Caminante aleatorio

Un caminante aleatorio (o *random walk*, en inglés) es un tipo de modelo estocástico en el cual la posición de una partícula en cierto instante depende de su posición en el instante previo y alguna variable aleatoria que determina su subsecuente dirección y la longitud de paso.

Este tipo de modelo es muy útil para hacer simulaciones en Física Estadística; por ejemplo, se puede utilizar para modelar el movimiento browniano de moléculas en un gas.

Caminante aleatorio en una dimensión

El modelo más sencillo de un caminante aleatorio es aquel en el que una partícula:

- se mueve en un tiempo discreto y en intervalos de tiempo uniformes;
- se mueve en un espacio discreto de una sola dimensión con tamaño de paso uniforme;
- en cada paso, tiene igual probabilidad de moverse en cualquiera de los dos sentidos posibles.

Los primeros dos puntos son fáciles de modelar: para el primer punto, podemos modelar el tiempo con los números naturales y, para el segundo, podemos modelar el espacio con los números enteros, suponiendo -por simplicidad- que la posición inicial de la partícula es el origen (`0`) y que el tamaño de casa paso es `1`. En otras palabras, para simular el paso del tiempo podemos iterar sobre un arreglo de números naturales consecutivos y, para registar cada posición, podemos crear un arreglo que inicialmente sólo contenga la posición inicial, (`[0]`) y posteriormente añadir las posiciones subsecuentes como entradas a este arreglo.

La cuestión ahora es, ¿cómo modelamos el tercer punto?

La función `rand`

Julia tiene varias funciones para generar números aleatorios (puedes consultar su documentación aquí). Una de ellas es `rand`. Para conocer lo que hace esta función a través de ejemplos, crea celdas individuales para cada uno de los siguientes comandos y ejecuta cada una de ellas varias veces hasta que tengas una idea de qué es lo que hacen:

- `rand()`
- `rand(Int)`
- `rand(Bool)`

```
0.0630130585626234
```

```
rand()
```

```
-3412078598684120374
```

```
rand(Int)
```

```
false
```

```
rand(Bool)
```

Ahora, haz lo mismo para los siguientes comandos, creando las nuevas celdas *debajo* de la celda que define a la variable `n=1` (y arriba del **Ejercicio**) y cambiando el valor de `n` (ejecutando la celda en la que se define) varias veces antes de revisar la documentación de nuevo:

- `rand(n)`
- `rand(Int,n)`
- `rand(Bool,n)`

```
1×5 Matrix{Float64}:
 2.4726  1.99368  1.74213  0.757567  2.50453
```

```
begin
    aleatorioUniforme(0,5,5)
end
```

```
N = 4
```

```
N=4
```

```
[0.44669, 0.471123, 0.74962, 0.422415]
```

```
rand(N)
```

```
[-8405746349308367021, -4352522975623217574, 3877162
```

```
rand(Int,N)
```

```
[true, false, false, true]
```

```
rand(Bool,N)
```

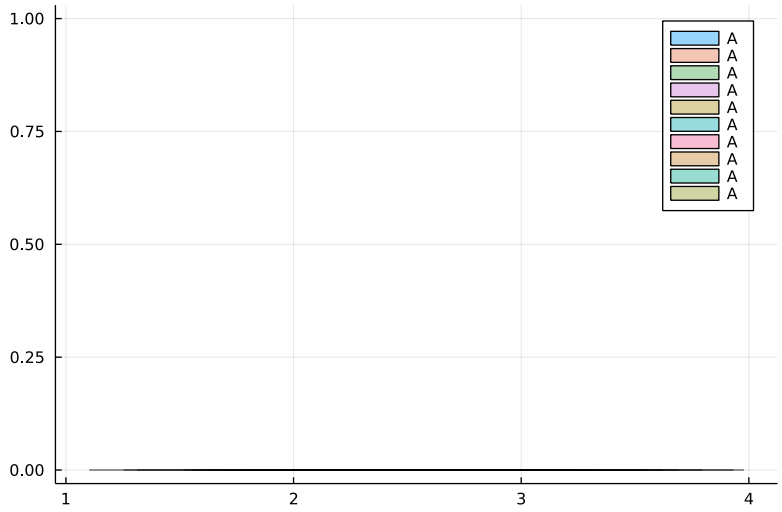
Ejercicio Crea una función `aleatorioUniforme` que tome como argumentos dos números `a` y `b`, y devuelva como salida un número aleatorio distribuido uniformemente en el intervalo $[a, b] \subset \mathbb{R}$. (Sugerencia: Utiliza `rand()` y un poco de aritmética.)

```
using  Distributions   , Plots            #Este
paquete permite utilizar la función Uniform
```

```
aleatorioUniforme (generic function with 1 method)
```

```
function aleatorioUniforme(a,b,n)       #Función
aleatorio uniforme, con variables a, b y n
  rand(Uniform(a, b), 1, n)             #Escoge
un número aleatorio entre a, b, de una fila y n
columnas
end
```

Ejercicio Verifica con un histograma que tu función `aleatorioUniforme` realmente cumpla la propiedad desada. (Pista: ¿Cómo debería verse el histograma?)




```
begin
    histogram([aleatorioUniforme(0,1,10)],
    bins=20, alpha=0.4, label="A")
end
```

Modelando una caminata aleatoria

Como estamos modelando el espacio con los números enteros, cada paso sucesivo de nuestro caminante aleatorio debería sumar o restar `1` a la posición anterior con igual probabilidad.

Ejercicio Crea una función `unPaso` que no tome argumentos de entrada y devuelva sólamente los valores `1` y `-1` con igual probabilidad. (Pista: La forma más sencilla de hacerlo es usando `rand(Bool)` y un poco de aritmética...)

```
unPaso (generic function with 1 method)
```

```
function unPaso()
    -(-1)^rand(Bool)
end
```

```
-1
```

```
unPaso()
```

Ejercicio Crea una función `variosPasos` que tome como argumento un número entero `n` y devuelva un arreglo con `n` entradas, donde cada una de ellas tiene la misma probabilidad de ser `1` ó `-1`. (Pista: Recuerda que colocar un punto (`.`) antes de un operador aritmético hace que funcione con arreglos.)

```
variosPasos (generic function with 1 method)
```

```
function variosPasos(M)
    rand((-1, 1), 1, M)    #Para agilizar, se
    dan los posibles números (-1 y 1), en un
    arreglo de 1 fila y M columnas
end
```

```
1×8 Matrix{Int64}:
 -1  -1  1  1  -1  -1  -1  1
```

```
variosPasos(8)
```

Ejercicio Crea una función `caminataAleatoria` que

- tome como entrada un número de pasos `n` y
- devuelva como salida un arreglo con `n+1` posiciones, incluyendo la posición inicial `0`, siguiendo una caminata aleatoria. (Sugerencia: Puedes usar cualquiera de las funciones `unPaso` o `variosPasos`; la que te resulte más cómoda.)

```
caminataAleatoria (generic function with 1 method)
```

```
function caminataAleatoria(f)
#Definimos la función caminataAleatoria con f la
cantidad de pasos
    c=zeros(f+1, 1)
    #Hacemos un array ceros, con una fila y f+1
    columnas
    for i in 2:f+1                        #Para
    toda i desde dos hasta f+1; comenzamos desde
    2 porque la entrada de m[1,1] debe ser la
    posición inical
        c[i, 1]=c[i-1,1]+unPaso()      #Función
        recursiva, la entrada [1,i] se va
        definir a partir de la entrada anterior,
                                        #es
        decir [1,i-1] más la función un paso
    end
    return c
    #Devuelve la matriz c
end
```

Ejercicio Crea una función `graficaCaminata` que

- tome como entrada un arreglo -que supondremos que simula una caminata aleatoria en una dimensión- y
- devuelva como salida una gráfica de posición contra número de pasos, con etiquetas en los ejes.

(Sugerencia: Crea un bloque de código con `begin` y `end` para poder hacer todo en una sola celda de Pluto.)

Luego, en una celda aparte, define una variable `x` como un número entero positivo y aplica tu función `graficaCaminata` a `caminataAleatoria(x)` para generar gráficas de caminatas aleatorias.

```
graficaCaminata (generic function with 1 method)
```

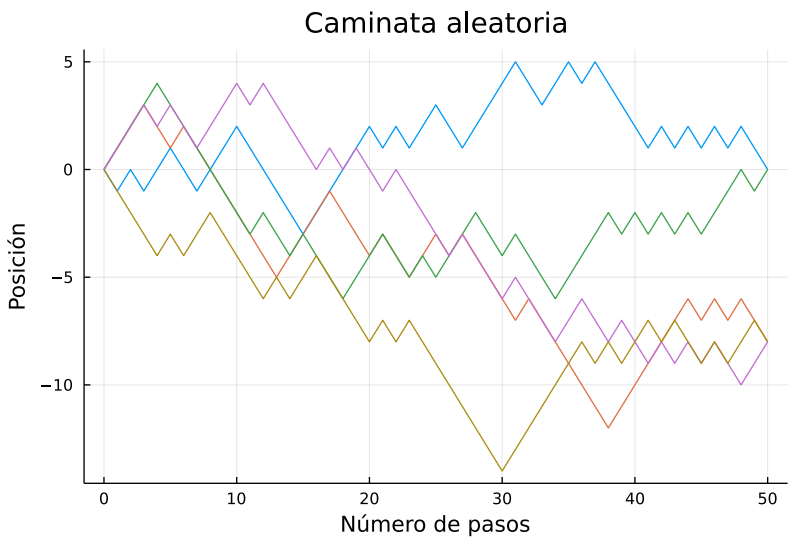
```
function graficaCaminata(G)     #FUNCION
GRAFICACAMINATA
    Y=caminataAleatoria(G)      #Posición
    respecto a Y
    return plot(0:G,Y,title = "Caminata
    aleatoria", xlabel = "Número de pasos",
    ylabel = "Posición", color = "lightpink2",
    label =false)
    #Devuelve la gráfica comenzando en cero de
    uno en uno hasta G(cantidad de pasos), y del
    eje Y devuelve el arreglo de
    camianataaleatoria
end
```


```
graficaCaminata(100)
```

Ejercicio Crea una función `graficaCaminata!` que sea una versión modificadora de la función anterior y genera una gráfica con 5 caminatas aleatorias.

```
graficaCaminata! (generic function with 1 method)
```

```
function graficaCaminata!(H,I) #H son la
cantidad de caminatas aleatorias e I los pasos
    plot()              #Grafica
    for i in 1:H        #Para i de 1 hasta H
        plot!(0:I,caminataAleatoria(I),title =
        "Caminata aleatoria", xlabel = "Número
        de pasos", ylabel = "Posición", label
        =false, lef=false) #En esencia, lo mismo
        que arriba
    end
    plot!()             #Muestra la gráfica
end
```




```
graficaCaminata!(5,50)
```

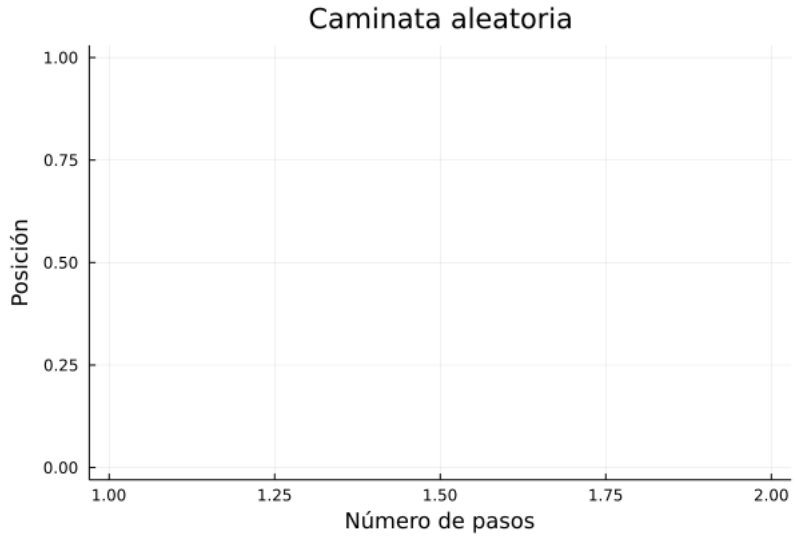
Ejercicio Crea una función `animaCaminata` que

- tome como entrada un arreglo -que supondremos que simula una caminata aleatoria en una dimensión- y
- devuelva como salida una *animación* de la caminata. (Sugerencia: Usa tu función `graficaCaminata` .)

```
animCaminata1D (generic function with 1 method)
```

```
function animCaminata1D(pasito)
anim = @animate for i = 1:pasito #Anima de 1
hasta pasito
x1d = collect(1:1:pasito)   #Arreglo del eje x
y1d=caminataAleatoria(10)#Arreglo dej eje y
    plot(x1d[1:i], y1d[1:i],title = "Caminata
aleatoria", xlabel = "Número de pasos", ylabel =
"Posición", color = "lightpink2", label =false)
end #Grafica x1d,y1d desde 1 hasta pasito

gif(anim, "animCaminata1D.gif", fps = 1)
end
```




```
animCaminata1D(10)
```

```
Saved animation to
fn:  "C:\\Users\\ACE\\Documents\\GitHub\\Tareas\\animC
```

Caminante aleatorio en dos dimensiones

Generalicemos nuestro modelo de caminante aleatorio suponiendo que ahora nuestra partícula se mueve en un espacio *continuo* de *dos* dimensiones espaciales; con tamaño de paso *continuo* y *variable*. Es decir que, a pesar de que seguiremos modelando con un tiempo discreto e intervalos de tiempo uniformes, ahora el espacio será *continuo* y tendrá *dos* dimensiones espaciales.

Ejercicio Crea una función `caminataAleatoria2D` que

- tome como entrada un número de pasos `n` y
- devuelva como salida un arreglo con dos subarreglos que tengan `n+1` "posiciones" cada uno -uno de posiciones horizontales y otro de posiciones verticales, incluyendo las posiciones iniciales `0` en cada caso-, simulando una caminata aleatoria.

Utiliza tu función `aleatorioUniforme` para generar números aleatorios en el intervalo $[-1, 1]$ y suma números generados por esta función a las posiciones horizontales y verticales para simular un paso continuo en dos dimensiones.

```
caminataAleatoria2D (generic function with 1 method)
```

```
function caminataAleatoria2D(R)
          #Definimos la función
caminataAleatoria con f la cantidad de pasos
    caminata2D=[caminataAleatoria(R)
    caminataAleatoria(R)] #Usando la función
    caminataAleatoria, se forma una matriz de
    Rx2 (R cantidad de pasos)
end
```

```
5×2 Matrix{Float64}:
  0.0   0.0
  1.0  -1.0
  0.0  -2.0
 -1.0  -3.0
  0.0  -2.0
```

```
caminataAleatoria2D(4)
```

Ejercicio Crea una función `graficaCaminata2D` que

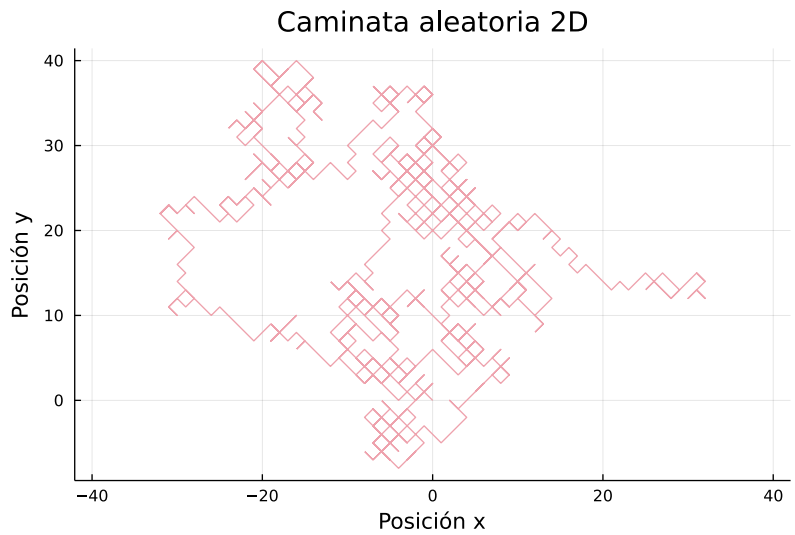
- tome como entrada un arreglo con dos subarreglos - que, supondremos, simulan una caminata aleatoria en dos dimensiones- y
- devuelva como salida una gráfica bidimensional que muestre la trayectoria de la caminata.

(Sugerencia: Crea un bloque de código con `begin` y `end` para poder hacer todo en una sola celda de Pluto.)

Luego, en una celda aparte, define una variable `x` como un número entero positivo y aplica tu función `graficaCaminata2D` a `caminataAleatoria2D(x)` para generar gráficas de caminatas aleatorias en dos dimensiones.

```
graficaCaminata2D (generic function with 1 method)
```

```
function graficaCaminata2D(Q) #Otra manera de hacerlo c;
    X=caminataAleatoria(Q)     #Posición en x
    Y=caminataAleatoria(Q)     #Posición en y
    return plot(X,Y,title = "Caminata aleatoria 2D", xlabel = "Posición x", ylabel = "Posición y", color = "lightpink2", label =false, aspect_ratio=:equal)       #Devuelve la gráfica de X,Y
end
```

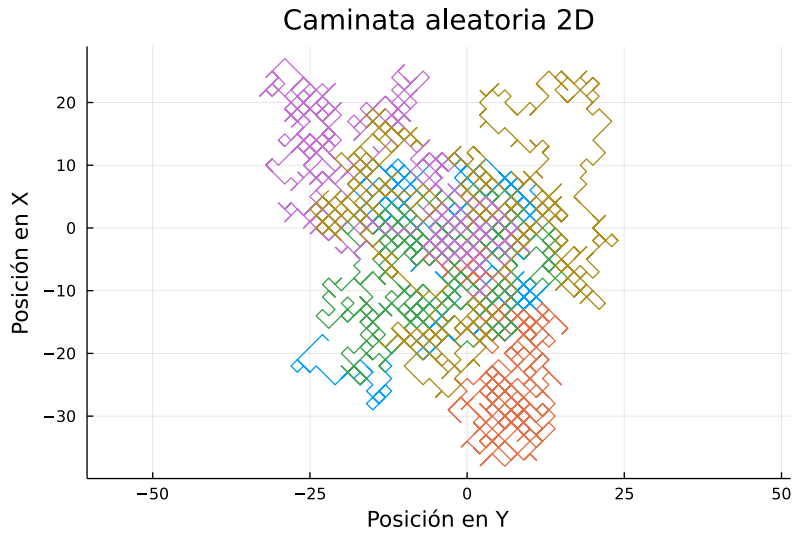



```
graficaCaminata2D(1000)
```

Ejercicio Crea una función `graficaCaminata2D!` que sea una versión modificadora de la función anterior y genera una gráfica con 5 caminatas aleatorias.

```
graficaCaminata2D! (generic function with 1 method)
```

```
function graficaCaminata2D!(S,T) #S son la
cantidad de caminatas aleatorias y T los pasos
    plot()              #Grafica
    for i in 1:S        #Para i de 1 hasta H
        plot!
        (caminataAleatoria(T),caminataAleatoria(T
        ),title = "Caminata aleatoria 2D", xlabel
         = "Posición en Y", ylabel = "Posición
        en X", label =false, lef=false,
        aspect_ratio=:equal) #Grafica en X una
        caminata y en Y otra
    end
    plot!()             #Muestra la gráfica
end
```




```
graficaCaminata2D!(5,1000)
```

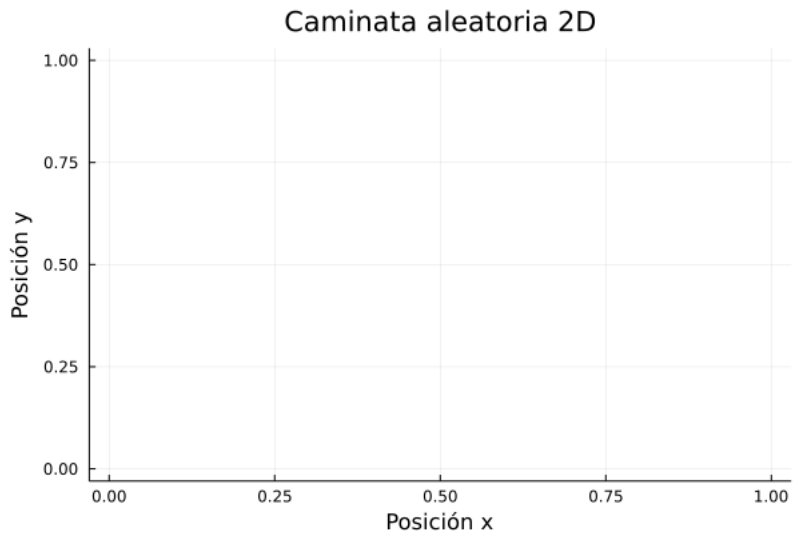
Ejercicio Crea una función `animaCaminata2D` que

- tome como entrada un arreglo con dos subarreglos - que, supondremos, simulan una caminata aleatoria en dos dimensiones- y
- devuelva como salida una *animación* de la caminata. (Sugerencia: Usa tu función `graficaCaminata2D` .)

```
animaCaminata2D (generic function with 1 method)
```

```
function animaCaminata2D(pasos) #Funcion animacaminata2D
anim = @animate for i = 1:pasos #Desde 1 hasta el último paso
x2d=caminataAleatoria(pasos)    #comienza con la gráfica de x1d
y2d=caminataAleatoria(pasos)    #comienza con la gráfica de y1d
    plot(x2d[1:i], y2d[1:i],title = "Caminata aleatoria 2D", xlabel = "Posición x", ylabel = "Posición y", color = "lightpink2", label =false)
end

gif(anim, "animaCaminata2D.gif", fps = 1)
end
```



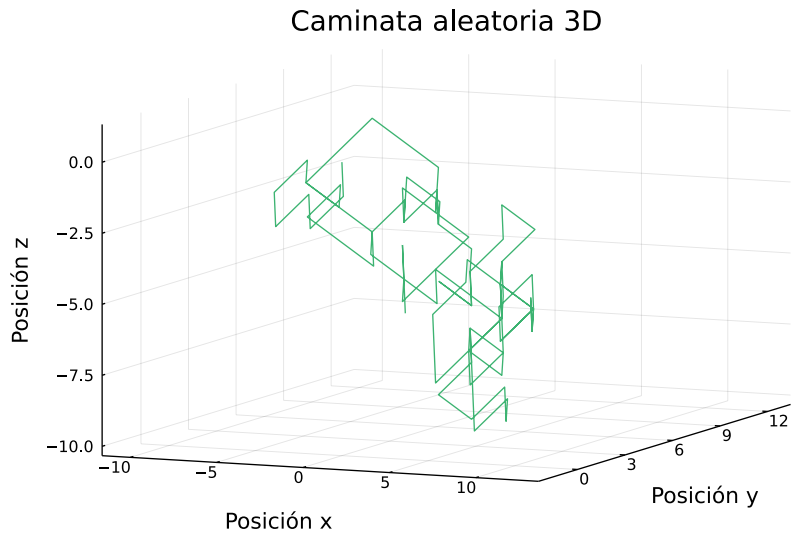

```
animaCaminata2D(10)
```

```
Saved animation to
fn:  "C:\\Users\\ACE\\Documents\\GitHub\\Tareas\\anima
```

Ejercicio ¡Haz una caminata aleatoria en tres dimensiones espaciales y grafícala! (Así se modelan, por ejemplo, las partículas en un gas).

```
graficaCaminata3D (generic function with 1 method)
```

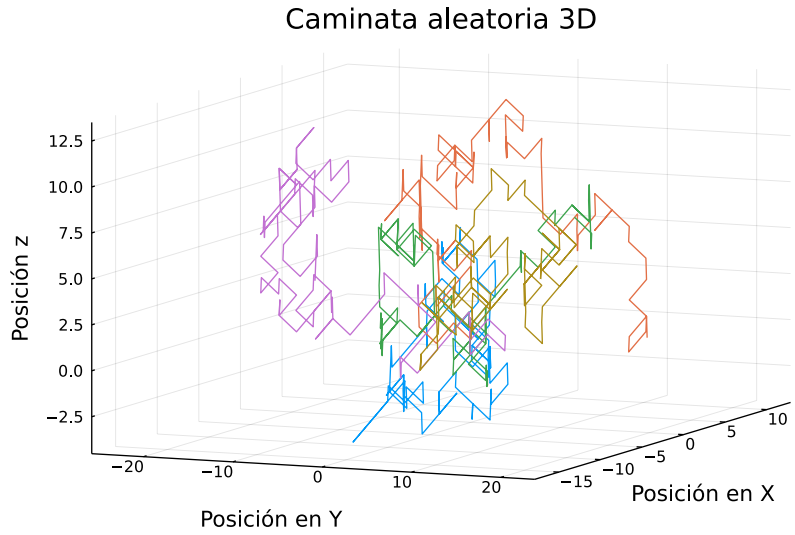
```
function graficaCaminata3D(U) #Función caminata
3D
    p=caminataAleatoria(U)     #Posición en x
    q=caminataAleatoria(U)     #Posición en y
    r=caminataAleatoria(U)     #Posición en z
    return plot(p,q,r,title = "Caminata
    aleatoria 3D", xlabel = "Posición x", ylabel
    = "Posición y", zlabel = "Posición z", color
    = "mediumseagreen
", label =false, aspect_ratio=:equal)
#Devuelve la gráfica de p,q,r
end
```




```
graficaCaminata3D(100)
```

```
graficaCaminata3D! (generic function with 1 method)
```

```
function graficaCaminata3D!(v,w) #v son la
cantidad de caminatas aleatorias y w los pasos
    plot()              #Grafica
    for i in 1:v        #Para i de 1 hasta H
        plot!
        (caminataAleatoria(w),caminataAleatoria(w
        ),caminataAleatoria(w),title = "Caminata
        aleatoria 3D", xlabel = "Posición en Y",
        ylabel = "Posición en X", zlabel =
        "Posición z", label =false, lef=false,
        aspect_ratio=:equal) #Grafica en X una
        caminata, Y otra, Z
    end
    plot!()             #Muestra la gráfica
end
```




```
graficaCaminata3D!(5,100)
```