Universidad Nacional Autónoma de México

Facultad de Ingeniería

Asignatura: **Laboratorio de Computación Gráfica e Interacción de Humano-Computadora**

Profesor: **Ing. Luis Sergio Valencia**

Proyecto Final: **Zona residencial**

Integrantes:

García Padrón César Alejandro

Suárez Espinoza Mario Alberto

Grupo: **03**

Semestre: **2020-2**

11/Mayo/2020

Índice

Manual de usuario

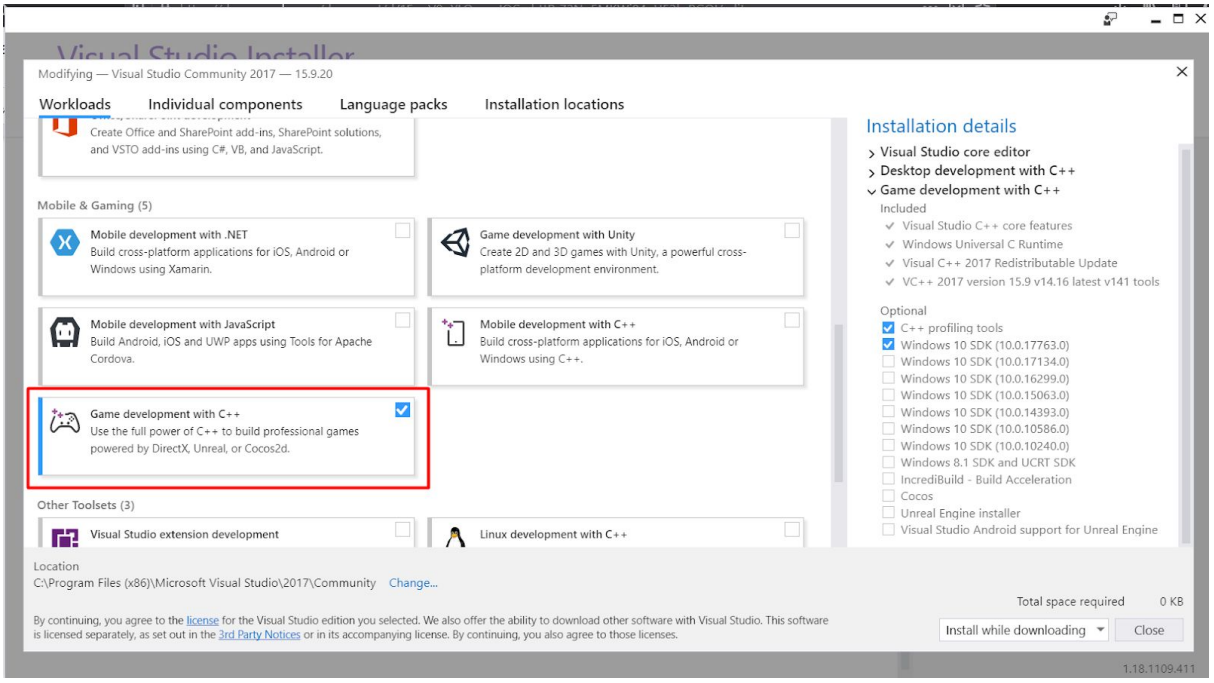
1. Preparación del ambiente

1.1 Requisitos previos

Como requisito indispensable, se requiere que la máquina del usuario esté corriendo sistema operativo Windows y que tenga instalado Visual Studio 2017 con alguna paquetería de C++. En caso de cumplir con este requisito, saltar al paso 1.2.

En caso de no tenerlo, puede obtener Visual Studio 2017 de la siguiente ruta: http://luissergiov.sodvi.com/practicas/vs_community__2036914200.1533234269.exe

Se descarga un ejecutable. Iniciarlo e instalar alguna paquetería de C++. Se recomienda elegir “Desarrollo de videojuegos con C++”

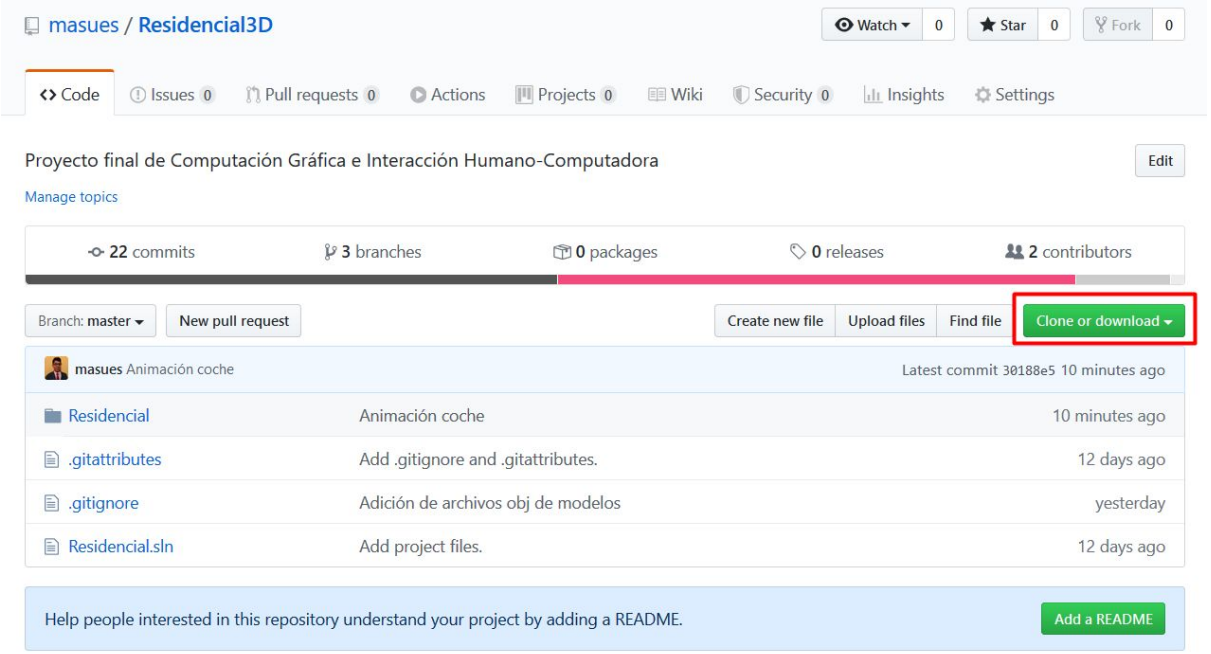


Una vez terminada la instalación de Visual Studio y de la paquetería, está listo para continuar con el paso 1.2

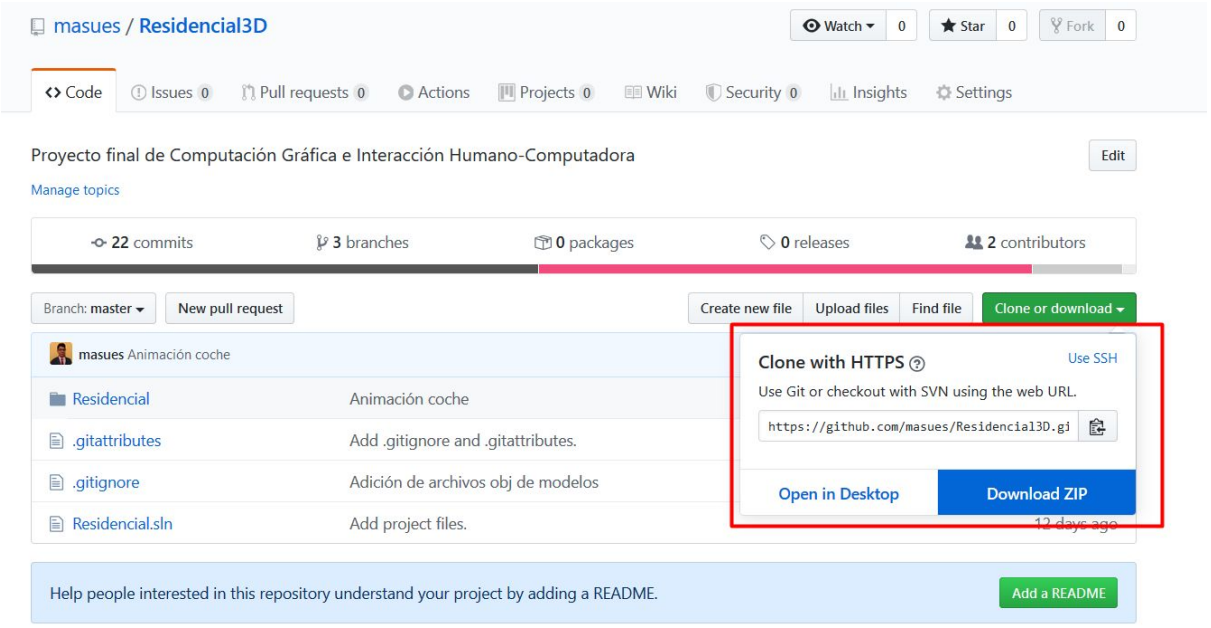
1.2 Descargar el comprimido del proyecto

En primera instancia se debe descargar el archivo comprimido del proyecto, para ello ingresar a la siguiente url: https://github.com/masues/Residencial3D/

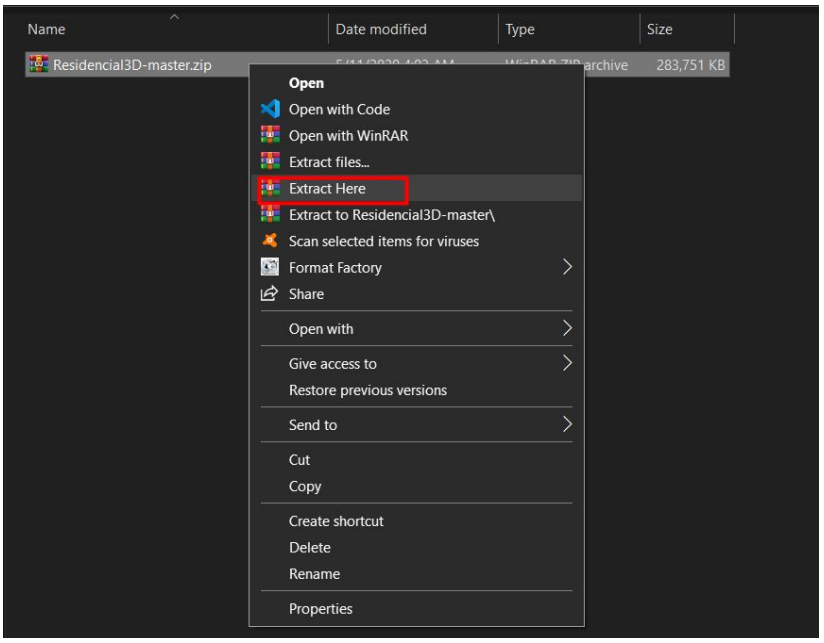
A continuación dar clic en “Clone or download”



Y a continuación seleccionar “Download ZIP”



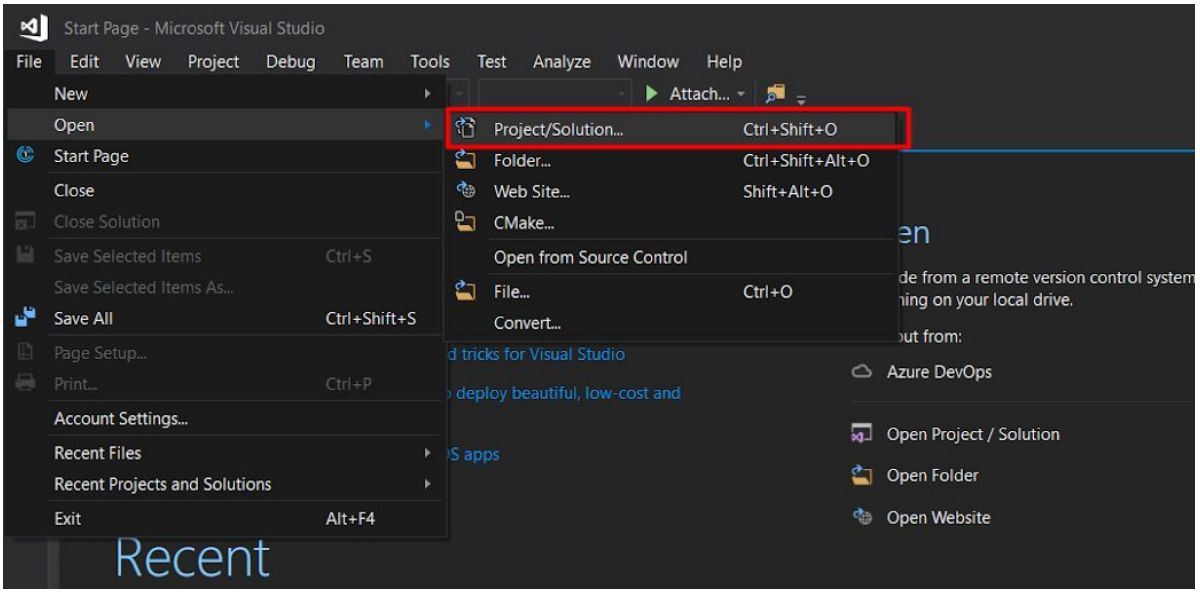
Posteriormente dirigirse a la la ubicación en donde se haya descargado, y descomprimirlo en la dirección deseada.

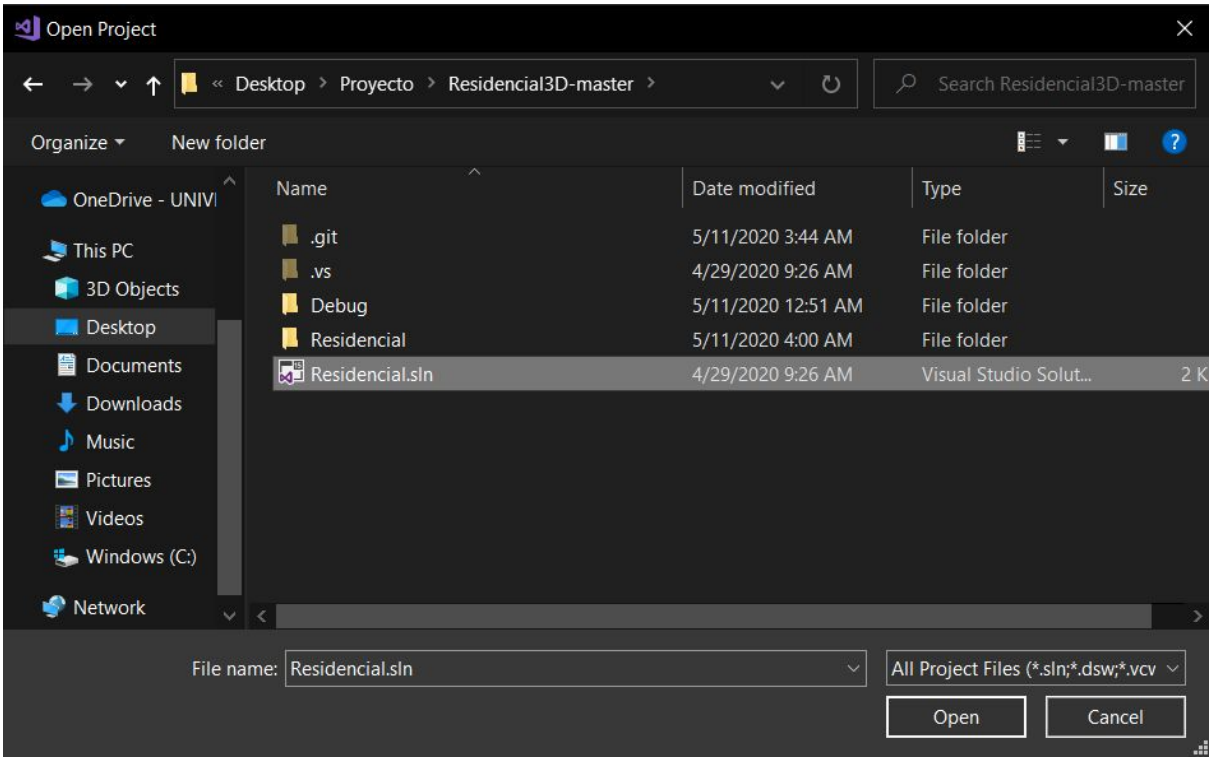


1.2 Configuraciones del proyecto

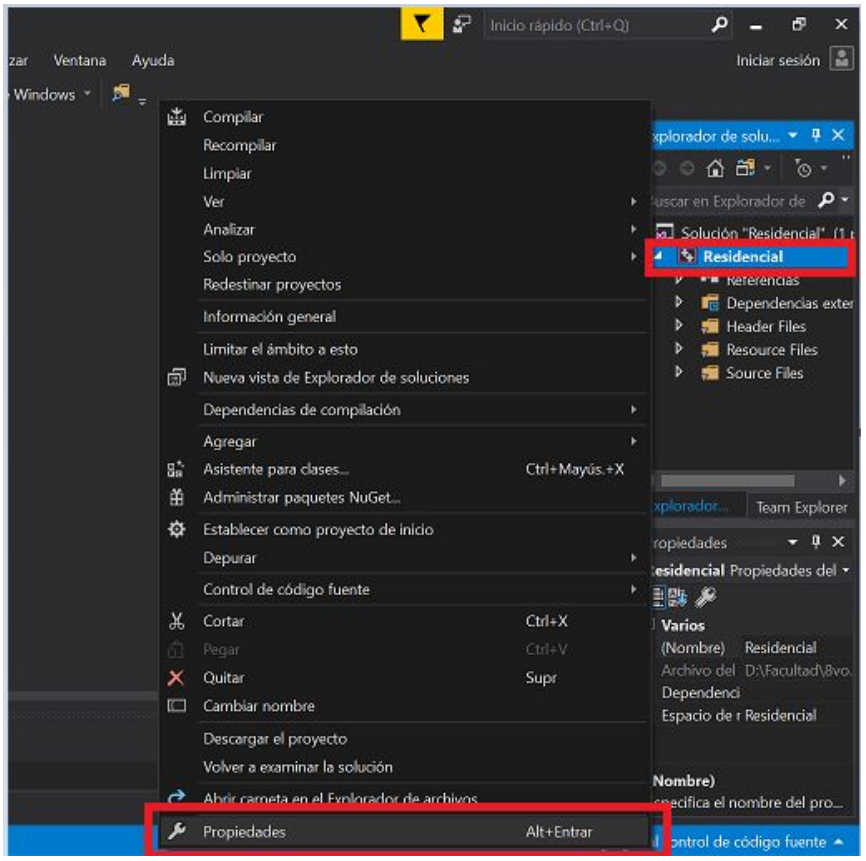
El proyecto requiere de la inclusión y configuración de ciertas bibliotecas, que Visual Studio no las tiene por default, por lo que es necesario, realizar una serie de pasos previos a la ejecución del archivo.

1. Abrir el programa Visual Studio
2. Dirigirse a *Archivo* > *Abrir* > *Proyecto o solución* dirigirse a la ubicación del archivo descomprimido y seleccionar la solución del proyecto (Archivo llamado Residencial.sln).



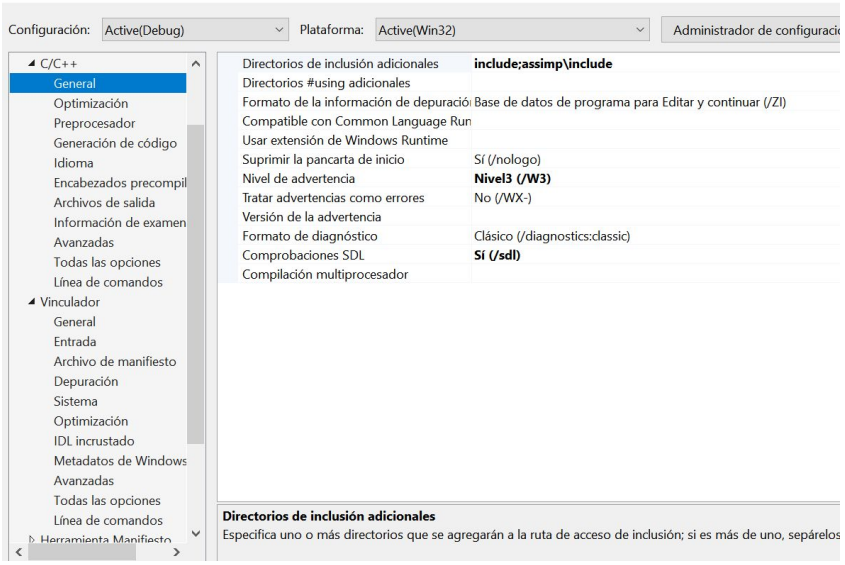


3. Ubicar dentro de Visual Studio el árbol de de archivos, dar click derecho sobre el nombre del proyecto y dirigirse a las propiedades.

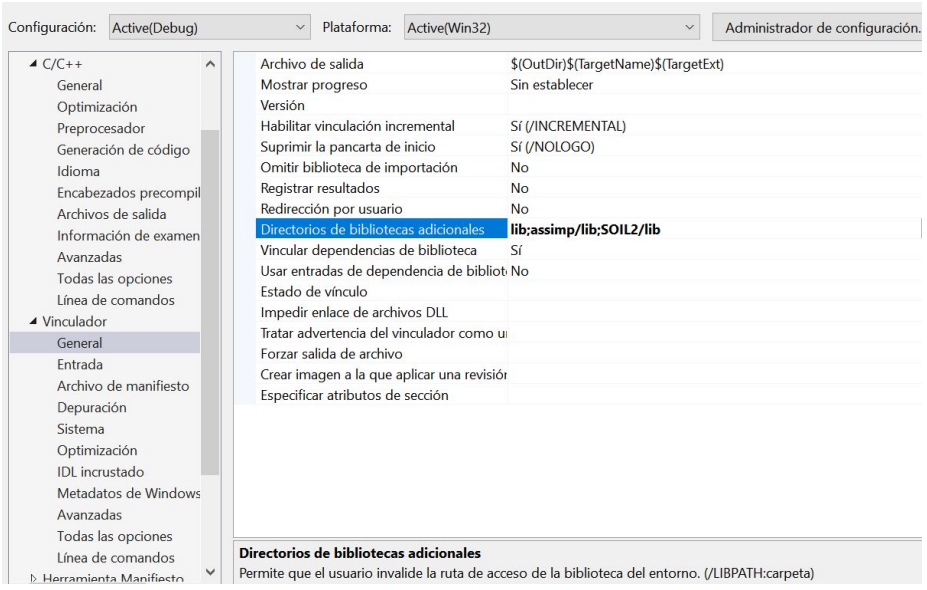


4. Seleccionar la pestaña *C/C++* > *General*

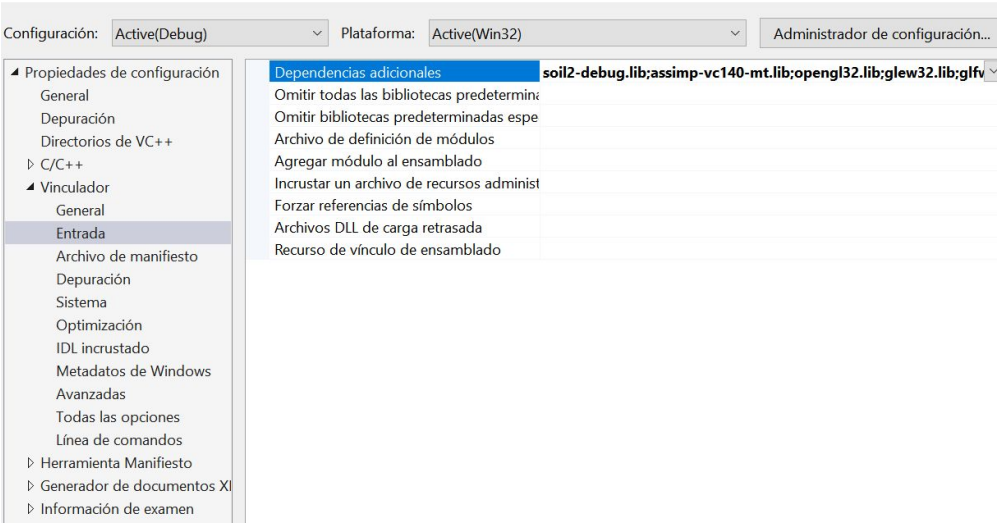
a. Escribir en el campo *Directorios de inclusión adicionales* lo siguiente: **include;assimp\include**



5. Seleccionar la pestaña *Vinculador > General*
   a. Escribir en el campo *Directorios de bibliotecas adicionales* lo siguiente: **lib;assimp/lib;SOIL2/lib**

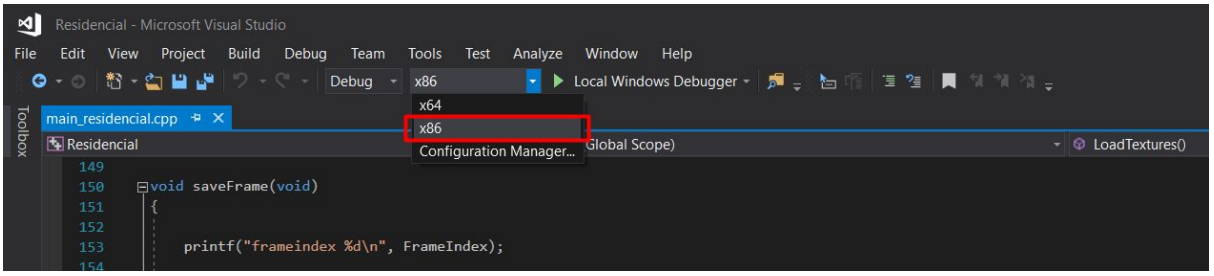


6. Seleccionar la pestaña *Vinculador > Entrada*
   a. Escribir en el campo *Dependencias adicionales* lo siguiente: **soil2-debug.lib;assimp-vc140-mt.lib;opengl32.lib;glew32.lib;glfw3.lib;**

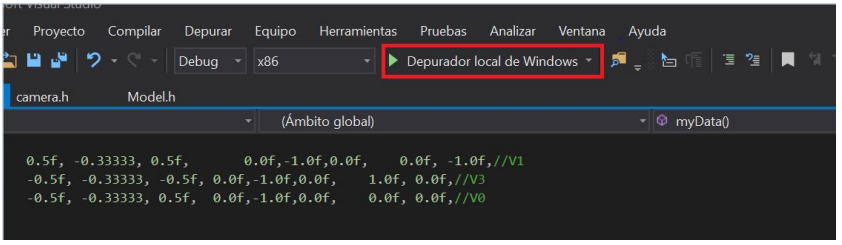


#Nota: No borrar lo que ya estaba escrito

7. Asegurarse que se tiene seleccionada la opción Debug con “x86”



8. Finalmente presionar aceptar, la ventana de las propiedades del proyecto se cerrará y se podrá ejecutar presionando en el botón *Depurador local de Windows.*



2. Información de animaciones

Animación 1: Leia

La animación 1 consiste en Lego Leia (personaje de StarWars) realizando movimientos. Primero gira la cabeza, luego corre hacia adelante para dar un salto, y caer en split. Esta animación fue realizada por Keyframes y posee 6 keyframes precargados. En los primeros 2 keyframes mueve la cabeza, en los siguientes 2 keyframes se mueve hacia adelante y mueve sus dos pies (simulando el correr), en el siguiente keyframe da un salto, levanta el brazo izquierdo y baja el derecho, para en el último caer en diagonal, hacer un split con las piernas, girar su torso, su cuello, levantar el brazo derecho y bajar el izquierdo.

Para iniciar esta animación, se necesita presionar la tecla 'P'. Si se quiere volver a iniciar, en cualquier instante es posible volver a presionarla.

Código

Inicialización de Keyframes

```
//Keyframe 0
KeyFrame[0].posX = 0;
KeyFrame[0].posY = 0;
KeyFrame[0].posZ = -26;
KeyFrame[0].rotRodIzq = 0;
KeyFrame[0].giroMonito = 0;
KeyFrame[0].movBrazo = 0;
KeyFrame[0].rotCabeza = 40;
//Keyframe 1
KeyFrame[1].posX = 0;
KeyFrame[1].posY = 0;
KeyFrame[1].posZ = -26;
KeyFrame[1].rotRodIzq = 0;
KeyFrame[1].giroMonito = 0;
KeyFrame[1].movBrazo = 0;
KeyFrame[1].rotCabeza = -40;
//Keyframe 2
KeyFrame[2].posX = 0;
KeyFrame[2].posY = 0;
KeyFrame[2].posZ = -6;
```

```
  KeyFrame[2].rotRodIzq = 40;
  KeyFrame[2].giroMonito = 0;
  KeyFrame[2].movBrazo = 0;
  KeyFrame[2].rotCabeza = 0;
  //Keyframe 3
  KeyFrame[3].posX = 0;
  KeyFrame[3].posY = 0;
  KeyFrame[3].posZ = 16;
  KeyFrame[3].rotRodIzq = -40;
  KeyFrame[3].giroMonito = 0;
  KeyFrame[3].movBrazo = 0;
  KeyFrame[3].rotCabeza = 0;
  //Keyframe 4
  KeyFrame[4].posX = 0;
  KeyFrame[4].posY = 9;
  KeyFrame[4].posZ = 36;
  KeyFrame[4].rotRodIzq = 0;
  KeyFrame[4].giroMonito = 0;
  KeyFrame[4].movBrazo = -60;
  KeyFrame[4].rotCabeza = 0;
  //Keyframe 5
  KeyFrame[5].posX = 10;
  KeyFrame[5].posY = -6;
  KeyFrame[5].posZ = 56;
  KeyFrame[5].rotRodIzq = 80;
  KeyFrame[5].giroMonito = 40;
  KeyFrame[5].movBrazo = 60;
  KeyFrame[5].rotCabeza = 40;
```

Manipulación de Keyframes

```
float posX = 0.0f, //variables de manipulación del dibujo
posY = 0.0f,
posZ = 0.0f,
rotRodIzq = 0.0f, //Pie
giroMonito = 0.0f,
movBrazo = 0.0f,
rotCabeza = 0.0f;

float incX = 0.0f, //Variables para el cálculo de incrementos
```

```
incY = 0.0f,
incZ = 0.0f,
rotInc = 0.0f,
giroMonitoInc = 0.0f,
movBrazoInc = 0.0f,
rotCabezaInc = 0.0f;

#define MAX_FRAMES 9
int i_max_steps = 20; //Cantidad máxima de frames intermedios
int i_curr_steps = 0;
typedef struct _frame
{
  //Variables para GUARDAR Key Frames
  float posX;   //Variable para PosicionX
  float posY;   //Variable para PosicionY
  float posZ;   //Variable para PosicionZ
  float rotRodIzq;
  float giroMonito;
  float movBrazo;
  float rotCabeza;

}FRAME;

FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 6;     //introducir datos
bool play = false;
int playIndex = 0;

void saveFrame(void)
{

  printf("frameindex %d\n", FrameIndex);

  KeyFrame[FrameIndex].posX = posX;
  KeyFrame[FrameIndex].posY = posY;
  KeyFrame[FrameIndex].posZ = posZ;
  printf("posX=%f, posY=%f, posZ=%f\n", posX, posY, posZ);

  KeyFrame[FrameIndex].rotRodIzq = rotRodIzq;
  KeyFrame[FrameIndex].giroMonito = giroMonito;
```

```
  KeyFrame[FrameIndex].movBrazo = movBrazo;
  KeyFrame[FrameIndex].rotCabeza = rotCabeza;
  printf("rotRodIzq=%f, giroMonito=%f\nmovBrazo=%f, rotCabeza=%f\n",
rotRodIzq, giroMonito, movBrazo, rotCabeza);

  FrameIndex++;
}


//Sustituye lo que tiene actualmente el personaje por
//lo guardado en el cuadro clave cero
void resetElements(void)
{
  posX = KeyFrame[0].posX;
  posY = KeyFrame[0].posY;
  posZ = KeyFrame[0].posZ;

  rotRodIzq = KeyFrame[0].rotRodIzq;
  giroMonito = KeyFrame[0].giroMonito;
  movBrazo = KeyFrame[0].movBrazo;
  rotCabeza = KeyFrame[0].rotCabeza;

}

void interpolation(void)
{

  incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) /
i_max_steps;
  incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) /
i_max_steps;
  incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) /
i_max_steps;

  rotInc = (KeyFrame[playIndex + 1].rotRodIzq - KeyFrame[playIndex].rotRodIzq)
/ i_max_steps;
  giroMonitoInc = (KeyFrame[playIndex + 1].giroMonito -
KeyFrame[playIndex].giroMonito) / i_max_steps;
  movBrazoInc = (KeyFrame[playIndex + 1].movBrazo -
KeyFrame[playIndex].movBrazo) / i_max_steps;
```

```
  rotCabezaInc = (KeyFrame[playIndex + 1].rotCabeza -
KeyFrame[playIndex].rotCabeza) / i_max_steps;
}
```

Dibujo de Lego Leia

```
//Lego Leia
  model = tmp;
  model = glm::scale(model, glm::vec3(0.046187f, 0.046187f, 0.046187f));

  //Cuerpo Leia
  model = glm::translate(model, glm::vec3(0.51f, 10.491f, -0.782f));
  model = glm::translate(model, glm::vec3(posX, posY, posZ));
   tmp = model = glm::rotate(model, glm::radians(giroMonito), glm::vec3(0.0f,
1.0f, 0.0));
  shader.setMat4("model", model);
  cuerpoLeia.Draw(shader);

  //Pierna Der Leia
  model = glm::translate(tmp, glm::vec3(-3.539f, 0.035f, 0.027f));
  model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0));
   model = glm::rotate(model, glm::radians(-rotRodIzq), glm::vec3(1.0f, 0.0f,
0.0f));
  shader.setMat4("model", model);
  piernaDerLeia.Draw(shader);

  //Pierna Izq Leia
  model = glm::translate(tmp, glm::vec3(3.642f, 0.06f, 0.001f));
  model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
   model = glm::rotate(model, glm::radians(rotRodIzq), glm::vec3(1.0f, 0.0f,
0.0f));
  shader.setMat4("model", model);
  piernaIzqLeia.Draw(shader);

  //Brazo derecho Leia
  model = glm::translate(tmp, glm::vec3(-5.667f, 12.835f, 0.197f));
   model = glm::rotate(model, glm::radians(-movBrazo), glm::vec3(1.0f, 0.0f,
0.0f));
  shader.setMat4("model", model);
  brazoDerLeia.Draw(shader);
```

```
  //Brazo izquierdo Leia
  model = glm::translate(tmp, glm::vec3(5.598f, 12.291f, -0.498f));
   model  =  glm::rotate(model,  glm::radians(movBrazo),  glm::vec3(1.0f,  0.0f,
0.0f));
  shader.setMat4("model", model);
  brazoIzqLeia.Draw(shader);

  //Cabeza Leia
  model = glm::translate(tmp, glm::vec3(0.237f, 16.214f, 0.016f));
   model  =  glm::rotate(model,  glm::radians(rotCabeza),  glm::vec3(0.0f,  1.0f,
0.0));
  shader.setMat4("model", model);
  cabezaLeia.Draw(shader);
```

Animación 2: Frisbee

En la animación 2 tenemos el movimiento de un frisbee siguiendo una trayectoria parabólica que a su vez, respeta lo dictado por la ecuación (1).

$$y = -0.05z^2 + 4 \qquad Ec.\ (1)$$

Para activar esta animación es necesario presionar la tecla 'E', posterior a ello el frisbee comenzará un recorrido del punto A al punto B y viceversa de manera infinita. Si se vuelve a presionar la tecla 'E', el frisbee se pausa por un instante, y para activarlo se tiene que volver a presionar la tecla 'E'.

Código

Variables para la animación del frisbee

```
bool animacionFrisbee = false;
bool regresoFrisbee = false;
float movFrisbee_z = 8.7f;
float movFrisbee_y = 0.2f;
```

Animación del frisbee

```
//Animación Frisbee
  if (animacionFrisbee)
  { //Función y=-0.05z^2 +4
    if (!regresoFrisbee)
    {
      if (movFrisbee_z >= -8.0f)
      {
        movFrisbee_z -= 1.0;
        movFrisbee_y = (-0.05 * (pow(movFrisbee_z, 2))) + 4;
      }
      else
        regresoFrisbee = true;
    }
    else
    {
      if (movFrisbee_z <= 8.5f)
      {
        movFrisbee_z += 1.0;
```

```
        movFrisbee_y = (-0.05 * (pow(movFrisbee_z, 2))) + 4;
      }
      else
        regresoFrisbee = false;
    }
```

Dibujo del frisbee

```
//frisbee
  model = glm::translate(sectorC, glm::vec3(-4.0f, movFrisbee_y,
movFrisbee_z)); //-7.0f
  model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
  model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f,
0.0));
  shader.setMat4("model", model);
  frisbee.Draw(shader); //
```

Animación 3: Coche

La animación 3 consiste en un auto que realiza un recorrido en un cuarto de circunferencia, para salir de un estacionamiento y a continuación recorre la calle en línea recta.

La animación consta de dos estados. Descritos de la siguiente forma:

Estado 1: Recorre media circunferencia hacia atrás. Este recorrido utiliza la ecuación paramétrica de la circunferencia, la cual es

$$x = centro(x) + r * cos(\theta)$$

$$z = centro(z) + r * sin(\theta)$$

Dónde

x,z son la posición del auto en cada instante de tiempo

centro(x),centro(z) son la posición del centro de la circunferencia

r es el radio de la circunferencia

θ es el ángulo del recorrido. En este caso un parámetro el cual incrementa cada instante de tiempo para poder generar el recorrido.

Estado 2: Este es un recorrido simple en línea recta. Una vez que el auto ingresa a la calle, este inicia su recorrido hacia adelante y se detiene después de recorrer hacia adelante 45 unidades (en términos del escenario) en total.

Para iniciar esta animación es necesario presionar la tecla ‘espacio’. Para reiniciarla, se puede volver a presionar esta tecla.

Código

Variables para animación del carro

```
//Animación Carro
float movAuto_z,
    movAuto_x,
    orienta = -90.0f;
int recorridoCoche = 1;
bool animacion = false;
float r = 15;
float centroX = -21;
float centroZ = -25;
float thetaRot = 90;
#define PI 3.14159265
```

Animación coche

```
//Animación Coche
if (animacion)
{
  switch (recorridoCoche)
  {
  case 1:
    movAuto_x = centroX + r * cos(thetaRot * PI / 180);
    movAuto_z = centroZ + r * sin(thetaRot * PI / 180);
    orienta = -thetaRot;
    thetaRot -= 1.2;
    if (thetaRot < 0) //Dos vueltas
      recorridoCoche = 2;
    break;
  case 2:
    movAuto_z += 0.8f;
    orienta = 0.0f;
    if (movAuto_z > 20.0f)
    {
      recorridoCoche = 1;
      animacion = false;
    }
    break;
  }
}
```

Dibujo del carro

```
//Carro
  model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.0f));
  model = glm::translate(model, glm::vec3(movAuto_x, 0.0f, movAuto_z));
   tmp  =  model  =  glm::rotate(model,  glm::radians(orienta),  glm::vec3(0.0f,
1.0f, 0.0f));
  model = glm::scale(model, glm::vec3(0.010208f, 0.010208f, 0.010208f));
  shader.setMat4("model", model);
  carro.Draw(shader);

  model = glm::translate(tmp, glm::vec3(0.85f, 0.25f, 1.29f));
  model = glm::scale(model, glm::vec3(0.010208f, 0.010208f, 0.010208f));
```

```
  shader.setMat4("model", model);
  llantas.Draw(shader); //Izq delantera

  model = glm::translate(tmp, glm::vec3(-0.85f, 0.25f, 1.29f));
  model = glm::scale(model, glm::vec3(0.010208f, 0.010208f, 0.010208f));
    model  =  glm::rotate(model,  glm::radians(180.0f),  glm::vec3(0.0f,  1.0f,
0.0f));
  shader.setMat4("model", model);
  llantas.Draw(shader); //Der delantera

  model = glm::translate(tmp, glm::vec3(-0.85f, 0.25f, -1.45f));
  model = glm::scale(model, glm::vec3(0.010208f, 0.010208f, 0.010208f));
    model  =  glm::rotate(model,  glm::radians(180.0f),  glm::vec3(0.0f,  1.0f,
0.0f));
  shader.setMat4("model", model);
  llantas.Draw(shader); //Der trasera

  model = glm::translate(tmp, glm::vec3(0.85f, 0.25f, -1.45f));
  model = glm::scale(model, glm::vec3(0.010208f, 0.010208f, 0.010208f));
  shader.setMat4("model", model);
  llantas.Draw(shader); //Izq trase
```

Animación 4: Balón

La animación 4 consiste en recrear la caída libre de un balón de basquetbol, simulando que cae de la asote de un edificio de 10 metros. Para simplificar el movimiento no se consideró la rotación sobre el mismo eje. El balón cae y rebota hasta que pierda su energía.

Para realizar la animación se utilizaron leyes físicas, y una máquina de estados para poder programarla.

Caso 0: Una vez que se le dé la señal comienza la caída del balón siguiendo la siguiente ecuación:

$$y = h - (0.5)(g)(t)^2$$

Caso 1: Una vez que llegaba al suelo cambiamos la trayectoria del balón, hacia el lado opuesto, avanzaba hasta llegar al 60% de la altura previa siguiendo la siguiente ecuación:

$y = (g)(t)^2$

Donde :

$y =$ *altura actual*

$h =$ *altura máxima*

$g =$ *Fuerza de gravedad* $(9.81\ m/s^2)$

$t =$ *Tiempo. Para esta animación se hacían intervalos de* $0.5s$ *para su óptima visualización*

Caso 2: Caso auxiliar para terminar el rebote del balón. Es decir, se van iterando los casos 0 y 1. Cuando la altura sea menor a 20 cm. se detiene el proceso pasando al caso 2.

Código:

Variables

```
bool animacionBalon = false;
bool regresoBalon = false;
float alturaFija = 10.0f;
float movBslon_x = -11.0f;
float movBalon_z = 0.0f;
float movBalon_y = 10.0f;
float contadorTiempo = 0.0f;
float altura = alturaFija;
float velocidad = 1.0f;
int estadosBalon = 0;
```

Animación

```
float aux;
if (animacionBalon) {
	switch (estadosBalon) {
	case 0://Hacia abajo
		if (movBalon_y > 0.0f) {
			contadorTiempo += 0.5;
			aux = (altura - (0.5*9.81*pow(contadorTiempo, 2)));
			if (aux < 0.0)
				movBalon_y = 0.0;
			else
				movBalon_y = aux;
		}
		else {
			velocidad = (9.81*contadorTiempo) / 0.5;//
			altura = altura * 0.6;
			contadorTiempo = 0.0f;
			if (altura < 0.2)
				estadosBalon = 2;
			else
				estadosBalon = 1;
```

```
		}
		break;
	case 1://Hacia arriba
		if (movBalon_y < altura) {
			contadorTiempo += 0.5;
			aux = 9.81*pow(contadorTiempo, 2);
			if (aux > altura)
				movBalon_y = altura;
			else
				movBalon_y += 0.5;
			//movBalon_y = velocidad * contadorTiempo -
(0.5*9.81*pow(contadorTiempo, 2));
		}
		else {
			contadorTiempo = 0.0f;
			estadosBalon = 0;
		}

	case 2:
		movBalon_y = movBalon_y;
		break;
	default:
		break;
	}
```

Dibujo

```
model = glm::translate(glm::mat4(1.0f), glm::vec3(movBslon_x, movBalon_y, movBalon_z));
	model = glm::scale(model, glm::vec3(0.03f, 0.03f, 0.03));
	shader.setMat4("model", model);
	balon.Draw(shader);
```

Animación 5: Taza

Esta animación tarta de reconstruir una curiosidad en donde se unen los universos de Gravity Falls y Rick and Morty. Esto al hacer viajar una taza de un portal a otro en dentro del escenario.

Para lograr realizarla fue necesario utilizar una máquina de estados. Al dar la seña el objeto se trasladaba en dirección z negativo, hacia un portal, una vez que se se cubría todo el objeto se trasladaba a una ubicación donde se encontraba el segundo portal, posteriormente se trasladaba sobre el eje z positivo, dando un efecto que que la taza era absorbida por los portales.

El recorrido mencionado se repetía.

Código

Variables

```
bool animacionTaza = false;
bool regresoTaza = false;
float movTaza_z = 2.0f;
float movTaza_x = 0.0f;
float movTaza_y = 2.5f;
int casosTaza = 0;
```

Animación

```
if (animacionTaza) {
        switch (casosTaza) {
        case 0:
                if (movTaza_z >= 0.2)
                        movTaza_z -= 0.5;
                else
                        casosTaza = 1;
                break;
        case 1:
                movTaza_x = 42.44f;
                movTaza_y = 1.0f;
                movTaza_z = 5.2f;
                casosTaza = 2;
                break;
        case 2:
                if (movTaza_z <= 7.0)
                        movTaza_z += 0.5;
                else
                        casosTaza = 3;
                break;
        case 3:
                if (movTaza_z >= 5.2)
                        movTaza_z -= 0.5;
                else
                        casosTaza = 4;
                break;
        case 4:
                movTaza_x = 0.0f;
                movTaza_y = 2.5f;
                movTaza_z = 0.2;
                casosTaza = 5;
                break;
        case 5:
                if (movTaza_z <= 2.0)
                        movTaza_z += 0.5;
                else
                        casosTaza = 0;
                break;
```

```
            default:
                break;
            }
    }
```

Dibujo

```
//Taza
model = glm::translate(cenPortal, glm::vec3(movTaza_x, movTaza_y, movTaza_z));
model = glm::scale(model, glm::vec3(0.15f, 0.15f, 0.15f));
model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0));
shader.setMat4("model", model);
taza.Draw(shader);  //
```

3. Cronograma de actividades

3.1 Herramientas

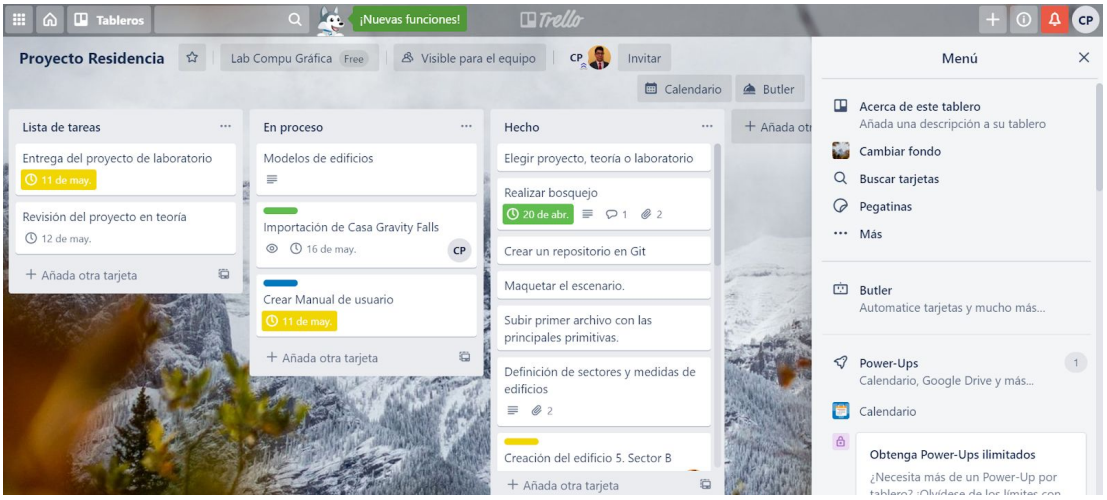
Realizar este proyecto fue un proceso largo y no fácil, se requirió de una buena organización para poder completarlo ordenadamente. Para ello se utilizó la herramienta, recomendada, Trello, un programa que ayuda a manejar proyectos entre equipos de trabajo.

Trello tiene la funcionalidad de crear tareas, las cuales se pueden editar para categorizarlas, etiquetarlas, asignarlas a ciertos miembros del equipo y ajustar una fecha y hora de vencimiento. Todas estas características nos ayudaron a no perder de vista los objetivos y tiempos.

En esta herramienta enlistamos una serie de actividades que eran necesarias para el proyecto, las cuales las dividimos en 3:

- Aquellas que solo estaban planeadas
- Actividades en proceso
- Actividades finalizadas

Además de Trello, utilizamos otros medios para realizar el proyecto, una carpeta en Google Drive, un repositorio en GitHub y la aplicación de WhatsApp. Cada aplicación tenía funciones especiales, pero en general nos ayudaban a mantener una buena comunicación.



3.2 Proceso

1. Decidir basar el proyecto en el propuesto por el Ing. Sergio Valencia, profesor del laboratorio, o el propuesto por el Ing. Roque, profesor de la parte teórica, al final optamos por el primero.
2. Buscar ejemplos de zonas residenciales, edificios, elementos que se podrían incluir, etc.
3. Realizar un bosquejo-croquis del escenario a plasmar, contemplando edificios, elementos extra y animaciones.

4. Acordar forma de trabajo entre los integrantes del equipo, incluyendo tiempos y actividades por realizar por cada miembro.
5. Acordar las bases del proyecto, tales como las medidas del escenario y de cada elemento que lo compone.
6. Crear un repositorio en GitHub con las elementos básicos del proyecto, es decir, bibliotecas, archivos de encabezados, primitivas, entre otros.
7. Búsqueda de modelos que conformarán el escenario, esencialmente, edificios.
8. Configuración de los modelos encontrados para hacer que éstos sean aptos a las necesidades.
9. Búsqueda de elementos extra, como un frisbee, fuente, etc.
10. Configuración de los modelos extra para adaptarlos al proyecto.
11. Crear las a animaciones solicitadas.
12. Escribir el manual de usuario.
13. Modificar las luces
14. Crear 2 animaciones que faltaban
15. Agregar elementos extras como portales.
16. Anexar los cambios en el manual de usuario.
17. Especificar de mejor manera las animaciones.

4. Funcionalidades del proyecto (Teclas)

El proyecto tiene diversas funcionalidades, que se manejan por diferentes teclas.

Control del escenario

- **Mouse:** Movimiento de la cámara.
- **A:** Desplazamiento en dirección izquierda del escenario. Dirección -x.
- **D:** Desplazamiento en dirección derecha del escenario. Dirección x.
- **W:** Desplazamiento hacia enfrente. Dirección z.
- **S:** Desplazamiento hacia atrás. Dirección -z.
- **repág (teclado latinoamericano), pageup (teclado americano):** Desplazamiento hacia arriba. Dirección y.
- **avpág (teclado latinoamericano), pagedown (teclado americano):** Desplazamiento hacia abajo. Dirección -y
- **izquierda:** Rotación del escenario completo sobre el eje Y en sentido negativo.
- **derecha:** Rotación del escenario completo sobre el eje Y en sentido positivo.

Animación 1: Leía

- **Y, H:** Desplazamiento hacia adelante y hacia atrás.
- **G, J:** Desplazamiento hacia la derecha y a la izquierda.
- **I, K:** Desplazamiento hacia arriba y hacia abajo.
- **N, M:** Giro de la cabeza hacia la derecha y a la izquierda.
- **X, C:** Movimiento de los pies hacia adelante y hacia atrás.
- **V, B:** Rotación del personaje completo hacia la derecha y a la izquierda sobre el eje Y.
- **R, F:** Movimiento de las manos hacia arriba y hacia abajo.
- **L:** Se guarda el keyframe.
- **P:** Inicio de la animación.

Animación 2: Frisbee

- **E:** Activar y desactivar el movimiento del frisbee.

Animación 3: Automóvil

- **espacio:** Activar e iniciar, o desactivar y volver al estado inicial del auto.

Animación 4: Balón

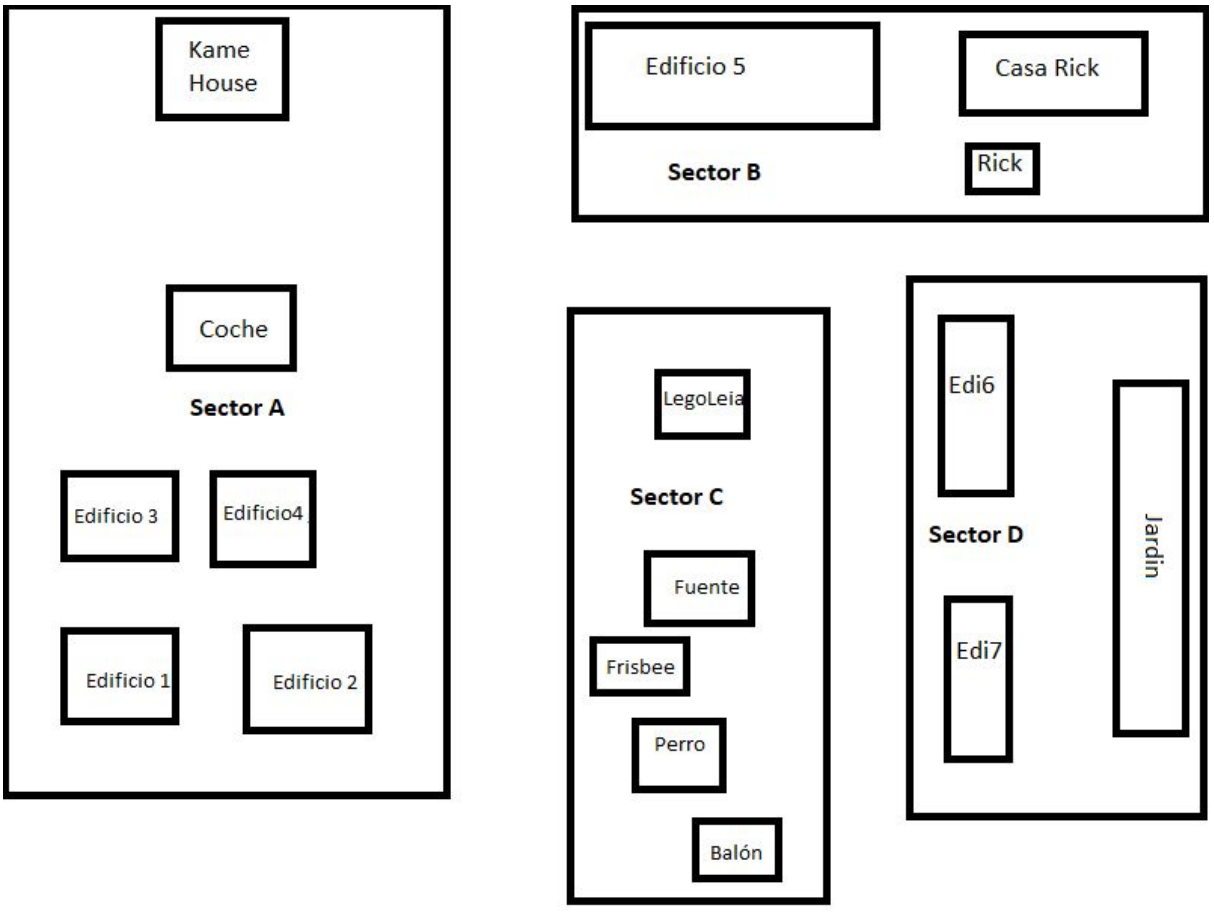
- **U:** Iniciar el proceso de caída libre

Animación 3: Automóvil

- **T:** Comenzar con el recorrido de la taza a través de los portales.

5. Información del escenario

Mapa del escenario



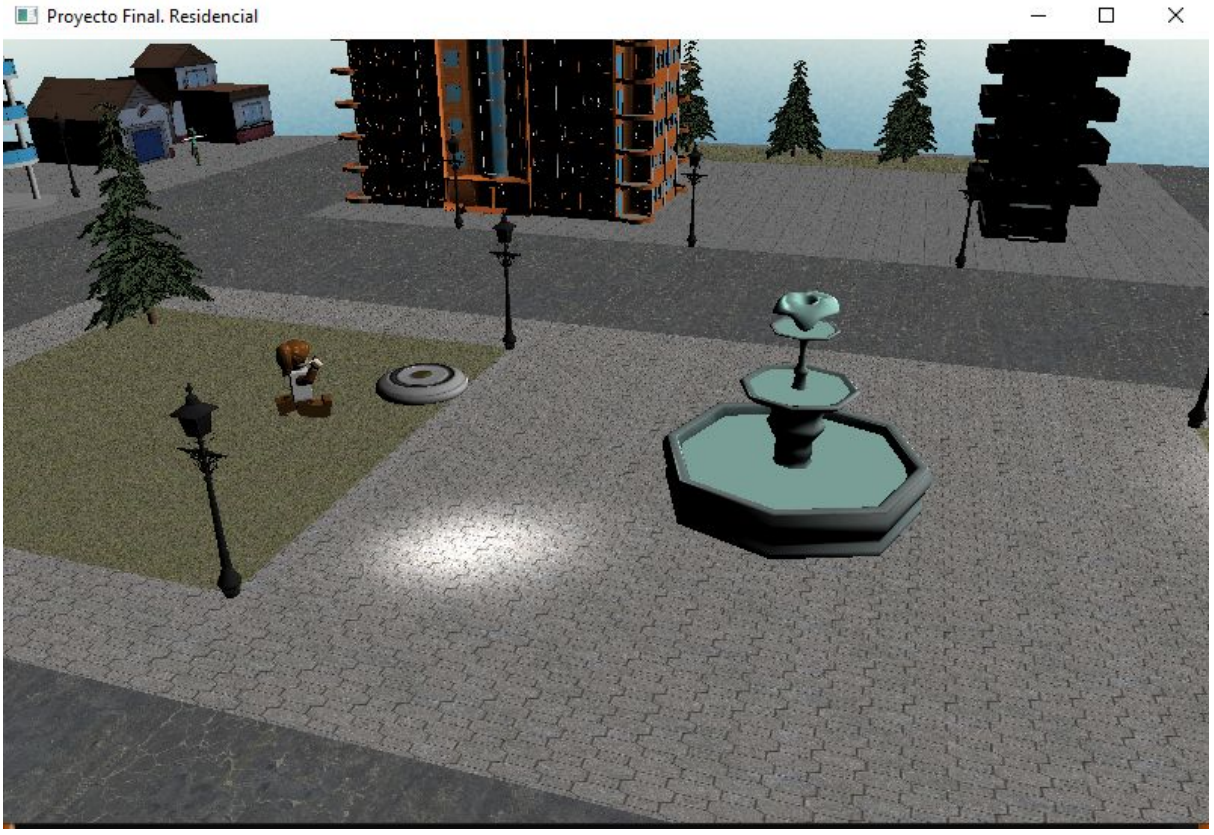
Fotos del escenario

Casa Rick Sánchez (Rick and Morty)

Sector D

Calle

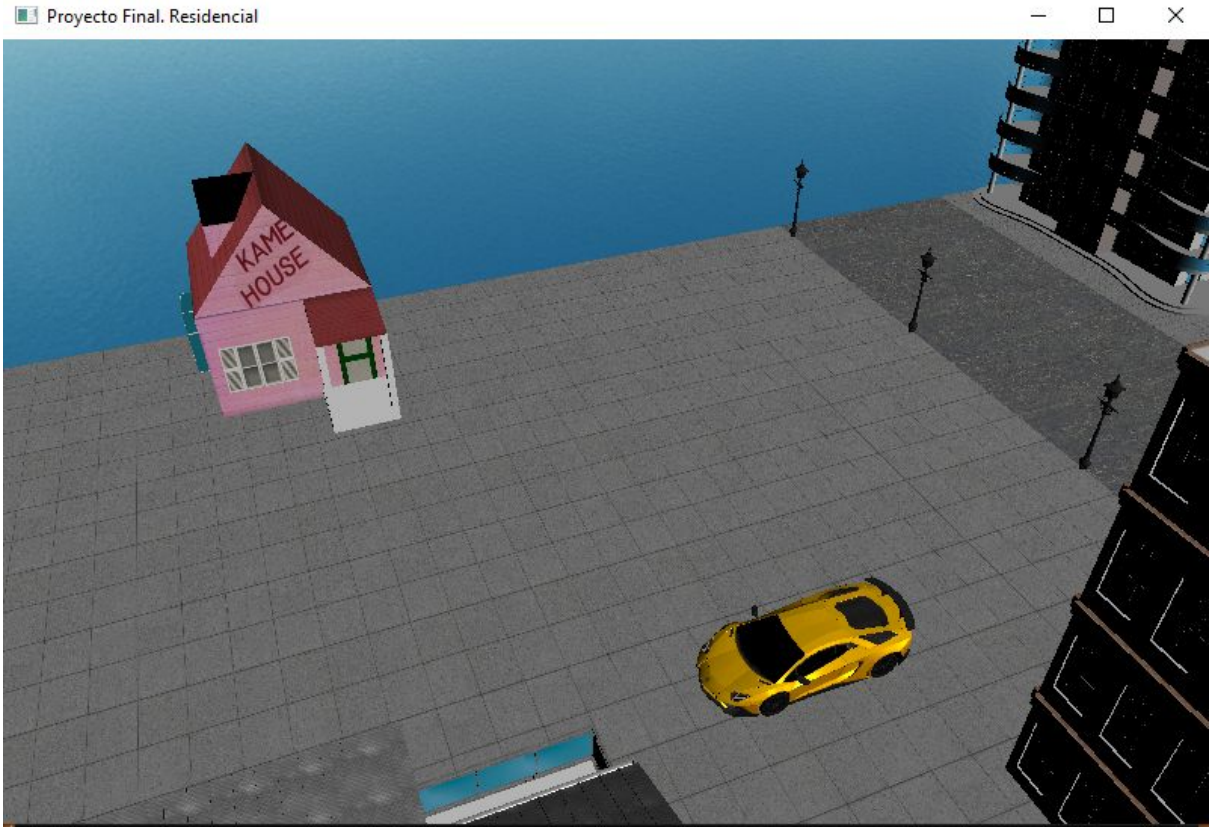
Movimiento Frisbee

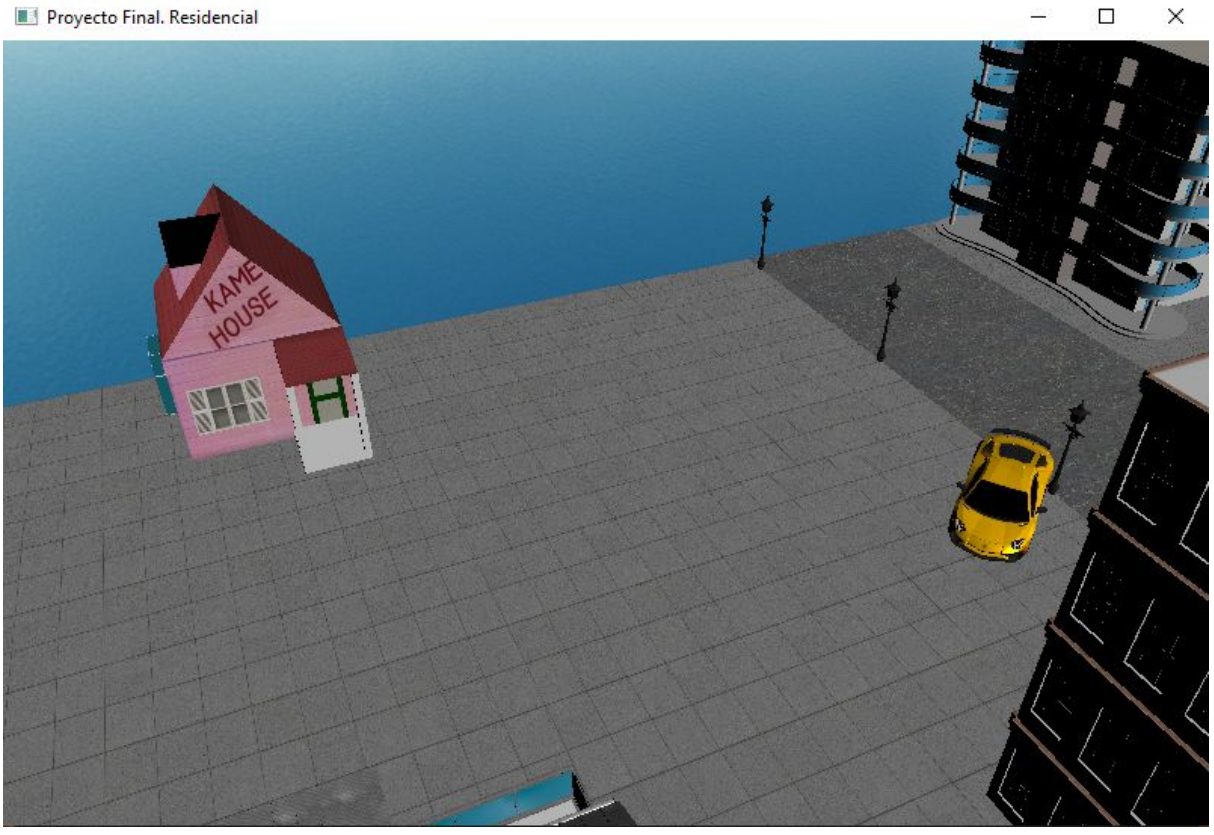


Movimiento Lego Leia

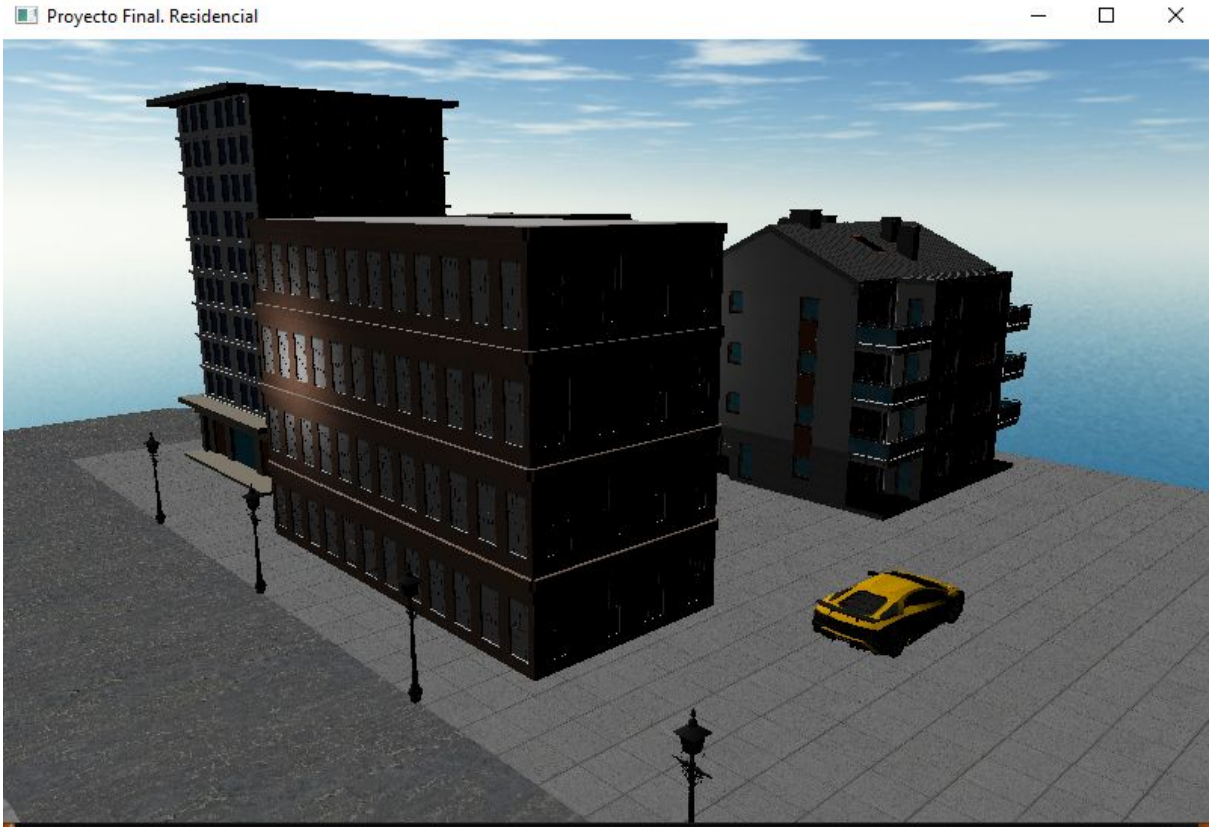
Kame House (Dragon Ball Z)

Movimiento Coche

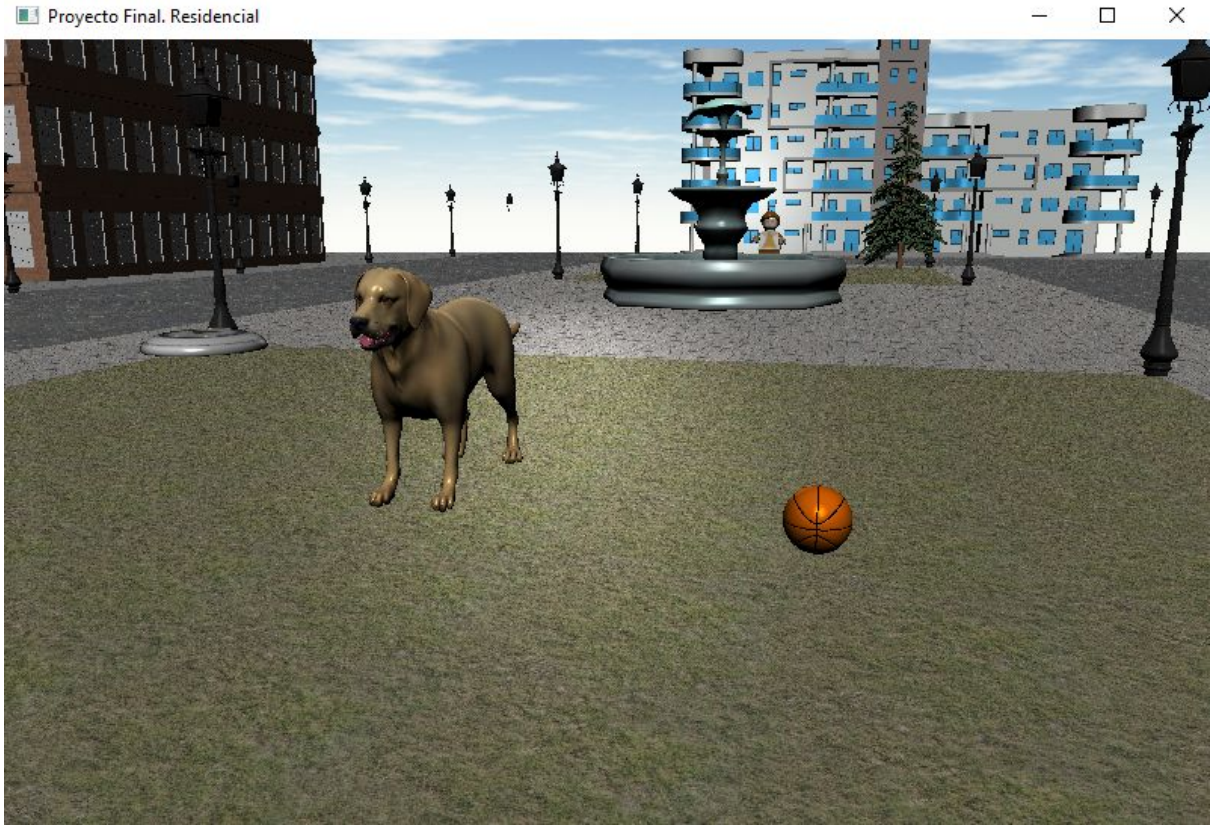




Sector A



Perro y Balón

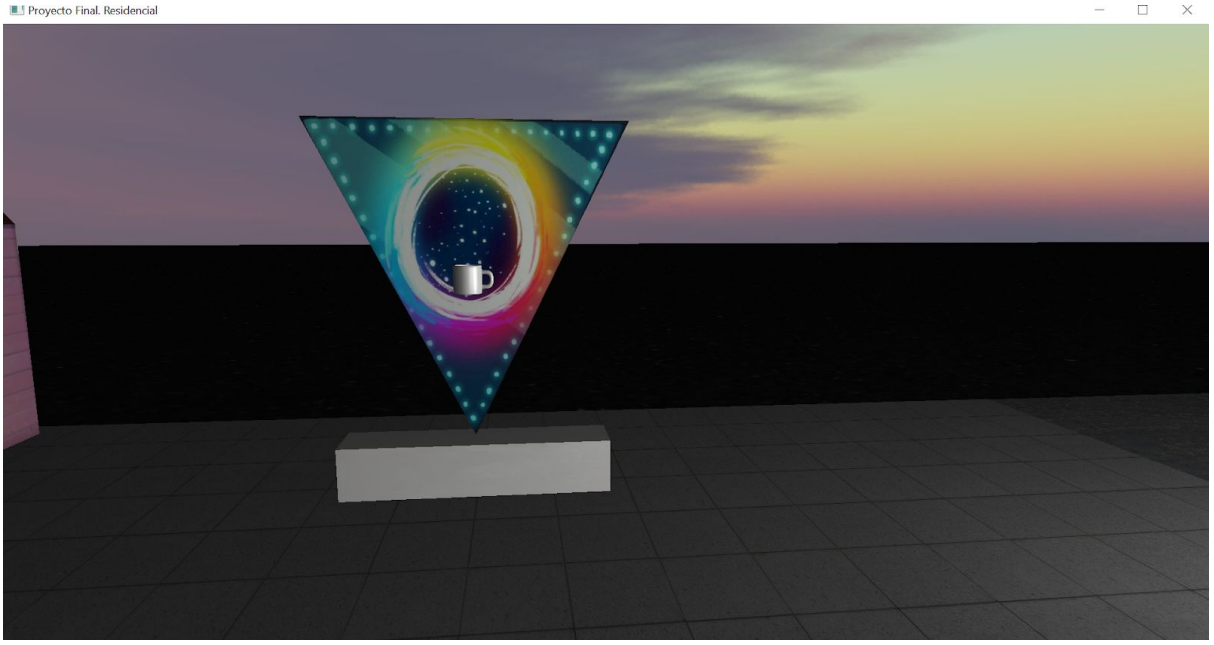


Escenario con luz solar y de faros

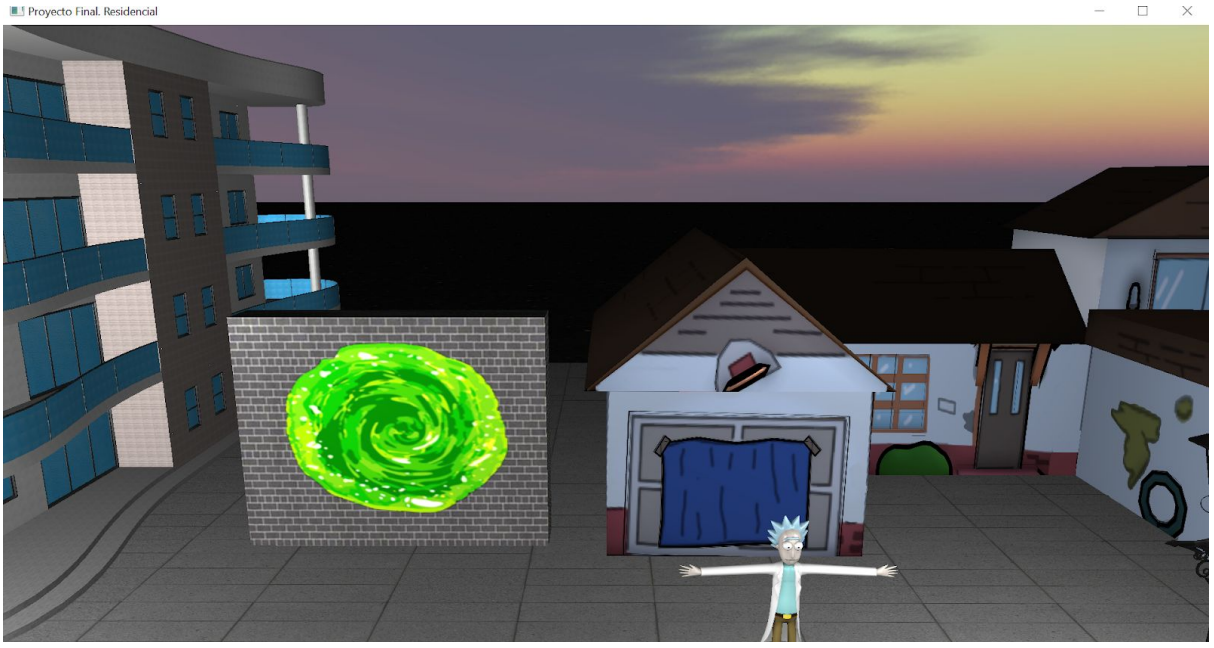
Escenario con de faros

Escenario sin luz

Portal 1



Portal 2



6. Comentarios finales

Este proyecto fue muy productivo, pues logramos poner a prueba los conocimientos adquiridos a lo largo del semestre acerca de la materia. Fue un trabajo completo y pesado, pues a pesar de no fue tan complicado, hablando en términos de programación, requirió de mucho tiempo, paciencia y dedicación.

Fue un proceso lleno de retos, con diversas subtareas que se debían completar, muchas de ellas dependían de factores, que en ocasiones hizo que nos desviaremos del plan original. No obstante logramos adaptarnos a las circunstancias y encontrar las soluciones apropiadas a los problemas que surgían y mantener la calidad que queríamos en un principio.

Cabe resaltar que casi todas las edificaciones son modelos que tuvimos que tratar en 3ds max para poder utilizarlos. La casa de Rick y la Kame House son hechas enteramente con primitivas en código duro, y modelado jerárquico.

Omitiendo algunos puntos como la configuraciones de los modelos que queríamos cargar, sabíamos implementar lo necesario para completar el proyecto. Sin embargo fue un proceso que requirió tiempo y mucho esfuerzo, y fue necesario de una buena organización y comunicación por parte del equipo que afortunadamente tuvimos y logramos cumplir con los requisitos y objetivos del trabajo.

Quedamos contentos con el resultado, pues se aprecia una aplicación real de lo que aprendimos, tenemos un escenario animado con gráficos aceptables. No obstante sabemos que se pueden incluir más animaciones y objetos.