Curso:

Arquitectura de Software Empresarial

Grupo:

Teoría 1 – Laboratorio B

Tema:

Caso De Estudio: Juego del Jankenpon

Autores y Códigos:

Díaz Diaz Melio Josue	(0201814005)
García Romero Antonio Alfons	(0201814022)
Lujan Rojas Nelvin Eduardo	(0201814007)
Romero Loli Harby Breyner	(0201814025)

Escuela:

Escuela Profesional de Ingeniería de Sistemas e Informática - VI Ciclo

Docente:

Mg. Macedo Alcántara Dayán Fernando

Nuevo Chimbote-Perú

2020

Contenido

Rock Paper Scissors Lizard Spock

1. Caso de Estudio

Se requiere implementar el juego del JanKenPon. Este juego es también llamado PIEDRA PAPEL o TIJERA que consiste en mostrar una elección en forma simultánea entre dos jugadores. EL ganador de la jugada se designará de acuerdo a la siguiente tabla:

ELEMENTO 1	ELEMENTO 2	**GANADOR**
PIEDRA	PAPEL	**PAPEL**
PAPEL	TIJERA	**TIJERA**
PIEDRA	TIJERA	**PIEDRA**

- En caso de que los dos jugadores mostraran el mismo elemento, será un EMPATE
- La partida consistirá de 5 jugadas, el ganador será el que tenga más jugadas ganadas.
    - **Modificación:** Se establecerá un máximo de puntos y cuando uno de los 2 llegue, gana

- El juego debe funcionar tanto en consola, como en escritorio.
- El juego debe considerar la partida entre dos jugadores y también una partida contra la máquina.
    - **Modificación:** Se considera la partida entre 2 maquinas

- El juego debe estar bien validado (validaciones de entrada y del negocio)
- El juego permitirá guardar en cualquier momento de la partida.
- A menos que el juego que haya finalizado se podrá recuperar una partida *para continuarlo y terminarlo.*
- El juego debe soportar la opción de guardar en un archivo de tipo texto (txt)

Se solicita lo siguiente:

- Realizar la captura de Requisitos tanto funcionales como no funcionales:
- Realizar los diagramas de casos de uso y la especificación de cada uno.
- Realizar las interfaces de usuario (Mockups)
- Realizar el diagrama del dominio.
- Realizar el análisis y diseño de la aplicación. (Diagramas de Robustez y Secuencia)
- Diagrama de clases.
- Implementar la arquitectura MVC para la aplicación en Consola
- Implementar la arquitectura MVP con vista Pasiva para la aplicación en Consola y de Escritorio.
- Implementar una arquitectura de persistencia que soporte varios orígenes de datos.

2. Requerimientos

2.1 Requisitos Funcionales

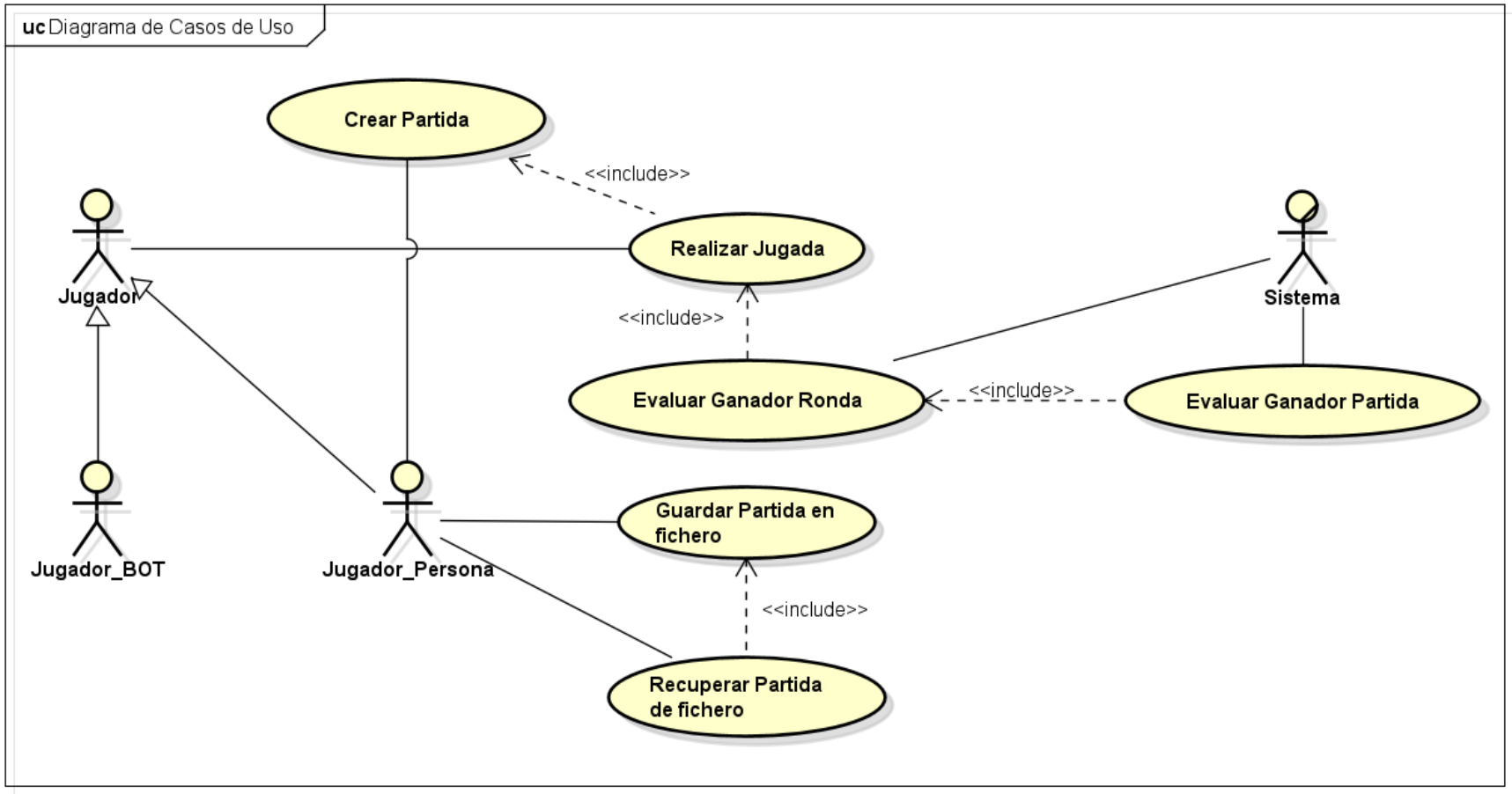
RF_N°	Descripción de los Requisitos Funcionales -Administradores
RF_01	Permitir realizar partidas entre jugadores, jugador contra la máquina o maquina contra máquina.
RF_02	Calcular con exactitud los puntos de las rondas que han sido jugadas, y determinar un ganador.
RF_03	Guardar una partida en cualquier momento de su desarrollo.
RF_04	Recuperar una partida guardada, y continuarla desde donde esta se quedó, excepto cuando ya finalizó.
RF_05	El resultado de las rondas será determinado por las reglas básicas del yankenpo.

2.2 Requisitos No Funcionales

RNF_N°	Descripción de los Requisitos No Funcionales
RNF_01	El sistema debe tener un tiempo de respuesta no mayor a 5 segundos.
RNF_02	Debe ser capaz de validar los datos de entrada para evitar excepciones.
RNF_03	Debe mostrar mensajes de error en caso de que se produzcan excepciones.
RNF_04	Debe ser capaz de funcionar tanto en modo gráfico como en modo consola.

3. Diagramas

3.1 Diagrama de Casos de Uso



3.2 Especificación de Casos de Uso

- Crear Partida

Especificación de Caso de Uso		
Caso de Uso	Crear Partida	
Precondiciones	Iniciar Programa.	
Flujo de Eventos	**Básico**	El usuario del sistema al ver la ventana de tipos de partida hace click en la que desea elegir, y junto a esto ingresar los puntos de victoria de la partida para que el sistema los valide (Estos deben ser positivos). A continuación, el usuario registra los jugadores depende el tipo de partida que se Eligio. Se muestra la Ventana Pantalla en la que se realizara el siguiente caso de Uso.
	Alternativos	• Puntos Negativos: El sistema muestra un mensaje que informa que los puntos de victoria no son válidos. • El usuario cierra la Ventana Menú: El sistema Termina • Error al registrar Jugadores: El sistema captura el tipo de error y muestra un mensaje que informa el tipo de error capturado.
Post condiciones	El sistema guarda la partida creada en una entidad	

- Realizar Jugada

Especificación de Caso de Uso		
Caso de Uso	Realizar jugada.	
Precondiciones	Haber creado una partida.	
Flujo de Eventos	**Básico**	El usuario del sistema ingresa a la Ventana Jugada, y el sistema le muestra una lista de las jugadas disponibles que pueden realizarse. El usuario da clic en la jugada de su elección, y el sistema se encarga de validarla. Si la jugada elegida es correcta, el sistema guarda la jugada realizada, y la agrega a su respectiva ronda.
	Alternativos	• Jugada incorrecta: El sistema muestra un mensaje que informa que la jugada elegida no es válida. • El usuario cierra la Ventana Jugada: El sistema muestra un mensaje preguntando al usuario si desea guardar la partida. • Error al registrar Jugada: El sistema captura el tipo de error y muestra un mensaje que informa el tipo de error capturado.
Post condiciones	El sistema guarda la jugada realizada por el usuario.	

- Evaluar Ganador Ronda

Especificación de Caso de Uso			
Caso de Uso	Evaluar Ganador Ronda		
Descripción	Es el proceso en el cual se evalúa al ganador de la ronda.		
Precondiciones	El jugador debe haberse registrado con anterioridad.		
Flujo de Eventos	1	Básico	El sistema verifica la vista ronda, evalúa al ganador de la ronda, muestra los resultados y los guarda en su ronda respectiva. Después de concluir el análisis de la ronda, agrega otra ronda y comienza una partida.
	2	Alternativos	Sí el usuario desea guardar el resultado de la ronda, necesita dar clic en "guardar resultados".
Postcondiciones	El sistema guarda la ronda jugada.		

- Evaluar Ganador Partida

Especificación de Caso de Uso			
Caso de Uso	Evaluar Ganador Partida		
Descripción	Es el proceso en el cual se evalúa al ganador de la partida.		
Precondiciones	El sistema evalúe el ganador de la ronda		
Flujo de Eventos	1	Básico	El sistema verifica la vista partida, evalúa al ganador de la partida, muestra los resultados y los guarda en caso que exista un ganador. Después de concluir la evaluación de la partida almacena los datos.
	2	Alternativos	Sí se desea guardar los resultados finales, tendría que existir un ganador obligatoriamente, en caso contrario no se podría efectuar el guardado.
Postcondiciones	Se debe realizar las jugadas para luego evaluarlas.		

- Guardar Partida en Fichero

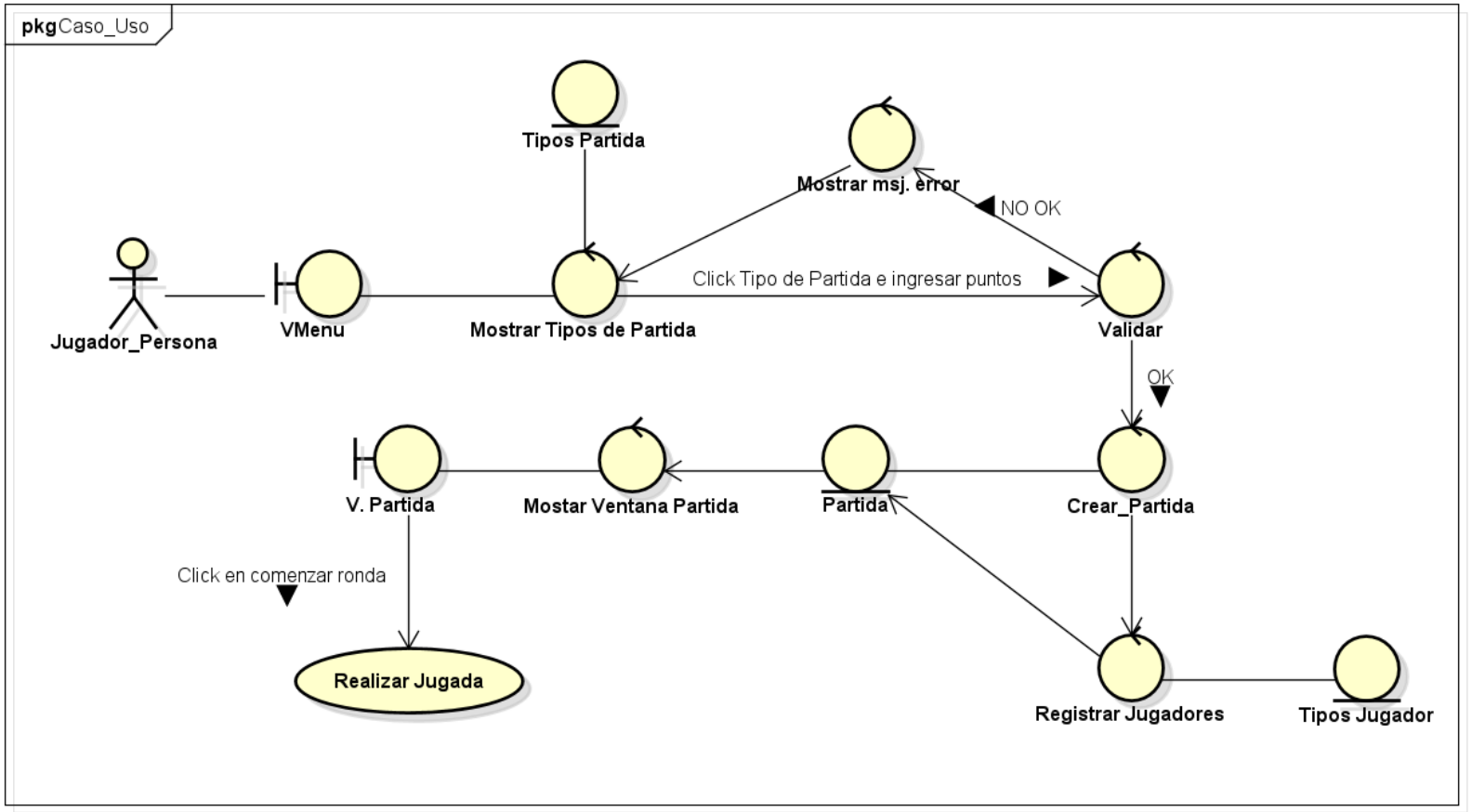
Especificación de Caso de Uso		
Caso de Uso	Guardar Partida en Fichero	
Precondiciones	Partida aún no Finalizada	
Flujo de Eventos	**Básico**	El usuario del sistema al estar en una partida, tiene la opción de guardarla en fichero. A continuación, el usuario presiona en guardar en fichero para salvar lo avanzado hasta ese momento. Se crea un nuevo fichero con nombre el id de la partida actual y se llenan con los datos de los jugadores y las jugadas en cada ronda existente Cada vez que se guarde una partida existe la opción de iniciar una nueva sin afectar los datos de la guardada.
	Alternativos	• Partida ya Finalizada: El sistema deshabilita la opción de seguir jugando. • Error al guardar datos: El sistema captura el tipo de error y muestra un mensaje que informa el tipo de error capturado. • Fichero no Existente: El sistema muestra un mensaje de error informando que no existe la dirección del fichero.
Post condiciones	El sistema guarda la partida creada en una entidad	

- Recuperar Partida de Fichero

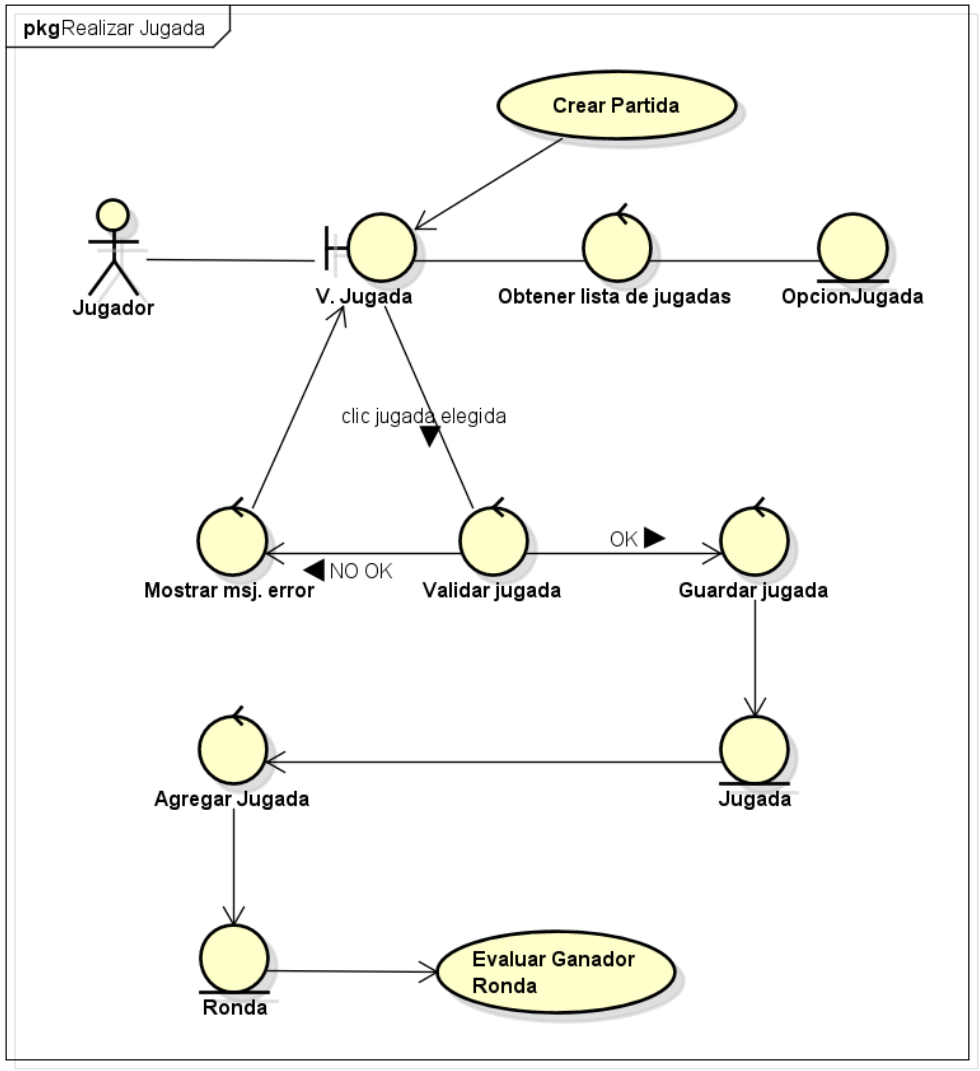
Especificación de Caso de Uso		
Caso de Uso	Recuperar Partida de Fichero	
Precondiciones	Partida guardada en fichero	
Flujo de Eventos	**Básico**	El usuario tiene la opción de iniciar una nueva partida o reanudar una partida ya comenzada que se ha guardado en el fichero. En este caso el usuario tiene que dar clic en recuperar partida para comenzar a jugar una partida ya guardada anteriormente.
	Alternativos	• Partida guardada: El sistema habilita la reanudación de la partida guardada. • Fichero no Existente: El sistema muestra un error de fichero de una partida que no ha sido guardado, haya finalizado o no existe.
Post condiciones	Reanudación de Partida	

3.3 Diagramas de Robustez

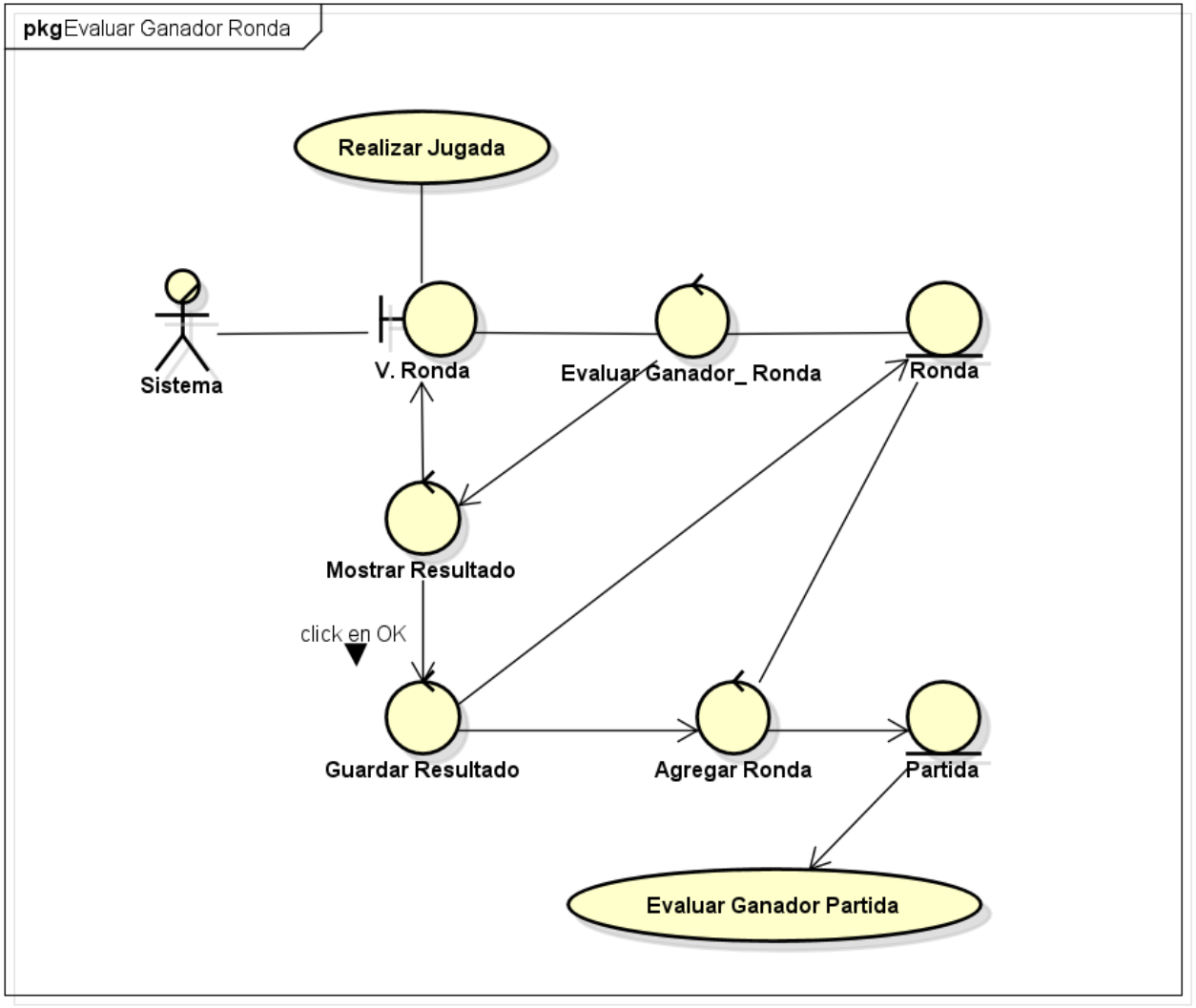
- ❖ Crear Partida



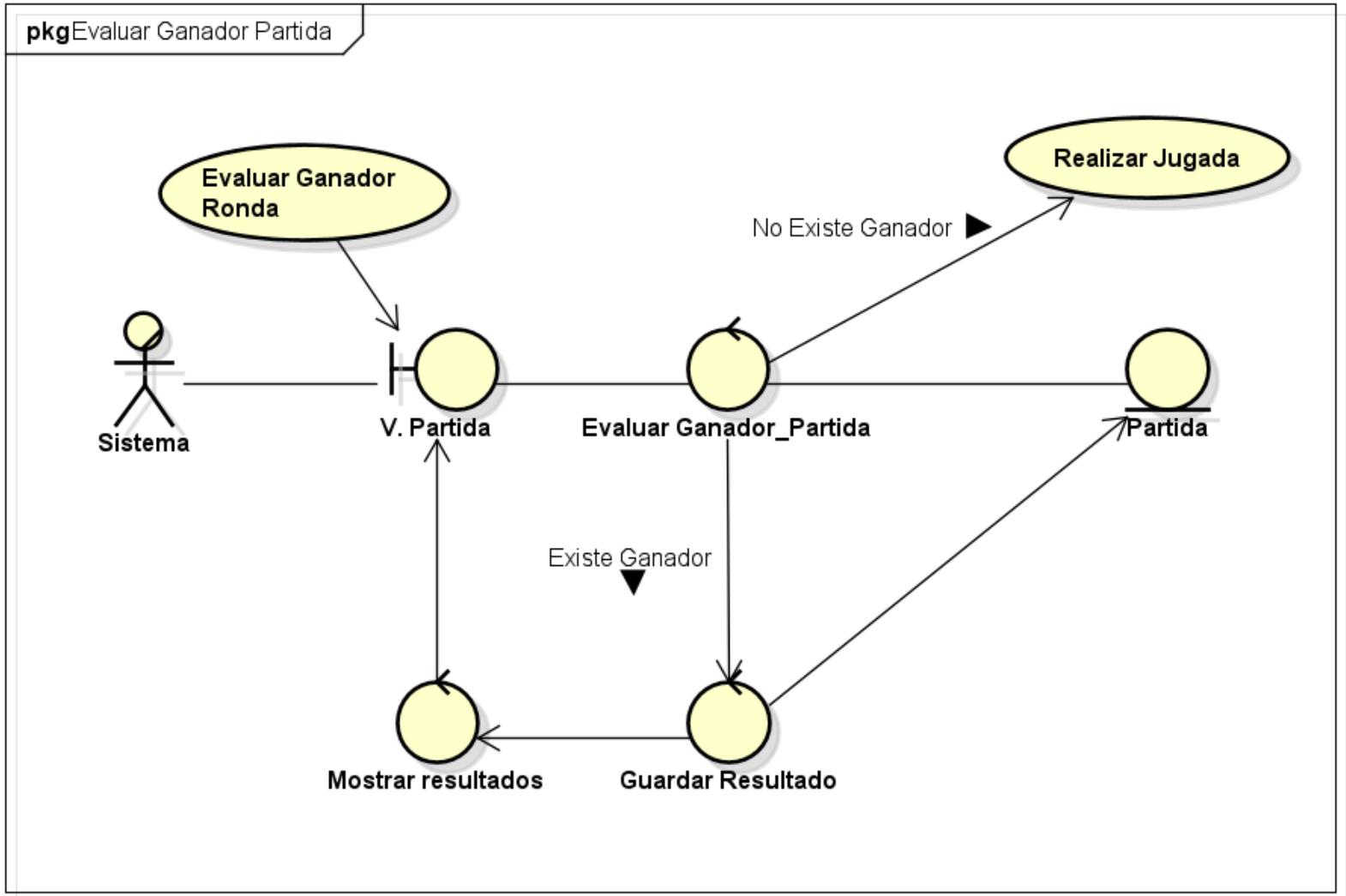
❖ Realizar Jugada



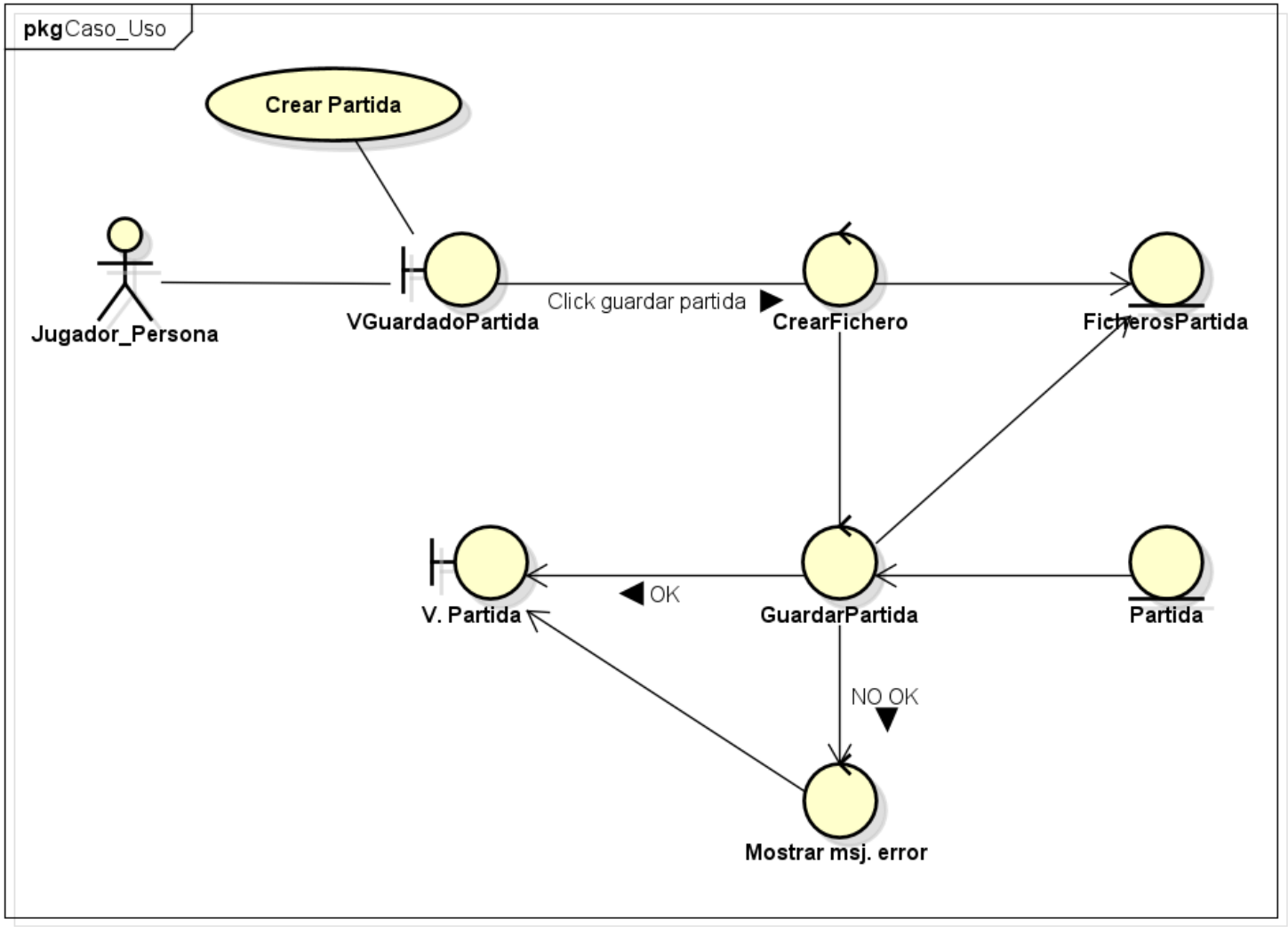
❖ Evaluar Ganador Ronda



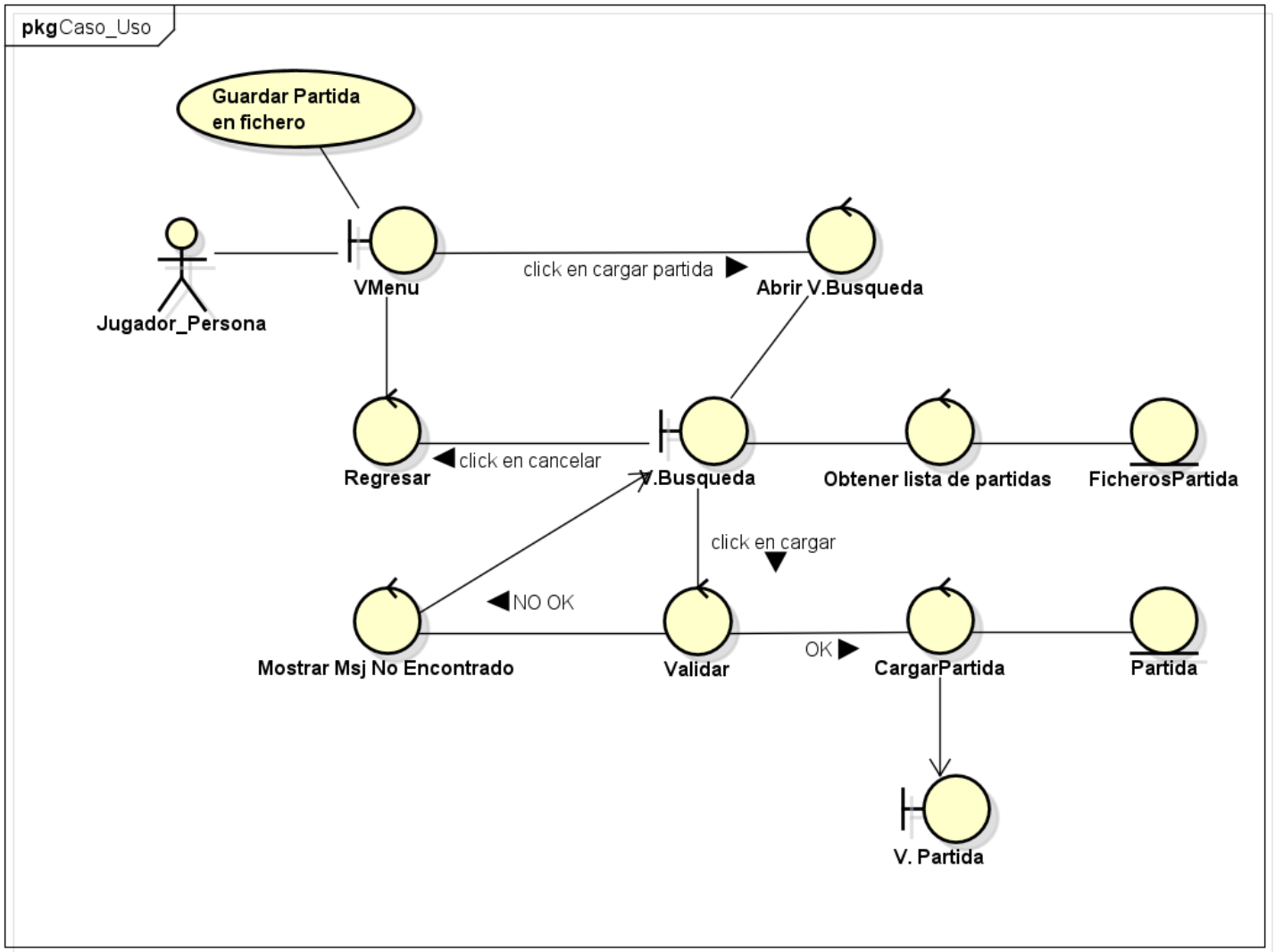
❖ Evaluar Ganador Partida



❖ Guardar Partida en Fichero

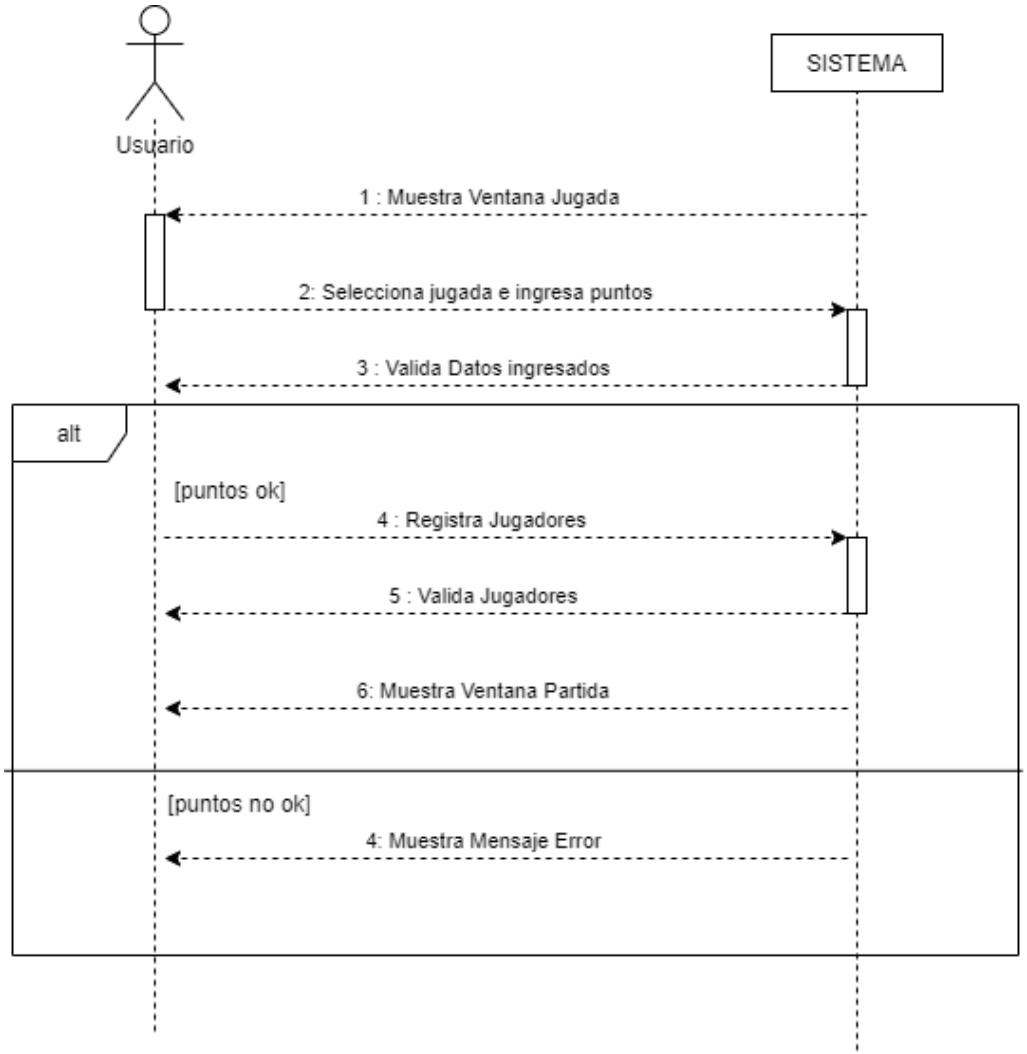


❖ Recuperar Partida de Fichero

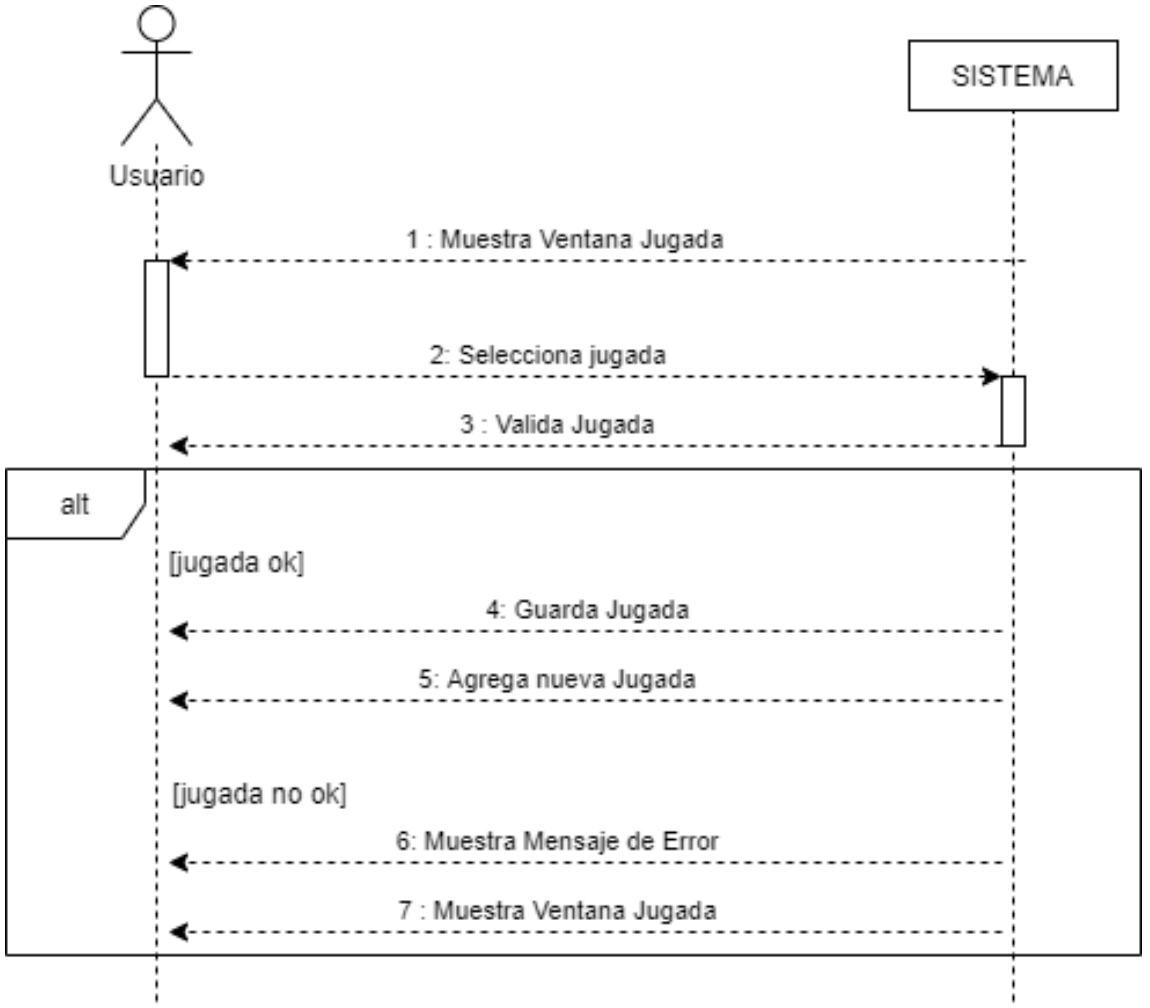


3.4 Diagramas de Secuencia

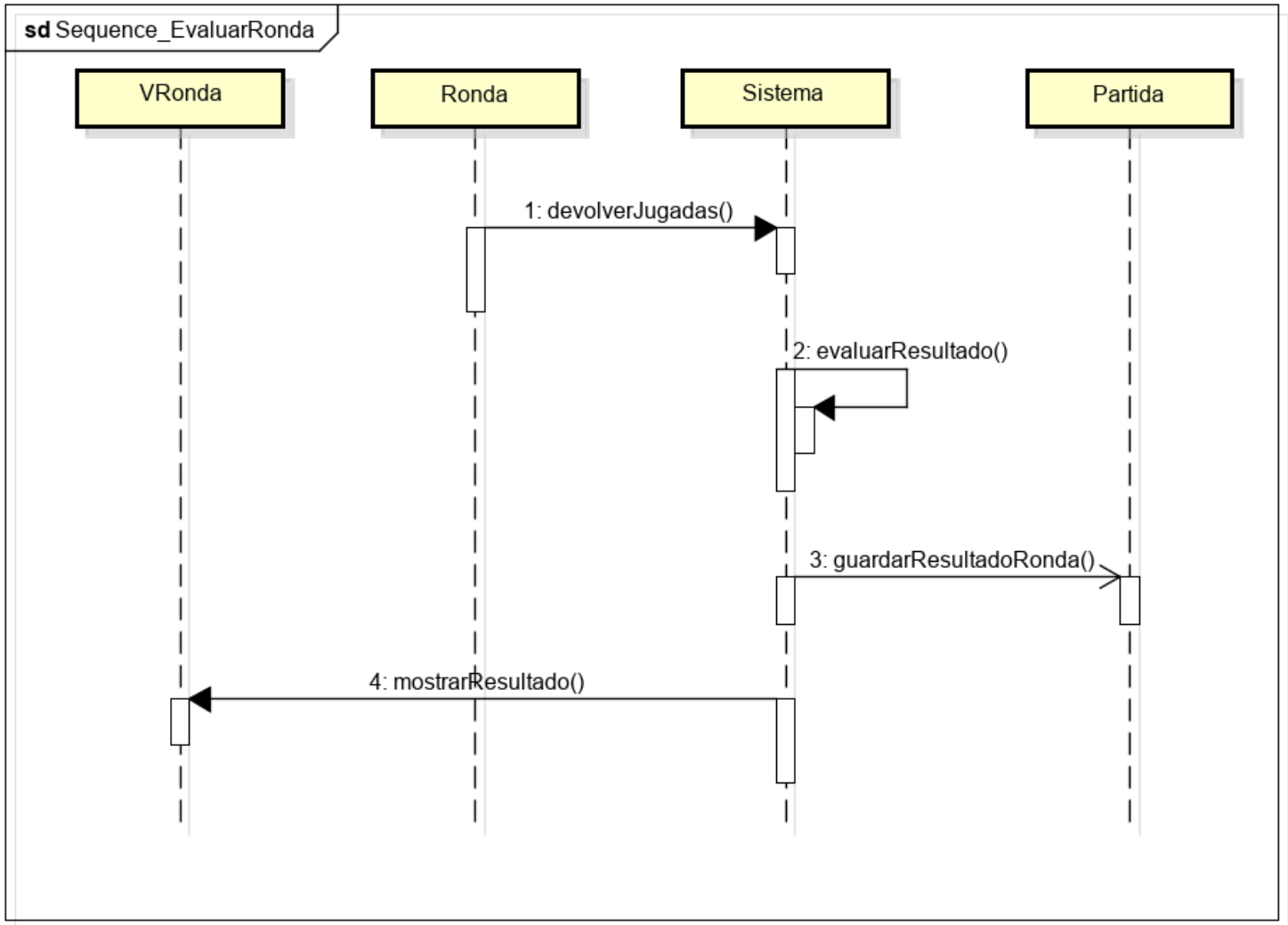
- ❖ Crear Partida



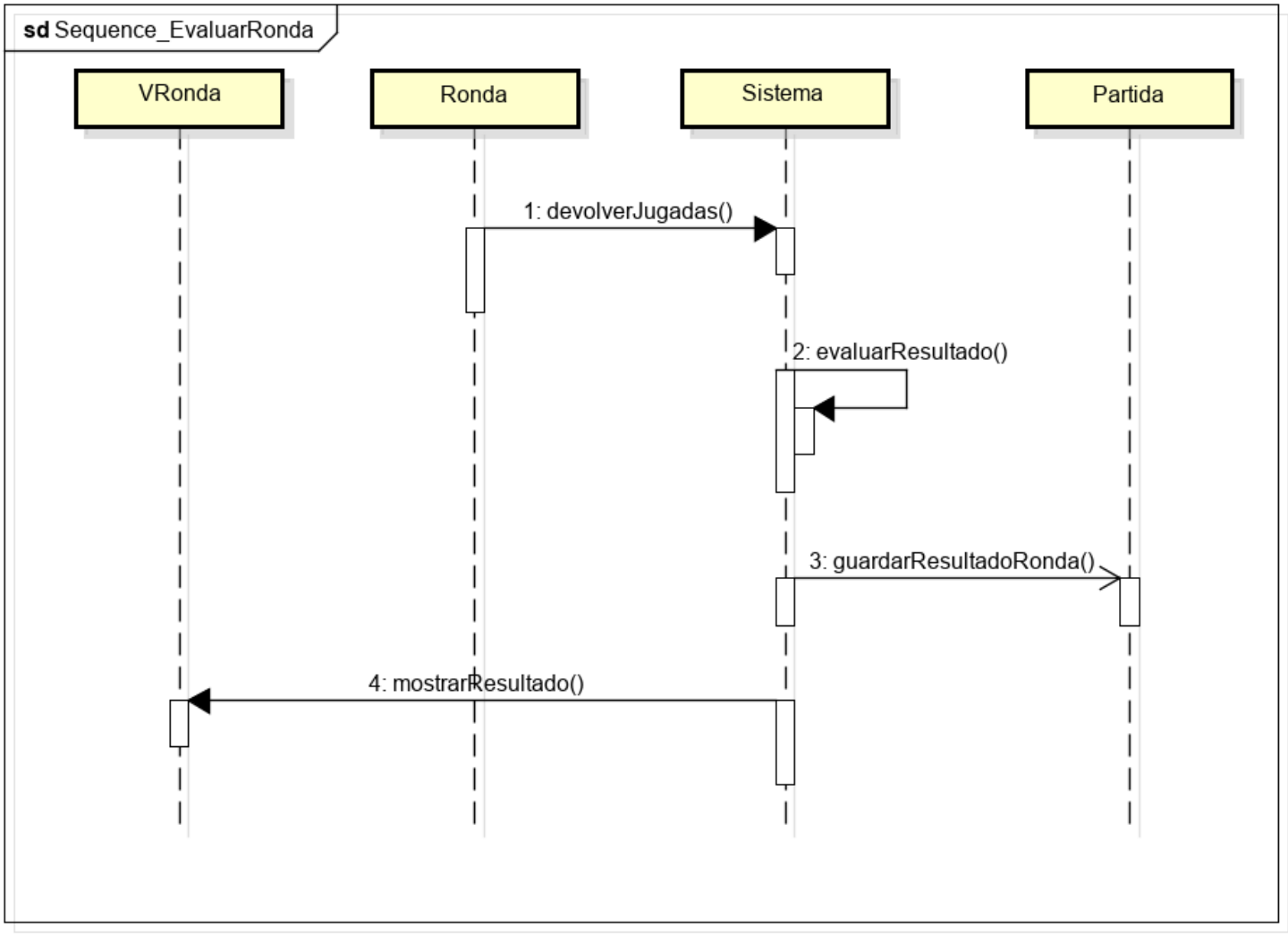
❖ Realizar Jugada



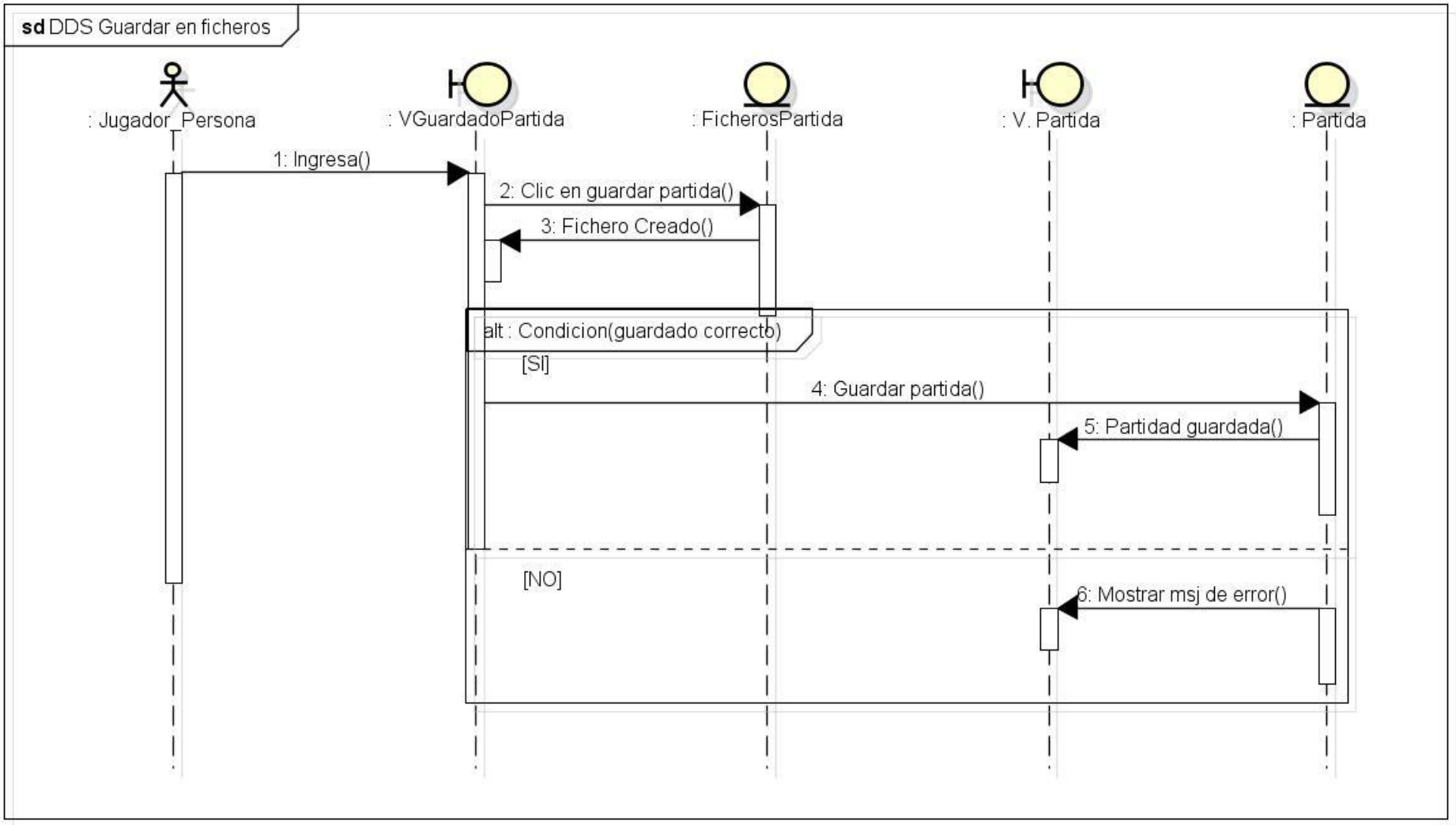
❖ Evaluar Ganador Ronda



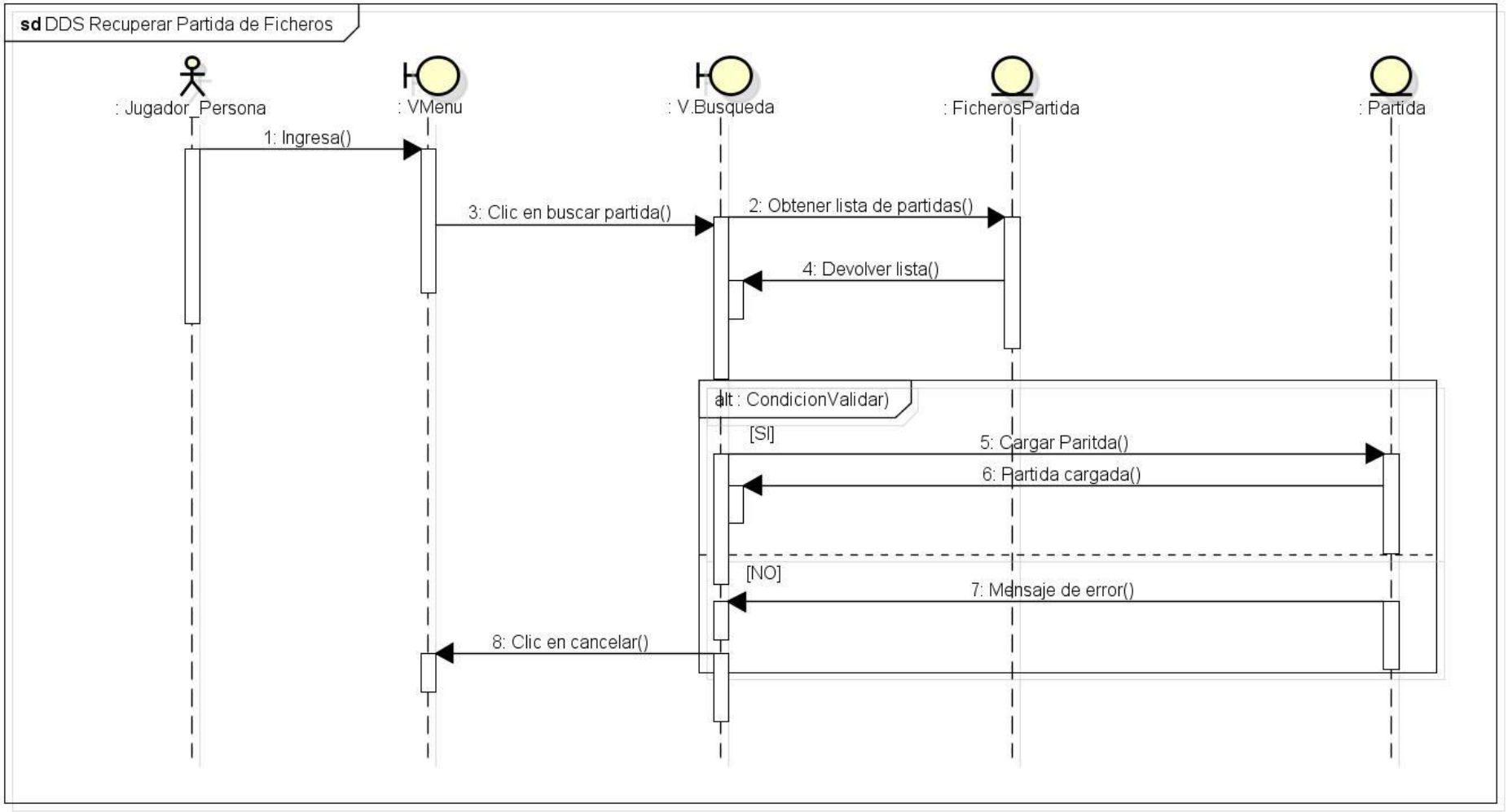
❖ Evaluar Ganador Partida



❖ Guardar Partida en Fichero



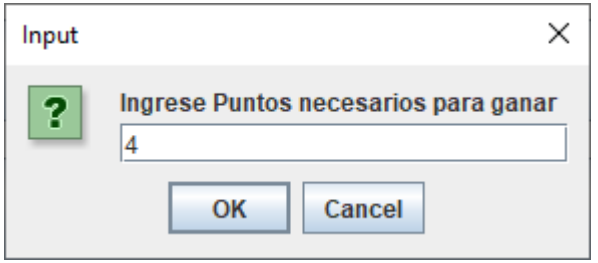
❖ Recuperar Partida de Fichero

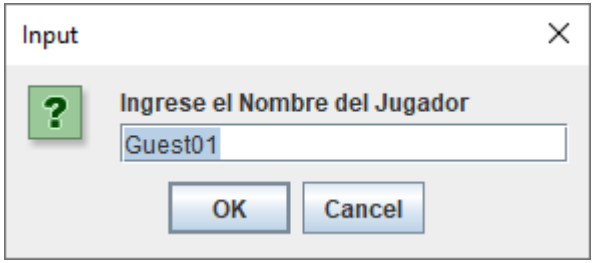


4. Wireframe

[Elaborado en lucidChart]

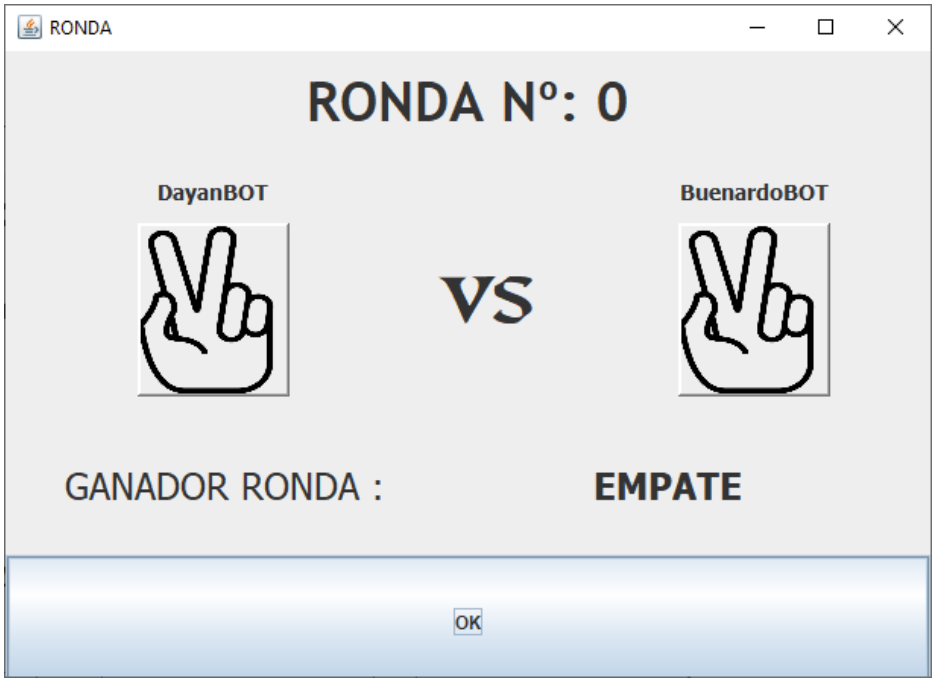
5. MockUps

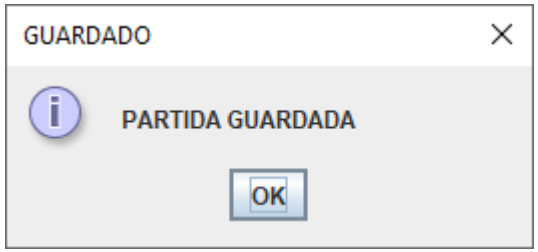





Partida de R P S L V
PARTIDA #7 (BvB mode)
DayanBOT
BuenardoBOT
0
-
0
GANADOR PARTIDA :
POR DECIDIR
GUARDAR PARTIDA
SALIR
COMENZAR RONDA

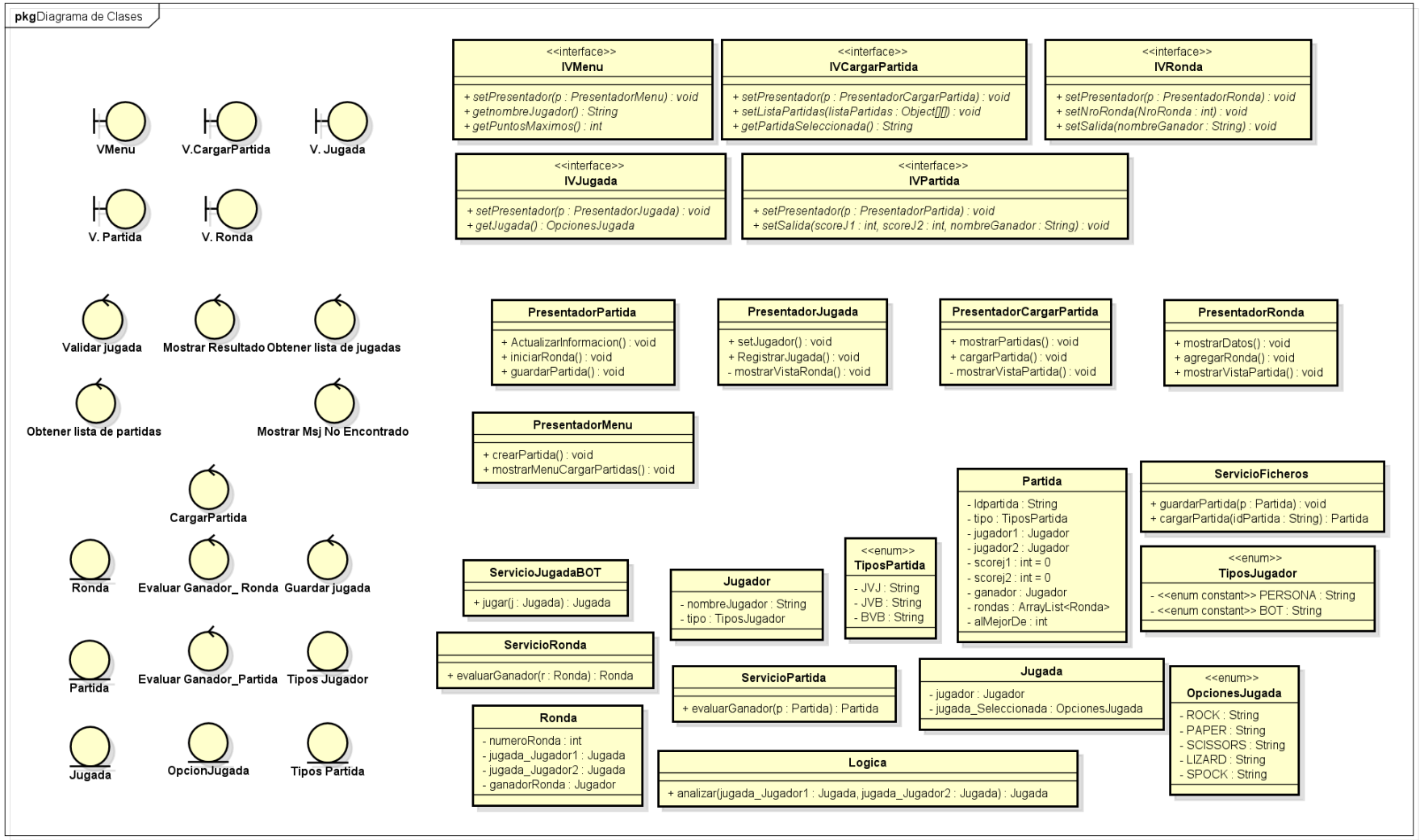

Jugada
HAZ TU JUGADA:
Guest01
ROCK
PAPER
SCISSORS
LIZARD
SPOCK
GOOOO!


RONDA
RONDA Nº: 0
DayanBOT
BuenardoBOT
VS
GANADOR RONDA :
EMPATE
OK


GUARDADO
PARTIDA GUARDADA
OK

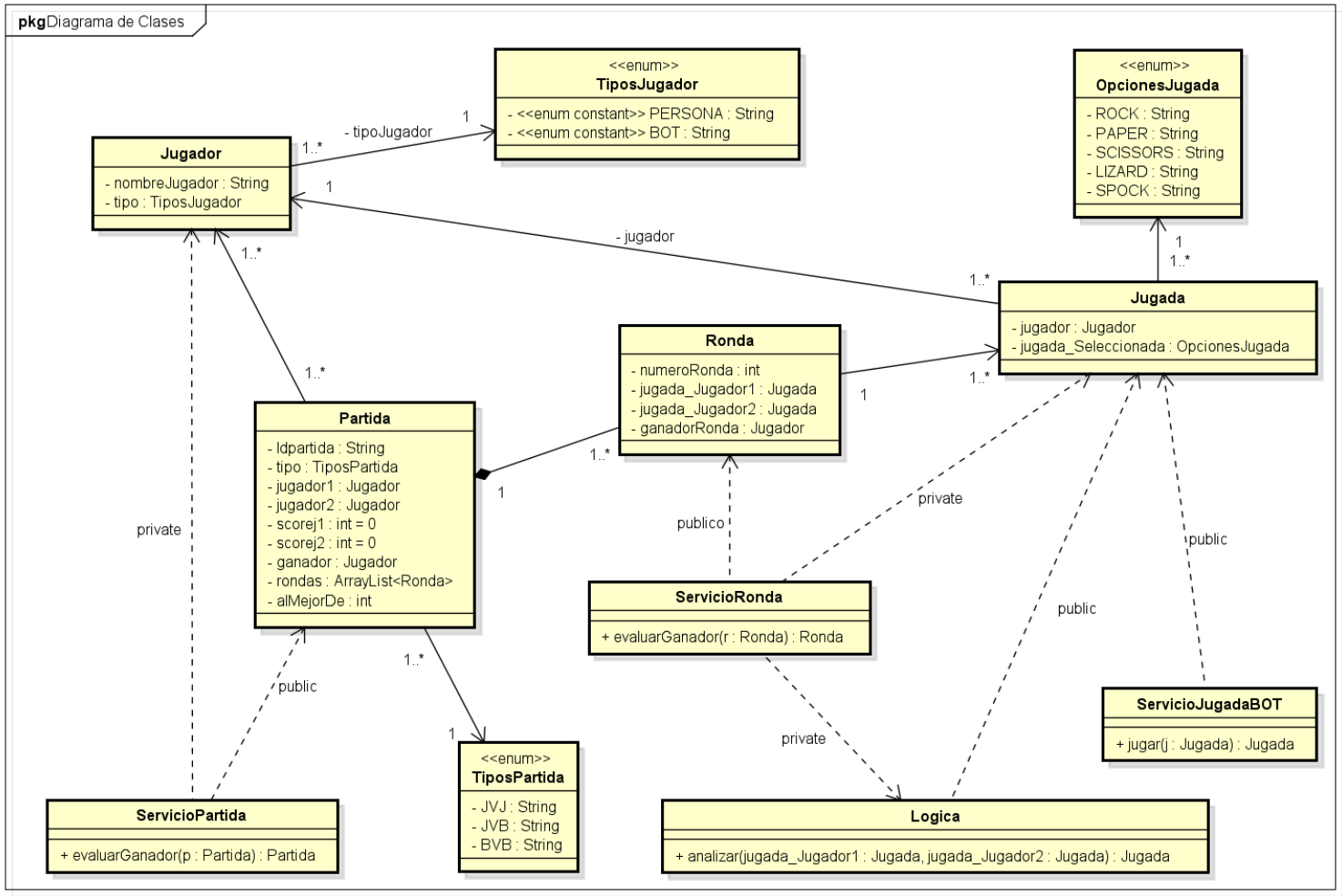

Menu Rock Paper Scissor Lizard Spock
LISTA DE PARTIDAS
idPartida PtosMaximos tipoPartida Jugador 1 Tipo J1 Jugador 2 Tipo J2
1 4 BvB Buenard... BOT DayanBOT BOT
2 4 BvB CHAVIX PERSONA DayanBOT BOT
3 3 BvB DayanBOT BOT Buenard... BOT
4 3 JvB Guest01 PERSONA PablitoBOT BOT
5 3 BvB DayanBOT BOT Buenard... BOT
6 3 BvB DayanBOT BOT Buenard... BOT
CARGAR
CANCELAR

6. Estrategia utilizada para obtener las clases

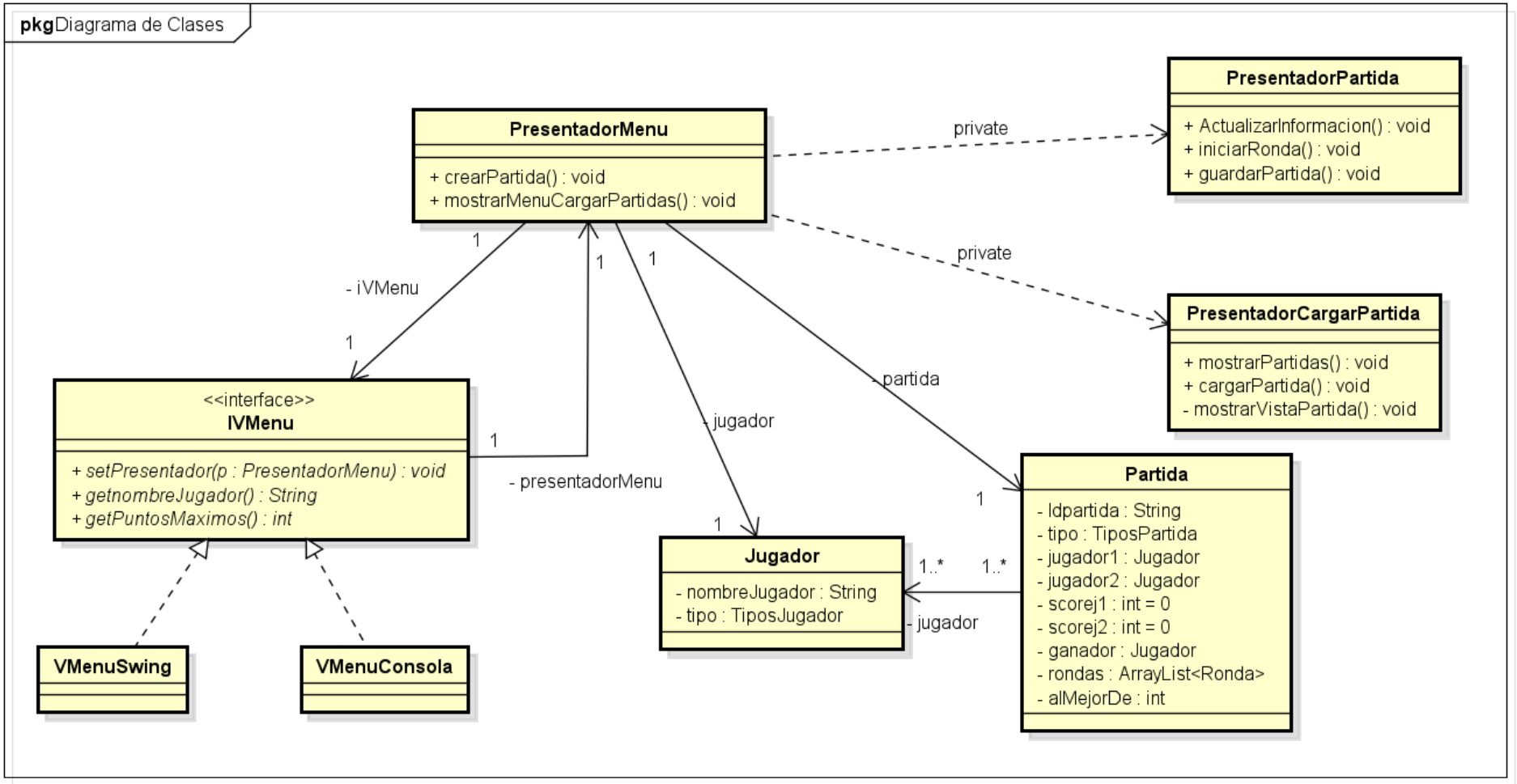


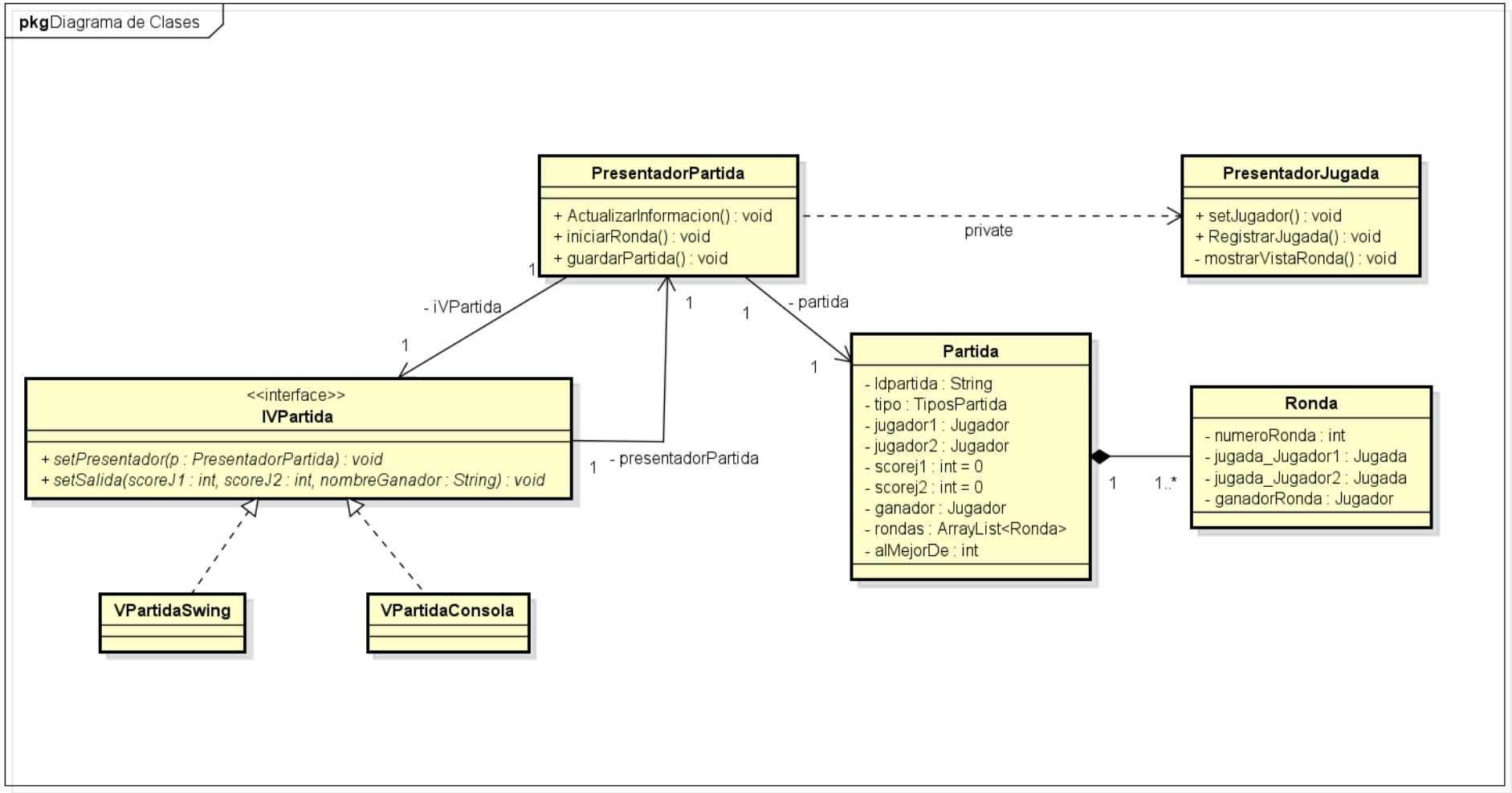
7. Diagrama de Clases

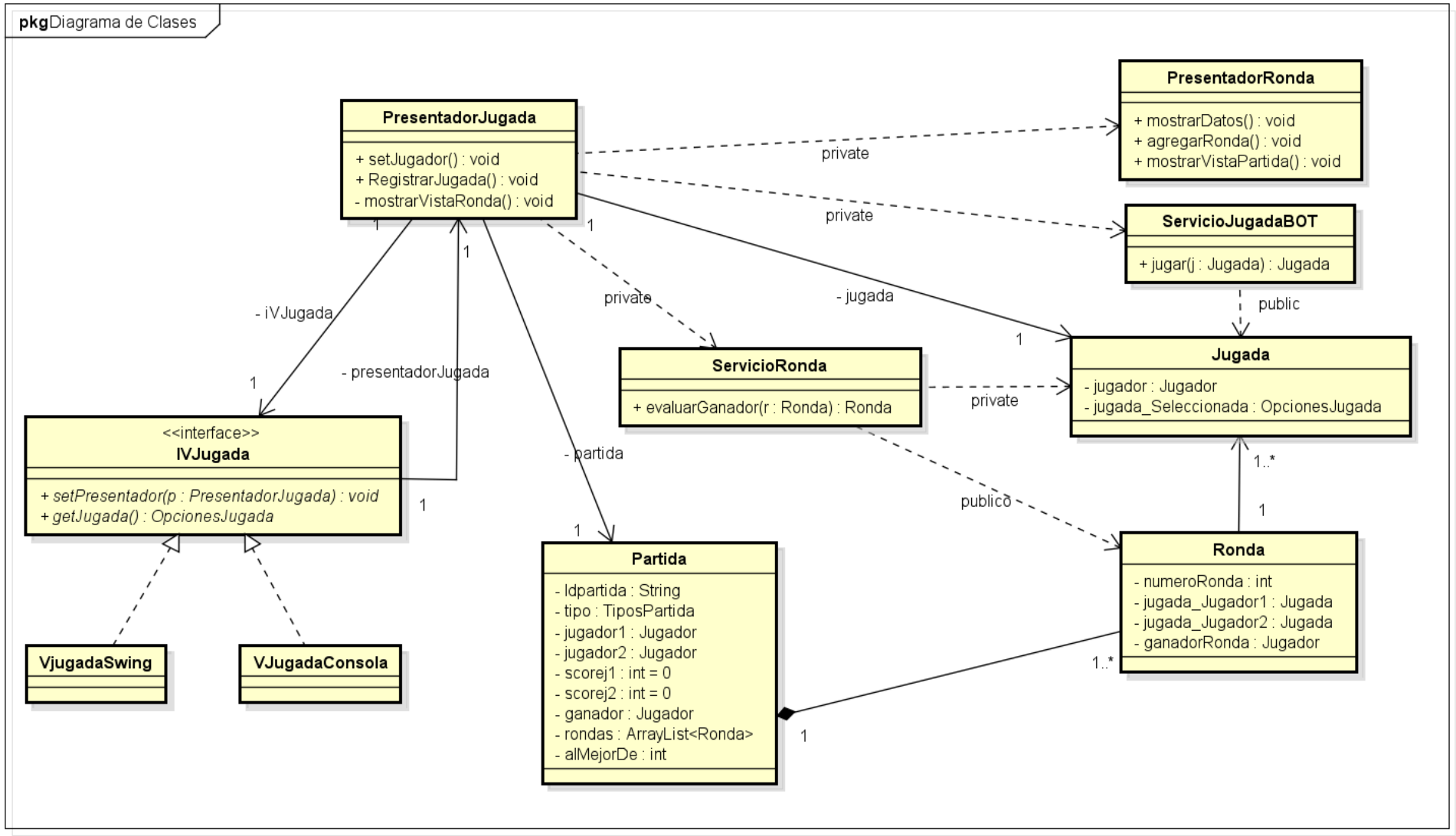
❖ Modelo

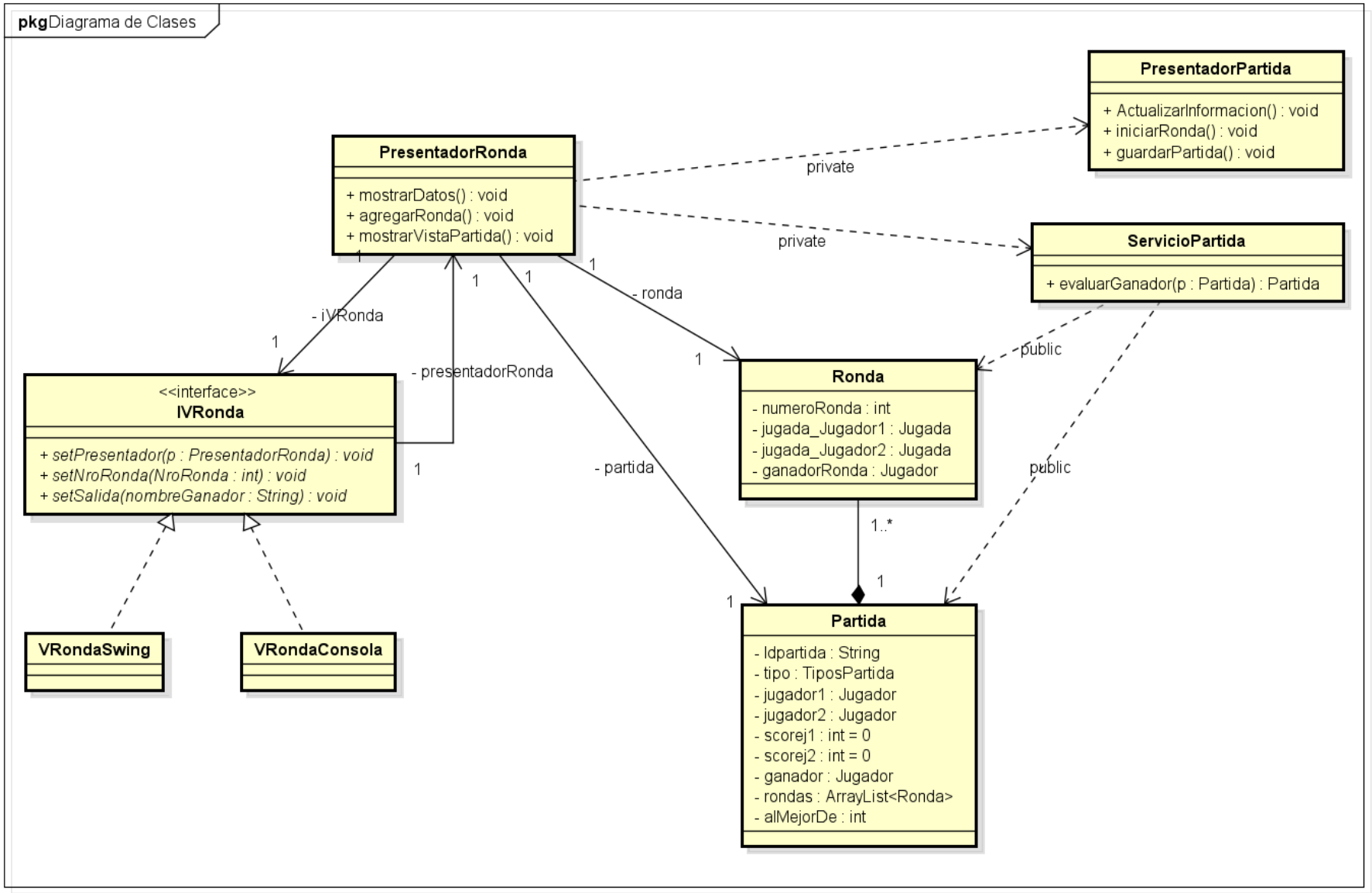


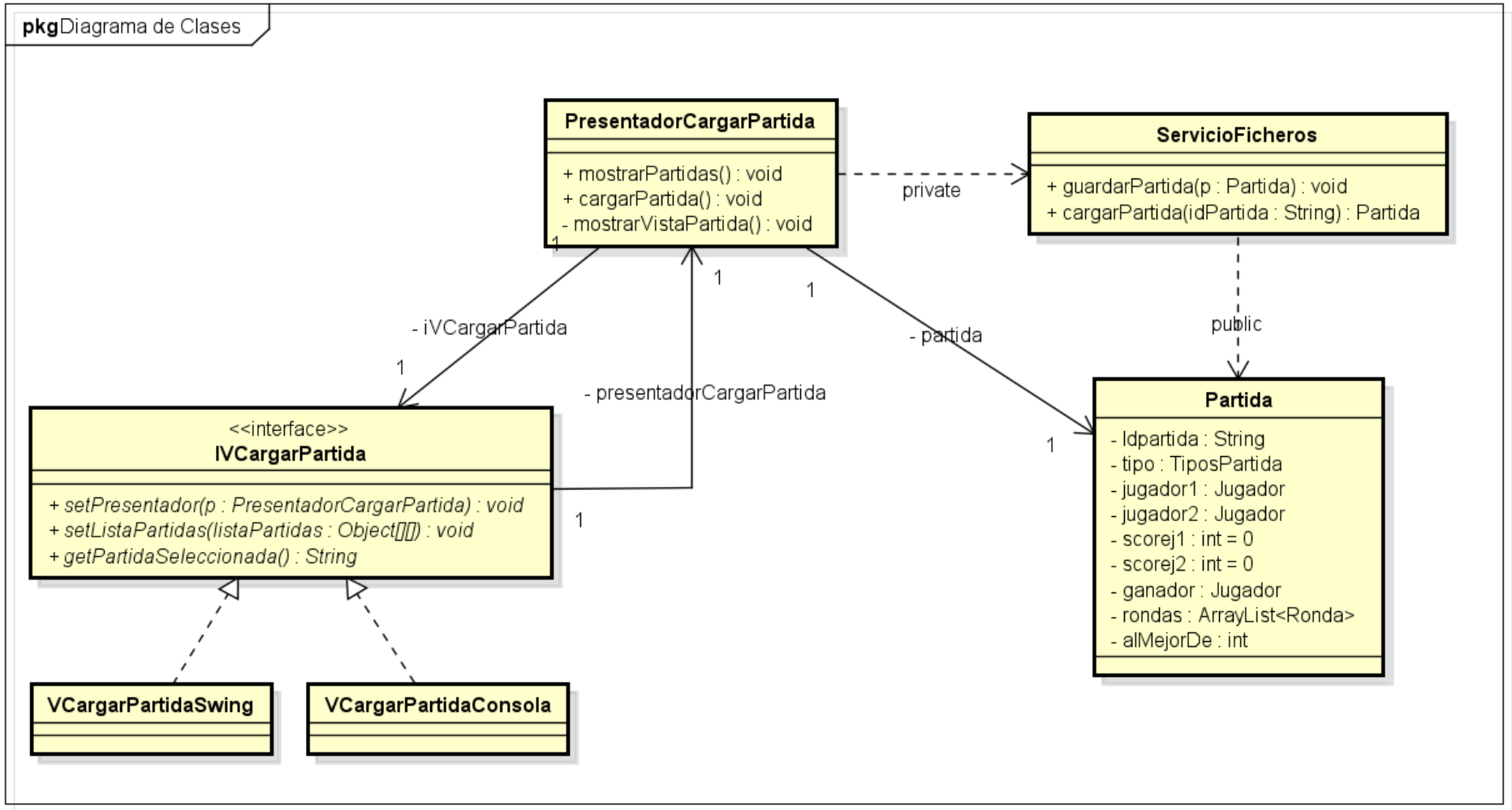
❖ **MVP Vista Pasiva**



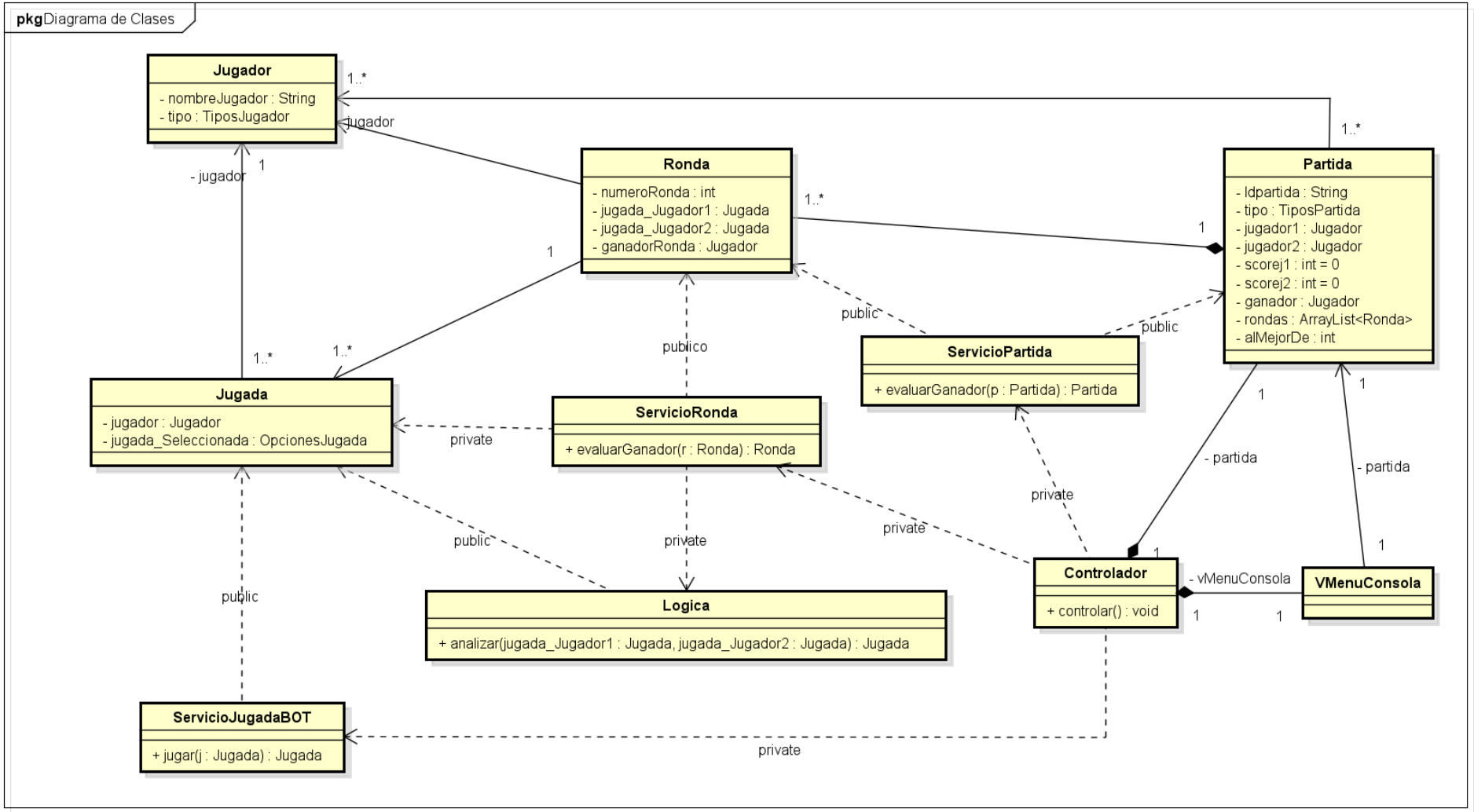

pkgDiagrama de Clases
PresentadorPartida
+ ActualizarInformacion() : void
+ iniciarRonda() : void
+ guardarPartida() : void
PresentadorJugada
+ setJugador() : void
+ RegistrarJugada() : void
- mostrarVistaRonda() : void
private
1
- iVPartida
1
1
1
- partida
1
<<interface>>
IVPartida
+ setPresentador(p : PresentadorPartida) : void
+ setSalida(scoreJ1 : int, scoreJ2 : int, nombreGanador : String) : void
1
- presentadorPartida
Partida
- Idpartida : String
- tipo : TiposPartida
- jugador1 : Jugador
- jugador2 : Jugador
- scorej1 : int = 0
- scorej2 : int = 0
- ganador : Jugador
- rondas : ArrayList<Ronda>
- alMejorDe : int
1
1..*
Ronda
- numeroRonda : int
- jugada_Jugador1 : Jugada
- jugada_Jugador2 : Jugada
- ganadorRonda : Jugador
VPartidaSwing
VPartidaConsola


pkgDiagrama de Clases
PresentadorJugada
+ setJugador() : void
+ RegistrarJugada() : void
- mostrarVistaRonda() : void
PresentadorRonda
+ mostrarDatos() : void
+ agregarRonda() : void
+ mostrarVistaPartida() : void
ServicioJugadaBOT
+ jugar(j : Jugada) : Jugada
ServicioRonda
+ evaluarGanador(r : Ronda) : Ronda
Jugada
- jugador : Jugador
- jugada_Seleccionada : OpcionesJugada
<<interface>>
IVJugada
+ setPresentador(p : PresentadorJugada) : void
+ getJugada() : OpcionesJugada
VjugadaSwing
VJugadaConsola
Partida
- Idpartida : String
- tipo : TiposPartida
- jugador1 : Jugador
- jugador2 : Jugador
- scorej1 : int = 0
- scorej2 : int = 0
- ganador : Jugador
- rondas : ArrayList<Ronda>
- alMejorDe : int
Ronda
- numeroRonda : int
- jugada_Jugador1 : Jugada
- jugada_Jugador2 : Jugada
- ganadorRonda : Jugador
private
private
private
- jugada
public
- iVJugada
- presentadorJugada
- partida
private
publico
1
1..*

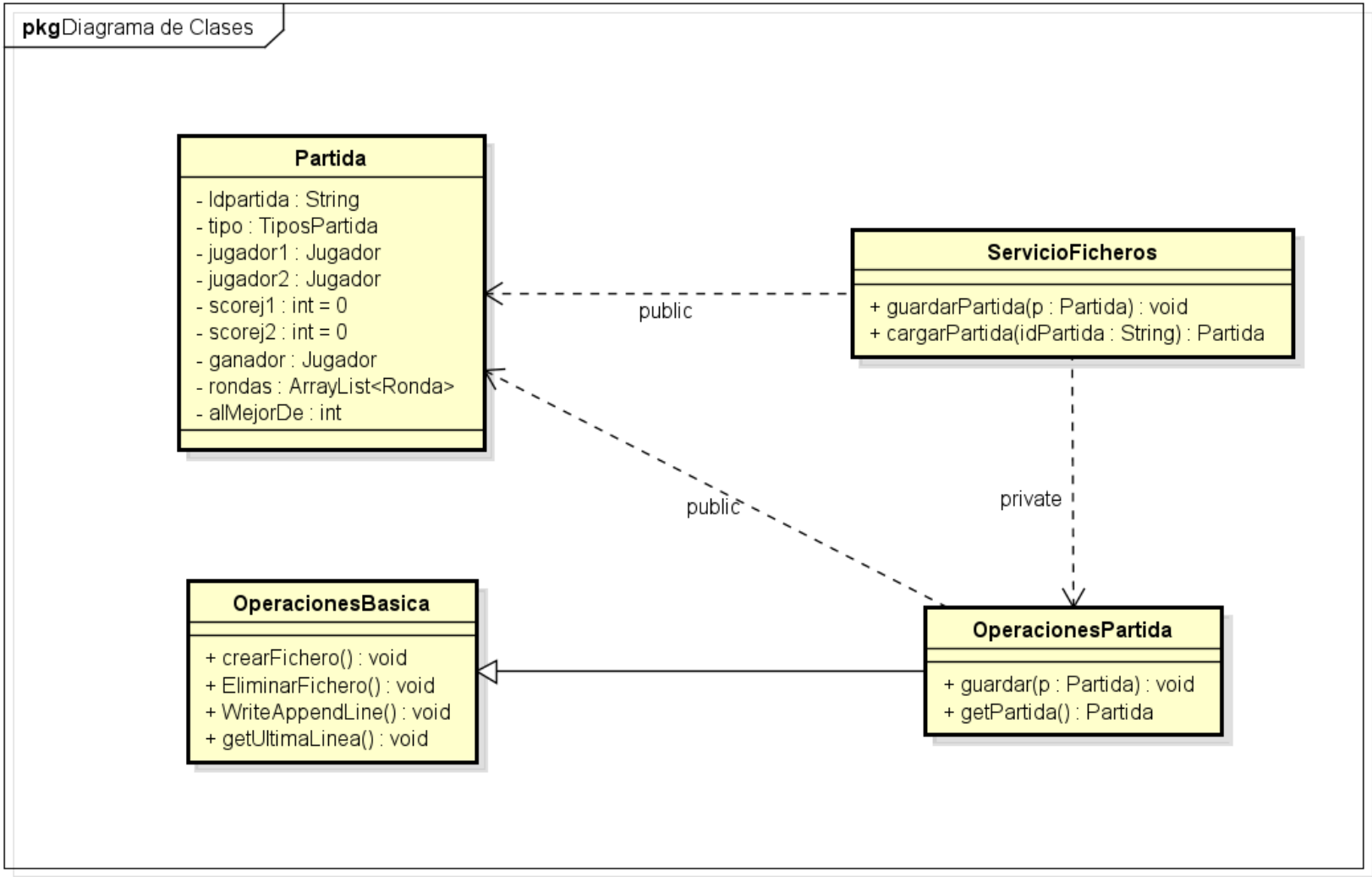

pkgDiagrama de Clases
PresentadorPartida
+ ActualizarInformacion() : void
+ iniciarRonda() : void
+ guardarPartida() : void
PresentadorRonda
+ mostrarDatos() : void
+ agregarRonda() : void
+ mostrarVistaPartida() : void
private
private
ServicioPartida
+ evaluarGanador(p : Partida) : Partida
- iVRonda
- presentadorRonda
- ronda
- partida
public
public
<<interface>>
IVRonda
+ setPresentador(p : PresentadorRonda) : void
+ setNroRonda(NroRonda : int) : void
+ setSalida(nombreGanador : String) : void
Ronda
- numeroRonda : int
- jugada_Jugador1 : Jugada
- jugada_Jugador2 : Jugada
- ganadorRonda : Jugador
1..*
VRondaSwing
VRondaConsola
Partida
- Idpartida : String
- tipo : TiposPartida
- jugador1 : Jugador
- jugador2 : Jugador
- scorej1 : int = 0
- scorej2 : int = 0
- ganador : Jugador
- rondas : ArrayList<Ronda>
- alMejorDe : int


pkgDiagrama de Clases
PresentadorCargarPartida
+ mostrarPartidas() : void
+ cargarPartida() : void
- mostrarVistaPartida() : void
ServicioFicheros
+ guardarPartida(p : Partida) : void
+ cargarPartida(idPartida : String) : Partida
private
public
- iVCargarPartida
- presentadorCargarPartida
- partida
1
<<interface>>
IVCargarPartida
+ setPresentador(p : PresentadorCargarPartida) : void
+ setListaPartidas(listaPartidas : Object[][]) : void
+ getPartidaSeleccionada() : String
Partida
- Idpartida : String
- tipo : TiposPartida
- jugador1 : Jugador
- jugador2 : Jugador
- scorej1 : int = 0
- scorej2 : int = 0
- ganador : Jugador
- rondas : ArrayList<Ronda>
- alMejorDe : int
VCargarPartidaSwing
VCargarPartidaConsola

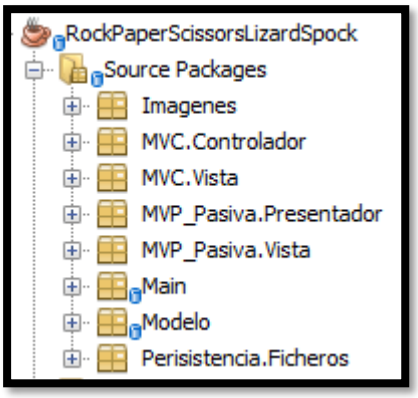
❖ **MVC**


pkgDiagrama de Clases
Jugador
- nombreJugador : String
- tipo : TiposJugador
Ronda
- numeroRonda : int
- jugada_Jugador1 : Jugada
- jugada_Jugador2 : Jugada
- ganadorRonda : Jugador
Partida
- Idpartida : String
- tipo : TiposPartida
- jugador1 : Jugador
- jugador2 : Jugador
- scorej1 : int = 0
- scorej2 : int = 0
- ganador : Jugador
- rondas : ArrayList<Ronda>
- alMejorDe : int
Jugada
- jugador : Jugador
- jugada_Seleccionada : OpcionesJugada
ServicioRonda
+ evaluarGanador(r : Ronda) : Ronda
ServicioPartida
+ evaluarGanador(p : Partida) : Partida
Logica
+ analizar(jugada_Jugador1 : Jugada, jugada_Jugador2 : Jugada) : Jugada
Controlador
+ controlar() : void
VMenuConsola
ServicioJugadaBOT
+ jugar(j : Jugada) : Jugada
jugador
- jugador
publico
public
private
- partida
- vMenuConsola
1..*
1

❖ **Persistencia**



8. Diagrama de Paquetes



9. Clases

Modelo

- **Clase Jugada**

```
public class Jugada {

    private Jugador jugador;
    private OpcionesJugada jugada_Seleccionada;

    public Jugada(Jugador jugador, OpcionesJugada
jugada_Seleccionada) {
        this.jugador = jugador;
        this.jugada_Seleccionada = jugada_Seleccionada;
    }

    public Jugada() {
    }

    // GETTERS AND SETTERS
```

- **Clase Jugador**

```
public class Jugador {

    private String nombreJugador;
    private TiposJugador tipo;

    public Jugador(String nombreJugador, TiposJugador tipo) {
        this.nombreJugador = nombreJugador;
        this.tipo = tipo;
    }

    public Jugador() {
    }

  // SETTERS AND GETTER

    @Override
    public String toString() {
        return "Jugador{" + "nombreJugador=" + nombreJugador + ",
tipo=" + tipo + '}';
    }

}
```

- **Clase Lógica**

```
class Logica {

    Jugada analizar(Jugada jugada_Jugador1, Jugada jugada_Jugador2)
{
        switch (jugada_Jugador1.getJugada_Seleccionada()) {
            case ROCK:
                switch (jugada_Jugador2.getJugada_Seleccionada()) {
                    case ROCK:
                        return null;
```

```
                case PAPER:
                    return jugada_Jugador2;
                case SCISSORS:
                    return jugada_Jugador1;
                case LIZARD:
                    return jugada_Jugador1;
                case SPOCK:
                    return jugada_Jugador2;
            }
            break;

        case PAPER:
            switch (jugada_Jugador2.getJugada_Seleccionada()) {
                case ROCK:
                    return jugada_Jugador1;
                case PAPER:
                    return null;
                case SCISSORS:
                    return jugada_Jugador2;
                case LIZARD:
                    return jugada_Jugador2;
                case SPOCK:
                    return jugada_Jugador1;
            }
            break;

        case SCISSORS:
            switch (jugada_Jugador2.getJugada_Seleccionada()) {
                case ROCK:
                    return jugada_Jugador2;
                case PAPER:
                    return jugada_Jugador1;
                case SCISSORS:
                    return null;
                case LIZARD:
                    return jugada_Jugador1;
                case SPOCK:
                    return jugada_Jugador2;
            }
            break;

        case LIZARD:
            switch (jugada_Jugador2.getJugada_Seleccionada()) {
                case ROCK:
                    return jugada_Jugador2;
                case PAPER:
                    return jugada_Jugador1;
                case SCISSORS:
                    return jugada_Jugador2;
                case LIZARD:
                    return null;
                case SPOCK:
                    return jugada_Jugador1;
            }
            break;

        case SPOCK:
```

```
                    switch (jugada_Jugador2.getJugada_Seleccionada()) {
                        case ROCK:
                            return jugada_Jugador1;
                        case PAPER:
                            return jugada_Jugador2;
                        case SCISSORS:
                            return jugada_Jugador1;
                        case LIZARD:
                            return jugada_Jugador2;
                        case SPOCK:
                            return null;
                    }
                    break;
            }
            System.out.println("ESTO NO TIENE QUE PASAR - REVISA
LOGICA.ANALIZAR()");
            return null;
        }
;
}
```

- **Clase Opciones Jugada**

```
public enum OpcionesJugada {
    ROCK('R'),
    PAPER('P'),
    SCISSORS('S'),
    LIZARD('L'),
    SPOCK('V');
    private char caracter;

    OpcionesJugada(char c) {
        this.caracter = c;
    }

    public char getCaracter() {
        return caracter;
    }

}
```

- **Clase Partida**

```
public class Partida {

    private String IdPartida;
    private TiposPartida tipo;
    private Jugador jugador1;
    private Jugador jugador2;
    private int scorej1 = 0;
    private int scorej2 = 0;
    private Jugador ganador;

    private ArrayList<Ronda> Rondas = new ArrayList();
    private int alMejorDe = 3;

    public void agregarRonda(Ronda r) {
        Rondas.add(r);
```

```
    }

    public boolean existeGanador() {
        return (ganador != null);
    }

    //GETTERS N SETTERS

    @Override
    public String toString() {
        return "Partida{" + "IdPartida=" + IdPartida + ", tipo=" +
tipo + ", jugador1=" + jugador1 + ", jugador2=" + jugador2 + ",
scorej1=" + scorej1 + ", scorej2=" + scorej2 + ", ganador=" +
ganador + ", Rondas=" + Rondas + ", alMejorDe=" + alMejorDe + '}';
    }
}
```

- **Class Ronda**

```
public class Ronda {
    private int numeroRonda;
    private Jugada jugada_Jugador1;
    private Jugada jugada_Jugador2;
    private Jugador ganadorRonda;

    public Ronda(int numeroRonda) {
        this.numeroRonda = numeroRonda;
    }

    public Ronda() {
    }
//GETTERS AND SETTERS
```

- **Class ServicioJugadaBot**

```
public class ServicioJugadaBOT {
    public Jugada jugar(Jugada j){
        int
indiceRandom=(int)( Math.random()*(OpcionesJugada.values().lengt
h-1) );

j.setJugada_Seleccionada(OpcionesJugada.values()[indiceRandom]);

        return j;
    }
}
```

- **Class ServicioPartida**

```
public class ServicioPartida {
    public Partida evaluarGanador(Partida p){
        int scorej1=0;
        int scorej2=0;
        for (Ronda ronda : p.getRondas()) {
            Jugador ganadorRonda = ronda.getGanadorRonda();
            if (ganadorRonda==p.getJugador1()) {
                scorej1++;
```

```
                }
                if (ganadorRonda==p.getJugador2()) {
                    scorej2++;
                }
            }
        if (scorej1>=p.getAlMejorDe()) {
            p.setGanador(p.getJugador1());
        }
        if (scorej2>=p.getAlMejorDe()) {
            p.setGanador(p.getJugador2());
        }

        p.setScorej1(scorej1);
        p.setScorej2(scorej2);

        return p;
             }
        }
```

- **Class ServicioRonda**

```
public class ServicioRonda {
    public Ronda evaluarGanador(Ronda r){
        Logica logica = new Logica();
        Jugada jugada_Ganadora =

logica.analizar(r.getJugada_Jugador1(),r.getJugada_Jugador2());

        if (jugada_Ganadora != null) {
            r.setGanadorRonda(jugada_Ganadora.getJugador());
        }

        return r;
    }
}
```

- **Class TiposJugador**

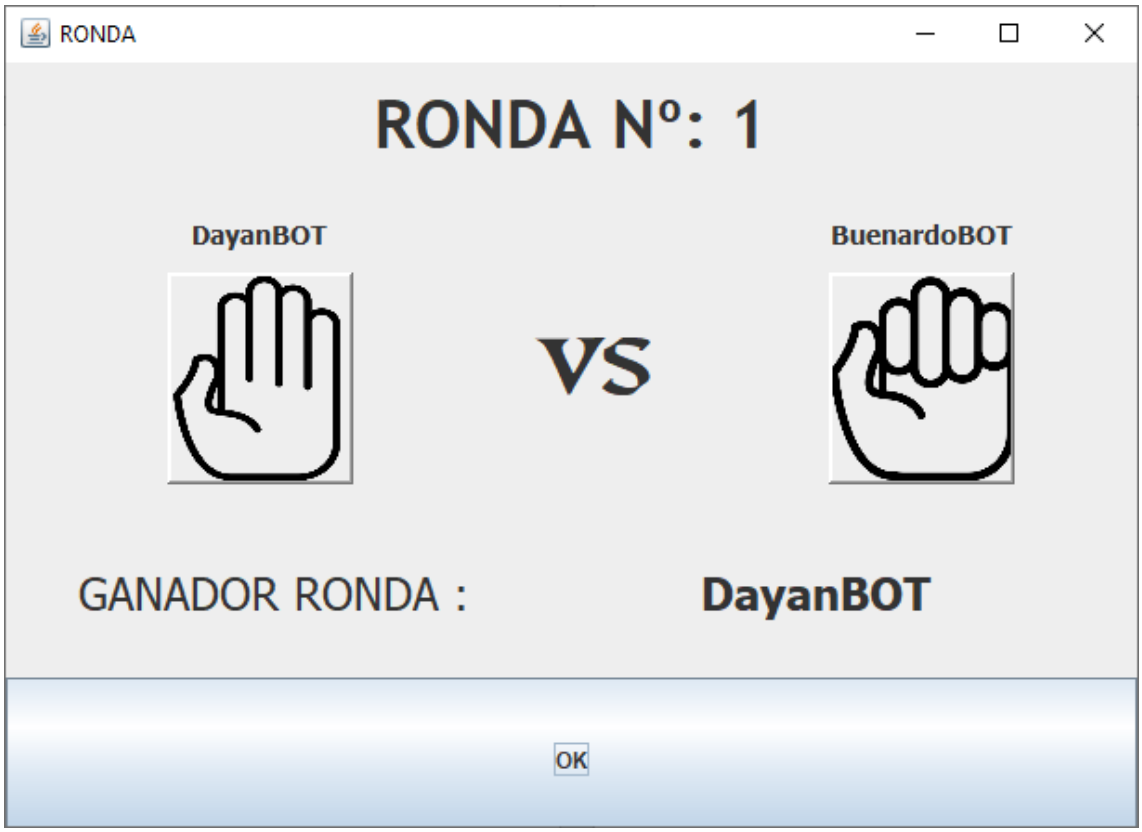
```
public enum TiposJugador {
    PERSONA, BOT;
}
```

- **Class TiposPartida**

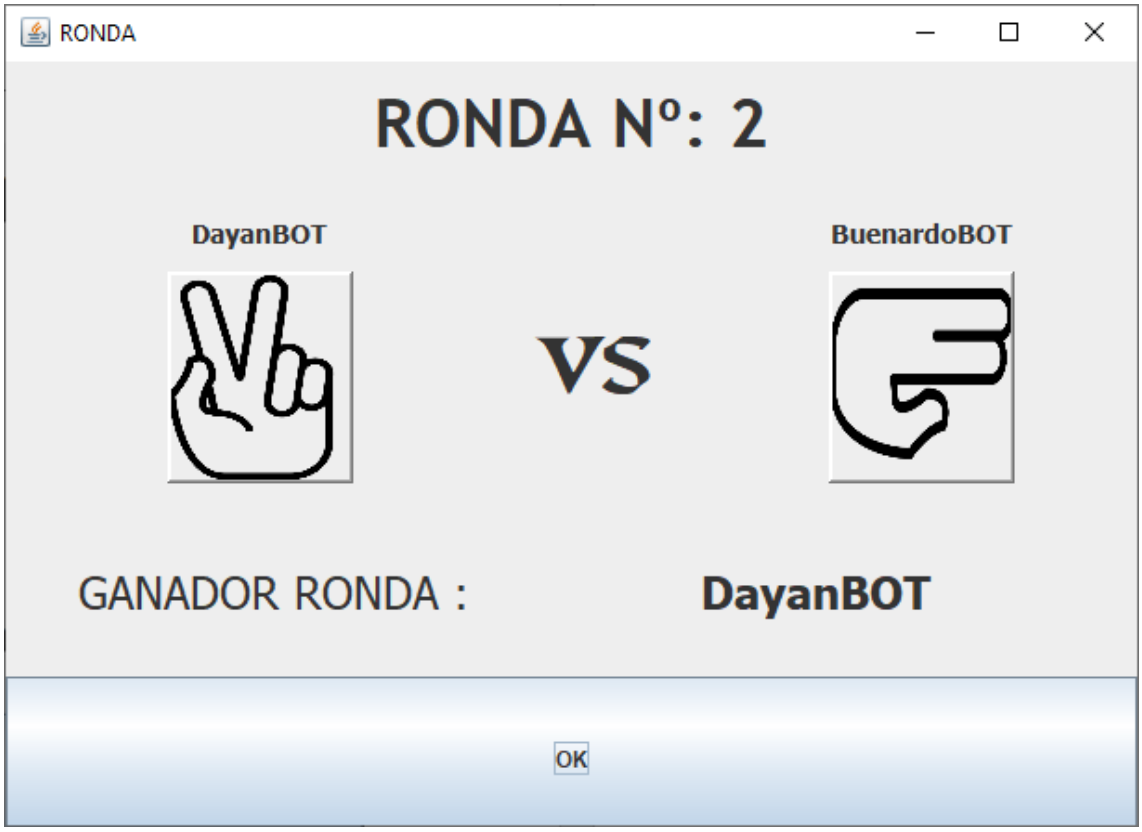
```
public enum TiposJugador {
    PERSONA, BOT;
}
```

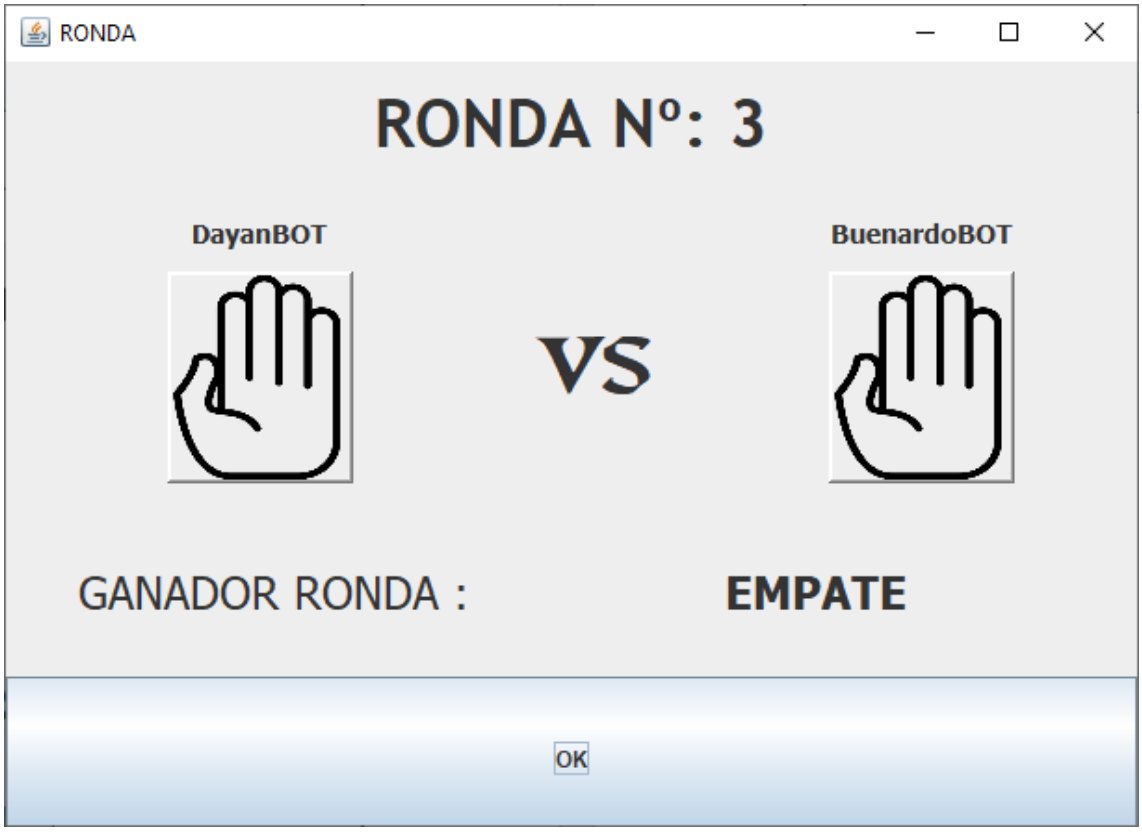
Lo demás se encuentra en el repositorio, debido al peso del informe.

10. RUN

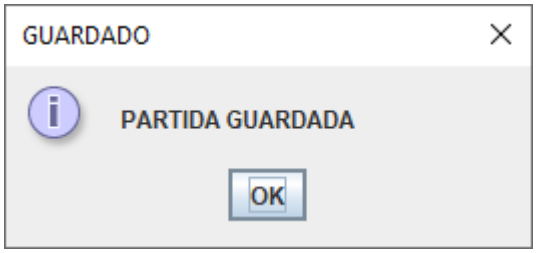

RONDA
RONDA Nº: 1
DayanBOT
BuenardoBOT
VS
GANADOR RONDA :
DayanBOT
OK


Partida de R P S L V
PARTIDA #8 (BvB mode)
DayanBOT
BuenardoBOT
1
-
0
GANADOR PARTIDA :
POR DECIDIR
GUARDAR PARTIDA
SALIR
COMENZAR RONDA


RONDA
RONDA Nº: 2
DayanBOT
BuenardoBOT
VS
GANADOR RONDA :
DayanBOT
OK

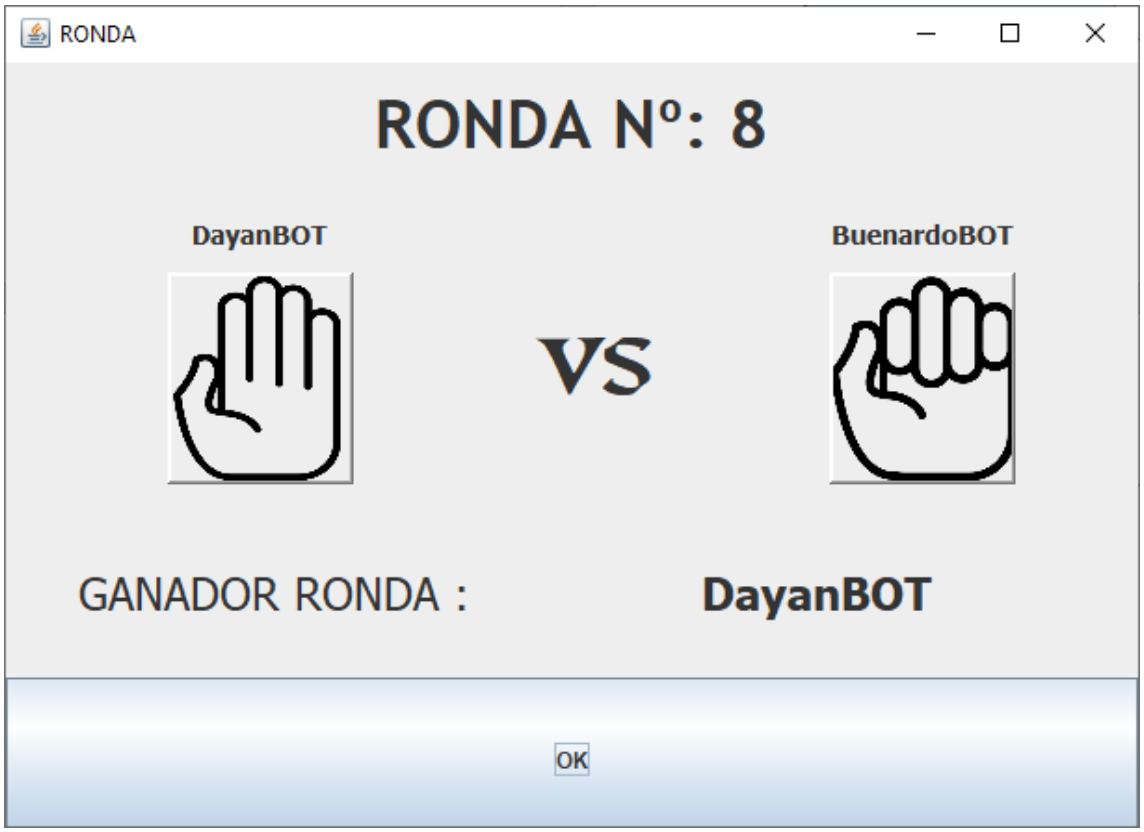

RONDA
RONDA Nº: 3
DayanBOT
BuenardoBOT
VS
GANADOR RONDA :
EMPATE
OK

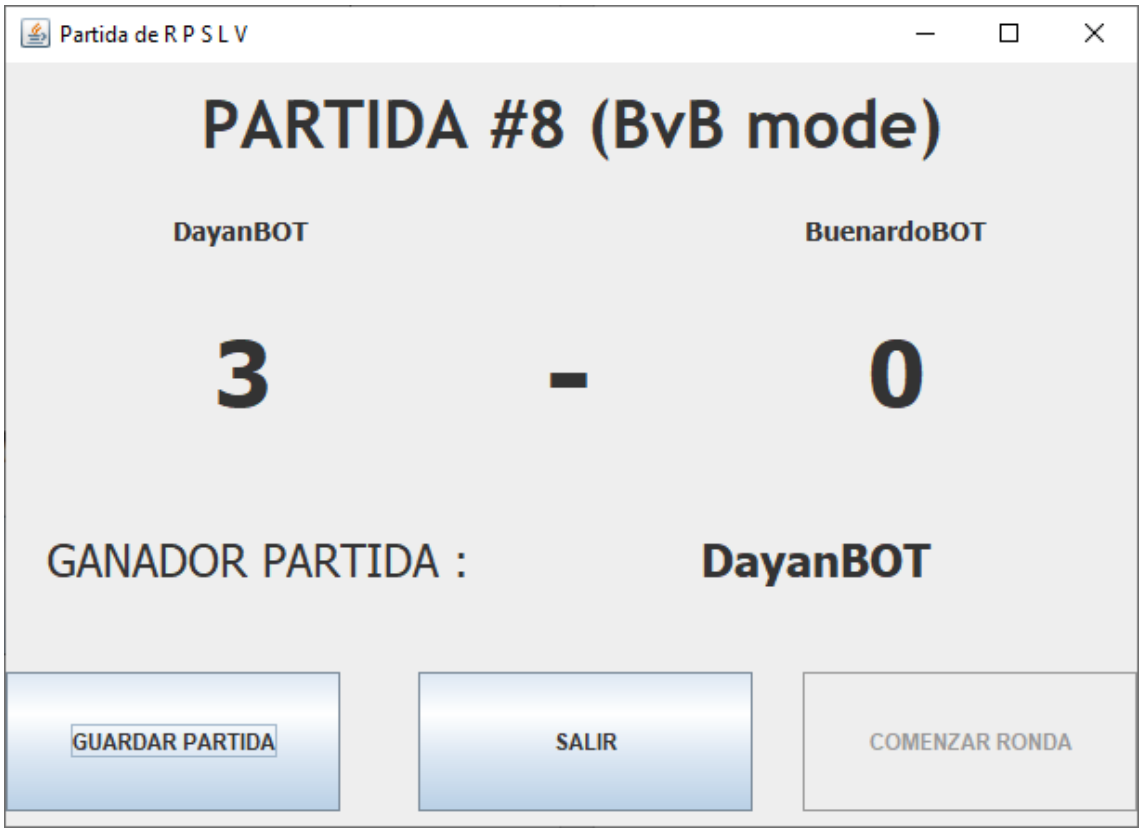

Partida de R P S L V
PARTIDA #8 (BvB mode)
DayanBOT
BuenardoBOT
2
-
0
GANADOR PARTIDA :
POR DECIDIR
GUARDAR PARTIDA
SALIR
COMENZAR RONDA

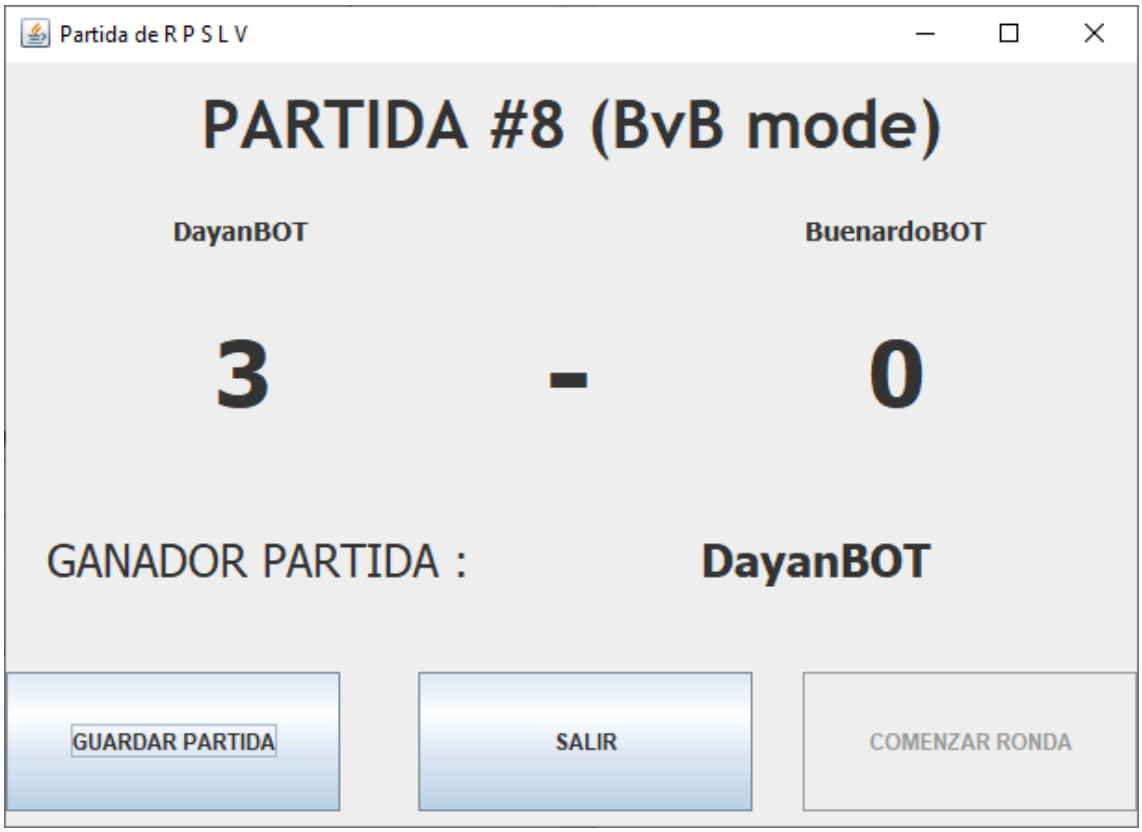

GUARDADO
PARTIDA GUARDADA
OK


Menu Rock Paper Scissor Lizard Spock
LISTA DE PARTIDAS
idPartida
PtosMaximos
tipoPartida
Jugador 1
Tipo J1
Jugador 2
Tipo J2
1 4 BvB Buenard... BOT DayanBOT BOT
2 4 BvB CHAVIX PERSONA DayanBOT BOT
3 3 BvB DayanBOT BOT Buenard... BOT
4 3 JvB Guest01 PERSONA PablitoBOT BOT
5 3 BvB DayanBOT BOT Buenard... BOT
6 3 BvB DayanBOT BOT Buenard... BOT
7 3 BvB DayanBOT BOT Buenard... BOT
8 3 BvB DayanBOT BOT Buenard... BOT
CARGAR
CANCELAR


Partida de R P S L V
PARTIDA #8 (BvB mode)
DayanBOT
BuenardoBOT
2
-
0
GANADOR PARTIDA :
POR DECIDIR
GUARDAR PARTIDA
SALIR
COMENZAR RONDA


RONDA
RONDA Nº: 8
DayanBOT
BuenardoBOT
VS
GANADOR RONDA :
DayanBOT
OK


Partida de R P S L V
PARTIDA #8 (BvB mode)
DayanBOT
BuenardoBOT
3
-
0
GANADOR PARTIDA :
DayanBOT
GUARDAR PARTIDA
SALIR
COMENZAR RONDA



11. Repositorio

- ❖ GIT
- ❖ OneDrive