UNIVERSIDAD INTERNACIONAL DEL ECUADOR – UIDE

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

LÓGICA DE PROGRAMACIÓN

Manual Técnico

PRESENTADO POR

Odalis Anangono

Keyla Imba

Pamela Toapanta

Índice

1. Introducción

El presente documento describe el diseño, implementación y funcionamiento técnico del proyecto "GANAR O MORIR", desarrollado en Python utilizando la librería Tkinter para la creación de la interfaz de usuario.

1.1. Objetivo del Proyecto

Implementar una aplicación interactiva basada en el juego clásico del ahorcado, donde el usuario debe adivinar palabras relacionadas con animales mediante la selección de letras individuales.

1.2. Alcance

El proyecto contempla la creación de una interfaz gráfica completa, sistema de puntuación, retroalimentación visual mediante emojis y efectos visuales, y lógica de juego que valida las entradas del usuario.

2. Requerimientos del Sistema

2.1. Requerimientos de Software

- Python versión 3.6 o superior
- Módulo Tkinter (incluido en la instalación estándar de Python)
- Sistema operativo compatible: Windows, macOS o Linux

2.2. Requerimientos de Hardware

- Procesador: 1 GHz o superior

- Memoria RAM: 512 MB mínimo
- Resolución de pantalla: mínimo 800x600 píxeles
- Espacio en disco: 50 MB

3. Librerías y Dependencias

El proyecto utiliza exclusivamente módulos de la biblioteca estándar de Python:

tkinter: Librería principal para la construcción de interfaces gráficas de usuario (GUI). Proporciona widgets como ventanas, botones, etiquetas y marcos para la interacción con el usuario.

tkinter.messagebox: Submódulo de tkinter que permite la creación de cuadros de diálogo modales para mostrar información, advertencias y mensajes de error al usuario.

random: Módulo para la generación de selecciones aleatorias. Se utiliza para elegir una palabra del conjunto de datos disponible al inicio de cada partida.

3.1. Instalación de Dependencias

Las librerías mencionadas se incluyen por defecto en la instalación estándar de Python. En sistemas Linux donde tkinter no esté preinstalado, se puede instalar mediante:

sudo apt-get install python3-tk

4. Arquitectura del Software

4.1. Paradigma de Programación

El proyecto implementa un enfoque de programación procedural con manejo de eventos (event-driven programming), característico de las aplicaciones con interfaz gráfica.

4.2. Estructura Modular

El código se organiza en los siguientes componentes funcionales:

a. **Módulo de Configuración** Contiene las constantes que definen la apariencia visual de la aplicación:

- Paleta de colores (esquema neon)
- Configuración tipográfica
- Estados visuales del personaje (diccionario STICKMAN)

b. **Módulo de Utilidades** Funciones auxiliares que proporcionan servicios generales:

- *center_window():* Posicionamiento de ventanas
- *calcular_puntos():* Algoritmo de puntuación
- *parpadeo_rojo():* Efectos visuales

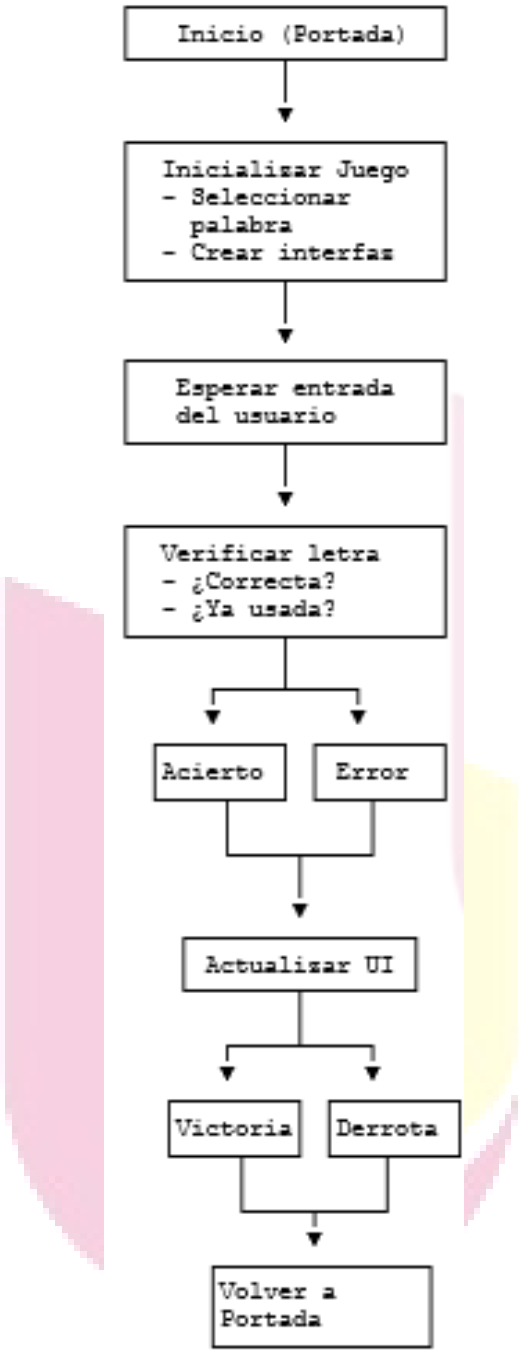
c. **Módulo de Presentación** Gestión de la pantalla inicial:

- *mostrar_portada():* Interfaz de bienvenida y punto de entrada

d. **Módulo Principal de Juego** Contiene la lógica central de la aplicación:

- *iniciar_juego():* Inicialización y control del flujo del juego
- *actualizar_ui():* Actualización de la interfaz gráfica
- *verificar_letra():* Validación y procesamiento de entrada

5. Diagrama de Flujo



6. Estructura del Proyecto

6.1. Organización de Archivos

El proyecto consiste en un único archivo ejecutable:

ahorcado_neon.py

6.2. Estructura Interna del Código

Sección 1: Configuración Global (líneas 5-22)

- Definición de constantes para colores
- Definición de fuentes tipográficas
- Diccionario de estados del personaje visual

Sección 2: Funciones Auxiliares (líneas 25-88)

- Utilidades de interfaz
- Funciones de cálculo
- Efectos visuales

Sección 3: Lógica de Aplicación (líneas 91-217)

- Función principal del juego
- Manejo de estado
- Procesamiento de eventos

6.3. Estructura de Datos

El estado del juego se almacena en un diccionario con la siguiente estructura:

```
state = {
    "palabra": str,      # Palabra objetivo a adivinar
    "guiones": list,     # Representación visual del progreso
    "errores": list,     # Registro de letras incorrectas
    "intentos": int,     # Contador de intentos restantes
    "puntos": float      # Puntuación acumulada
}
```

6.4. Conjunto de Datos

El vocabulario del juego se define mediante una lista estática:

animales = ["perro", "gato", "tigre", "leon", "lobo", "zorro"]

7. Descripción de Componentes

7.1. Interfaz Gráfica de Usuario

Ventana Principal de Juego:

- Título de aplicación
- Etiqueta de palabra (muestra progreso con guiones bajos)
- Contador de intentos restantes
- Representación visual del estado (personaje ASCII)
- Lista de errores cometidos
- Indicador de puntuación
- Teclado virtual alfabético

Ventana de Portada:

- Título del juego
- Botón de inicio

7.2. Lógica de Juego

Inicialización:

1. Selección aleatoria de palabra del conjunto de datos
2. Inicialización del estado del juego
3. Construcción de la interfaz gráfica
4. Vinculación de eventos a manejadores

Procesamiento de Entrada:

1. Captura de letra seleccionada

2. Normalización (conversión a minúsculas)
3. Validación de uso previo
4. Búsqueda en palabra objetivo
5. Actualización de estado correspondiente
6. Refresco de interfaz

Condiciones de Terminación:

- Victoria: Todas las letras han sido adivinadas
- Derrota: El contador de intentos llega a cero

7.3. Sistema de Puntuación

Se implementa un algoritmo de puntuación proporcional:

$$Puntos = (Intentos\_Restantes / Intentos\_Iniciales) \times 100$$

Donde:

- Intentos_Iniciales = 6
- Intentos_Restantes = valor actual del contador

Esto produce una escala de puntuación de 0 a 100 puntos.

7.4. Retroalimentación Visual

El sistema proporciona múltiples formas de retroalimentación:

Visual:

- Actualización del personaje según intentos restantes
- Cambio de color en lista de errores
- Efecto de parpadeo en estado crítico (1 intento restante)

Mediante Diálogos:

- Advertencia inicial sobre la categoría
- Mensaje de última oportunidad
- Notificación de victoria o derrota

8. Algoritmos Implementados

8.1. Algoritmo de Verificación de Letra

Entrada: letra (carácter)

Salida: actualización de estado

```
letra - convertir_a_minúscula(letra)
SI letra EN guiones O letra EN errores ENTONCES
   RETORNAR  // Letra ya procesada
FIN SI
SI letra EN palabra ENTONCES
   PARA cada posición i en palabra HACER
      SI palabra[i] = letra ENTONCES
         guiones[i] ← letra
      FIN SI
  FIN PARA
  actualizar_interfaz()
  SI no hay guiones bajos EN guiones ENTONCES
     mostrar_victoria()
  FIN SI
SI NO
  añadir letra a errores
  decrementar intentos
  actualizar_interfaz()
  SI intentos = 0 ENTONCES
     mostrar_derrota()
  FIN SI
FIN SI
```

8.2. Algoritmo de Centrado de Ventana

Entrada: ventana (objeto Tk)

Salida: ventana centrada en pantalla

actualizar_geometría(ventana)

ancho_ventana - obtener_ancho(ventana)

alto_ventana - obtener_alto(ventana)

ancho_pantalla - obtener_ancho_pantalla()

alto_pantalla - obtener_alto_pantalla()

x - (ancho_pantalla - ancho_ventana) / 2

y - (alto_pantalla - alto_ventana) / 2

establecer_posición(ventana, x, y)

9. Consideraciones Técnicas

9.1. Complejidad Computacional

- Verificación de letra: O(n) donde n es la longitud de la palabra
- Actualización de interfaz: O(1)
- Complejidad espacial: O(n) para almacenamiento de estado

9.2. Manejo de Eventos

La aplicación utiliza el paradigma event-driven mediante:

- Funciones lambda para vincular eventos de botones
- Método after() para efectos visuales temporizados
- Loop principal mainloop() para el ciclo de eventos

9.3. Limitaciones Conocidas

- Conjunto de palabras limitado (6 opciones)
- No persistencia de datos (sin guardado de puntuaciones)

- Interfaz en un solo idioma
- Sin niveles de dificultad variables

10. Posibles Extensiones

El diseño modular del proyecto permite futuras mejoras:

Funcionalidades adicionales:

- Implementación de categorías múltiples
- Sistema de persistencia de puntuaciones
- Niveles de dificultad (palabras más largas, menos intentos)
- Modo multijugador
- Integración de efectos de sonido

Mejoras técnicas:

- Separación en múltiples archivos (MVC completo)
- Implementación de pruebas unitarias
- Internacionalización (soporte multi-idioma)
- Base de datos para vocabulario extenso

11. Conclusiones

El proyecto cumple con los objetivos establecidos al proporcionar una implementación funcional del juego del ahorcado con interfaz gráfica. La arquitectura modular facilita el mantenimiento y futuras extensiones. El uso de librerías estándar garantiza la portabilidad entre diferentes sistemas operativos.

La aplicación demuestra conceptos fundamentales de programación como manejo de eventos, estructuras de datos, algoritmos de búsqueda y diseño de interfaces de usuario.

12. Anexos

12.1. Código fuente completo

```
import tkinter as tk
import tkinter.messagebox as mb
import random

# COLORES Y FUENTES

COLOR_FONDO = "#0a0a0f"
COLOR_NEON_ROSA = "#ff4df0"
COLOR_NEON_CIAN = "#00fff2"
COLOR_NEON_VERDE = "#28ffbf"

FUENTE_TITULO = ("Arial", 28, "bold")
FUENTE_SUBTITULO = ("Arial", 14)
FUENTE_RESALTADA = ("Arial", 20, "bold")

# MUÑECO  (estados 6 a 0)

STICKMAN = {
    6: "  \\😀/  \n   |   \n  / \\ ",
    5: "  \\😄/  \n   |   \n  /    ",
    4: "  \\😳/  \n   |   \n       ",
    3: "   😳   \n  /|   \n       ",
    2: "   😵   \n   |   \n       ",
    1: "   😵   \n       \n       ",
    0: "   💀   \n       \n       "
}

# CENTRAR VENTANA

def center_window(win):
    win.update_idletasks()
    w, h = win.winfo_width(), win.winfo_height()
    ws, hs = win.winfo_screenwidth(), win.winfo_screenheight()
    x = (ws // 2) - (w // 2)
    y = (hs // 2) - (h // 2)
    win.geometry(f"+{x}+{y}")

# CÁLCULO DE PUNTOS

def calcular_puntos(inicial, restantes):
    try:
        return (restantes / inicial) * 100
    except:
```

```
        return 0

# PORTADA

def mostrar_portada():
    portada = tk.Tk()
    portada.title("GANAR 🐱 O MORIR 🐷")
    portada.config(bg=COLOR_FONDO)

    tk.Label(
        portada,
        text="GANAR 🐱 O MORIR 🐷",
        fg=COLOR_NEON_ROSA,
        bg=COLOR_FONDO,
        font=FUENTE_TITULO
    ).pack(pady=40)

    tk.Button(
        portada,
        text="INICIAR",
        command=lambda: (portada.destroy(), iniciar_juego()),
        font=FUENTE_RESALTADA,
        fg=COLOR_NEON_CIAN,
        bg=COLOR_FONDO,
        bd=0
    ).pack(pady=20)

    portada.geometry("400x300")
    center_window(portada)
    portada.mainloop()

# PARPADEO ROJO

def parpadeo_rojo(target_window, flashes=4, delay=120):
    original = target_window.cget("bg")
    def efecto(i):
        if i <= 0:
            try:
                target_window.config(bg=original)
            except:
                pass
            return
        try:
            target_window.config(bg="#ff0000")
        except:
```

```
            pass
        target_window.after(delay, lambda: (
            target_window.config(bg=original),
            target_window.after(delay, lambda: efecto(i - 1))
        ))
    efecto(flashes)

# INICIAR JUEGO

def iniciar_juego():
    global ventana_juego
        ventana_juego = tk.Tk()
        ventana_juego.title("Ahorcado Neon")
        ventana_juego.config(bg=COLOR_FONDO)

        # Lista de animales
        animales = [
            "perro", "gato", "tigre", "leon", "lobo", "zorro"
        ]

        palabra = random.choice(animales)

        # Mensaje inicial
        mb.showinfo("ADVERTENCIA 🐾", "Debes adivinar el nombre de un
   ANIMAL 🐾")

        intentos_iniciales = 6
        state = {
            "palabra": palabra,
            "guiones": ["_" for _ in palabra],
            "errores": [],
            "intentos": intentos_iniciales,
            "puntos": 0
        }

        # UI

        tk.Label(
            ventana_juego, text="Ahorcado ",
            fg=COLOR_NEON_ROSA, bg=COLOR_FONDO,
            font=FUENTE_TITULO
        ).pack(pady=10)

        lbl_palabra = tk.Label(
            ventana_juego, text=" ".join(state["guiones"]),
```

```
            fg=COLOR_NEON_CIAN, bg=COLOR_FONDO,
            font=FUENTE_RESALTADA
        )
        lbl_palabra.pack(pady=10)

        lbl_intentos = tk.Label(
            ventana_juego, text=f"Intentos: {state['intentos']}",
            fg=COLOR_NEON_VERDE, bg=COLOR_FONDO,
            font=FUENTE_SUBTITULO
        )
        lbl_intentos.pack()

        stickman_label = tk.Label(
            ventana_juego, text=STICKMAN[state["intentos"]],
            fg=COLOR_NEON_ROSA, bg=COLOR_FONDO,
            font=("Courier", 20, "bold"), justify="center"
        )
        stickman_label.pack()

        errores_label = tk.Label(
            ventana_juego, text="Errores: ",
            fg="red", bg=COLOR_FONDO,
            font=FUENTE_SUBTITULO
        )
        errores_label.pack(pady=5)

        lbl_score = tk.Label(
            ventana_juego, text="Puntos: 0",
            fg=COLOR_NEON_CIAN, bg=COLOR_FONDO,
            font=FUENTE_SUBTITULO
        )
        lbl_score.pack()

        # ACTUALIZAR UI

        def actualizar_ui():
            lbl_palabra.config(text=" ".join(state["guiones"]))
            lbl_intentos.config(text=f"Intentos: {state['intentos']}")
            v = max(0, min(6, state["intentos"]))
            stickman_label.config(text=STICKMAN.get(v, STICKMAN[0]))
            errores_label.config(text="Errores: " + ",
  ".join(state["errores"]))
            try:
                state["puntos"] = (state["intentos"] /
  intentos_iniciales) * 100
```

```
            except:
                state["puntos"] = 0
            lbl_score.config(text=f"Puntos: {int(state['puntos'])}")

        # VERIFICAR LETRA

        def verificar_letra(letra):
            letra = letra.lower()

            if letra in state["guiones"] or letra in state["errores"]:
                return

            if letra in state["palabra"]:
                for i, c in enumerate(state["palabra"]):
                    if c == letra:
                        state["guiones"][i] = letra
                actualizar_ui()

                if "_" not in state["guiones"]:
                    mb.showinfo("GANASTE 🐱", "¡Eres un ganador! 🐱")
                    ventana_juego.destroy()
                    mostrar_portada()
            else:
                state["errores"].append(letra)
                state["intentos"] -= 1
                actualizar_ui()

                if state["intentos"] == 1:
                    mb.showinfo(
                        "Última oportunidad",
                        "😝 ¿ESTÁS LISTO PARA MORIR?\n\n🐱 ¿O ESTÁS LISTA PARA REMONTAR?"
                    )
                    parpadeo_rojo(ventana_juego, flashes=4, delay=120)

                if state["intentos"] <= 0:
                    mb.showerror("DERROTA 💀", f"Lo siento, eres un perdedor 😵\nEl animal era: {state['palabra']}")
                    ventana_juego.destroy()
                    mostrar_portada()

        # TECLADO
        teclado_frame = tk.Frame(ventana_juego, bg=COLOR_FONDO)
        teclado_frame.pack(pady=20)

```

```
        filas = ["ABCDEFGHIJKLM", "NOPQRSTUVWXYZ"]
        for fila in filas:
            fila_frame = tk.Frame(teclado_frame, bg=COLOR_FONDO)
            fila_frame.pack()
            for letra in fila:
                tk.Button(
                    fila_frame,
                    text=letra,
                    command=lambda l=letra: verificar_letra(l),
                    font=FUENTE_SUBTITULO,
                    fg=COLOR_NEON_CIAN,
                    bg=COLOR_FONDO,
                    width=3
                ).pack(side="left", padx=2, pady=2)

        ventana_juego.geometry("550x550")
        center_window(ventana_juego)
        ventana_juego.mainloop()

    mostrar_portada()

```