**Monty Hall Cuántico**

**Carlota Fernández del Riego**

**Hugo Iglesias Pombo**

**Contenido**

[1 Introducción 1](#_heading=h.g5x5e1bafc1y)

[1.1 Descripción 1](#_heading=h.by5m1a740dhk)

[1.2 Planteamiento 1](#_heading=h.htho4ov6y5pv)

[1.3 Diferenciación 2](#_heading=h.g2sdahsv273g)

[2 Implementación 2](#_heading=h.x61btvgre2u7)

[2.1 Modificaciones 2](#_heading=h.fs98yl4atsh6)

[2.2 Resultados 2](#_heading=h.9t1mqi5xgymd)

[3 Conclusión 2](#_heading=h.eae6ewp4y86d)

# Introducción

## Descripción

El presente proyecto implementa el problema de Monty Hall utilizando un modelo completamente cuántico. Todos los componentes del juego se modelan como operaciones cuánticas unitarias, sin intervención de lógica clásica en el flujo del juego.

* **Escenario del problema:**

Un concursante se enfrenta a tres puertas. Detrás de una hay un premio; las otras dos están vacías. El flujo del juego es:

1. El concursante elige una puerta inicial
2. El presentador (Monty Hall) abre una puerta vacía que no es la elegida
3. El concursante decide: ¿mantiene su elección o cambia a la puerta restante?

* **Objetivo del proyecto:**

Implementar el problema de Monty Hall usando exclusivamente operadores cuánticos unitarios, demostrando que la ventaja de cambiar de puerta surge de la estructura del operador de Monty.

## Planteamiento

#### Arquitectura de Qubits

El circuito utiliza 6 qubits organizados en tres registros:

|  |  |  |
| --- | --- | --- |
| Qubits | Registro | Función |
| q0, q1 | Premio | Codifica la puerta donde está el premio |
| q2, q3 | Jugador | Codifica la elección del jugador |
| q4, q5 | Monty | Codifica la puerta revelada por Monty |

#### 

#### Codificación de Puertas

|  |  |
| --- | --- |
| Estado cuántico | Puerta |
| |00⟩ | Puerta 0 |
| |01⟩ | Puerta 1 |
| |10⟩ | Puerta 2 |
| |11⟩ | No utilizado |

#### 

#### Operadores Cuánticos

#### El proyecto implementa tres operadores cuánticos:

1. **Preparación del premio:** Circuito que genera la superposición 1/√3(|00⟩ + |01⟩ + |10⟩)
2. **Operador de Monty (U\_monty):** Matriz unitaria de 64×64 que implementa las reglas de revelación
3. **Operador de Cambiar (U\_cambiar):** Matriz unitaria de 16×16 que implementa la estrategia de cambio

## Diferenciación

La implementación cuántica se diferencia de la clásica en varios aspectos fundamentales:

|  |  |  |
| --- | --- | --- |
| Aspecto | Implementación Clásica | Implementación Cuántica |
| Estado del premio | Valor definido oculto | Superposición cuántica |
| Operador de Monty | Algoritmo imperativo | Matriz unitaria 64×64 |
| Estrategia cambiar | Condicional clásico | Matriz unitaria 16×16 |
| Cálculo de probabilidades | Simulación Monte Carlo | Análisis de amplitudes |

# Implementación

## Modificaciones

### Preparación del Estado Inicial

El premio se prepara en superposición uniforme de las tres puertas válidas:

* **Circuito de preparación:**

q1: ─[RY(θ)]──●──  
 │  
 q0: ─────────[CH]── (Hadamard controlado)  
   
 donde θ = 2·arccos(√(2/3)) ≈ 1.2310 rad

* **Derivación matemática:**
  1. RY(θ)|0⟩ = √(2/3)|0⟩ + √(1/3)|1⟩ en q1
  2. CH aplica Hadamard a q0 solo cuando q1=0
  3. Estado resultante: 1/√3(|00⟩ + |01⟩ + |10⟩)

El estado inicial completo del sistema (6 qubits) es:

donde el formato es |monty, jugador, premio⟩ y jugador=0, monty=0 inicialmente.

### Operador de Monty (Matriz Unitaria 64×64)

El operador de Monty se implementa como una transformación unitaria que actúa sobre los 6 qubits del sistema. La matriz cumple las reglas del juego:

* Monty **nunca** revela la puerta del premio
* Monty **nunca** revela la puerta del jugador
* Si hay dos opciones válidas, crea **superposición cuántica**
* **Tabla de verdad (jugador = 0):**

|  |  |
| --- | --- |
| Premio | Monty revela |
| 0 | 1/√2(|01⟩ + |10⟩) [superposición] |
| 1 | |10⟩ [puerta 2] |
| 2 | |01⟩ [puerta 1] |

* **Implementación en Python:**

def crear\_operador\_monty() -> np.ndarray:  
 dim = 64 # 2^6 qubits  
 U = np.zeros((dim, dim), dtype=complex)  
   
 for premio in range(4):  
 for jugador in range(4):  
 # Determinar opciones válidas para Monty  
 opciones = [p for p in [0,1,2] if p != premio and p != jugador]  
   
 if len(opciones) == 1:  
 # Determinista  
 U[encode(premio, jugador, opciones[0]),   
 encode(premio, jugador, 0)] = 1.0  
 else:  
 # Superposición uniforme  
 for opt in opciones:  
 U[encode(premio, jugador, opt),  
 encode(premio, jugador, 0)] = 1/sqrt(2)  
   
 return U

### Operador de Cambiar (Matriz Unitaria 16×16)

El operador de cambiar modifica la elección del jugador basándose en la puerta revelada por Monty:

**Regla:** Nueva elección = la puerta que no es la original ni la revelada

|  |  |  |
| --- | --- | --- |
| Jugador original | Monty reveló | Nueva elección |
| 0 | 1 | 2 |
| 0 | 2 | 1 |
| 1 | 0 | 2 |
| 1 | 2 | 0 |
| 2 | 0 | 1 |
| 2 | 1 | 0 |

* **Implementación:**

def crear\_operador\_cambiar() -> np.ndarray:  
 dim = 16 # 2^4 qubits (jugador + monty)  
 U = np.zeros((dim, dim), dtype=complex)  
   
 for jugador in range(4):  
 for monty in range(4):  
 if jugador < 3 and monty < 3 and jugador != monty:  
 nueva = [p for p in [0,1,2] if p != jugador and p != monty][0]  
 U[encode(nueva, monty), encode(jugador, monty)] = 1.0  
 else:  
 U[encode(jugador, monty), encode(jugador, monty)] = 1.0  
   
 return U

## Resultados

#### Estado después del operador de Monty (estrategia MANTENER)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Estado | Premio | Jugador | Monty | Amplitud | Probabilidad |
| |010000⟩ | 0 | 0 | 1 | +0.4082 | 16.67% ✓ |
| |010010⟩ | 2 | 0 | 1 | +0.5774 | 33.33% |
| |100000⟩ | 0 | 0 | 2 | +0.4082 | 16.67% ✓ |
| |100001⟩ | 1 | 0 | 2 | +0.5774 | 33.33% |

**P(ganar) = 33.33%** (suma de probabilidades donde premio = jugador)

#### 

#### Estado después del operador CAMBIAR

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Estado | Premio | Jugador | Monty | Amplitud | Probabilidad |
| |011000⟩ | 0 | 2 | 1 | +0.4082 | 16.67% |
| |011010⟩ | 2 | 2 | 1 | +0.5774 | 33.33% ✓ |
| |100100⟩ | 0 | 1 | 2 | +0.4082 | 16.67% |
| |100101⟩ | 1 | 1 | 2 | +0.5774 | 33.33% ✓ |

**P(ganar) = 66.67%** (suma de probabilidades donde premio = jugador)

#### Tabla Resumen

|  |  |  |
| --- | --- | --- |
| Estrategia | P(ganar) calculada | P(ganar) teórica |
| **MANTENER** | 33.33% | 33.33% |
| **CAMBIAR** | 66.67% | 66.67% |

# Conclusión

### Implementación Cuántica Completa

Este proyecto demuestra que el problema de Monty Hall puede implementarse completamente usando operadores cuánticos unitarios:

1. **Estado inicial:** El premio se prepara en superposición cuántica 1/√3(|0⟩ + |1⟩ + |2⟩)
2. **Operador de Monty:** Una matriz unitaria de 64×64 que aplica las reglas del juego cuánticamente, creando superposición cuando hay múltiples opciones válidas
3. **Operador de Cambiar:** Una matriz unitaria de 16×16 que modifica la elección del jugador de forma reversible
4. **Análisis de amplitudes:** Las probabilidades se calculan sumando |amplitud|² de los estados donde premio = jugador

### 

### Resultado Principal

La simulación cuántica confirma los resultados teóricos:

* **CAMBIAR** da probabilidad de ganar = **66.67%**
* **MANTENER** da probabilidad de ganar = **33.33%**

La ventaja de cambiar emerge de la estructura del operador de Monty: al revelar una puerta vacía, Monty genera información que se codifica en los qubits q4, q5. El operador de cambiar utiliza esta información para modificar la elección del jugador, concentrando la probabilidad de victoria en la puerta correcta.

### Implicación Teórica

El problema de Monty Hall ilustra un principio fundamental: la información generada por una observación (la revelación de Monty) puede utilizarse para mejorar decisiones posteriores. En el modelo cuántico, esta información se representa mediante la evolución de las amplitudes de probabilidad a través de operadores unitarios.

* **Tecnologías Utilizadas:** - Python 3.x - PyQuil (Framework para computación cuántica) - WavefunctionSimulator - NumPy (álgebra lineal)