

KEVIN ESTUARDO PALACIOS
QUIÑONEZ

MANUAL TECNICO

201902278



Indice

I. Introducción

II. Objetivos

III. Dirigido

IV. Especificación Técnica

1. Requisitos de Hardware

2. Requisitos de Software

V. Lógica del Programa

VI. Métodos Utilizados

I. Introducción

El programa pacman fue diseñado con el fin de ayudar a los estudiantes mediante la creación de un juego, que es realizado mediante listas.

II. Objetivos

El objetivo de este Manual Técnico, es poder orientar a los estudiantes de ingeniería en la creación de Objetos, para poder facilitar y simplificar la creación de Métodos.

III. Dirigido

Este Manual Técnico va dirigido a todos aquellos programadores que esten interesado en el lenguaje de Python, ya que este lenguaje es muy interesante y tiene un entorno muy agradable

También fue hecho con el fin de poder medir nuestro conocimiento mediante el % que creamos de la práctica 3.

IV. Especificación Técnica

Requisitos de Hardware

P

- Computadora de Escritorio o Portatil.
- Mínimo 8GB de Memoria RAM.
- 10 GB Disponibles del Disco Duro.
- Procesador Intel Core i3 o superior.
- Procesador a 64 bits.
- Pantalla con Resolución Gráfica de 1024*768 píxeles

Requisitos de Software


- Tener Instalado Windows 7 o Superior.
- Tener Instalado Visual Studio Code.
- Tener Instaladas las extensiones de Python: Polacode, Prettier- Code formatter, Pylance, Python.

V. Lógica del Programa

Este es el menú con sus respectivos métodos a llamar.

```
#menu del juego juego instanciado con jugador
def option(jugador):
    while True:
        print("_____")
        print("|PACMAN- 🍒 -IPC 1- 🍒 -2022|")
        print("_____")
        print("|1.      INICIAR JUEGO|")
        print("|2.   TABLA POSICIONES|")
        print("|3.           SALIR|")
        print("_____")
        opcion = int(input("INGRESE UNA OPCION ... "))
        if opcion ==1:
            juego(jugador)#llamando al metodo del juego
        elif opcion ==2:
            Position()#llamo las posiciones
        elif opcion ==3:
            break#se rompe el while
```

Se crea la clase Alimento con sus objetos a utilizar



```
class Alimento():
    #Se crea el encapsulamiento
    def __init__(self, posx, posy):
        self.posx = posx
        self.posy = posy
        self.isEat = False
    def getPosX(self):
        return self.posx
    def getPosY(self):
        return self.posy
    def isComido(self):
        return self.isEat
    def setPosY(self, _posy):
        self.posy = _posy
    def setPosX(self, _posx):
        self.posx = _posx
    def setEat(self):
        self.isEat = True
```


Se crea la clase Jugador con sus objetos a utilizar

[illegible]

For que utiliza comidas y posición del jugador, retornara el tablero



```
#Arreglo utilizado Para mostrar al jugador y comida
for comida in lista_comida:
    if not comida.isComido():
        tablero[comida.getPosX()][comida.getPosY()] = "@"
tablero[jugador.getPosX()][jugador.getPosY()] = "C"
return tablero
```

Método utilizado para Crear Comida



```
def CrearComidas(lista_comida, comidas: int):
    indice = 0
    while indice < comidas:
#mientras indice sea menor a comida se generan las comidas
    random

        #13 posiciones de las comidas
        posxcg = random.randint(0,12)
        posycg = random.randint(0,12)
        estaOcupado = False
        for comidas_busqueda in lista_comida:
            if comidas_busqueda.getPosX() == posxcg and
comidas_busqueda.getPosY() == posycg:
                estaOcupado = True
        if not estaOcupado:
            comida_creada = Alimento(posxcg, posycg)
            lista_comida.append(comida_creada)
            indice = indice + 1
```


Método utilizado para verificar la siguiente posición tiene comida

```
def siguienteHayComida(jugador, lista_comida):  
    for comida in lista_comida:  
        if jugador.getPosX() == comida.getPosX() and jugador  
.getPosY() == comida.getPosY():  
            comida.setEat()  
            jugador.addPuntos()  
            return True  
    return False
```

Método utilizado para verificar si aun hay comidas

```
def aunHayComidas(lista_comiditas):  
    for comida in lista_comiditas:  
        if not comida.isComido():  
            return False  
    return True
```

Métodos para realizar movimientos en el tablero



```
def moverArriba(jugador, lista_comiditas):
    posX= jugador.getPosX()-1#filas
    posy= jugador.getPosY()#columnas
    if posX >= 0:
        jugador.setPosX(int(posx))
        jugador.setPosY(int(posy))
        movimiento = siguienteHayComida(jugador,
lista_comiditas)
        jugador.addMovimiento()
def moverAbajo(jugador, lista_comiditas):
    posX= jugador.getPosX()+1#filas
    posy= jugador.getPosY()#columnas
    if posX <= 12:
        jugador.setPosX(int(posx))
        jugador.setPosY(int(posy))
        movimiento = siguienteHayComida(jugador,
lista_comiditas)
        jugador.addMovimiento()
def moverIzquierda(jugador, lista_comiditas):
    posX= jugador.getPosX()#filas
    posy= jugador.getPosY()-1#columnas
    if posy >= 0:
        jugador.setPosX(int(posx))
        jugador.setPosY(int(posy))
        movimiento = siguienteHayComida(jugador,
lista_comiditas)
        jugador.addMovimiento()
def moverDerecha(jugador, lista_comiditas):
    posX= jugador.getPosX()#filas
    posy= jugador.getPosY()+1#columnas
    if posy <= 12 :
        jugador.setPosX(int(posx))
        jugador.setPosY(int(posy))
        movimiento = siguienteHayComida(jugador,
lista_comiditas)
        jugador.addMovimiento()
```


Métodos de Movimientos utilizando if llamamos cada método

```
def movimientos(jugador, lista_comiditas):  
    #Mientras movimiento sea verdadero, se efectuaran los movim  
    #ientos ASDW,aswd,4568  
    while True:  
        movimiento = input("Movimiento: ")  
        if str(movimiento)=="w" or str(movimiento)=="8" or  
str(movimiento)=="W":  
            moverArriba(jugador,lista_comiditas)  
        if str(movimiento)=="s" or str(movimiento)=="5" or  
str(movimiento)=="S":  
            moverAbajo(jugador,lista_comiditas)  
        if str(movimiento)=="a" or str(movimiento)=="4" or  
str(movimiento)=="A":  
            moverIzquierda(jugador,lista_comiditas)  
        if str(movimiento)=="d" or str(movimiento)=="6" or  
str(movimiento)=="D":  
            moverDerecha(jugador,lista_comiditas)  
        if str(movimiento)=="e":  
            return option(jugador)  
        print(" - JUGADOR:{0} - PUNTOS: {1} - MOVIMIENTOS:  
{2}".format(jugador.nombre, jugador.getPuntos(),jugador  
.getMovimientos()))  
        tablero = pintarTablero(lista_comiditas, jugador)  
        ImpresionDeTablero(tablero)  
        if jugador.getPuntos() > 40 or aunHayComidas(  
lista_comiditas):  
            auxilio= Jugador()  
            auxilio.nombre = jugador.nombre  
            auxilio.puntos = jugador.puntos  
            auxilio.movimientos = jugador.movimientos  
            todosjugadores.append(auxilio)  
            jugador.nombre = ""  
            jugador.puntos = 0  
            jugador.movimientos = 0  
            break
```

Método que Genera Posiciones Aleatorias




```
#Generador de Posiciones Aleatorias de jugador
def PosicionAleJugador(jugador,lista_comiditas):
    while True:
        posx = random.randint(0,12)
        posy = random.randint(0,12)
        isOcupado = False
        for comida in lista_comiditas:
            try:
                if comida.posx== posx and comida.posy==posy
:
                    isOcupado = True
            except Exception as e:
                """"""
        if not isOcupado:
            jugador.setPosX(posx)
            jugador.setPosY(posy)
            break;
```

Se genera Ordenamiento



```
#Acá se definen los 3 primeros lugares
def Position():
    todosjugadores.sort()
    if len(todosjugadores) == 1:
        print("1. {0} - MOVIMIENTOS:{1} PUNTOS:{2} ".
format(todosjugadores[0].nombre, str(todosjugadores[0]
].movimientos),str(todosjugadores[0].puntos)))
    elif len(todosjugadores) == 2:
        print("1. {0} - MOVIMIENTOS:{1} PUNTOS:{2} ".
format(todosjugadores[0].nombre, str(todosjugadores[0]
].movimientos),str(todosjugadores[0].puntos)))
        print("2. {0} - MOVIMIENTOS:{1} PUNTOS:{2} ".
format(todosjugadores[1].nombre, str(todosjugadores[1]
].movimientos),str(todosjugadores[1].puntos)))
    elif len(todosjugadores) == 3:
        print("1. {0} - MOVIMIENTOS:{1} PUNTOS:{2} ".
format(todosjugadores[0].nombre, str(todosjugadores[0]
].movimientos),str(todosjugadores[0].puntos)))
        print("2. {0} - MOVIMIENTOS:{1} PUNTOS:{2} ".
format(todosjugadores[1].nombre, str(todosjugadores[1]
].movimientos),str(todosjugadores[1].puntos)))
        print("2. {0} - MOVIMIENTOS:{1} PUNTOS:{2} ".
format(todosjugadores[2].nombre, str(todosjugadores[2]
].movimientos),str(todosjugadores[2].puntos)))
```

Método jugador que llama el final la parte lógica del juego con sus respectivas listas.



```
def juego(jugador):  
    #Se definen las comidas  
    nombre = input("Por favor escribe tu nombre: ")  
    jugador.nombre=nombre  
    alimento_pedido = random.randint(1,int((13*13)*0.4))  
    lista_comidas = []  
    CrearComidas(lista_comidas, alimento_pedido)  
    PosicionAleJugador(jugador,lista_comidas)  
    #inicia el tablero  
    tablero = pintarTablero(lista_comidas, jugador)  
    ImpresionDeTablero(tablero)  
    ##Realización de juegos  
    movimientos(jugador, lista_comidas)  
    #Se manda a llamar el menú  
    jugador = Jugador()  
    option(jugador)
```