Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–3661 – Laboratorio de Lenguajes de Programación I.
Septiembre–Diciembre 2025

Proyecto 3: Ruby
Piedra, Papel, Tijera, Lagarto, Spock
(20 pts)

Descripción General

El objetivo de este proyecto es implementar el juego extendido de P̈iedra, Papel, Tijera, Lagarto, Spockütilizando el lenguaje **Ruby**.

El estudiante deberá modelar la lógica del juego utilizando una jerarquía de clases para los tipos de jugadas y otra para las estrategias de inteligencia artificial. Además, se deberá implementar una interfaz gráfica sencilla utilizando la gema `Shoes`.

Reglas del Juego

El juego sigue las reglas estándar popularizadas por la serie *The Big Bang Theory*:

- Tijera corta a Papel.
- Papel tapa a Piedra.
- Piedra aplasta a Lagarto.
- Lagarto envenena a Spock.
- Spock rompe a Tijera.
- Tijera decapita a Lagarto.
- Lagarto devora a Papel.
- Papel desautoriza a Spock.
- Spock vaporiza a Piedra.
- Piedra aplasta a Tijera.

Especificaciones Técnicas

El sistema debe estar modularizado en clases que representen las entidades del juego. A continuación se detallan los requisitos de implementación.

1. Jerarquía de Jugadas

Debe existir una clase padre `Jugada` y cinco subclases que hereden de ella: `Piedra`, `Papel`, `Tijera`, `Lagarto` y `Spock`.

Cada clase debe implementar al menos los siguientes métodos:

- `to_s`: Retorna la representación en string del elemento.
- `puntos(contrincante)`: Recibe una instancia de una jugada contraria y retorna una dupla (array) de dos enteros `[ptos_propios, ptos_contrario]`.
  - `[1, 0]` si la instancia actual gana.
  - `[0, 1]` si la instancia actual pierde.
  - `[0, 0]` si hay empate.

2. Jerarquía de Estrategias

Se debe implementar una clase padre `Estrategia` y diversas subclases que definan el comportamiento de los jugadores. Todas las estrategias deben implementar el método `prox(jugada_anterior_oponente)` que retorna la siguiente `Jugada`.

1. **Manual**: Solicita al usuario que introduzca su jugada por teclado o interfaz.
2. **Uniforme**: Recibe una lista de movimientos posibles en su constructor. Selecciona la próxima jugada al azar con una distribución de probabilidad uniforme sobre dicha lista (eliminando duplicados).
3. **Sesgada**: Recibe un mapa (Hash) donde las claves son las jugadas y los valores son ṕesos:̇ Debe seleccionar la jugada aleatoriamente respetando la probabilidad definida por estos pesos.
4. **Copiar**: En la primera ronda juega aleatorio o manual. En las rondas subsiguientes, repite la última jugada realizada por el oponente.
5. **Pensar**: Mantiene un registro histórico de la frecuencia con la que el oponente juega cada opción. Calcula la probabilidad de la próxima jugada del oponente basándose en su historial y selecciona la jugada que venza a la opción más probable del contrincante.

```ruby
# Ejemplo de firma para Estrategia
class Estrategia
    @@semillaPadre = 42
    def prox(j = nil)
        # Implementación en subclases
    end
end
```

3. La Partida y la Interfaz Gráfica

Se debe implementar la clase `Partida` que gestione el flujo del juego entre dos jugadores. Esta clase debe utilizar la librería `Shoes` para mostrar una interfaz gráfica.

Requisitos de la Partida:

- Debe permitir configurar los nombres y las estrategias de ambos jugadores al inicio.
- Debe soportar dos modos de juego:
  - **Rondas**: Se juega un número fijo N de iteraciones.
  - **Alcanzar**: Se juega hasta que uno de los jugadores alcance N puntos.
- Debe mostrar visualmente las jugadas (imágenes), el puntaje acumulado y permitir avanzar a la siguiente ronda mediante un botón.

> **Nota sobre Shoes**: Para instalar Shoes en su entorno, siga las instrucciones oficiales o utilice JRuby si es necesario. El código debe ejecutar el bloque `Shoes.app`.

Formato de Entrega

Debe entregar un archivo comprimido `.zip` o `.tar.gz` que contenga:

- El archivo fuente `RPTLS.rb` con todas las clases implementadas.
- Las imágenes necesarias (`Piedra.png`, `Papel.png`, etc.) en la misma carpeta.
- Un archivo `README.md` explicando cómo ejecutar el proyecto y qué dependencias instalar.

Ejemplo de Ejecución

El programa debe permitir seleccionar estrategias dinámicamente. Un ejemplo de instanciación en código (para pruebas sin GUI) sería:

```
# Configuración de estrategias
jug1 = Uniforme.new([ :Piedra, :Papel, :Tijera, :Lagarto, :Spock ])
jug2 = Pensar.new()

# Inicialización de partida
partida = Partida.new({ :Jugador1 => jug1, :Jugador2 => jug2 })

# Ejecutar modo 'Alcanzar' 5 puntos
partida.alcanzar(5)
```

La interfaz gráfica debe presentar selectores para las estrategias (`list_box`) y campos de texto para los parámetros de las estrategias complejas (como la lista para `Uniforme` o el Hash para `Sesgada`).

Entrega

Este proyecto se realizará en equipos de 2 integrantes y debe ser subido a *GitHub*. Su repositorio debe contener su archivo **main.rb** y un `README.md` identificado con el nombre y número de carnet de los integrantes junto con una breve explicación de su implementación. La fecha límite de entrega es el **miércoles de diciembre de 2025 a las 11:59 pm**.
