PROGRAMACIÓN WEB 2

Profesor(a):
Carlo Jose Luis Corrales Delgado

Estudiantes:
Mamani Anahua, Victor Narciso

Repositorio GitHub:
https://github.com/VictorMA18/Lab05-Python

Video:
https://github.com/VictorMA18/Lab05-Python

7 de mayo, 2024

Ejercicios de Python

Para iniciar los ejercicios vamos a primero instalar un paquete de python el cual es

```
pip install pygame
```

Funciones Picture

La clase Picture permite realizar diversas operaciones sobre imágenes representadas como listas de cadenas de caracteres. Incluye métodos para crear espejos verticales (verticalMirror) y horizontales (horizontalMirror), invertir los colores de la imagen (negative), unir imágenes horizontalmente (join) y verticalmente (up), superponer una imagen sobre otra (under), repetir la imagen horizontalmente (horizontalRepeat) y verticalmente (verticalRepeat), y rotar la imagen 90 grados (rotate). Además, tiene métodos para inicializar una imagen (init), comparar dos imágenes (eq), y una función auxiliar para invertir colores (invColor). Cada método devuelve una nueva instancia de Picture con la transformación aplicada, permitiendo una manipulación flexible de las imágenes

Para iniciar con los ejercicios tenemos que primero implementar las Funciones de la clase Picture para su posterior uso con funciones para sus objetos

Funciones Picture

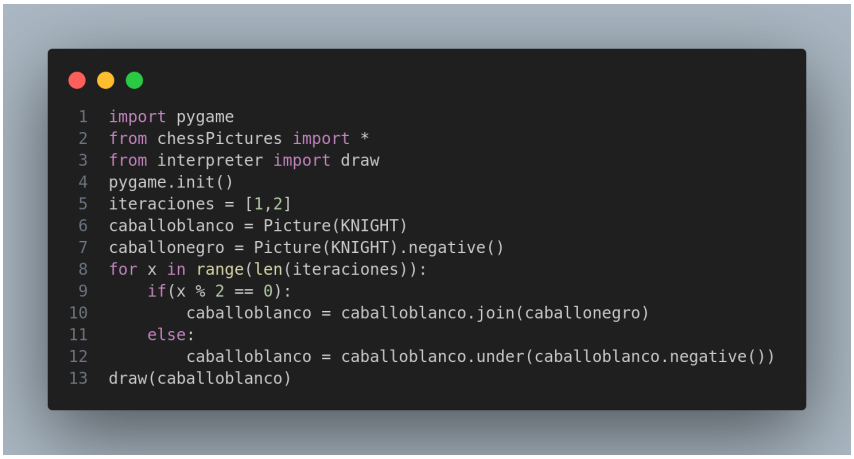
```python
from colors import *
class Picture:
  def __init__(self, img):
    self.img = img

  def __eq__(self, other):
    return self.img == other.img

  def _invColor(self, color):
    if color not in inverter:
      return BLACK
    return inverter[color]

  def verticalMirror(self):
    """ Devuelve el espejo vertical de la imagen """
    vertical = []
    for value in self.img:
      vertical.append(value[::-1])
    return Picture(vertical)

  def horizontalMirror(self):
    nuevo = []
    for x in range(len(self.img)):
      nuevo.append(self.img[(len(self.img) - 1) - x])
    return Picture(nuevo)

  def negative(self):
    nuevo = []
    for x in range(len(self.img)):
      cadena = ""
      for y in range(len(self.img[x])):
        cadena += self._invColor(self.img[x][y])
      nuevo.append(cadena)
    return Picture(nuevo)


  def join(self, p):
    nuevo = []
    for x in range(len(self.img)):     # Iterar sobre los elementos de la imagen actual
        nuevo.append(self.img[x] + p.img[x])
    return Picture(nuevo)

  def up(self, p):
    nuevo = []
    for x in range(len(p.img)):
        nuevo.append(p.img[x])
    for y in range(len(self.img)):
        nuevo.append(self.img[y])
    return Picture(nuevo)

  def under(self, p):
    nuevo = []
    for x in range(len(self.img)):
      cadena = ""
      for y in range(len(self.img[x])):
        if(p.img[x][y] == " "):
          cadena += self.img[x][y]
        else:
          cadena += p.img[x][y]
      nuevo.append(cadena)
    return Picture(nuevo)

  def horizontalRepeat(self, n):
    nuevo = []
    if(n == 1):
      return self
    else:
      for x in range(len(self.img)):
        nuevo.append(self.img[x] * n)
      return Picture(nuevo)

  def verticalRepeat(self, n):
    nuevo = []
    if(n == 1):
      return self
    else:
      for i in range(n):
        for x in range(len(self.img)):
          nuevo.append(self.img[x])
      return Picture(nuevo)

  #Extra: Sólo para realmente viciosos
  def rotate(self):
    nuevo = []
    for x in range(len(self.img)):
      cadena = ""
      for y in range(len(self.img[x])):
        cadena += self.img[-y][x]
      nuevo.append(cadena)
    return Picture(nuevo)
```

Figura 1: Codigo

Primer Ejercicio

En este código de Python, primero inicializo Pygame y luego importo módulos personalizados para manipular imágenes de piezas de ajedrez. Creo una imagen de un caballo blanco y otra de un caballo negro, que es la versión negativa del blanco. Luego, en un bucle que se ejecuta dos veces, combino estas imágenes de diferentes maneras: si el índice es par, uno el caballo blanco con el negro horizontalmente; si es impar, coloco el caballo blanco sobre su versión negativa. Finalmente, dibujo la imagen resultante usando la función draw, creando así una composición gráfica de caballos blancos y negros según las operaciones realizadas en el bucle.

Para esto vamos poner el primer codigo python el cual es:



```python
import pygame
from chessPictures import *
from interpreter import draw
pygame.init()
iteraciones = [1,2]
caballoblanco = Picture(KNIGHT)
caballonegro = Picture(KNIGHT).negative()
for x in range(len(iteraciones)):
    if(x % 2 == 0):
        caballoblanco = caballoblanco.join(caballonegro)
    else:
        caballoblanco = caballoblanco.under(caballoblanco.negative())
draw(caballoblanco)
```

Figura 2: Codigo

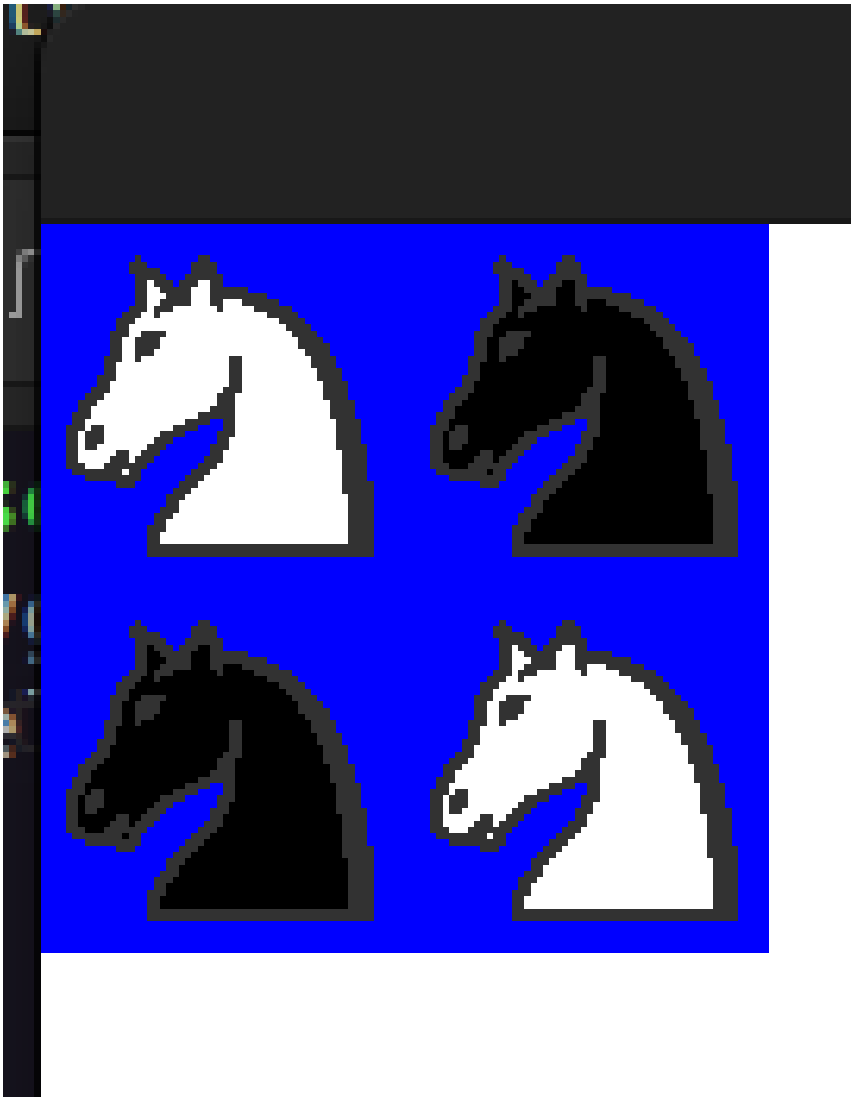


Figura 3: Ejecucion

Segundo Ejercicio

En este código de Python, primero inicializo Pygame e importo módulos personalizados para manipular imágenes de piezas de ajedrez. Creo una imagen de un caballo blanco y otra de un caballo negro, que es la versión negativa del blanco. Luego, en un bucle que se ejecuta dos veces, combino estas imágenes de diferentes maneras: si el índice es par, uno el caballo blanco con el negro horizontalmente; si es impar, coloco el caballo blanco sobre su versión reflejada verticalmente. Finalmente, dibujo la imagen resultante usando la función draw, creando así una composición gráfica de caballos blancos y negros según las operaciones realizadas en el bucle.

Para esto vamos poner el segundo codigo python el cual es:

```python
import pygame
from chessPictures import *
from interpreter import draw
pygame.init()
iteraciones = [1,2]
caballoblanco = Picture(KNIGHT)
caballonegro = Picture(KNIGHT).negative()
for x in range(len(iteraciones)):
    if(x % 2 == 0):
        caballoblanco = caballoblanco.join(caballonegro)
    else:
        caballoblanco = caballoblanco.under(caballoblanco.verticalMirror())
draw(caballoblanco)
```

Figura 4: Codigo

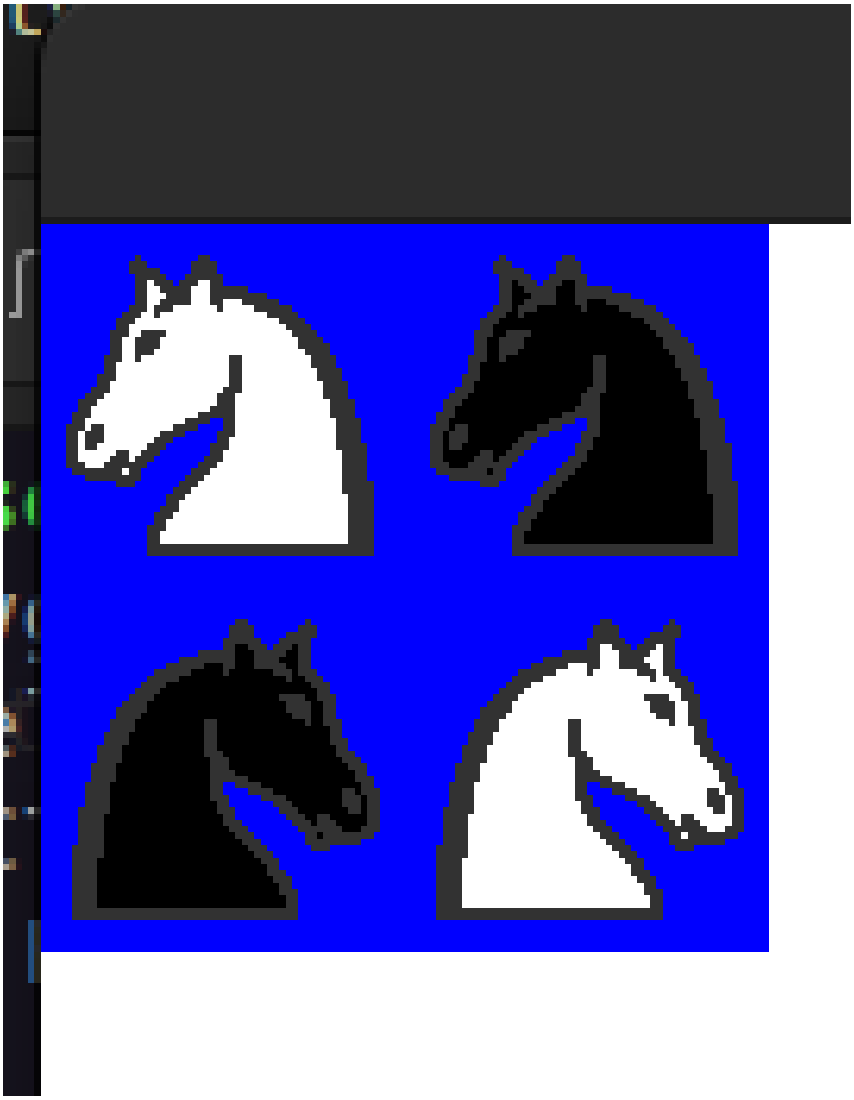


Figura 5: Ejecucion

Tercer Ejercicio

En este código de Python, primero inicializo Pygame e importo módulos personalizados para trabajar con imágenes de piezas de ajedrez. Creo una imagen de una reina blanca y una imagen vacía. Luego, en un bucle que se ejecuta cinco veces, combino estas imágenes de diferentes maneras: en la primera iteración, coloco la reina blanca sobre la imagen vacía; en las iteraciones posteriores a la segunda, uno la imagen vacía con la reina blanca horizontalmente. Finalmente, dibujo la imagen resultante usando la función draw, creando así una composición gráfica de la reina blanca según las operaciones realizadas en el bucle.

Para esto vamos poner el tercer codigo python el cual es:

```python
import pygame
from chessPictures import *
from interpreter import draw
pygame.init()
reynablanca = Picture(QUEEN)
null = Picture("")
for x in range(0,5):
    if(x < 1):
        null = null.up(reynablanca)
    if(x > 1):
        null = null.join(reynablanca)
draw(null)
```

Figura 6: Codigo

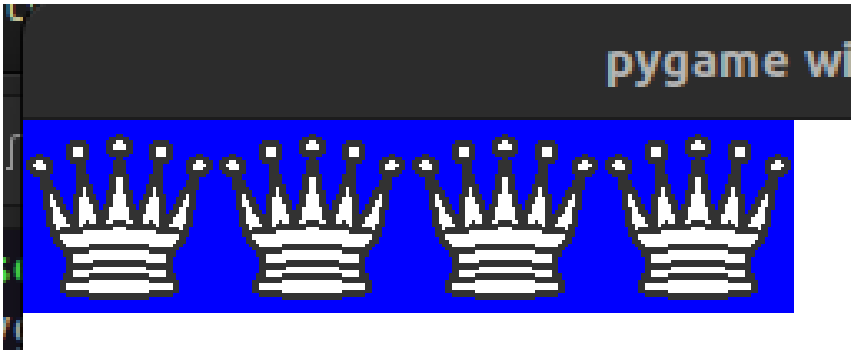


Figura 7: Ejecucion

Cuarto Ejercicio

En este código de Python, primero inicializo Pygame e importo módulos personalizados para trabajar con imágenes de piezas de ajedrez. Creo imágenes para representar una fila de casillas blancas y negras,

así como una imagen vacía. Luego, en un bucle que se ejecuta cuatro veces, combino estas imágenes de diferentes maneras: en la primera iteración, coloco la fila de casillas blancas y negras sobre la imagen vacía; en las iteraciones posteriores, uno la imagen vacía con la fila de casillas blancas y negras horizontalmente. Finalmente, dibujo la imagen resultante usando la función draw, creando así una composición gráfica de casillas blancas y negras según las operaciones realizadas en el bucle.

Para esto vamos poner el cuarto codigo python el cual es:

```python
import pygame
from chessPictures import *
from interpreter import draw
pygame.init()
fila = Picture(None)
null = Picture("")
casilleroblanco = Picture(SQUARE)
casilleronegro = Picture(SQUARE).negative()
for x in range(0,4):
    if(x < 1):
        null = null.up(casilleroblanco.join(casilleronegro))
    if(x >= 1):
        null = null.join(casilleroblanco.join(casilleronegro))
draw(null)
```

Figura 8: Codigo

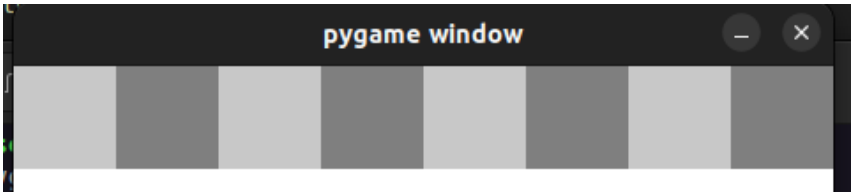


Figura 9: Ejecucion

Quinto Ejercicio

En este código de Python, primero inicializo Pygame e importo módulos personalizados para trabajar con imágenes de piezas de ajedrez. Creo imágenes para representar una fila de casillas blancas y negras, así como una imagen vacía. Luego, en un bucle que se ejecuta cuatro veces, combino estas imágenes de diferentes maneras: en la primera iteración, coloco la fila de casillas blancas y negras (invertidas) sobre la imagen vacía; en las iteraciones posteriores, uno la imagen vacía con la fila de casillas blancas y negras (invertidas) horizontalmente. Finalmente, dibujo la imagen resultante usando la función draw, creando así una composición gráfica de casillas blancas y negras (invertidas) según las operaciones realizadas en el bucle.

Para esto vamos poner el quinto codigo python el cual es:

```python
import pygame
from chessPictures import *
from interpreter import draw
pygame.init()
fila = Picture(None)
null = Picture("")
casilleroblanco = Picture(SQUARE)
casilleronegro = Picture(SQUARE).negative()
for x in range(0,4):
    if(x < 1):
        null = null.up(casilleroblanco.join(casilleronegro).negative())
    if(x >= 1):
        null = null.join(casilleroblanco.join(casilleronegro).negative())
draw(null)
```

Figura 10: Codigo

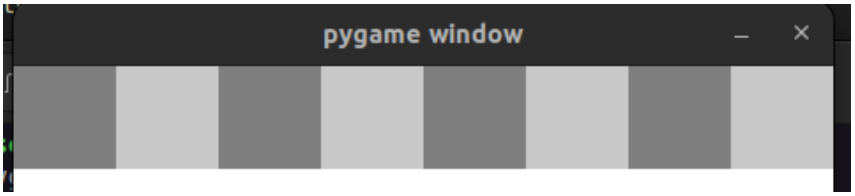


Figura 11: Ejecucion

Sexto Ejercicio

En este código de Python, primero se inicializa Pygame y se importan módulos personalizados para trabajar con imágenes de piezas de ajedrez. Se crea una imagen vacía y dos imágenes para representar casillas blancas y negras, esta última con un efecto negativo aplicado. Luego, en un bucle que se ejecuta cuatro veces, se combinan estas imágenes para formar una fila de casillas blancas y negras. Después, en otro bucle que se ejecuta dos veces, se aplican operaciones a la imagen resultante: en las iteraciones pares, se coloca la imagen negativa sobre sí misma; en las impares, se coloca la imagen original sobre sí misma. Finalmente, se dibuja la imagen resultante utilizando la función draw, creando así una composición gráfica de casillas blancas y negras con efectos según las operaciones realizadas en los bucles.

Para esto vamos poner el sexto codigo python el cual es:

```python
import pygame
from chessPictures import *
from interpreter import draw
pygame.init()
null = Picture("")
casilleroblanco = Picture(SQUARE)
casilleronegro = Picture(SQUARE).negative()
for x in range(0,4):
    if(x < 1):
        null = null.up(casilleroblanco.join(casilleronegro))
    if(x >= 1):
        null = null.join(casilleroblanco.join(casilleronegro))
for i in range(0,2):
    if(i % 2 == 0):
        null = null.up(null.negative())
    else:
        null = null.up(null)
draw(null)
```

Figura 12: Codigo

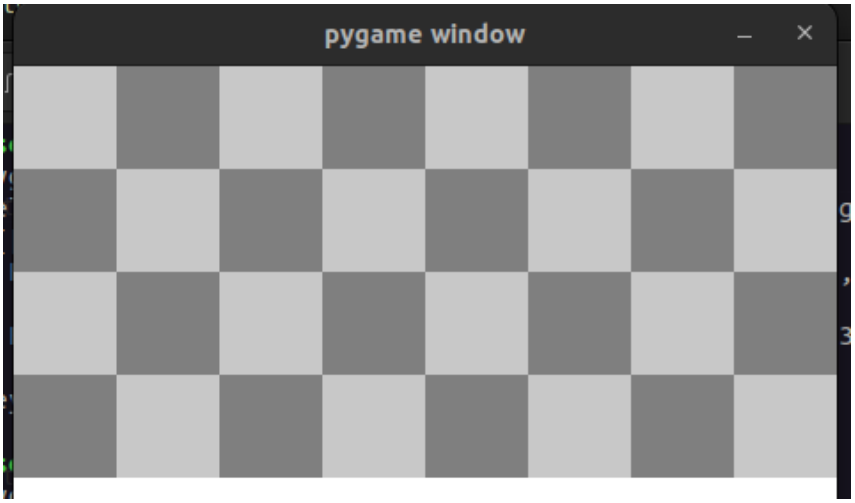
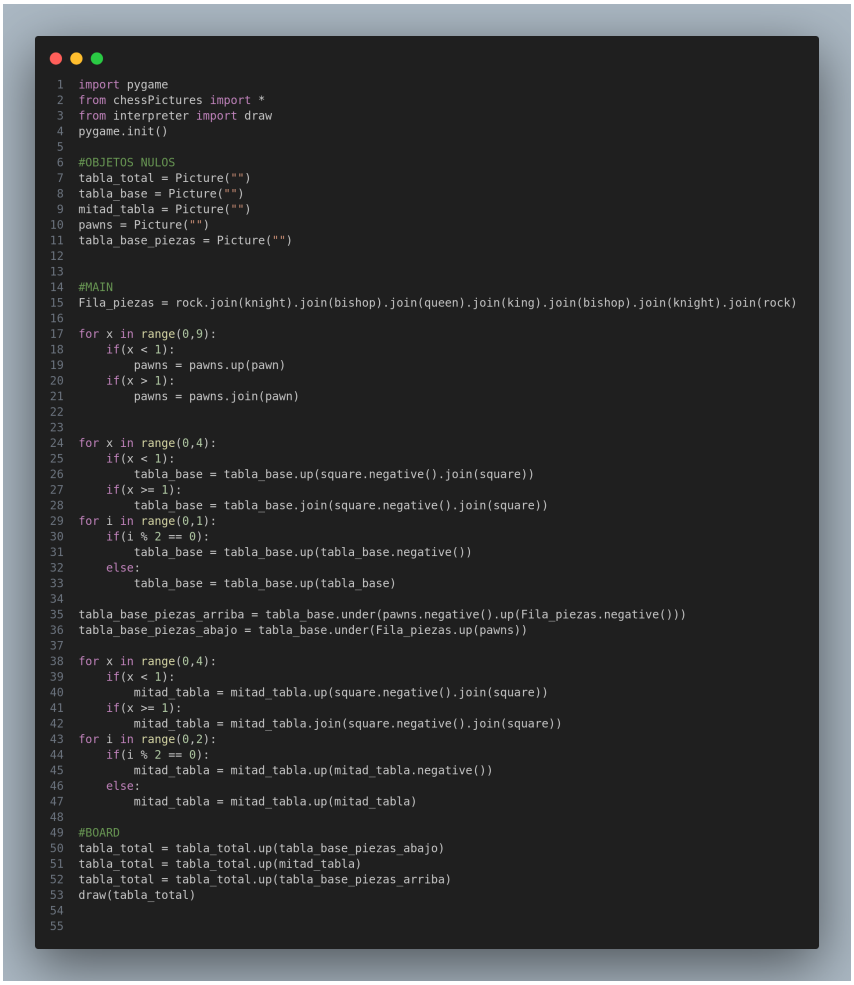


Figura 13: Ejecucion

Septimo Ejercicio

Este código en Python utiliza Pygame y módulos personalizados para construir un tablero de ajedrez visualmente representado con las piezas correctamente posicionadas. Comienza inicializando Pygame y estableciendo los objetos necesarios para el tablero y las piezas. Luego, mediante bucles y operaciones de unión y superposición, se construyen las diferentes partes del tablero y se colocan las piezas según las reglas del ajedrez. Las piezas se posicionan en la base del tablero, con una fila de peones en la parte inferior y las piezas principales en la parte superior. El resultado es una representación gráfica detallada de un tablero de ajedrez con todas las piezas colocadas correctamente, listo para ser visualizado con la función draw.

Para esto vamos poner el septimo codigo python el cual es:



```python
import pygame
from chessPictures import *
from interpreter import draw
pygame.init()

#OBJETOS NULOS
tabla_total = Picture("")
tabla_base = Picture("")
mitad_tabla = Picture("")
pawns = Picture("")
tabla_base_piezas = Picture("")


#MAIN
Fila_piezas = rock.join(knight).join(bishop).join(queen).join(king).join(bishop).join(knight).join(rock)

for x in range(0,9):
    if(x < 1):
        pawns = pawns.up(pawn)
    if(x > 1):
        pawns = pawns.join(pawn)


for x in range(0,4):
    if(x < 1):
        tabla_base = tabla_base.up(square.negative().join(square))
    if(x >= 1):
        tabla_base = tabla_base.join(square.negative().join(square))
for i in range(0,1):
    if(i % 2 == 0):
        tabla_base = tabla_base.up(tabla_base.negative())
    else:
        tabla_base = tabla_base.up(tabla_base)

tabla_base_piezas_arriba = tabla_base.under(pawns.negative().up(Fila_piezas.negative()))
tabla_base_piezas_abajo = tabla_base.under(Fila_piezas.up(pawns))

for x in range(0,4):
    if(x < 1):
        mitad_tabla = mitad_tabla.up(square.negative().join(square))
    if(x >= 1):
        mitad_tabla = mitad_tabla.join(square.negative().join(square))
for i in range(0,2):
    if(i % 2 == 0):
        mitad_tabla = mitad_tabla.up(mitad_tabla.negative())
    else:
        mitad_tabla = mitad_tabla.up(mitad_tabla)

#BOARD
tabla_total = tabla_total.up(tabla_base_piezas_abajo)
tabla_total = tabla_total.up(mitad_tabla)
tabla_total = tabla_total.up(tabla_base_piezas_arriba)
draw(tabla_total)


```

Figura 14: Codigo

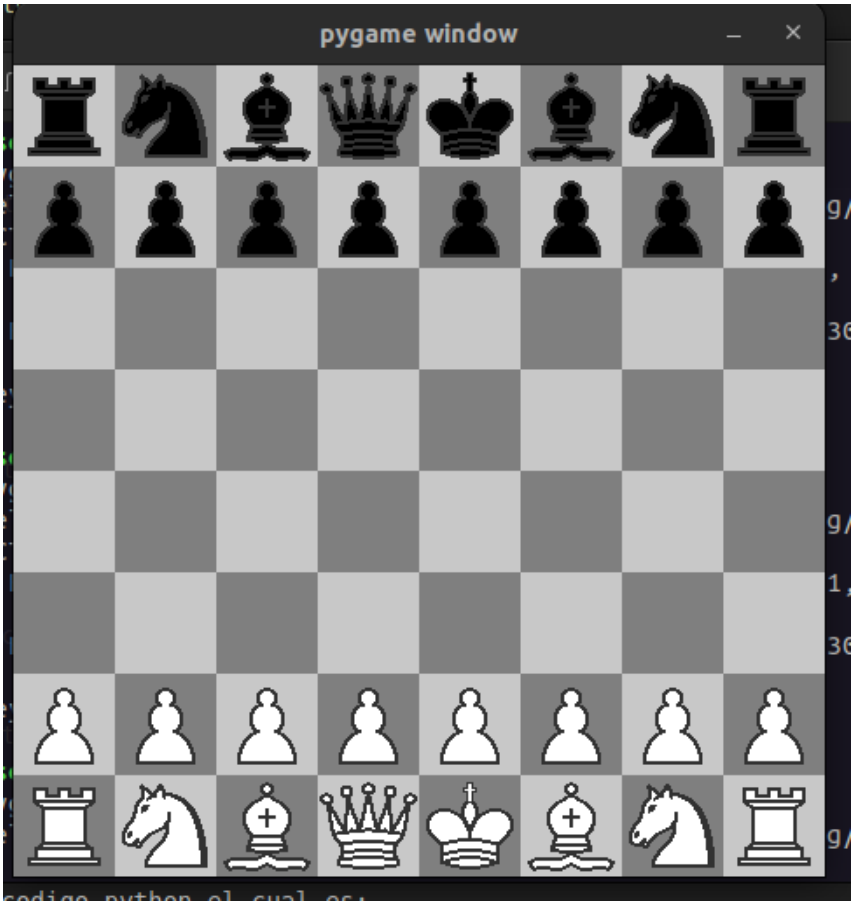


Figura 15: Ejecucion

URL de video de explicación: `https://drive.google.com/file/d/1FKJIwx4yqkJdi3IXroJYgPwxZLg_5Hz0/view?usp=sharing`

URL de repositorio de GitHub: `https://github.com/VictorMA18/Lab05-Python`