include "ControlV.h"

      #include "Cuadrado.h"
      #include "Triangulo.h"
      #include "Circulo.h"
      #include "Figuras.h"
      #include "KeyGlazz.h"
      #include "Background.h"

      ///CONSTRUCTOR
      Background::Background(){}
      ///DESTRUCTOR
      Background::~Background(){}

      ///GLOBALES
      extern ControlV CVBG[20];
      extern int band[50];

      void Background::BG1(int in){
          //Llamado al algoritmo de Bresenham
          Bresenham lineshow;
          //Figuras
          OLinea linea;
          Circulo O;
          Cuadrado N;
          Triangulo A;
          KeyGlazz K;
          Figuras F;
          switch(in){
          case 1:
              ///Mesa
              glColor3f(0.2, 0.0, 0.7);
              CVBG[0].setDT(270,20);
              CVBG[0].setDS(2.8,1.6);
              O.drawCirculo(CVBG[0]);
              CVBG[0].setDT(270,20);
              CVBG[0].setDS(3.6,2.4);
              O.drawCirculo(CVBG[0]);
              linea.setLinea(-270,57,-275,80);
              lineshow.Bresenbased(linea);
```

```
56.             linea.setLinea(-260,57,-255,80);
57.             lineshow.Bresenbased(linea);
58.             linea.setLinea(-158,-10,-134,-5);
59.             lineshow.Bresenbased(linea);
60.             linea.setLinea(-156,-20,-132,-25);
61.             lineshow.Bresenbased(linea);
62.             linea.setLinea(-405,-5,-382,-10);
63.             lineshow.Bresenbased(linea);
64.             linea.setLinea(-407,-25,-384,-20);
65.             lineshow.Bresenbased(linea);
66.             linea.setLinea(-275,-97,-280,-120);
67.             lineshow.Bresenbased(linea);
68.             linea.setLinea(-255,-97,-250,-120);
69.             lineshow.Bresenbased(linea);
70.
71.             //vasos mesa
72.             CVBG[0].setDS(-0.2,-0.2);
73.             CVBG[0].setDT(190,30);
74.             F.drawVaso(CVBG[0]);
75.
76.             CVBG[0].setDT(210,10);
77.             F.drawVaso(CVBG[0]);
78.
79.             CVBG[0].setDT(360,40);
80.             F.drawVaso(CVBG[0]);
81.
82.             //platos
83.             glColor3f(1,1,1);
84.             CVBG[0].setDS(-0.2,-0.4);
85.             CVBG[0].setDtheta(0.2);
86.             CVBG[0].setDT(260,-30);
87.             N.drawCuadrado(CVBG[0]);
88.             CVBG[0].setDtheta(-0.2);
89.             CVBG[0].setDT(280,70);
90.             N.drawCuadrado(CVBG[0]);
91.             CVBG[0].setDtheta(0);
92.
93.             //Contorno boton
94.             glColor3f(0.5, 0.5, 0.0);
95.             CVBG[0].setDT(269,15);
96.             CVBG[0].setDS(0.1,-0.2);
97.             N.drawCuadrado(CVBG[0]);
98.             CVBG[0].setDT(269,19);
99.             CVBG[0].setDS(0.4,0.1);
100.            N.drawCuadrado(CVBG[0]);
101.            //Boton
102.            glColor3f(1.0, 0.0, 0.0);
103.            CVBG[0].setDT(270,15);
104.            CVBG[0].setDS(-0.5,-0.6);
105.            O.drawCirculo(CVBG[0]);
106.            linea.setLinea(-284,-15,-284,-25);
107.            lineshow.Bresenbased(linea);
108.            linea.setLinea(-255,-15,-255,-25);
109.            lineshow.Bresenbased(linea);
110.            glColor3f(1.0, 1.0, 1.0);
111.
112.            //Paredes
```

```
113.            glPointSize(2.0);
114.            linea.setLinea(100,220,200,140);
115.            lineshow.Bresenbased(linea);
116.            linea.setLinea(200,140,200,50);
117.            lineshow.Bresenbased(linea);
118.            linea.setLinea(200,50,300,50);
119.            lineshow.Bresenbased(linea);
120.            linea.setLinea(300,50,300,200);
121.            lineshow.Bresenbased(linea);
122.            linea.setLinea(300,200,500,200);
123.            lineshow.Bresenbased(linea);
124.
125.            linea.setLinea(90,220,195,135);
126.            lineshow.Bresenbased(linea);
127.            linea.setLinea(195,135,195,45);
128.            lineshow.Bresenbased(linea);
129.            linea.setLinea(195,45,305,45);
130.            lineshow.Bresenbased(linea);
131.            linea.setLinea(305,45,305,195);
132.            lineshow.Bresenbased(linea);
133.            linea.setLinea(305,195,500,195);
134.            lineshow.Bresenbased(linea);
135.
136.            linea.setLinea(10,220,195,76);
137.            lineshow.Bresenbased(linea);
138.            linea.setLinea(195,45,195,-10);
139.            lineshow.Bresenbased(linea);
140.            linea.setLinea(195,-10,305,-10);
141.            lineshow.Bresenbased(linea);
142.            linea.setLinea(305,-10,305,140);
143.            lineshow.Bresenbased(linea);
144.            linea.setLinea(305,140,500,140);
145.            lineshow.Bresenbased(linea);
146.
147.            //Paredes inferiores
148.            linea.setLinea(195,-220,195,-130);
149.            lineshow.Bresenbased(linea);
150.            linea.setLinea(195,-130,305,-130);
151.            lineshow.Bresenbased(linea);
152.            linea.setLinea(305,-130,405,-220);
153.            lineshow.Bresenbased(linea);
154.
155.            linea.setLinea(200,-220,200,-135);
156.            lineshow.Bresenbased(linea);
157.            linea.setLinea(200,-135,305,-135);
158.            lineshow.Bresenbased(linea);
159.            linea.setLinea(305,-135,400,-220);
160.            lineshow.Bresenbased(linea);
161.
162.            ///Decoracion de paredes
163.            glPointSize(1.0);
164.            //Pared central
165.            CVBG[0].setDT(-248,-18);
166.            CVBG[0].setDS(1,-0.5);
167.            N.drawCuadrado(CVBG[0]);
168.            //Circulos: Pared central
169.            glColor3f(1.0,0.0,0.0);
```

```
170.            CVBG[0].setDT(-248,-19);
171.            CVBG[0].setDS(-0.8,-0.8);
172.            O.drawCirculo(CVBG[0]);
173.            glColor3f(0.0,1.0,0.0);
174.            CVBG[0].setDT(-228,-19);
175.            CVBG[0].setDS(-0.8,-0.8);
176.            O.drawCirculo(CVBG[0]);
177.            glColor3f(0.0,0.0,1.0);
178.            CVBG[0].setDT(-268,-19);
179.            CVBG[0].setDS(-0.8,-0.8);
180.            O.drawCirculo(CVBG[0]);
181.
182.            //Pared derecha
183.            //lineas gruesas
184.            glPointSize(5.0);
185.            glColor3f(1.0, 1.0, 1.0);
186.            linea.setLinea(350,168,500,168);
187.            lineshow.Bresenbased(linea);
188.            linea.setLinea(320,153,500,153);
189.            lineshow.Bresenbased(linea);
190.            linea.setLinea(380,183,500,183);
191.            lineshow.Bresenbased(linea);
192.            //keyglazz
193.            glPointSize(1.0);
194.            CVBG[0].setDT(-330,-173);
195.            CVBG[0].setDS(0.2,0.2);
196.            K.drawKeyGlazz(CVBG[0]);
197.
198.            ///DECORACION SUELO
199.            //Cuadrado moviendose
200.            if(band[5] == 0){
201.                CVBG[4].setDT(-350,-100);
202.                band[5] = 1;
203.            }
204.            else{
205.                if(band[6] == 0){
206.                    CVBG[4].setDT(CVBG[4].getDtx() - 10,-100);
207.                    if(CVBG[4].getDtx() < -455){
208.                        band[6] = 1;
209.                    }
210.                }
211.                else{
212.                    CVBG[4].setDT(CVBG[4].getDtx() + 10,-100);
213.                    if(CVBG[4].getDtx() > -350){
214.                        band[6] = 0;
215.                    }
216.                }
217.            }
218.            N.drawCuadrado(CVBG[4]);
219.            linea.setLinea(320,80,500,80);
220.            lineshow.Bresenbased(linea);
221.
222.            glPointSize(3.0);
223.            linea.setLinea(320,70,500,70);
224.            lineshow.Bresenbased(linea);
225.            glPointSize(1.0);
226.
```

```
227.
228.            //Circulo estirandose
229.            glColor3f(0.3,0.3,0.9);
230.            if(band[4] == 0){
231.                CVBG[5].setDT(-400,-10);
232.                CVBG[5].setDS(-0.5,-1);
233.                band[4] = 1;
234.            }
235.            else{
236.                if(band[7] == 0){
237.                    CVBG[5].setDS(CVBG[5].getDsx() -
    0.05,CVBG[5].getDsy() - 0.05);
238.                    if(CVBG[5].getDsx() < -0.5){
239.                        band[7] = 1;
240.                    }
241.                }
242.                else{
243.                    CVBG[5].setDS(CVBG[5].getDsx() +
    0.05,CVBG[5].getDsy() + 0.05);
244.                    if(CVBG[5].getDsx() > 0){
245.                        band[7] = 0;
246.                    }
247.                }
248.            }
249.            O.drawCirculo(CVBG[5]);
250.
251.            glColor3f(1,1,1);
252.            CVBG[0].setDT(-400,-10);
253.            CVBG[0].setDS(1,0);
254.            N.drawCuadrado(CVBG[0]);
255.            CVBG[0].setDT(-400,-5);
256.            CVBG[0].setDS(1.4,0.4);
257.            N.drawCuadrado(CVBG[0]);
258.
259.            //ventana
260.            glColor3f(1.0,1.0,1.0);
261.            linea.setLinea(100,200,150,160);
262.            lineshow.Bresenbased(linea);
263.            linea.setLinea(100,160,150,120);
264.            lineshow.Bresenbased(linea);
265.            linea.setLinea(100,200,100,160);
266.            lineshow.Bresenbased(linea);
267.            linea.setLinea(150,160,150,120);
268.            lineshow.Bresenbased(linea);
269.            //Estrellas
270.            CVBG[3].setDS(-0.8,-0.8);
271.            CVBG[3].setDT(-110,-180);
272.            CVBG[3].setDtheta(0);
273.            A.drawTriangulo(CVBG[3]);
274.            CVBG[3].setDT(-110,-180+4);
275.            CVBG[3].setDtheta(3.141);
276.            A.drawTriangulo(CVBG[3]);
277.            //Estrellitas
278.            glPointSize(3.0);
279.            glBegin(GL_POINTS);
280.                glVertex2i(130,160);
281.                glVertex2i(140,140);
```

```
282.            glEnd();
283.            glPointSize(1.0);
284.
285.
286.            //Escotilla
287.            glColor3f(1.0,0.0,0.0);
288.            glPointSize(2.0);
289.            CVBG[2].setDT(-130,140);
290.            CVBG[2].setDS(0.1,-0.1);
291.            N.drawCuadrado(CVBG[2]);
292.            glColor3f(1.0,1.0,1.0);
293.            glPointSize(1.0);
294.            CVBG[1].setDT(-130,140);
295.            CVBG[1].setDS(0,-0.2);
296.            N.drawCuadrado(CVBG[1]);
297.
298.            //lineas escotilla
299.            linea.setLinea(120,-140,140,-140);
300.            lineshow.Bresenbased(linea);
301.            linea.setLinea(120,-135,140,-135);
302.            lineshow.Bresenbased(linea);
303.            linea.setLinea(120,-145,140,-145);
304.            lineshow.Bresenbased(linea);
305.
306.            break;
307.        case 2:
308.            //Estrellas
309.            glColor3f(1.0,1.0,1.0);
310.            CVBG[3].setDS(-0.8,-0.8);
311.
312.            CVBG[3].setDT(120,90);
313.            CVBG[3].setDtheta(0);
314.            A.drawTriangulo(CVBG[3]);
315.            CVBG[3].setDT(121,90+4);
316.            CVBG[3].setDtheta(3.141);
317.            A.drawTriangulo(CVBG[3]);
318.
319.            CVBG[3].setDT(400,150);
320.            CVBG[3].setDtheta(0);
321.            A.drawTriangulo(CVBG[3]);
322.            CVBG[3].setDT(400,150+4);
323.            CVBG[3].setDtheta(3.141);
324.            A.drawTriangulo(CVBG[3]);
325.
326.            CVBG[3].setDT(-400,-150);
327.            CVBG[3].setDtheta(0);
328.            A.drawTriangulo(CVBG[3]);
329.            CVBG[3].setDT(-400,-150+4);
330.            CVBG[3].setDtheta(3.141);
331.            A.drawTriangulo(CVBG[3]);
332.
333.            CVBG[3].setDT(-330,-90);
334.            CVBG[3].setDtheta(0);
335.            A.drawTriangulo(CVBG[3]);
336.            CVBG[3].setDT(-330,-90+4);
337.            CVBG[3].setDtheta(3.141);
338.            A.drawTriangulo(CVBG[3]);
```

```
339.
340.                 CVBG[3].setDT(-390,170);
341.                 CVBG[3].setDtheta(0);
342.                 A.drawTriangulo(CVBG[3]);
343.                 CVBG[3].setDT(-390,170+4);
344.                 CVBG[3].setDtheta(3.141);
345.                 A.drawTriangulo(CVBG[3]);
346.
347.                 CVBG[3].setDT(270,-170);
348.                 CVBG[3].setDtheta(0);
349.                 A.drawTriangulo(CVBG[3]);
350.                 CVBG[3].setDT(270,-170+4);
351.                 CVBG[3].setDtheta(3.141);
352.                 A.drawTriangulo(CVBG[3]);
353.
354.                 CVBG[3].setDT(450,-50);
355.                 CVBG[3].setDtheta(0);
356.                 A.drawTriangulo(CVBG[3]);
357.                 CVBG[3].setDT(450,-50+4);
358.                 CVBG[3].setDtheta(3.141);
359.                 A.drawTriangulo(CVBG[3]);
360.
361.                 CVBG[3].setDT(-170,110);
362.                 CVBG[3].setDtheta(0);
363.                 A.drawTriangulo(CVBG[3]);
364.                 CVBG[3].setDT(-170,110+4);
365.                 CVBG[3].setDtheta(3.141);
366.                 A.drawTriangulo(CVBG[3]);
367.
368.                 //Estrellitas
369.                 glPointSize(3.0);
370.                 glBegin(GL_POINTS);
371.                     glVertex2i(10,-150);
372.                     glVertex2i(-10,150);
373.                     glVertex2i(370,-80);
374.                     glVertex2i(-290,110);
375.                     glVertex2i(-330,-90);
376.                     glVertex2i(190,90);
377.                 glEnd();
378.                 break;
379.             }
380.         }
381.
```

- Stage

La clase `Stage` es el escenario principal donde se muestran los dibujos de `Background` además de mostrar las todas figuras principales que no están en `Background`, como los tripulantes de la nave, algunos objetos que se mueven y decoran el ambiente, el cambio de escena con ayuda de `supercont`.

Código Header:

```
1. #include <iomanip>
2. #include <stdlib.h>
3. #include <vector>
4. #ifndef STAGE_H
```

```
5. #define STAGE_H
6. #include "Bresenham.h"
7.
8. class Stage{
9.     public:
10.             Stage(); //Contructor
11.             ~Stage(); //Destructor
12.             void Theater();
13.     };
14.
15.     #endif // STAGE_H
16.
```

Código CPP:

```
1. #ifdef __APPLE__
2. #include <GLUT/glut.h>
3. #else
4. #include <GL/glut.h>
5. #endif
6.
7. #include <iomanip>
8. #include <stdlib.h>
9.
10.     #include "ModeladoC.h"
11.     #include "Bresenham.h"
12.     #include "ControlV.h"
13.     #include "Stage.h"
14.
15.     #include "Cuadrado.h"
16.     #include "Triangulo.h"
17.     #include "Circulo.h"
18.     #include "KeyGlazz.h"
19.     #include "Figuras.h"
20.     #include "Background.h"
21.
22.     ///GLOBALES
23.     extern ControlV CV[20];
24.     extern int band[50],supercont;
25.     extern float red;
26.
27.     ///CONSTRUCTOR
28.     Stage::Stage(){}
29.
30.     ///DESTRUCTOR
31.     Stage::~Stage(){}
32.
33.     ///ESCENARIO
34.     void Stage::Theater(){
35.         //Llamado al algoritmo de Bresenham
36.         Bresenham lineshow;
37.         //Llamado a las funciones de figuras
38.         Background fondo;
39.         //Figuras
40.         OLinea linea;
41.         Cuadrado N;
42.         Triangulo A;
```

```
43.          Circulo O;
44.          KeyGlazz K;
45.          Figuras F;
46.
47.      ///BACKGROUND GENERAL
48.          glColor3f(1.0, 1.0, 1.0);
49.          glPointSize(1.0);
50.          linea.setLinea(-500,220,500,220);
51.          lineshow.Bresenbased(linea);
52.          linea.setLinea(-500,-220,500,-220);
53.          lineshow.Bresenbased(linea);
54.
55.          //Marca de agua xd
56.          CV[20].setDT(475,-190);
57.          CV[20].setDS(1,1);
58.          K.drawKeyGlazz(CV[20]);
59.
60.          //Alerta
61.          glPointSize(19.0);
62.          glColor3f(red, 0.0, 0.0);
63.          linea.setLinea(-500,230,500,230);
64.          lineshow.Bresenbased(linea);
65.          linea.setLinea(-500,-230,500,-230);
66.          lineshow.Bresenbased(linea);
67.          glColor3f(1.0, 1.0, 1.0);
68.
69.          glPointSize(1.0);
70.
71.      ///  - - -  A C T O   U N O  - - -
72.
73.          if(supercont < 330){ //330
74.              //Alerta barra superior e inferior
75.              if(supercont > 250 && supercont < 330){
76.                  if(band[8] < 1){
77.                      red = red + 0.1;
78.                      if(red > 1){
79.                          band[8] = 1;
80.                      }
81.                  }
82.                  else{
83.                      red = red - 0.1;
84.                      if(red < 0){
85.                          band[8] = 0;
86.                      }
87.                  }
88.              }
89.              ///BACKGROUND
90.              fondo.BG1(1);
91.              /// FIGURA 1
92.              CV[1].setDT(100,60);
93.              CV[1].setDS(10,10);
94.              F.drawAmongUs(CV[1],1,0,0);
95.              /// FIGURA 2
96.              //CV[1].setDT(350,-120);
97.              CV[5].setDS(10,10);
98.              F.drawAmongUs(CV[5],1,0,1);
99.
```

```
100.            if(band[9] == 0){
101.                CV[5].setDT(500,-120);
102.                band[9] = 1;
103.            }
104.            else{
105.                if(band[10] == 0){
106.                    CV[5].setDT(CV[5].getDtx() - 2,-120);
107.                    if(CV[5].getDtx() < 260){
108.                        band[10] = 1;
109.                    }
110.                }
111.            }
112.
113.            /// FIGURA 3
114.            //El among que muere
115.            if(supercont > 250){
116.                CV[1].setDT(-100,-30);
117.                CV[1].setDS(10,10);
118.                F.drawAmongUsMuerto(CV[1],1,1,0);
119.
120.                //Signo admiracion
121.                CV[1].setDT(105,0);
122.                CV[1].setDS(-0.7,-0.7);
123.                glColor3f(1.0,0.0,0.0);
124.                O.drawCirculo(CV[1]);
125.
126.                CV[1].setDT(105,-40);
127.                CV[1].setDS(-0.6,0.2);
128.                N.drawCuadrado(CV[1]);
129.
130.            }
131.            else{
132.                CV[1].setDT(-100,-30);
133.                CV[1].setDS(10,10);
134.                F.drawAmongUs(CV[1],1,1,0);
135.            }
136.            /// FIGURA 4
137.            //Asesino
138.            CV[2].setDS(10,10);
139.            CV[2].setDtheta(0);
140.            F.drawAmongUs(CV[2],0,1,0);
141.            //Punto inicial
142.            if(band[0] == 0){
143.                CV[2].setDT(-400,90);
144.                band[0] = 1;
145.            }
146.            else{
147.                //Movimiento en x
148.                if(band[1] == 0){
149.                    CV[2].setDT(CV[2].getDtx() + 3,90);
150.                    if(CV[2].getDtx() > -60){
151.                        band[1] = 1;
152.                    }
153.                }
154.                //Movimiento en y
155.                else if(band[1] == 1){
```

```
156.                     CV[2].setDT(CV[2].getDtx(),CV[2].getDty() -
   1);
157.                     if(CV[2].getDty() < -30){
158.                         band[1] = 2;
159.                     }
160.                 }
161.             }
162.             //Cajungas
163.             glColor3f(1.0,1.0,1.0);
164.             CV[3].setDT(200,-150);
165.             CV[3].setDS(10,10);
166.             F.drawCaja(CV[3]);
167.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 18,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 18);
168.             lineshow.Bresenbased(linea);
169.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 15,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 15);
170.             lineshow.Bresenbased(linea);
171.             CV[4].setDT(CV[3].getDtx() + 7,CV[3].getDty() + 6);
172.             CV[4].setDS(-0.9,-0.9);
173.             O.drawCirculo(CV[4]);
174.
175.             CV[3].setDT(150,-150);
176.             CV[3].setDS(10,10);
177.             F.drawCaja(CV[3]);
178.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 18,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 18);
179.             lineshow.Bresenbased(linea);
180.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 15,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 15);
181.             lineshow.Bresenbased(linea);
182.             CV[4].setDT(CV[3].getDtx() + 7,CV[3].getDty() + 6);
183.             CV[4].setDS(-0.9,-0.9);
184.             O.drawCirculo(CV[4]);
185.
186.             CV[3].setDT(100,-150);
187.             CV[3].setDS(10,10);
188.             F.drawCaja(CV[3]);
189.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 18,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 18);
190.             lineshow.Bresenbased(linea);
191.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 15,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 15);
192.             lineshow.Bresenbased(linea);
193.             CV[4].setDT(CV[3].getDtx() + 7,CV[3].getDty() + 6);
194.             CV[4].setDS(-0.9,-0.9);
195.             O.drawCirculo(CV[4]);
196.
197.             CV[3].setDT(450,180);
198.             CV[3].setDS(10,10);
199.             F.drawCaja(CV[3]);
200.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 18,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 18);
201.             lineshow.Bresenbased(linea);
202.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 15,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 15);
203.             lineshow.Bresenbased(linea);
```

```
204.             CV[4].setDT(CV[3].getDtx() + 7,CV[3].getDty() + 6);
205.             CV[4].setDS(-0.9,-0.9);
206.             O.drawCirculo(CV[4]);
207.
208.             CV[3].setDT(40,180);
209.             CV[3].setDS(10,10);
210.             F.drawCaja(CV[3]);
211.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 18,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 18);
212.             lineshow.Bresenbased(linea);
213.             linea.setLinea(-CV[3].getDtx() -20,-CV[3].getDty() +
   10 - 15,-CV[3].getDtx() -2 ,-CV[3].getDty() + 1 - 15);
214.             lineshow.Bresenbased(linea);
215.             CV[4].setDT(CV[3].getDtx() + 7,CV[3].getDty() + 6);
216.             CV[4].setDS(-0.9,-0.9);
217.             O.drawCirculo(CV[4]);
218.         }
219.         else{
220.
221.     ///  - - -  A C T O   D O S  - - -
222.
223.             ///BACKGROUND
224.             fondo.BG1(2);
225.             red = 0; //Barra superior e inferior de color negro
226.             glPointSize(1.0);
227.             //Amongus rodando y trasladandose de izquierda a
   derecha
228.             CV[2].setDtheta(CV[2].getDtheta()+0.1);
229.             F.drawAmongUs(CV[2],0,1,0);
230.             if(band[2] == 0){
231.                 CV[2].setDT(500,0);
232.                 band[2] = 1;
233.             }
234.             else{
235.                 if(band[3] == 0){
236.                     CV[2].setDT(CV[2].getDtx() - 5,0);
237.                     if(CV[2].getDtx() < -530){
238.                         band[3] = 1;
239.                     }
240.                 }
241.             }
242.             //Nombre del jugador
243.             glColor3f(1.0,1.0,1.0);
244.             glPointSize(3.0);
245.             CV[1].setDT(0,50);
246.             if(CV[1].getDsx() < 5)
247.             CV[1].setDS(CV[1].getDsx() + 0.1,-0.9);
248.             N.drawCuadrado(CV[1]);
249.
250.             //Reset
251.             if(supercont > 600){
252.                 supercont = 0;
253.                 band[0] = 0;
254.                 band[1] = 0;
255.                 band[2] = 0;
256.                 band[3] = 0;
257.                 band[4] = 0;
```

```
258.                band[5] = 0;
259.                band[6] = 0;
260.                band[7] = 0;
261.                band[8] = 0;
262.            }
263.        }
264.    }
```

¿QUE SE DIBUJÓ?

Among Us es un juego muy parecido al clásico "El asesino" que hemos jugado alguna vez en cartas, pero llevado al mundo de los videojuegos. Estás en una nave espacial, y dos impostores que están "entre nosotros", de ahí el título, tienen que matar a los otros ocho tripulantes antes de que estos los descubran.

Lo que se implementó fue una forma muy resumida del juego, donde un tripulante (el tripulante verde) asesina a otro tripulante (el amarillo), pero otro tripulante (el rojo) se da cuenta de los hechos, así que decide, junto con los demás tripulantes (el rosa, por ejemplo) sacarlo de la nave ya que representa una amenaza para el resto de la tripulación.

Ilustración 4: Juego "Among Us" el cual se basó el grafo

El objetivo principal de un Tripulante es completar todas las tareas sin ser asesinado por un Impostor, con el objetivo secundario de encontrar a todos los Impostores y expulsarlos del mapa. Los Tripulantes que han sido asesinados por un Impostor, o expulsados, se convierten en fantasmas.

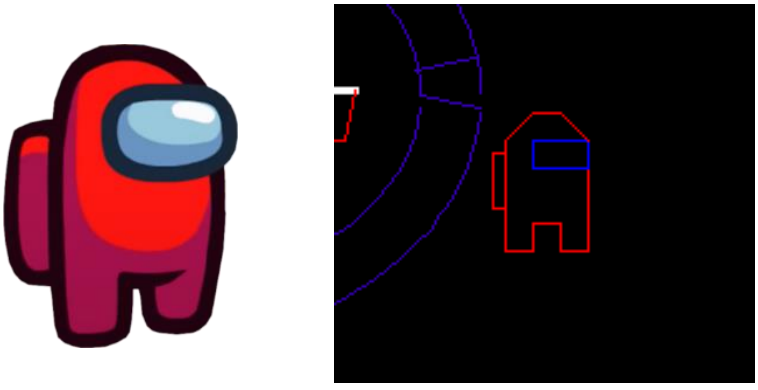


Ilustración 5: Comparación del tripulante original al graficado con OpneGL

El escenario fue basado en una sección de la nave del juego, específicamente la cafetería, se trató de que el graficado sea lo mas fiel al juego, incluyendo sus efectos visuales cuando un tripulante es notificado como muerto ante la demás tripulación, además de añadir los efectos de eyección del tripulante impostor (el malo).

Las siguientes imágenes muestras los dos escenarios del juego original recreados con OpenGL.

Cafetería:

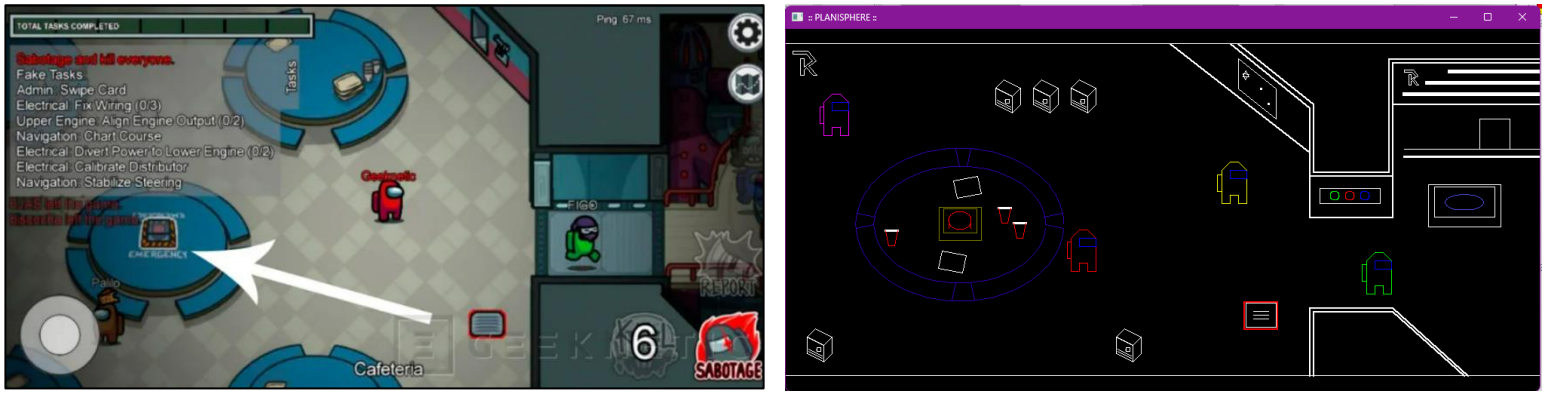


Ilustración 8: cafetería del juego original comparada con la graficada en OpenGL.

Eyección:

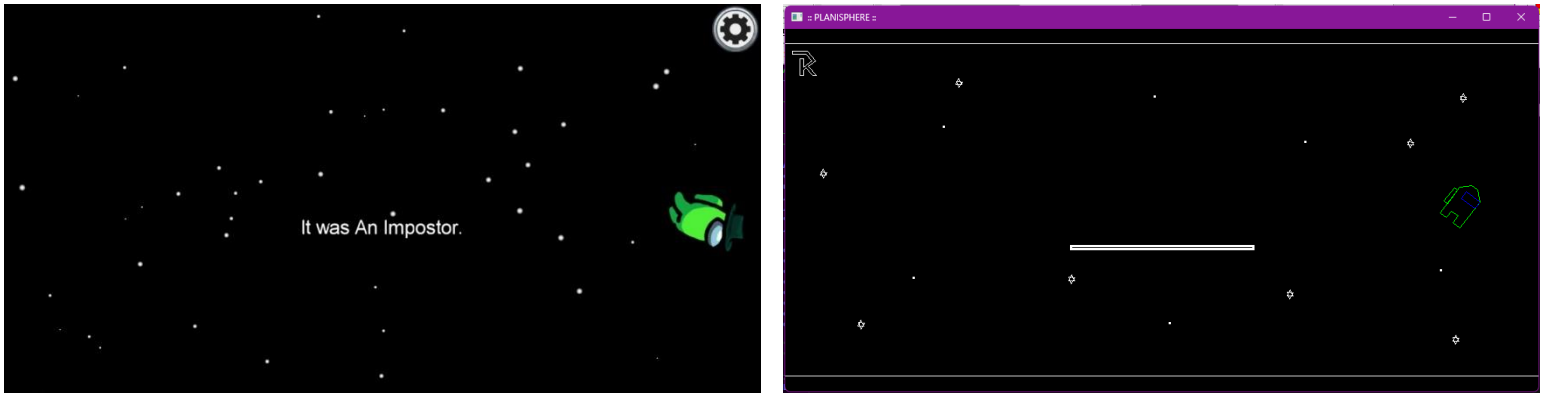
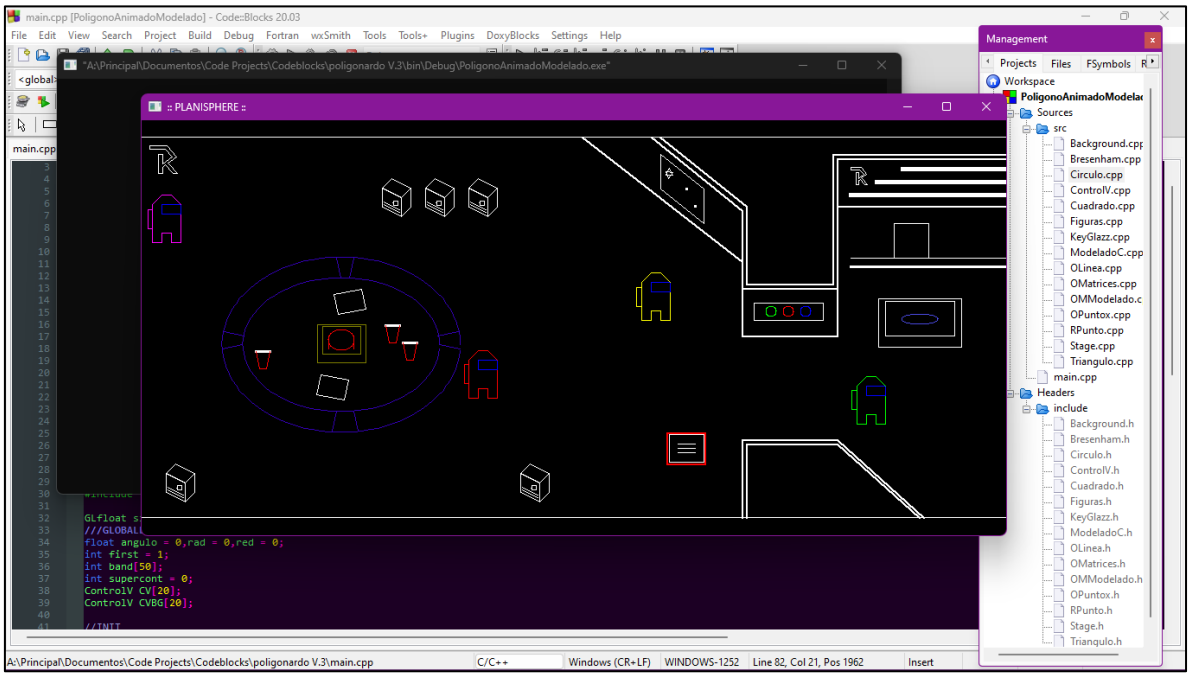
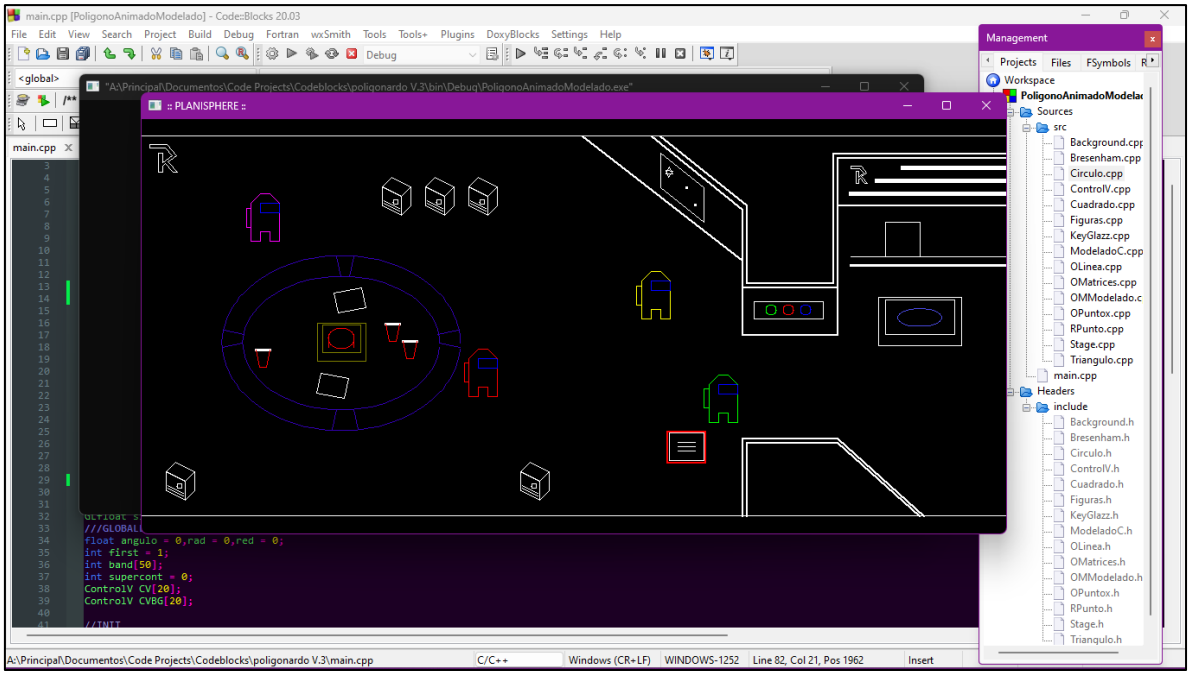


Ilustración 9: Eyección en el juego original comparado con el graficado en OpenGL.

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

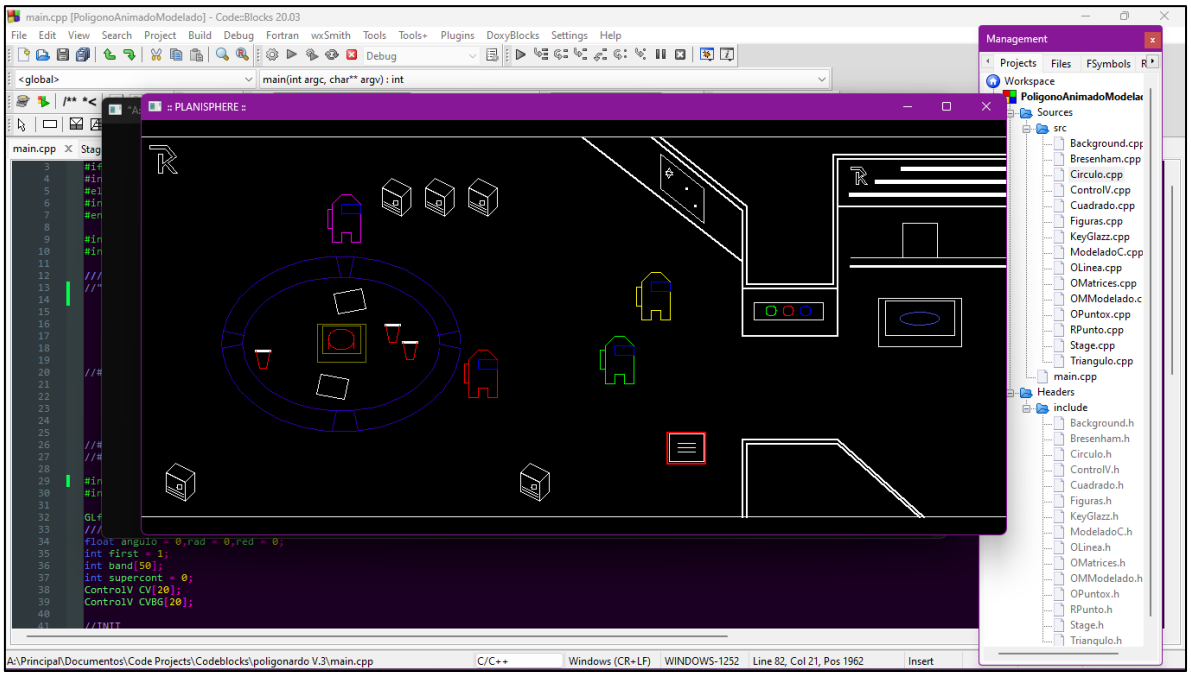
EJECUCIÓNES

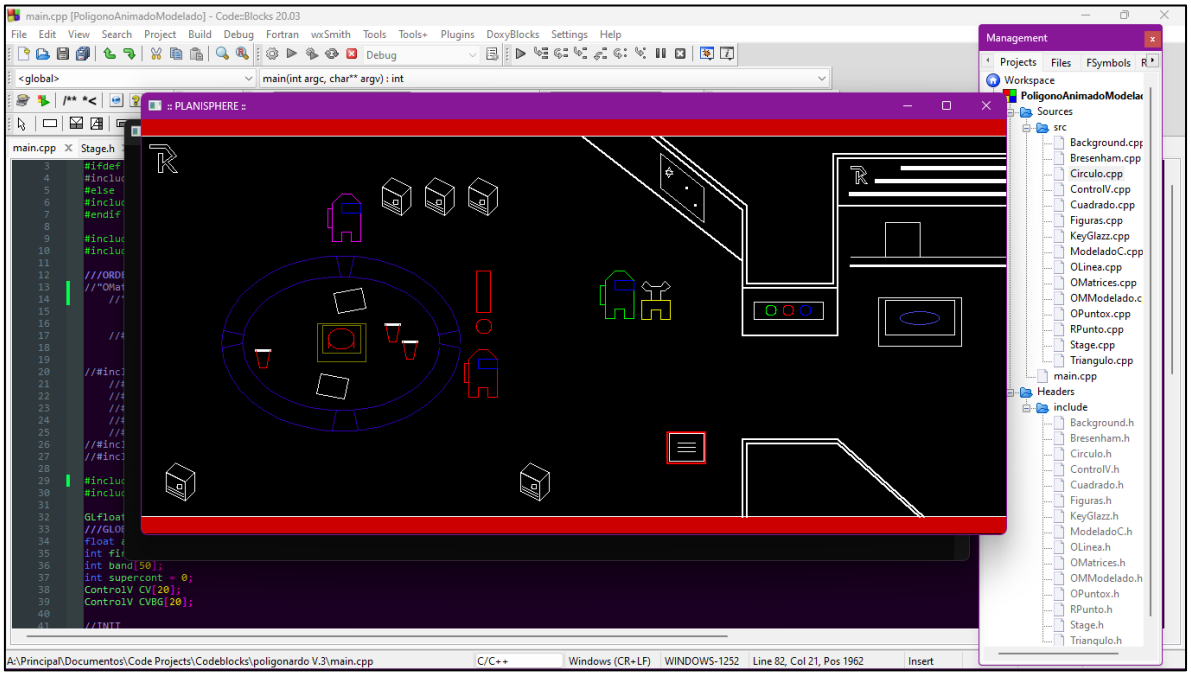




Jesús Huerta Aguilar

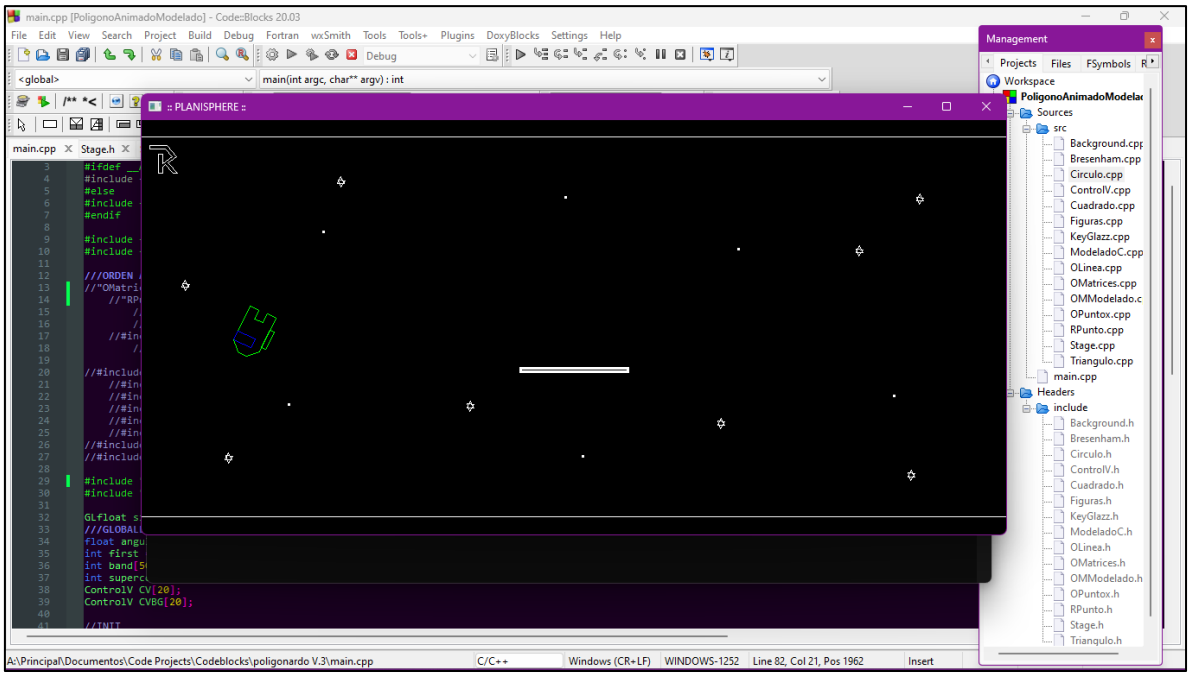
BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

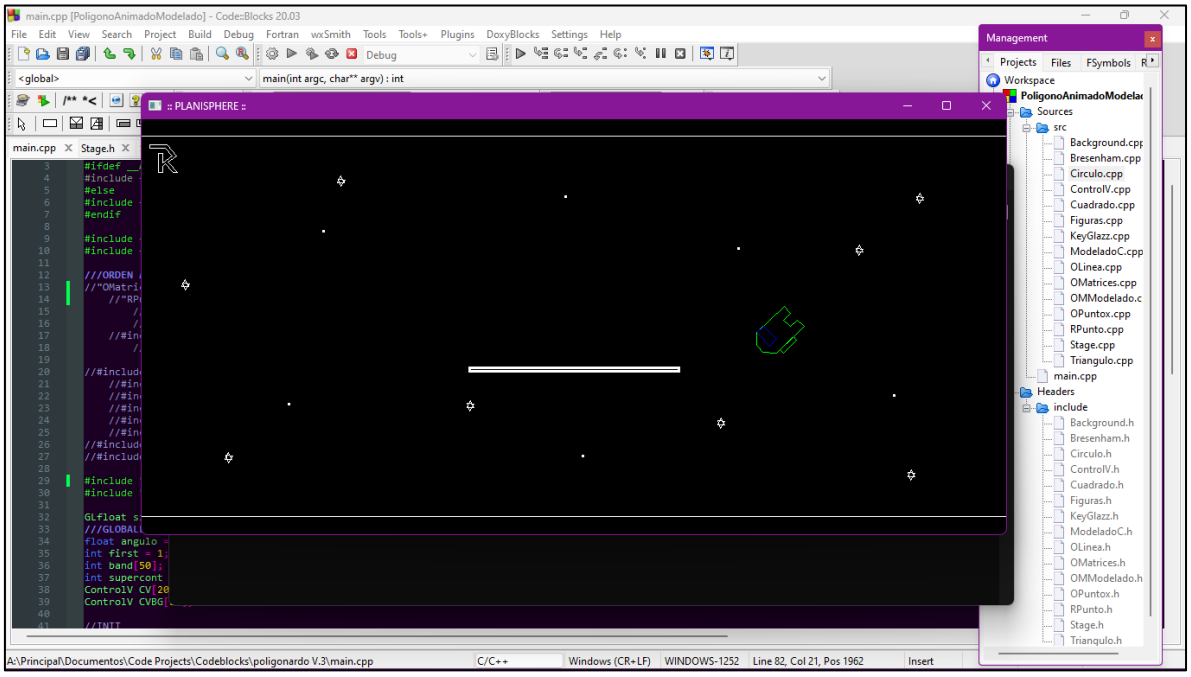




Jesús Huerta Aguilar

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN





Jesús Huerta Aguilar

CONCLUSIONES

Durante el desarrollo del proyecto, pude comprender muchos aspectos tanto en la programación en C++ como programar con la librería de OpenGL, uno de los aspectos que pude comprender muy bien fue el uso de clases ya que en este proyecto fue de vital importancia su uso para poder entender mejor el código además de reducir muchas líneas de código, otro de los aspectos importantes aprendidos fue el desarrollo e implementación de matrices dentro de C++ además de su uso para las operaciones de traslación, rotación, escalamiento y traslación inversa. Por parte de la librería de OpenGL, pude aprender muchas funciones y trucos para su manejo adecuado, además de entender el uso de Bresenham como algoritmo para unificar dos puntos pivote con mas puntos entre estos, muchos de estos aspectos aprendidos se han aplicado en otras materias, además de seguir los consejos del profesor sobre la implementación del proyecto de forma modular para poder así reusar el código sin problemas.

BIBLIOGRAFIA

- undefined [Devs' Den]. (2017, 28 marzo). 24. Vectores STL (std::vector) - C++ de 0 a Experto! [Vídeo]. YouTube. Recuperado 23 de septiembre de 2022, de https://www.youtube.com/watch?v=-_PksoQxRPA
- Lenka, C. (2020, 14 febrero). Vector of Vectors in C++ STL with Examples. geeksforgeeks. Recuperado 23 de septiembre de 2022, de https://www.geeksforgeeks.org/vector-of-vectors-in-c-stl-with-examples/
- Escuela de Ingenierias Industriales. (2020). Uso del contenedor vector con funciones. Recuperado 23 de septiembre de 2022, de https://www2.eii.uva.es/fund_inf/cpp/temas/9_vectores/vector_con_funciones.html
- Calling a void function on a c++ class. (2018, 3 noviembre). Stack Overflow. Recuperado 23 de septiembre de 2022, de https://stackoverflow.com/questions/53127820/calling-a-void-function-on-a-c-class
- Orellana, E. (s. f.). Introducción al manejo de gráficos utilizando OpenGL y Qt. crduran. Recuperado 27 de septiembre de 2022, de http://crduran.ubb.cl/old-html-based-site/notes/qt/opengl-qt.pdf