Contenido

[Capítulo 6. Diseño del Sistema 2](#_Toc103973400)

[6.1 Arquitectura del sistema: Vista de bloques de construcción 2](#_Toc103973401)

[6.1.1 Nivel 1 3](#_Toc103973402)

[6.1.2 Nivel 2 4](#_Toc103973403)

[6.1.3 Nivel 3 4](#_Toc103973404)

[6.2 Descripción detallada de las clases 8](#_Toc103973405)

[6.2.1 Controladores 8](#_Toc103973406)

[6.2.2 Red Neuronal 15](#_Toc103973407)

[6.2.3 Interfaz Gráfica 16](#_Toc103973408)

[6.3 Vista de tiempo de ejecución 19](#_Toc103973409)

[6.3.1 Opciones 19](#_Toc103973410)

[6.3.2 Ciclo de una partida 19](#_Toc103973411)

[6.4 Diseño de la Interfaz 21](#_Toc103973412)

[6.4.1 Acciones del adversario 22](#_Toc103973413)

[6.4.2 Guerreras, armas usadas y favor 22](#_Toc103973414)

[6.4.3 Acciones del jugador 23](#_Toc103973415)

[6.4.4 Mano del jugador 24](#_Toc103973416)

[6.4.5 Acción elegida 25](#_Toc103973417)

[6.4.6 Botón de aceptar 26](#_Toc103973418)

[6.4.7 Pantalla de información final 26](#_Toc103973419)

# Diseño del Sistema

## Arquitectura del sistema: Vista de bloques de construcción

En la siguiente imagen (*Imagen 6.2.a*) podemos ver una vista general de los bloques de construcción que se describirán en los siguientes puntos. La leyenda de la imagen se comparte con la de todos los puntos de esta sección de la documentación.

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente

Imagen 6.2.a – Vista en bloques completa

Leyenda:

|  |  |
| --- | --- |
|  | Persona que interactua con la aplicación |
|  | Bloque de caja negra |
|  | Bloque de caja blanca |
|  | Ficheros de caja negra |
|  | Ficheros de caja blanca |
|  | Clase o archivo de funciones |
|  | Llamada a función sin paso de datos |
|  | Llamada a función con paso de datos |

### Nivel 1

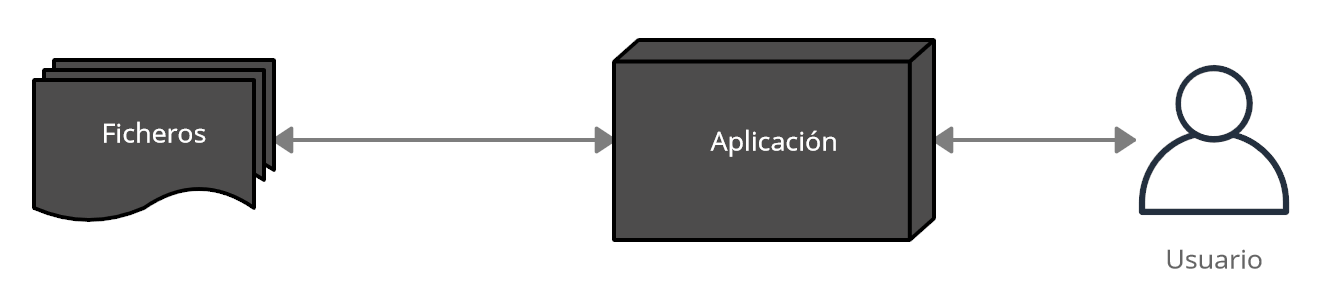


Imagen 6.2.1.a – Vista en bloques de nivel 1

En el nivel 1 podemos ver como la aplicación puede comunicarse y recibir la información del usuario que la esté ejecutando. Cabe destacar que en caso de que se esté ejecutando en modo generación de datos o entrenamiento de la red neuronal el usuario no interactúa con la aplicación, esta comunicación se produce únicamente cuando se ejecuta en modo jugar.

También podemos ver como la aplicación se comunica con ficheros externos. Esta comunicación sí se realiza para los 3 modos de ejecución como veremos más adelante.

### Nivel 2

Diagrama

Descripción generada automáticamente

Imagen 6.2.2.a – Vista en bloques de nivel 2

En el nivel 2 podemos ver como la aplicación tiene 3 módulos principales.

El módulo de controladores está en el centro de ellos porque es el encargado de gestionar el resto, este se comunica con el paquete de la red neuronal tanto para pedir datos como para recibirlos, así como con el de la interfaz gráfica. También puede generar información que se guarda en ficheros.

El módulo de la red neuronal es el encargado de gestionar la parte de entrenamiento y predicción de esta. Hace uso y guarda archivos, además de comunicase con el módulo de controladores.

El paquete de interfaz gráfica es el encargado de comunicar el módulo de controladores con el del usuario.

### Nivel 3

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente

Imagen 6.2.3.a – Vista en bloques de nivel 3

En este nivel podemos ver todas las clases de la aplicación y la relación entre ellas.

Desde el *main* se puede llamar a 3 clases dependiendo del modo de ejecución en el que estemos. Para el modo de generación de datos se llama la clase *ControladorGeneradorDatos*, para el modo de entrenamiento de la red neuronal se llama a la clase *Entrenamiento* y para el modo jugar se llama a la clase *ControladorPartida*.

#### Controladores

Diagrama

Descripción generada automáticamente

Imagen 6.2.3.1.a – Detalle del módulo de controladores de la vista en bloques de nivel 3

En el módulo de controladores tenemos 2 clases que se llaman desde el *main*.

La clase *ControladorGeneradorDatos* sirve para llamar a la clase *ControladorPartida* el número de veces que se establezcan en la parametrización, de esta clase recibe los datos de los tableros y acciones que se han realizado en cada partida y se guardan en los ficheros de datos generados.

La clase *ControladorPartida* es el centro de la aplicación, ya que es la que se encarga de gestionar y comunicar entre sí gran parte del resto de clases. En el caso de que la aplicación se ejecute en modo de generación de datos, la *ControladorPartida* será llamada un numero parametrizado de veces, pero en todas ellas hará lo mismo que si fuera una ejecución en modo jugar, como podemos ver en el apartado 6.3.2 de la documentación. Recibe el tablero de la clase *ControladorTablero* y se la envía a la clase Red Neuronal (*ControladorRedNeuronal*), *ControladorJugador* o *ControladorBot* dependiendo de que jugadores estén jugando esa partida. De esas clases recibirá la acción realizada y se la pasará a clase *ControladorTablero*.

La clase *ControladorJugador* sirve de interfaz intermediaria entre la clase *ControladorPartida* y la interfaz gráfica de usuario.

La clase Red Neuronal (*ControladorRedNeuronal*) sirve de intermediaria entre la clase *ControladorPartida* y la clase de *Predicción* que forma parte del módulo de la red neuronal.

La clase *ControladorTablero* es la que se encarga de toda la gestión del tablero, lo inicializa al inicio de cada turno para enviárselo a la clase *ControladorPartida*, luego recibe las acciones que realicen los diferentes jugadores desde la clase *ControladorPartida* y guarda los cambios que realizan esas acciones en el tablero, devolviéndoselo de nuevo para la siguiente acción.

La clase *ControladorBot* sirve como jugador aleatorio. Esta clase recibe un tablero y realiza una acción completamente aleatoria dentro de las acciones posibles y con unas cartas aleatorias suficientes para realizar esa acción que estén disponibles en la mano de este jugador. Sirve como jugador inicial para el entrenamiento y como jugador de dificultad baja para el modo jugar si así se define en el archivo de parametrización.

Hay que destacar que todas las clases que sirven como jugadores de la partida tienen un patrón de métodos públicos definido. Este patrón se compone de los métodos:

* \_\_init\_\_(miNombre, miNumero)
* decidirAccion(tablero)
* decidirAccionDeSeleccion(tablero)
* finish()

Estos métodos serán descritos en su correspondiente clase más adelante.

#### Red Neuronal

Diagrama

Descripción generada automáticamente

Imagen 6.2.3.2.a - Detalle del módulo de la red neuronal de la vista en bloques de nivel 3

En el módulo de la red neuronal hay únicamente 2 clases, la clase de *Entrenamiento* y la de *Predicción*.

La clase de *Entrenamiento* se ejecuta directamente desde el main, lee los archivos de datos y genera los archivos del modelo.

La clase de *Predicción* carga los archivos del modelo, recibe el tablero de la clase de red neuronal y le devuelve una predicción de una acción que será la que realice como jugador.

#### Interfaz grafica

Diagrama

Descripción generada automáticamente

Imagen 6.2.3.2.a - Detalle del módulo de la interfaz gráfica de la vista en bloques de nivel 3

En la interfaz gráfica tenemos dos clases que utilizan Tkinter, la clase *GUI\_Tkinter* lo utiliza para hacer la ventana gráfica donde el usuario interactuará como jugador con la aplicación, de manera que esta clase pinta el tablero y ofrece al usuario las posibles acciones que puede realizar. Una vez seleccionada la acción esta vuelve a los controladores para seguir la partida.

En este módulo también existe la clase auxiliar *Popup\_Tkinter* sirve para informar al usuario de la finalización del programa.

## Descripción detallada de las clases

A continuación, se va a hacer una descripción de las clases con sus de la aplicación separadas por los paquetes que se han explicado en el punto anterior (Arquitectura del sistema: Vista de bloques de construcción).

### Controladores

|  |
| --- |
| **Nombre de la Clase** |
| ControladorPartida |
| **Descripción de la clase** | |
| Clase controladora de la partida, se encarga de los turnos y las rondas además de guardar las acciones que han llevado a cabo los jugadores. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_controladorTablero | Instancia del controlador del tablero en el que se guarda el estado del tablero y se ejecutan las acciones. |
| \_\_accionesJ1 | Cadena de texto en la que se guardan las acciones que ha ejecutado el jugador 1 para su futuro guardado. |
| \_\_accionesJ2 | Cadena de texto en la que se guardan las acciones que ha ejecutado el jugador 2 para su futuro guardado. |
| \_\_tableroJ1 | Cadena de texto en la que se guardan los tableros que le llegaron al jugador 1 y sobre los cuales formo la acción correspondiente. |
| \_\_tableroJ2 | Cadena de texto en la que se guardan los tableros que le llegaron al jugador 2 y sobre los cuales formo la acción correspondiente. |
| \_\_winnnerNumero | Guarda el número del ganador al final de cada ronda para poder saber si se ha terminado la partida. |
| \_\_winner | Copia de la instancia del jugador que ha ganado la partida. |
| \_\_jugador1 | Instancia del jugador correspondiente al orden 1. |
| \_\_jugador2 | Instancia del jugador correspondiente al orden 2. |
| \_\_jugador1Inicio | Instancia del jugador copia de jugador 1, pero que no cambia a lo largo de la partida dependiendo del turno, se usa para declarar el ganador. |
| \_\_jugador2Inicio | Instancia del jugador copia de jugador 1, pero que no cambia a lo largo de la partida dependiendo del turno, se usa para declarar el ganador. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase ControladorPartida, se definen todos los atributos privados de la misma. |
| run | Método ejecutor de la clase ControladorPartida, inicializa los jugadores y ejecuta el método \_\_start con una captura de posibles errores que se mostraran en un popup. |
| \_\_initJugadores | Método que instancia los atributos de jugador1 y jugador2 dependiendo del modo y nivel seleccionados en la parametrización. |
| \_\_start | Método encargado de realizar el bucle de rondas hasta que en una de ellas haya un ganador. |
| \_\_ronda | Método encargado de inicializar la ronda correspondiente y realizar el bucle de las acciones de los jugadores. |
| \_\_accion | Método encargado de pedir a los jugadores que realicen una acción, si la acción requiere de interacción por parte del otro jugador también se le pedirá que haga la selección. |
| \_\_guardarAccion | Método encargado de guardar la acción en formato cadena de texto y el tablero dados en su correspondiente atributo de la clase. |
| getAccionesJ1 | Método para obtener el atributo de tipo cadena de texto: accionesJ1. |
| getAccionesJ2 | Método para obtener el atributo de tipo cadena de texto: accionesJ2. |
| getTablerosJ1 | Método para obtener para el atributo de tipo cadena de texto: tablerosJ1. |
| getTablerosJ2 | Método para obtener para el atributo de tipo cadena de texto: tablerosJ1. |
| getWinner | Método para obtener para el atributo winner que implementa una clase jugador. |

Tabla 6.2.1.a – Descripción de la clase ControladorPartida

|  |
| --- |
| **Nombre de la Clase** |
| ControladorTablero |
| **Descripción de la clase** | |
| Clase controladora del tablero de juego, almacena la información de la situación actual del tablero en la que permite realizar acciones. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_tablero | Guarda una matriz con el estado actual del tablero. |
| \_\_mazoArmas | Guarda un array con las cartas que quedan en el mazo de robo. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase ControladorTablero, se definen e inicializan todos los atributos privados de la misma. |
| \_\_initMazo | Método que inicializa el mazo de las cartas con todas las cartas de armas disponibles. |
| \_\_borrarAcciones | Método que se encarga de borrar del tablero las acciones realizadas por ambos jugadores. |
| \_\_borrarArmas | Método que se encarga de borrar del tablero las armas usadas por ambos jugadores. |
| \_\_robarCarta | Método que elimina del mazo de cartas una carta aleatoria y la devuelve. Lanza una excepción si el mazo no tiene cartas que robar. |
| \_\_repartoDeCartas | Método que se encarga de repartir 6 cartas del mazo a cada jugador y ordenar las manos de dichos jugadores. |
| \_\_conseguirCarta | Método que asigna una carta del mazo a la mano del jugador con el numero dado por parámetro. |
| \_\_soltarCarta | Método que recibe el array de la mano y el valor de la carta que sustituye por 0. |
| \_\_eliminarCarta | Método que recibe un array y un valor, devuelve el array eliminando el valor una vez. |

|  |  |
| --- | --- |
| \_\_getFilaAcciones | Método que recibe el índice del jugador y devuelve el índice de sus acciones usadas. |
| \_\_getMano | Método que recibe el índice del jugador y devuelve el índice de su mano. |
| \_\_ordenarMano | Método que recibe el índice de la mano del jugador y ordena su mano. |
| \_\_getNumeroCartasEn  AccionSeleccionada | Método que devuelve el número de cartas que contiene el array que se le pasa por parámetro, sin incluir la primera posición que corresponde al tipo de acción. Además, lanza una excepción si las cartas no están bien ordenadas en el array y contiene ceros en medio. |
| \_\_comprobarAccion1 | Método que lanza una excepción si el número de cartas no concuerda con el tipo de acción 1 o si esta acción ya ha sido usada. |
| \_\_comprobarAccion2 | Método que lanza una excepción si el número de cartas no concuerda con el tipo de acción 2 o si esta acción ya ha sido usada. |
| \_\_comprobarAccion3 | Método que lanza una excepción si el número de cartas no concuerda con el tipo de acción 3 o si esta acción ya ha sido usada. |
| \_\_comprobarAccion4 | Método que lanza una excepción si el número de cartas no concuerda con el tipo de acción 4 o si esta acción ya ha sido usada. |
| \_\_comprobarAccion  Decision3 | Método que lanza una excepción si el número de cartas no concuerda con el tipo de acción de decisión 3 o si esta acción no concuerda con la esperada. |
| \_\_comprobarAccion  Decision4 | Método que lanza una excepción si el número de cartas no concuerda con el tipo de acción de decisión 4 o si esta acción no concuerda con la esperada. |
| \_\_guardarAccion1 | Método que guarda la carta seleccionada del array dado por parámetro para la acción 1 en la fila de acciones del tablero dada por parámetro y elimina esa carta de la mano dada por parámetro. |
| \_\_guardarAccion2 | Método que guarda la carta seleccionada del array dado por parámetro para la acción 2 en la fila de acciones del tablero dada por parámetro y elimina esa carta de la mano dada por parámetro. |
| \_\_guardarAccion3 | Método que guarda las cartas seleccionadas del array dado por parámetro la acción 3 en la fila de acción pendiente, establece la acción 3 como usada en la fila de acciones del tablero dada por parámetro y elimina esas cartas de la mano dada por parámetro. |
| \_\_guardarAccion4 | Método que guarda las cartas seleccionadas del array dado por parámetro la acción 4 en la fila de acción pendiente, establece la acción 4 como usada en la fila de acciones del tablero dada por parámetro y elimina esas cartas de la mano dada por parámetro. |
| \_\_guardarAccionDecision3 | Método que guarda las cartas seleccionadas del array dado por parámetro para la acción de decisión 3 en la fila de acción pendiente, sumamos las cartas seleccionadas a los jugadores en las filas de armas usadas y deja la fila de acción pendiente vacía. |

|  |  |
| --- | --- |
| \_\_guardarAccionDecision4 | Método que guarda las cartas seleccionadas del array dado por parámetro para la acción de decisión 4 en la fila de acción pendiente, sumamos las cartas seleccionadas a los jugadores en las filas de armas usadas y deja la fila de acción pendiente vacía. |
| \_\_guardarSecreto | Método que toma el valor de la carta de secreto de la fila del jugador seleccionada y la guarda en la fila de armas del jugador seleccionada. |
| \_\_sumarCarta | Método que dada una carta suma 1 a la columna correspondiente a esa carta y a la fila dada. |
| \_\_getGanador | Método que calcula todos los puntos de cada jugador y devuelve el ganador. |
| \_\_manoLlena | Método que comprueba el número de cartas en la mano dada y devuelve y está llena o no. |
| initRonda | Método que inicializa el mazo con todas las cartas menos una y reparte las cartas iniciales. |
| jugadorRobaCarta | Método que, dado el índice del jugador, roba una carta del mazo de cartas y la guarda en la mano de dicho jugador. |
| getVistaTablero | Método que, dado el índice del jugador, devuelve una matriz de información parcial para ese jugador. |
| realizarAccion | Método que, dado el índice del jugador y una acción, comprueba que ese jugador puede realizar la acción, que esta está bien formada y de ser así la ejecuta haciendo los cambios correspondientes en el tablero. |
| hayAccionPendiente | Método que, dado el índice del jugador y una acción, comprueba que ese jugador puede realizar la acción, que esta está bien formada y de ser así ejecuta haciendo los cambios correspondientes en el tablero. |
| finalizarTurno | Método que realiza todas las acciones necesarias para dar el turno por terminado y devuelve el jugador ganador en caso de que lo hubiera. |

Tabla 6.2.1.b – Descripción de la clase ControladorTablero

|  |
| --- |
| **Nombre de la Clase** |
| ControladorBot |
| **Descripción de la clase** | |
| Clase controladora del jugador de bot aleatorio. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_miNombre | Define el nombre para leerlo en los logs. |
| \_\_miNumero | Define el orden del jugador, puede ser 1 o 2. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase ControladorBot, recibe el nombre y el numero para guardarlo en sus respectivos atributos. |
| decidirAccion | Método para generar una acción aleatoria, recibe la matriz del tablero y devuelve un array con una acción correcta aleatoria dentro de las posibles con las cartas aleatorias que tenga en mano. |
| decidirAccionDeSeleccion | Método para generar la acción de selección pendiente con cartas aleatorias, recibe la matriz del tablero y devuelve un array con la acción correctamente formada con las cartas aleatorias dadas por la acción pendiente. |
| finish | Método que sirve para cerrar los hilos pendientes de los jugadores, en este caso no es necesario cerrar ninguno. |
| getMiNombre | Método para obtener para el atributo de tipo cadena de texto: miNombre. |
| getMiNumero | Método para obtener para el atributo de tipo cadena de texto: miNumero. |

Tabla 6.2.1.c – Descripción de la clase ControladorBot

|  |
| --- |
| **Nombre de la Clase** |
| ControladorJugador |
| **Descripción de la clase** | |
| Clase controladora del jugador controlado por el usuario a través de la interfaz gráfica. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_miNombre | Define el nombre para leerlo en los logs. |
| \_\_miNumero | Define el orden del jugador, puede ser 1 o 2. |
| \_\_GUI | Instancia la clase GUI\_Tkinter que corresponde a la interfaz gráfica de usuario con la que interactuará el usuario. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase ControladorJugador, recibe el nombre y el numero para guardarlo en sus respectivos atributos. Además, inicializa el atributo GUI que implementa la clase GUI\_Tkinter. |
| decidirAccion | Método para generar una acción, recibe la matriz del tablero y devuelve un array con una acción correcta que será seleccionada por el usuario a través del Método \_\_pedirAccion. |
| decidirAccionDeSeleccion | Método para generar la acción de selección pendiente, recibe la matriz del tablero y devuelve un array con la acción correctamente formada con las cartas seleccionadas por el usuario a través del Método \_\_pedirAccion. |
| \_\_pedirAccion | Método que unifica los Métodos de decidirAccion y decidirAccionDeSeleccion en el que se llama a los Métodos de printTabla, start y obtenerAccion para pintar el tablero en la interfaz de usuario y esperar a que el usuario seleccione la acción que desee. |
| finish | Método que sirve para cerrar el hilo de la interfaz gráfica. |
| getMiNombre | Método para obtener para el atributo de tipo cadena de texto: miNombre. |
| getMiNumero | Método para obtener para el atributo de tipo cadena de texto: miNumero. |

Tabla 6.2.1.d – Descripción de la clase ControladorJugador

|  |
| --- |
| **Nombre de la Clase** |
| ControladorRedNeuronal |
| **Descripción de la clase** | |
| Clase controladora del jugador controlado por la red neuronal entrenada anteriormente. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_miNombre | Define el nombre para leerlo en los logs. |
| \_\_miNumero | Define el orden del jugador, puede ser 1 o 2. |
| \_\_prediccion | Instancia la clase Prediccion que corresponde a la parte de la red neuronal encargada de generar predicciones. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase ControladorRedNeuronal, recibe el nombre y el numero para guardarlo en sus respectivos atributos. Además, inicializa el atributo Prediccion que implementa la clase Prediccion. |
| decidirAccion | Método para generar una acción, recibe la matriz del tablero y devuelve un array con una acción correcta que será seleccionada por la red neuronal y procesada por el Método \_\_procesarAccion. |
| \_\_procesarAccion | Método encargado de procesar el resultado emitido por la red neuronal para transformarlo en una acción correctamente formada y válida para el tablero actual. Esto se debe a que la red neuronal devuelve el valor en porcentajes de acierto, que no están exentos de fallos. |
| \_\_obtenerCartasEnMano  YAccionesPosibles | Método encargado de devolver una lista con las cartas que hay en la mano y otra lista con las acciones posibles que puede realizar el jugador para el tablero dado. Este Método es necesario para el Método \_\_procesarAccion. |
| \_\_obtenerAccionCount | Método que devuelve el número de cartas que debe tener la acción a realizar dada por parámetro. Este método es necesario para el Método \_\_procesarAccion. |
| \_\_eliminarCarta | Método que devuelve un array con las cartas en mano a la que se le ha retirado la carta seleccionada. Este Método es necesario para el Método \_\_procesarAccion. |
| decidirAccionDeSeleccion | Método para generar la acción de selección pendiente, recibe la matriz del tablero y devuelve un array con la acción correctamente formada con las cartas seleccionadas por red neuronal y procesada por el Método \_\_procesarAccionDeSeleccion. |
| \_\_procesarAccionDe  Seleccion | Método encargado de procesar el resultado emitido por la red neuronal para transformarlo en una acción de selección correctamente formada y válida para el tablero actual. Esto se debe a que la red neuronal devuelve el valor en porcentajes de acierto, que no están exentos de fallos. |
| \_\_seleccionarCartasAccion  DeSeleccionRegalo | Método encargado de procesar el resultado emitido por la red neuronal para la acción de selección específica de tipo regalo para una acción pendiente dada. |
| \_\_seleccionarCartasAccion  DeSeleccionCompeticion | Método encargado de procesar el resultado emitido por la red neuronal para la acción de selección específica de tipo competición para una acción pendiente dada. |

|  |  |
| --- | --- |
| \_\_crearAccionCompleta | Método que crea un array acción a partir del tipo de acción y las cartas seleccionadas. |
| finish | Método que sirve para cerrar los hilos pendientes de los jugadores, en este caso no es necesario cerrar ninguno. |
| getMiNombre | Método para obtener para el atributo de tipo cadena de texto: miNombre. |
| getMiNumero | Método para obtener para el atributo de tipo cadena de texto: miNumero. |

Tabla 6.2.1.e – Descripción de la clase ControladorRedNeuronal

|  |
| --- |
| **Nombre de la Clase** |
| ControladorGeneradorDatos |
| **Descripción de la clase** | |
| Clase controladora de la generación de datos para el entrenamiento de la red neuronal. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_partidasGanadas1 | Define el número de partidas que ha ganado el primer jugador. |
| \_\_partidasGanadas2 | Define el número de partidas que ha ganado el segundo jugador. |
| \_\_controladorPartida | Instancia la clase ControladorPartida, que se va instanciando con cada nueva simulación. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase ControladorGeneradorDatos, se inicializan los atributos de la clase. |
| run | Método ejecutor de la clase ControladorGeneradorDatos, reinicia los archivos y ejecuta partidas un número de veces parametrizado. |
| \_\_guardarGanador | Método que dado un jugador obtiene los datos de su partida y los guarda en los ficheros de partidas ganadas. |
| \_\_resetArchivo | Método que elimina los archivos generados anteriormente y los deja limpios para una nueva generación. |
| \_\_guardarEnArchivo | Método de escritura en fichero del texto dado. |

Tabla 6.2.1.f – Descripción de la clase ControladorGeneradorDatos

### Red Neuronal

|  |
| --- |
| **Nombre de la Clase** |
| Entrenamiento |
| **Descripción de la clase** | |
| Clase encargada de utilizar los datos de entrenamiento para guardar el modelo entrenado. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_cnn | Contendrá una instancia de la clase Sequential de TensorFlow. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase Entrenamiento, se define el atributo cnn. |
| run | Método ejecutor de la clase Entrenamiento, se realiza el preprocesado de datos, la creación del modelo y el entrenamiento. |
| \_\_preProcesadoDeDatos | Método encargado de cargar los datos de entrenamiento desde los ficheros y transformarlos en matrices y arrays que reconoce la red neuronal. |
| \_\_creacionModelo | Método encargado de instanciar el atributo cnn con la clase Sequential. |
| \_\_establecerCapas | Método encargado de definir las capas y filtros que va a tener la red neuronal. |
| \_\_complileAndFit | Método encargado de compilar el modelo para su entrenamiento. |
| \_\_guardarModelo | Método encargado de guardar el modelo generado con los pesos de este en los archivos .h5 para su posterior carga por parte de la clase Prediccion. |

Tabla 6.2.2.a – Descripción de la clase Entrenamiento

|  |
| --- |
| **Nombre de la Clase** |
| Prediccion |
| **Descripción de la clase** | |
| Clase encargada de utilizar el modelo entrenado para generar predicciones. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_cnn | Contendrá una instancia de la clase Sequential. |
| \_\_resultado | Contendrá el resultado de la última predicción realizada. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase Entrenamiento, se define los atributos e inicializarlos. |
| \_\_cargarModelo | Método encargado de cargar el modelo entrenado de los ficheros .h5. |
| predecir | Método que, dado un tablero de entrada, genera una predicción y la guarda en el atributo resultado. |
| obtenerPrediccionCampo | Método que, dado un el índice del campo del que se desea obtener la predicción realizada y los posibles valores aceptables, devuelve el valor más probable. |

Tabla 6.2.2.b – Descripción de la clase Prediccion

### Interfaz Gráfica

|  |
| --- |
| **Nombre de la Clase** |
| GUI\_Tkinter |
| **Descripción de la clase** | |
| Clase encargada de generar la ventana principal de la aplicación usando la librería tkinter. | |
| **Atributos** | |
| **Nombre** | **Descripción** |
| \_\_accionGuardada | Array en el que se van guardando la información de la acción que se está generando. |
| \_\_cartasRestantes | El número de cartas que faltan para terminar de completar la acción actual. |
| \_\_accionPendiente | Array con la acción pendiente que llega. |
| \_\_window | Instancia la clase Tk de tkinter. |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| \_\_init\_\_ | Método constructor de la clase GUI\_Tkinter, se definen e inicializan todos los atributos privados de la misma. |
| printTabla | Método encargado de pintar el tablero que nos llega. |
| \_\_limpiarAccion | Método encargado de vaciar el array de acción guardada y borrar del tablero la acción que se había seleccionado. |
| \_\_comprobarAccion  Pendiente | Método encargado de comprobar si existe una acción pendiente y modificar el tablero para que se muestre la misma. |
| \_\_bloquearAcciones  NormalesYMano | Método encargado de deshabilitar todos los botones de las acciones y de la mano. |
| \_\_addGuerreras | Método encargado de añadir una fila de imágenes de cartas grandes. |
| \_\_addMarcadores | Método encargado de añadir una fila de marcadores dada una lista de números. |
| \_\_addSusAcciones | Método encargado de añadir una fila con las acciones usadas del adversario. |
| \_\_addMisAcciones | Método encargado de añadir una fila con las acciones usadas del jugador. |
| \_\_addMiMano | Método encargado de añadir una fila con las cartas de la mano del jugador. |
| \_\_addAccionSeleccionada | Método encargado de añadir el label con la acción seleccionada. |
| \_\_addAccionPendiente | Método encargado de añadir las cartas de la acción pendiente. |
| \_\_addAccionPendiente5 | Método encargado de añadir las cartas de la acción pendiente para el tipo 5. |
| \_\_addAccionPendiente6 | Método encargado de añadir las cartas de la acción pendiente para el tipo 6. |
| \_\_addCartaGrande | Método para añadir una imagen de carta grande en la posición dada con la imagen dada. |
| \_\_addMarcadorValor | Método para añadir el marcador del valor en la posición dada con la imagen dada. |
| \_\_addMarcador | Método para añadir marcadores en la posición dada con el texto dado. |
| \_\_addAccionPropia | Método para añadir un botón, en la posición dada, de acción activa/inactiva dependiendo del tipo dado. |

|  |  |
| --- | --- |
| \_\_addAccionEnemiga | Método encargado de añadir una imagen de acción realizada/no realizada en la posición dada. |
| \_\_getTextoAccion | Método encargado de seleccionar el texto para un tipo dado. |
| \_\_addCartaPeque | Método encargado de añadir un botón con una carta pequeña. |
| \_\_addCartaAccionPendiente | Método encargado de añadir un botón de una carta de acción pendiente. |
| \_\_addCartaOculta | Método encargado de añadir un botón con una carta oculta. |
| \_\_addMarcoExplicativo | Método encargado de añadir un botón con una carta oculta. |
| \_\_addLabelConImagen | Método encargado de añadir un label con una imagen para una posición, con un tamaño, imagen, texto y borde dados y numero de filas = 1. |
| \_\_addLabelConImagenY  TamanioFilas | Método encargado de añadir un label con una imagen para una posición, con un tamaño, imagen, texto, numero de filas y borde dados. |
| \_\_addLabelConSoloTexto | Método encargado de añadir un label in imagen para una posición, con un tamaño, texto y borde dados. |
| \_\_addBotonConImagen | Método encargado de añadir un botón con una imagen para una posición, con un tamaño, imagen, texto, numero de filas y borde dados. |
| \_\_addButtonConTexto | Método encargado de añadir un botón con un texto para una posición, con una imagen, texto y borde dados. |
| \_\_seleccionarAccion | Método que se ejecuta al dar a un botón de selección de acción. Elimina la acción anteriormente seleccionada y pinta la nueva para poder seleccionar las cartas correspondientes. |
| \_\_seleccionarCarta | Método que se ejecuta al seleccionar una carta, en caso de que la acción admita una nueva carta pinta la carta seleccionada en la sección de la acción seleccionada y bloquea la carta de la mano. |
| \_\_seleccionarCarta  Pendiente5 | Método que se ejecuta al seleccionar una carta cuando esta la acción de selección pendiente de tipo 5. |
| \_\_seleccionarCarta  Pendiente6 | Método que se ejecuta al seleccionar una carta cuando esta la acción de selección pendiente de tipo 6. |
| \_\_noAccion | Método auxiliar para no hacer nada al pulsar acciones que no están disponibles. |
| \_\_printAceptar | Método que pinta el botón de aceptar para enviar la acción seleccionada. |
| \_\_borrarAceptar | Método que borra el botón de aceptar para que no se envíe una acción a medias. |
| start | Método que inicia el bucle de tkinter. |
| \_\_pressAceptar | Método sale del bucle de tkinter para poder seguir con la ejecución del programa y aplicar la acción seleccionada. |
| cerrar | Método que destruye la ventana de tkinter cuando se termina la partida. |
| obtenerAccion | Método que devuelve el atributo de accionGuardada. |

Tabla 6.2.3.a – Descripción de la clase GUI\_Tkinter

|  |
| --- |
| **Nombre de la Clase** |
| Popup\_Tkinter |
| **Descripción de la clase** | |
| Clase encargada de generar la ventana pequeña de información (o popup). | |
| **Métodos** | |
| **Nombre** | **Parámetros y descripción** |
| sendMensaje | Método que construye el popup con el texto dado y un botón de aceptar para cerrarlo. |

Tabla 6.2.3.b – Descripción de la clase Popup\_Tkinter

## Vista de tiempo de ejecución

### Opciones

Como hemos visto anteriormente existen 3 opciones parametrizables a la hora de ejecutar la aplicación. Estas opciones se arrancan desde el mismo main y llaman a su clase inicial correspondiente.

* Generar datos
* Entrenar a la red neuronal
* Jugar

La vista de tiempo de ejecución de entrenar a la red neuronal es la más simple de las 3, ya que únicamente se establece el modelo de la red neuronal y se arranca, no hay más que el uso de una sola clase.

Las vistas de generación de datos y jugar es prácticamente la misma, únicamente se diferencian en quienes son los jugadores de cada partida y el número de partidas que se juegan.

Para la opción de generación de datos el *main* llaman a la clase *ControladorGeneradorDatos*, que se encarga de hacer las llamadas a la clase *ControladorPartida* y guardar los datos en un archivo.

Para la opción de Jugar se llama desde el *main* directamente a la clase *ControladorPartida* que es la encargada de gestionar el ciclo completo de ejecución de una partida.

### Ciclo de una partida

Como acabamos de ver la clase *ControladorPartida* se encarga de gestionar la partida con un bucle que realiza una ronda completa hasta que uno de los 2 jugadores gana.

Cada ronda se compone de a su vez de un bucle de 4 turnos tal y como se ve representado en la imagen 6.3.2.a.

Al inicio de cada ronda la clase *ControladorPartida* pide a la clase *ControladorTablero*.

En cada ciclo del turno cada jugador realiza una acción, siempre en el mismo orden de jugadores para cada turno y intercambiando el orden para cada ronda. El jugador inicial recibe el estado actual del tablero y decide que acción realizar. Si la acción es la acción de “secreto” o la acción de “renuncia” (es decir, es una acción simple) esta acción llega a la clase *ControladorPartida* que se encarga de enviarla a la clase *ControladorTablero* para que haga los cambios correspondientes en el tablero y devolvérselo a la clase *ControladorPartida*. Si la acción es de “regalo” o “competición” (es decir, una acción compleja) después de que el jugador realice la acción y la clase *ControladorTablero* devuelva el resultado en el tablero igual que en una acción simple, la clase *ControladorPartida* envía la acción de decisión en el tablero al jugador 2 para que elija las cartas, este una vez ha elegido las cartas le devuelve esa acción a la clase *ControladorPartida* y esta vuelve a enviársela a la clase *ControladorTablero* para que guarde dicha decisión en el tablero.

Una vez terminada la acción del jugador 1 se realiza la acción del jugador 2.

A modo de resumen, cada partida consta de al menos 1 ronda. Cada ronda consta de 4 turnos en los que se realizan 2 acciones, una por cada jugador.

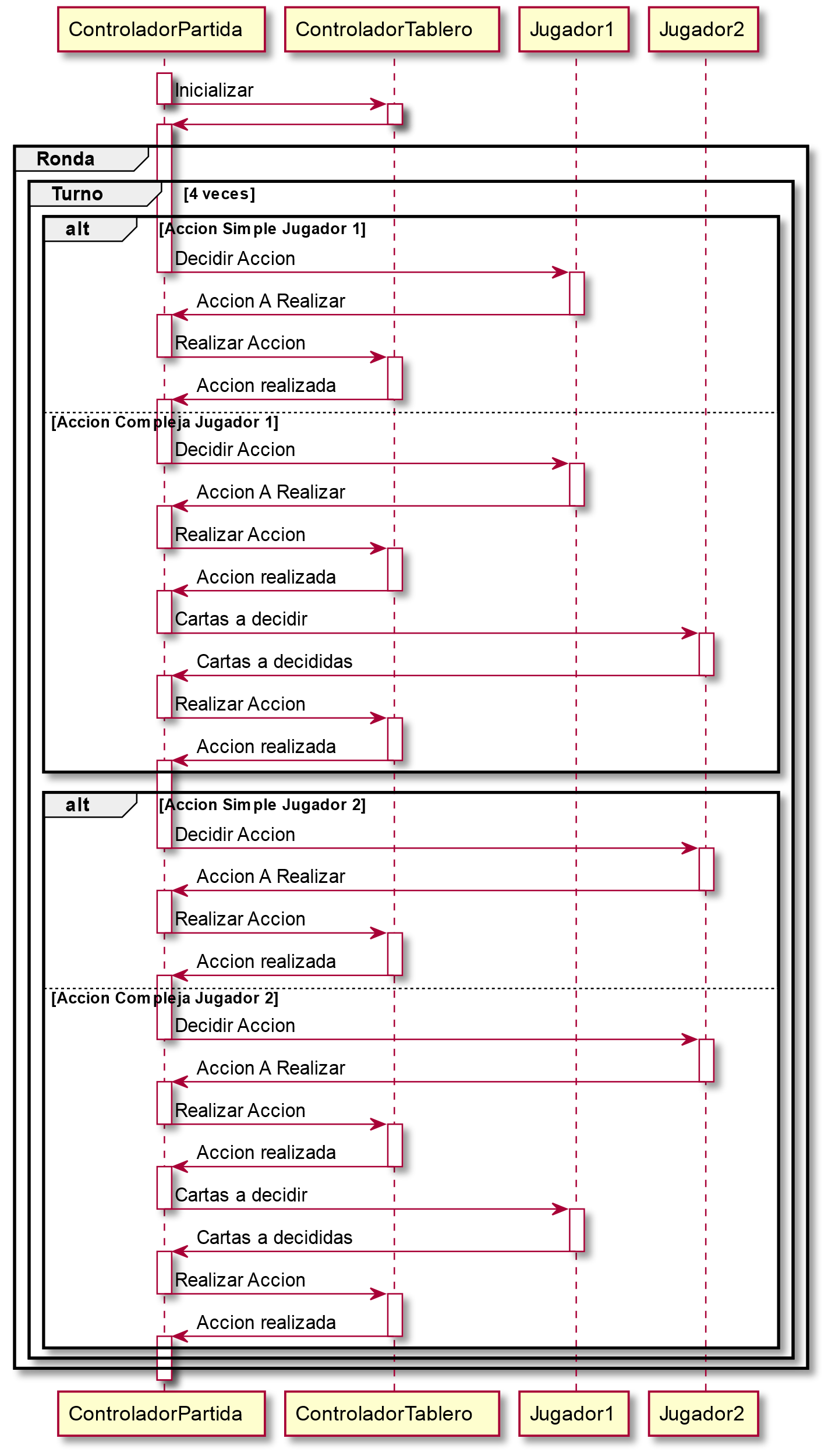


Imagen 6.3.2.a – Diagrama de flujo de una partida

## Diseño de la Interfaz

La interfaz gráfica se compone de una única pantalla en la que el usuario tiene una visión parcial de la situación actual del tablero.

Se compone de una matriz de 7x7 casillas donde cada fila corresponde a un conjunto diferente de datos que explicaremos a continuación.

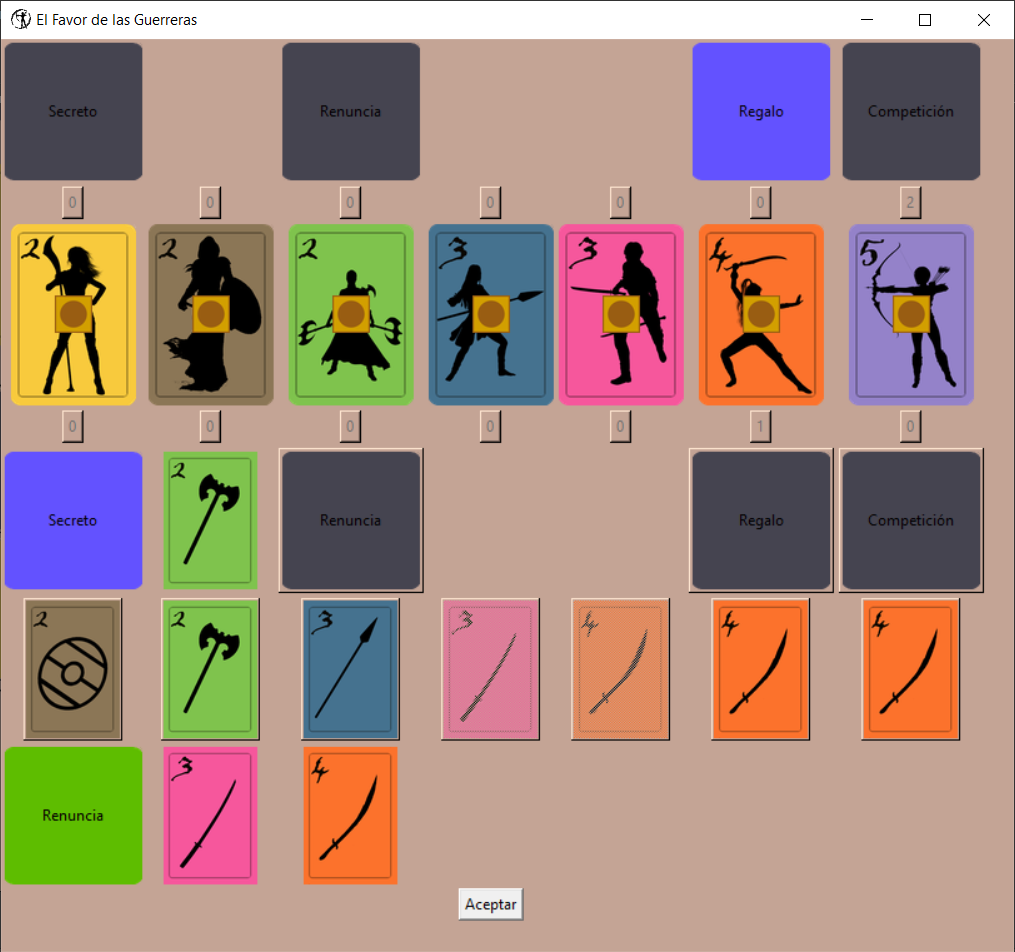


Imagen 6.4.a – Ejemplo de interfaz de usuario

### Acciones del adversario



Imagen 6.4.1.a – Ejemplo de acciones del adversario sin utilizar

Al inicio de la partida el adversario no habrá utilizado ninguna de las acciones, por lo que se mostraran todas en un color oscuro indicando esto tal y como se muestra en la Imagen 4.1.1.a.



Imagen 6.4.1.b – Ejemplo de acciones del adversario utilizadas

Cuando el adversario utilice alguna acción esta pasará a estar en un color azul más claro.

Existe un espacio entre la acción de secreto y la acción de renuncia que representa la carta que el adversario haya usado para la acción de secreto. Si no se ha utilizado el hueco estará vacío, mientras que si se ha utilizado aparecerá una carta gris, ya que es una carta oculta y el jugador no puede saber que carta es.

También existen dos espacios entre la acción de renuncia y la de regalo que representan las dos cartas de renuncia del adversario en caso de haber usado ya esta acción al igual que con la carta de secreto.

### Guerreras, armas usadas y favor

**

Imagen 6.4.2.a – Ejemplo de visualización de las guerreras

Estas 3 filas están orientadas, es decir, la parte superior corresponde al adversario y la parte inferior corresponde al jugador.

La fila del medio representa a las guerreras, y el marcador central representan su favor actual. Al inicio de la partida este favor estará siempre en el centro representando que son neutrales. Sin embargo, tras la primera ronda, si ningún jugador ha ganado el favor cambiará dependiendo de las acciones de la ronda anterior de manera que estará en la parte superior de la carta de la guerrera en caso de que esa guerrera este a favor del adversario (imagen 4.1.2.b) o en la parte inferior en caso de este a favor del jugador (imagen 4.1.2.c).

|  |  |
| --- | --- |
| Imagen 6.4.2.b Ejemplo de guerrera con el favor del adversario | Imagen 6.4.2.c – Ejemplo de guerrera con el favor del jugador |
|  |  |

Los marcadores numéricos que aparecen encima y debajo de las cartas de las guerreras representan el número de cartas que cada jugador ha jugador en favor de esa guerrera durante esta ronda. Los marcadores superiores corresponden al adversario y los inferiores al jugador. Hay que tener en cuenta, las acciones de secreto y renuncia guardan las cartas en la parte de acciones usadas y no se tienen en cuenta hasta el final de la ronda.

### Acciones del jugador



Imagen 6.4.3.a – Ejemplo de acciones del jugador sin utilizar

Esta fila es muy similar a la de acciones del adversario al inicio, pero tiene dos diferencias fundamentales. Para empezar las acciones son interactivas, por lo que el usuario puede pulsar encima de ellas para seleccionarla. Cuando se selecciones una acción aparecerá marcada más abajo. Si se desea se puede cambiar de acción seleccionando una diferente, además seleccionar de nuevo una acción reinicia las cartas que hayan sido seleccionadas para esta acción.



Imagen 6.4.3.b – Ejemplo de acciones del jugador utilizadas

Al igual que con las acciones del adversario cuando ya han sido utilizadas en esta ronda se marcan en azul más claro, pero a diferencia del adversario en esta ocasión si se podrán ver las cartas seleccionadas para las acciones de secreto y renuncia. Además, las acciones utilizadas dejan de ser interactiva, ya que el usuario no puede volver a utilizarlas.



Imagen 6.4.3.b – Ejemplo de acciones del jugador bloqueadas

Cuando el jugador se encuentre con una acción de decisión en la que tenga que elegir cartas del adversario, las acciones del jugador quedarán bloqueadas, tornándose en un tono más grisáceo, pero mostrando diferencia entre las acciones usadas y no usadas, así como las cartas usadas para las acciones de secreto y renuncia para poder tomar una decisión con toda la información del tablero.

### Mano del jugador



Imagen 6.4.4.a – Ejemplo de mano del jugador con todas las cartas disponibles

En esta fila encontramos las cartas que tiene en la mano el jugador. El color y el símbolo representan a que guerrera corresponden. Estas cartas son interactivas, para poder seleccionarlas hay que haber elegido antes una acción, y dependiendo de la acción elegida se podrán seleccionar un numero distinto de cartas.



Imagen 6.4.4.b – Ejemplo de mano del jugador con cartas seleccionadas

Cuando se seleccione alguna carta esta quedará bloqueada, de manera que no se podrá volver a seleccionar. Si se desea cambiar de cartas se puede volver a seleccionar la acción borrando así las cartas seleccionadas.



Imagen 6.4.4.c – Ejemplo de mano del jugador bloequeada

Cuando el jugador se encuentre con una acción de decisión en la que tenga que elegir cartas del adversario, las cartas del jugador quedarán bloqueadas al igual que las acciones.

### Acción elegida



Imagen 6.4.5.a – Ejemplo de acción elegida

Esta fila aparecerá en un principio vacía, pero irá cambiando, dependiendo de la acción y las cartas que seleccione el jugador. No es interactiva, ya que el jugador no puede pulsar en ninguna casilla. En caso de que el jugador tenga que realizar una acción de decisión esta aparecerá en esta fila.

#### Acción por realizar

En la primera casilla se muestra la acción que ha seleccionado el jugador (ver sección 4.1.3). A continuación, se muestran las cartas que ha seleccionado el jugador. Es importante destacar que en para la acción de competición el orden en el que se seleccionen las cartas tiene relevancia, ya que las dos primeras formarán un grupo y las dos segundas el otro grupo. Para el resto de las acciones el orden es indiferente.

#### Acción de decisión



Imagen 6.4.5.2.a – Ejemplo de acción de decisión

Cuando el jugador tenga que realizar una acción de decisión en el lugar de la acción aparecerá una breve descripción de lo que debe hacer. En caso de ser una acción de regalo tendrá que seleccionar una carta de las 3, la cual aparecerá bloqueada después de ser seleccionada, pudiendo seleccionar cualquiera de las otras dos si el jugador cambia de opinión.

En caso de ser una acción de competición el jugador podrá elegir entre el grupo de las dos primeras o el grupo de las dos últimas. Al pulsar sobre una carta se quedan marcadas las dos cartas de ese grupo, pudiendo cambiar de grupo en caso de cambio de opinión.

### Botón de aceptar

En último lugar tenemos el botón de aceptar, este botón aparece únicamente cuando el jugador ha seleccionado una acción y el correspondiente grupo de cargas o ha seleccionado las cartas que desea en la acción de decisión. En caso de que el jugador cambie de acción y las cartas seleccionadas se reinicien el botón desaparecerá hasta que el jugador vuelva a seleccionar el numero correcto de cartas.

Al pulsar este botón la acción queda registrada y se envía a la aplicación, para que el adversario realice su acción y le vuelva a tocar el turno al jugador.

### Pantalla de información final

Cuando se acaba la partida se cierra la pantalla principal y aparece una pequeña ventana anunciando el resultado de la partida.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Imagen 6.4.7.a – Ventana de información de resultado