SEP SNEST DGEST

# TECNOLÓGICO NACIONAL DE MÉXICO

# INSTITUTO TECNOLÓGICO DE TOLUCA

Carrera:
Ingeniería en Sistemas

Materia:
***Graficación***

Nombre del proyecto:

**Objeto en 3D**

P R E S E N T A:

Alumno:

Jair Garduño Rodriguez

Nombre docente:

Rocio Elizabeth Pulido Alba

Metepec, Estado de México, a 10 de Marzo de 2024

**Reporte de Prácticas**

| Practica No # | | |
|---|---|---|
| Título de la Práctica<br>**Objeto en 3D** | Periodo Escolar<br>ENERO-JUNIO | Fecha de Elaboración<br>28/02//2024 |
| Desarrollada por | | |
| No. Control | Nombre del (los) Alumno(s) | |
| 21281153 | Jair Garduño Rodriguez | |

## Introducción del tema tratado

La creación y manipulación de objetos en tres dimensiones (3D) es una habilidad fundamental en el campo del desarrollo de software y la animación gráfica. En este conjunto de prácticas, exploraremos los conceptos básicos y avanzados de la representación de objetos en 3D utilizando la biblioteca Pygame en el lenguaje de programación Python. Pygame ofrece una plataforma versátil y accesible para el desarrollo de gráficos en 2D y 3D, lo que lo convierte en una opción ideal para aquellos que deseen explorar el mundo de la programación gráfica.

Durante estas prácticas, aprenderemos a crear y manipular objetos tridimensionales, aplicar técnicas de iluminación y materiales para mejorar su apariencia visual, y explorar diversas técnicas de renderizado para lograr efectos realistas. Además, experimentaremos con la interacción del usuario y la creación de entornos interactivos en 3D, lo que nos permitirá aplicar nuestros conocimientos en la creación de proyectos prácticos, como juegos o simulaciones.

## Definición de Problema

**Objetivo General**

- Desarrollar habilidades prácticas y conocimientos teóricos en la creación y manipulación de objetos en 3D utilizando la biblioteca Pygame en Python.

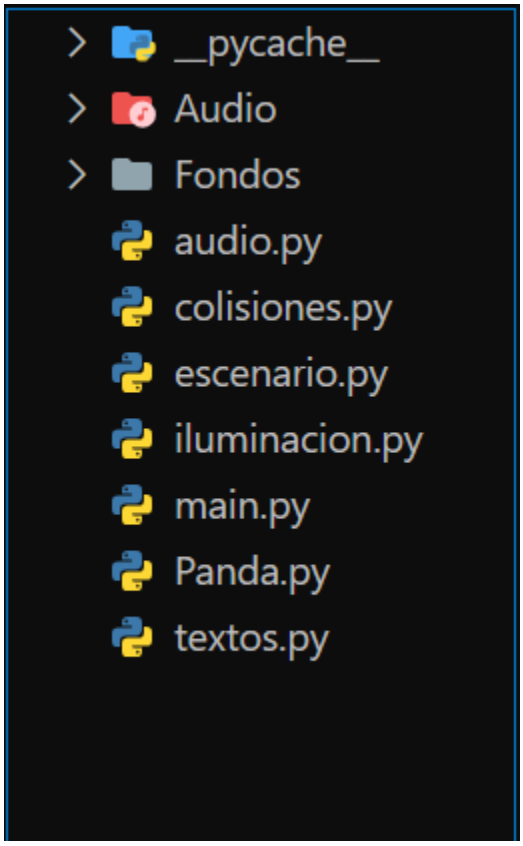
**Objetivos Específicos**

- Aprender los fundamentos de la representación de objetos en 3D, incluyendo conceptos como vértices, caras, y normales.
- Familiarizarse con la biblioteca Pygame y su capacidad para renderizar gráficos en 3D.
- Practicar la creación de objetos tridimensionales simples, como cubos, esferas, y pirámides, utilizando Pygame.
- Entender los principios básicos de la transformación y manipulación de objetos en 3D, incluyendo rotación, traslación, y escalamiento.

- Experimentar con la iluminación básica y materiales para mejorar la apariencia visual de los objetos en 3D.

## Resultado Explicado

Empezare explicando lo que es la estructura de mi proyecto, en primer lugar este proyecto tiene dos carpetas principales, audio y fondos. En esta carpeta se encuentran las imágenes y los audios que tendremos que utilizar en el proyecto, cabe aclarar que los audios tienen un formato en .wav




Ahora explicare lo que es mi clase audio, en esta clase nos encargamos de ver los canales de audio junto con las rutas donde se encontraran los archivos de audio en formato .wav

```python
import pygame

class ReproductorAudio:
    def __init__(self, num_canales=2):
        pygame.mixer.init()
        pygame.mixer.set_num_channels(num_canales)

    def reproducir_audio(self, ruta, canal=0):
        pygame.mixer.Channel(canal).play(pygame.mixer.Sound(ruta))

    def detener_audio(self, canal=0):
        pygame.mixer.Channel(canal).stop()

```

Siguiendo con la explicación la siguiente es la clase textos, esta clase nos la dio la profesora para que nosotros pudiéramos colocar los textos para las instrucciones y para donde realmente lo necesitemos como en el caso de los datos del alumno.

```python
import pygame as py
from OpenGL.GLU import *
from OpenGL.GL import *
from OpenGL.GLUT import *

def texto(text, posx, posy, posz, sizeFont, R, G, B, RB, GB, BB):
    font = py.font.Font(None, sizeFont)
    text_surface = font.render(text, True, (R, G, B), (RB, GB, BB))
    text_data = py.image.tostring(text_surface, "RGBA", True)
    glRasterPos3d(posx, posy, posz)
    glDrawPixels(text_surface.get_width(), text_surface.get_height(), GL_RGBA, GL_UNSIGNED_BYTE, text_data)
```

Justo después de explicar esta pequeña clase continuare explicando mi clase escenario, realmente lo que hace esta clase es sacar el ID de la o las imágenes que vamos a utilizar para nuestro proyecto, estas claves son almacenadas en un diccionario para que cada que se llame no se tengan que estar generando de nuevo puesto que utiliza demasiados recursos.

```python
import pygame as py
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
from PIL import Image

class Escenario:
    texture_ids = {}  # Diccionario para almacenar los IDs de textura

    def __init__(self):
        pass

    @classmethod
    def load_texture(cls, filename):
        with Image.open(filename) as im:
            ix, iy, image = im.size[0], im.size[1], im.tobytes("raw", "RGBX", 0, -1)

        texture_id = glGenTextures(1)
        glBindTexture(GL_TEXTURE_2D, texture_id)
        glPixelStorei(GL_UNPACK_ALIGNMENT, 1)
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, ix, iy, 0, GL_RGBA, GL_UNSIGNED_BYTE, image)
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)

        return texture_id
```

Como lo podemos ver los métodos ya están comentados con que hace cada una de las líneas, sin embargo, procederé a explicarlo. Como primera parte el siguiente método que aparece se basa en generar la textura con el nombre del archivo, esto lo regresa después para que sea almacenado en diccionario. Después el método escenario es donde se encargará de dibujar el cubo que hace al escenario, este mismo método manda a llamar a los demás puesto que se podría ver como el control de este mismo.

```
    @classmethod
    def get_texture_id(cls, filename):
        if filename not in cls.texture_ids:  # Verificar si la textura ya ha sido cargada
            cls.texture_ids[filename] = cls.load_texture(filename)  # Si no se ha cargado, cargarla

        return cls.texture_ids[filename]  # Retornar el ID de textura correspondiente al archivo

    @classmethod
    def escenario(cls, filename):
        texture_id = cls.get_texture_id(filename)  # Obtener el ID de textura correspondiente al archivo

        glEnable(GL_TEXTURE_2D)
        glBindTexture(GL_TEXTURE_2D, texture_id)  # Usar el ID de textura obtenido

        # Definir vértices y coordenadas de textura para el techo, piso y laterales
        vertices = [
            (-30, -25, 30), (30, -25, 30), (30, 25, 30), (-30, 25, 30),  # Frente
            (-30, -25, -30), (30, -25, -30), (30, 25, -30), (-30, 25, -30),  # Atrás
            (-30, 25, 30), (30, 25, 30), (30, 25, -30), (-30, 25, -30),  # Techo
            (-30, -25, 30), (30, -25, 30), (30, -25, -30), (-30, -25, -30),  # Piso
            (-30, -25, 30), (-30, 25, 30), (-30, 25, -30), (-30, -25, -30),  # Lateral izquierdo
            (30, -25, 30), (30, 25, 30), (30, 25, -30), (30, -25, -30)  # Lateral derecho
        ]

```

Aquí están las coordenadas de cada cara del cubo, además de como se están dibujando gracias a esos dos ciclos for.

```
        tex_coords = [
            (0, 0), (1, 0), (1, 1), (0, 1),  # Frente
            (0, 0), (1, 0), (1, 1), (0, 1),  # Atrás
            (0, 0), (1, 0), (1, 1), (0, 1),  # Techo
            (0, 0), (1, 0), (1, 1), (0, 1),  # Piso
            (0, 0), (1, 0), (1, 1), (0, 1),  # Lateral izquierdo
            (0, 0), (1, 0), (1, 1), (0, 1)   # Lateral derecho
        ]

        # Dibujar el escenario con los nuevos vértices y coordenadas de textura
        for i in range(0, len(vertices), 4):
            glBegin(GL_QUADS)
            glColor(1, 1, 1)
            for j in range(4):
                glTexCoord2f(tex_coords[i + j][0], tex_coords[i + j][1])
                glVertex3f(*vertices[i + j])
            glEnd()

        glDisable(GL_TEXTURE_2D)
```

Mi clase de iluminación solo se encarga de activar estas texturas en la figura.

```
Click here to ask Blackbox to help you code faster
from OpenGL.GLU import *
from OpenGL.GL import *
from OpenGL.GLUT import *

iluminacion_activada = True

def toggle_iluminacion():
    global iluminacion_activada
    iluminacion_activada = not iluminacion_activada

    if iluminacion_activada:
        glEnable(GL_LIGHTING)
    else:
        glDisable(GL_LIGHTING)

def iluminacion(R, G, B):
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glEnable(GL_LIGHT1)
    glEnable(GL_LIGHT2)
    glEnable(GL_LIGHT3)
    glEnable(GL_DEPTH_TEST)

    # Light properties
    light_ambient = (0.0, 0.0, 0.0, 1.0)
    light_diffuse = (R, G, B, 1.0)

```

Como lo podemos observar solo se encarga de activar los efectos en todo el objeto.

```
52
53      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material_ambient)
54      glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse)
55      glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material_specula
56      glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, 50.0)
57
58
59  def display():
60      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
61      glLoadIdentity()
62      glColor3f(1.0, 1.0, 1.0)
63      glutSolidTeapot(1.0)
64      glutSwapBuffers()
```

Ahora, la clase panda solo se encarga de dibujar cada parte de mi figura, pero como son cubos realmente las 1000 líneas que tiene la clase son idénticas con excepción de los colores y las coordenadas que tiene cada método, puesto que cada parte de la figura la separe por método. Ahora mismo muestro un ejemplo de lo que me refieron.

```
  6  class Panda:
412      def draw_eye1(self,color,color2):
413          # Derecho
414          glBegin(GL_QUADS)
415          #glColor3f(0.5, 0.5, 0.5)
416          glColor3f(color[0],color[1],color[2])  # Color blanco
417          glVertex3f(3.2, -0.49, 0.9)
418          glVertex3f(3, -0.49, 0.9)
419          glVertex3f(3, -0.49, 0.7)
420          glVertex3f(3.2, -0.49, 0.7)
421          glEnd()
422
423          glBegin(GL_QUADS)
424          glColor3f(color2[0],color2[1],color2[2])
425          glVertex3f(3, -0.49, 0.9)
426          glVertex3f(2.8, -0.49, 0.9)
427          glVertex3f(2.8, -0.49, 0.7)
428          glVertex3f(3, -0.49, 0.7)
429          glEnd()
430
431          # Derecho lineas
432          glBegin(GL_LINE_LOOP)
433          glColor3f(0, 0.2, 0.6)  # Color rojo
434          glVertex3f(3.2, -0.49, 0.9)
435          glVertex3f(3, -0.49, 0.9)
436          glVertex3f(3, -0.49, 0.7)
437          glVertex3f(3.2, -0.49, 0.7)
438          glEnd()
```

Ahora la clase principal (main):

Lo que importa esta clase son los módulos de pygame, OpenGL, la clase escenario, la clase Panda, la clase Textos, la clase iluminación y la clase Audio.

```
1   import pygame
2   from pygame.locals import *
3   from OpenGL.GL import *
4   from OpenGL.GLU import *
5   from escenario import Escenario as es
6   from Panda import Panda
7   import textos as txt
8   import iluminacion as li
9   import audio as au
```

El método constructor inicializa lo que es pygame, da todos los parámetros que se usan para esto y se crea un objeto de tipo panda para poder hacer uso de los métodos de la clase, una variable de tipo booleano con un valor verdadero, las coordenadas en X,Y para las traslaciones del panda, también esta un objeto de tipo Audio para checar la reproducción y los canales, dos variables de tipo tupla para ver los colores de los ojos, una variable con la ruta de mi escenario principal y variables de la cámara.

```
11  class Main:
12      # Función para inicializar Pygame y OpenGL
13
14      def __init__(self):
15          pygame.init()
16          width, height = 800, 600
17          window = pygame.display.set_mode((width, height), DOUBLEBUF | OPENGL)
18          pygame.display.set_caption('Panda')
19          self.pd = Panda()
20          self.op = True
21          self.x = 0
22          self.y = 0
23          self.audio = au.ReproductorAudio(num_canales=2)
24          self.colorOjo = (0.5, 0.49, 0.5)
25          self.colorOjo2 = (1,1,1)
26          self.escenario = "Fondos/R.jpeg"
27          glClearColor(0.0, 0.0, 0.0, 1.0)
28          glMatrixMode(GL_PROJECTION)
29          gluPerspective(45, (width / height), 0.1, 130.0)
30          glMatrixMode(GL_MODELVIEW)
31          glEnable(GL_DEPTH_TEST)
32          self.camera_position = [0, 0, 15.0]
33          self.camera_speed = 0.1
```

Ahora el método draw lo que hace es ver los eventos que se generan para que se cargue todo lo esencial de la cámara y procedamos a colocar el escenario, ahora tuve que hacer una pequeña rotación de mi panda puesto que quedaba viendo hacia arriba y quería que volteara a ver hacia la cámara.

Después como en todo el codigo procederemos a ver los eventos del ratón el primero que aparecerá es para ver mis datos personales.

```
66      # Función para dibujar la escena
67      def draw(self):
68          keys = pygame.key.get_pressed()
69          glLoadIdentity()
70          gluLookAt(
71              self.camera_position[0], self.camera_position[1], self.camera_position[2],
72              self.camera_position[0], self.camera_position[1], self.camera_position[2] - 1,
73              0, 1, 0
74          )
75          es.escenario(self.escenario)
76
77          glRotatef(180, 0, 1, 1)  # Rotar 90 grados alrededor del eje X
78
79          if keys[pygame.K_q]:
80
81              if(self.op):
82                  self.op = not(self.op)
83              txt.texto("Acerca de:", -1, 0, 5.0, 25, 255, 255, 255, 0, 0, 0)
84              txt.texto("Alumno: Jair Garduño Rodriguez", -1, 0, 4.5, 25, 255, 255, 255, 0, 0, 0)
85              txt.texto("No. Control: 21281153", -1, 0, 4, 25, 255, 255, 255, 0, 0, 0)
86              pygame.time.wait(100)
87
```

Los siguientes eventos será para poder colocar el menú.

```
90          if self.op:
91                  # TEXTO POSX POSY POSZ TAM R G B RB GB BB
92              txt.texto("Instrucciones", -1, 0, 5.0, 25, 255, 255, 255, 0, 0, 0)
93              txt.texto("Teclas de flecha: Mover cámara", -1, 0, 4.5, 25, 255, 255, 255, 0, 0, 0)
94              txt.texto("W/S: Avanzar/Retroceder", -1, 0, 4.0, 25, 255, 255, 255, 0, 0, 0)
95              txt.texto("L SHIFT: Restaurar posición de la cámara", -1, 0, 3.5, 25, 255, 255, 255, 0, 0, 0)
96              txt.texto("O/P: Cerrar ojo Izquierdo/Derecho", -1, 0, 3.0, 25, 255, 255, 255, 0, 0, 0)
97              txt.texto("L: Tristeza", -1, 0, 2.5, 25, 255, 255, 255, 0, 0, 0)
98              txt.texto("K: Sonrisa", -1, 0, 2.0, 25, 255, 255, 255, 0, 0, 0)
99              txt.texto("I: Mostrar Lengua", -1, 0, 1.5, 25, 255, 255, 255, 0, 0, 0)
100             txt.texto("U/J: Mover patas delanteras", -1, 0, 1.0, 25, 255, 255, 255, 0, 0, 0)
101             txt.texto("M/N: Mover patas traseras", -1, 0, 0.5, 25, 255, 255, 255, 0, 0, 0)
102             txt.texto("Y: Mover ambas patas traseras", -1, 0, 0.0, 25, 255, 255, 255, 0, 0, 0)
103             txt.texto("1/2/3/4/5: Cambiar fondo", -1, 0, -0.5, 25, 255, 255, 255, 0, 0, 0)
104             txt.texto("Z/X/C/V: Derecha/Izquierda/Adelante/Atras", -1, 0, -1, 25, 255, 255, 255, 0, 0, 0)
105             txt.texto("G: Expresión, Movimiento, Sonido, Escenario", -1, 0, -1.5, 25, 255, 255, 255, 0, 0, 0)
106             txt.texto("H/B: Encender sonido/Apagar sonido", -1, 0, -2, 25, 255, 255, 255, 0, 0, 0)
107             txt.texto("T: Ocultar/Mostrar instrucciones", -1, 0, -2.5, 25, 255, 255, 255, 0, 0, 0)
108             txt.texto("Q: Acerca de Jair", -1, 0, -3, 25, 255, 255, 255, 0, 0, 0)
109         glTranslatef(self.x,self.y,0)
110
111         if keys[pygame.K_t]:
112             self.op = not(self.op)
113             pygame.time.wait(100)
114
```

Continuando con los eventos sigue lo que es evento para que el panda se pueda mover en el escenario en X,Y.

Y el ultimo que se ve en la siguiente captura es para la iluminación.

Después como no tengo algún movimiento con la cabeza y el cuerpo simplemente los mando a llamar.

```
117         if keys[pygame.K_z]:
118             self.x += 1
119             pygame.time.wait(100)
120         if keys[pygame.K_x]:
121             self.x -= 1
122             pygame.time.wait(100)
123         if keys[pygame.K_c]:
124             self.y += 1
125             pygame.time.wait(100)
126         if keys[pygame.K_v]:
127             self.y -= 1
128             pygame.time.wait(100)
129 
130 
131         if keys[pygame.K_e]:
132             # Iluminacion
133             li.iluminacion(1.0, 1.0, 1.0)  # Luz blanca
134             li.toggle_iluminacion()
135 
136         self.pd.draw_body()
137 
138         self.pd.draw_face()
139 
```

Lo siguiente viene a ser para los ojos para que se vea como si parpadeara el panda, en el caso de que no parpadee solo se mostrara sus ojos con el color original que tenia. Y procedemos a dibujar las orejas.

Ahora el ultimo evento que se ve es para dibujar su hocico.

```
140         if keys[pygame.K_o]:
141             self.pd.draw_eye1((0, 0.2, 0.6),(0, 0.2, 0.6))
142             self.audio.reproducir_audio("Audio/1.wav",canal=0)
143             pygame.time.wait(150)
144 
145         else:
146             self.pd.draw_eye1(self.colorOjo,self.colorOjo2)
147 
148         if keys[pygame.K_p]:
149             self.pd.draw_eye2((0, 0.2, 0.6),(0, 0.2, 0.6))
150             self.audio.reproducir_audio("Audio/2.wav",canal=0)
151             pygame.time.wait(150)
152         else:
153             self.pd.draw_eye2(self.colorOjo,self.colorOjo2)
154 
155 
156         self.pd.draw_muzzle()
157 
158         if keys[pygame.K_l]:
159             self.pd.draw_mouth()
160             self.audio.reproducir_audio("Audio/3.wav",canal=0)
161             pygame.time.wait(150)
```

Continuamos con los eventos para que se dibujen las expresiones del panda, hasta que llegamos al evento G puesto que este evento combina varias cosas, combina un movimiento, sonido, escenario y una expresión.

```
163         if keys[pygame.K_k]:
164             self.pd.draw_mouth_happy()
165             self.audio.reproducir_audio("Audio/4.wav",canal=0)
166             pygame.time.wait(150)
167         elif keys[pygame.K_i]:
168             self.pd.draw_mouth_leng()
169             self.audio.reproducir_audio("Audio/5.wav",canal=0)
170             pygame.time.wait(150)
171         self.pd.draw_ears()
172 
173         if keys[pygame.K_g]:
174             es.escenario("Fondos/fondo1.jpg")
175             self.pd.draw_mouth_leng()
176             self.audio.reproducir_audio("Audio/3.wav",canal=0)
177             pygame.time.wait(150)
178             self.pd.draw_leg_inf_left()
179             self.pd.draw_leg_inf_right()
180             glRotatef(20, 1, 0, 0)  # Rotar 90 grados alrededor del eje X
181             glTranslatef(0,0,1)
182             self.pd.draw_leg_sup_left()
183             self.pd.draw_leg_sup_right()
184             glLoadIdentity()
```

Continuamos haciendo los movimientos de las patas, puesto que por cada movimiento va teniendo una acción de pata por pata.

```
186         if keys[pygame.K_u]:
187             self.pd.draw_leg_inf_left()
188             self.pd.draw_leg_sup_left()
189             self.pd.draw_leg_inf_right()
190             glRotatef(20, 1, 0, 0)  # Rotar 90 grados alrededor del eje X
191             glTranslatef(0,0,1)
192             self.pd.draw_leg_sup_right()
193             glLoadIdentity()
194 
195 
196         elif keys[pygame.K_j]:
197             self.pd.draw_leg_inf_left()
198             self.pd.draw_leg_sup_right()
199             self.pd.draw_leg_inf_right()
200             glRotatef(20, 1, 0, 0)  # Rotar 90 grados alrededor del eje X
201             glTranslatef(0,0,1)
202             self.pd.draw_leg_sup_left()
203             glLoadIdentity()
204 
205         if keys[pygame.K_m]:
206             self.pd.draw_leg_sup_left()
207             self.pd.draw_leg_sup_right()
208             self.pd.draw_leg_inf_right()
209             glRotatef(20, 1, 0, 0)  # Rotar 90 grados alrededor del eje X
210             glTranslatef(0,0,2)
211             self.pd.draw_leg_inf_left()
212             glLoadIdentity()
```

```
215         if keys[pygame.K_n]:
216             self.pd.draw_leg_sup_left()
217             self.pd.draw_leg_sup_right()
218             self.pd.draw_leg_inf_left()
219             glRotatef(20, 1, 0, 0)  # Rotar 90 grados alrededor del eje X
220             glTranslatef(0,0,2)
221             self.pd.draw_leg_inf_right()
222             glLoadIdentity()
223         else:
224             self.pd.draw_leg_inf_right()
225 
226         if keys[pygame.K_y]:
227             self.pd.draw_leg_inf_left()
228             glRotatef(20, 1, 0, 0)  # Rotar 90 grados alrededor del eje X
229             glTranslatef(0,0,1)
230             self.pd.draw_leg_sup_left()
231             self.pd.draw_leg_sup_right()
232             glLoadIdentity()
233         else:
234             self.pd.draw_leg_sup_left()
235             self.pd.draw_leg_sup_right()
236 
```

Ahora por ultimo están los eventos del audio y para los fondos, haciendo uso de sus respectivos objetos.

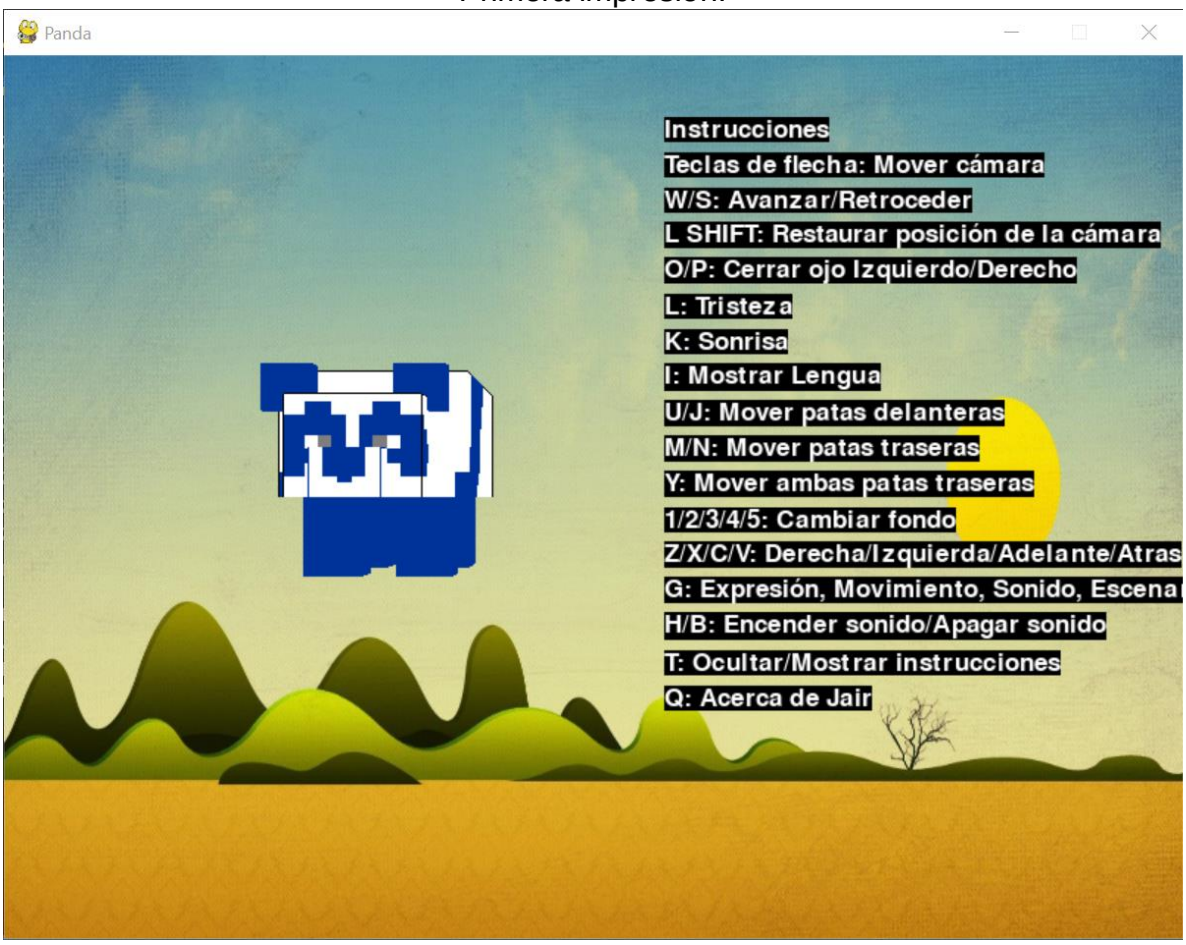
```
238         if keys[pygame.K_h]:
239             self.audio.reproducir_audio("Audio/zelda.wav",canal=1)
240         elif keys[pygame.K_b]:
241             self.audio.detener_audio(canal=1)
242 
243 
244 
245         if keys[pygame.K_1]:
246             self.escenario = ("Fondos/R.jpeg")
247         elif keys[pygame.K_2]:
248             self.escenario = ("Fondos/fondo1.jpg")
249         elif keys[pygame.K_3]:
250             self.escenario = ("Fondos/fondo2.jpg")
251         elif keys[pygame.K_4]:
252             self.escenario = ("Fondos/fondo3.jpg")
253         elif keys[pygame.K_5]:
254             self.escenario = ("Fondos/fondo4.jpg")
255 
256         pygame.display.flip()
257         pygame.time.wait(10)
258 
```

Por ultimo en la función main tenemos una llamada al objeto Main y tenemos lo que es el ciclo principal del programa, junto a su respectivo método.

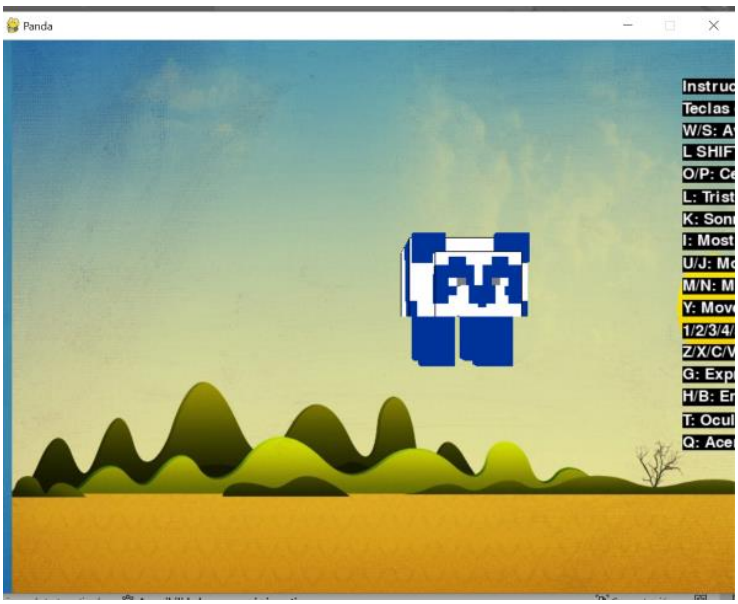
```
259     # Función principal del programa
260 def main():
261     main_instance = Main()
262 
263     running = True
264     while running:
265         running = main_instance.handle_input()
266         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
267 
268         main_instance.draw()
269 
270     pygame.quit()
271 
272 if __name__ == "__main__":
273     main()
274 
```

Explicación del programa ejecutándose:

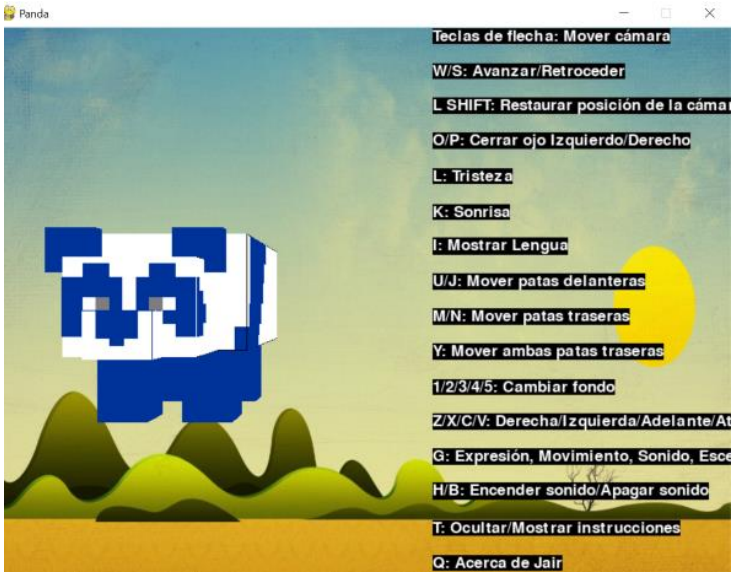
Primera impresión:



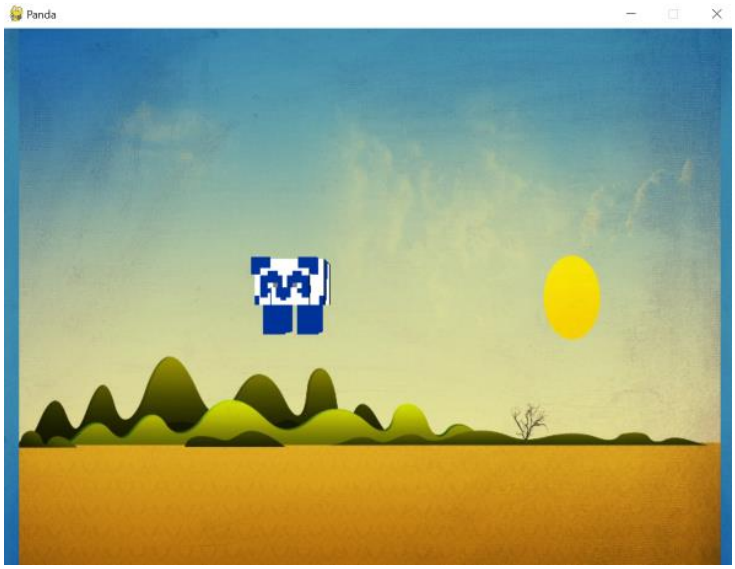

Movimiento de cámara Arriba, Izquierda, Derecha, Abajo (Se entiende mas en video).



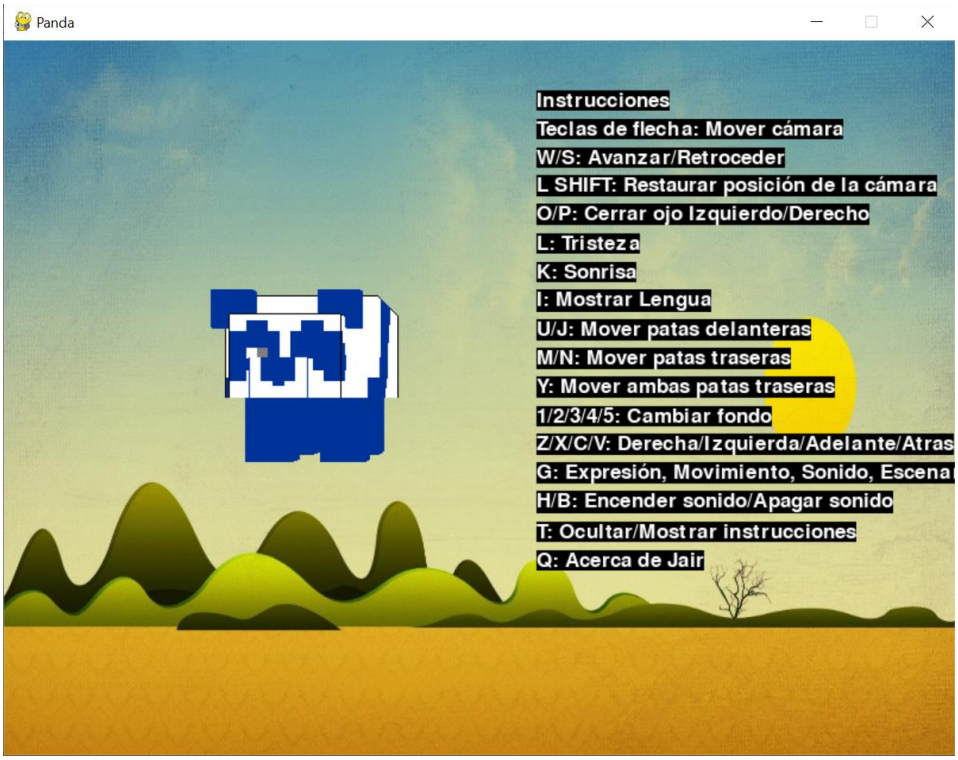
Zoom In:



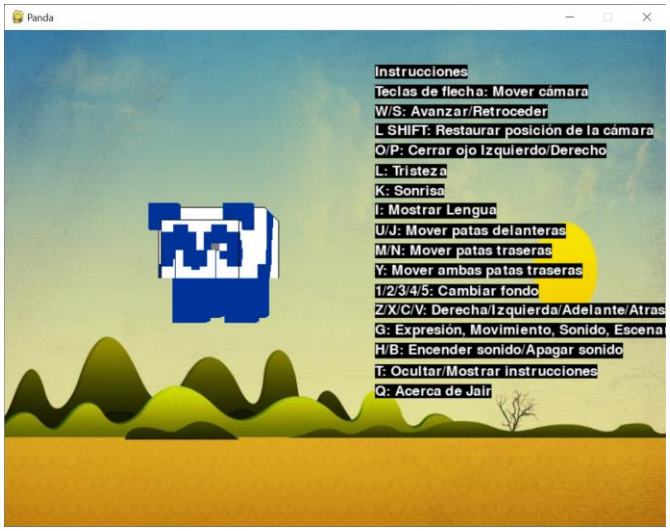

Zoom out:



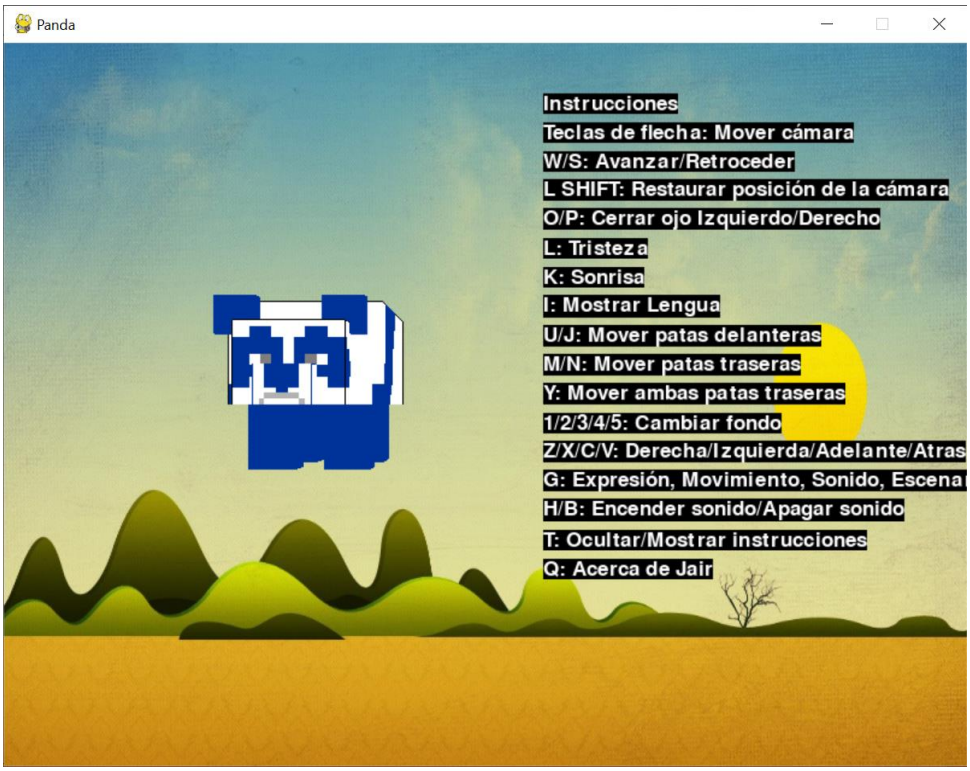

Cerrar un ojo:



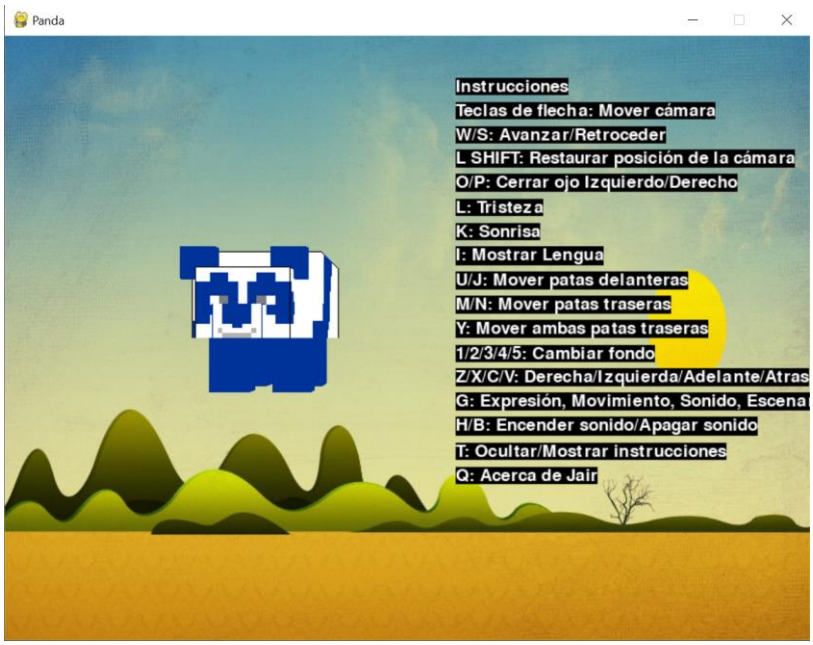

Cerrar otro ojo:



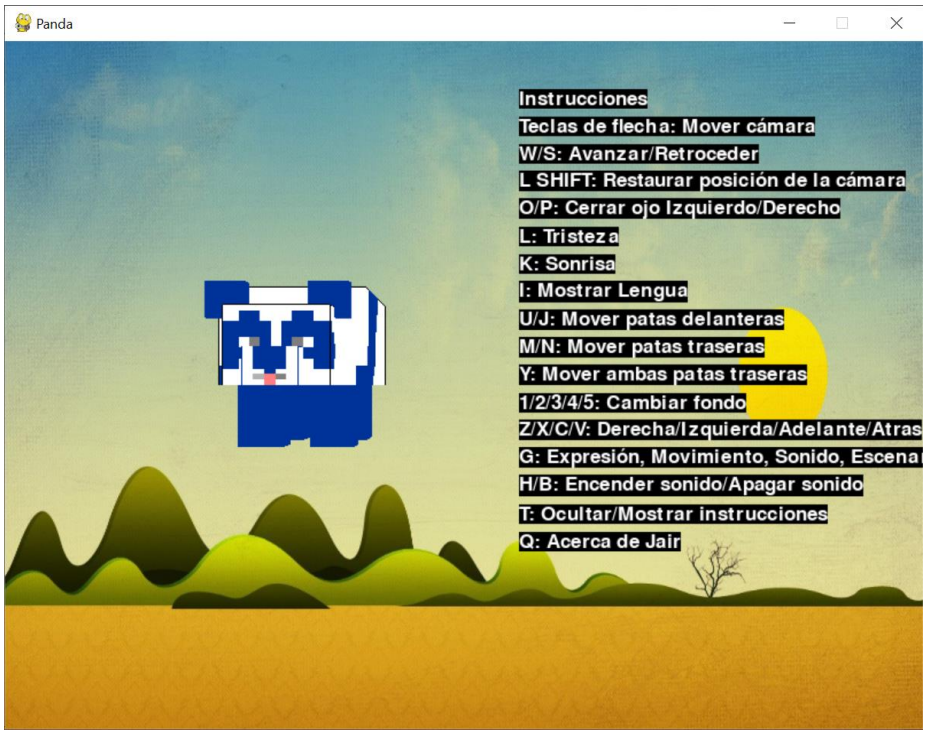
Tristeza:



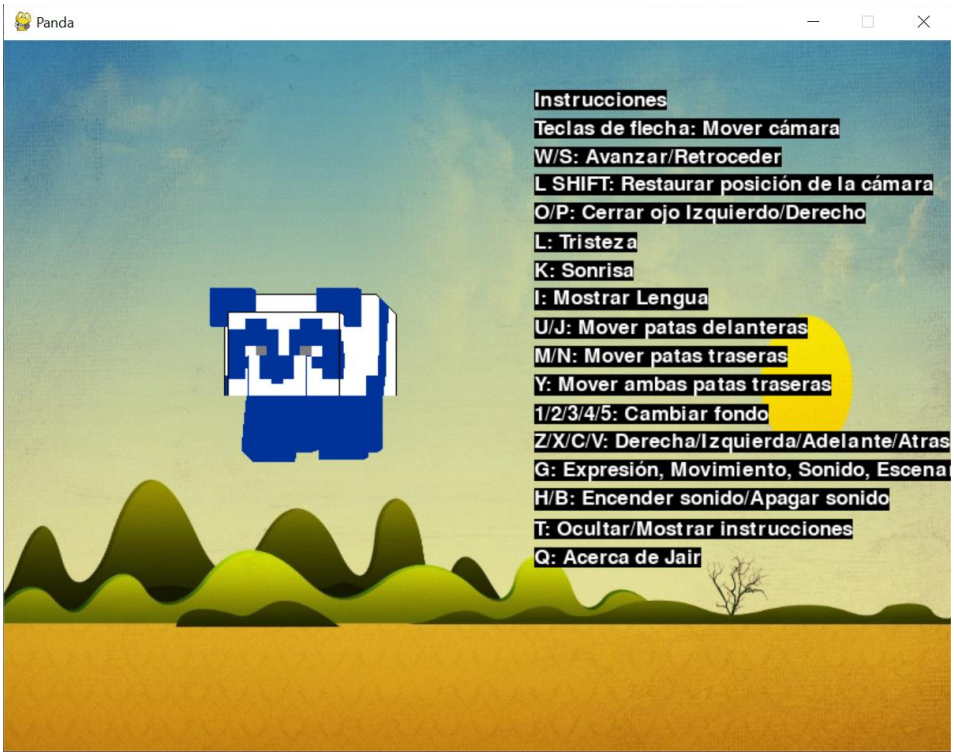

Sonrisa:



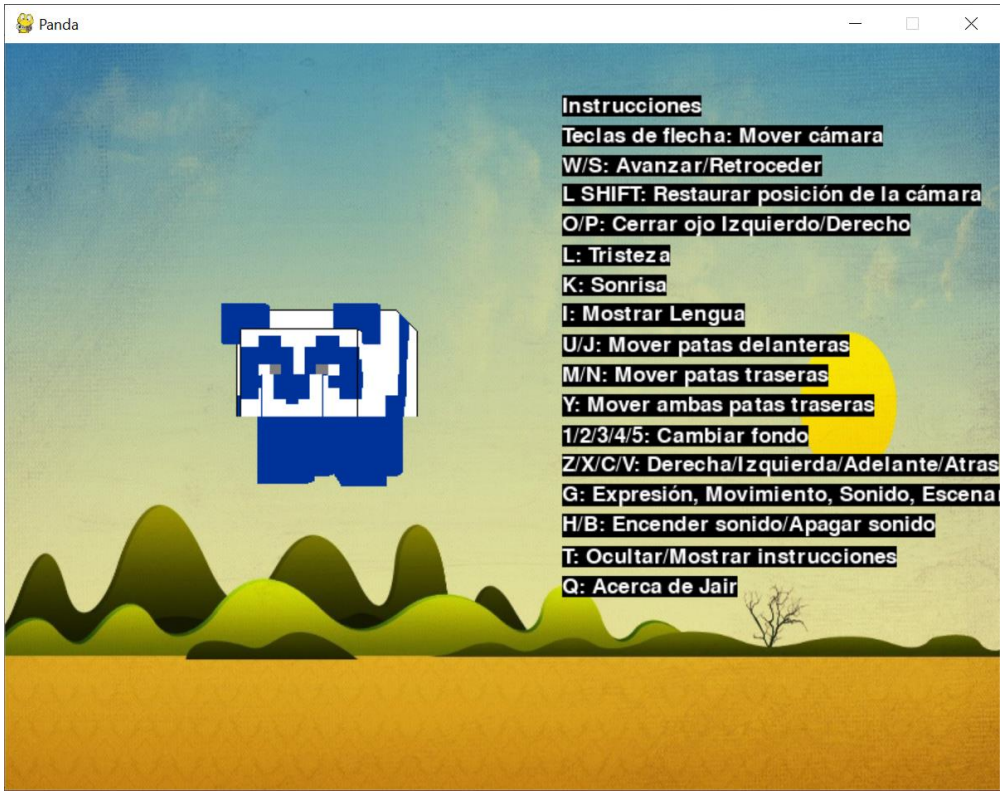
Mostrar lengua:



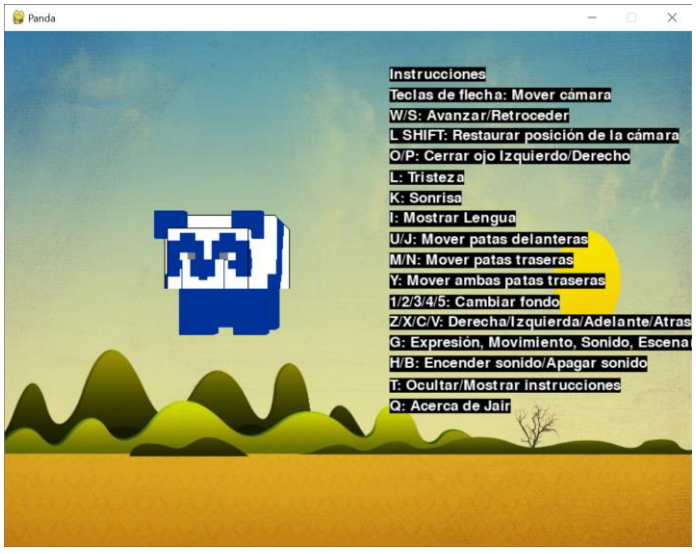

Mover pata derecha delantera:



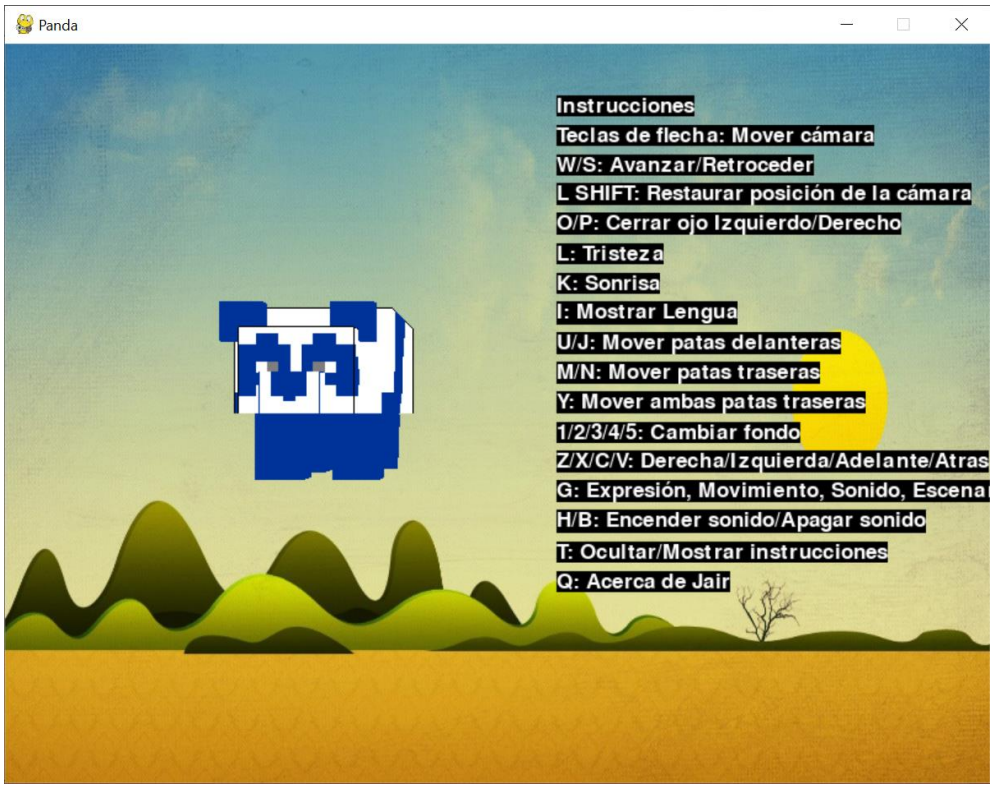
Mover pata izquierda delantera:



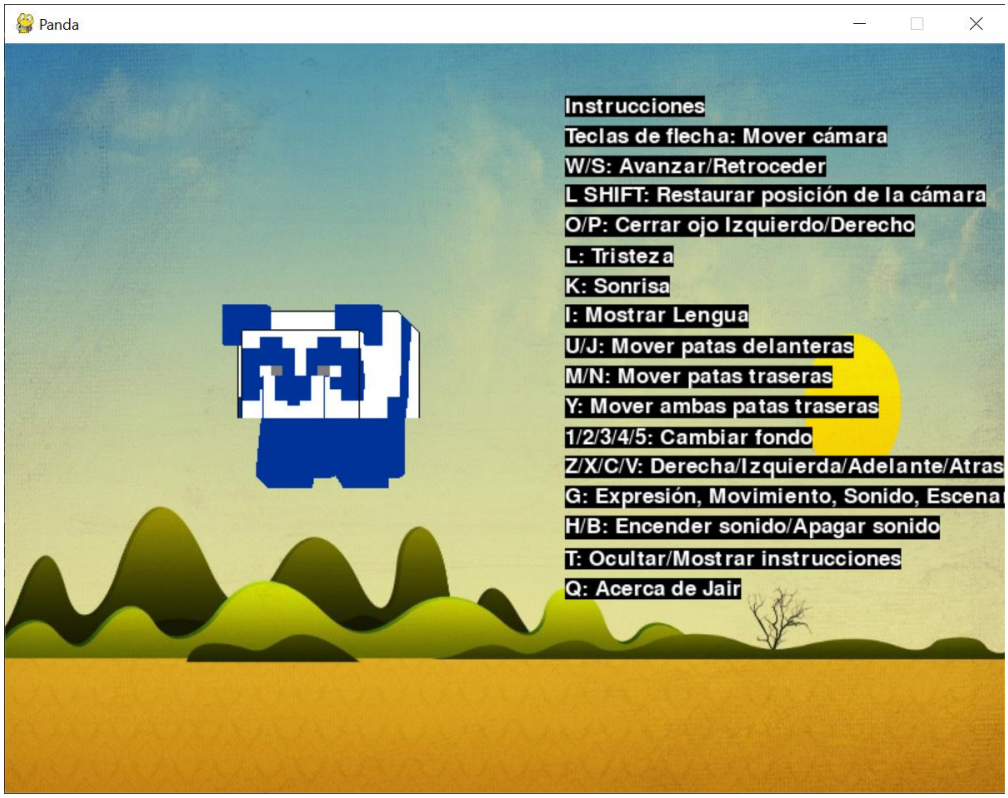

Mover pata derecha trasera:



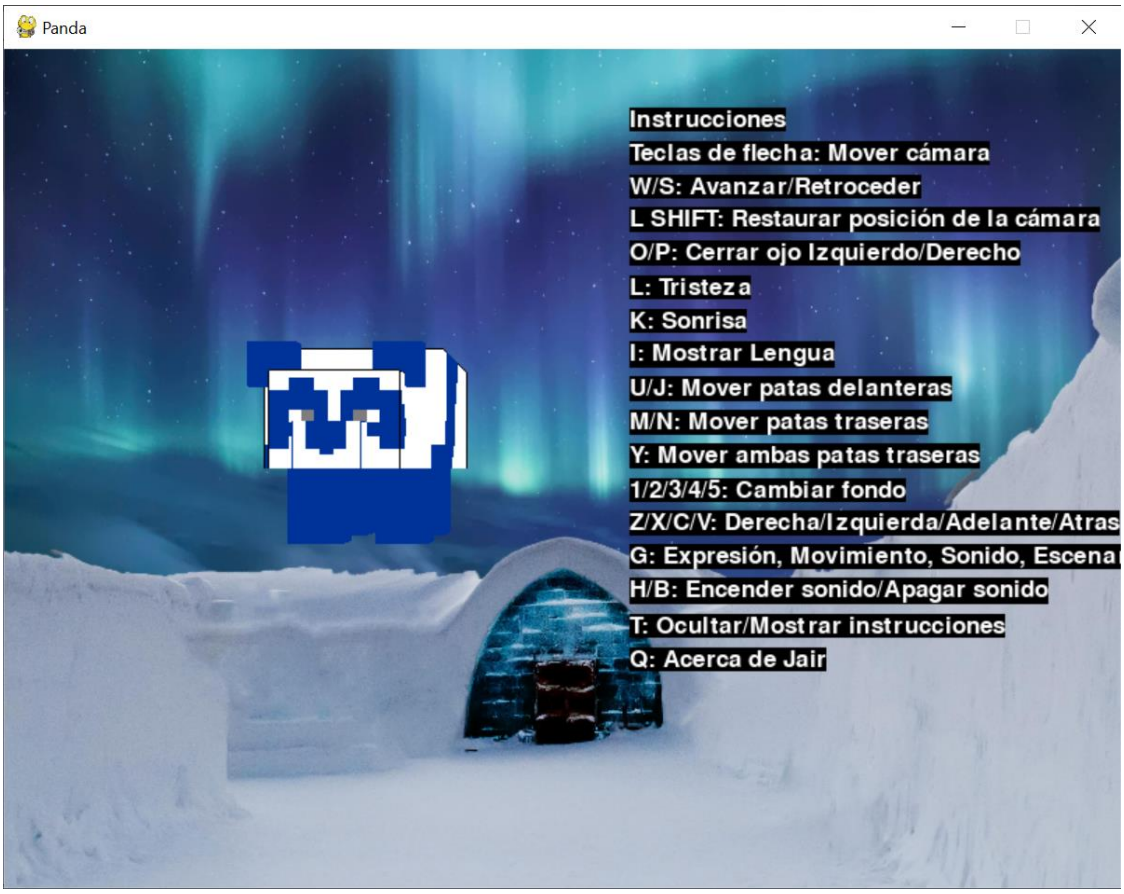
Mover pata izquierda trasera:



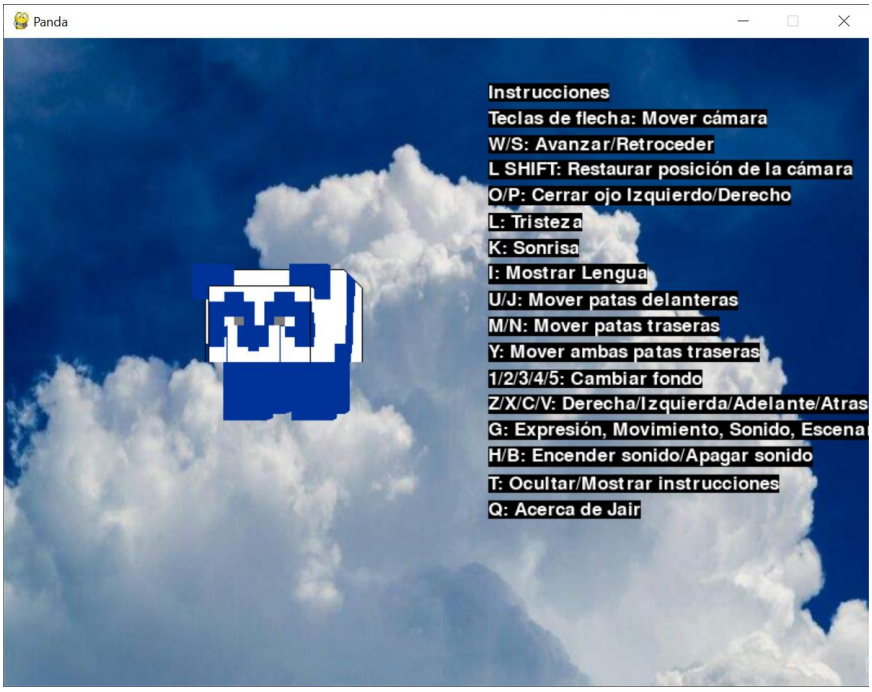

Mover ambas patas:



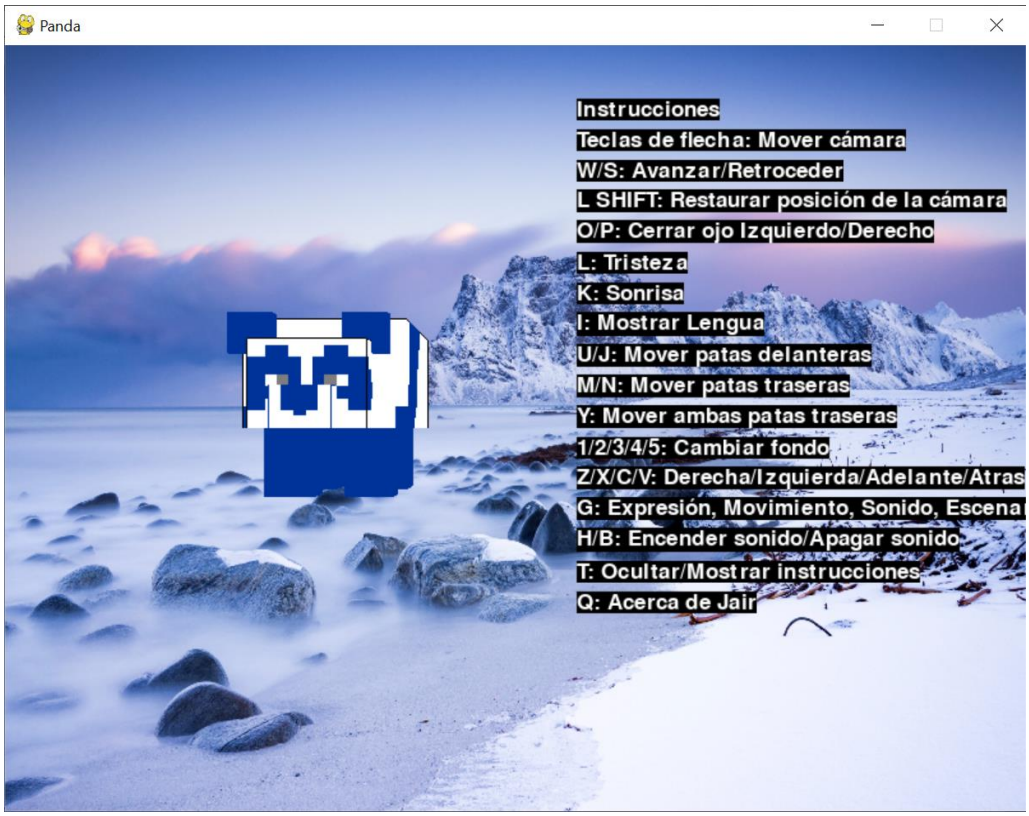
Fondo 2:



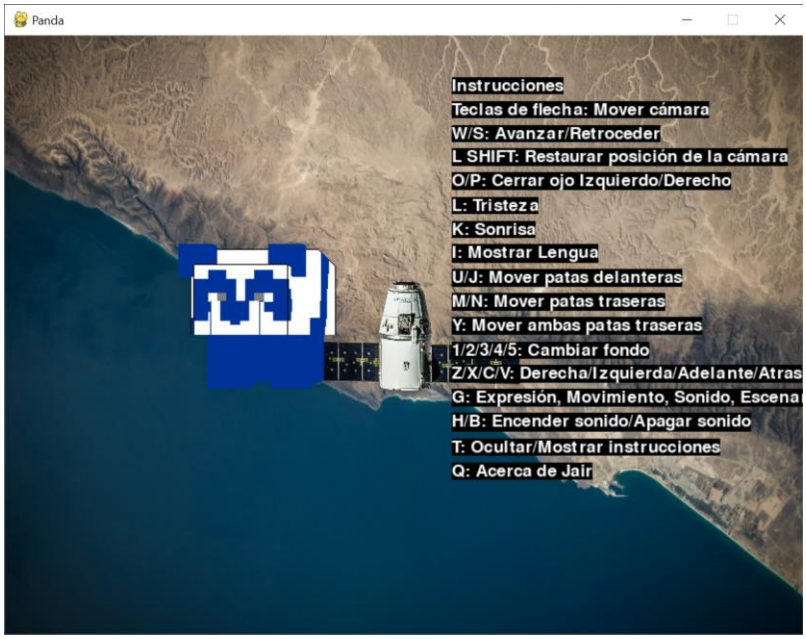

Fondo 3:



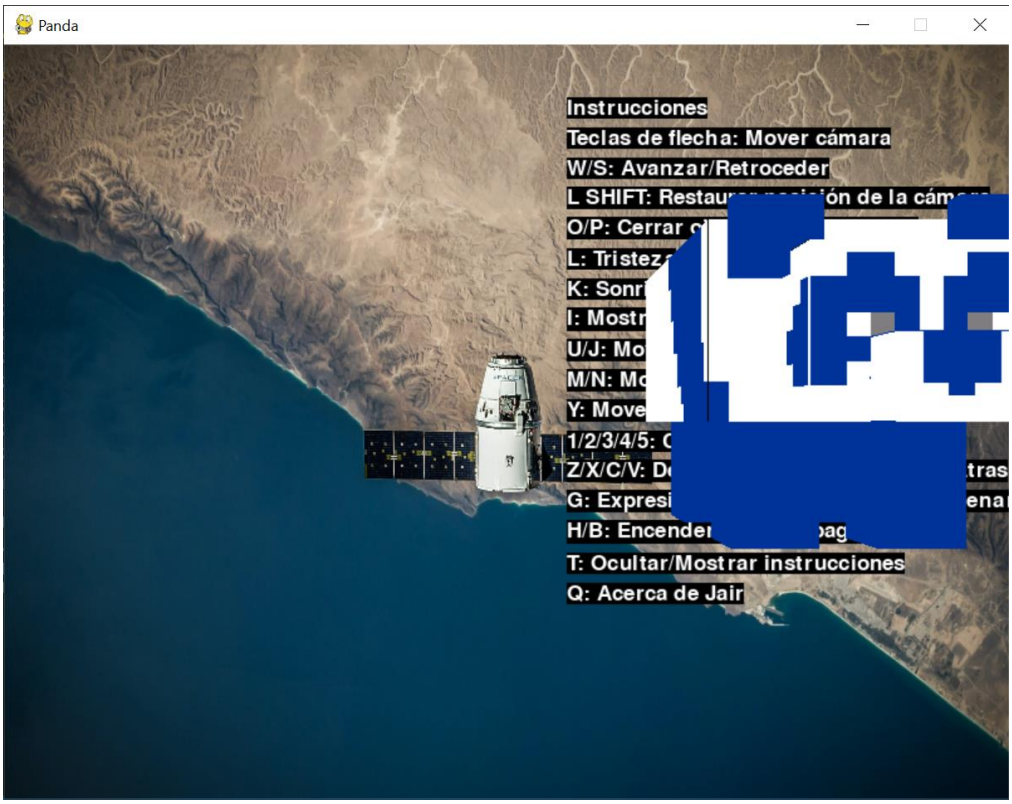
Fondo 4:



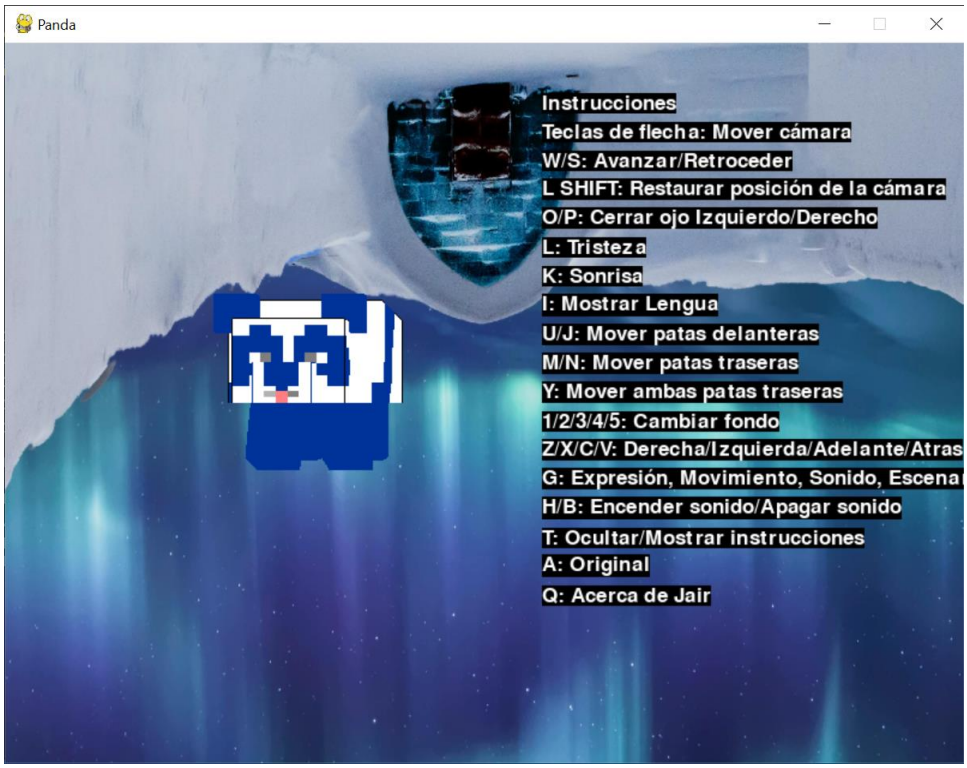

Fondo 5:



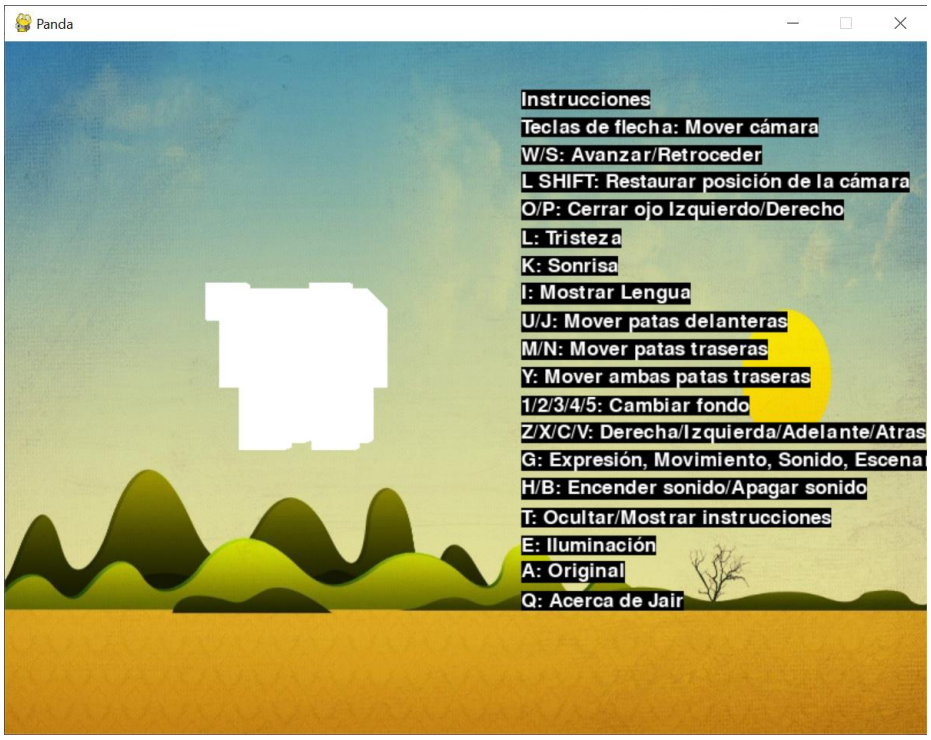
Movimiento hacia enfrente, izquierda, derecha, atrás:



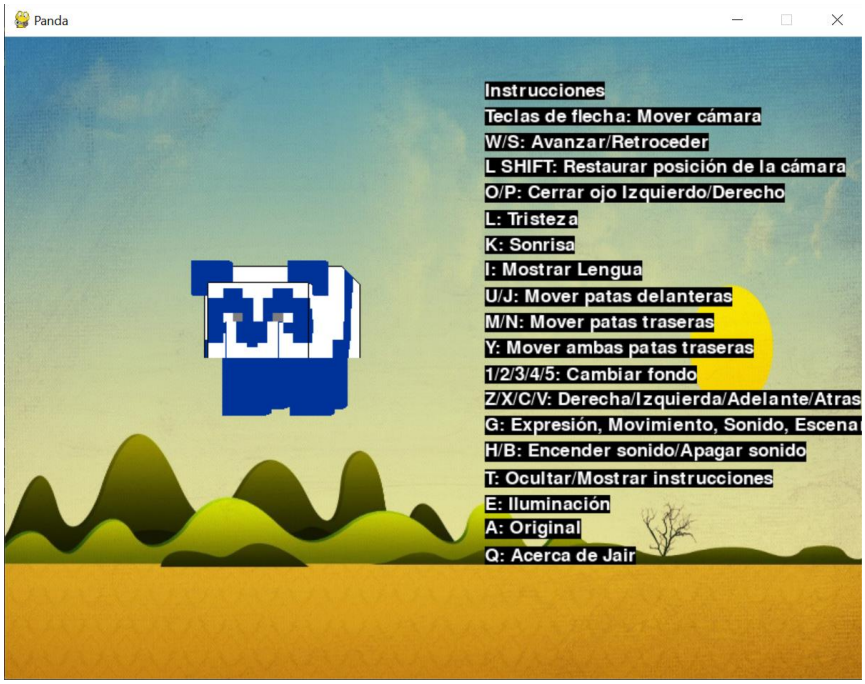

Expresión movimiento



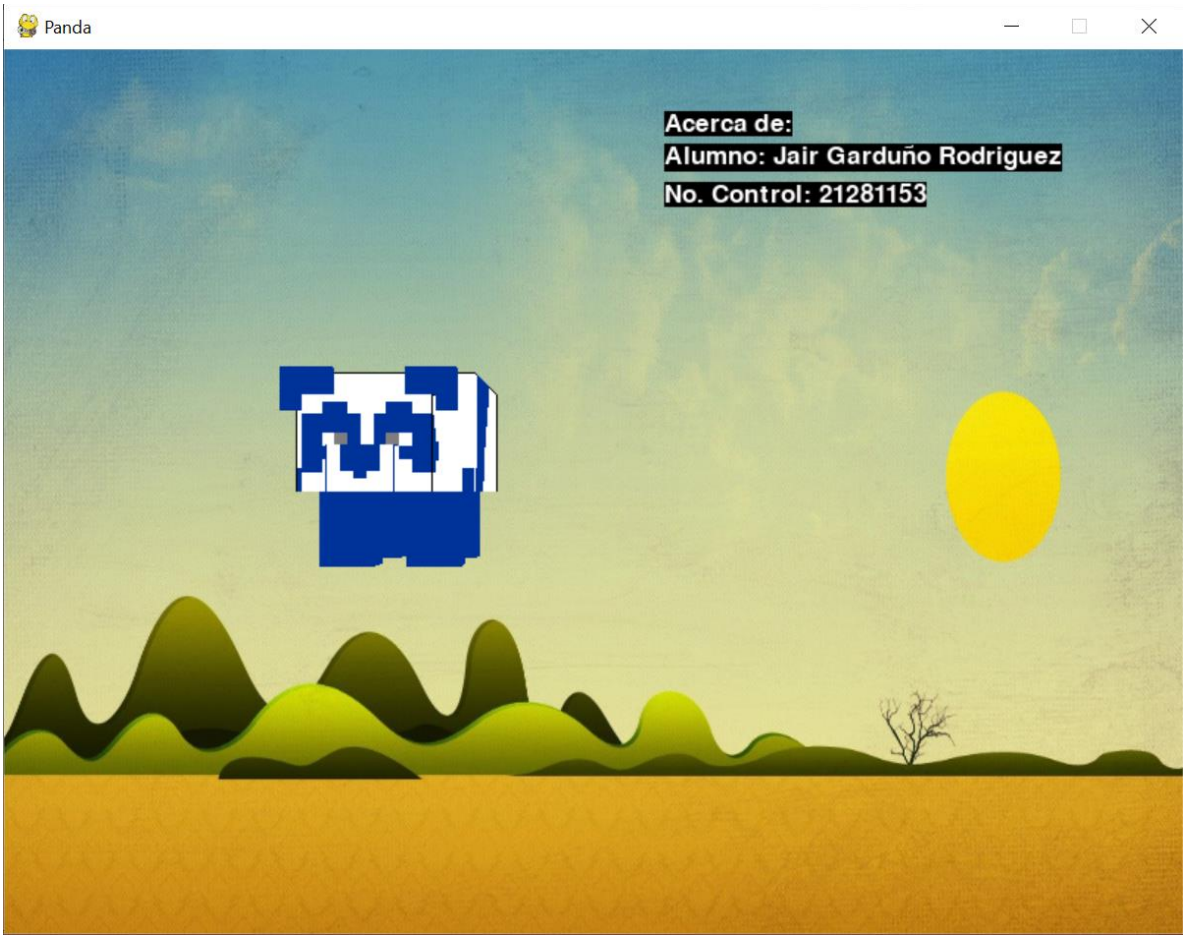
iluminación:




Original:




Acerca de:




## Conclusiones

En conclusión, estas prácticas nos han proporcionado una sólida comprensión de los principios fundamentales y avanzados de la creación y manipulación de objetos en 3D utilizando Pygame. Hemos aprendido a representar objetos tridimensionales, aplicar técnicas de iluminación y materiales, y utilizar diversas técnicas de renderizado para mejorar la calidad visual de nuestros proyectos. Además, hemos experimentado con la interacción del usuario y la creación de entornos interactivos en 3D, lo que nos ha permitido aplicar nuestros conocimientos en la creación de juegos y simulaciones

## Bibliografía


- Batoćanin, V. (Fecha no especificada). Advanced OpenGL in Python with PyGame and PyOpenGL. Stack Abuse.
- Knowivate Developers. (2023, 17 de abril). Python Game Development with Pygame and PyOpenGL. Knowivate.
- Stack Overflow. (2023, 9 de noviembre). Drawing 3D objects in pygame window using OpenGL
- GameDev Academy. Pygame 3D Tutorial - Complete Guide