Pensamiento Computacional CBC

.UBA EXACTAS

Trabajo Práctico Final

La ruina del jugador

Nombre del grupo: Mandrolfia

Tema: Blackjack

Integrantes del grupo:

- Nahuel Nimeth, DNI: 45820278, Mail: nahuel23nimeth@gmail.com
- Mariangel Montilla, DNI: 96104330, Mail: mariangelmontillamendez@gmail.com

Curso:

- Turno: (Tarde)
- Comisión: (29001/29002/29003/29004)

2do cuatrimestre 2024

ÍNDICE

0. Descripción del problema

En esta sección deberán describir en forma general el blackjack, de forma que lo pueda entender alguien que no lo conoce. Además, se deberán incluir detalles en caso de que se tomen en consideración algunas variantes distintas a cómo se conoce el problema en forma general.

El Blackjack es un juego de apuestas cuyo objetivo es obtener una puntuación lo más cercana posible a 21, sin pasarse. Los jugadores se enfrentan individualmente al casino, representado por el crupier. Se utiliza una o más barajas estándar de 52 cartas.

Valor de las cartas:

Las cartas numeradas del 2 al 10 valen su valor nominal.
Las figuras (J, Q, K) valen 10 puntos cada una.
El as puede valer 1 u 11 puntos, dependiendo de lo que más convenga al jugador en cada momento.
Desarrollo del juego:

Al inicio de cada ronda, cada jugador y el crupier recibe dos cartas. En el blackjack americano, el crupier muestra una de sus cartas inmediatamente. Si esta carta es un as o una figura, revela la segunda carta para verificar si tiene blackjack (21 puntos con dos cartas). En el blackjack europeo, la segunda carta del crupier solo se muestra al final de la ronda.

Decisiones del jugador:

Una vez repartidas las cartas iniciales, el jugador tiene varias opciones:

Pedir carta: Solicitar una carta adicional para intentar acercarse a 21.
Plantarse: Quedarse con las cartas que tiene y no pedir más.
Doblar: Duplicar su apuesta inicial y recibir una sola carta adicional.

Dividir: Si las dos cartas iniciales tienen el mismo valor, el jugador puede separarlas en dos manos distintas, realizando una apuesta adicional por la segunda mano.
El crupier: El crupier sigue reglas fijas: debe pedir cartas hasta alcanzar un total de 17 o más. Si se pasa de 21, pierde automáticamente.

Apuesta: Cada jugador realiza una apuesta antes de que se repartan las cartas. El monto de la apuesta debe estar dentro de los límites establecidos por el casino.

Ganador: Gana el jugador que obtenga una puntuación más cercana a 21 sin pasarse, o todos los jugadores que empaten con el crupier y tengan una puntuación menor a 21.

1. Análisis inicial

En esta sección deberán completar las preguntas que ya respondieron en la primera clase, teniendo en consideración lo que hicieron después en las clases de trabajo en computadora, de forma de que quede todo consistente.

1.1. ¿Cómo modelarían una carta de Blackjack, suponiendo que nos interese tanto el palo como el número de cada carta? ¿Y si nos interesase sólo su valor, sin importar su palo?

Para modelar una carta de Blackjack se deberá hacer una función que tenga en su interior tupla del mazo de cartas y este tenga a su vez los valores de las cartas, para las cartas de letras, cada una ha de valer diez puntos por lo que solo se pondría diez en lugar de su letra.

Para el as crearemos la función:

```
def contar_as(mano):
    contador = 0
    i = 0
    while i < len(mano):
        if mano[i] == 1:
            contador += 1
        i += 1
    return contador
```

Esta función se encargará de contar cuantos haces han salido a lo largo de la partida, la idea es que después de que salga un haz (si es al principio) se le dará valor de 11, si es el segundo haz que salga, valdrá automáticamente 1. También se implementara que si es el primer haz y la mano tiene menos o igual que 10 puntos valdrá 11 automaticamente

```
def crear_mazo():
    palos = ['Corazones', 'Diamantes', 'Tréboles', 'Picas']
    valores = [2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
    mazo = [(palo, valor) for palo in palos for valor in valores]
    random.shuffle(mazo)
    return mazo

def crear_mazo_simple(): valores = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10,
10, 10] # A=1, J=10, Q=10, K=10 mazo = valores * 4 # 4 barajas, una por
cada palo return mazo
```

1.2. ¿Cómo podrían calcular el valor total de una mano (conjunto de cartas)? Elijan una mano cualquiera como ejemplo y, usando cada uno de los modelos que propusieron como respuesta al ítem anterior, describan con sus palabras cómo calcularían el valor de la mano en Python.

Podemos representar cada carta como una tupla (valor, palo) y luego calcular el valor de la mano siguiendo las mismas reglas de Blackjack.

Cada carta es una tupla en la que el primer valor es el valor de la carta y el segundo es el palo (Teniendo en cuenta que en el Blackjack el valor del palo no suma o resta puntos, es solo la clasificación de la pinta).

Para calcular el valor de la mano, iteramos sobre cada carta y sumamos sus valores. Si encontramos un As (valor 1 u 11), lo contamos inicialmente como 11 y luego tratamos de ajustarlo como 1 si la mano lo permite.

Si el total es menor a 11 y no tenemos Ases, los convertimos a 11 para mejorar la mano sin exceder 21.

1.3. En diferentes clases de la materia trabajamos con las probabilidades de que un evento aleatorio ocurra. Para responder sin escribir código: ¿Cómo creen que podrían estimar la probabilidad de ganar una mano de Blackjack?

…

Juegos de prueba, le pondremos a cada jugador una estrategia diferente, en la que eligen cuando plantarse y cuando seguir, se hará jugar varias partidas para que hayan datos de las diferentes estrategia y partidas.

Crearemos una partida con jugadores que cuenten con diferentes estrategias, la idea es probar en código cada partida dándole como parámetro las diferentes estrategias, por ejemplo una estrategia seria plantarse siempre antes de los 17 puntos, se crearía la función,

probabilidad_de_ganar (estrategia, límite) en la estrategia se le adjunta el pedir o plantarse y en el límite que decide el jugador.

Además de ello necesitaremos tener en cuenta la cantidad de cartas, la cantidad de jugadores y la mano inicial del crupier y la del jugador. Son variables que tener en cuenta

1.4. ¿Qué se necesita para simular una mano de una partida de Blackjack con múltiples jugadores? Armen un listado de los elementos del juego que necesitan incluir en su modelo en Python

…

- **Mazo de cartas**: Crear el mazo, barajarlo, y repartir las cartas.
- **Jugadores**:Número de jugadores, mano de cartas para cada jugador, límites de puntaje o decisiones de cuándo plantarse.
- **Crupier**: Mano del crupier y Lógica para jugar según reglas predefinidas (por ejemplo, pedir cartas si tiene menos de 17 puntos).
- **Contar puntos**: Función para calcular el puntaje de cada mano, ajustando Aces como 1 o 11.
- **Reglas de juego**: Decisión de cuándo cada jugador o el crupier debe pedir carta o plantarse y Lógica para determinar si un jugador o el crupier se pasa de 21.
- **Condiciones de victoria**: Función para comparar los puntajes finales y determinar el ganador.
- **Apuestas y Dinero** (Opcional): Registro del dinero de los jugadores y apuestas realizadas.
- **Estrategia de los jugadores** (Opcional): Permitir que los jugadores tengan diferentes estrategias para pedir cartas o plantarse.

1.5. Va a interesarnos explorar diferentes dinámicas o estrategias de apuestas. Podríamos por ejemplo preguntarnos a qué jugador le iría ”mejor”: Uno que sólo se planta con más de 16, otro que lo hace sólo con más de 17, otro con 18, etc. ¿Cómo podrían determinar si una estrategia es “mejor” que otra?

…

Juegos de prueba, le pondremos a cada jugador una estrategia diferente, en la que eligen cuando plantarse y cuando seguir, se hará jugar varias partidas para que hayan datos de las diferentes estrategia y partidas.

Crearemos una partida con jugadores que cuenten con diferentes estrategias, la idea es probar en código cada partida dándole como parámetro las diferentes estrategias, por ejemplo una estrategia seria plantarse siempre antes de los 17 puntos, se crearía la función, probabilidad_de_ganar (estrategia, límite) en la estrategia se le adjunta el pedir o plantarse y en el límite que decide el jugador.

Además de ello necesitaremos tener en cuenta la cantidad de cartas, la cantidad de jugadores y la mano inicial del crupier y la del jugador. Son variables que tener en cuenta

Se tendrían que poner a jugar a los jugadores con diferentes estrategias entre sí, y medir resultados

2. Primera parte de la resolución

Acá deberá incluir cada uno de los ejercicios que incluimos en la consigna en la 1era clase de laboratorio, explicar cómo es que realizaron la resolución, e incluir el código que usaron. En caso de que hubieran tomado alguna suposición para resolver el problema, deberán escribirlo en el ítem correspondiente.

2.1. Crear un mazo

Para crear el mazo nos basamos en crear una lista de valores, para las cartas de J, Q y K las pusimos solo como 10.

```
#1
#crea un mazo
def crear_mazo():
    #cartas = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
    valores = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
    mazo = valores*4
    return mazo
```

Otra manera en que hicimos para darle además del valor de la carta el nombre, fue creando un diccionario con el string del “nombre” de la carta y su valor.

```
#1
# crea un mazo
def crear_mazo():
     valores = {'A': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,
'8': 8, '9': 9, '10': 10, 'J': 10, 'Q': 10, 'K': 10}
    mazo = valores * 4
    return mazo
```

2.2. Crear un mazo para partida

Para crear el mazo se creó la función crear_mazo_partida que recibe como parámetro el número de barajas y cuenta con mazo (crear_mazo * n_barajas) que con la variable crear_mazo la modifica agregándole la multiplicación por el número de barajas. Luego se le agrega shuffle para mezclarlo.

```
def crear_mazo_partida(n_barajas):
    mazo = crear_mazo() * n_barajas
    random.shuffle(mazo)
    return mazo
```

2.3. ¡Dame una carta!:

Para la función para tomar una carta se define la función tomar_carta que recibe como parámetro a “mazo” y se le agrega el método .pop para que saque una carta, en este caso en la posición 0 que sería la primera carta del mazo

```
def tomar_carta(mazo):
    carta = mazo.pop(0)
    return carta
```

2.4. Cálculo de puntajes:

Esta función tiene un objetivo muy específico: contar la cantidad de ases que hay en una mano de cartas. Para lograr esto, la función recibe como entrada una lista que representa la mano de cartas. Luego, itera sobre cada carta de esta lista. Si encuentra una carta cuyo valor es 1 (que representa a un as), suma uno a un contador. Al finalizar el recorrido de todas las cartas, la función devuelve el valor final del contador, que indica el número total de ases encontrados en la mano.

```
def contar_as(mano):
    contador = 0
    i = 0
    while i < len(mano):
        if mano[i] == 1:
            contador += 1
        i += 1
    return contador
```

Esta función se encarga de calcular el valor total de una mano de cartas, considerando la particularidad del as. Recibe como entrada la misma lista que representa la mano.
Primero, la función determina la cantidad de ases en la mano utilizando la función contar_as. Luego, inicia un contador para llevar la cuenta del valor total de la mano. A

continuación, itera sobre cada carta de la mano. Si la carta es un as y al sumarle 11 el valor total de la mano no supera 21, se le asigna un valor de 11 al as. Si hay más de un as o si al sumarle 11 se pasa de 21, se le asigna un valor de 1 al as. Para las demás cartas, se suma su valor nominal al valor total de la mano. Finalmente, la función devuelve el valor total calculado.

```
def contar_puntos(mano):
    i = 0
    cant_as = contar_as(mano)
    puntos = 0
    while i < len(mano):
        if mano[i] == 1 and puntos + 11 < 21 and cant_as == 1:
            puntos += 11
        else:
            puntos += mano[i]
        i += 1
    return puntos
```

2.5. *Determinar si hay que seguir tomando cartas del mazo:*

La función puede_tomar_carta(mano, limite) recibe dos parámetro y devuelve True o false dependiendo Si el parámetro mano es menor al parámetro límite que sería el límite para el jugador.

```
def puede_tomar_carta(mano, limite):
    return contar_puntos (mano) < limite
```

2.6. *Jugar una partida completa:*

Este codigo se encarga de que se juegue la partida con los parametro del numero de barajas (mazo) y el limite del jugador.

```
def jugar_partida (n_barajas, limite_jugador):
    mazo = crear_mazo_partida(n_barajas)
    mano = []
    mano_crupier = []
    i = 0
    while i < 2:
        mano.append(tomar_carta(mazo))
        mano_crupier.append(tomar_carta(mazo))
        i += 1
    while contar_puntos(mano) < limite_jugador:
        mano.append(tomar_carta(mazo))
    mano_crupier= crupier(mano_crupier,contar_puntos(mano), mazo)
```

2.7. ¿Quién ganó?

Esta función se encargará de devolver un booleano centrándose en el casino, si el puntaje_final_crupier es menor o igual a 21 y es mayor que puntaje_final_jugador O si puntaje_final_jugador no llega a 21, devuelve win, dándole la victoria al crupier

```
#define si gano o perdio el casino devolviendo true o false
def casino_ganador(puntaje_final_jugador, puntaje_final_crupier):
    win = False
    if puntaje_final_crupier <= 21 and puntaje_final_crupier >
puntaje_final_jugador or puntaje_final_jugador > 21:
        win = True
    return win

#verifica que haya empate
def empate(puntaje_final_jugador, puntaje_final_crupier):
    empate = False
    if puntaje_final_crupier <= 21 and puntaje_final_crupier ==
puntaje_final_jugador:
        empate = True
    return empate
```

La función empate devuelve true si puntaje_final_crupier y puntaje_final_jugador es menor a 21 y tienen el mismo puntaje

2.8. Apostar (una partida):

La función

```
def apuesta(n_barajas, limite_jugador, dinero_total, dinero_apostado):
    dinero = dinero_total
    mano_jugador, mano_crupier = jugar_partida(n_barajas,
limite_jugador)
    if casino_ganador(contar_puntos(mano_jugador),
contar_puntos(mano_crupier)) == True and
empate(contar_puntos(mano_jugador), contar_puntos(mano_crupier)) ==
False:
        dinero -= dinero_apostado
    elif casino_ganador(contar_puntos(mano_jugador),
contar_puntos(mano_crupier)) == False and
empate(contar_puntos(mano_jugador), contar_puntos(mano_crupier)) ==
True:
        dinero += dinero_apostado
    elif empate(contar_puntos(mano_jugador),
contar_puntos(mano_crupier)) == True:
        dinero = dinero
    return dinero
```

2.9. La ruina del jugador:

```
def ruina_del_jugador(n_barajas, limite_jugador, dinero_total,
dinero_apostado):
    dinero = dinero_total
    evolucion_dinero = [dinero_total]
    while dinero != 0:
        dinero = apuesta(n_barajas, limite_jugador, dinero,
dinero_apostado)
        evolucion_dinero.append(dinero)
    return evolucion_dinero
```

…

3. Otras preguntas que nos interesa resolver

Acá deberá incluir las preguntas adicionales que les planteamos en la 2da clase de laboratorio. Como en este caso no les damos instrucciones precisas de qué deben resolver, deberán incluir más detalles de cómo y por qué decidieron resolver cada ítem de determinada manera. Se deberán incluir todos los gráficos y explicaciones que crean necesarias para responder las consignas planteadas.

3.1. Queremos explorar cómo cambia (si es que lo hace) la probabilidad de ganarle al casino dependiendo de cuál sea el límite de puntos a partir del que nuestro jugador decide plantarse.

Para cada *límite* entre 1 y 21 (1, 2, 3, …, 21), simular muchas partidas de BlackJack (de un único jugador contra el crupier) y estimar la probabilidad de ganarle al casino en cada caso. Mostrar los resultados mediante un gráfico. ¿Qué interpretaciones se pueden hacer a partir del gráfico?

Similarmente, nos interesa además saber si la probabilidad de ganar o perder depende de la cantidad de barajas francesas en el mazo de juego. Simular muchas partidas para un límite fijo (ej. 17) variando la cantidad de barajas en el mazo. ¿Observan cambios en la probabilidad de ganar del jugador en función de la cantidad de barajas en el mazo?

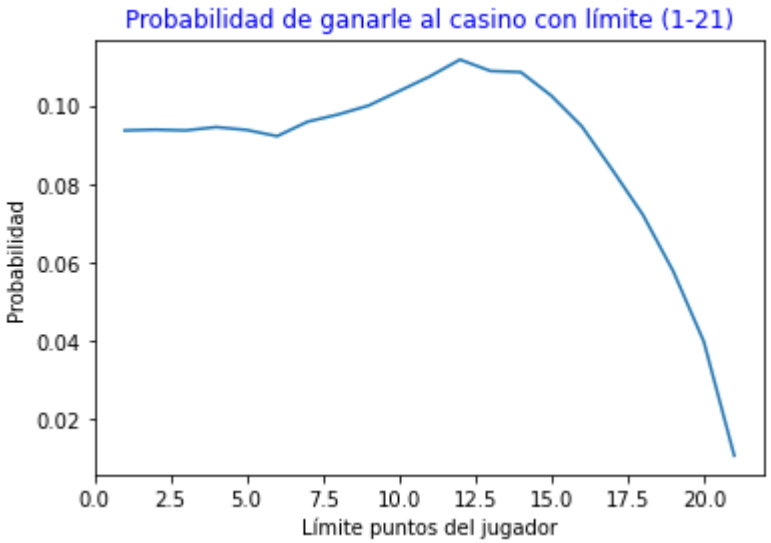
Decidimos trabajar modificando las funciones ***crupier, jugar_partida, apuesta, ruina_del_jugador*** agregando el argumento tope a cada una de ellas, para seleccionar el tope del crupier, esto también ayudará a lidiar con el ejercicio 3.3 de esta consigna.
La función **crupier_tope**, estipula un comportamiento del crupier pudiendo variar donde se para el crupier pidiendo cartas

La función **jugar_partida_con_tope**, juega una partida donde decidimos el tope que tiene el crupier
La **funcion apuesta_tope** , genera apuestas donde estipulamos el comportamiento del crupier mediante “tope”
La función **ruina_del jugador_tope**, simula partidas hasta que el jugador se quede sin dinero y definimos el tope del crupier

```
def crupier_tope(mano, puntaje_jugador, mazo, tope):
    mano_crupier = mano
    while contar_puntos(mano_crupier) < puntaje_jugador and contar_puntos(mano_crupier)
< tope:
        mano_crupier.append(tomar_carta(mazo))
    return mano_crupier

def jugar_partida_con_tope(n_barajas, limite_jugador, tope):
    mazo = crear_mazo_partida(n_barajas)
    mano = []
    mano_crupier = []
    i = 0
    while i < 2:
        mano.append(tomar_carta(mazo))
        mano_crupier.append(tomar_carta(mazo))
        i += 1
    while contar_puntos(mano) < limite_jugador:
        mano.append(tomar_carta(mazo))
    mano_crupier= crupier_tope(mano_crupier,contar_puntos(mano), mazo, tope)
    return mano, mano_crupier

def apuesta_tope(n_barajas, limite_jugador, dinero_total, dinero_apostado, tope):
    dinero = dinero_total
    mano_jugador, mano_crupier = jugar_partida_con_tope(n_barajas, limite_jugador, tope)
    if casino_ganador(contar_puntos(mano_jugador), contar_puntos(mano_crupier)) == True
and empate(contar_puntos(mano_jugador), contar_puntos(mano_crupier)) == False:
        dinero -= dinero_apostado
    elif casino_ganador(contar_puntos(mano_jugador), contar_puntos(mano_crupier)) ==
False and empate(contar_puntos(mano_jugador), contar_puntos(mano_crupier)) == True:
        dinero += dinero_apostado
    elif empate(contar_puntos(mano_jugador), contar_puntos(mano_crupier)) == True:
        dinero = dinero
    return dinero

def ruina_del_jugador_tope(n_barajas, limite_jugador, dinero_total, dinero_apostado,
tope):
    dinero = dinero_total
    evolucion_dinero = [dinero_total]
    while dinero != 0:
```

```
        dinero = apuesta_tope(n_barajas, limite_jugador, dinero, dinero_apostado, tope)
        evolucion_dinero.append(dinero)
    return evolucion_dinero

def dinamica_ganadora(lista):
    contador = 0
    i=0
    while i < len(lista)-1:
        if lista[i] < lista[i+1]:
            contador += 1
        i = i+1
    return contador / len (lista)

def simular_con_tope(cant_simulaciones, n_barajas, dinero_total, dinero_apostado, tope):
    lista = [[]]*cant_simulaciones
    promedios_sim = []
    for s in range(cant_simulaciones):
        promedios = []
        for l in range(1,22):
            lista[s]=ruina_del_jugador_tope(n_barajas, l, dinero_total, dinero_apostado,
tope)

            promedios.append(dinamica_ganadora(lista[s]))
        promedios_sim.append(promedios)
    return promedios_sim

def promedio(lista):
    return sum(lista)/len(lista)

def grafico(cant_simulaciones, n_barajas, dinero_total, dinero_apostado, tope):
    resultados = simular_con_tope(cant_simulaciones, n_barajas, dinero_total,
dinero_apostado, tope)
    promedios_sim=list (map(promedio, zip(*resultados)))
    plt.plot(range(1,22), promedios_sim)
    plt.title("Probabilidad de ganarle al casino con límite (1-21)", color = 'Blue')
    plt.ylabel("Probabilidad")
    plt.xlabel("Límite puntos del jugador")
```

Simulamos 2000 veces para obtener un resultado más suavizado. En el gráfico se observa que la probabilidad de que el jugador gane es mayor cuando se detiene con aproximadamente 13 cartas. A partir de ese punto, las probabilidades disminuyen drásticamente. Por lo tanto, apostar deteniéndose con más de 13 cartas sería un error que seguramente te costará tu dinero.



3.2. Además de la probabilidad de ganarle al casino en una partida cualquiera, nos interesa ver qué es lo que le ocurre a diferentes perfiles de apostador a lo largo de partidas (y apuestas) sucesivas. Utilicen las funciones que implementaron la clase pasada para simular estos tres perfiles:

- Jugador **serial:** Nunca se planta. Pide cartas hasta obtener BlackJack (21 puntos) o pasarse. Hace siempre la apuesta máxima.
- Jugador **moderado:** Se planta si su puntaje es 18 o más. Hace siempre la apuesta promedio.
- Jugador **reservado**: Se planta si su puntaje es 16 o más. Hace siempre la apuesta mínima.

Supongamos que cada jugador comienza con 200 fichas en total, y juega hasta alcanzar su **ruina** o hasta terminar la partida número 100. La apuesta máxima es 15 fichas, la mínima es 5, y la promedio es 10 fichas.

En un mismo gráfico, muestren qué ocurre con el dinero total de cada perfil de apostador a lo largo de la secuencia de partidas. ¿Qué parecen indicar los resultados obtenidos?

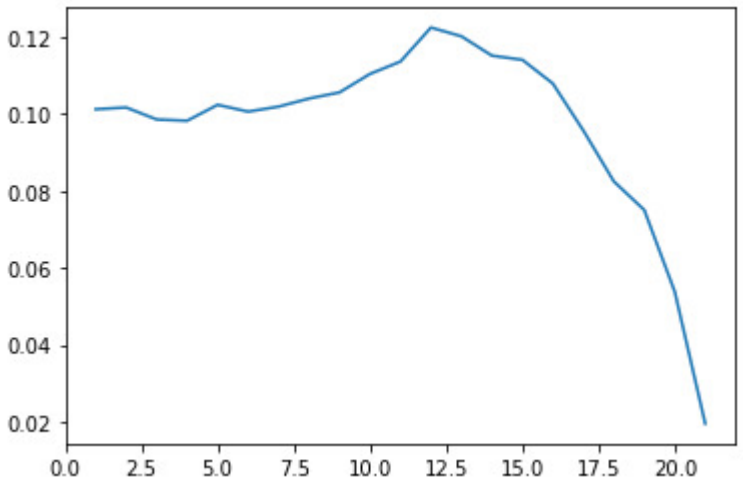
…

3.3. Hasta ahora estuvimos respetando las reglas del juego original, donde el crupier está obligado a plantarse a partir de los 17 puntos. ¿Cómo creen que cambiarían las probabilidades de ganar del casino si el crupier se plantase en cambio a partir de los 16, los 18 o los 20 puntos?

Simular partidas de BlackJack variando tanto el límite del jugador como el del crupier entre los 15 y los 21 puntos. Para cada combinación posible de límites, simular al menos 1000 partidas de BlackJack y calcular las

probabilidades de ganar del jugador en cada caso. Graficar e interpretar los resultados.

…



4. Conclusiones del trabajo

Acá deberán poner sus conclusiones sobre el trabajo con la resolución del problema:

- *Qué dificultades encontraron y cómo las solucionaron.*
- *Evaluar críticamente el resultado del trabajo comparado con lo que ustedes habían pensado en la primera clase, donde se hizo el análisis del problema (sin usar computadoras).*
- *Indicar algunas cosas que quedaría para trabajar a futuro a partir de los que hicieron como parte del trabajo, ya que sea una variante de lo hecho, o porque implicaría incluir más funcionalidades o extensiones sobre lo pedido.*

El valor del as: Fue necesario implementar la función contar_as para manejar correctamente el valor del as, asignándole 1 u 11 dependiendo de la situación, lo cual inicialmente generó complicaciones en la lógica para calcular los puntos totales.

Partidas con varios jugadores: La implementación de la función jugar_partida_muchos_jugadores presentó retos, especialmente al sincronizar las acciones de múltiples jugadores y determinar el ganador en función de las diferentes manos y límites.

Gráfico de resultados: Se encontraron dificultades al crear la función para graficar los porcentajes de partidas ganadas y perdidas. Ajustar los cálculos y representar los datos correctamente en el gráfico tomó más tiempo del esperado.

Estrategias de simulación: Al simular partidas con límites (simular_con_tope), fue necesario depurar errores relacionados con la acumulación y promedio de resultados, lo que inicialmente afectaba la precisión de las simulaciones.

En comparación con lo que habíamos pensado en la primera clase, el código que obtuvimos al final fue más práctico, con menos líneas de código, pero mucho más funcional.

Quedaría trabajar en la última función, que es la de los gráficos. Nuestra intención era crear un código que simulara múltiples partidas para los jugadores aplicando diferentes estrategias. Logramos implementarlo, pero creemos que nos gustaría profundizar más en este aspecto para explorar su potencial.