Simulador del Juego de Caramelos

Aplicación Streamlit con Explicación

Edilberto Mamani Emanuel

Julio 2025

1. Objetivo del Juego

El objetivo del "Juego de Caramelos" es crear **chupetines** (*lollipops*) a partir de una distribución inicial de caramelos de distintos tipos. El jugador debe lograr al menos un chupetín por cada persona del grupo, usando reglas de combinación e intercambio.

2. Reglas del Juego

- Hay distintos tipos de caramelos (ej.: limón, huevo, pera).
- Cada persona recibe una cantidad fija de caramelos.
- Se pueden fabricar chupetines combinando cierto número de caramelos.
- Si no se puede fabricar chupetines, pueden venderse para obtener más caramelos.

3. Componentes y Parámetros

- **Personas:** número total de participantes.
- **Caramelos por persona:** cantidad inicial asignada.
- **Chupetín:** requiere una cantidad de caramelos balanceados.
- **Caramelos extra:** obtenidos al fabricar o vender un chupetín.

4. Lógica del Juego

1. Se genera una distribución inicial de caramelos.
2. Se acumulan todos en un inventario común.
3. Mientras se pueda, se fabrican chupetines.
4. Si no alcanza, se vende uno para obtener caramelos y continuar.
5. El proceso se repite hasta lograr un chupetín por persona o se termina el recurso.

5. Modos de Simulación

- **Simulación Individual:** ejecución paso a paso.
- **Reparto Manual:** el usuario elige qué caramelos recibe cada persona.
- **Simulaciones Múltiples:** múltiples ejecuciones con análisis estadístico.

6. Código Completo de la Aplicación (Streamlit)

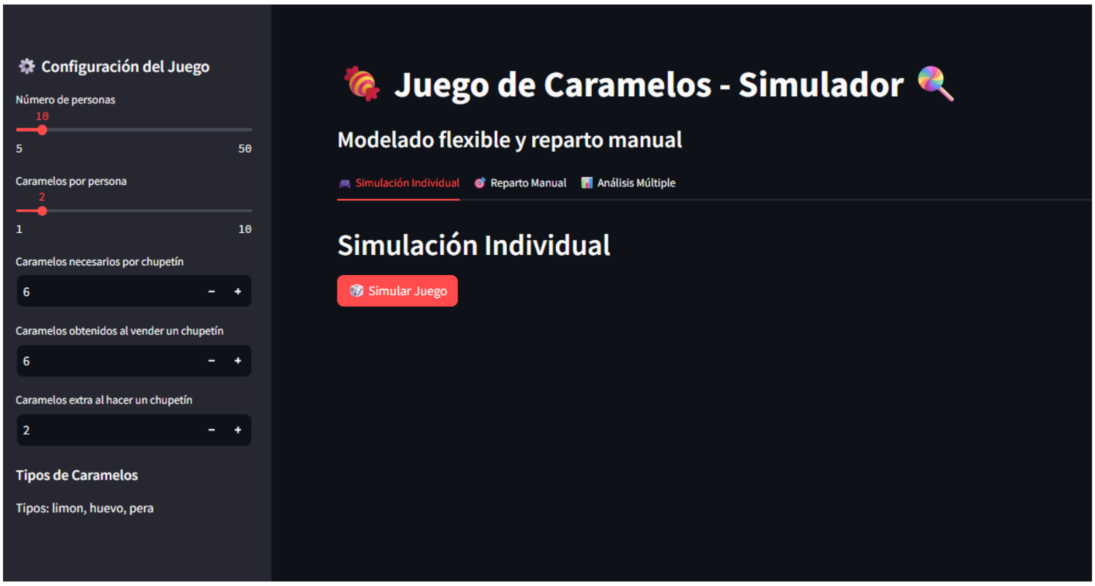
```python
import streamlit as st
import random
from collections import Counter
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Imports opcionales
try:
    import optuna
    OPTUNA_AVAILABLE = True
except ImportError:
    OPTUNA_AVAILABLE = False

class CandyGameModel:
    def __init__(self,
                 candy_types=['limon', 'huevo', 'pera'],
                 num_people=10,
                 candies_per_person=2,
                 candies_per_lollipop=6,
                 candies_from_selling=6,
                 extra_candies_from_making=2):

        self.candy_types = candy_types
        self.num_people = num_people
        self.candies_per_person = candies_per_person
        self.candies_per_lollipop = candies_per_lollipop
        self.candies_from_selling = candies_from_selling
        self.extra_candies_from_making = extra_candies_from_making

        # Calculamos cu ntos de cada tipo necesitamos para hacer un chupet n
        self.candies_per_type_for_lollipop = max(1, candies_per_lollipop // len(candy_types))

    def can_make_lollipop(self, inventory):
        """Verifica si se puede hacer un chupet n con el inventario actual"""
        for candy_type in self.candy_types:
            if inventory[candy_type] < self.candies_per_type_for_lollipop:
                return False
        return True

    def make_lollipop(self, inventory, steps):
        """Hace un chupet n y actualiza el inventario"""
        # Usar los caramelos necesarios
        for candy_type in self.candy_types:
            inventory[candy_type] -= self.candies_per_type_for_lollipop

        # Agregar caramelos extra estrat gicos
        missing = self.get_missing_candies(inventory)
        extra = []

        for candy, amount in missing.items():
            extra.extend([candy] * min(amount, self.extra_candies_from_making - len(extra)))

        while len(extra) < self.extra_candies_from_making:
            extra.append(random.choice(self.candy_types))

        for candy in extra:
            inventory[candy] += 1

        cost_description = f"usando␣{self.candies_per_type_for_lollipop}␣de␣cada␣tipo"
        steps.append(f"Se␣hizo␣1␣chupet n␣({cost_description})␣y␣se␣recibieron␣{self.extra_candies_from_making}␣caramelos␣extra:␣{extra}")
        return 1

    def sell_lollipop(self, inventory, steps):
        """Vende un chupet n para obtener caramelos"""
        missing = self.get_missing_candies(inventory)
        chosen = []

        for candy, amount in missing.items():
            chosen.extend([candy] * min(amount, self.candies_from_selling - len(chosen)))

        while len(chosen) < self.candies_from_selling:
            chosen.append(random.choice(self.candy_types))

        for candy in chosen:
            inventory[candy] += 1

        steps.append(f"Se␣vendi ␣1␣chupet n␣para␣recibir␣{self.candies_from_selling}␣caramelos:␣{chosen}")

    def get_missing_candies(self, inventory):
        """Calcula qu  caramelos faltan para la pr xima combinaci n"""
        missing = {}
        for candy_type in self.candy_types:
            missing[candy_type] = max(0, self.candies_per_type_for_lollipop - inventory[candy_type])
        return dict(sorted(missing.items(), key=lambda x: -x[1]))

    def simulate_game(self, max_iterations=1000, custom_distribution=None):
        """Simula un juego completo con distribuci n opcional personalizada"""
        steps = []

        # Reparto inicial - usar custom_distribution si se proporciona
        if custom_distribution:
            people_candies = custom_distribution
        else:
            people_candies = [random.choices(self.candy_types, k=self.candies_per_person)
                              for _ in range(self.num_people)]

        all_candies = [c for pair in people_candies for c in pair]
        inventory = Counter(all_candies)

        lollipops = 0
        exchanges = 0
        iterations = 0

        steps.append("   ␣Configuraci n␣del␣juego:")
        steps.append(f"Tipos␣de␣caramelos:␣{self.candy_types}")
        steps.append(f"Personas:␣{self.num_people}")
        steps.append(f"Caramelos␣por␣persona:␣{self.candies_per_person}")
        steps.append(f"Caramelos␣necesarios␣por␣chupet n:␣{self.candies_per_lollipop}␣({self.candies_per_type_for_lollipop}␣de␣cada␣tipo)")
        steps.append(f"Caramelos␣obtenidos␣al␣vender:␣{self.candies_from_selling}")
        steps.append(f"Caramelos␣extra␣al␣hacer:␣{self.extra_candies_from_making}")
        steps.append("")

        steps.append("   ␣Reparto␣inicial:")
        for i, candies in enumerate(people_candies, 1):
            steps.append(f"Persona␣{i}:␣{candies}")
        steps.append(f"Inventario␣inicial:␣{dict(inventory)}")
        steps.append("")

        # Hacer chupetines mientras se pueda
        while self.can_make_lollipop(inventory):
            lollipops += self.make_lollipop(inventory, steps)

        # Si no alcanza, vender chupetines y continuar
        while lollipops < self.num_people and iterations < max_iterations:
            if lollipops == 0:
                steps.append("   ␣No␣se␣pueden␣hacer␣m s␣chupetines␣ni␣vender.")
                break

            self.sell_lollipop(inventory, steps)
            lollipops -= 1
            exchanges += 1

            while self.can_make_lollipop(inventory):
                lollipops += self.make_lollipop(inventory, steps)

            iterations += 1

        # Resultados
        success = lollipops >= self.num_people

        return {
            'steps': steps,
            'lollipops': lollipops,
            'exchanges': exchanges,
            'success': success,
            'iterations': iterations,
            'final_inventory': dict(inventory),
            'people_candies': people_candies
        }

def create_manual_distribution_interface(candy_types, num_people, candies_per_person):
    """Crea la interfaz para reparto manual de caramelos"""
    st.subheader("      ␣Reparto␣Manual␣de␣Caramelos")
    st.info(f"Asigna␣{candies_per_person}␣caramelos␣a␣cada␣una␣de␣las␣{num_people}␣personas")

    # Inicializar distribuci n en session_state si no existe
    if 'manual_distribution' not in st.session_state:
        st.session_state.manual_distribution = [[] for _ in range(num_people)]

    # Verificar si el n mero de personas cambi
    if len(st.session_state.manual_distribution) != num_people:
        st.session_state.manual_distribution = [[] for _ in range(num_people)]

    # Botones para generar distribuciones autom ticas
    col1, col2, col3 = st.columns(3)

    with col1:
        if st.button("      ␣Generar␣Aleatorio"):
            st.session_state.manual_distribution = [
                random.choices(candy_types, k=candies_per_person)
                for _ in range(num_people)
            ]

    with col2:
        if st.button("      ␣Distribuci n␣Balanceada"):
            # Intentar distribuir de manera m s equitativa
            total_candies = num_people * candies_per_person
            candies_per_type = total_candies // len(candy_types)
            remaining = total_candies % len(candy_types)

            # Crear pool de caramelos balanceado
            candy_pool = []
            for i, candy_type in enumerate(candy_types):
                count = candies_per_type + (1 if i < remaining else 0)
                candy_pool.extend([candy_type] * count)

            random.shuffle(candy_pool)

            # Distribuir entre personas
            st.session_state.manual_distribution = []
            for i in range(num_people):
                start_idx = i * candies_per_person
                end_idx = start_idx + candies_per_person
                st.session_state.manual_distribution.append(candy_pool[start_idx:end_idx])

    with col3:
        if st.button("      ␣Limpiar␣Todo"):
            st.session_state.manual_distribution = [[] for _ in range(num_people)]

    # Interfaz para cada persona
    distribution_valid = True

    for i in range(num_people):
        st.write(f"**Persona␣{i+1}:**")

        # Crear columnas para los selectores
        cols = st.columns(candies_per_person + 1)

        # Asegurar que la lista tenga el tama o correcto
        while len(st.session_state.manual_distribution[i]) < candies_per_person:
            st.session_state.manual_distribution[i].append(candy_types[0])

        # Selectores para cada caramelo
        for j in range(candies_per_person):
            with cols[j]:
                current_value = st.session_state.manual_distribution[i][j]
                new_value = st.selectbox(
                    f"Caramelo␣{j+1}",
                    candy_types,
                    index=candy_types.index(current_value) if current_value in candy_types else 0,
                    key=f"person_{i}_candy_{j}"
                )
                st.session_state.manual_distribution[i][j] = new_value

        # Mostrar resumen de la persona
        with cols[-1]:
            person_counter = Counter(st.session_state.manual_distribution[i])
            st.write("**Resumen:**")
            for candy_type in candy_types:
                count = person_counter[candy_type]
                if count > 0:
                    st.write(f"{candy_type}:␣{count}")

    # Mostrar resumen total
    st.subheader("      ␣Resumen␣Total")

    all_candies = [candy for person in st.session_state.manual_distribution for candy in person]
    total_counter = Counter(all_candies)

    col1, col2 = st.columns(2)

    with col1:
        st.write("**Distribuci n␣Total:**")
        for candy_type in candy_types:
            count = total_counter[candy_type]
            st.write(f"      ␣{candy_type.title()}:␣{count}")

    with col2:
        # Verificar si la distribuci n es v lida
        total_candies = sum(len(person) for person in st.session_state.manual_distribution)
        expected_total = num_people * candies_per_person

        if total_candies == expected_total:
            st.success(f"   ␣Distribuci n␣v lida:␣{total_candies}␣caramelos")
        else:
            st.error(f"   ␣Distribuci n␣inv lida:␣{total_candies}/{expected_total}␣caramelos")
            distribution_valid = False

        # Mostrar balance
        if len(candy_types) > 1:
            min_count = min(total_counter[ct] for ct in candy_types)
            max_count = max(total_counter[ct] for ct in candy_types)
            balance = max_count - min_count

            if balance <= 1:
                st.info(f"      ␣Muy␣balanceado␣(diff:␣{balance})")
            elif balance <= 3:
                st.warning(f"      ␣Moderadamente␣balanceado␣(diff:␣{balance})")
            else:
                st.error(f"      ␣Desbalanceado␣(diff:␣{balance})")

    return st.session_state.manual_distribution if distribution_valid else None

def optimize_with_optuna(base_config, n_trials=50, progress_bar=None):
    """Optimiza los par metros del juego usando Optuna"""
    if not OPTUNA_AVAILABLE:
        raise ImportError("Optuna␣no␣est ␣instalado.␣Instala␣con:␣pip␣install␣optuna")

    def objective(trial):
        # Par metros a optimizar
        candies_per_lollipop = trial.suggest_int('candies_per_lollipop', 3, 15)
        candies_from_selling = trial.suggest_int('candies_from_selling', 4, 20)
        extra_candies_from_making = trial.suggest_int('extra_candies_from_making', 1, 8)

        # Crear modelo con par metros optimizados
        model = CandyGameModel(
            candy_types=base_config['candy_types'],
            num_people=base_config['num_people'],
            candies_per_person=base_config['candies_per_person'],
            candies_per_lollipop=candies_per_lollipop,
            candies_from_selling=candies_from_selling,
```

```python
            extra_candies_from_making=extra_candies_from_making
        )

        # Simular m ltiples juegos para obtener estad sticas robustas
        successes = 0
        total_exchanges = 0

        for _ in range(20):  # 20 simulaciones por trial
            result = model.simulate_game()
            if result['success']:
                successes += 1
            total_exchanges += result['exchanges']

        success_rate = successes / 20
        avg_exchanges = total_exchanges / 20

        # Funci n objetivo: maximizar tasa de  xito , minimizar intercambios
        return success_rate - (avg_exchanges * 0.01)  # Penalizar muchos intercambios

    study = optuna.create_study(direction='maximize')

    if progress_bar:
        for i in range(n_trials):
            study.optimize(objective, n_trials=1)
            progress_bar.progress((i + 1) / n_trials)
    else:
        study.optimize(objective, n_trials=n_trials)

    return study

def create_visualizations(results_data):
    """Crea visualizaciones con Plotly"""

    # Gr fico de distribuci n de intercambios
    exchange_counts = [r['exchanges'] for r in results_data]
    exchange_dist = Counter(exchange_counts)

    fig_exchanges = px.bar(
        x=list(exchange_dist.keys()),
        y=list(exchange_dist.values()),
        title="Distribuci n␣de␣Intercambios",
        labels={'x': 'N mero␣de␣Intercambios', 'y': 'Frecuencia'},
        color=list(exchange_dist.values()),
        color_continuous_scale='Viridis'
    )

    # Gr fico de tasa de  xito
    successes = sum(1 for r in results_data if r['success'])
    success_rate = (successes / len(results_data)) * 100

    fig_success = go.Figure(go.Indicator(
        mode = "gauge+number",
        value = success_rate,
        domain = {'x': [0, 1], 'y': [0, 1]},
        title = {'text': "Tasa␣de␣ xito ␣(%)"},
        gauge = {
            'axis': {'range': [None, 100]},
            'bar': {'color': "darkblue"},
            'steps': [
                {'range': [0, 50], 'color': "lightgray"},
                {'range': [50, 80], 'color': "yellow"},
                {'range': [80, 100], 'color': "green"}
            ],
            'threshold': {
                'line': {'color': "red", 'width': 4},
                'thickness': 0.75,
                'value': 90
            }
        }
    ))

    return fig_exchanges, fig_success

def main():
    # Configuraci n de la p gina
    st.set_page_config(
        page_title="Juego␣de␣Caramelos␣ ",
        page_icon=" ",
        layout="wide",
        initial_sidebar_state="expanded"
    )

    # T tulo principal
    st.title(" ␣Juego␣de␣Caramelos␣-␣Simulador␣ ")
    st.markdown("###␣Modelado␣flexible␣y␣reparto␣manual")

    # Sidebar para configuraci n
    st.sidebar.header(" ␣Configuraci n␣del␣Juego")

    # Par metros del juego (ahora con inputs num ricos)
    num_people = st.sidebar.slider("N mero␣de␣personas", 5, 50, 10)
    candies_per_person = st.sidebar.slider("Caramelos␣por␣persona", 1, 10, 2)

    # Variables clave que ahora puedes modificar con st.number_input
    candies_per_lollipop = st.sidebar.number_input(
        "Caramelos␣necesarios␣por␣chupet n",
        min_value=3, max_value=20, value=6, step=1
    )
    candies_from_selling = st.sidebar.number_input(
        "Caramelos␣obtenidos␣al␣vender␣un␣chupet n",
        min_value=3, max_value=25, value=6, step=1
    )
    extra_candies_from_making = st.sidebar.number_input(
        "Caramelos␣extra␣al␣hacer␣un␣chupet n",
        min_value=1, max_value=10, value=2, step=1
    )

    # Tipos de caramelos
    st.sidebar.subheader("Tipos␣de␣Caramelos")
    candy_types = ['limon', 'huevo', 'pera']  # Podr amos hacer esto configurable tambi n
    st.sidebar.write(f"Tipos:␣{',␣'.join(candy_types)}")

    # Crear modelo
    model = CandyGameModel(
        candy_types=candy_types,
        num_people=num_people,
        candies_per_person=candies_per_person,
        candies_per_lollipop=candies_per_lollipop,
        candies_from_selling=candies_from_selling,
        extra_candies_from_making=extra_candies_from_making
    )

    # Tabs principales
    tab1, tab2, tab3 = st.tabs(["  ␣Simulaci n␣Individual", " ␣Reparto␣Manual", "  ␣An lisis␣M ltiple"])

    with tab1:
        st.header("Simulaci n␣Individual")

        if st.button("  ␣Simular␣Juego", type="primary"):
            with st.spinner("Simulando..."):
                result = model.simulate_game()

            col1, col2 = st.columns([2, 1])

            with col1:
                st.subheader("  ␣Log␣de␣Simulaci n")
                log_text = "\n".join(result['steps'])
                st.text_area("Pasos␣de␣la␣simulaci n:", log_text, height=400)

            with col2:
                st.subheader("  ␣Resultados")

                # M tricas principales
                st.metric("  ␣Chupetines␣totales", result['lollipops'])
                st.metric("  ␣Intercambios␣realizados", result['exchanges'])

                # Estado final
                if result['success']:
                    st.success(" ␣ Objetivo ␣logrado!")
                else:
                    st.error(" ␣No␣se␣logr ␣el␣objetivo")

                # Inventario final
                st.subheader("  ␣Inventario␣Final")
                for candy, count in result['final_inventory'].items():
                    st.write(f"**{candy.title()}:**␣{count}")

                # Reparto inicial
                st.subheader("  ␣Reparto␣Inicial")
                people_df = pd.DataFrame([
                    {"Persona": i+1, "Caramelos": ",␣".join(candies)}
                    for i, candies in enumerate(result['people_candies'])
                ])
                st.dataframe(people_df, use_container_width=True)

    with tab2:
        st.header(" ␣Reparto␣Manual␣de␣Caramelos")

        # Crear interfaz de reparto manual
        manual_distribution = create_manual_distribution_interface(
            candy_types, num_people, candies_per_person
        )

        if manual_distribution:
            st.subheader(" ␣Simular␣con␣Reparto␣Manual")

            if st.button(" ␣Simular␣con␣Distribuci n␣Manual", type
               ="primary"):
                with st.spinner("Simulando␣con␣distribuci n␣manual...")
                   :
                    result = model.simulate_game(custom_distribution=
                       manual_distribution)

                col1, col2 = st.columns([2, 1])

                with col1:
                    st.subheader(" ␣Log␣de␣Simulaci n")
                    log_text = "\n".join(result['steps'])
                    st.text_area("Pasos␣de␣la␣simulaci n:", log_text,
                       height=400, key="manual_log")

                with col2:
                    st.subheader(" ␣Resultados")

                    # M tricas principales
                    st.metric(" ␣Chupetines␣totales", result['
                       lollipops'])
                    st.metric(" ␣Intercambios␣realizados", result['
                       exchanges'])

                    # Estado final
                    if result['success']:
                        st.success(" ␣ Objetivo ␣logrado!")
                    else:
                        st.error(" ␣No␣se␣logr ␣el␣objetivo")

                    # Inventario final
                    st.subheader(" ␣Inventario␣Final")
                    for candy, count in result['final_inventory'].items
                       ():
                        st.write(f"**{candy.title()}:**␣{count}")

                    # An lisis de la distribuci n manual
                    st.subheader(" ␣An lisis␣de␣Distribuci n")
                    all_candies = [c for person in manual_distribution
                       for c in person]
                    counter = Counter(all_candies)

                    for candy_type in candy_types:
                        count = counter[candy_type]
                        percentage = (count / len(all_candies)) * 100
                        st.write(f"**{candy_type.title()}:**␣{count}␣({
                           percentage:.1f}%)")

    with tab3:
        st.header("An lisis␣de␣M ltiples␣Simulaciones")

        num_simulations = st.slider("N mero␣de␣simulaciones", 10, 1000, 100)

        if st.button("      ␣Ejecutar␣Simulaciones␣M ltiples", type="primary"):
            progress_bar = st.progress(0)
            status_text = st.empty()

            results = []

            for i in range(num_simulations):
                progress_bar.progress((i + 1) / num_simulations)
                status_text.text(f"Simulaci n␣{i␣+␣1}/{num_simulations}")

                result = model.simulate_game()
                results.append(result)

            status_text.text("   ␣Simulaciones␣completadas!")

            # Calcular estad sticas
            successes = sum(1 for r in results if r['success'])
            success_rate = (successes / num_simulations) * 100
            exchanges = [r['exchanges'] for r in results]
            avg_exchanges = sum(exchanges) / num_simulations

            # Mostrar m tricas
            col1, col2, col3, col4 = st.columns(4)

            with col1:
                st.metric("      ␣Tasa␣de␣ xito ", f"{success_rate:.1f}%")

            with col2:
                st.metric("      ␣Promedio␣Intercambios", f"{avg_exchanges:.2f}")

            with col3:
                st.metric("      ␣Min␣Intercambios", min(exchanges))

            with col4:
                st.metric("      ␣Max␣Intercambios", max(exchanges))

            # Crear y mostrar visualizaciones
            fig_exchanges, fig_success = create_visualizations(results)

            col1, col2 = st.columns(2)
            with col1:
                st.plotly_chart(fig_exchanges, use_container_width=True)
            with col2:
                st.plotly_chart(fig_success, use_container_width=True)

            # Tabla de distribuci n
            st.subheader("      ␣Distribuci n␣Detallada")
            exchange_dist = Counter(exchanges)
            dist_df = pd.DataFrame([
                {
                    "Intercambios": exchanges,
                    "Frecuencia": count,
                    "Porcentaje": f"{(count/num_simulations)*100:.1f}%"
                }
                for exchanges, count in sorted(exchange_dist.items())
            ])
            st.dataframe(dist_df, use_container_width=True)

if __name__ == "__main__":
    main()
```

Listing 1: Código del Juego de Caramelos en Streamlit

7. Capturas de Pantalla

A continuación se presentan cinco imágenes del funcionamiento de la aplicación:



Configuración del Juego

Número de personas: 10 (5–50)

Caramelos por persona: 2 (1–10)

Caramelos necesarios por chupetín: 6

Caramelos obtenidos al vender un chupetín: 6

Caramelos extra al hacer un chupetín: 2

Tipos de Caramelos

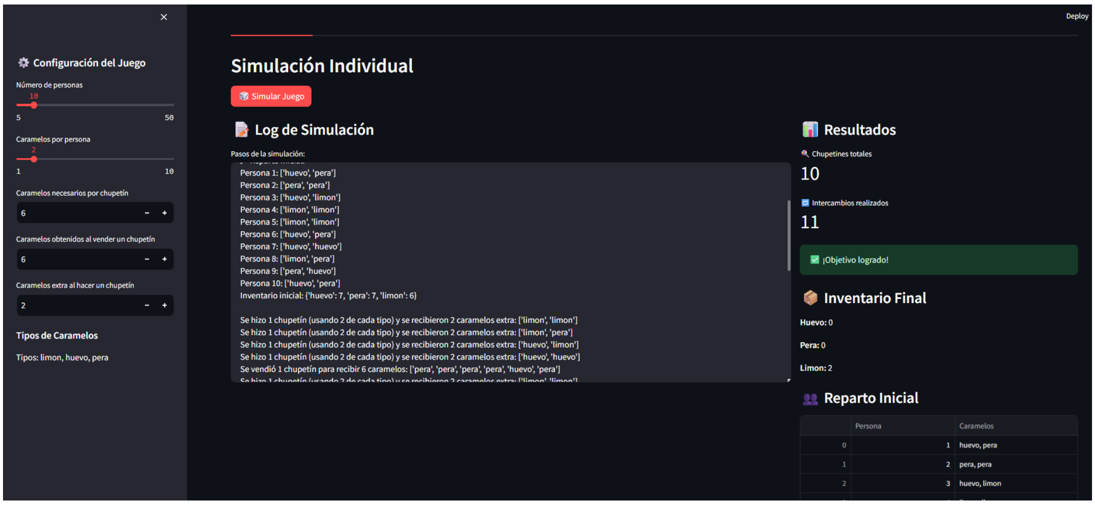
Tipos: limon, huevo, pera

Juego de Caramelos - Simulador

Modelado flexible y reparto manual

Simulación Individual

Figura 1: Pantalla de configuración inicial con controles en la barra lateral.



⚙ Configuración del Juego

Número de personas: 10 (5 – 50)

Caramelos por persona: 2 (1 – 10)

Caramelos necesarios por chupetín: 6

Caramelos obtenidos al vender un chupetín: 6

Caramelos extra al hacer un chupetín: 2

Tipos de Caramelos

Tipos: limon, huevo, pera

Simulación Individual

Log de Simulación

Pasos de la simulación:

```
Persona 1: ['huevo', 'pera']
Persona 2: ['pera', 'pera']
Persona 3: ['huevo', 'limon']
Persona 4: ['limon', 'limon']
Persona 5: ['limon', 'limon']
Persona 6: ['huevo', 'pera']
Persona 7: ['huevo', 'huevo']
Persona 8: ['limon', 'pera']
Persona 9: ['pera', 'huevo']
Persona 10: ['huevo', 'pera']
Inventario inicial: {'huevo': 7, 'pera': 7, 'limon': 6}

Se hizo 1 chupetín (usando 2 de cada tipo) y se recibieron 2 caramelos extra: ['limon', 'limon']
Se hizo 1 chupetín (usando 2 de cada tipo) y se recibieron 2 caramelos extra: ['limon', 'pera']
Se hizo 1 chupetín (usando 2 de cada tipo) y se recibieron 2 caramelos extra: ['huevo', 'limon']
Se hizo 1 chupetín (usando 2 de cada tipo) y se recibieron 2 caramelos extra: ['huevo', 'huevo']
Se vendió 1 chupetín para recibir 6 caramelos: ['pera', 'pera', 'pera', 'pera', 'huevo', 'pera']
Se hizo 1 chupetín (usando 2 de cada tipo) y se recibieron 2 caramelos extra: ['limon', 'limon']
```

Resultados

Chupetines totales: 10

Intercambios realizados: 11

✅ ¡Objetivo logrado!

Inventario Final

Huevo: 0

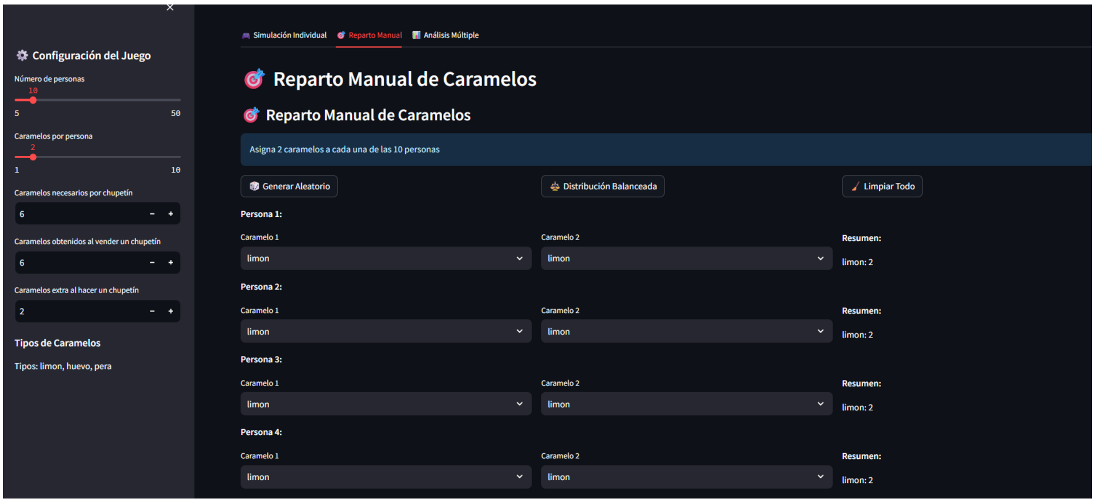
Pera: 0

Limon: 2

Reparto Inicial

	Persona	Caramelos
0	1	huevo, pera
1	2	pera, pera
2	3	huevo, limon

Figura 2: Simulación individual mostrando el log de pasos y el inventario final.



⚙ Configuración del Juego

Número de personas: 10 (5 – 50)

Caramelos por persona: 2 (1 – 10)

Caramelos necesarios por chupetín: 6

Caramelos obtenidos al vender un chupetín: 6

Caramelos extra al hacer un chupetín: 2

Tipos de Caramelos

Tipos: limon, huevo, pera

Reparto Manual de Caramelos

Reparto Manual de Caramelos

Asigna 2 caramelos a cada una de las 10 personas

Persona 1:

Caramelo 1: limon | Caramelo 2: limon | **Resumen:** limon: 2

Persona 2:

Caramelo 1: limon | Caramelo 2: limon | **Resumen:** limon: 2

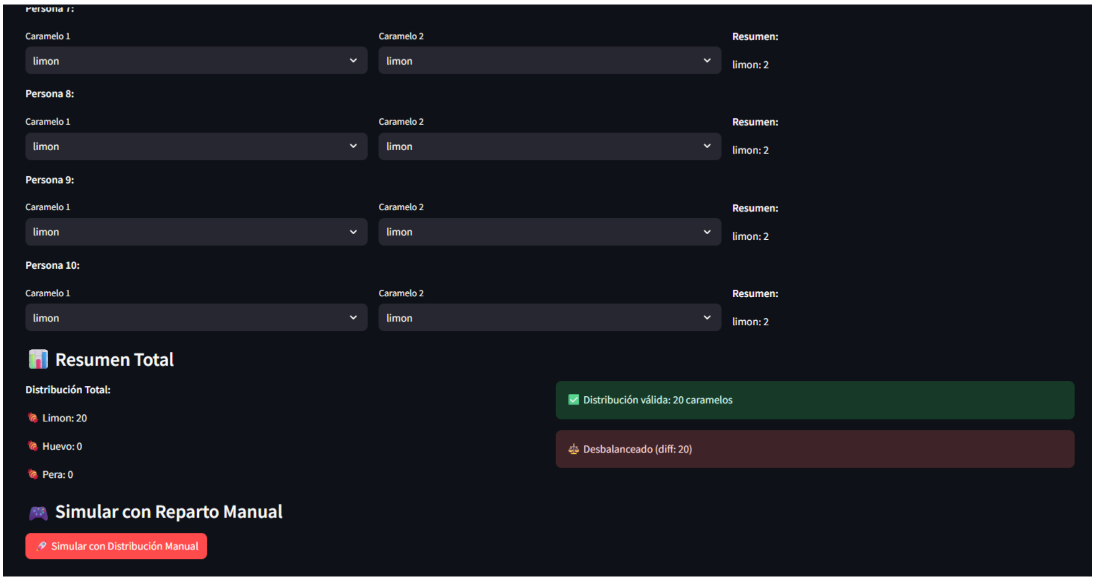
Persona 3:

Caramelo 1: limon | Caramelo 2: limon | **Resumen:** limon: 2

Persona 4:

Caramelo 1: limon | Caramelo 2: limon | **Resumen:** limon: 2

Figura 3: Distribución manual de caramelos con selectores por persona.



Persona 7:

Caramelo 1: limon | Caramelo 2: limon | Resumen: limon: 2

Persona 8:

Caramelo 1: limon | Caramelo 2: limon | Resumen: limon: 2

Persona 9:

Caramelo 1: limon | Caramelo 2: limon | Resumen: limon: 2

Persona 10:

Caramelo 1: limon | Caramelo 2: limon | Resumen: limon: 2

📊 Resumen Total

Distribución Total:

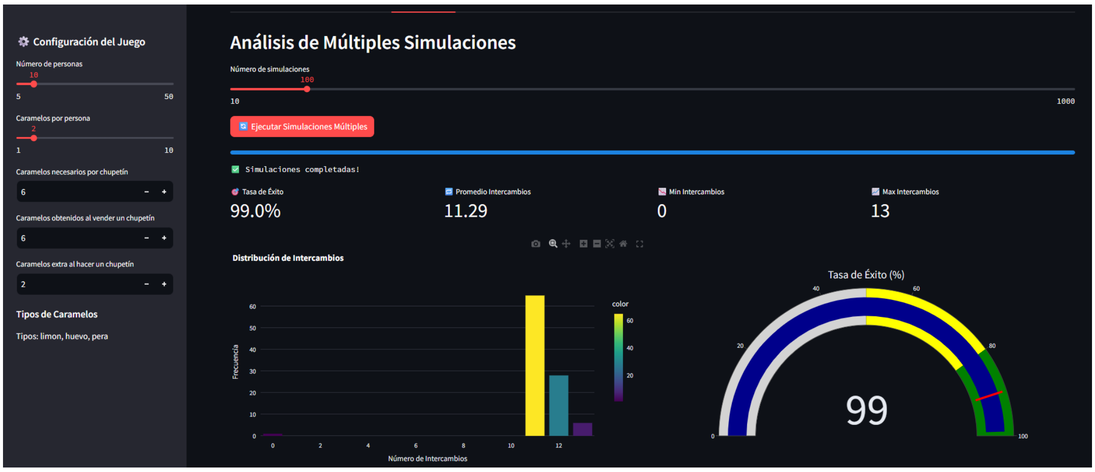
- Limon: 20
- Huevo: 0
- Pera: 0

✅ Distribución válida: 20 caramelos

⚖️ Desbalanceado (diff: 20)

🎮 Simular con Reparto Manual

Figura 4: Análisis múltiple con resultados estadísticos y métricas clave.



Configuración del Juego

Número de personas: 10 (5–50)

Caramelos por persona: 2 (1–10)

Caramelos necesarios por chupetín: 6

Caramelos obtenidos al vender un chupetín: 6

Caramelos extra al hacer un chupetín: 2

Tipos de Caramelos

Tipos: limon, huevo, pera

Análisis de Múltiples Simulaciones

Número de simulaciones: 100 (10–1000)

✅ Simulaciones completadas!

Tasa de Éxito: 99.0%

Promedio Intercambios: 11.29

Min Intercambios: 0

Max Intercambios: 13

Figura 5: Visualización de gráficos con Plotly: intercambios y tasa de éxito.