## Implementación del Algoritmo en lenguaje R

## Introducción y estructura del algoritmo en lenguaje R

En este apartado se va a tratar sobre cómo se ha implementado el Algoritmo diseñado en lenguaje R.

Tal y como se ha explicado en el apartado 3.4.1, en función de las rondas, el cálculo de los valores de probabilidad varía, por lo que es necesario crear diversas funciones que se almacenarán en distintos scripts de R. Es necesario tener en mente cómo se van a enfocar las funciones de cara al enlace, pues la llamada de las funciones desde el enlace dependerá de cómo se creen las funciones. De todas formas, las llamadas a las funciones desde el enlace y su estructura se explicará en el apartado 3.6.

Lo primero es la creación de los marcos de datos, o dataframes, que representen ambas tablas (la tabla de pesos y la tabla de triple probabilidad de acción), siendo los archivos data.csv y triple.csv.

Pasamos al planteamiento de las funciones necesarias. Teniendo en cuenta el diseño del algoritmo, se puede hacer un esquema de las funciones necesarias para cada fase, así como los datos necesarios:

* **Preflop**

Para el cálculo de la toma de decisiones del algoritmo durante el preflop, necesitamos únicamente la mano; por tanto tendremos como datos iniciales los números y los palos de las cartas de la mano.

Lo primero es necesario una función para convertir estos datos en estructura Mano. Después es necesario descartas esas cartas de Mano del Mazo (ya que no existen en el mazo), así como descartar las correspondientes combinaciones de esas cartas en las tablas.

Por último, hace falta una función que calcule cómo sería el valor de la tabla de triple probabilidad en función de la fórmula de Chen (por tanto, sería necesario una función para calcular la fórmula de Chen y una función para generar todas las combinaciones de cartas restantes). Además, es necesario también una función que actualice el valor de la tabla de pesos en caso de que haya una acción previa.

* **Flop**

En esta ronda, necesitaremos como datos las 3 cartas de la mesa, así como la apuesta actual de cada jugador. Por lo tanto, tendremos como datos iniciales los números y los palos de dichas cartas y el valor de ambas apuestas.

Es necesario una función para convertir los datos de las cartas en una estructura Mesa. Se descartarían las cartas y los valores de las tablas usando una forma similar al del preflop.

En este punto, son necesarias las funciones para calcular tanto la Fuerza de Mano, el Potencial y los Odds, con las fórmulas indicadas previamente, así como una función para actualizar la tabla de triple probabilidad con estos datos.

* **Turn**

En esta ronda, necesitaremos como datos las 4 cartas de la mesa, así como la apuesta actual de cada jugador. Por lo que tendremos como datos iniciales el número y el palo de la cuarta carta (ya que las otras tres ya las tenemos de la ronda anterior), así como las apuestas.

Las funciones que necesitamos en este punto son las funciones equivalentes a las de la ronda anterior.

* **River**

En esta última ronda, necesitaremos los datos de las 5 cartas de la mesa. Por tanto, tendremos como datos iniciales el número y el palo de la quinta carta.

En este caso, necesitaremos una función que calcule la fuerza de Mano, así como las funciones de descarte de pesos y actualización de pesos.

De esta manera, hemos podido enumerar varias funciones que nos son necesarias para el funcionamiento del algoritmo.

Hay que tener en cuenta que en R no existen clases, por tanto es necesario utilizar otra forma para definir elementos como el Mazo, las cartas en la mesa o en la mano.

Para ello, utilizaremos matrices para definirlos, siguiendo la estructura de [Número Palo].

Comenzamos con la explicación de los scripts que conforman las funciones del algoritmo.

## Funciones auxiliares

En este apartado voy a englobar las funciones que, sin ser las funciones descritas en el apartado anterior, en el apartado 3.4.1 o en el 2.3, son necesarias para el funcionamiento correcto del algoritmo.

### inicializa()

Esta función sirve para cargar en memoria todas las demás funciones, para que el programa pueda funcionar solo llamando a esta función.

Para llamar a las otras funciones, se utiliza el comando *source(“NombreScript.R”)* por cada uno de los scripts de funciones.

Además, iguala el valor del dataframe prob al de prob\_ini, de manera que inicializa esa dataframe.

### crearMazo()

Esta función sirve para generar un elemento Mazo equivalente a un elemento de la clase Mazo del motor de juego. Es decir, genera una matriz de 52 filas y 2 columnas con el comando *matrix* cuyo nombre de columna es “Num” y “Palo” respectivamente.

La salida de esta función es la propia matriz Mazo.

### cargaPesos() y cargaTriple()

Estas dos funciones sirven para leer los datos de los dataframes correspondientes con la función *read.csv* y teniendo como salida de dicha función la matriz correspondiente a ese dataframe.

En el caso de *cargaPesos()* la función obtiene la información de data.csv y devuelve la matriz de pesos, mientras que *cargaTriple()* lee el dataframe triple.csv y devuelve la matriz de tripre probabilidad.

### NumtoChar(n1,n2) y PalotoChar(P1,P2)

Estas funciones auxiliares sirven para convertir en texto el número y el palo de las cartas para poder manejarlo en la tabla de Pesos ya que, como se mencionó en el apartado 3.4.1, las filas de esa tabla son N1N2 y las columnas de la tabla son P1P2, siendo la salida de las funciones N1N2 o P1P2 respectivamente.

Primero se determina el número total a convertir en texto.

En el caso de *NumtoChar*, ya que los números varían del 2 al 14 (J, Q, K y A equivalen a 11, 12, 13 y 14 respectivamente), al número mayor de los dos se le multiplica por 100 y se le suma el otro (ya que es posible que alguno de los números tenga decenas. En el caso de *PalotoChar*, a P1 se le multiplica por 10 y se le suma P2 (siendo 1 para tréboles, 2 para picas, 3 para diamantes y 4 para corazones).

Posteriormente, a la cifra se le utiliza la función *as.character(number)*, y la salida de esa función es la que se utiliza como salida.

Poniendo un ejemplo de cómo se vería la salida de cada una de las salidas: suponiendo el caso de tener como cartas de mano el 2 de corazones [2,4] y el 5 de picas [5,2]:

La cifra en el caso de *NumtoChar(2,5)* sería 5\*100+2=502 y la cifra de *PalotoChar(2,4)* sería 2\*10+4=24.

### leerPesos(mano, pesos) y leerTriple(mano, triple)

Estas dos funciones devuelven el valor pertinente a esa mano dentro de las tablas correspondientes (tabla de pesos en el caso de *leerPesos* y tabla de triple probabilidad en el caso de *leerTriple*).

En el caso de *leerPesos*, primero comprueba cuál de las dos cartas de la mano tiene el número mayor y llama a las funciones *NumtoChar(mano[x,1],mano[y,1])* y *PalotoChar(mano[x,2],mano[y,2]),* siendo x el número de inidice de la carta con número mayor (1 ó 2) e y el otro de los índices. Las salidas de ambas funciones se guardan en las variables Num y Palo, respectivamente , y la salida de la función *leerPesos* es el valor de pesos[Num,Palo].

Para la función de *leerTriple*, funciona de manera similar, sin necesidad de tener que recurrir a las funciones *NumtoChar* ni *PalotoChar*, ya que las primeras 4 columnas de la tabla Triple equivalen a n1, p1,n2 y p2. Las otras tres columnas representan la probabilidad de Pasar, de Ver o de Subir.

En este caso, la carta 1 es la que tiene el número mayor de las dos cartas, y la carta 2 es la que tiene el menor número de ambas. La función buscará la fila que coincida que sus primeras cuatro columnas sea n1, p1,n2 y p2 y almacenará las tres últimas columnas de esa fila en un vector numérico, que será la salida de la función.

En el caso de que, en cualquiera de las dos funciones, se tenga una pareja, para determinar cuál de las dos cartas ocupa la primera posición se compara el número del palo. La que menor palo tenga será la que ocupe la primera posición (para seguir la misma estructura que la tabla de pesos mostrada en el apartado 3.4.1).

### modificaPesos(mano, pesos, valor) y modificaTriple(mano, triple, valor)

Estas funciones se utilizan para asignar el valor de entrada a la entrada correspondiente de la tabla de pesos o de triple probabilidad respectivamente.

La búsqueda en las tablas es idéntica a las funciones *leerPesos* y *modificaTriple*, pero en vez de leer el valor de la tabla, lo sobreescribe con el valor de entrada. La salida de las funciones es la tabla modificada con el nuevo valor.

### ordenaCartas(mano, mesa)

El principal uso de esta función es unir ambas matrices y ordenar todas las cartas incluidas en mano y mesa y devolverlas ordenadas en una única matriz.

Primero realiza la unión usando *rbind*, y después recorre la matriz usando dos bucles *for* para comparar cada elemento con los posteriores. Si el número de un elemento j es mayor que el número del elemento i, intercambia las posiciones de ambos elementos (siendo i>j siempre).

### combinaMazo(mazo)

Esta función sirve para hacer las combinaciones (n 2) posibles de las cartas restantes del mazo, siendo n el número de cartas en el mazo.

Para ello, se va creando una matriz de 4 columnas (n1, p1, n2, p2). Para definir las filas que se añaden, se utilizan dos bucles *for* que recorren la matriz mazo usando los índices i y j respectivamente. Estos bucles comparan la carta i con la carta j y, si son distintas, añade la combinación con el comando *rbind(matriz, c(mazo[i,],mazo[j,])).*

La salida de esta función es la matriz de combinaciones de dimensiones 4xn.

### Paquete “Holdem” y calcularJugada(jugada)

En el repositorio CRAN, distribuidores de R, existe un paquete que simula el funcionamiento del Texas Hold’em, que es el paquete Holdem[[1]](#footnote-1). Ese paquete incluye todas las funciones necesarias para hacer la simulación completa en R. Pero, dado que no todas las funciones terminan de amoldarse al funcionamiento del algoritmo y solo solo se va a implementar en R el algoritmo de juego, no el motor de juego, solamente se van a utilizar un determinado conjunto de funciones: las funciones que se encargan de comprobar qué tipo de jugada se tienen con las cartas de mano y mesa, que se van a utilizar en la función *calcularJugada*. Las funciones utilizadas de este paquete son las siguientes:

* **Strflsh1(Num,Palo):** que comprueba si se tiene escalera de color.
* **Four1(Num):** que comprueba si se tiene póker.
* **Full1(Num):** que comprueba si se tiene full house.
* **Flush1(Num, Palo):** que comprueba si se tiene color.
* **Straight1(Num):** que comprueba si se tiene escalera.
* **Trip1(Num):** que comprueba si se tiene trio.
* **Twopair1(Num):** que comprueba si se tiene dobles parejas.
* **Onepair1(Num):** que comprueba si se tiene una pareja.

Todas estas funciones tienen en común la salida: devuelven el valor de la carta más alta de la jugada en caso de que haya jugada o 0 si no se tiene la jugada.

La función *calcularJugada* se basa en la función *handeval* del paquete Holdem, que tiene esa misma funcionalidad dentro del paquete, pero dando como resultado valores distintos de los que se usan en este proyecto.

De esta manera, la función *calcularJugada* recibe una matriz de cartas ordenadas por la función *OrdenaCartas* e irá llamando a las funciones correspondientes para ir comprobando las jugadas.

La salida de las funciones de jugadas se almacenará en la variable a1 y se comprobará si esa variable es mayor que 0 (que significará que se tiene esa jugada). La comprobación irá en el orden en el que están listadas y devolverá el valor entero correspondiente a la jugada (de la tabla del apartado 3.2).

En el caso de que strflsh1 devuelva un número mayor que 0, se comprobará el valor de a1, que si es 14, significará que la jugada es Escalera Real en vez de Escalera de color, siendo la salida de la función 9. En el caso de que todas las funciones de jugada devuelvan 0, la salida de la función *calcularJugada* será 0.

### valorCartaAlta(num)

Esta función sirve para calcular el valor de la carta alta dentro de la fórmula de Chen, devolviendo el valor correspondiente al número de la carta (como se mostraba en el aparatado 2.3).

Los valores de salida, en función de Num son los siguientes:

* + **num=14:** *return(10)*
  + **num=13:** *return(8)*
  + **num=12:** *return(7)*
  + **num=11:** *return(6)*
  + **else:** *return(num/2)*

## Funciones Principales

Este apartado engloba el resto de las funciones del algoritmo.

### fijarMano(n1,p1,n2,p2), fijarMesaFlop(n1,p1,n2,p2,n3,p3) y fijarMesaPostflop(mesa,n1,p1)

Estas tres funciones se utilizan para convertir la entrada de nuevas cartas de número ni y palo pi en la correspondiente matriz mano o mesa.

En el caso de *fijarMesaPostflop,* añade la carta de número n1 y palo p1 a la matriz mesa usada como entrada de la función.

En los tres casos, se consigue usando *rbind* y *c(ni,pi)*.

La salida de las tres funciones es la correspondiente matriz: Mano en *fijarMano* y Mesa tanto en *fijarMesaFlop* como en *fijarMesaPostflop*.

### descarteCartas(cartas,Mazo)

Esta función sirve para descartar las cartas del mazo.

Para ello, primero cuenta los elementos de la matriz cartas y la matriz mazo usando *nrow*, para poder iniciar un doble bucle *for* parabuscar los elementos de la matriz cartas dentro de la matriz Mazo. Si coincide el elemento de la matriz Mazos con cualquiera de los elementos de la matriz cartas, lo sustituye por un valor NA.

Posteriormente, elimina todas las filas de la matriz Mazo que tengan un valor NA con el comando *na.omit*. Esta matriz Mazo es la que se utiliza como salida de la matriz.

,

### Funciones para descartar valores de las tablas

En este apartado voy a tratar las 6 funciones que se utilizan para descartar valores de las tablas. Las funciones que descartan de la tabla de pesos son *descartePesosCarta(carta, mazo, datos), descartePesosFlop(mesa, mazo, datos) y descartePesosMano(mano, mazo, datos)*; mientras que las funciones que descartan de la tabla de triple probabilidad son *descarteTripleCarta(carta, mazo,datos), descarteTripleFlop(mesa,mazo,datos) y descarteTripleMano(mano,mazo,datos).*

El funcionamiento de las funciones es prácticamente idéntico entre los de una tabla y otra: las funciones *descartePesosCarta* y *descarteTripleCarta* son las encargadas de igual a 0 el valor o valores de la correspondiente tabla usando las funciones *modificaPesos* y *modificaTriple,* respectivamente, siendo el valor de entrada a estas funciones modifica 0 ò un triple cero según corresponda.

Las funciones restantes (*descartePesosFlop*, *descartePesosMano*, *descarteTripleFlop* y *descarteTripleMano*) llaman a la función *descartePesosCarta* o *descarteTripleCarta*, según corresponda, tantas veces como corresponda: 3 veces en las de Flop y 2 veces en las de Mano.

### chenFormula(mano)

Esta función sirve para calcular el valor de la fórmula de chen para la mano que se utiliza como entrada para la función.

Siguiendo las pautas explicadas en el apartado 2.3, primero determina cual de las dos cartas de la mano es la carta más alta, la cual se introduce como valor para llamar a la función *valorCartaAlta*, cuya salida de almacena en una variable a.

Tras eso, se comprueba si el número de ambas cartas es igual (habiendo una pareja), lo cual hace que la variable a tome el valor a=a\*2.

Posteriormente, compara el valor del palo de la carta para comprobar si son del mismo palo, en caso afirmativo, a=a+2.

Lo siguiente que calcula es la distancia entre ambas cartas, almacenando la diferencia entre los números de ambas cartas en la variable gap. En función del valor gap, resta 1, 2, 4 o 5 a la variable a.

Por último, suma 1 al valor de a en caso de que la carta alta sea menor que 12 (Q) y el valor de gap sea <2.

La salida de la función es a.

### actualizaPesos(accion,pesos,triple)

Esta es la función que se utiliza para modificar la tabla de pesos tras una acción. Tal y como se explicó en el apartado 3.4.1, cuando se produzca una acción, se buscará el valor de esa acción en cada combinación de cartas de la tabla de triple probabilidad y el nuevo valor de la tabla de pesos es el anterior valor de la tabla de pesos por esos valores.

Para conseguir esto, se crea un bucle *for* que recorre todas las filas de la matriz triple. Durante ese bucle, usando la función *fijarMano* con los 4 primeros valores de esa fila, se crea una matriz mano que se utilizará para obtener los valores de las dos tablas usando *leerPesos* y *leerTriple*. Estos valores se almacenan en las variables valor\_pesos y triple\_prob

Una vez obtenido los valores correspondientes a esa mano, se crea la variable *valor\_nuevo*, que será la multiplicación de *valor\_pesos* por el valor de *triple\_prob* correspondiente a la acción.

Por último, se utiliza la función *modificaPesos* con esa mano, los datos pesos y *valor\_nuevo*, sobrescribiendo la tabla pesos con ese valor.

Estas acciones se repiten n veces que es el resultado de *nrow(triple)*, es decir, el número de filas de la matriz triple.

La salida de esta función es la tabla pesos, una vez sobrescrita.

### calcularFuerzaMano(mano, mesa, mazo, Pesos)

Esta función es la codificación de la fórmula homónima que aparece en el apartado 3.4.1, en la sección Flop y Turn.

Lo primero que hace es conseguir el valor de la jugada que tiene el algoritmo en ese momento. Para ello, primero obtiene la matriz jugada usando la función ordenarCartas y con esa matriz llama a la función calcularJugada para obtener dicho valor.

Tras eso, genera las combinaciones de manos restantes con las cartas del mazo usando combinaMazo y se inicia un bucle for con tantas repeticiones como número de combinaciones.

En ese bucle, obtiene la mano usando fijarMano, se crea una matriz jugada\_op llamando a la función ordenarCartas con esa mano y la matriz mesa y se calcula el valor valor\_op usando calcularJugada con la matriz jugada\_op.

Con esa mano se lee el peso correspondiente a esa mano con leerPesos y se suma a la variable de conteo que corresponda según si valor\_op es mayor, menor o igual al valor calculado antes del bucle.

Una vez acabado el bucle se dispone de tres variables: superiores, iguales e inferiores, que corresponden al sumatorio de los pesos donde el valor\_op es mayor, igual o inferior al valor calculado.

Con estas tres variables, obtenemos FMT, que es la suma de las tres, FMSup, que es la división de inferiores entre FMT, FMIg, que es el resultado de iguales/FMT, y FMInf, que es el cociente entre superiores y FMT.

Por último, se crea el vector FM con los tres valores usando c(FMInf,FMig,FMSup), siendo este vector la salida de la función.

### calcularPotencial(mano, Mesa, mazo, pesos)

Esta función es la codificación de la fórmula homónima que aparece en el apartado 3.4.1, en la sección Flop y Turn.

La función crea tanto el vector VPT como la matriz VP 3x3, ambas con todos sus valores siendo 0. Después consigue el valor de la jugada que tiene el algoritmo en ese momento. Para ello, primero obtiene la matriz jugada usando la función ordenarCartas y con esa matriz llama a la función calcularJugada para obtener dicho valor, que se almacena en la variable valorjugador.

Tras eso, genera las combinaciones de manos restantes con las cartas del mazo usando combinaMazo y se inicia un bucle for con tantas repeticiones como número de combinaciones. En ese bucle, obtiene la mano usando fijarMano, se crea una matriz jugada\_op llamando a la función ordenarCartas con esa mano y la matriz mesa y se calcula el valor valor\_op usando calcularJugada con la matriz jugada\_op.

Con esa mano se lee el peso correspondiente a esa mano con leerPesos y se crea una matriz mazoaux que es la salida de la función descarteCartas(mano\_op,mazo). Tras eso, se hace la comparación de valor\_op con valorjugador. En función de si valor\_op es mayor, igual o menor, el peso de esa mano se almacenará en VPT[1], VPT[2] o VPT[3] respectivamente. El índice del vector se almacenará en la variable x.

En este momento, se inicia un segundo bucle for con el tamaño de mazoaux. Dentro de este bucle se crea una matriz mesaaux como concatenación de mesa y el valor j de la matriz mazoaux. En este momento se calculan las matrices jugada\_aux y jugada\_op\_aux con la nueva mesaaux, utilizando ordemarCartas con mano y mano\_op respectivamente, y se calculan los nuevos valores de jugada valorjugador\_aux y valor\_op\_aux con calcuclarJugada usando estas matrices.

Se compara valor\_op\_aux con valorjugador\_aux y, en función de si es mayor, igual o menor, se almacenará en VP[x,1], VP[x,2] y VP[x,3] respectivamente, acabando con el segundo bucle.

Cuando finalice el primer bucle, se calculan PotPositivo (VP[1,2]/VPT[1]) y PotNegativo (VP3,1/VPT3), con los que se forma el vector de salida.

### calculoOddMano(mano, mesa, mazo)

Esta función es la codificación de la fórmula homónima que aparece en el apartado 3.4.1, en la sección Flop y Turn.

Primero define la variable de conteo cartamejora, que se inicializa con valor 0. Tras esos, calcula el valor de la jugada que tiene el algoritmo en ese momento. Para ello, primero obtiene la matriz jugada usando la función ordenarCartas y con esa matriz llama a la función calcularJugada para obtener dicho valor, que se almacena en la variable valorjugador.

Después se inicia un bucle for con tantas repeticiones como número de elementos en la matriz mazo. Con este bucle se busca contabilizar con qué cartas mejora la variable valorjugador.

Para eso, se genera una matriz mesaaux concatenando mesa y el elemento i de la matriz mazo. Posteriormente se calcula la matriz jugada\_aux, que es la salida de la función ordenarCartas, matriz que se utiliza a su vez para calcular el valorjugador\_aux usando calcularJugada.

Después, se hace una comparación entre valorjugador y valorjugador\_aux. Si valorjugador\_aux>valorjugador, se suma 1 a la variable cartamejora.

Una vez finalice el bucle, se calcula la diferencia entre los elementos de mazo y cartamejora. Y, por último, se calcula la variable de salida OddMano como cociente entre cartamejora y diferencia.

### calculoOddsBote(apuesta,ap\_oponente)

Esta función es la codificación de la fórmula homónima que aparece en el apartado 3.4.1, en la sección Flop y Turn.

Primero se calculan la diferencia entre ap\_oponente y apuesta, que se almacena en la variable diferencia, y el bote, que es la suma de apuesta, ap\_oponente y diferencia.

Tras eso, se calcula el valor de salida OddBote, que es el cociente entre diferencia y bote.

### ajusteBayes(triple,accion,ronda, valorJugada)

Esta función es la codificación de la fórmula homónima que aparece en el apartado 3.4.1.

Lo primero que se hace es obtener el valor de probabilidad actual de cada uno de los patrones, leyendo del dataframe prob.csv. Posteriormente, se carga la tabla de valores de Bayes correspondiente a la ronda (tablasprobpreflop.csv en caso de ser en el preflop o tablasprobpostflop.csv en caso contrario). El razonamiento de cómo se han definido estos valores se encuentra en el apartado 3.4.1.4.

Después, se calcula la probabilidad de cada uno de los patrones usando la fórmula del teorema de Bayes.

Con estos cálculos hechos, se puede determinar cual de los tres es más probable usando comparativos if enlazados. Una vez que se determine cuál de los tres patrones es el más probable, se hace el ajuste correspondiente.

Para determinar el ajuste correspondiente, es necesario comprobar el valor de la jugada que se tiene en ese momento, así como la acción y la ronda.

* Si el patrón es Maniaco: su punto débil es enfrentarse a jugadores con manos fuertes, así como a jugadores que no sigan su ritmo de subidas. Por tanto:
  + Si se tiene un valor de jugada alto: el valor de la probabilidad de pasar se multiplicará por 0.6. Ese 40% se sumará a partes iguales a las probabilidades de ver y subir.
  + Si no se tiene un valor de jugada alto: los valores de probabilidades de ver y subir se multiplicarán por 0.8. Ese valor de diferencia de cada uno se sumará a la probabilidad de pasar.
* Si el patrón es Roca: Su punto débil es enfrentarse a jugadores agresivos que le roben dinero en apuestas que no pueda seguir. También hay que considerar que el patrón Roca actuará de forma agresiva si tiene una mano fuerte. Por tanto:
  + Si la acción no ha sido subida: el valor de la probabilidad de pasar se multiplicará por 0.6. Ese 40% se sumará a partes iguales a las probabilidades de ver y subir.
  + Si la acción ha sido subir:
    - Si se tiene un valor de jugada alto: el valor de la probabilidad de pasar se multiplicará por 0.6. Ese 40% se sumará a partes iguales a las probabilidades de ver y subir.
    - Si no se tiene un valor de jugada alto: los valores de probabilidades de ver y subir se multiplicarán por 0.8. Ese valor de diferencia de cada uno se sumará a la probabilidad de pasar.
* Si el patrón es Calling Station: Su punto débil es el mismo que el de Maniaco, aunque al no presentar tanta presión, los valores de jugada son mas permisivos a la hora de discernir que acciones tomar. Pero los ajustes son los mismos que en el caso de Maniaco.

Tras hacer el ajuste, en caso de hacerlo, se sobrescribe el valor del dataframe prob.csv con los nuevos valores de probabilidad de los patrones.

La salida de esta función es el vector que engloba la triple probabilidad de acción del algoritmo.

### calculaProbPreflop(mano, mazo, accion) y calculaProbPreflopIni(mano,mazo)

Estas dos funciones se utilizan para calcular el valor de la triple probabilidad en la fase de preflop, siendo *calculaProbPreflopIni* usada en el caso de que sea la primera vez que el algoritmo actúe mientras que *calculaProbPreflop* es la segunda vez que actúa.

El funcionamiento de ambas funciones es prácticamente idéntico: primero se crea una matriz llamada *valoresPreflop*, que incluye los valores de la tabla correspondiente de la sección preflop del apartado 3.4.1. En el caso de *calculaProbPreflopIni*, siempre será tabla inicial, mientras que en el caso de *calculaProbPreflop* puede ser la tabla inicial o la tabla de resubida.

Tras eso, se aplica la fórmula de chen con la función chenFormula utilizando mano como entrada de dicha función. Con este valor, podemos obtener el valor de probabilidad triple de acción en función de la correspondiente tabla.

Por último, se aplica el ajuste de la fórmula de Bayes utilizando ajusteBayes. La salida de esta última función será la salida de estas funciones.

### calculoProbabilidadAccion(mesa, mazo, triple, ronda)

Esta función es la codificación de la fórmula homónima que aparece en el apartado 3.4.1, en la sección Flop y Turn. Esta función sirve para actualizar la probabilidad triple de acción de cada una de las manos restantes.

Lo primero que se hace es obtener el valor de probabilidad actual de cada uno de los patrones, leyendo del dataframe prob.csv. Posteriormente, se carga la tabla de probabilidades correspondiente a la ronda (tablaspreflop.csv en caso de ser en el preflop o tablaspostflop.csv en caso contrario).

El siguiente paso es ejecutar combinaMazo para obtener las manos posibles con las cartas restantes del mazo. Una vez tenemos esa combinación de cartas, se inicia un bucle for en función del número de combinaciones.

En ese bucle, primero se crea la matriz mano\_op con fijarMano y, se asignan los valores a modificar según las tablas que están en el apartado 3.4.1 y la ronda en la que se encuentre, tras lo cual se ejecuta modificaTriple con esa mano y esos valores para sobrescribir la tabla triple.

Cabe destacar que el valor de Mesa en caso de que ronda sea 1 será 0, ya que no se necesita ese valor.

La salida de esta función es la tabla triple sobrescrita con estos cambios.

### calculoProbFlop(mano, mesa, mazo, pesos, triple, aa, ao)

Esta función se utiliza para calcular la probabilidad de acción durante las rondas de Flop o Turn, utilizando la fórmula que se encuentra en el apartado 3.4.2.2.

Para esto, se llamarán las formulas calcularFuerzaMano, calcularPotencial, calculoOddMano y calculoOddsBote, almacenando sus respectivos valores en variables.

Tras esto, se calcula el valor de la probabilidad de s y p usando la fórmula arriba mencionada. Con estos dos valores, se puede calcular v, que es 1-(s+p). Con estos tres valores, se crea un vector.

Este vector, junto al cálculo del valor de la jugada, se introducen como entrada en la función ajusteBayes, para realizar la pertinente corrección de Bayes.

La salida de esta función es el vector de salida de ajusteBayes.

### calculoProbRiver(mano, mesa, mazo ,pesos)

Esta función tiene como objetivo calcular la probabilidad de acción durante la ronda de River, siendo equivalente a calculoProbFlop. Esta función utiliza la fórmula encontrada en el apartado 3.4.2.3.

Para hacer este cálculo, es necesario llamar a la fórmula calcularFuerzaMano, salida que se almacenará en el vector FM.

Se calcula el valor de la jugada con calcularJugada después de haber utilizado ordenarCartas para obtener un vector jugada. Con este valor, se tienen todos los datos para utilizar la función ajusteBayes, que generará el vector salida de esta función.

1. <https://cran.r-project.org/web/packages/holdem/holdem.pdf> [↑](#footnote-ref-1)