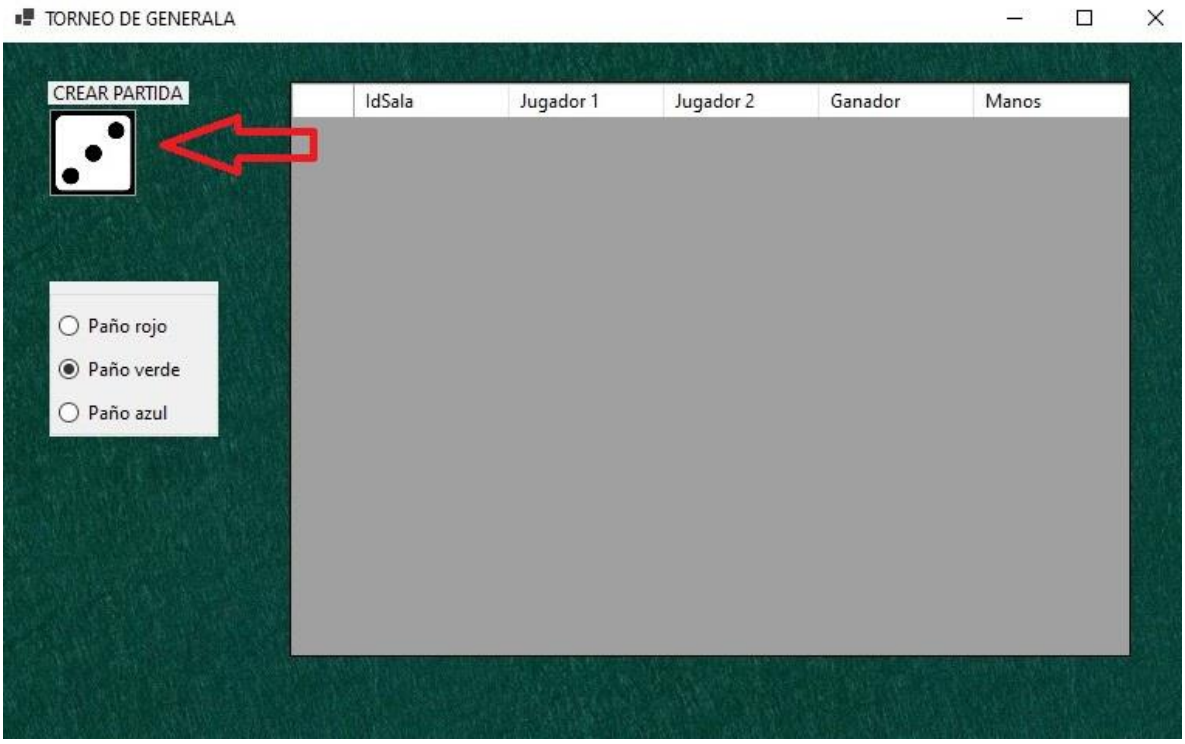
# TP N°2 (2°D)

GENERALA

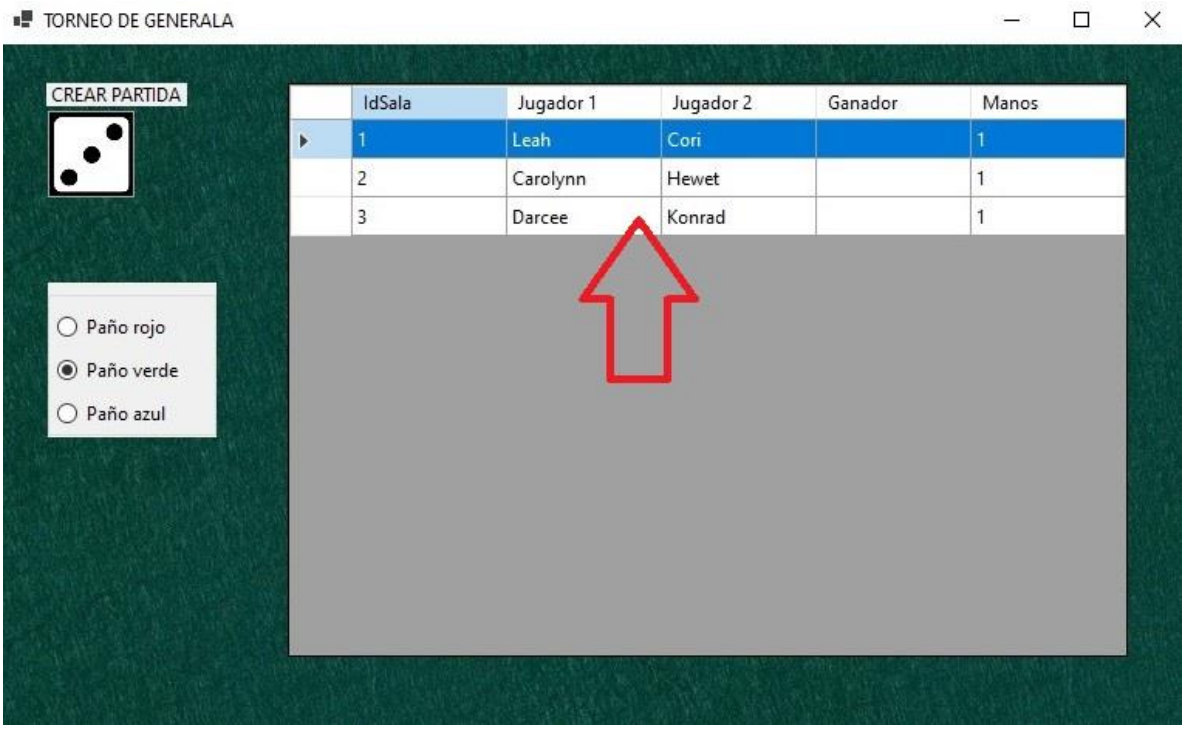
Sosa Maximiliano

La aplicación inicia con la ventana de partidas.

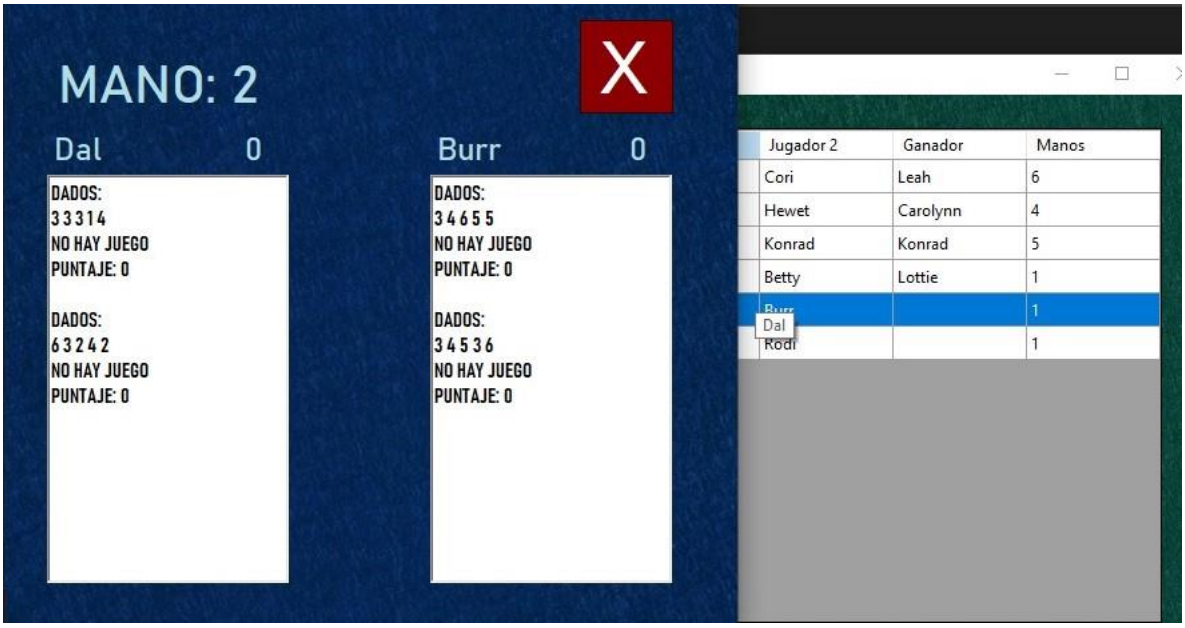
Las partidas se inician con el botón **CREAR PARTIDA**.



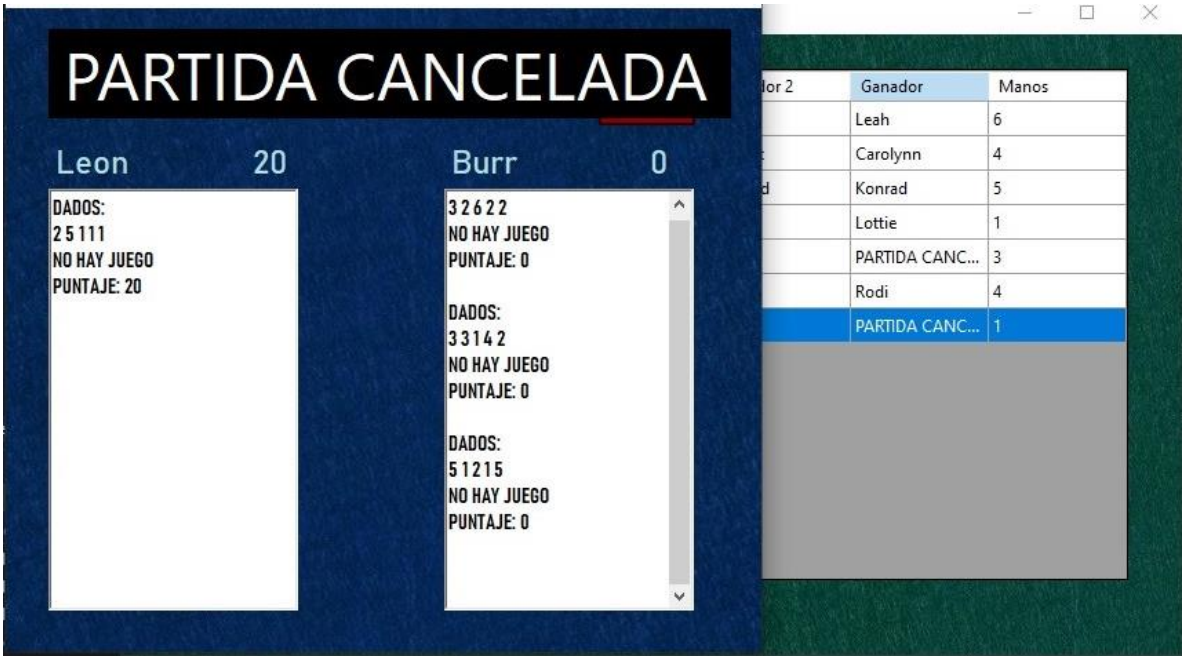

Las partidas iniciadas se visualizan en la tabla de partidas



Para ver el desarrollo de una partida se hace doble click sobre la fila



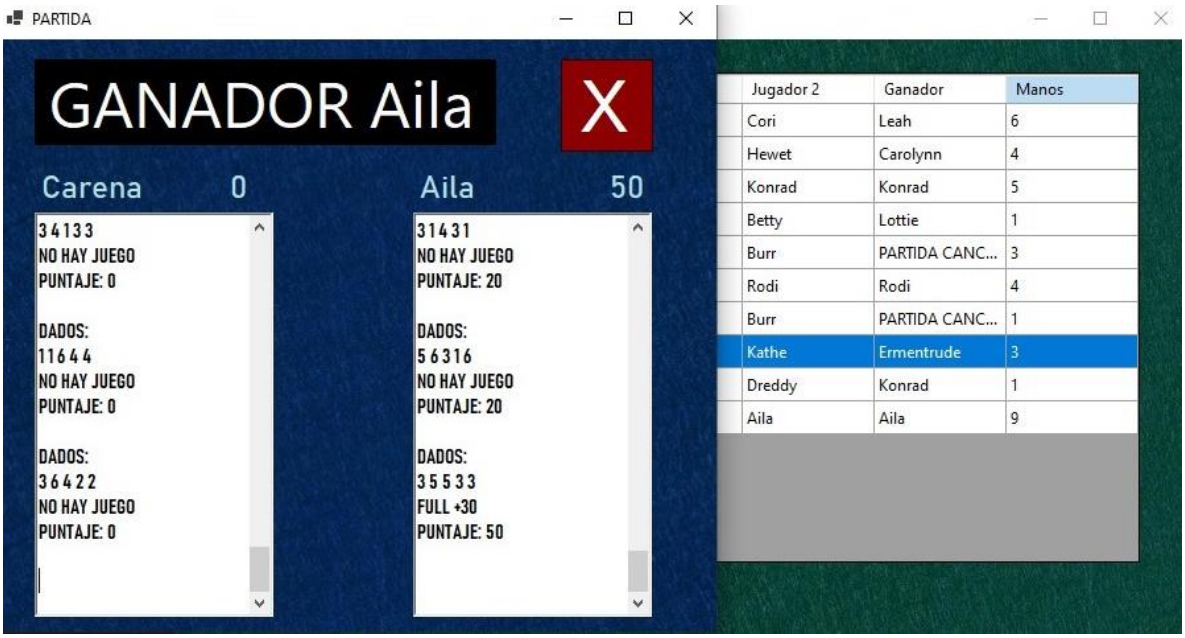

Para cancelar una partida en marcha se presiona el botón rojo




Al terminar una partida se crea en el escritorio una carpeta con un archivo .json y una carpeta con un archivo .txt que contiene la lista de los jugadores ganadores de cada partida

Si un jugador gana la partida se muestra la leyenda **GANADOR**.

Gana el jugador que llega a >= 100 puntos.




## APLICACIÓN DE LOS EJERCICIOS:

## SQL

Creé una clase **ParticipantesDB** que se conecta con la base de datos y trae a través del método **ObtenerJugadores** una lista de jugadores.

```
namespace Entidades
{
    4 referencias
    public class ParticipantesDB
    {

        string connectionString;
        SqlConnection connection;
        SqlCommand command;

        1 referencia
        public List<Jugador> ObtenerJugadores()
        {
            List<Jugador> participantes = new List<Jugador>();

            try
            {
                connectionString = "Server = .; Database = SegundoTP; Trusted_Connection = True; Encrypt = False;";
                connection = new SqlConnection(connectionString);

                command = new SqlCommand();

                connection.Open();

                command.Connection = connection;

                command.CommandType = System.Data.CommandType.Text;

                command.CommandText = "SELECT * FROM Participantes3";

                SqlDataReader reader = command.ExecuteReader();


                while (reader.Read())
                {
                    int id = reader.GetInt32(0);
                    string nombre = reader.GetString(1);
                    string apellido = reader.GetString(2);
```

**DELEGADOS**

En la clase Delegados declaro tres delegados propios.

```
7   namespace Entidades
8   {
        3 referencias
9       public static class Delegado
10      {
11          public delegate void Tirar(Jugador jugador1, Jugador jugador2);
12
13          public delegate bool Ganador(Jugador jugador);
14
15          public delegate void MostrarJuego(string texto, string texto2);
16      }
```

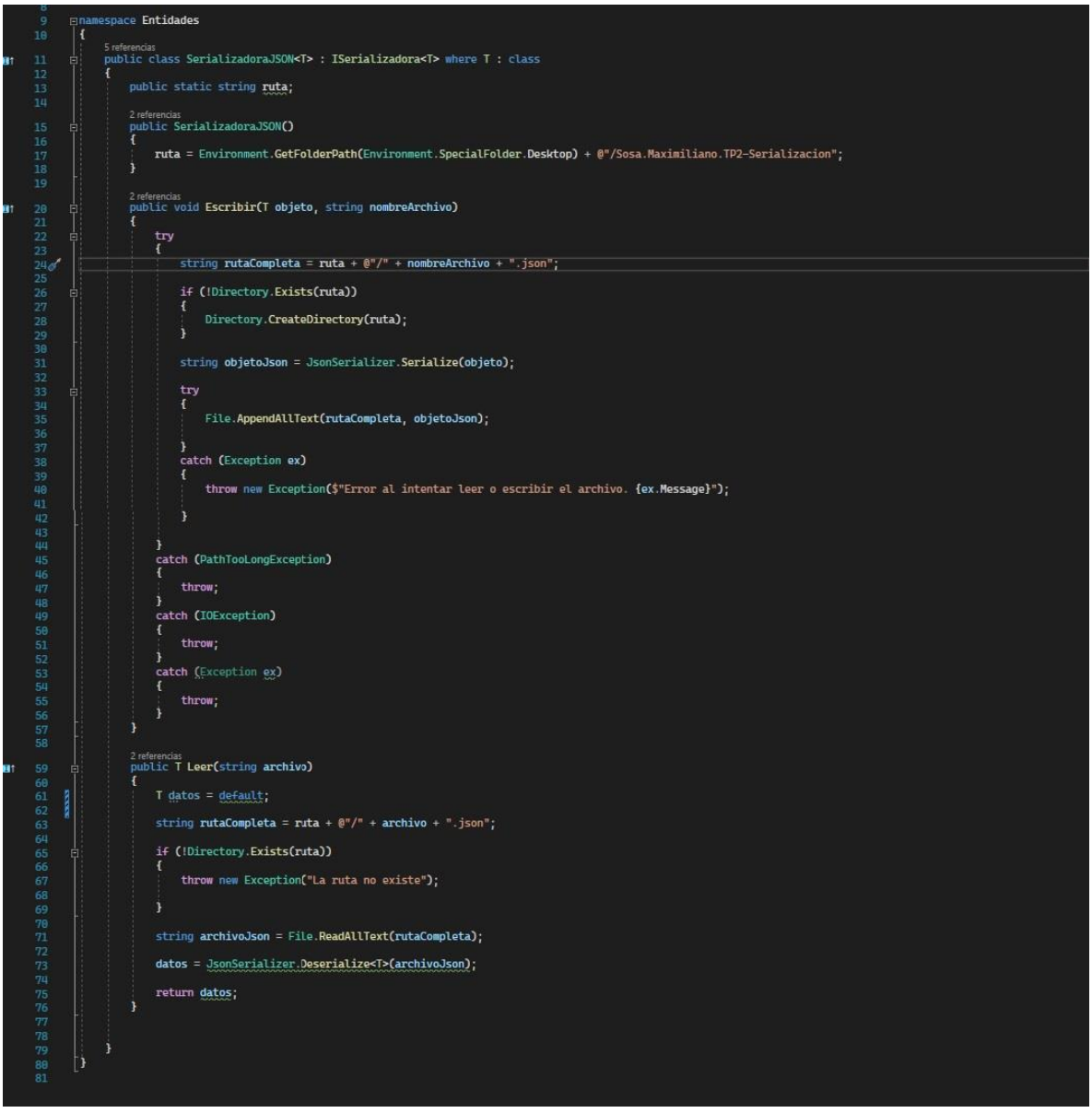
Les asigno los métodos.

```
16      public partial class FrmPartida : Form
17      {
18          Sala sala;
19          public int mano;
20          Delegado.Ganador DGanador;
21          Delegado.MostrarJuego DMostrarJuego;
22          public Action<string> cancelada;
23          public int idForm;
24
            1 referencia
25          public FrmPartida()
26          {
27              InitializeComponent();
28
29              mano = 0;
30              DMostrarJuego = ActualizarDatos;
31
32              sala = new Sala(DMostrarJuego);
33              sala.IdSala = FrmPrincipal.idFormPartida;
34              idForm = FrmPrincipal.idFormPartida;
35              Torneo.Salas.Add(sala);
36              cancelada = Mostrar;
37              sala.PartidaTerminada += cancelada;
38              DGanador = Sala.HayGanador;
39
```

## EXCEPCIONES, INTERFACES, GENERICS, SERIALIZACION

Tengo dos clases **SERIALIZADORAS** que implementan una **INTERFAZ GENÉRICA** que contiene dos firmas de métodos que son **Escribir** y **Leer**.

Se lanzan **EXCEPCIONES** en caso de que no pueda escribir en el archivo o si la ruta de lectura no existe.



```
namespace Entidades
{
    5 referencias
    public class SerializadoraJSON<T> : ISerializadora<T> where T : class
    {
        public static string ruta;

        2 referencias
        public SerializadoraJSON()
        {
            ruta = Environment.GetFolderPath(Environment.SpecialFolder.Desktop) + @"/Sosa.Maximiliano.TP2-Serializacion";
        }

        2 referencias
        public void Escribir(T objeto, string nombreArchivo)
        {
            try
            {
                string rutaCompleta = ruta + @"/" + nombreArchivo + ".json";

                if (!Directory.Exists(ruta))
                {
                    Directory.CreateDirectory(ruta);
                }

                string objetoJson = JsonSerializer.Serialize(objeto);

                try
                {
                    File.AppendAllText(rutaCompleta, objetoJson);

                }
                catch (Exception ex)
                {
                    throw new Exception($"Error al intentar leer o escribir el archivo. {ex.Message}");

                }

            }
            catch (PathTooLongException)
            {
                throw;
            }
            catch (IOException)
            {
                throw;
            }
            catch (Exception ex)
            {
                throw;
            }
        }

        2 referencias
        public T Leer(string archivo)
        {
            T datos = default;

            string rutaCompleta = ruta + @"/" + archivo + ".json";

            if (!Directory.Exists(ruta))
            {
                throw new Exception("La ruta no existe");

            }

            string archivoJson = File.ReadAllText(rutaCompleta);

            datos = JsonSerializer.Deserialize<T>(archivoJson);

            return datos;
        }


    }
}
```

## UNIT TESTING

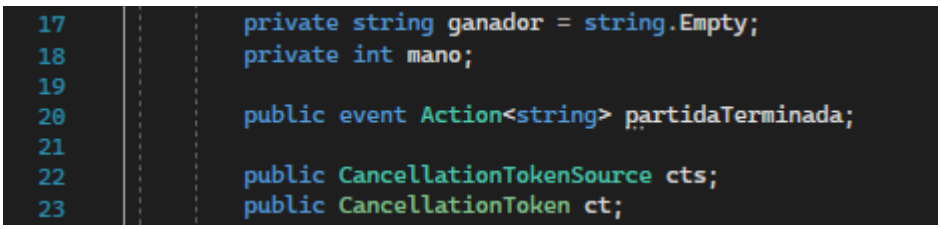
Hay una clase de test unitarios aplicados a varios métodos de diferentes clases.

```
namespace Entidades.Test
{
    [TestClass]
    0 referencias
    public class ReglasDeberia
    {

        [DataRow(1, 2, 3, 4, 5)]
        [DataRow(1, 3, 5, 2, 4)]
        [DataRow(5, 4, 3, 2, 1)]
        [TestMethod]
        0 referencias
        public void ValidarEscaleraMenor(int dado1, int dado2, int dado3, int dado4, int dado5)
        {
            List<int> dados = new List<int>();

            dados.Add(dado1);
            dados.Add(dado2);
            dados.Add(dado3);
            dados.Add(dado4);
            dados.Add(dado5);

            bool resultadoEsperado = true;
            bool resultado = Reglas.EscaleraMenor(dados);
            Assert.AreEqual(resultadoEsperado, resultado);
        }

        [DataRow(2, 3, 4, 5, 6)]
        [DataRow(2, 4, 6, 3, 5)]
        [DataRow(6, 5, 4, 3, 2)]
        [TestMethod]
        0 referencias
        public void ValidarEscaleraMayor(int dado1, int dado2, int dado3, int dado4, int dado5)
        {
            List<int> dados = new List<int>();

            dados.Add(dado1);
            dados.Add(dado2);
            dados.Add(dado3);
            dados.Add(dado4);
            dados.Add(dado5);
```

## EVENTOS

En la clase **Sala** declaro un evento y lo utilizo para enviar el mensaje de **"PARTIDA CANCELADA"** cuando se presiona el botón.

**(Funciona, pero no sé si está bien aplicado.)**



```
        private string ganador = string.Empty;
        private int mano;

        public event Action<string> partidaTerminada;

        public CancellationTokenSource cts;
        public CancellationToken ct;
```

```
79
80          if (ct.IsCancellationRequested)
81          {
82              partidaTerminada?.Invoke("PARTIDA CANCELADA");
83          }
84
```

## TASK

En el constructor de la clase **Sala** ejecuto una tarea que hace que ni bien se instancie una sala se inicie una partida en otro hilo.

```
1 referencia
25      public Sala(MostrarJuego DMostrar)
26      {
27          dados = new List<int>();
28          for (int i = 0; i < 5; i++)
29          {
30              dados.Add(rand.Next(1, 7));
31          }
32
33          cts = new CancellationTokenSource();
34          ct = cts.Token;
35
36          this.jugador1 = Torneo.Participantes[rand.Next(0, 100)];
37          this.jugador2 = Torneo.Participantes[rand.Next(0, 100)];
38
39          Task task = Task.Run(() => { this.JugarPartida(DMostrar, ct); });
40
41      }
```