Creando un Cubo de Rubik's 2x2

Bastián Rubio Moya

Universidad de O'Higgins

Resumen

Por medio de este trabajo hemos podido realizar un programa, en el cual se puede podemos resolver un Cubo de Rubik 2x2. Para la creación de dicho programa se utilizó el lenguaje de programación Python junto con las librerías OpenGl, PyQt5, entre otras.

Introducción

Para poder realizar este proyecto, se requieren una serie de conocimientos previos, primero se necesita saber cómo funcionan las transformaciones geométricas principales para la mayoría de los videojuegos que son la translación y rotación en tres dimensiones.

A continuación mostraremos como llevamos a cabo dicho juego en Python, ocupando las transformaciones geométricas (Traslación y Rotación), la creación del tablero y el movimiento de sus piezas, un botón para dar una jugada aleatoria del oponente y por último un contador de jugadas realizadas.

Materiales

Los materiales que se utilizaron fueron los siguientes:

- **Python**: Lenguaje de programación.
- **Visual Studio Code**: IDLE donde se puede trabajar con Python.
- **QtDesigner**: Programa para la visualización y creación del proyecto.
- **Librerías de Python**: sys, PyQt5, OpenGl, random.

Creación de la interfaz

Antes de crear el Cubo de Rubik en 3D, debemos crear la interfaz donde se mostrará el rompecabezas, para ello utilizamos el programa QtDesigner donde creamos la vista y los botones necesarios para el funcionamiento.

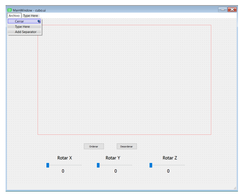


Figura 1. Creación de la interfaz.

Una vez que tenemos la interfaz vamos con los siguientes pasos.

Configuración de la interfaz

Una vez creada la interfaz debemos configurarla en nuestro archivo .py, para poder visualizar el cubo 3D, a continuación se mostrará el código base.

```
class Ventana(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = uic.loadUi('cubo.ui')
        self.ui.setWindowTitle('Rubik')
        self.ui.setWindowIcon(QtGui.QIcon('uoh.jpg'))
        self.ui.setWindowFlags(Qt.WindowMinimizeButtonHint)

        self.viewer3D = Viewer3DWidget(self)
        self.ui.OpenGLLayout.addWidget(self.viewer3D)
        self.ui.show()
```

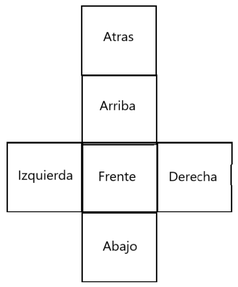
Figura 2. Configuración de la interfaz.

Creación de las piezas

Luego de tener nuestra interfaz donde veremos el rompecabezas, pasaremos a crear los cubos necesarios para construir el Cubo de Rubik. Como queremos crear un cubo en tres dimensiones, tenemos que tener en cuenta que este debe tener 6 caras, las cuales se pueden descomponer de la siguiente forma:

```
# Cuadrado 1
glBegin(GL_QUADS)
glColor3f(1,1,1)
glVertex3f(-0.5,-0.5, 0.5)
glVertex3f( 0.5,-0.5, 0.5)
glVertex3f( 0.5, 0.5, 0.5)
glVertex3f(-0.5, 0.5, 0.5)
glEnd()
```

(a) Código del cubo.



(b) Cubo en 2D.

Entonces, como tiene 6 caras debemos crear 6 cuadrados en dos dimensiones y luego juntarlos, el código para realizar esto es el siguiente: Ahora que tenemos la primera pieza veamos como crear el Cubo completo.

Creación del cubo

Ahora que tenemos una pieza, lo que debemos hacer es trasladar dicha pieza basándose en coordenadas del plano cartesiano en tres dimensiones. Para ello utilizaremos la función glTranslatef(x, y, z) con cada uno de los cubos, con una matriz de coordenadas para ir posicionándolos según correspondan.

```
translates_coords = [
    [0.0,0.0,0.0],
    [0.0,1.0,0.0],
    [1.0,0.0,0.0],
    [1.0,1.0,0.0],
    [0.0,0.0,1.0],
    [0.0,1.0,1.0],
    [1.0,0.0,1.0],
    [1.0,1.0,1.0]

]
```

Figura 4. Matriz de coordenadas

Dichas coordenadas son lo que se moverán los cubos en los respectivos ejes x, y, z.

Colores del cubo

Cada cubo varía en sus colores, en este caso se utilizaron los colores rojo, verde, azul, amarillo, naranja y blanco. Para pintar cada cara utilizamos listas por cada cubo donde se controla que color va en qué cara. Al momento de colorear el cubo se hace mediante la función glColor3f.

```
self.c0 = [0,1,2,3,4,5] # Atras Abajo Izquierda
self.c1 = [0,1,2,3,4,5] # Atras Arriba Izquierda
self.c2 = [0,1,2,3,4,5] # Atras Abajo Derecha
self.c3 = [0,1,2,3,4,5] # Atras Arriba Derecha
self.c4 = [0,1,2,3,4,5] # Adelante Abajo Izquierda
self.c5 = [0,1,2,3,4,5] # Adelante Arriba Izquierda
self.c6 = [0,1,2,3,4,5] # Adelante Abajo Derecha
self.c7 = [0,1,2,3,4,5] # Adelante Arriba Derecha
```

Figura 5. Colores en cada cara

Siendo 1=Rojo, 2=Verde, 3=Azul, 4=Amarillo, 5=Naranja y 6=Blanco, y el orden de la lista es frente, atrás, arriba, abajo, izquierda, derecha.

Movimiento de las piezas

Esta es la parte más complicada del software debido a que, tenemos que realizar diversas funciones para asegurar el correcto comportamiento del movimiento que se realizará. Primero debemos averiguar cuáles son los cubos que deberán moverse, es decir, los que cambiaran de colores, para ello realizamos el siguiente algoritmo:

Figura 6. Algoritmo para seleccionar cubos

Luego de tener estos cubos, realizamos su cambio de color.

```
if(movimiento == 1): #Arriba
    frente = cubos[i][0]
    atras = cubos[i][1]
    arriba = cubos[i][2]
    abajo = cubos[i][3]

    cubos[i][0] = abajo
    cubos[i][1] = arriba
    cubos[i][2] = frente
    cubos[i][3] = atras
```

Figura 7. Función rotación de colores

Ahora realizamos lo mismo para cada movimiento posible los cuales son: Arriba, Abajo, Derecha y Izquierda.

Resultado final

A continuación mostraremos el resultado finalizado.

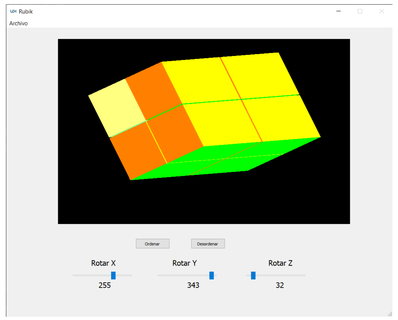


Figura 8. Tablero finalizado

Podemos notar que hay un cubo que es de distinto color, eso significa que es el cubo seleccionado, el cual se puede cambiar mediante el uso de las flechas que tiene el teclado. Además podemos rotar el cubo mediante los sliders modificando los valores de rotación de los distintos ejes.

Conclusión

Gracias a este proyecto, pude aprender como crear correctamente figuras complejas en tres dimensiones en Python utilizando OpenGl y PyQt5. Con esto, nos permite crear en un futuro software de mayor nivel que nos permitan ayudar al mundo en ámbitos como la aviación, industrias de todo tipo, etc. Ya que estas industrias requieren el modelamiento en tres dimensiones de los objetos. Finalmente, debemos tener en cuenta que basándonos en estas transformaciones geométricas se pueden explicar muchísimos fenómenos de nuestra vida cotidiana.