UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

FACULTAD DE COMPUTACIÓN

UTEC

CARRERA DE CIENCIAS DE LA COMPUTACIÓN
Y SISTEMAS DE INFORMACIÓN

INFORME DEL PROYECTO 1 - SNAKE V.2

INTEGRANTES:

- Godoy Torres Diego Fabricio - Código: 202510325
- Meniz Cueva Luana Yolanda - Código: 202510593
- Ortega Capacute Piero Alejandro - Código: 202510580
- Ortega Hernández Nicole Alice - Código: 202510647

DOCENTE:

Julio Eduardo Yarasca Moscol

ASIGNATURA:

Programación I

SECCIÓN:

Laboratorio 1.04

FECHA:

27 de junio de 2025

Lima, Perú

Introducción

Gracias al avance tecnológico, la industria de los videojuegos ha evolucionado significativamente en las últimas décadas, ofreciendo experiencias visuales innovadoras y la posibilidad de conectarse con jugadores de todo el mundo. Estos juegos se han integrado a la vida cotidiana, transformándose en una forma de entretenimiento global.

Sin embargo, han perdido la simplicidad característica de los títulos clásicos. Juegos como *Snake* marcaron la infancia de generaciones anteriores, pues podían disfrutarse en dispositivos compactos y contaban con mecánicas accesibles. Además, sus principios han sido fundamentales en la enseñanza de programación básica, así como en el desarrollo de sistemas de inteligencia artificial y aprendizaje automatizado.

Por esta razón, hemos diseñado un código que replica las características esenciales de este icónico juego, manteniendo su esencia mientras exploramos su implementación técnica. Esta vez implementando unas mejoras a nuestro código para que sea más estético, avanzado e inmersivo.

Definiciones:

En el código desarrollado para la implementación del juego Snake se emplearon múltiples variables. Estas cumplen diversas funciones esenciales dentro del programa, ya sea el control del movimiento de la serpiente, la gestión de la posición de los elementos en pantalla *(como las manzanas o el cuerpo de la serpiente)*, el seguimiento de la puntuación del jugador, la detección de colisiones, y el control del estado general del juego *(inicio, pausa o fin de partida)*. En esta parte del informe vamos a presentar y explicar el funcionamiento de algunas de las variables más importantes:

En la parte inicial del código se muestran las siguientes variables: **Iserp**, **Imanz**, **Imatriz, Jugadores, Mv, Meses.**

a. **Iserp:** Es utilizado como la inicialización de la serpiente, este se utiliza para crear el cuerpo de la serpiente al iniciar cada partida.
b. **Imanz:** Indica la ubicación de las manzanas dentro de la matriz inicial (*Imatriz*). Esta variable se modificará en función de la interacción de la serpiente con las casillas que contengan una manzana.
c. **Imatriz:** Representa el mapa del juego, es decir, es la zona en donde se moverá la serpiente.
d. **Jugadores:** Este se encarga de ser el nombre del archivo JSon.
e. **Mv:** La lista llamada de esta manera es la encargada de recolectar la información de los movimientos que hace el jugador dentro de su lista.
f. **Meses:** Esta variable simplifica el trabajo de designación de mes de acuerdo con su número representante.

Además, en las líneas 22 al 25, encontramos un conjunto de comandos encargados de abrir una canción escogida por el grupo, esta se reproducirá durante toda la ejecución del programa y se cerrará al presionar la tecla 4 en el menú principal. Algo bastante simple pero que adiciona un mejor engagement al usuario.

Dentro de la función ***game*** se definen seis variables, de las cuales tres desempeñan un papel fundamental en la lógica del juego: **serp**, **pj** y **sp**

a. **serp:** Esta variable define el cuerpo de la serpiente a lo largo de la partida, y su estado puede modificarse dependiendo de si su cabeza atraviesa una casilla que contenga una manzana.
b. **pj:** Este representa el puntaje que tiene el jugador durante toda la partida, la variable varía en el caso que se coma una manzana.
c. **sp:** Este valor representa si el juego debe continuar o no. "sp" puede variar si se incumplen las reglas del juego "Snake", tales como:
    1. Si la serpiente se choca con si misma.
    2. Si existen más de 3 manzanas en el tablero
    3. Si la serpiente sale del tablero inicial.

Funcionamiento del programa

1. Función Limpieza de Pantalla:

Esta función es realmente corta, apenas toma una línea y la única funcionalidad que tiene es estética. Para la activación de esta función, primero tenemos que importar "os", que son las funcionalidades del sistema operativo. Después, "cls" que es el comando de consola para limpiar la pantalla en Windows. Consecuentemente le sigue una verificación con os.name, que en nuestro caso (dispositivo Windows, reiterando), y la comparamos "nt" que sería el adecuado de acuerdo a nuestro sistema operativo. Finalmente le adicionamos un else, en caso se use este código en cualquier otro SO, como lo pueden ser Linux o macOS.

```
def limpieza_pantalla():
    os.system('cls' if os.name == 'nt' else 'clear')
```

2. Función inicio:

Al comenzar el programa, se llama a la función ***inicio*** *(sin parámetros porque trabaja con variables locales)* que a su vez llama a la función ***limpieza_pantalla*** que se encarga de lo ya mencionado, a su vez llamamos a la función ***cargar_jugadores*** para registrar los datos de jugadores desde el archivo JSON. También, se invoca la función ***Pinicio*** y se guarda lo que

se retorna en la variable ***x***, que luego se convierte en entero. Esta última función permite al jugador seleccionar una de las cuatro opciones disponibles. Según la elección realizada, el programa procederá de distinta manera: finalizará o ejecutará funciones adicionales en función de la decisión del usuario.

```
def inicio():
    limpieza_pantalla()
    cargar_jugadores()
    x = Pinicio()
    x = int(x)
    if (x == 1):
        newgamers = prevgame()
        print(newgamers)
    elif (x == 2):
        record()
    elif (x == 3):
        info_jugadores()
    elif (x == 4):
        print("Saliendo del juego...")
        time.sleep(1)
        print("Hasta luego!!!!")
        pygame.mixer.music.stop()
        return 0
    return inicio()
```

1. Si el jugador selecciona la **opción 1**, se llamará a la función ***prevgame***, que a su vez invocará la función ***game***, cuya lógica se explicará más adelante.
2. Si el jugador elige la **opción 2**, se ejecutará la función ***record***, la cual mostrará otro menú donde se encontrarán 2 funciones más, las cuales son ***Ranking*** y ***Ganadores del mes***, que hablaremos de ellas más adelante.
3. Si el jugador opta por la **opción 3**, el programa lo mandará a una función donde o muestra que no hay datos de jugadores pasados o muestra una tabla con toda la información obtenida mediante el uso del juego, se hablará más a detalle en su momento.
4. Si el jugador elige la **opción 4**, lo que el programa hará es imprimir unos strings de despedida además de cerrar el bucle de reproducción de la música que se puso en el principio del código.

3. Función guardar_jugadores:

Primero, el archivo ***"jugadores.json"*** se abre en modo escritura (***"w"***). Si ya existe, los datos anteriores son sobrescritos; si no, el archivo se crea automáticamente.

La estructura ***with open(...) as f:*** crea una referencia al archivo abierto a la que se le llama ***f*** y asegura el cierre correcto del archivo después de escribir los datos, evitando posibles errores de acceso.

La función ***json.dump()*** procesa el diccionario **jugadores**, convirtiéndolo a formato JSON, el cual mantiene la misma estructura pero con un estándar más universal y ampliamente compatible.

Posteriormente, los datos transformados se escriben dentro del archivo **"jugadores.json"**, que actúa como el almacenamiento final para registrar y preservar la información del juego.

```
def guardar_jugadores():
    with open("jugadores.json", "w") as f:
        json.dump(jugadores, f)
```

4. Función cargar_jugadores:

Primero, se indica que la función modificará el diccionario ***jugadores*** a nivel global, permitiendo que los datos cargados sean accesibles en todo el programa. Luego, se abre el archivo ***"jugadores.json"*** en modo lectura (***"r"***) y se crea una referencia al archivo abierto, permitiendo que se acceda a él usando la variable ***f***, seguidamente se cargan los datos en **jugadores** usando ***json.load(f)*** y cuando el bloque with termina, el archivo se cierra automáticamente. Si el archivo existe, **jugadores** ahora tendrá los registros guardados. En caso jugadores.json no exista, la función captura el error ***(FileNotFoundError)*** y asigna {} a **jugadores**, evitando que el programa falle.

```python
def cargar_jugadores():
    global jugadores
    try:
        with open("jugadores.json", "r") as f:
            jugadores = json.load(f)
    except FileNotFoundError:
        jugadores = {}
```

5. Función Pinicio:

La función ***Pinicio()*** presenta el menú principal del juego en consola, permitiendo al usuario visualizar 4 opciones: Empezar el juego (**"1"**), visualizar los récords (**"2"**), visualizar la información de los jugadores (**"3"**) o salir del programa (**"4"**).

Para optimizar la presentación, incorpora pausas estratégicas (*time.sleep(0.2)*) antes de cada print, proporcionando un flujo más natural y legible. Asimismo, implementa una validación de entrada, garantizando que el usuario solo pueda elegir entre las opciones disponibles.

Finalmente, la función devuelve la selección del usuario como string, la cual será procesada en inicio() para determinar el flujo de ejecución del programa.

```python
def Pinicio():
    limpieza_pantalla()
    print("## MENU DE INICIO ##")
    time.sleep(0.2)
    print()
    print("1. Empezar el juego")
    time.sleep(0.2)
    print("2. Récord")
    time.sleep(0.2)
    print("3. Info de Jugadores")
    time.sleep(0.2)
    print("4. Salir")
    time.sleep(0.2)
    print()
    print("####################")
    print()
    x = input("Seleccione una opcion: ")
    while x not in ["1", "2", "3", "4"]:
        x = input("Por favor vuelva a selecionar la opcion: ")
    return x
```

OPCIÓN 1: Comenzar el juego

6. Función prevgame:

La función ***prevgame()*** primeramente limpia la pantalla llamando la función ***limpieza_pantalla()***, continuamente, vacía la lista de comida (***imanz***), asegurando que no hayan restos de la partida anterior. Seguidamente, se muestra el título del inicio con una pausa *(time.sleep(0.5))*, para dar un mejor efecto visual. Se solicita el nombre del jugador antes de comenzar la partida y se guarda en la variable "***x***", luego se pide el correo del usuario y se guarda en la variable correo, posteriormente con un while se asegura que el jugador haya ingresado el carácter @ y el dominio adecuado. Siguiendo con la explicación, se pide la fecha actual y se guarda en la variable fecha, si es que se ingresa una fecha no válida (ya sea un mes mayor a 12 o menor que 1), se le pedirá que vuelva a ingresar nuevamente lo pedido. Llegando al final de la primera parte de la función, creamos una variable llamada "nuevo_id", donde se le asignará un ID al usuario nuevo, con un string constante designado "SY" y un número entero a su derecha, medido por la longitud adicionado con 1 de la cantidad de jugadores. Al acabar ese proceso, llamamos a la variable global de jugadores donde ingresamos toda la información relacionada del jugador.

```
def prevgame():
    limpieza_pantalla()
    imanz.clear()
    print()
    print("## INICIO DEL JUEGO ##")
    time.sleep(0.5)
    print()
    x = input("Ingrese su nombre: ")
    correo = input("Ingrese su correo: ")
    while "@" not in correo or "utec.edu.pe" not in correo:
        correo = input("Asegúrese de que su correo incluya @ y dominio utec: ")
    fecha = input("Ingrese la fecha (MM/AA): ")
    fecha_dividida = fecha.split("/")
    while len(fecha_dividida) != 2 or int(fecha_dividida[0]) >= 13 or int(fecha_dividida[0]) <= 0 or int(fecha_dividida[1]) < 0:
        fecha = input("Ingrese nuevamente la fecha (MM/AA): ")
        fecha_dividida = fecha.split("/")
    print("\n####################")
    nuevo_id = f"SY{len(jugadores)+1}"
    jugadores[x] = {
        'id': nuevo_id,
        'puntaje': 0,
        'correo': correo,
        'fecha': fecha,
        'movimientos': 0
    }
```

A continuación, para que esta última se almacene en el diccionario **jugadores**. Se llama a la función ***guardar_jugadores*** y posteriormente a la función ***game***. A esta última se le envía la serpiente inicial, el tablero y el nombre del jugador, lo que retorna esta función, se guarda en las variables ***nombre***, ***puntaje y movimientos*** .

También se verifica y actualiza el récord dependiendo de la puntuación obtenida. Si el nuevo puntaje es mayor que el anterior, se actualiza el récord y se guarda nuevamente en ***jugadores.json*** para conservar la mejora. Finalmente, se devuelve el nombre, el puntaje

final del jugador y su ID correspondiente, que será SYn, donde "n" es el número de jugador, este es muy importante para la posterior búsqueda en la función info_jugadores.

```
guardar_jugadores()

nombre, puntaje, movimientos = game(iserp.copy(), imatriz, x, 0, 0, 0)

if puntaje > jugadores[nombre]["puntaje"]:
    jugadores[nombre]["puntaje"] = puntaje

jugadores[nombre]["movimientos"] = mv.copy()
mv.clear()
print(f"\n✔ Juego guardado exitosamente. Tu ID es: {nuevo_id}")
input("Presiona ENTER para continuar...")
return (nombre, puntaje)
```

7. Función game:

La función game se define con 6 parámetros necesarios para el funcionamiento del programa:

a. **serp:** Es representada como una lista con las coordenadas de la serpiente
b. **mz:** Es la matriz del tablero
c. **name:** Contiene al nombre del jugador
d. **pj:** Simboliza el puntaje acumulado
e. **sp:** Se utiliza para ver el estado del juego, nos permite saber si ,durante la partida, el jugador rompió alguna regla del juego. Si se detecta una violación a las normas, el valor de *sp* se actualizará a **1**, lo que provocará la finalización de la función. En cambio, si no se altera ninguna regla, el juego continuará con *sp* en **0**. Por defecto, *sp* siempre comienza con un valor de **0** al iniciar una nueva partida, garantizando un estado inicial sin restricciones.
f. **mov:** Esta variable se activa únicamente cuando la serpiente no consume ninguna manzana durante los últimos 20 movimientos. Si en algún momento la serpiente ingiere una manzana, el valor de la variable se restablecerá automáticamente a **0**, reiniciando el contador.

Apenas al iniciar esta función, se limpiará la pantalla mediante la conocida función limpieza_pantalla y se imprimirá un contador con la cantidad de movimientos realizados.

Desarrollo de la función por bloques:

I. Durante la ejecución de la función *game*, se verifica el valor de la variable *sp*. Si *sp* es igual a **1** ***(caso base)*** , la función finalizará de inmediato y retornará el nombre del jugador junto con su puntaje.

```
if sp == 1:
    print("Fin del juego")
    print("Puntaje total: " + str(pj))
    return (name, pj, mov)
```

En caso contrario, el código continuará su ejecución sin interrupciones.

II. En esta sección del código, se utiliza una tupla denominada ***k***, la cual representa una posición aleatoria dentro del tablero de juego. Esta tupla contiene dos valores enteros correspondientes a la fila y la columna, restringidos de manera que siempre pertenezcan a una ubicación válida dentro del cuadro. Si la posición ***k*** coincide con una parte del cuerpo de la serpiente, se generará un nuevo valor para evitar conflictos. En el caso que hayan 0 manzanas, será necesario agregar una nueva posición con el valor de ***k***, el cual se añadirá a la lista ***imanz***, asegurando su incorporación correcta en el juego.

```
k = (random.randint(0, 9), random.randint(0, 19))
while mov % 20 == 0 and k in serp:
    k = (random.randint(0, 9), random.randint(0, 19))
if mov % 20 == 0 or len(imanz) == 0:
    imanz.append(k)
```

III. Esta sección del código se encargará de representar gráficamente el tablero de juego, incluyendo la serpiente, las manzanas y los espacios vacíos. Durante la ejecución, el programa determinará si en cada posición debe imprimirse un **punto**, una **parte del cuerpo de la serpiente**, o una **manzana** con su respectivo color asignado. Para llevar a cabo esta representación, se utilizarán los valores ***serp***, ***imanz*** y ***mz***, que gestionan la ubicación de cada elemento dentro del tablero.

```
for i in range(10):
    for j in range(20):
        if (i, j) in serp:
            if (i, j) == serp[-1]:
                print(Fore.BLUE + ">" + Style.RESET_ALL, end="")
            else:
                print(Fore.BLUE + "-" + Style.RESET_ALL, end="")
        elif (i, j) in imanz:
            print(Fore.RED + "@" + Style.RESET_ALL, end="")
        else:
            print(Fore.GREEN + mz[i][j] + Style.RESET_ALL, end="")
    print()
```

IV. En esta sección del código, se verifica si la cantidad de manzanas en el tablero es igual a **3**. Si se alcanza este límite, la partida finalizará inmediatamente y se retornará el **nombre del jugador**

```
if len(imanz) == 3:
    print()
    print("Fin del juego")
    print("Puntaje total: ", pj)
    return (name, pj, mov)
```

junto con su **puntaje y** su **cantidad de movimientos** desde la última recolección de manzana**.**. En caso contrario, el juego continuará su ejecución sin interrupciones.

V. En esta sección del código, se presentan las opciones disponibles para el jugador una vez que se ha mostrado el tablero con la serpiente y las manzanas. Si el jugador realiza una selección no válida, el programa solicitará que elija nuevamente, asegurando que solo se admitan decisiones permitidas dentro del juego. Además, en el caso que se usen teclas admitidas de movimiento (excluyendo el exit), se agregarán a una lista denominada "mv", que será próximamente mostrada en la función info_jugadores.

```
print("Arriba (w)")
print("Abajo (s)")
print("Izquierda (a)")
print("Derecha (d)")
print("Salir (exit)")
ele = input("Seleccione un movimiento: ")
while ele not in ["w", "s", "a", "d", "exit"]:
    ele = input("Seleccione nuevamente el movimiento: ")
if ele != "exit":
    mv.append(ele)
```

VI. La variable ***Serp2*** almacena una copia del cuerpo de la serpiente antes de que se realice el movimiento, mientras que ***new*** representa la nueva cabeza tras la acción del jugador.

Después de que el usuario seleccione una opción, el código determinará la nueva posición de la cabeza de la serpiente, con condiciones en el caso que pase la serpiente los límites de la matriz. Habiendo declarado esas condiciones, como que si la serpiente sube a la fila 9, aparezca en la fila 0 y viceversa o si está en la columna 19, aparezca en la columna 0 y viceversa.

```
serp2 = serp.copy()
new = (0, 0)
last = serp2[len(serp2) - 1]
if ele == "w":
    new = (9, last[1]) if last[0] == 0 else (last[0] - 1, last[1])
elif ele == "s":
    new = ((last[0] + 1) % 10, last[1])
elif ele == "a":
    new = (last[0], 19) if last[1] == 0 else (last[0], last[1] - 1)
elif ele == "d":
    new = (last[0], (last[1] + 1) % 20)
elif ele == "exit":
    sp = 1
serp2.append(new)
if new not in imanz:
    serp2.pop(0)
if new in imanz:
    pj += 1
    imanz.remove(new)
    if len(imanz) == 0:
        mov = -1
```

Una vez realizado el movimiento, se verificará si la nueva cabeza coincide con la ubicación de alguna manzana. **Si la serpiente come una manzana**, su cola permanecerá intacta, la manzana será eliminada del tablero y se añadirá un punto al marcador. **Si no**

consume una manzana, la cola de la serpiente se eliminará como parte del desplazamiento natural.

En caso de que la serpiente haya comido una manzana, se evaluará cuántas quedan en el tablero. **Si el número de manzanas restantes es cero**, la variable *movs* se establecerá en **-1**, lo que permitirá que, al volver a ejecutar la función *game*, se genere una nueva manzana en el tablero.

VII. En esta última sección del código, se verifica si la longitud de la serpiente corresponde con la puntuación obtenida con las manzanas adicionadas con 3 enteros (la longitud inicial de la serpiente). En caso de ser diferentes, la variable sp se iguala a 1, lo cual significa un error y se finalizará la función game, para imprimir la función y el ID correspondiente.

```
for i in serp2:
    if len(set(serp2)) != pj + 3:
        sp = 1
return game(serp2, imatriz, name, pj, sp, mov + 1)
```

OPCIÓN 2: Récord de jugadores

Esta función solamente sirve como un menú conector o intermediario de otras funciones, empezando por llamar la funcion *limpieza_pantalla* e imprimir el nombre del menú, seguido de varios **time.sleep** después de cada línea impresa que muestra las dos opciones de menú para mejorar la legibilidad del programa. Añadimos un input llamado elección y 3 condicionales para redirigir a la funciones correspondientes. En el caso que haya un input diferente a los requeridos, se ingresará a un loop para obtener lo requerido.

```
def record():
    limpieza_pantalla()
    print("## MENU RECORD ##")
    time.sleep(0.5)
    print()
    print("1. Ranking")
    time.sleep(0.2)
    print("2. Ganadores del mes")
    time.sleep(0.2)
    print()
    print("############")
    eleccion = input("Seleccione una opcion: ")
    if eleccion == "1":
        ranking()
        return
    elif eleccion == "2":
        ganadores_mes()
        return
    elif eleccion.lower() == "s":
        return inicio()

    while eleccion not in ("1", "2", "s", "S"):
        eleccion = input("Seleccione una de las opciones disponibles: ")
        if eleccion == "1":
            ranking()
            return
        elif eleccion == "2":
            ganadores_mes()
            return
        elif eleccion.lower() == "s":
            return inicio()
```

8. Función Ranking:

Primero, limpiamos la pantalla llamando esta función, después, se muestra el título de la sección y se introduce una breve pausa **(time.sleep(0.5))** para mejorar la presentación y fluidez del contenido.

Si la variable jugadores está vacía ({}), significa que no hay datos almacenados, por lo que se muestra un mensaje de aviso informando al usuario.

En caso de que existan jugadores registrados, se emplea la función **sorted()**, que permite ordenar los elementos de **jugadores.items()**, una lista de tuplas generada a partir del diccionario **jugadores**. En cada tupla, el primer elemento corresponde al nombre del jugador y el segundo a un diccionario con sus estadísticas, incluyendo su puntaje. El parámetro **key**, define el criterio de ordenación. En este caso, se utiliza una función **lambda**, la cual actúa como una función anónima dentro de **sorted()**, extrayendo el puntaje de cada jugador ***(j[1]["puntaje"])*** para determinar el orden. Adicionalmente, el parámetro **reverse**=True indica que la lista se ordenará de forma descendente, colocando primero a los jugadores con mayor puntuación.

Luego, **enumerate()** permite recorrer la lista de jugadores asignando un número automático a cada uno, iniciando en 1 (por defecto, enumerate() empieza en 0).

a. i: Almacena el número de posición
b. name: Contiene el nombre del jugador
c. datos: Guarda el diccionario con su ID y puntaje.

Finalmente, se imprime la clasificación en un formato claro y gráfico, para luego, solicitar al usuario que ingrese **"s"** para salir. En caso de que el usuario introduzca un valor incorrecto, el programa seguirá preguntando hasta recibir una respuesta válida. Para concluir, la función devuelve **0**, lo que indica su finalización y permite que el flujo del programa regrese al menú principal.

```python
def ranking():
    limpieza_pantalla()
    print("### LISTA DE GANADORES ###")
    time.sleep(0.5)
    print()
    if not jugadores:
        print("Aún no hay historial de jugadores ni datos pasados :c\n")
    else:
        organizado = sorted(jugadores.items(),
                            key=lambda j: j[1]["puntaje"], reverse=True)
        print("----------+---------------+--------------------")
        print("| ID      | NOMBRE        | PUNTOS             |")
        print("----------+---------------+--------------------")
        for i, (name, datos) in enumerate(organizado, 1):
            print(
                f"| {datos['id']:<8}|{name:<12}   | {datos['puntaje']} PUNTOS          |")
            time.sleep(0.2)
        print("----------+---------------+--------------------")
        print()
    print("######################")
    x = input("Ingrese la letra 's' para regresar al menú principal: ")
    while x.lower() != "s":
        x = input("Por favor, vuelva a ingresar la letra 's' para regresar: ")
    return 0
```

9. Función Ganadores del mes:

En esta función iniciamos con las funciones limpieza_pantalla y cargar_jugadores como ya es costumbre. A continuación, imprimimos un string que escribe la función en la que estamos. Nuevamente tenemos dos posibles casos. En el primer caso, se imprimirá un string que comenta que no hay historial de jugadores, en el caso que así sea, de otra manera se empezará con una nueva lógica. En esta última, creamos un diccionario vacío llamado "ganadores", en ella llegaremos a agregarán los datos de mes en formato 01, 02, etc y un valor de nombre y puntaje de manera descendente. Consecuentemente iteramos en el diccionario meses y añadimos una condición donde en el caso que el mes en formato numérico exista, se imprimirá el mes en formato string, gracias al diccionario meses, después se escribirá el nombre del usuario que obtuvo la mejor puntuación durante ese periodo. En el caso que no exista un jugador en aquel mes, devolveremos un string que escriba el mes de manera string y un "Ninguno" o "No existe", diferentes dependiendo del mes, para que al momento de imprimir, sea más estético. Finalmente agregamos un input para regresar al menú principal.

```
def ganadores_mes():
    limpieza_pantalla()
    cargar_jugadores()
    print("### GANADORES DEL MES ###\n")
    time.sleep(0.3)
    if not jugadores:
        print("Aún no hay historial de jugadores ni datos pasados :c\n")
    else:
        ganadores = {}
        for names, j in jugadores.items():
            fecha = j["fecha"].split('/')[0]
            puntaje = j["puntaje"]
            if fecha not in ganadores or puntaje > ganadores[fecha][1]:
                ganadores[fecha] = (names, puntaje)

        for mes, i in meses.items():  # retorna enero 01
            if i in ganadores:
                print(f"{mes}:    {ganadores[i][0]}")
            else:
                if mes in ["Enero", "Marzo", "Mayo", "Julio", "Septiembre", "Noviembre"]:
                    print(f"{mes}:  Ninguno ")
                else:
                    print(f"{mes}:  No existe")
            time.sleep(0.1)
    print()
    input("Presione \"s\" para volver al menú: ")
```

OPCIÓN 3: Info de Jugadores

10. Función Info Jugadores:

Para esta función, empezamos con la limpieza de pantalla y entrada de una función externa llamada cargar_jugadores además de una variable llamada target donde se ingresa el ID del

jugador a buscar. Posteriormente imprimimos el título del menú y seguimos creando la variable llamada user, que la llenaremos con un None para posteriormente escribir dentro de ella el nombre del usuario en un for donde se iterará dentro de archivo Json, jugadores. En el caso que se encuentre al jugador, hacemos una nueva verificación para comprobar que no es un None y consecuentemente agregamos los movimientos en formato string a un string vacío.

```
def info_jugadores():
    limpieza_pantalla()
    cargar_jugadores()
    target = input("Ingrese el ID del jugador: ")
    print("### INFORMACIÓN DE JUGADORES ###\n")
    time.sleep(0.3)

    user = None
    for nombre, datos in jugadores.items():
        if datos["id"] == target:
            user = nombre
            jugador = datos
            break

    if user:
        movimientos = jugador.get("movimientos", [])
        if isinstance(movimientos, list):
            movimientos = ", ".join(movimientos)
```

A continuación, creamos un ancho máximo de la columna movimientos, y asignamos una variable que se encargue de controlar esta misma para fines netamente gráficos. Posteriormente crearemos una tabla mediante la impresión de strings para luego imprimir los datos del jugador pedido, como su ID, nombre de usuario, puntaje y lista de los movimientos que anteriormente encontramos. Finalmente imprimimos una fila del tamaño del ancho de la tabla para dar la sensación de tabla ordenada. En el caso que no haya información en el archivo Json, se imprimirá un string que avise de esta situación.

```
    ancho_columna = 15
    mov_lines = [movimientos[i:i + ancho_columna]
                 for i in range(0, len(movimientos), ancho_columna)]
    print("---------+--------------+-----------+----------------------------+-----------------")
    print("| ID     | NOMBRE       | PUNTAJE   | CORREO                     | MOVIMIENTOS     |")
    print("---------+--------------+-----------+----------------------------+-----------------")

    print(
        f"| {jugador['id']:<6} | {user:<13} | {jugador['puntaje']:<10} | {jugador['correo']:<28} | {mov_lines[0]:<15} |")
    for linea in mov_lines[1:]:
        print(f"| {'':<6} | {'':<13} | {'':<10} | {'':<28} | {linea:<15} |")
    print("-------------------------------------------------------------------------------------")
else:
    print("No hay datos de jugadores con ese ID :( ")

print()
time.sleep(0.3)
x = input("Ingrese la letra 's' para regresar al menú principal: ")
while x.lower() != "s":
    x = input("Por favor, vuelva a ingresar la letra 's' para regresar: ")
return 0
```

Ejemplos de ejecución:

A continuación presentaremos ejemplos de entradas y salidas de nuestro programa:

1.

```
## MENU DE INICIO ##

1. Empezar el juego
2. Récord
3. Info de Jugadores
4. Salir

####################

Seleccione una opcion: 1
```

2

```
## INICIO DEL JUEGO ##

Ingrese su nombre: Gorgoy
Ingrese su correo: gorgot@utec.edu.pe
Ingrese la fecha (MM/AA): 07/2025
```

3

```
Cantidad de movimientos: 3
## SNAKE ##
Jugador:  Gorgoy
Puntaje:  14
.....................
.....................
.....................
.....................
.....................
....--...............
....------>..........
....--------.........
.....................
.................@...
Arriba (w)
Abajo (s)
Izquierda (a)
Derecha (d)
Salir (exit)
Seleccione un movimiento:
```

4

```
## MENU DE INICIO ##

1. Empezar el juego
2. Récord
3. Info de Jugadores
4. Salir

####################

Seleccione una opcion: 2
```

5

```
## MENU RECORD ##

1. Ranking
2. Ganadores del mes

#############
Seleccione una opcion: 1
```

6

```
### LISTA DE GANADORES ###

----------+---------------+--------------------
| ID      | NOMBRE        | PUNTOS             |
----------+---------------+--------------------
| SY1     |Diego          | 6 PUNTOS           |
| SY3     |Luana          | 5 PUNTOS           |
| SY4     |Nicole         | 2 PUNTOS           |
| SY2     |Piero          | 1 PUNTOS           |
----------+---------------+--------------------

######################
Ingrese la letra 's' para regresar al menú principal:
```

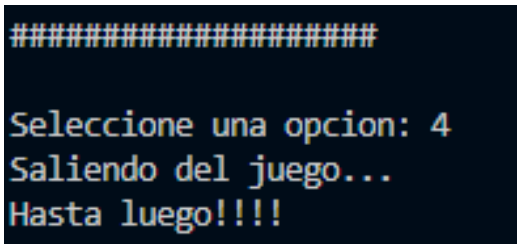
7

```
### GANADORES DEL MES ###

Enero:    Luana
Febrero:    Nicole
Marzo:    Piero
Abril:  No existe
Mayo:  Ninguno
Junio:    Diego
Julio:  Ninguno
Agosto:  No existe
Septiembre:  Ninguno
Octubre:  No existe
Noviembre:  Ninguno
Diciembre:  No existe

Presione "s" para volver al menú:
```

8



Conclusiones:

Para finalizar con este informe, nos gustaría brindar conclusiones sobre el código realizado.

- La elaboración del programa presenta una estructura bien definida. El código está organizado en funciones separadas y cada una cumple y gestiona distintos aspectos del juego, lo que permite y facilita la compresión total del programa.

- En el desarrollo del código, se implementan controles adecuados para evitar que ocurran errores al ejecutar el programa. De este modo, se asegura que el usuario ingrese opciones válidas antes de procesar.

- El código cuenta con interactividad y fluidez a lo largo de la ejecución. Se han agregado pausas estratégicas en el flujo del programa para mejorar la experiencia del usuario y hacer la ejecución más natural.

- Se manejan correctamente las condiciones de salida o finalización. El código ejecuta apropiadamente los casos en que el juego debe terminar, como cuando la serpiente choca consigo misma, cuando toca algún borde de la matriz o cuando se llega al límite de manzanas establecido.

- Se ha empleado una gestión eficiente de datos. El almacenamiento y recuperación de jugadores en un archivo JSON asegura la preservación de los puntajes y permite el seguimiento de récords de manera estructurada.