

Adrian Malaver Machado 20231020068

Juan Camilo Lopez Walteros 20231020194

JUEGO SNAKE

Este juego es muy popular desde la época de los dumb phones (teléfonos tontos), pues Nokia lo incluía en sus dispositivos en aquellos tiempos, un juego sencillo pero entretenido que consistía en dirigir una serpiente evitando chocarse, aunque contrario a lo que muchos piensan este juego no surgió de Nokia, sino que nace en el Arcade Blockade y fue lanzado por primera vez en 1976 por Gremlin Industries. El juego llego a los hogares cuando Atari lanzo una versión no oficial para consolas y poco después en 1978, llegaría Worm a los ordenadores, que sería la versión para ordenador del posterior Snake.

# OBJETIVO:

Con la creación del juego Snake en Python busca implementar y comprender de una forma didáctica e investigativa el funcionamiento, los beneficios y la ventaja que nos ofrece el paradigma de la programación orientada a objetos para la eficacia y eficiencia de los programas y en este caso el juego de la serpiente.

-lograr entender los conceptos de herencia, polimorfismo, clases y objetos en Python, y su uso para la resolución de problemas o creación de programas con su uso.

-analizar como la programación orientada a objetos permite una mayor comprensión del código y ahorro del mismo, porque se asemeja más a la realidad.

-comprender en código de Python, como es la sintaxis de la programación orientada a objetos en este lenguaje en particular.

# EN QUE CONSISTE EL JUEGO DE LA SERPIENTE (SNAKE):

Este juego ha tenido muchas versiones a lo largo de la historia, desde juegos multijugador, hasta laberintos de serpiente, pero la versión clásica y la que lo hizo famoso es bastante sencilla de comprender. Hay una serpiente en un cuarto la cual empieza a comer manzanas y con ello a aumentar su tamaño y la puntuación del usuario, pero debe evitar a toda costa chocar contra alguno de los límites del cuarto u contra si misma porque de esta forma el usuario perderá, este juego tiene la característica de que a medida que la serpiente aumenta su tamaño, la dificultad de dirigirla sin chocar es mayor, por ello este juego tiene un final, un momento en que el usuario perderá debido a que el tamaño de la serpiente será igual al área del cuarto y de esta forma la serpiente dirigida por el usuario se estrellara contra una pared del cuarto, o continuara enrollándose sobre si misma hasta chocar contra ella misma y perder.

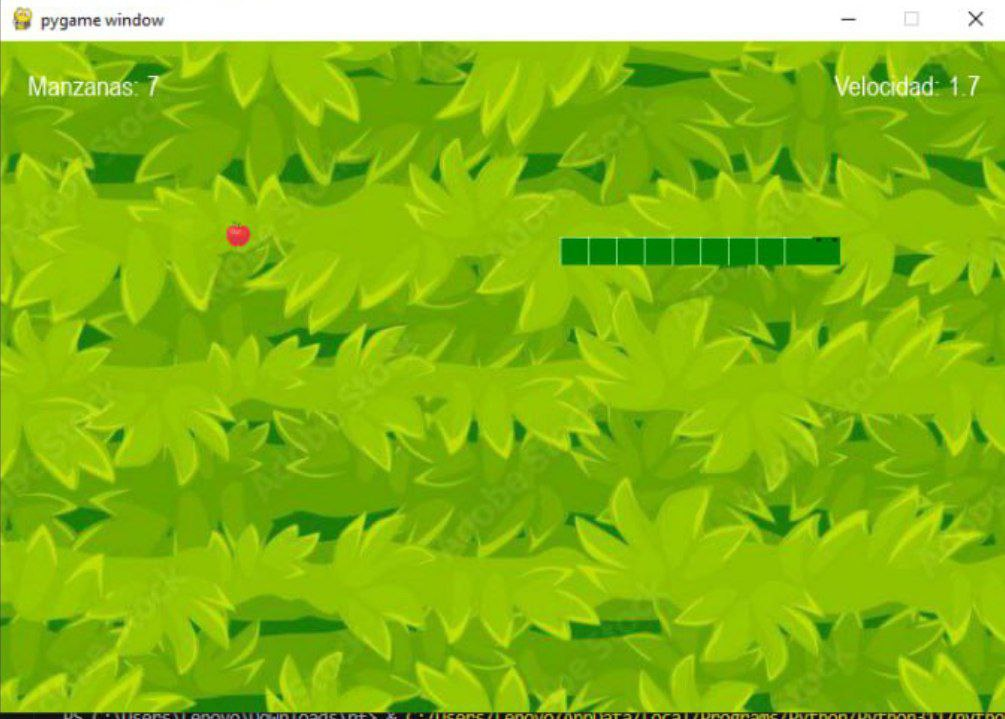


# COMO SERA NUESTRA VERSION DEL JUEGO DE LA SERPIENTE:

Nuestro juego de la serpiente cambiara en cuanto a dos cosas, cada vez que la serpiente se come un determinado numero de manzanas, la serpiente aumentara su velocidad poco a poco hasta que alcance un límite de velocidad, también cuando la serpiente alcance cierta velocidad, cada manzana que aparezca en el mapa estará moviéndose de lado a lado para que así sea más difícil para el usuario alcanzarlas con la cabeza de la serpiente, porque tanto la serpiente como las manzanas irán aumentando la velocidad a medida que la serpiente se hace más grande, así como también se mostrara el puntaje que lleva el usuario hasta el momento y la velocidad que tiene la serpiente, además habran dos tipos de posiones, una para disminuir la velocidad, cada vez que la velocidad llegue a un punto limite establecido y también una posion de tamaño que aparecerá cuando se llegue a las 20 manzanas comidas y se reducirá en 11 bloques el tamaño de la serpiente.

Nuestro juego tendrá las siguientes características:

* La serpiente aumentara su velocidad a medida que se coma un determinado número de manzanas.
* Las manzanas empezaran a moverse lateralmente cuando la serpiente alcance una velocidad de 1.4.
* Las manzanas aumentaran su velocidad de movimiento lateral a medida que la serpiente coma manzanas.
* Se mostrará en la ventana el numero de manzanas comidas, la velocidad de la serpiente y la velocidad de las manzanas.
* Para que el juego no se vuelva frustrante, cada cierta cantidad de manzanas comidas por la serpiente, aparecerá de forma aleatoria en el mapa unas pociones.
* Habrá una posion para reducir la velocidad y esto se mostrara en el marcador.
* Cada cierta velocidad alcanzada aparecerá una posion morada que disminuira la velocidad actual de la serpiente
* A las 20 manzanas comidas aparecerá una posion azul que reducirá el tamaño de la serpiente quitándole 11 bloques de su cuerpo



**CODIGO:**

1. **principal.py**:
2. # principal.py
3. import pygame
4. import sys
5. from serpiente import Serpiente
6. from manzana import Manzana
7. from posion\_velocidad import PosionVelocidad
8. from posion\_tamaño import PosionTamano
9. pygame.init()
10. ANCHO = 720
11. ALTO = 480
12. FONDO = pygame.image.load("fondo.png")
13. WIN = pygame.display.set\_mode((ANCHO, ALTO))
14. SCORE\_FONT = pygame.font.SysFont("Arial", 20)
15. sonido\_coin1 = pygame.mixer.Sound("coin1.wav")
16. sonido\_coin2 = pygame.mixer.Sound("coin2.wav")
17. def main():
18. serpiente = Serpiente()
19. manzana = Manzana(ANCHO, ALTO)
20. posion\_velocidad = PosionVelocidad(ANCHO, ALTO, serpiente.velocidad)
21. posion\_tamano = PosionTamano(ANCHO, ALTO)
22. manzanas\_comidas = 0
23. velocidad = 1
24. fps = pygame.time.Clock()
25. while True:
26. fps.tick(10 \* velocidad)
27. for event in pygame.event.get():
28. if event.type == pygame.QUIT:
29. mostrar\_mensaje("Fin del juego. Manzanas comidas: {}".format(manzanas\_comidas))
30. pygame.quit()
31. sys.exit()
32. if event.type == pygame.KEYDOWN and serpiente.direccion[1] != 20:
33. if event.key == pygame.K\_UP:
34. serpiente.mover\_arriba()
35. if event.type == pygame.KEYDOWN and serpiente.direccion[1] != -20:
36. if event.key == pygame.K\_DOWN:
37. serpiente.mover\_abajo()
38. if event.type == pygame.KEYDOWN and serpiente.direccion[0] != -20:
39. if event.key == pygame.K\_RIGHT:
40. serpiente.mover\_derecha()
41. if event.type == pygame.KEYDOWN and serpiente.direccion[0] != 20:
42. if event.key == pygame.K\_LEFT:
43. serpiente.mover\_izquierda()
44. WIN.blit(FONDO, (0, 0))
45. serpiente.dibujar(WIN, (255, 255, 255))
46. if posion\_tamano.verificar\_colision(serpiente.cuerpo[0], manzanas\_comidas):
47. posion\_tamano.aparecida = True
48. serpiente.disminuir\_tamano()
49. sonido\_coin2.play()
50. if posion\_tamano.aparecer(manzanas\_comidas):
51. WIN.blit(posion\_tamano.imagen, posion\_tamano.pos)
52. if velocidad >= 1.4:
53. manzana.mover()
54. WIN.blit(manzana.imagen, manzana.pos)
55. if velocidad >= 2.7:
56. WIN.blit(posion\_velocidad.imagen, posion\_velocidad.pos)
57. if posion\_velocidad.verificar\_colision(serpiente):
58. posion\_velocidad.aparecer(serpiente.velocidad)
59. velocidad -= 0.5
60. sonido\_coin2.play()
61. serpiente.mover()
62. if manzana.verificar\_colision(serpiente):
63. manzanas\_comidas += 1
64. velocidad += 0.1
65. manzana.aumentar\_velocidad()
66. sonido\_coin1.play()
67. if serpiente.morir(ANCHO, ALTO):
68. mostrar\_mensaje("Manzanas comidas: {}".format(manzanas\_comidas))
69. pygame.quit()
70. sys.exit()
71. texto\_manzanas = SCORE\_FONT.render("Manzanas: {}".format(manzanas\_comidas), 1, (255, 255, 255))
72. WIN.blit(texto\_manzanas, (20, 20))
73. texto\_velocidad = SCORE\_FONT.render("Velocidad: {:.1f}".format(velocidad), 1, (255, 255, 255))
74. WIN.blit(texto\_velocidad, (ANCHO - texto\_velocidad.get\_width() - 20, 20))
75. pygame.display.update()
76. def mostrar\_mensaje(mensaje):
77. pygame.font.init()
78. font = pygame.font.SysFont("Arial", 36)
79. text = font.render(mensaje, True, (255, 255, 255))
80. WIN.blit(text, (ANCHO // 2 - text.get\_width() // 2, ALTO // 2 - text.get\_height() // 2))
81. pygame.display.update()
82. pygame.time.delay(3000)
83. if \_\_name\_\_ == "\_\_main\_\_":
84. main()

-Contiene el bucle principal del juego.

-Importa y utiliza las clases anteriores (Serpiente, Manzana, PosionTamano) junto

con la clase PosionVelocidad.

-Maneja eventos de teclado para controlar la dirección de la serpiente y actualiza

la pantalla en cada ciclo del bucle.

- Muestra mensajes de fin de juego y actualiza la puntuación y velocidad.

**2. posion\_tamaño.py**

# posion\_tamaño.py

import pygame

import random

class PosionTamano:

    def \_\_init\_\_(self, ancho, alto):

        self.ancho = ancho

        self.alto = alto

        self.generar()

        self.imagen = pygame.image.load("posion\_tamaño.png")

        self.imagen = pygame.transform.scale(self.imagen, (20, 20))

        self.aparecida = False  # Agregamos un atributo 'aparecida'

    def generar(self):

        x = random.randrange(0, self.ancho, 20)

        y = random.randrange(0, self.alto, 20)

        self.pos = pygame.Rect(x, y, 20, 20)

    def aparecer(self, manzanas\_comidas):

        casos = {20: True, 30: True, 40: True}

        return casos.get(manzanas\_comidas, False) and not self.aparecida

    def desaparecer(self):

        self.aparecida = False

        self.generar()  # Generamos una nueva posición al desaparecer

    def verificar\_colision(self, cabeza\_serpiente, manzanas\_comidas):

        if cabeza\_serpiente.colliderect(self.pos) and self.aparecer(manzanas\_comidas):

            return True

        return False

- Define la clase PosionTamano, que representa las pócimas de tamaño que la

serpiente puede consumir para disminuir su longitud.

- Contiene métodos para generar, hacer aparecer/desaparecer, y verificar la colisión

de la posion con la cabeza de la serpiente.

1. **posion\_velocidad.py**:
2. #posion\_velocidad.py
3. import pygame
4. import random
5. class PosionVelocidad:
6. def \_\_init\_\_(self, ancho, alto, velocidad\_serpiente):
7. self.ancho = ancho
8. self.alto = alto
9. self.velocidad\_serpiente = velocidad\_serpiente
10. self.generar()
11. self.imagen = pygame.image.load("posion\_velocidad.png")  # Carga la imagen de la poción de velocidad
12. self.imagen = pygame.transform.scale(self.imagen, (20, 20))  # Escala la imagen a 20x20
13. def generar(self):
14. x = random.randrange(0, self.ancho, 20)
15. y = random.randrange(0, self.alto, 20)
16. self.pos = pygame.Rect(x, y, 20, 20)
17. def verificar\_colision(self, serpiente):
18. if serpiente.cuerpo[0].colliderect(self.pos):
19. self.generar()
20. serpiente.velocidad\_aumentar = True
21. return True
22. return False
23. def aparecer(self, velocidad\_serpiente):
24. self.velocidad\_serpiente = velocidad\_serpiente
25. self.generar()

-Define la clase PosionVelocidad, que representa las pócimas de velocidad que la

serpiente puede consumir para aumentar temporalmente su velocidad.

-Contiene métodos para generar la posición de la pócima y verificar la colisión con

la serpiente.

1. **manzana.py**:
2. # manzana.py
3. import pygame
4. import random
5. class Manzana:
6. def \_\_init\_\_(self, ancho, alto):
7. self.ancho = ancho
8. self.alto = alto
9. self.velocidad = 0  # Inicialmente, la velocidad es 0
10. self.velocidad\_inicial = 1
11. self.generar()
12. self.imagen = pygame.image.load("manzana.png")  # Carga la imagen de la manzana
13. self.imagen = pygame.transform.scale(self.imagen, (20, 20))  # Escala la imagen a 20x20
14. def generar(self):
15. x = random.randrange(0, self.ancho, 20)
16. y = random.randrange(0, self.alto, 20)
17. self.pos = pygame.Rect(x, y, 20, 20)
18. def mover(self):
19. self.pos.y += self.velocidad
20. # Si la manzana sale de la pantalla, la generamos de nuevo
21. if self.pos.y > self.alto:
22. self.generar()
23. def aumentar\_velocidad(self):
24. self.velocidad += 0.1
25. def verificar\_colision(self, serpiente):
26. if serpiente.cuerpo[0].colliderect(self.pos):
27. self.generar()
28. serpiente.agregar = True
29. if self.velocidad > 0:  # Solo aumentar la velocidad si ya se mueven las manzanas
30. self.aumentar\_velocidad()
31. return True
32. return False

- Define la clase Manzana, que representa las manzanas que la serpiente puede

comer para aumentar su puntaje y velocidad.

-Contiene métodos para generar la posición de la manzana, moverla, aumentar la

velocidad, y verificar la colisión con la serpiente.

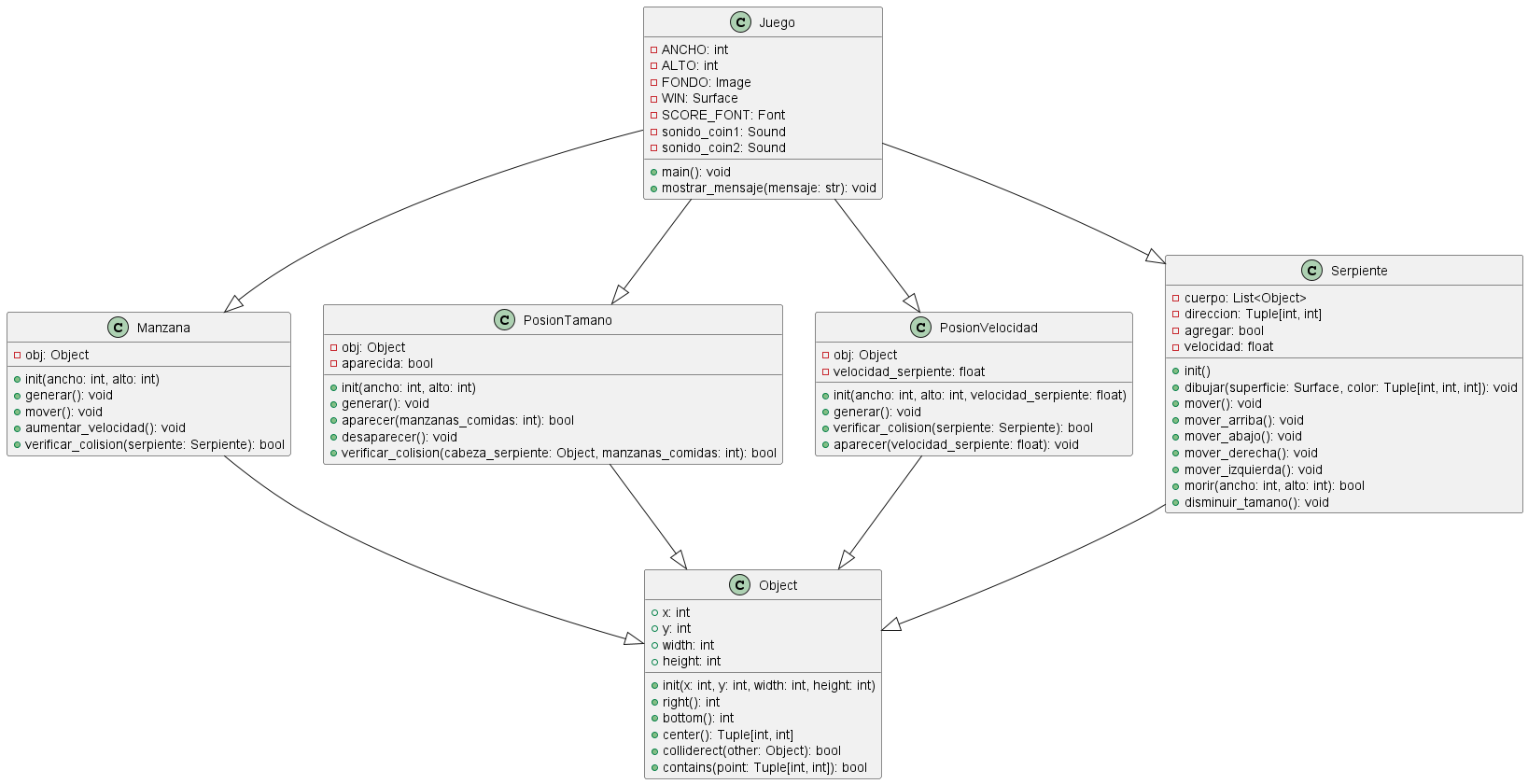
1. **serpiente.py**:
2. # serpiente.py
3. import pygame
4. class Serpiente:
5. def \_\_init\_\_(self):
6. self.cuerpo = [pygame.Rect(320, 240, 20, 20), pygame.Rect(320, 250, 20, 20), pygame.Rect(320, 260, 20, 20)]
7. self.direccion = (0, -20)
8. self.agregar = False
9. self.velocidad = 1
10. self.image\_cabeza = pygame.image.load("cabeza.png")
11. self.image\_cabeza = pygame.transform.scale(self.image\_cabeza, (20, 20))
12. self.image\_cuerpo = pygame.image.load("cuerpo.png")
13. self.image\_cuerpo = pygame.transform.scale(self.image\_cuerpo, (20, 20))
14. def dibujar(self, superficie, color):
15. superficie.blit(self.image\_cabeza, self.cuerpo[0].topleft)
16. for bloque in self.cuerpo[1:]:
17. superficie.blit(self.image\_cuerpo, bloque.topleft)
18. def mover(self):
19. if self.agregar:
20. nuevo\_bloque = pygame.Rect(self.cuerpo[0].x + self.direccion[0], self.cuerpo[0].y + self.direccion[1], 20, 20)
21. self.cuerpo.insert(0, nuevo\_bloque)
22. self.agregar = False
23. else:
24. for i in range(len(self.cuerpo) - 1, 0, -1):
25. self.cuerpo[i].x = self.cuerpo[i - 1].x
26. self.cuerpo[i].y = self.cuerpo[i - 1].y
27. self.cuerpo[0].x += self.direccion[0] \* self.velocidad
28. self.cuerpo[0].y += self.direccion[1] \* self.velocidad
29. def mover\_arriba(self):
30. self.direccion = (0, -20)
31. def mover\_abajo(self):
32. self.direccion = (0, 20)
33. def mover\_derecha(self):
34. self.direccion = (20, 0)
35. def mover\_izquierda(self):
36. self.direccion = (-20, 0)
37. def morir(self, ancho, alto):
38. if (
39. self.cuerpo[0].x >= ancho
40. or self.cuerpo[0].y >= alto
41. or self.cuerpo[0].x <= -20
42. or self.cuerpo[0].y <= -20
43. ):
44. return True
45. for i in self.cuerpo[1:]:
46. if self.cuerpo[0].colliderect(i):
47. return True
48. return False
50. def disminuir\_tamano(self):
51. if len(self.cuerpo) > 10:
52. for \_ in range(10):
53. self.cuerpo.pop()

-Define la clase Serpiente, que representa la serpiente controlada por el jugador.

-Contiene métodos para dibujar la serpiente, moverla en diferentes direcciones,

verificar colisiones, morir en caso de colisiones especificas, y disminuir su

tamaño.



El diagrama de clases refleja la estructura y la relación entre las clases en un juego de serpiente implementado en Python con la biblioteca Pygame. La clase base Object encapsula las propiedades y métodos comunes a objetos bidimensionales, proporcionando funcionalidades como la detección de colisiones y operaciones geométricas básicas. Las clases derivadas, como Manzana, PosionTamano, PosionVelocidad, y Serpiente, heredan de Object y añaden funcionalidades específicas del juego. La clase Juego coordina la lógica principal del juego, manejando eventos, actualizando elementos y representando la interfaz gráfica. Este diseño permite una implementación modular y extensible, facilitando la incorporación de nuevas características o modificaciones en el juego.

**Conclusiones**

En este juego, la programación orientada a objetos (POO) se utiliza de manera efectiva para organizar y estructurar el código. Cada componente del juego (serpiente, manzana, pócima de tamaño y velocidad) se modela como una clase independiente, lo que facilita la gestión, extensión y modificación del código. La encapsulación permite que cada clase tenga su propio estado y comportamiento, reduciendo la complejidad y mejorando la legibilidad.

La POO también facilita la reutilización del código, ya que las clases se pueden utilizar en otros proyectos o ampliar con relativa facilidad. La jerarquía de clases refleja la relación lógica entre los elementos del juego, proporcionando una estructura clara y comprensible.

Además, el uso de clases y objetos permite una implementación más modular y mantenible. Cada clase tiene su propia responsabilidad específica, lo que simplifica el mantenimiento y la solución de problemas. La interacción entre objetos se gestiona de manera cohesiva a través de métodos y atributos, promoviendo un diseño más eficiente y estructurado.

­­­

En resumen, la programación orientada a objetos en este juego ofrece ventajas significativas en términos de organización, reutilización y mantenibilidad del código, contribuyendo a un desarrollo más estructurado y manejable.