Tabla de Contenidos

Cmon dice....2
- Propuesta....2
  - Etapa 1: Planificación y Preparación....2
  - Etapa 2: Implementación de las Funciones Básicas....2
  - Etapa 3: Manejo del Flujo de Juego....3
  - Etapa 4: Cálculo de Puntajes....3
  - Etapa 5: Generación de Informes y Salida de Datos....3
  - Etapa 6: Pruebas y Documentación....3
  - Etapa 7: Finalización....3
- Implementacion de TDA’s....3
  - 1. TDA Pila....4
    - Uso en el juego:....4
  - 2. TDA Cola....4
    - Uso en el juego:....4
  - 3. TDA Lista....4
    - Uso en el juego:....4
- Posibles Problemas....4
  - Casos clave a tener en cuenta:....4
  - Posibles problemas o errores que podrían surgir:....5
  - Cuando no hay suficientes vidas:....6

Cmon dice.

Es una variante del juego "Simón dice", donde los jugadores deben memorizar y reproducir secuencias de colores que se les muestran en pantalla.

Este tiene reglas detalladas, como la cantidad de vidas, tiempos para responder, y la opción de deshacer movimientos usando vidas.

También se especifica que se debe implementar diferentes niveles de dificultad y se tiene que obtener la configuración inicial desde un archivo de texto (`config.txt`).

Algunos puntos clave:

1. **Secuencias de colores**: Los jugadores deben reproducir secuencias formadas por los colores rojo (R), verde (V), amarillo (A) y naranja (N).
2. **Puntajes**: Se otorgan 3 puntos por cada secuencia correctamente ingresada sin usar vidas, y 1 punto si se usaron vidas.
3. **Vidas**: Pueden usarse para deshacer jugadas incorrectas o cuando el jugador no ingresa ninguna secuencia.
4. **Dificultad**: Configurable mediante un archivo externo, afectando los tiempos y cantidad de vidas disponibles.
5. **Informe de resultados**: Al final de cada juego, se genera un informe con los resultados, incluyendo las secuencias, respuestas de los jugadores, y el puntaje obtenido.

El objetivo del proyecto es implementar este juego en C, siguiendo las reglas y condiciones del enunciado.

Propuesta.

Etapa 1: Planificación y Preparación

- **Tarea 1:** Revisión inicial del enunciado y comprensión del proyecto. *Todos los integrantes deben participar.*
- **Tarea 2:** Configurar el repositorio en GitHub y subir el enunciado del trabajo práctico.
- **Tarea 3:** Creación de un archivo `config.txt` de ejemplo con las configuraciones básicas del juego.

Etapa 2: Implementación de las Funciones Básicas

- **Tarea 4:** Implementación de la lectura y validación del archivo `config.txt`.
- **Tarea 5:** Diseño de la estructura para manejar jugadores (nombres, puntajes, vidas).

- **Tarea 6:** Implementación de la secuencia de colores aleatorios usando la API de números aleatorios.

Nota: Estas tres tareas pueden ejecutarse en paralelo, ya que no dependen entre sí.

Etapa 3: Manejo del Flujo de Juego

- **Tarea 7:** Implementación de la lógica del turno de los jugadores, mostrando secuencias y validando la respuesta.
  Depende de: Tareas 4, 5, y 6.
- **Tarea 8:** Manejo de las vidas (deshacer jugadas y mostrar secuencias según las vidas restantes).
  Depende de: Tareas 4 y 5.

Etapa 4: Cálculo de Puntajes

- **Tarea 9:** Cálculo de puntos según las reglas (3 puntos sin uso de vidas, 1 punto usando vidas).
  Depende de: Tarea 7.

Etapa 5: Generación de Informes y Salida de Datos

- **Tarea 10:** Implementación de la generación de informes con los resultados del juego (incluyendo fecha, hora y resultados de cada jugador).
  Depende de: Tarea 9.

Etapa 6: Pruebas y Documentación

- **Tarea 11:** Diseño y ejecución de casos de prueba documentados (mínimo 8).
  Depende de: Tareas 7, 9, y 10.
- **Tarea 12:** Captura de pantalla de los resultados de las pruebas y subida de la documentación al repositorio.

Etapa 7: Finalización

- **Tarea 13:** Verificación de que el código esté limpio (sin errores ni warnings).
  Depende de: Todas las tareas anteriores.

Implementacion de TDA’s.

Estas son los posibles usos que le podemos dar a cada TDA aprendido hasta hoy 06/10/24.

1. TDA Pila

Se puede utilizar para manejar el ingreso de la secuencia de cada jugador, asi se puede deshacer facilmente los ultimo **n** ingresos, siendo este menor o igual a la cantidad de vidas.

Uso en el juego:

- **Deshacer jugadas**: Cuando un jugador decide usar una vida para deshacer jugadas, puedes almacenar las secuencias en una pila y luego sacar de la pila para recuperar el estado anterior.

2. TDA Cola

Seria util para decidir el orden de los jugadores, ya que cuando un jugador termino su turno, este tendria que ir al “final de la cola”.

Uso en el juego:

- **Turnos de los jugadores**: Cuando se sortea el orden, puedes usar una cola para manejar el ciclo de turnos. Cada vez que un jugador termina su turno, se lo mueve al final de la cola.

3. TDA Lista

Puede usarse para almacenar las configuraciones, las secuencias de jugadas, los puntajes y la informacion de los jugadores.

Uso en el juego:

- **Almacenamiento de secuencias**: Una lista puede almacenar las secuencias de colores/letras generadas para cada jugador.
- **Puntajes**: Puedes llevar el registro de los puntajes de los jugadores en una lista.

Posibles Problemas.

Hay casos en los que puede ocurrir problemas o simplemente no se tienen en cuenta. Por las dudas, se replantean las mismas que estan en el pdf del tp.

Casos clave a tener en cuenta:

1. **Ingresar secuencia correctamente (sin usar vidas):**
    - Si el jugador reproduce la secuencia correcta sin cometer errores, avanza a la siguiente secuencia. En este caso, gana 3 puntos.
2. **Ingresar secuencia incorrecta:**

- Si el jugador ingresa una secuencia incorrecta y aún tiene vidas, el sistema le da la opción de deshacer jugadas y volver atrás. Gana 1 punto si usa una vida y corrige la secuencia. Si no tiene vidas, el juego termina para ese jugador.

3. **No ingresar secuencia a tiempo:**
    - Si el jugador no logra ingresar la secuencia en el tiempo disponible, el sistema indica que ha perdido el turno. Si tiene una vida, puede deshacer el turno y volver atrás. Si no tiene vidas, el turno se pierde por completo y el juego avanza al siguiente jugador.
4. **Uso de vidas para corregir errores:**
    - Cuando el jugador ingresa una secuencia incorrecta y tiene vidas, puede elegir cuántas jugadas deshacer, hasta el límite de las vidas que posee. Luego, debe intentar ingresar la secuencia correcta desde el punto en que se detuvo. Si usa vidas para corregir, la secuencia se muestra nuevamente.
5. **Empate en puntajes:**
    - Si al final de la partida dos o más jugadores tienen la misma cantidad de puntos, ambos jugadores son considerados ganadores.

Posibles problemas o errores que podrían surgir:

1. **Manejo incorrecto del archivo `config.txt`:**
    - Si el archivo tiene errores en los tiempos de configuración (más de 5 vidas o más de 20 segundos para jugar), el programa debe mostrar un mensaje de error y pedir una corrección. Asegúrate de validar el archivo correctamente antes de iniciar el juego.
2. **Sincronización del tiempo:**
    - El sistema debe gestionar bien el tiempo para mostrar la secuencia y el tiempo de respuesta del jugador. Si el temporizador no está bien ajustado, podría ser injusto para el jugador. Esto también podría causar problemas si el tiempo configurado en `config.txt` no es correcto.
3. **Falta de vidas:**
    - Si el jugador intenta deshacer una jugada y no tiene suficientes vidas, no se le debería permitir retroceder. El juego debe terminar para ese jugador y pasar al siguiente. En este caso, el jugador perdería la oportunidad de corregir el error, ya que ha usado todas sus vidas.
4. **Respuestas fuera del tiempo:**

- Si el jugador no responde dentro del tiempo límite y no tiene vidas, el turno se termina automáticamente. Si tiene vidas, el sistema debe darle la opción de deshacer, pero si no las tiene, debe pasar al siguiente jugador.

5. **Errores en la secuencia mostrada vs. ingresada:**
    - El sistema debe asegurarse de que al mostrar las secuencias, estas sean claras y no se sobreescriban. Si el jugador ingresa mal alguna parte, el sistema debe señalar qué secuencia ingresó mal (mostrando cuántas jugadas puede deshacer si tiene vidas disponibles).

Cuando no hay suficientes vidas:

Si un jugador se queda sin vidas, no podrá deshacer sus errores o fallos en el ingreso de la secuencia. Las implicaciones son:

- **Fallo en la secuencia:** El juego termina para ese jugador si ya no le quedan vidas. No podrá corregir errores ni volver atrás.
- **Finalización del turno:** El turno del jugador finaliza y pasa al siguiente. El jugador queda fuera del juego hasta que termine la partida.
- **Pérdida de puntos:** Si el jugador no tiene vidas y comete un error, no podrá ganar puntos adicionales y su puntaje final quedará registrado tal como está en ese momento.

Posibles cabezeras de funciones principales.

1. Menú Principal

`mostrar_menu()`

`Nota: Puede existir mas de uno.`

- **Descripción**: Muestra el menú principal del juego.
- **Parámetros**: Ninguno.
- **Retorno**: Opción seleccionada por el usuario (`int` o `char`).

`cargar_configuracion(const char* archivo_configuracion, tLista* configuraciones)`

`Nota: tLista puede ser cambiada por un vector de tipo tConfig`

- **Descripción**: Carga las configuraciones iniciales del juego desde un archivo de texto y las guarda en una lista.
- **Parámetros**:
    - `const char* archivo_configuracion`: El nombre del archivo de configuración.

    - `tLista* configuraciones`: Lista para almacenar las configuraciones del juego.
- **Retorno**: Ninguno (la lista `configuraciones` se modifica).

2. Gestión de Jugadores

Nota: se puede implementar TDA Lista Circular Simplemente Enlazada

`ingresar_jugadores(tCola* cola_jugadores)`

- **Descripción**: Permite ingresar los nombres de los jugadores en una cola.
- **Parámetros**:
    - `tCola* cola_jugadores`: Cola donde se insertarán los jugadores en orden.
- **Retorno**: devuelve la cantidad de jugadores (int).

`sortear_orden_jugadores(tCola* cola_jugadores, unsigned int cant)`

- **Descripción**: Realiza un sorteo aleatorio para determinar el orden de los jugadores en la cola.
- **Parámetros**:
    - `tCola* cola_jugadores`: Cola que contiene a los jugadores.
- **Retorno**: Ninguno (la cola se modifica con el nuevo orden aleatorio).

3. Gestión del Juego

`mostrar_secuencia(tPila* pila_secuencia, unsigned int tiempo_visualizacion)`

- **Descripción**: Muestra la secuencia de colores/letras almacenada en una pila.
- **Parámetros**:
    - `tPila* pila_secuencia`: Pila que contiene las secuencias de jugadas.
    - `unsigned int tiempo_visualizacion`: Tiempo (en segundos) durante el cual se muestra la secuencia.
- **Retorno**: Ninguno.

`validar_respuesta(tPila* pila_secuencia, tPila* pila_respuesta)`

- **Descripción**: compara las pilas de secuencia y respuesta, para saber si esta mal algun carácter.
- **Parámetros**:
    - `tPila* pila_secuencia`: Pila con la secuencia correcta de letras/colores.
    - `tPila* pila_respuesta`: La secuencia ingresada por el jugador.

- **Retorno**: `int` (1 si es correcta, 0 si es incorrecta).

`gestionar_vidas(tPila* pila_secuencia, int* vidas_restantes)`

`Nota: revisar como funciona el retroceso de la pila_respuesta`

- **Descripción**: Administra el uso de vidas cuando los jugadores cometen errores o se acaba el tiempo.
- **Parámetros**:
  - `tPila* pila_secuencia`: Pila con la secuencia actual.
  - `int* vidas_restantes`: Puntero a la cantidad de vidas que le quedan al jugador.
- **Retorno**: Ninguno.

`calcular_puntaje(int* puntaje_actual, int secuencia_correcta, int vidas_usadas)`

- **Descripción**: Calcula el puntaje del jugador según la secuencia ingresada y vidas usadas. (complementaria)
- **Parámetros**:
  - `int* puntaje_actual`: Puntero al puntaje actual del jugador.
  - `int secuencia_correcta`: 1 si la secuencia es correcta, 0 si no lo es.
  - `int vidas_usadas`: Número de vidas usadas por el jugador.
- **Retorno**: Ninguno.

4. API para Secuencias

generar_num_aleatorio(): a definir

- **Descripción:** Llama a la API para recibir un numero aleatorio entre 5 y 15. (posible archivo de configuracion interna)

`generar_secuencia(tLista* lista_secuencia, int longitud)`

- **Descripción**: Llama a una API externa para generar secuencias aleatorias de numeros, los “castea” a los caracteres de colores y las almacena en una lista.
- **Parámetros**:
  - `tLista* lista_secuencia`: Lista donde se almacenarán las secuencias generadas.
  - `int longitud`: Longitud de la secuencia.
- **Retorno**: Ninguno (modifica la lista de secuencias).

5. Manejo de Archivos

`guardar_turno(FILE *archivo_informe, tTurno* turno_actual)`

- **Descripción**: Guarda en un archivo el registro del turno actual. (no cierra el flujo)
- **Parámetros**:
  - `FILE *archivo_informe`: puntero de tipo FILE* al archivo del informe.
  - `tTurno* turno_actual`: turno actual de la partida
- **Retorno**: retornar errores de archivo.

`cargar_niveles_dificultad(const char* archivo_dificultad, tConfig* configuracion)`

- **Descripción**: Carga desde un archivo los niveles de dificultad del juego y los almacena en una lista. (necesario para hacerlo lo mas generico posible)
- **Parámetros**:
  - `const char* archivo_dificultad`: Nombre del archivo que contiene las configuraciones de dificultad.
  - `tConfig* configuracion`: Lista donde se almacenarán las configuraciones de dificultad.
- **Retorno**: Ninguno.

6. Decoración y Música

`decorar_consola()`

- **Descripción**: Decora la consola utilizando colores o caracteres especiales. Posiblemente detalles visuales o manejo de cmd/terminal.
- **Parámetros**: Ninguno.
- **Retorno**: Ninguno.

`reproducir_musica_fondo(const char* archivo_musica)`

- **Descripción**: Reproduce música de fondo durante el juego. (se puede usar librería windows.h o Simple DirectMedia Layer).
- **Parámetros**:
  - `const char* archivo_musica`: Nombre del archivo de música que se reproducirá.
- **Retorno**: Ninguno.

Estructura de Datos.

Tendriamos que definir los tipos de datos que utilizaremos para manejar la informacion.

Propuestas:

- tConfig:
  - char nivel: nivel de dificultad (F=facil,N=normal,D=dificil)
  - unsigned int tiempo_muestra: tiempo de visualizacion de secuencia completa (en segundos,menor o igual a 20)
  - unsigned int tiempo_turno: tiempo que tiene el jugador para contestar (en segundos,menor o igual a 20)
  - unsigned cant_vidas: cantidad de vidas. (menor o igual a 5)
- tSecuenciaItem (tipo tPila?):
  - int index: un mero entre 1 y 4 inclusive
  - char color: letra que representa el color
- tJugador:
  - char nombre[16]: nombre del jugador
  - unsigned int puntaje_turno: puntaje del turno
  - unsigned int puntaje_total: puntaje total
  - unsigned int vidas_consumidas: cantidad de vidas consumidas
  - int vidas_total: cantidad de vidas total
- tTurno:
  - tJugador * jugador_actual: jugador actual
  - const char * o tPila * secuencia: secuencia del turno

Definicion de secuencia.

- Se solicita a la API una longitud para la secuencia (la misma para todos los jugadores, esta indica la “cantidad de turnos”).
- La secuencia se ingresa/recibe una letra a la vez, se muestra en pantalla.
- El jugador debe reingresar toda la secuencia en cada turno (va aumentando).
- El juego debe indicar al usuario si se equivoco al ingresar un carácter de la secuencia para usar una vida y/o saltar el turno.