Asignatura	**Sistemas Concurrentes**
Carrera	**Ing. en Informática**
Plan	**Ajuste 2012**
Ciclo	**4to**
Cuatrimestre	**7mo**
Tema/Título	**Manual de Uso TP7**
Profesor	**Mariano Cosentino**

Grupo de Trabajo

ID/Matrícula	APELLIDO, Nombres	Correo Electrónico
ID 000-18-1410	**BELTRAME, Juan**	**juan.beltrame@comunidad.ub.edu.ar**
ID 000-17-8550	**MARTIN, Lucas**	**lucas.martin@comunidad.ub.edu.ar**
ID 000-18-0325	**GUERINI, Ignacio**	**Ignacio.guerini@comunidad.ub.edu.ar**
ID 000-17-9104	**BAZAN, Agustin**	**agustin.bazan@comunidad.ub.edu.ar**
ID 000-18-1980	**AYMONINO, Lucio**	**lucio.aymonino@comunidad.ub.edu.ar**
ID 000-20-0999	**GONZALEZ, Manuel**	**manuel.gonzalezgamito@comunidad.ub.edu.ar**

Grilla de calificación

Concepto	Propuesta	Marco Teórico	Desarrollo propio	Conclusiones	Fuentes y Referencias
Sobresaliente (10)					
Distinguido (9-8)					
Bueno (7-6)					
Aprobado (5-4)					
Insuficiente (3-2-1)					
Reprobado (0)					
NOTA					

Comentario adicional del Profesor:

Manual de Uso Detallado del Juego de Generala

Introducción

Este documento es un manual de uso detallado para “Generala”una implementación en Python del juego de dados conocido como Generala. El programa simula el juego utilizando cuatro estrategias diferentes para decidir cómo jugar, y realiza simulaciones concurrentes para comparar su desempeño. Este manual describe el propósito del programa, cómo funciona, las reglas del juego implementadas, las estrategias utilizadas, cómo ejecutar el programa y cómo interpretar los resultados.

El objetivo principal del programa es simular partidas del juego de Generala utilizando diferentes estrategias de decisión (aleatoria, codiciosa, de puntaje inmediato e inteligente) y comparar sus resultados en términos de puntaje promedio, puntaje máximo, puntaje mínimo y tiempo de ejecución.

1. Descripción General del Juego de Generala

1.1. ¿Qué es la Generala?

La Generala es un juego de dados tradicional en el que los jugadores intentan obtener combinaciones específicas de dados para acumular puntos en diferentes categorías. En este programa, se simula una versión del juego en la que un jugador lanza cinco dados hasta tres veces por turno, pudiendo elegir qué dados conservar y cuáles volver a lanzar para maximizar el puntaje en una categoría específica.

1.2. Categorías de Puntuación

El programa define las siguientes categorías de puntuación:

- **Números (1 al 6)**: Suma de todos los dados que muestran el número correspondiente (por ejemplo, para la categoría "3", se suman todos los dados que muestran un 3).
- **Escalera**: 20 puntos si los dados forman una secuencia de cinco números consecutivos (1-2-3-4-5 o 2-3-4-5-6).
- **Full**: 30 puntos si los dados tienen tres de un número y dos de otro (por ejemplo, 3-3-3-2-2).
- **Póker**: 40 puntos si los dados tienen al menos cuatro del mismo número (por ejemplo, 4-4-4-4-1).
- **Generala**: 50 puntos si los cinco dados muestran el mismo número (por ejemplo, 5-5-5-5-5).
- **Doble Generala**: 100 puntos si se obtiene otra Generala después de haber anotado una Generala previamente.

Cada categoría solo puede usarse una vez por partida, y el objetivo es maximizar el puntaje total al final de la partida, que consiste en completar todas las categorías (11 turnos en este caso).

1.3. Reglas Básicas Implementadas

- **Lanzamientos por turno**: Cada turno permite hasta tres lanzamientos de cinco dados. Después del primer y segundo lanzamiento, el jugador (o la estrategia) puede decidir qué dados conservar y cuáles relanzar.
- **Selección de categoría**: Al final de los tres lanzamientos (o antes, si se decide no relanzar), el jugador elige una categoría no utilizada para anotar el puntaje de los dados.
- **Puntaje**: El puntaje depende de la combinación de dados y la categoría seleccionada, según las reglas descritas arriba.
- **Doble Generala**: Solo se puede anotar si ya se ha anotado una Generala previamente.

2. Descripción del Programa

El programa Generala.py está diseñado para simular partidas de Generala utilizando cuatro estrategias diferentes para decidir qué dados conservar y qué

categoría elegir en cada turno. Además, utiliza hilos (threading) para ejecutar simulaciones concurrentes de cada estrategia y comparar sus resultados.

2.1. Componentes Principales

El programa está estructurado en varias funciones que manejan las reglas del juego, las estrategias y las simulaciones. A continuación, se describen los componentes principales:

- **roll_dice(n=5)**: Genera una tirada de n dados (por defecto 5), cada uno con un valor aleatorio entre 1 y 6.
- **score_category(dice, category, used_categories, generala_count)**: Calcula el puntaje de una tirada de dados para una categoría específica, considerando las categorías ya usadas y el conteo de Generalas previas.
- **Estrategias**:
    - **random_strategy()**: Elige dados para conservar y categorías de forma aleatoria.
    - **greedy_strategy()**: Conserva los dados del número más frecuente en cada tirada y elige la categoría que maximiza el puntaje inmediato.
    - **highest_immediate_score_strategy()**: Similar a la codiciosa, pero optimiza la selección de dados basada en el número con mayor frecuencia.
    - **intelligent_strategy()**: Evalúa las expectativas de puntaje para diferentes combinaciones (Generala, Póker, Full, Escalera, Número Alto) y elige la mejor opción en cada lanzamiento.
- **simular_estrategia(nombre, funcion_estrategia, n, resumen)**: Ejecuta n simulaciones de una estrategia específica y guarda los resultados (promedio, máximo, mínimo y duración) en un diccionario.
- **simular_estrategias_concurrente(n=5000)**: Ejecuta simulaciones concurrentes para todas las estrategias usando hilos y devuelve un resumen de los resultados.

2.2. Estrategias Detalladas

- **Estrategia Aleatoria (random_strategy)**:
    - **Descripción**: Conserva un número aleatorio de dados (entre 0 y 5) en cada lanzamiento y selecciona una categoría al azar entre las disponibles.

    - **Ventajas**: Simple y rápida, útil como referencia base para comparar con estrategias más sofisticadas.
    - **Desventajas**: No optimiza el puntaje, lo que resulta en puntajes bajos en promedio.
- **Estrategia Codiciosa (greedy_strategy)**:
    - **Descripción**: En cada lanzamiento, conserva los dados del número más frecuente y elige la categoría que maximiza el puntaje inmediato al final del turno.
    - **Ventajas**: Simple y efectiva para maximizar el puntaje en el corto plazo.
    - **Desventajas**: No considera el valor esperado de combinaciones futuras, lo que puede llevar a decisiones subóptimas.
- **Estrategia de Puntaje Inmediato (highest_immediate_score_strategy)**:
    - **Descripción**: Similar a la codiciosa, pero optimiza la selección de dados basándose en el número con mayor frecuencia en cada lanzamiento.
    - **Ventajas**: Mejora ligeramente la estrategia codiciosa al enfocarse en maximizar el puntaje inmediato.
    - **Desventajas**: Similar a la codiciosa, no considera el valor esperado a largo plazo.
- **Estrategia Inteligente (intelligent_strategy)**:
    - **Descripción**: Evalúa las probabilidades de obtener combinaciones específicas (Generala, Póker, Full, Escalera, Número Alto) basándose en el valor esperado de cada una. Conserva los dados que maximizan la probabilidad de lograr la combinación con mayor puntaje esperado y elige la categoría óptima al final del turno.
    - **Ventajas**: Considera el valor esperado, lo que la hace más estratégica y efectiva a largo plazo.
    - **Desventajas**: Más compleja computacionalmente, lo que puede aumentar el tiempo de ejecución.

3. Requisitos para Ejecutar el Programa

3.1. Requisitos de Software

- **Python**: Versión 3.6 o superior (el programa usa módulos estándar como random, threading, collections y time).

- **Sistema Operativo**: Cualquier sistema compatible con Python (Windows, macOS, Linux).
- **Dependencias**: No se requieren bibliotecas externas; todos los módulos utilizados son parte de la biblioteca estándar de Python.

3.2. Requisitos de Hardware

- Un ordenador con capacidad para ejecutar Python y manejar múltiples hilos (la mayoría de los ordenadores modernos cumplen con este requisito).
- Memoria RAM: Al menos 512 MB (las simulaciones con 1000 iteraciones por estrategia son ligeras).
- Procesador: Cualquier procesador moderno; las simulaciones concurrentes se benefician de múltiples núcleos.

4. Cómo Ejecutar el Programa

4.1. Preparación

- **Descarga o crea el archivo**: Asegúrate de tener el archivo `Generala.py` en tu directorio de trabajo.
- **Verifica Python**: Asegúrate de que Python esté instalado ejecutando `python --version` o `python3 --version` en la terminal.
- **Configura el entorno**: No se requiere configuración adicional, ya que el programa usa módulos estándar.

4.2. Ejecución

- **Abre una terminal**:
    - En Windows, usa la consola de comandos (`cmd`), PowerShell o un IDE como PyCharm.
    - En macOS o Linux, usa la terminal.
- **Navega al directorio**:
    - Usa el comando `cd` para ir al directorio donde está `Generala.py`. Por ejemplo:
    - bash
    - cd /ruta/al/directorio
- **Ejecuta el programa**:

- Usa el siguiente comando:
- bash
- python Generala.py
- o, si tu sistema requiere Python 3 explícitamente:
- bash
- python3 Generala.py

- **Parámetros opcionales**:
  - Por defecto, el programa ejecuta 1000 simulaciones por estrategia. Para cambiar este número, modifica el argumento `n` en la llamada a `simular_estrategias_concurrente(n)` en la sección `if __name__ == "__main__":`. Por ejemplo, para 500 simulaciones:
  - python
  - resultados = simular_estrategias_concurrente(500)

4.3. Salida del Programa

El programa imprime un resumen de los resultados para cada estrategia, incluyendo:

- **Promedio**: El puntaje promedio obtenido en las simulaciones.
- **Máximo**: El puntaje más alto obtenido en una partida.
- **Mínimo**: El puntaje más bajo obtenido en una partida.
- **Duración**: El tiempo (en segundos) que tomó ejecutar las simulaciones para esa estrategia.

Ejemplo de salida:

```
[Aleatoria] Terminó en 2.34 segundos.
[Codiciosa] Terminó en 2.56 segundos.
[Puntaje inmediato] Terminó en 2.60 segundos.
[Inteligente] Terminó en 3.12 segundos.

Estrategia: Aleatoria
  Promedio: 85.23
  Máximo: 150
  Mínimo: 50

Estrategia: Codiciosa
  Promedio: 120.45
  Máximo: 200
  Mínimo: 80
```

Estrategia: Puntaje inmediato
Promedio: 125.67
Máximo: 210
Mínimo: 85

Estrategia: Inteligente
Promedio: 140.89
Máximo: 250
Mínimo: 90

5. Personalización del Programa

5.1. Modificar el Número de Simulaciones

Para cambiar el número de simulaciones, edita la línea en if __name__ == "__main__"::

python

resultados = simular_estrategias_concurrente(1000) # Cambia 1000 por el número deseado

Un número mayor de simulaciones (por ejemplo, 5000) proporcionará resultados más precisos, pero aumentará el tiempo de ejecución.

5.2. Agregar Nuevas Categorías

Para agregar una nueva categoría de puntuación:

- Añade la categoría a la lista CATEGORIES:
- python
- CATEGORIES = ["1", "2", "3", "4", "5", "6", "Escalera", "Full", "Poker", "Generala", "Doble Generala", "Nueva Categoría"]
- Modifica la función score_category para incluir la lógica de puntuación de la nueva categoría:
- python

elif category == "Nueva Categoría":

```
    # Lógica para calcular el puntaje
```

- ```
      return puntaje
  ```
- Si la nueva categoría afecta las estrategias, actualiza las funciones como intelligent_strategy para incluirla en las decisiones.

5.3. Modificar las Estrategias

Para crear una nueva estrategia:

- Define una nueva función, por ejemplo:
- python

```
def nueva_estrategia():
    score_card = {}
    generala_count = 0
    # Lógica de la estrategia
```

- ```
      return sum(score_card.values())
  ```
- Añade la estrategia al diccionario estrategias en simular_estrategias_concurrente:
- python

```
estrategias = {
    "Aleatoria": random_strategy,
    "Codiciosa": greedy_strategy,
    "Puntaje inmediato": highest_immediate_score_strategy,
    "Inteligente": intelligent_strategy,
    "Nueva Estrategia": nueva_estrategia
```

- ```
  }
  ```

5.4. Cambiar las Probabilidades en la Estrategia Inteligente

La estrategia inteligente usa probabilidades fijas para calcular el valor esperado (por ejemplo, probs = {2: 0.03, 3: 0.15, 4: 0.42} para Generala). Para ajustar estas probabilidades:

- Modifica los valores en las funciones expected_generala, expected_poker, etc., dentro de intelligent_strategy.
- Por ejemplo:
  ```

- python

```
def expected_generala(count):
    probs = {2: 0.05, 3: 0.20, 4: 0.50}  # Nuevas probabilidades
```

- ```
      return 50 * probs.get(count, 0)
  ```

6. Interpretación de Resultados

6.1. ¿Qué Significan los Resultados?

- **Promedio**: Indica el desempeño promedio de una estrategia. Una estrategia con un promedio más alto es generalmente mejor.
- **Máximo y Mínimo**: Muestran el rango de puntajes posibles. Una estrategia con un máximo alto y un mínimo no muy bajo es más consistente.
- **Duración**: Indica el tiempo de ejecución. Estrategias más complejas (como la inteligente) suelen tomar más tiempo.

6.2. Comparación de Estrategias

- **Aleatoria**: Produce los puntajes más bajos debido a su falta de optimización.
- **Codiciosa y Puntaje Inmediato**: Mejoran el puntaje al enfocarse en maximizar el puntaje inmediato, pero no consideran el valor esperado a largo plazo.
- **Inteligente**: Tiende a obtener los mejores puntajes porque evalúa las probabilidades de combinaciones futuras, aunque puede ser más lenta.

7. Ejemplo de Uso Práctico

Supongamos que quieres comparar las estrategias con 500 simulaciones:

- Modifica el código:
- python
  ```

- resultados = simular_estrategias_concurrente(500)
- Ejecuta el programa:
- bash
- python Generala.py
- Analiza los resultados para determinar qué estrategia es la más efectiva. Por ejemplo, si la estrategia inteligente tiene un promedio de 140 y la aleatoria de 85, la inteligente es claramente superior.

8. Limitaciones y Posibles Mejoras

8.1. Limitaciones

- **Probabilidades estáticas**: La estrategia inteligente usa probabilidades fijas que podrían no ser óptimas para todas las situaciones.
- **Falta de interfaz gráfica**: El programa es de línea de comandos, lo que puede no ser intuitivo para usuarios no técnicos.
- **Simulaciones concurrentes**: Aunque usa hilos, Python tiene limitaciones con el GIL (Global Interpreter Lock), lo que puede afectar el rendimiento en máquinas con múltiples núcleos.

8.2. Posibles Mejoras

- **Interfaz gráfica**: Implementar una GUI usando Tkinter o Pygame para una experiencia más interactiva.
- **Optimización de probabilidades**: Calcular probabilidades dinámicamente basadas en el estado actual del juego.
- **Multiprocesamiento**: Usar multiprocessing en lugar de threading para mejorar el rendimiento en simulaciones concurrentes.
- **Persistencia de resultados**: Guardar los resultados en un archivo CSV o JSON para análisis posterior.

---

9. Preguntas Frecuentes

P: ¿Puedo cambiar el número de dados o lanzamientos por turno? R: Sí, modifica la función `roll_dice` para cambiar el número de dados o el bucle `while rolls_left > 0` para ajustar los lanzamientos.

P: ¿Cómo puedo ver los resultados de una sola estrategia? R: Llama directamente a `simular_estrategia` con la estrategia deseada, por ejemplo:

python

```
resumen = {}
simular_estrategia("Inteligente", intelligent_strategy, 1000, resumen)
print(resumen)
```

P: ¿Por qué la estrategia inteligente es más lenta? R: Evalúa múltiples caminos posibles y calcula valores esperados, lo que requiere más cálculos que las otras estrategias.