

Truco SAS

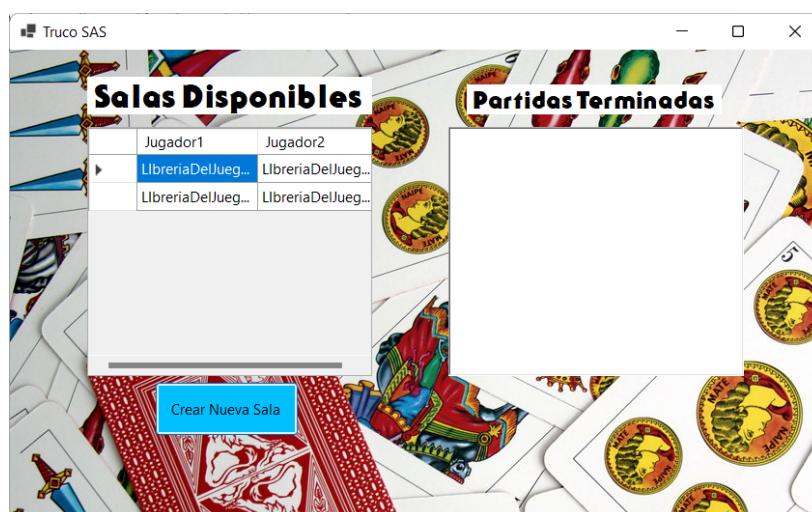
Aplicación para visualizar partidas de truco, generadas automáticamente, con jugadores precargados en Base de Datos Sql, que juegan de forma automática al crear una nueva sala.



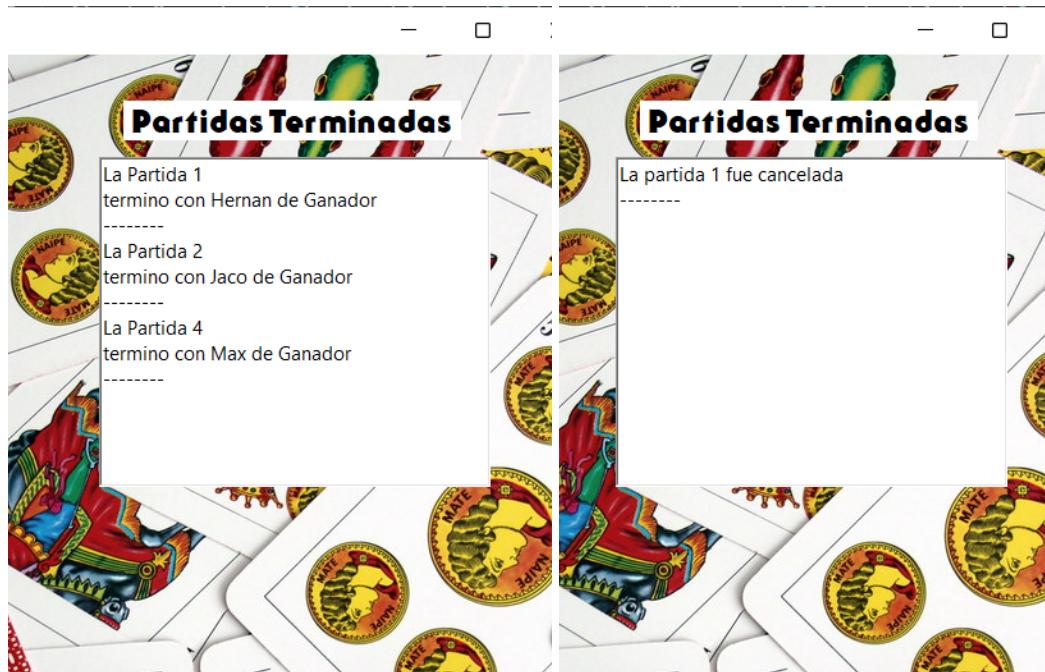
Captura del Inicio de la aplicación vista usuario.

Modo de uso:

Al iniciar la aplicación, las salas y los jugadores disponibles no aparecerán de forma automática. El botón “Crear Nueva Sala” iniciará una partida entre dos jugadores disponibles automáticamente, y se listará la sala en el recuadro de la izquierda. A su vez, a medida que las partidas terminan, sean porque llegaron a su fin, habiendo sido canceladas por el usuario o porque uno de los jugadores superó los 5 puntos, se listan en el recuadro derecho.



Ejemplo de Salas Abiertas



Ejemplo de cuando las partidas terminen

Cuando las partidas terminen, se listan en el RichTextBox de la derecha de la pantalla, sean canceladas o finalizadas. En el último caso se listará el ganador.

Vista de Juego

sala

Sala 1

Jugador 2: Hernan	Juanse Canta Envío Hernan Dice No Quiero Juanse Gana 1 punto por envío no querido El jugador Hernan, canto Truco
Jugador 1: Juanse	El jugador Juanse, dice quiero El jugador Juanse jugó un 3 de espada El jugador Hernan jugó un 2 de copa El jugador Juanse jugó un 7 de basto El jugador Hernan jugó un 1 de oro

Cerrar **Cancelar**

Haciendo Doble Click en un elemento de la lista del DataGridView , se abrirá la vista de la Sala correspondiente a la partida que se está jugando, y se visualiza, las jugadas de cada Jugador.

El botón Cerrar, cierra la ventana de reproducción del juego, pero no lo detiene, permitiendo volverlo a abrir cuando lo considere necesario.



Cuando el número de partidas iniciadas llegue a 8 no le permitirá seguir iniciando hasta que alguna sala termine

**Ejemplos de Excepciones controladas, con un mensaje en la vista.
(Ver detalle de codigo mas adelante)**



Funcionamiento de Código

El Juego Truco SAS está basado en el popular juego de truco argentino, donde se utilizan naipes españoles.

En la lógica del juego, se juega al envido y al truco de la siguiente forma:

Envido: Se comparan los palos de las 3 cartas del jugador, y se determina si canta:

- 1 - Con 3 cartas del mismo palo => Flor
- 2 - Dos cartas del mismo palo y una suma determinada por una preferencia(> a cierta suma)=> Envido
- 3 - Cartas todas diferentes, se devuelve el valor de la carta más alta, sin tener en cuenta las sotas

La lógica decide según la preferencia, si canta envido o simplemente no se canta nada, si un jugador canta, y el otro quiere, se procede a comparar los tantos de ambos jugadores. Si uno de los dos no quiere en envido, se le otorga un punto al jugador que lo canto.

```
//envido solo primera mano
if (nuevaRonda.NroMano == 1)
{
    Thread.Sleep(500);
    jugada1 = jugadorMano.CantarEnvido(26, jugadorMano.CantoEnvido, jugadorMano.CantoFlor);
    Thread.Sleep(500);
    jugada2 = jugadorPie.CantarEnvido(22, jugadorMano.CantoEnvido, jugadorMano.CantoFlor);

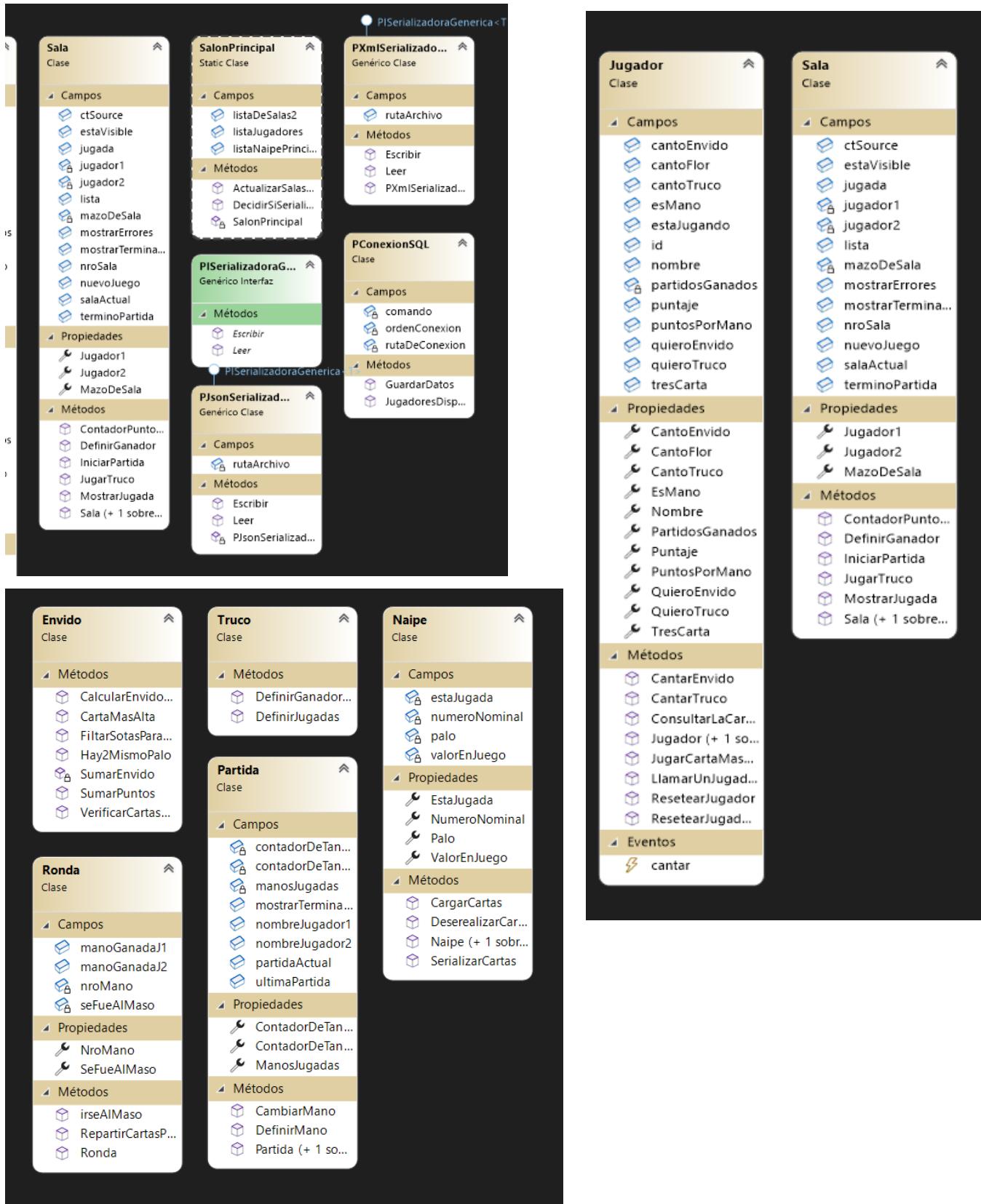
    if (jugada2 == 2)
    {
        Thread.Sleep(1000);
        jugada1 = jugadorMano.CantarEnvido(22, jugadorPie.CantoEnvido, jugadorPie.CantoFlor);
    }
    Envido.SumarPuntos(jugadorMano, jugadorPie, jugada1, jugada2, jugada);
}
```

Código Principal del Envido, en clase Sala

Truco: El truco se asemeja mas a una guerra de cartas, se maneja con una preferencia, al igual que el envido, pero en este caso se coloca el valor de preferencia de la carta mas alta, es decir, si hay una carta que supere ese valor(valor en juego, no el mismo valor de la carta), se canta, o se quiere.

Si se canta y se quiere, al final, el que tiene mayor cantidad de cartas mayores al otro jugador, Gana.
 Si no se canta, el punto va para el que tiene mas cartas mayores que el otro jugador, y, por ultimo,
 Si uno canta y el otro jugador no quiere, el punto va para el jugador que canto el truco, y no se procede a jugar, terminando esa Ronda.

Diagrama De Clases



Manejo de excepciones

A lo largo del programa, se trato de utilizar excepciones en todos los métodos posibles. Para evitar que rompa principalmente con valores nulos, o en para que en la devolucion de los metodos utilizados no existan valores erroneos.

Los Formularios usan Try Catch, y el salon principal muestra los mensajes de error en los label mostrados al principio del Readme.

Algunos ejemplos:

```
/// <summary></summary> Retorna 20 unidades o mas, hasta 33, si no esta dentro de eso
8 referencias | 3/3 pasando
public static int FiltarSotasParaEnvio(Naipe carta1, Naipe carta2)
{
    if (carta1 != null && carta2 != null)
    {
        if(carta1.Equals(carta2))
        {
            throw new Exception("ERROR!\nLas Cartas son de palos distintos");
        }
        else
        {
            if (carta1.Palo.Equals(carta2.Palo))
            {
                int acum = 20;
                if (carta1.NumeroNominal < 8)
                { acum += acum + carta1.NumeroNominal; }

                if (carta2.NumeroNominal < 8)
                { acum += acum + carta2.NumeroNominal; }

                if (acum > 33)
                { throw new Exception("ERROR!\nError! Error en el maso"); }

                return acum;
            }
        }
    }
    throw new Exception("ERROR!\nLas Cartas son iguales");
}
throw new Exception("ERROR!\nNo hay cartas");
}
```

```
1 referencia
private void dtg_listasalas_CellContentDoubleClick(object sender, DataGridViewCellEventArgs e)
{
    indice = -1;

    try
    {
        indice = dtg_listasalas.CurrentRow.Index;
        if (SalonPrincipal.listaDeSalas2[indice].estaVisible == false)
        {
            frm_sala nuevaVistaSala = new frm_sala(SalonPrincipal.listaDeSalas2[indice]);
            nuevaVistaSala.Show();
        }
        else
            throw new Exception("La Sala ya se encuentra en\nmodo vista");
    }
    catch(Exception ex)
    {
        lbl_error.Visible = true;
        lbl_error.Text=ex.Message;
    }
}
```

Incluso se utiliza, para serializar la primera vez que se utiliza el programa en una máquina nueva, el mazo de cartas, si no existe, lo serializa, para luego deserializarlo de la carpeta local de la máquina cada vez que lo necesita.

Clase Salon Principal

```

1 referencia
public static List<Naipes> DecidirSiSerializarCartas()
{
    List<Naipes> listaNaipesPrincipal = new List<Naipes>();
    try
    {
        listaNaipesPrincipal = Naipes.DeserealizarCartas();
    }
    catch(Exception)
    {
        Naipes.SerializarCartas();
        listaNaipesPrincipal = Naipes.DeserealizarCartas();
    }
    return listaNaipesPrincipal;
}

```

Serialization

Se utiliza en el mazo de cartas, con Json, para Serializar la primera vez desde la libreria del sistema, y luego usarlo desde la carpeta local como lo antes detallado.

Interfaz y Genericos

Utilizada para unificar, los métodos Leer y Escribir de las Clases, JsonSerializer y XML serializado

```

3 referencias
public interface PISerializadoraGenerica<T> where T : class, new()
{
    3 referencias
    void Escribir(T objeto, string str);
    3 referencias
    T Leer(string str);
}

```

Las clases JsonSerializer, y PXMLSerializadora son las que utilizan esta interfaz
 (se coloco la letra adelante, a todas las clases que tenian que ver con Persistencia de datos, para poder identificarlas mas rapida y facilmente)

Captura de la utilización de código en la clase

```

2 referencias
public static List<Naipes> DeserializarCartas()
{
    string carta;
    List<Naipes> listaDeCartas = new List<Naipes>();
    for(int i = 1; i < 41; i++)
    {
        PISerializadoraGenerica<Naipes> recuperar = new PJsonSerializadora<Naipes>();
        carta = "cartaNro" + i.ToString();
        Naipes naipe = recuperar.Leer(carta);
        listaDeCartas.Add(naipe);
    }
    return listaDeCartas;
}

1 referencia
public static void SerializarCartas()
{
    int numeroDeCarta = 0;
    string carta;
    List<Naipes> listaDeCartas = new List<Naipes>();
    listaDeCartas = Naipes.CargarCartas();

    PISerializadoraGenerica<Naipes> guardarLosNaipes = new PJsonSerializadora<Naipes>();
    foreach(Naipes unNaipes in listaDeCartas)
    {
        numeroDeCarta++;
        carta = "cartaNro" + numeroDeCarta.ToString();
        guardarLosNaipes.Escribir(unNaipes, carta);
    }
}

```

Uso de Delegados, task, ejemplos de la clase Sala

```

public CancellationTokenSource ctSource;
public Action<string> jugada;
public Task nuevoJuego;
public Action<string> mostrarErrores;
public Action<string> mostrarTerminadas;

```

Uso de Eventos, en Clase Jugador

```
public event Action<string> cantar;
```

Implementacion:

```
Thread.Sleep(500);
cartaMano = jugadorMano.JugarCartaMasAltaEnJuego();
jugada?.Invoke($"El jugador {jugadorMano.Nombre} jugó un {cartaMano.NumeroNominal} de {cartaMano.Palo}\n");

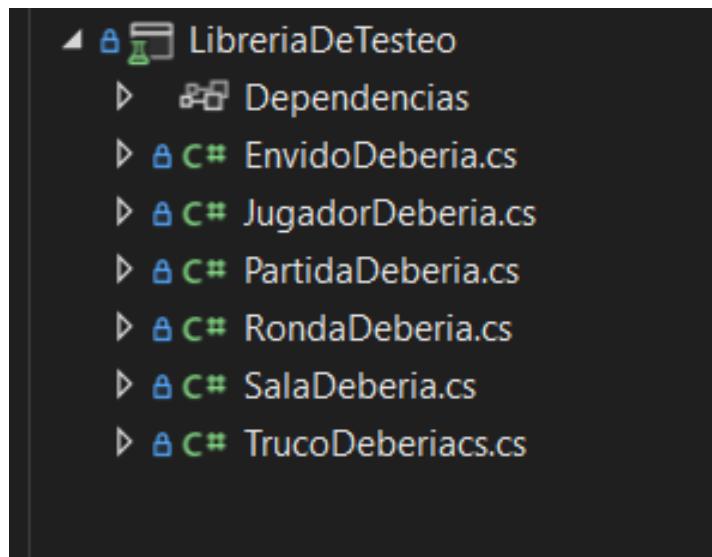
Thread.Sleep(500);
cartaPie = jugadorPie.JugarCartaMasAltaEnJuego();
jugada?.Invoke($"El jugador {jugadorPie.Nombre} jugó un {cartaPie.NumeroNominal} de {cartaPie.Palo}\n\n");

ContadorPuntosMano(cartaMano, cartaPie, jugadorMano, jugadorPie);
```

En la clase Sala, donde se encuentra la lógica del juego, se puede ver el ejemplo del delegado del tipo Action jugada, se colocó el signo "?" debido a que es un delegado que se implementa cuando el formulario de vista es abierto, mientras tanto es Null.

Tambien se ve el uso de la biblioteca Thread, el método Sleep, para frenar la aplicación por un determinado tiempo y simular que la partida se corre en tiempo real.

Unit Test



Se testeán todas las clases, probando métodos el su funcionamiento correcto, y el incorrecto, muchos métodos privados, se pasaron a públicos para poder testearlos también. Se completó un test con 60 testeos.