UNIVERSIDAD
COMPLUTENSE
MADRID

Fundamentos de la Programación

Implementación de la Práctica 3

Índice

1. Introducción

- ✓ Programa en C++ para jugar a un juego inspirado en **robot turtles**: www.robotturtles.com/instructions

- ✓ Cada jugador está representado por una tortuga que se mueve por el tablero, hasta que algún jugador alcanza una joya.

- ✓ Antes de comenzar el juego han de colocarse los siguientes elementos en el **tablero de 8x8**:
  - Una tortuga asociada a cada jugador (de 1 a 4), que mira en una cierta dirección (norte, sur, este, oeste).
  - Tantas joyas como jugadores haya.
  - Entre 0 y 20 muros de piedra. Estos muros son fijos e impiden avanzar a las tortugas.
  - Entre 0 y 12 bloques de hielo. Estos bloques también impiden avanzar a las tortugas pero se pueden derretir si se usa una pistola láser.
  - Entre 0 y 8 cajas. Las cajas se pueden empujar en la dirección del movimiento de la tortuga siempre que haya una casilla libre justo detrás en esa dirección donde ponerla. Al empujar la caja, tanto la caja como la tortuga avanzan una casilla.
  - El resto de casillas estarán vacías

- ✓ Cada **jugador** (máximo 4 jugadores):
  - ✓ Tiene un nombre
  - ✓ Está representado por una tortuga que en cada momento está en una cierta posición (fila, columna) del tablero
  - ✓ Dispone de un juego de cartas llamado **mano** con las que poder realizar acciones por el tablero y otro llamado **mazo** de donde tomar más cartas para incorporar a su mano (una a una).

- El **mazo inicial de un jugador** consta de 38 cartas que se mezclan de manera aleatoria.
  - 18 cartas “Avanza”
  - 8 cartas “Gira a la derecha”
  - 8 cartas “Gira a la izquierda”
  - 4 cartas “Pistola láser”
- La **mano inicial de un jugador** se forma al robar las tres cartas de la parte superior de su mazo inicial.

- ✓ Cuatro tipos de **cartas**:
  - Carta “Avanza”: permite a la tortuga avanzar una casilla en la dirección en la que mira si dicho movimiento es posible. La tortuga no puede salirse del tablero, ni atravesar un muro, ni ocupar el lugar que ya está ocupado por otra tortuga, ni empujar una caja que detrás no tenga una casilla libre, por lo que en esos casos la carta no tendrá efecto.
  - Carta “Gira a la derecha”: permite a la tortuga girar en el sentido de las agujas del reloj permaneciendo en la misma casilla.
  - Carta “Gira a la izquierda”: permite a la tortuga girar en el sentido contrario a las agujas del reloj permaneciendo en la misma casilla.
  - Carta “Pistola láser”: la pistola se dispara en la dirección hacia la que mira la tortuga y golpea lo primero que alcanza en esa dirección. Si es un muro de hielo éste se deshace. Y en cualquier otro caso no sucede nada.

- ✓ Se dispone de un registro de **puntuaciones de los jugadores**:
  - Se conoce el nombre y la puntuación de cada jugador
  - El registro se actualiza tras cada partida
- ✓ El tablero de juego y el registro de puntuaciones se cargan de archivo
- ✓ Los mazos y manos se generarán desde programa

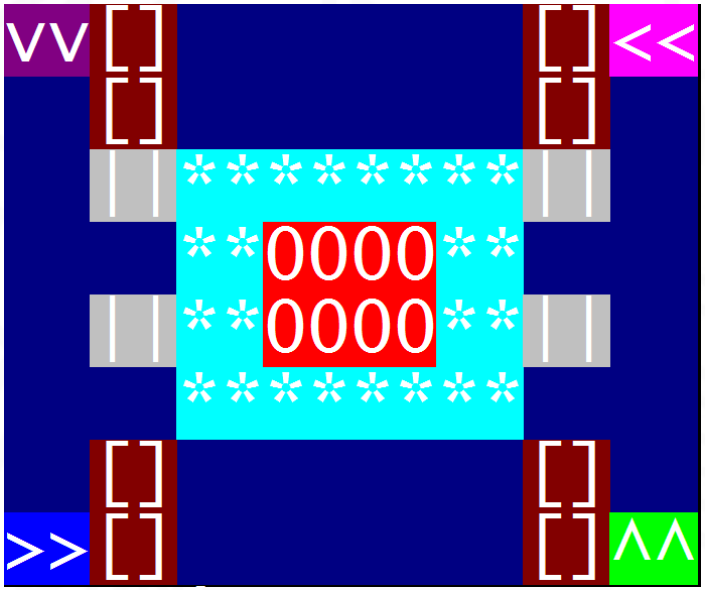
2. Funcionamiento General

- ✓ El programa comienza cargando la información de puntuaciones del fichero *puntuaciones.txt*.
- ✓ A continuación se muestra el siguiente menú (comportamiento cíclico):

```
1. Jugar
2. Mostrar puntuaciones
0. Salir
```

- ✓ Opción 1: el programa solicita el nombre del fichero de tableros, el número de jugadores y sus nombres, y carga el tablero correspondiente del fichero. Se generan los mazos y manos iniciales de los jugadores. Se juega la partida, hasta que alguien alcanza una joya.
  El jugador que gana la partida obtiene una puntuación igual al número de jugadores de la partida y su puntuación se actualiza.
- ✓ Opción 2: el programa muestra el registro de puntuaciones
- ✓ Opción 0: se actualiza el archivo *puntuaciones.txt* y finaliza el programa.

- ✓ Si se elige jugar, el programa muestra el estado del tablero de juego, los jugadores con sus manos y quién tiene el turno. Los jugadores juegan en orden creciente de número.




```
JUGADORES:

      1. Marina: 3 ^  3 <  2 >  2 ~
> 2. Alejandro: 4 ^  2 <  1 >  2 ~
```

- ✓ En su turno el jugador puede elegir entre:
  - ✓ Robar una carta del mazo (**R**): se quita la primera carta de su mazo y se añade a su mano.
  - Crear una secuencia de cartas a partir de su mano y ejecutarla (**E**)
    - Se solicita al usuario la secuencia que se desea crear.
    - Es necesario comprobar que la mano permite generar dicha secuencia.
    - La ejecución de la secuencia modifica las cartas de la mano (eliminando las cartas seleccionadas), el mazo (devuelve las cartas usadas al mismo) y el estado del tablero (con el movimiento de las tortugas, etc.)

3. Consideraciones de Diseño.

- ✓ Generaremos al menos 3 módulos (asociados a tipos de datos) para la solución:
  - ✓ **puntuaciones**: representación del registro de puntuaciones y funcionalidades asociadas
  - ✓ **secuencia**: representación de mazos y secuencias de cartas y funcionalidades asociadas
  - ✓ **juego**: representación de tablero y jugadores y funcionalidades asociadas
- ✓ Podemos incluir un cuarto módulo **dibujo** (de utilidades) que recoja las funcionalidades relacionadas con la interfaz

4. Módulo Puntuaciones: representación del registro de puntuaciones y funcionalidades asociadas

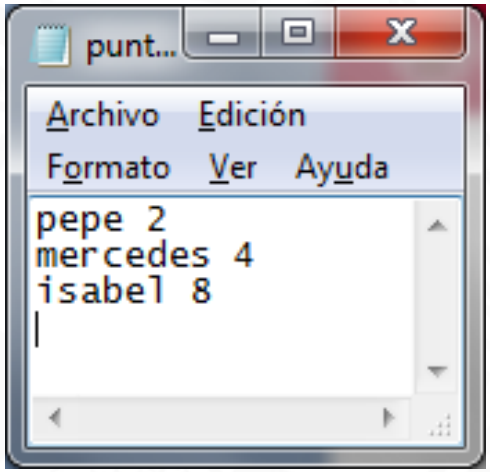
```
const int MAX_JUGADORES_HISTORIAL = 4;
 // poner la dimensión que se desee a la lista

typedef struct {
      string nombre;
      int puntuacion;} tPuntuacionJugador;

typedef struct {
      tPuntuacionJugador puntuaciones[MAX_JUGADORES_HISTORIAL];
      int num_jugadores;} tPuntuaciones;
```

- ✓ Incluye al menos los siguientes subprogramas públicos:
  - **bool cargarPuntuaciones(tPuntuaciones &puntos)**: carga las puntuaciones de los jugadores a partir del fichero.
  - **void guardarPuntuaciones(const tPuntuaciones & puntos)**: guarda las puntuaciones de los jugadores en el fichero.
  - **void mostrarPuntuaciones(const tPuntuaciones & puntos)**: muestra las puntuaciones de los jugadores.
  - **bool actualizarPuntuacion(tPuntuaciones & puntos, const string & nombre, int nuevos)**: si el jugador ya estaba, se incrementan sus puntos en nuevos puntos; si no está en el listado lo inserta con los nuevos puntos (si no hay espacio, se insertará solamente si hay algún jugador en el listado con menos puntuación que él, y en tal caso ocupará el lugar del que menos puntos tiene). Devuelve false si no se ha podido insertar.

- ✓ Ten en cuenta el formato del archivo puntuaciones.txt para implementar
  - **bool cargarPuntuaciones(tPuntuaciones &puntos)**
  - **void guardarPuntuaciones(const tPuntuaciones & puntos)**



5. Módulo Secuencia: representación de mazos y, en general, secuencias de cartas, y funcionalidades asociadas

```
const int MAX_CARTAS_MAZO = 38; // cuántas cartas en total
const int NUM_CARTAS_AVANZA = 18; // cuántas cartas avanza
const int NUM_CARTAS_IZQUIERDA = 8; // cuántas cartas girar izda
const int NUM_CARTAS_DERECHA = 8; // cuántas cartas girar dcha
const int NUM_CARTAS_LASER = 4; // cuántas cartas láser
const int NUM_TIPOS_CARTAS = 4;  // cuántos tipos de cartas 4 = avanza, izda, dcha, laser

typedef enum {avanzar,giroIzquierda,giroDerecha,laser} tCarta;
typedef struct{
        tCarta lista[MAX_CARTAS_MAZO];
        int numcartas; } tMazo;
```

- ✓ Incluye al menos los siguientes subprogramas públicos:
  - **void crearVacia(tMazo & mazo)**: crea una secuencia vacía de cartas.
  - **bool sacar(tMazo & mazo, tCarta& carta)**: saca la primera carta de la secuencia. Devuelve true si se ha podido sacar la carta o false en caso contrario.
  - **void insertar(tMazo & mazo, tCarta carta)**: inserta la carta al final de la secuencia.
  - **void crearMazoAleatorio(tMazo & mazo)**: crea un mazo aleatorio. Para ello rellenamos un mazo inicial con las cartas disponibles y aplicamos el método random_shuffle de la librería <algorithm> para generar una mezcla aleatoria del mismo.

- ✓ Si no implementas un módulo Dibujo para la interfaz, tendrás que incluir aquí subprogramas públicos para mostrar una carta, mostrar un mazo, …

6. Módulo Juego: representación de tablero y jugadores y funcionalidades asociadas

- ✓ Define constantes y tipos relacionados con el tablero.

```
const int  NUM_FILAS = 8; // alto tablero
const int  NUM_COLUMNAS = 8;  // ancho tablero
typedef enum {norte, este, sur, oeste} tDir;
typedef enum  {vacia, hielo, muro, caja, joya, tortuga} tEstadoCasilla;
typedef struct {
            int numero; // asocia tortuga com jugador
            tDir direccion;} tTortuga;
typedef struct {
            tEstadoCasilla estado;
            tTortuga tortuga;} tCasilla;

typedef tCasilla tTablero[NUM_FILAS][NUM_COLUMNAS];
```

- ✓ Define constantes y tipos relacionados con los jugadores.

```
typedef int tMano[NUM_TIPOS_CARTAS]; // representa cuantas cartas de cada tipo hay en la mano
typedef struct {
        int fila;
        int columna;} tCoordenada;
typedef struct {
        string nombre;
        tMazo mazo;
        tMano mano;
        tCoordenada posActual; } tJugador;
```

- ✓ Define un enumerado *tAccion* para representar las posibles acciones de los jugadores.

```
typedef enum {robarCarta, ejecutarSecuencia} tAccion;
```

- ✓ Define un tipo que aglutine todos los aspectos del juego (tablero, lista de jugadores, y quién tiene el turno).

```
const int MAX_JUGADORES = 4; // nº jugadores
typedef struct {
        tTablero tableroJuego;
        int nJugadores;
        tJugador jugadores[MAX_JUGADORES];
        int turno;} tJuego;
```

- ✓ Define un tipo enumerado *tTecla* para representar las teclas leídas al formar secuencia a ejecutar

```
typedef enum { AVANZA, DERECHA, IZQUIERDA, LASER, SALIR, NADA} tTecla;
```

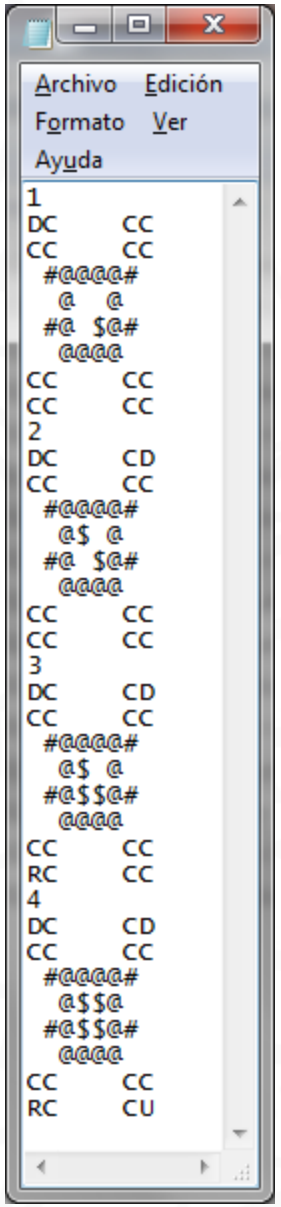
- ✓ Si no vas a implementar un módulo Dibujo para la interfaz, define aquí de forma privada una paleta de colores constante para las casillas, los jugadores y las cartas.

```
const int  NUM_TIPOS_CASILLAS = 6;  // nº de tipos de estado de casilla
const int NUM_COLORES =  NUM_TIPOS_CASILLAS + MAX_JUGADORES;
const int PALETA[NUM_COLORES] = { 1, 11, 7, 4, 12, 5, 13, 9, 10, 3 };
```

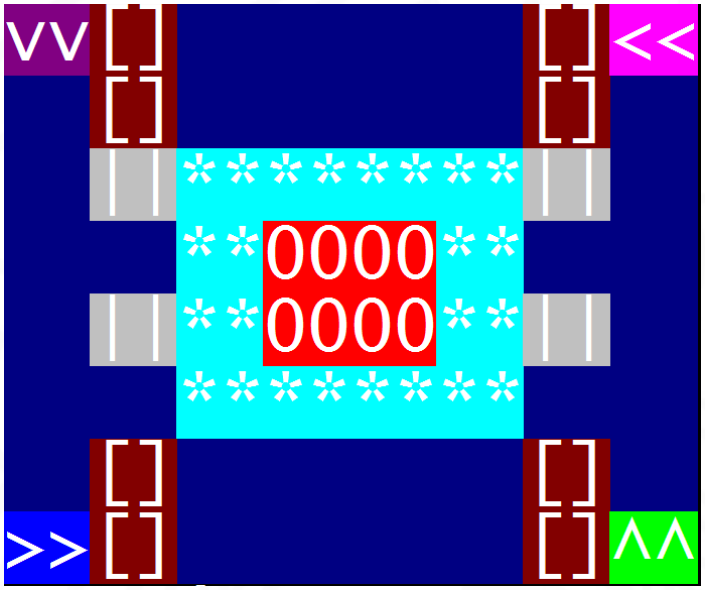
- ✓ Incluye al menos los siguientes subprogramas públicos:
  - **bool crearJuego(tJuego & juego):** solicita al usuario el nombre del fichero y el número de jugadores, y carga el tablero correspondiente desde ese fichero. También inicializa los mazos y manos de los jugadores.
  - **bool ejecutarTurno (tJuego &juego)**: lee la acción del jugador actual y la ejecuta. El booleano indica si el jugador que tiene el turno ha alcanzado una joya.
  - **bool accionRobar(tJuego & juego)**: el jugador con el turno roba una carta de su mazo si hay cartas disponibles. El booleano indica si ha sido posible robar la carta.
  - **bool accionSecuencia(tJuego & juego, tMazo & cartas)**: el jugador con el turno ejecuta la secuencia de cartas dada. El segundo parámetro contiene un subconjunto de las cartas que hay en la mano del jugador actual y se va consumiendo a medida que se ejecuta. El booleano indica si el jugador que tiene el turno ha alcanzado una joya en la ejecución de esta secuencia.
  - **void cambiarTurno (tJuego &juego)**: cambia el turno al jugador siguiente.

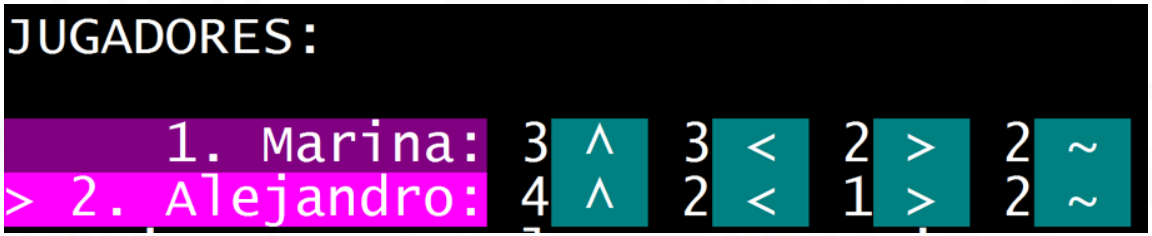
- ✓ Si no implementas un módulo Dibujo para la interfaz, tendrás que incluir aquí subprograma público para mostrar el estado del juego (tablero, manos y quién tiene el turno).
  - **void mostrarJuego(const tJuego & juego):** visualiza el estado actual del juego.

- Carga del tablero
  - desde archivo de texto que contiene 4 tableros, un tablero para cada número de jugadores que puede haber
  - ’#’ representa muro de piedra, ’@’ es muro de hielo, ’ ’ (blanco) es casilla vacía, ’$’ es joya y ’C’ es caja .
  - para representar una tortuga aparece una letra que indica la dirección en la que mira U, D, R, L (up, down, right y left).
  - Los jugadores se van añadiendo en el orden en que los encontramos al leer los datos de arriba abajo y de izquierda a derecha.



- ✓ Visualización del estado del juego





- Cada vez que visualices el estado del juego, borra primero el contenido de la ventana de la consola, de forma que siempre se muestren los elementos del juego en la misma posición.
  Para borrar la consola utiliza:

```
system("cls");
```

- Colores
  - Color del primer plano (con el que se muestran los trazos de los caracteres) por defecto: blanco
  - Color de fondo por defecto: negro.
  - Pero disponemos de 16 colores diferentes (con valores de 0 a 15). El 0 es el negro y el 15 el blanco Los demás son azul, verde, cian, rojo, magenta, amarillo y gris, en dos versiones, oscuro y claro.

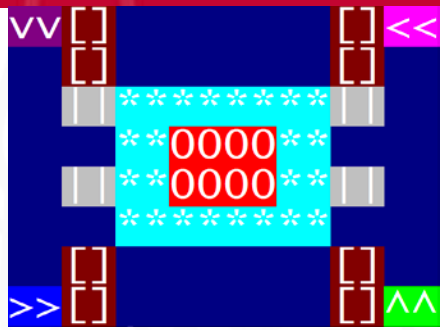
- Visual Studio incluye una biblioteca, *Windows.h*, que tiene, entre otras, rutinas para la consola. Una de ellas es *SetConsoleTextAttribute()*, que permite ajustar los colores de fondo y primer plano.

  Incluye en el módulo Juego (o en el módulo Dibujo si lo tienes) esa biblioteca y el procedimiento que aparece a continuación:

```
void colorFondo(int colorFondo){
HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
SetConsoleTextAttribute(handle, 15 | (colorFondo << 4));
}
```

  Si invocas al procedimiento con un color para el fondo (1 a 14) lo establecerá, con el color de primer plano en blanco (15)

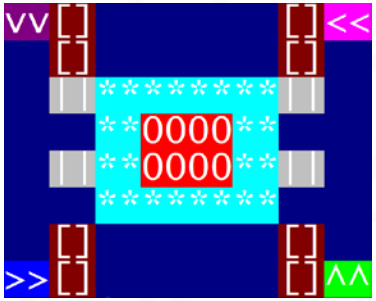
- Debes cambiar el color de fondo cada vez que tengas que dibujar una casilla y volverlo a poner a negro (0) a continuación.

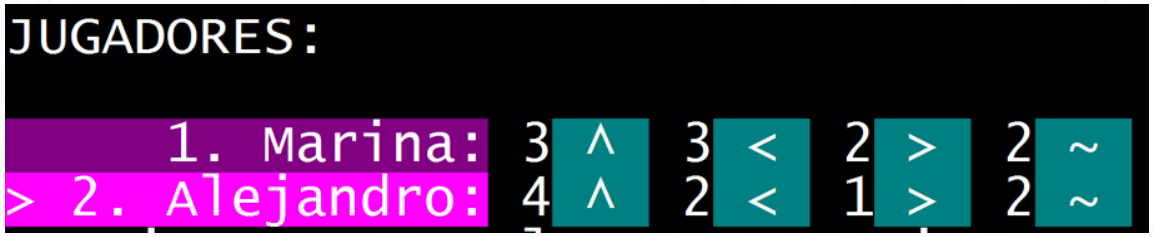


- ✓ Visualización del tablero
  - ✓ Cada tortuga tiene un color de fondo distinto, y diferente de los demás elementos del juego
  - ✓ Usa el gris para los muros de piedra, rojo para las joyas, azul clarito para el hielo y granate para las cajas.
  - ✓ Usa || para los muros de piedra, ** para los muros de hielo, [] para las cajas, 00 para las joyas; y para las tortugas que miran en las cuatro direcciones posibles usa >>, <<, ^^ y vv.
  - ✓ La paleta de colores te facilitará el conocer los colores de los 10 elementos donde haga falta (en este orden: casilla vacía, hielo, muro, caja, joya, tortuga1, tortuga2, tortuga3, tortuga4, cartamano)


```
const int  NUM_TIPOS_CASILLAS = 6;  // nº de tipos de estado de casilla
const int NUM_COLORES =  NUM_TIPOS_CASILLAS + MAX_JUGADORES;
const int PALETA[NUM_COLORES] = { 1, 11, 7, 4, 12, 5, 13, 9, 10, 3 };
```

- ✓ Visualización de las manos de los jugadores
  - ✓ Se mostrarán al lado de su nombre (en el color correspondiente) por orden, apareciendo un símbolo > delante del que tiene el turno.
  - ✓ Las cartas se visualizarán con un símbolo del tipo de carta y al lado el número de cartas de ese tipo: ^ para las cartas avanzar, < para girar a la izquierda, > para girar a la derecha, ~ para el láser.





- ✓ Obtención de la secuencia de cartas a ejecutar
  - Se solicita al usuario la secuencia que se desea crear, mediante las teclas de dirección (↑ para avanzar, → para girar a la derecha, ← para girar a la izquierda) y espacio para el láser. La tecla ENTER marca el fin de la secuencia.
  - Es necesario comprobar que la mano permite generar dicha secuencia.
  - La tecla ENTER sí genera un código ASCII (13), al igual que el espacio (32), pero las de dirección no tienen códigos ASCII asociados (son teclas especiales). Cuando se pulsan en realidad se generan dos códigos, uno que indica que se trata de una tecla especial y un segundo que indica de cuál se trata.

- Las teclas especiales generan dos códigos y no se pueden leer con *get()*. Podemos leerlas con *_getch()*, que devuelve un entero; si el entero devuelto corresponde a una tecla especial se llama una segunda vez para obtener el código.
- Implementa una función leerTecla que devuelva la tecla leída (el uso de *getch()* requerirá la biblioteca conio.h)

```
cin.sync();   int tecla = _getch();
if (tecla == 0xe0) {// Tecla especial
            tecla = _getch();
            switch (tecla) {
            case 72:  // devolver AVANZA
            case 77: // devolver DERECHA
            case 75: // devolver IZQUIERDA
            default: // devolver NADA
}}
else if (tecla == 32) // devolver LASER
else if (tecla==13) // devolver SALIR
else // devolver NADA
```

7. Programa Principal.

- ✓ Cargar puntuaciones y si la carga es correcta
  - ✓ Repetir:
    - ✓ Mostrar el menú y leer la opción
      - ✓ Si la opción es 1
        - ✓ Crear el juego y si la creación es correcta
          - ✓ Mientras no haya un ganador de la partida
            - ✓ Mostrar el juego
            - ✓ Ejecutar turno
            - ✓ Si hay ganador actualizar puntuaciones sino cambiar el turno
      - ✓ Si la opción es 2
        - ✓ Mostrar las puntuaciones

    hasta que el usuario elija salir
  - ✓ Guardar las puntuaciones

8.¿Por dónde empezamos?

- ✓ Empezaremos desarrollando los módulos con menos dependencias
  1. Puntuaciones
  2. Secuencia
  3. Juego (o juego y dibujo)
- ✓ Por último el main.cpp
- ✓ IMPORTANTE… Prueba exhaustivamente cada módulo por separado: haz un main en cada.cpp