DIJD: Dinámico e Interactivo Juego de Dulces

Simulación Probabilística y Análisis Estadístico

Harrison Capia Tintaya
Código: 221301

17 de julio de 2025

Resumen

El presente documento describe el desarrollo e implementación del **Dinámico e Interactivo Juego de Dulces (DIJD)**, una aplicación web que simula procesos estocásticos de intercambio y distribución de recursos. El juego implementa conceptos fundamentales de probabilidad, estadística computacional y teoría de juegos a través de una interfaz interactiva desarrollada con tecnologías web modernas y Python mediante Pyodide.

Índice

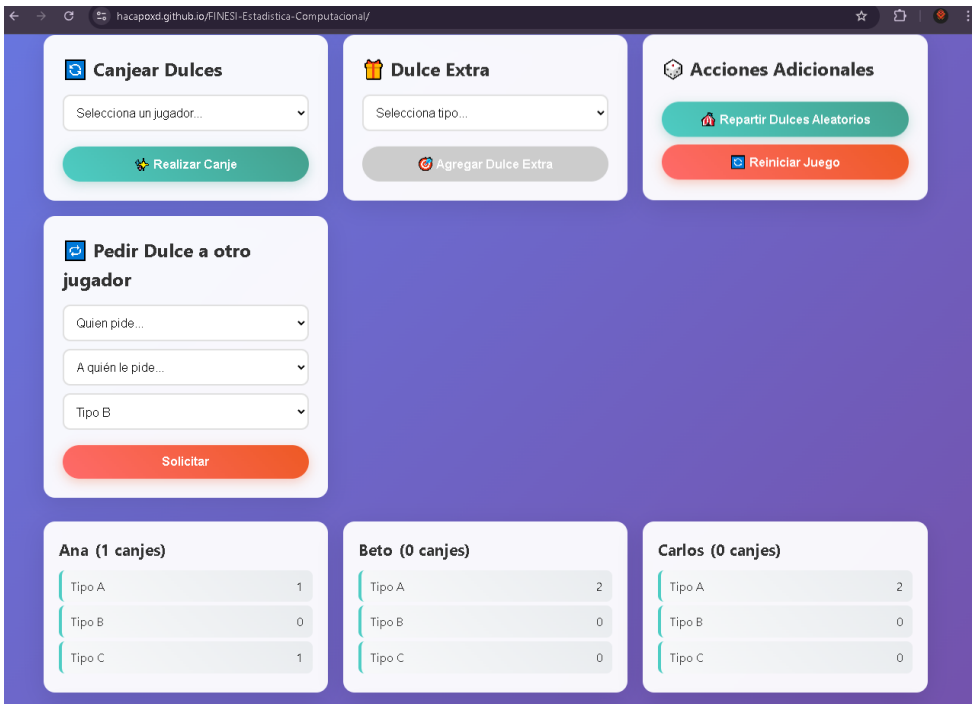

Figura 1: juego

1. Introducción

El DIJD es una simulación interactiva que modela procesos de intercambio de recursos (dulces) entre múltiples agentes (jugadores). Este sistema permite analizar patrones de distribución, estrategias de optimización y comportamientos emergentes en sistemas multi-agente.

1.1. Objetivos

- Implementar un modelo probabilístico de intercambio de recursos
- Analizar patrones de distribución estadística en tiempo real
- Proporcionar una herramienta educativa para conceptos de probabilidad
- Demostrar la aplicación de Python en el navegador mediante Pyodide

1.2. Enlaces del Proyecto

- **Juego en línea:** https://hacapoxd.github.io/FINESI-Estadistica-Computacional/
- **Repositorio:** https://github.com/Hacapoxd/FINESI-Estadistica-Computacional

2. Marco Teórico

2.1. Modelo Probabilístico

El juego implementa un modelo estocástico donde cada jugador J_i posee un vector de recursos:

$$\mathbf{R}_i = (r_{i,A}, r_{i,B}, r_{i,C}) \tag{1}$$

donde $r_{i,j}$ representa la cantidad de dulces del tipo $j \in \{A, B, C\}$ que posee el jugador i.

2.2. Condición de Canje

Un jugador puede realizar un canje exitoso si:

$$\text{mín}(r_{i,A}, r_{i,B}, r_{i,C}) \geq 1 \tag{2}$$

El canje transforma el estado del jugador según:

$$\mathbf{R}'_i = \mathbf{R}_i - (1, 1, 1) + \mathbf{e}_j \tag{3}$$

donde $\mathbf{e}_j$ es el vector unitario correspondiente al dulce extra elegido.

2.3. Distribución Aleatoria

La distribución inicial de dulces sigue una distribución uniforme discreta:

$$P(X = k) = \frac{1}{3}, \quad k \in \{A, B, C\} \tag{4}$$

Cada jugador recibe exactamente 2 dulces por ronda de distribución.

3. Arquitectura del Sistema

3.1. Componentes Principales

1. **Frontend (HTML/CSS/JavaScript):** Interfaz de usuario interactiva
2. **Backend (Python + Pyodide):** Lógica del juego y cálculos estadísticos
3. **Modelo de Datos:** Representación de jugadores y estados del juego

3.2. Clase Jugador

La clase fundamental del sistema implementa:

Listing 1: Estructura de la Clase Jugador

```
class Jugador:
    def __init__(self, nombre):
        self.nombre = nombre
        self.dulces = defaultdict(int)  # {A: n, B: m, C: p}
        self.chupetines = 0             # Canjes exitosos
        self.canjes_realizados = 0      # Contador de canjes
        self.salvado = False            # Estado de salvacion
```

3.3. Métodos Principales

- `dar_dulces_aleatorios()`: Distribuye 2 dulces aleatorios
- `canjear_dulces()`: Ejecuta la lógica de canje
- `solicitar_dulce()`: Implementa intercambio entre jugadores
- `dar_dulce_extra()`: Añade dulce de bonificación

4. Funcionalidades Implementadas

4.1. Sistema de Intercambio

- **Canje Automático:** Conversión de 3 dulces diferentes en 1 chupetín
- **Dulce Extra:** Bonificación post-canje

- **Solicitud de Dulces:** Intercambio directo entre jugadores
- **Distribución Aleatoria:** Reparto probabilístico de recursos

4.2. Sistema de Estadísticas

- Contador de jugadores activos
- Total de dulces en circulación
- Número de intercambios realizados
- Estado individual de cada jugador

5. Análisis Estadístico

5.1. Esperanza Matemática

La esperanza de dulces por tipo en la distribución inicial es:

$$E[X_j] = n \cdot 2 \cdot \frac{1}{3} = \frac{2n}{3} \tag{5}$$

donde n es el número de jugadores (9 en la implementación actual).

5.2. Varianza del Sistema

La varianza de la distribución de dulces por tipo:

$$\text{Var}(X_j) = n \cdot 2 \cdot \frac{1}{3} \cdot \frac{2}{3} = \frac{4n}{9} \tag{6}$$

5.3. Probabilidad de Canje Exitoso

Para un jugador con distribución uniforme de dulces, la probabilidad de poder realizar un canje después de k rondas de distribución requiere análisis combinatorio complejo.

6. Implementación Técnica

6.1. Tecnologías Utilizadas

- **HTML5/CSS3:** Estructura y diseño responsivo
- **JavaScript ES6+:** Lógica de interfaz y gestión de eventos
- **Python 3.x:** Lógica del juego y cálculos
- **Pyodide:** Ejecución de Python en el navegador
- **GitHub Pages:** Hosting y despliegue

6.2. Flujo de Ejecución

1. Carga de Pyodide y inicialización del intérprete Python
2. Creación de 9 jugadores con nombres predefinidos
3. Inicialización de la interfaz gráfica
4. Bucle principal de interacción usuario-sistema
5. Actualización en tiempo real de estadísticas

6.3. Gestión de Estados

El sistema mantiene consistencia entre el estado de Python y la interfaz JavaScript mediante:

- Sincronización bidireccional de datos
- Validación de acciones antes de ejecución
- Manejo de errores y estados inválidos
- Actualización reactiva de la interfaz

7. Casos de Uso y Aplicaciones

7.1. Educativo

- Enseñanza de conceptos de probabilidad
- Demostración de procesos estocásticos
- Análisis de estrategias de optimización
- Introducción a la programación científica

7.2. Investigación

- Estudio de sistemas multi-agente
- Análisis de comportamientos emergentes
- Modelado de economías simplificadas
- Validación de teorías de juegos

8. Resultados y Análisis

8.1. Métricas de Rendimiento

- Tiempo de carga: < 5 segundos
- Tiempo de respuesta por acción: < 100 ms
- Memoria utilizada: ≈ 50 MB
- Compatibilidad: Navegadores modernos

8.2. Observaciones del Comportamiento

- La estrategia óptima depende del estado global del sistema
- Los intercambios directos pueden ser más eficientes que esperar distribuciones aleatorias
- El equilibrio del sistema tiende hacia una distribución uniforme a largo plazo

9. Conclusiones

El DIJD demuestra exitosamente la aplicación de conceptos estadísticos y probabilísticos en un entorno interactivo. La implementación combina teoría matemática sólida con tecnologías web modernas, resultando en una herramienta educativa y de investigación versátil.

9.1. Logros Principales

- Implementación completa de un modelo probabilístico
- Interfaz intuitiva y responsiva
- Integración exitosa de Python en el navegador
- Sistema robusto de gestión de estados

9.2. Trabajo Futuro

- Implementación de algoritmos de optimización automática
- Análisis estadístico avanzado con visualizaciones
- Modo multijugador en tiempo real
- Exportación de datos para análisis posterior

10. Referencias

Referencias

[1] Pyodide Development Team. (2023). *Pyodide: Python with the scientific stack, compiled to WebAssembly*. https://pyodide.org/

[2] Ross, S. M. (2014). *Introduction to Probability Models* (11th ed.). Academic Press.

[3] Osborne, M. J., & Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press.

[4] Flanagan, D. (2020). *JavaScript: The Definitive Guide* (7th ed.). O'Reilly Media.

A. Código Fuente Principal

A.1. Clase Jugador Completa

Listing 2: Implementación Completa de la Clase Jugador

```
from collections import defaultdict
import random

TIPOS_DULCES = ['A', 'B', 'C']

class Jugador:
    def __init__(self, nombre):
        self.nombre = nombre
        self.dulces = defaultdict(int)
        self.chupetines = 0
        self.canjes_realizados = 0
        self.salvado = False

    def dar_dulces_aleatorios(self):
        for _ in range(2):
            tipo = random.choice(TIPOS_DULCES)
            self.dulces[tipo] += 1

    def canjear_dulces(self):
        if all(self.dulces[tipo] >= 1 for tipo in TIPOS_DULCES):
            for tipo in TIPOS_DULCES:
                self.dulces[tipo] -= 1
            self.chupetines += 1
            self.canjes_realizados += 1
            self.salvado = True
            return True
        return False

    def solicitar_dulce(self, otro, tipo):
        if otro.tiene_dulce(tipo):
```

```
31                 if otro.quitar_dulce(tipo):
32                     self.dar_dulce(tipo)
33                     return True
34             return False
```