Relatório Parcial do projeto Quantum Oracles - Como transformar problemas clássicos em quânticos

Por: Alexandre Silva

Orientador: Luis Hilário Tobler Garcia

## 1. Tarefas realizadas

*Todo material usado pode ser encontrado no meu perfil do* [*github*](https://github.com/Dpbm/scientific-initiation-1-quantum-oracles)*.*

### 1.1. desenvolvimento

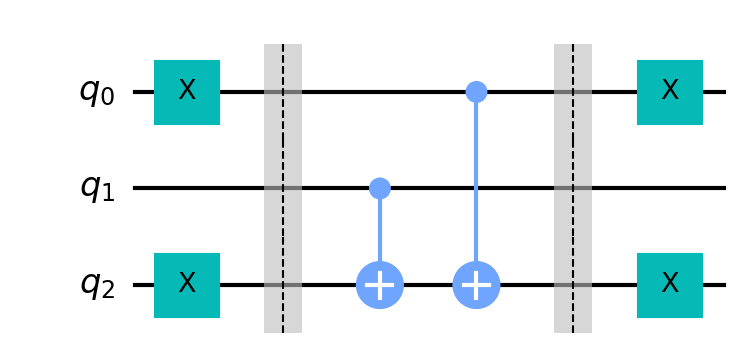
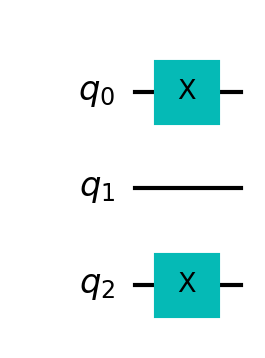
A partir da leitura, foram iniciados os testes, para validar tudo que estava sendo visto.

De início, foram testados algoritmos/circuitos, dos quais podem ser utilizados em diversas situações, os quais são:

#### 1.1.1. Boolean Oracle

O *Boolean Oracle* é um Oracle que, internamente, implementa uma função booleana, mapeando os valores da entrada para saída.

### *Exemplo - oracle booleano*



#### 1.1.2. Phase Oracle

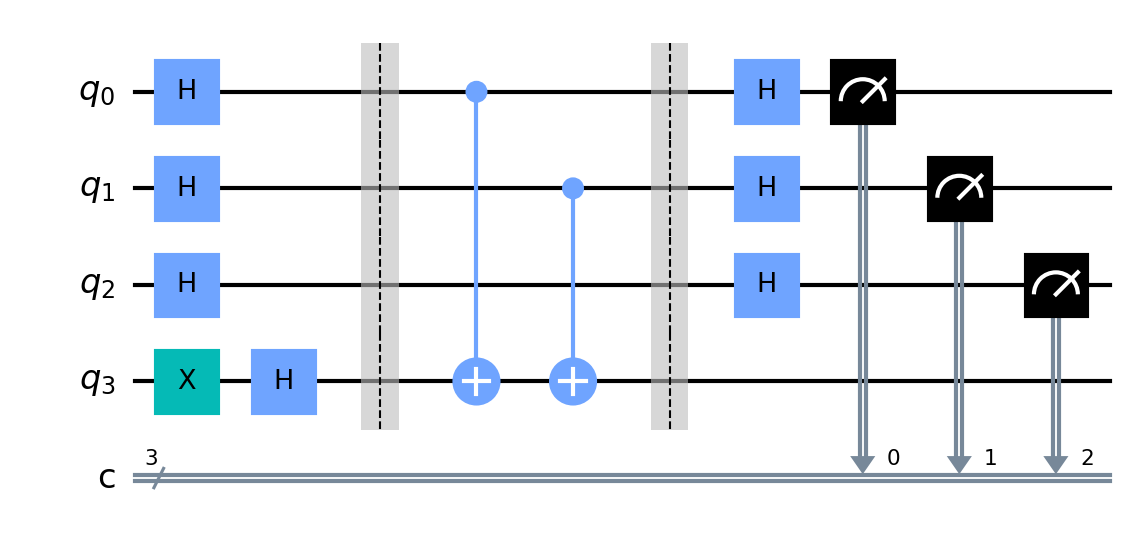
O *Phase Oracle* é um dispositivo que utiliza de fases para codificar valores dentro do *oracle*, através do efeito de *phase kickback* de *Controlled Gates*, ou ainda pelo uso de *gates* como *Z, CZ e CCZ*.

Sua utilidade é evidente quando usado em algoritmos de busca, como o de *Grover*, marcando os valores que são interessantes para a pesquisa, ou ainda quando precisamos de sobrepor valores.

Quando utilizado o método de *phase kickback*, seu circuito acaba precisando de um *qubit* *auxiliar*, responsável por realizar o efeito. Mapeando então os valores da entrada, para os mesmo valores da entrada mais o resultado da função (*phase*).

Com isso, após utilizar o phase oracle e removermos a superposição dos valores (*gate H*), temos então o valor do qual foi inserido dentro do oracle.

### *Exemplo - phase oracle com phase kickback*



Outra maneira mais simples de implementar é utilizando os gates que aplicam fases, como o Z, CZ, CCZ, etc. Para isso, basta definir quais os valores booleanos que queremos, inverter as posições que estão com o valor 0, utilizando o *gate X*.

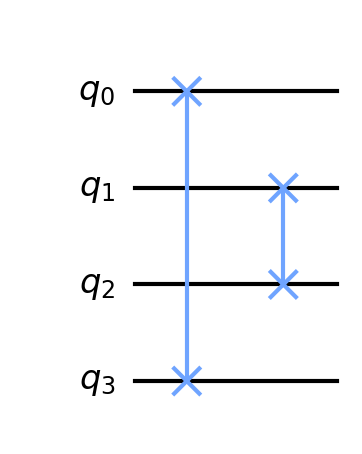
### *Exemplo - phase oracle com o CCZ gate, encodando o valor 011*

### 

#### 1.1.3. Minimal Oracle

O *Minimal Oracle* é utilizado quando a função em questão é unitária e reversível, o que significa que, se fizermos uma matriz representando os mapeamentos de valores dela, e a raiz da soma do quadrado das amplitudes de cada estado (valores das colunas) é igual a 1 e é possível distinguir os mapeamentos para cada valor. Podendo também ser visto como um *phase oracle* ou *boolean oracle* dependendo de sua implementação.

### Exemplo - minimal oracle com SWAP

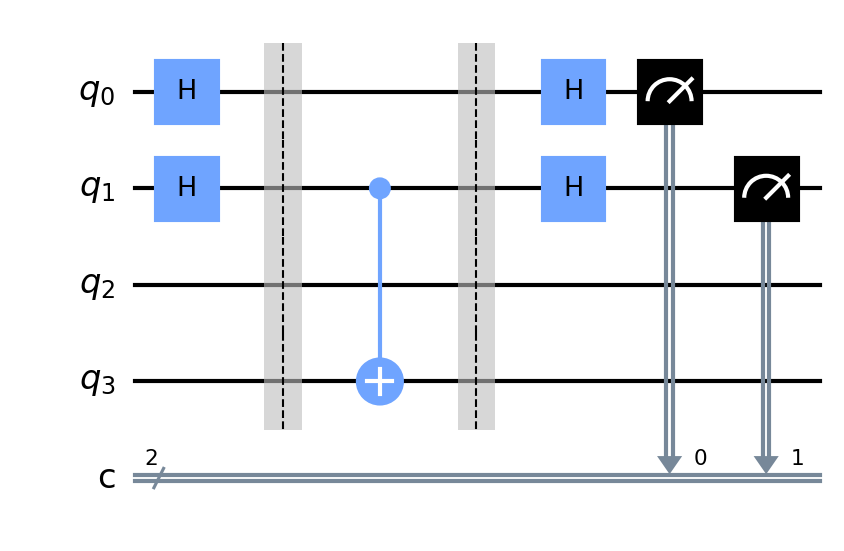


#### 1.1.4. Simon’s Oracle and Algorithm

Daniel R. Simon quando apresentou seu algoritmo quântico nos mostrou também um *oracle* interessante. Seu algoritmo é utilizado para encontrar o período de funções, ou seja, dado uma função *booleana*, tentamos encontrar quando , onde *x* é a entrada e *y* é a saída de *f(x)*.

Para montar esse oracle é necessário *n qubits* de entrada mais *n qubits* auxiliares, esses auxiliares serão usados para criar *entanglement* com as entradas.

### *Exemplo - algoritmo de Simon*



Repare que o *oracle* em si é apenas a parte do meio (*CX gate*), do qual age como um segredo escondido, assim como na criação de chaves criptográficas.

Sendo assim, ao adicionarmos o *Oracle*, estamos fazendo operações entre a matriz de entrada com o segredo, sendo esse segredo do exemplo a *string* 10.

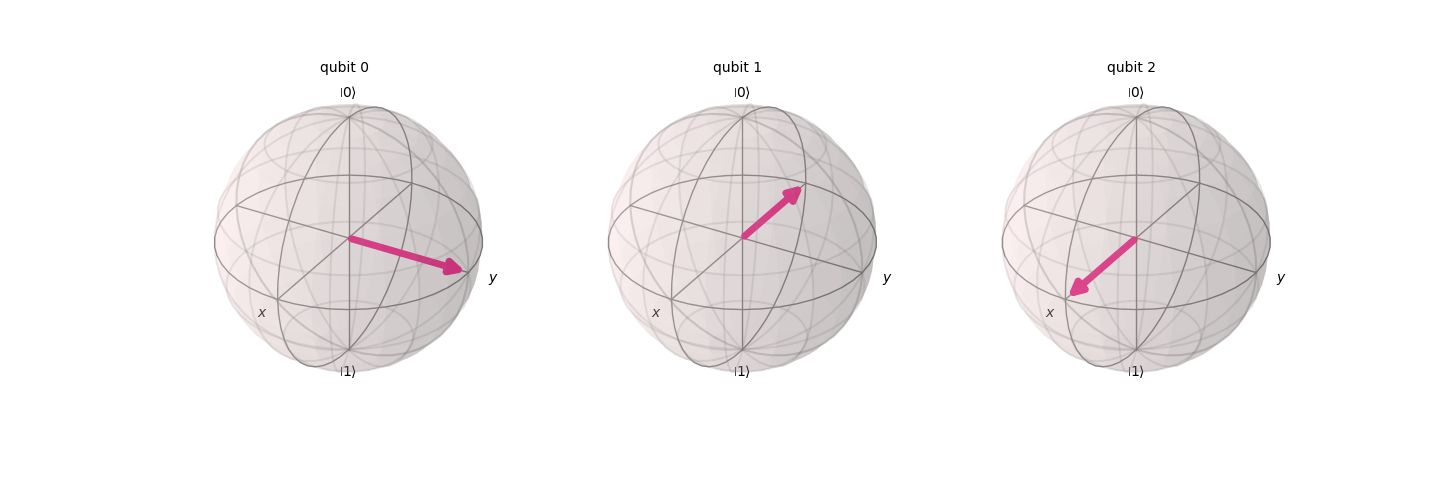
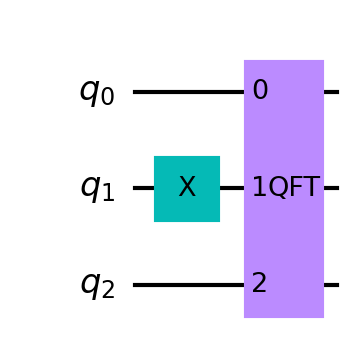
#### 1.1.5. Quantum Fourier Transformation (QFT)

A transformada quântica de Fourier é uma dos algoritmos mais utilizados em circuitos, junto com sua função inversa. Podendo também ser visto como um Oracle.

Seu uso se dá através da marginalização do eixo Z, mapeando os valores então para os eixos X e Y, sendo interessante para algoritmos que manipulam fases.

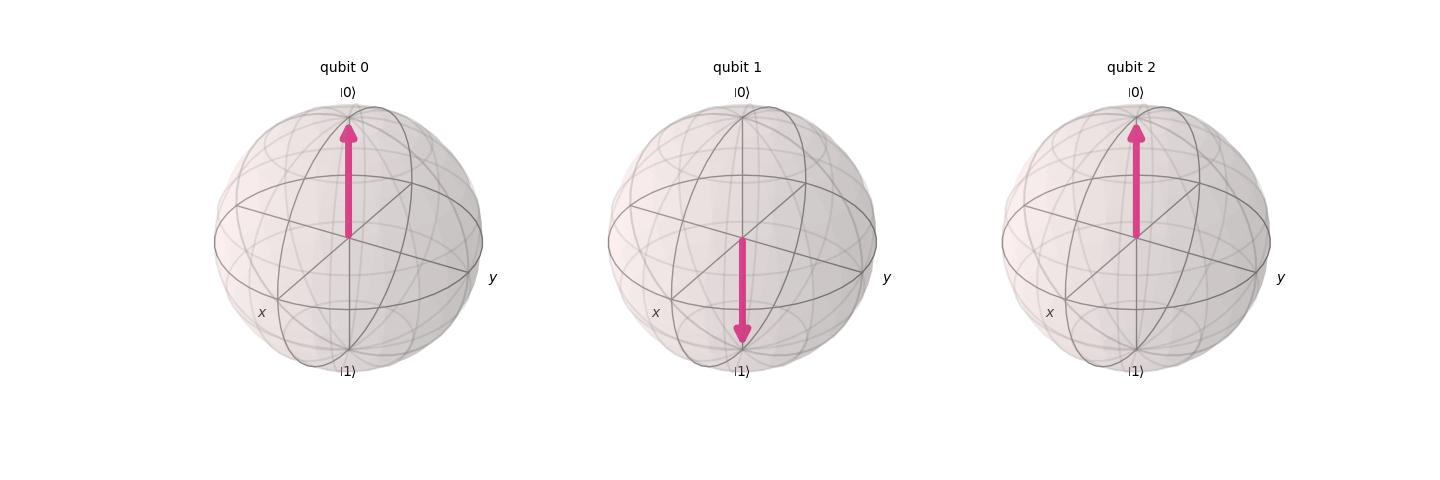
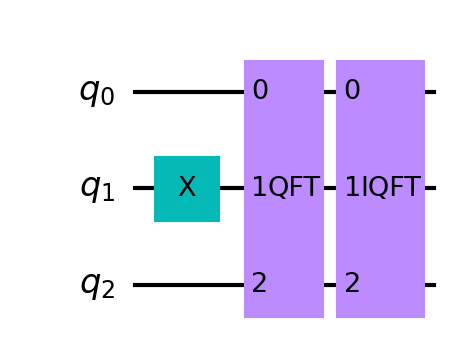
### 

### *Exemplo - QFT com 3 qubits*



Assim, quando queremos voltar para a base computacional (0 e 1), podemos utilizar o inverso para tirar do eixo de *Fourier* para o eixo Z.

### *Exemplo - uso da função inversa da QFT*



#### 1.1.6. Grover’s Algorithm

A partir do uso do *phase oracle*, podemos marcar os valores que queremos encontrar em um banco de dados desorganizado, e ao usá-lo em conjunto com o algoritmo de *Grover*, podemos encontrar os valores que desejamos em .

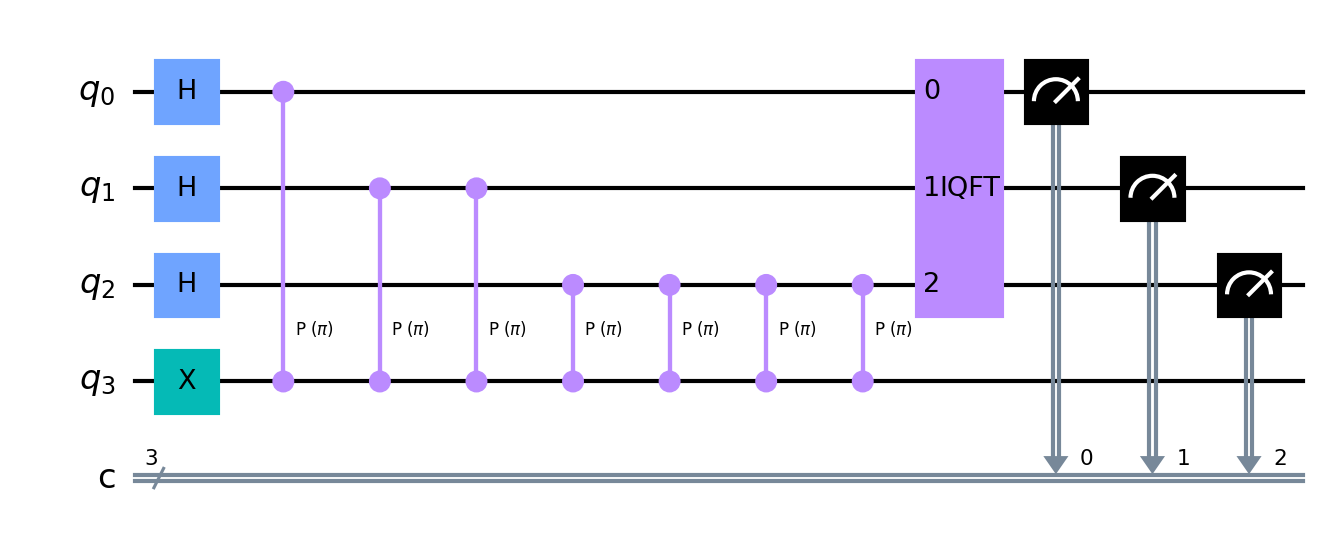
### *Exemplo - Grover algorithm*

Para isso, setamos um estado de superposição igualitária para todos os possíveis valores (os tres *H* no começo), passamos esses valores pelo *Oracle* que marcará com um sinal negativo (*phase*=-1) nos valores que queremos encontrar e então aplicamos um *diffuser*, do qual é responsável por aumentar a probabilidade de encontrar os valores que queremos.

#### 1.1.7. Quantum Phase Estimation(QPE)

Usando o inverso do *QFT*, podemos incluir dentro de um circuito uma certa fase, e através do *QPE*, podemos aproximar o ângulo na *bloch sphere* que causa essa fase.

### *Exemplo - QPE*

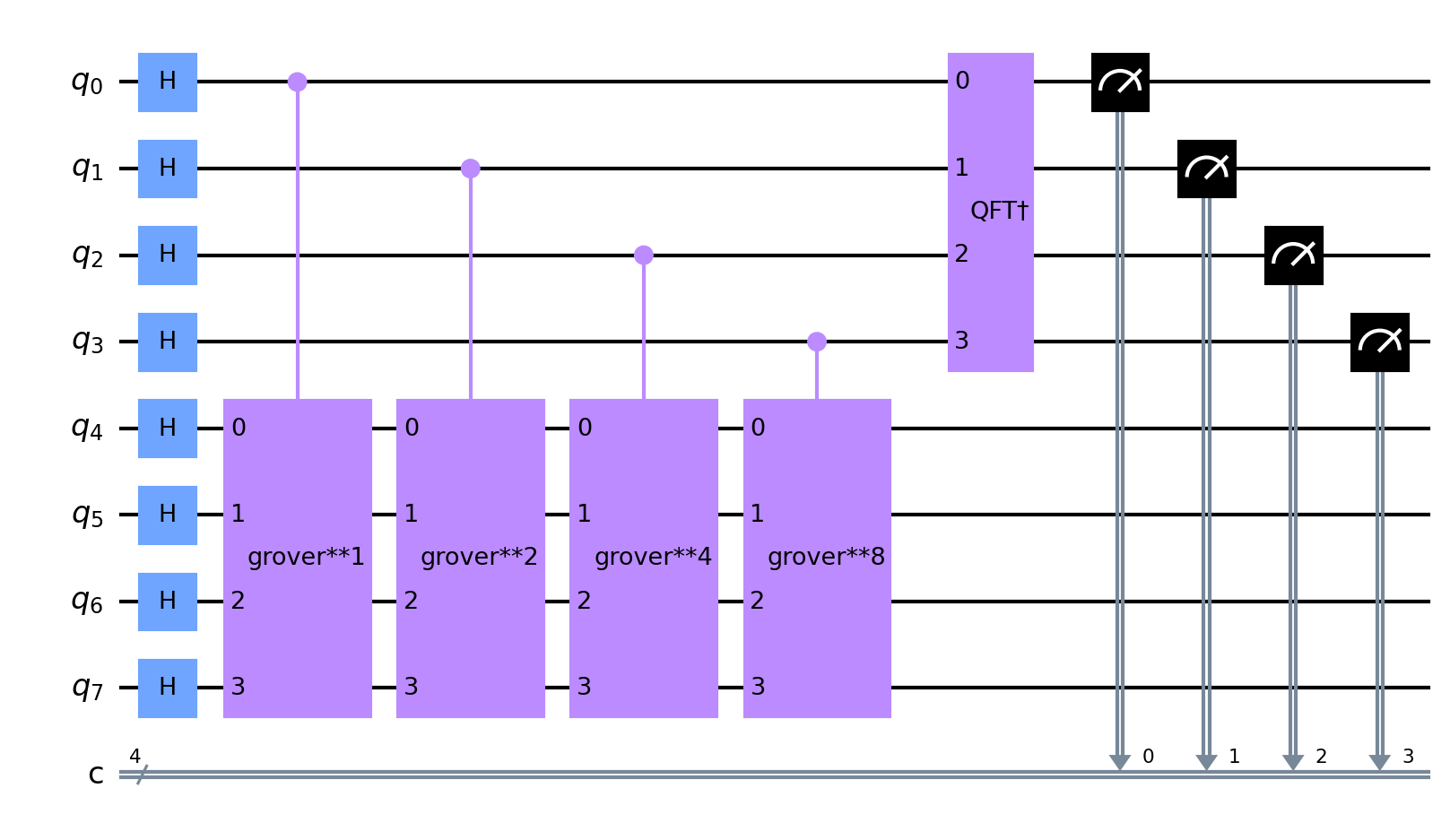


Sendo um algoritmo útil para a extração de informações, também usado por algoritmos como o de *Shor*.

#### 1.1.8. Quantum Counting

A partir da ideia do *QPE* e usando o inverso do *QFT* + os operadores de *Grover*, podemos fazer um algoritmo para contar valores dentro de um *set*.

### *Exemplo - Quantum Counting*



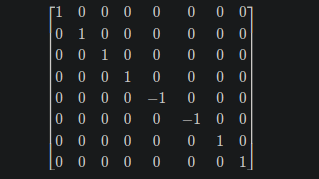
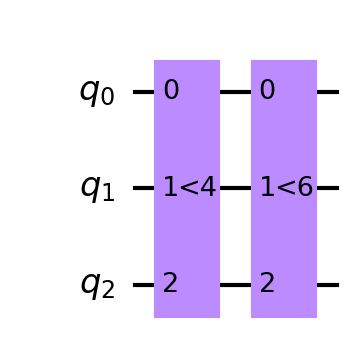
Para esse circuito precisamos de n qubits + n qubits auxiliares, do qual em cima temos o set com todos os valores possíveis e em baixo teremos a contagem em si.

Esse circuito também pode ser usado para diversos casos que é necessário contar. Contudo ele pode ser custoso em processamento, uma vez que precisamos executar o algoritmo de Grover muitas vezes, crescendo a uma taxa de iterações a cada *qubit* adicionado.

#### 1.1.9. Range de valores

Utilizando os *phase oracles*, podemos encodar certos *range* de valores dentro de cada um deles e, a partir de sua sobreposição podemos encontrar a diferença entre conjunto de valores.

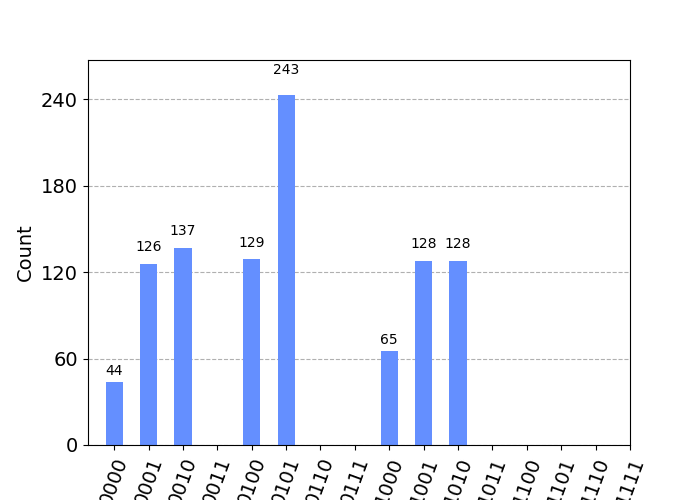
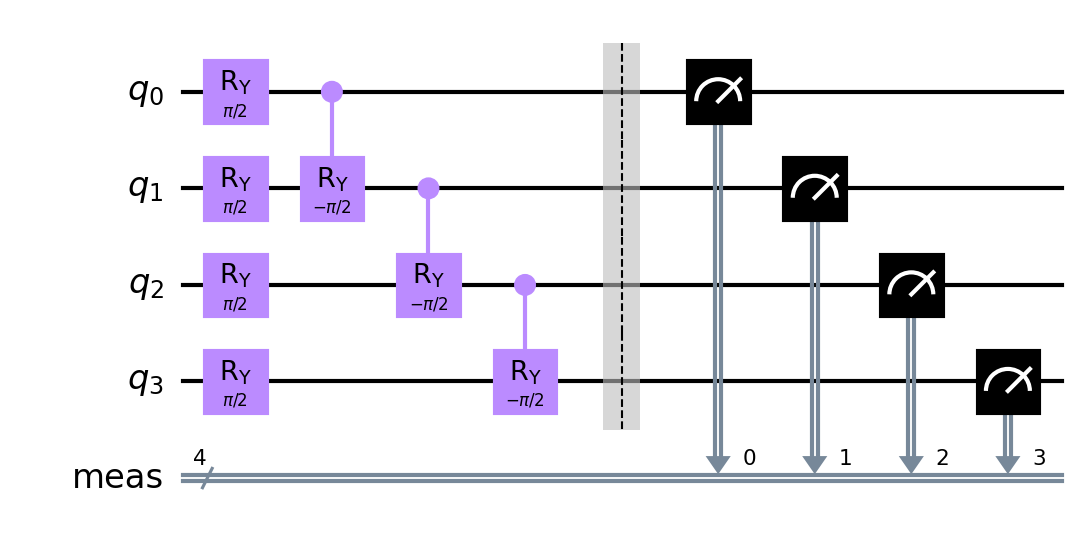
### *Exemplo - Exemplo Range de valores ()*



#### 1.1.10. Fibonacci

Por fim, o último circuito testado foi o de *fibonacci*. Esse circuito toma proveito da sequência de valores binários sem números 1s consecutivos (como 0110, 1100 e 1111). A partir disso, basta utilizar o algoritmo, rodá-lo n vezes, e contar a quantidade de valores que apareceram, havendo então uma aproximação para o valor de *n* de *fibonacci* utilizando *n qubits*, sendo *n* iniciado em 2 (F(2) = 3).

### *Exemplo - Exemplo Fibonacci (F(4) = 8).*



Podemos então utilizar esse algoritmo dentro de um oracle, e dividir os números que queremos encontrar.

Contudo, esse circuito acaba não sendo melhor do que a versão clássica, já que para termos o valor correto precisamos de muitas medições e para valores maiores precisamos de muitos *qubits*, o que acaba prejudicando seu uso.

### 1.2. Implementação dos algoritmos para os problemas

Com esses algoritmos em mãos, podemos começar a tentar resolver alguns problemas e verificar a solução quântica.

Durante as leituras e os testes, algumas ideias vieram à minha cabeça, as quais foram:

* Explorador de arquivos
* Conversor de milhas para Km
* Torres de Hanoi
* Buckshot Roulette(quantum-game)
* QRAM

Sendo que o último ainda não foi implementado estando em fase de estudos.

#### 1.2.1. Conversor de milhas para Km

Como um primeiro experimento, foram feitos alguns testes para a transformação de milhas para Km.

Uma das maneiras de fazer isso é tentar aproximar usando a sequência de fibonacci. Como já temos o algoritmo para fazer isso, só precisamos encontrar os relativos para os valores de milhas e km.

## 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

Para transformar milhas em Km, basta pegar um número da sequência (valor em milhas) e o seu sucessor será o relativo em Km.

Contudo, utilizando essa abordagem estamos limitados aos valores da sequência, para resolver isso, podemos quebrar o valor de milhas em pedaços que estão na sequência, por exemplo:

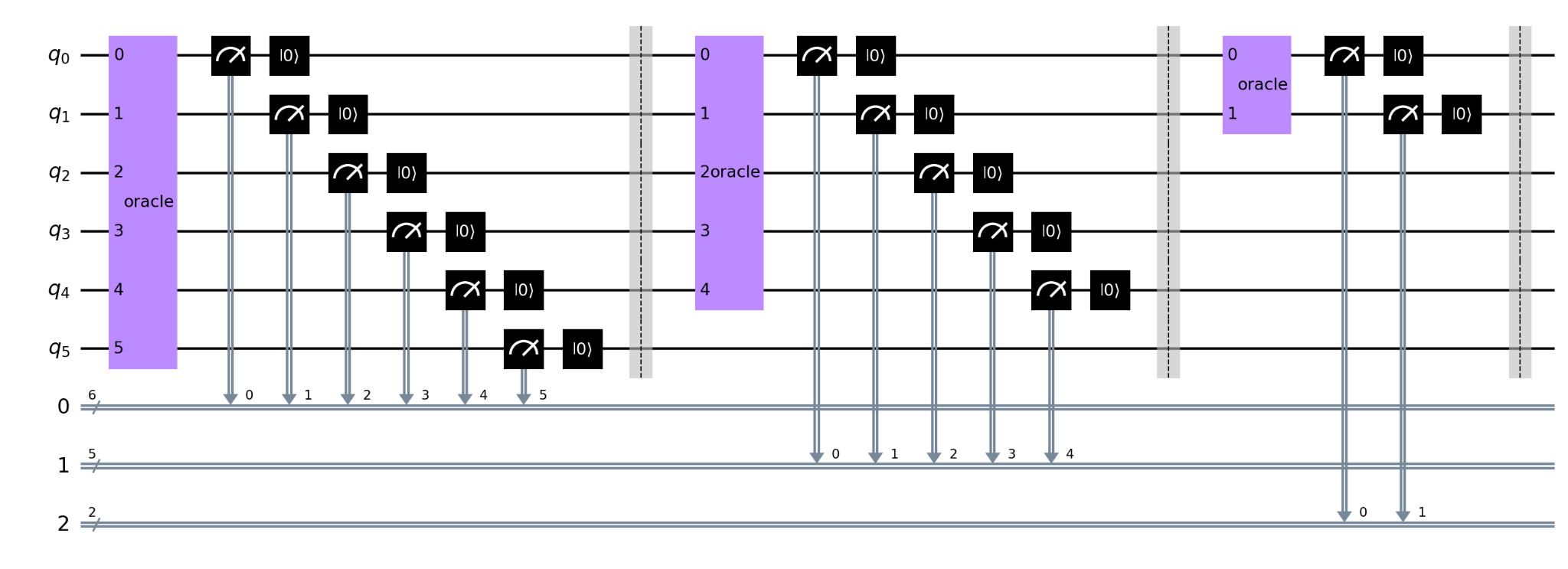
## 22 milhas → 21 milhas + 1 milha → 34 km + 1 km ~ 35 km

### Transformação exata de milhas para Km



Com isso em mente, podemos começar a fazer a implementação. No entanto, para o circuito precisamos saber qual a quantidade de *qubits* necessário para rodar o algoritmo, além disso precisamos também saber qual o número de *qubits* para cada pedaço do número final. Sendo assim, foi necessário fazer um algoritmo clássico para pré-processar a entrada e então quebrar o valor em pedaços.

### Algoritmo criado para converter 100 milhas para Km

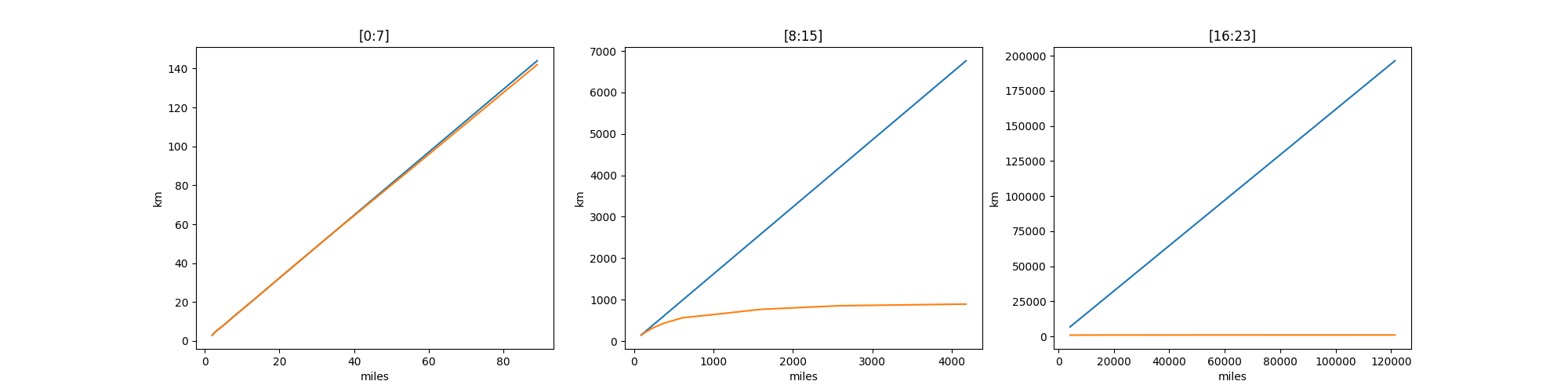


Após executar o algoritmo, é necessário pegar o resultados e multiplicar pela quantidade de vezes que cada um deveria ser adicionado ao circuito, além de adicionar também o resto dos valores 1 e 2 que não podiam ser calculados usando esse algoritmo, e então chegamos ao valor muito próximo ao correto no final.

Há a possibilidade de colocar os *oracles* múltiplas vezes, mas isso acaba crescendo muito o circuito e há uma maior taxa de erros.

Com esse teste realizado, é possível ver que esse método possui vários problemas, como a indisponibilidade do número 1 e 2, a dificuldade em ler o resultado (uma vez que é baseado em contar a quantidade de valores diferentes após n medições) e também a imprecisão, sendo necessários muitos rodadas de execução para ter melhores resultados.

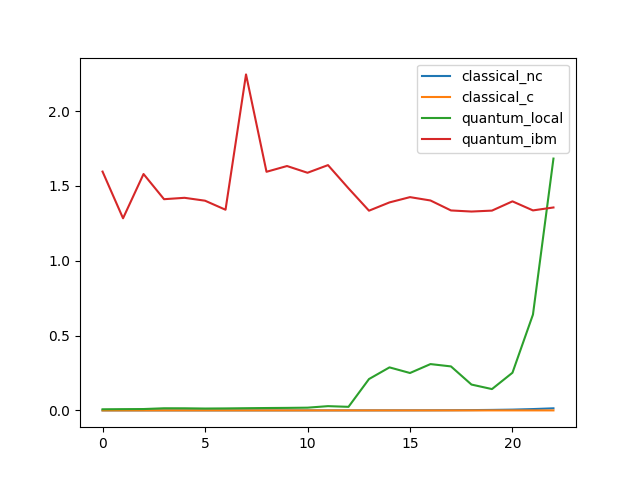
### Discrepância entre os valores resultantes do algoritmo clássico e quântico



Pela imagem acima é possível ver que para valores pequenos (gráfico mais à esquerda) os valores clássicos e quânticos se aproximam muito, mas a partir de quantidades maiores há uma diferença considerável.

Sendo assim, além de pré-processamento, esse algoritmo também precisa de pós-processamento utilizando computadores clássicos, além de levar muito mais tempo para executar do que utilizando um computador convencional com a técnica de memoization.

### Comparação - tempos de execução



*Obs: a execução em simuladores e os computadores da IBM acaba dependendo muito de valores externos, como uso de CPU antes e durante a execução, temperatura externa, error mitigation, etc.*

Sendo assim, esse método não se demonstra melhor do que as versões clássicas.

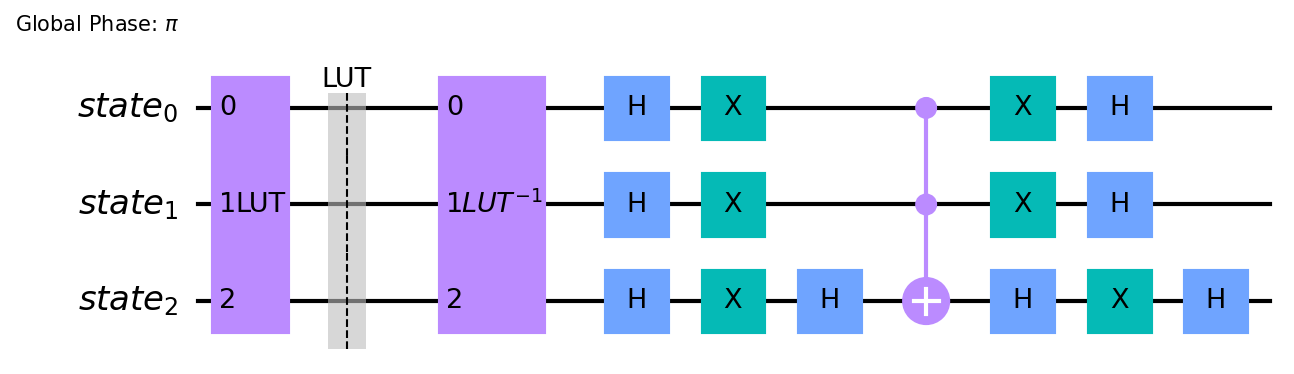
Contudo, ainda pode haver uma possibilidade de implementar essa solução usando encoding de matrizes e utilizando o dot product, operação da qual é embutida em circuitos quânticos. Ainda não consegui pensar em como encodar isso corretamente, mas já está em mente para passos futuros/pesquisas futuras.

#### 1.2.2. Explorador de arquivos

Utilizando da ideia do operador de Grover, foi implementada uma ideia de um possível gerenciador de arquivos quântico.

Aqui, o objetivo era conseguir encodar em “hashes” os nomes dos arquivos (como em uma look-up-table), e se aproveitando do algoritmo de busca encontrar esses arquivos.

### Algoritmo - Quantum File explorer



Nesse algoritmo, colocamos no primeiro bloco uma Look-up-table, com todos os arquivos existentes, encodados em formato de hash binário, e então marcados usando o phase oracle. Após colocar todos os possíveis arquivos em superposição, é usado a sobreposição de sets (como mostrado anteriormente), marcando apenas os valores requeridos pelo usuário.

Com a primeira etapa pronta, basta rodar o algoritmo de Groover e obter os arquivos esperados com a maior probabilidade possível.

### Comparação versão clássica e quântica

Mesmo o algoritmo de Groover sendo extremamente rápido comparado a busca descoordenada clássica, ainda assim poderiamos ter o maior benefício das Look-up-tables na versão clássica, uma vez que possui tempo constante , sendo apenas a melhor escolha quando a versão clássica possui tempo . Além de haver também a possibilidade de encontrar outro arquivo que não foi marcado devido a natureza quântica.

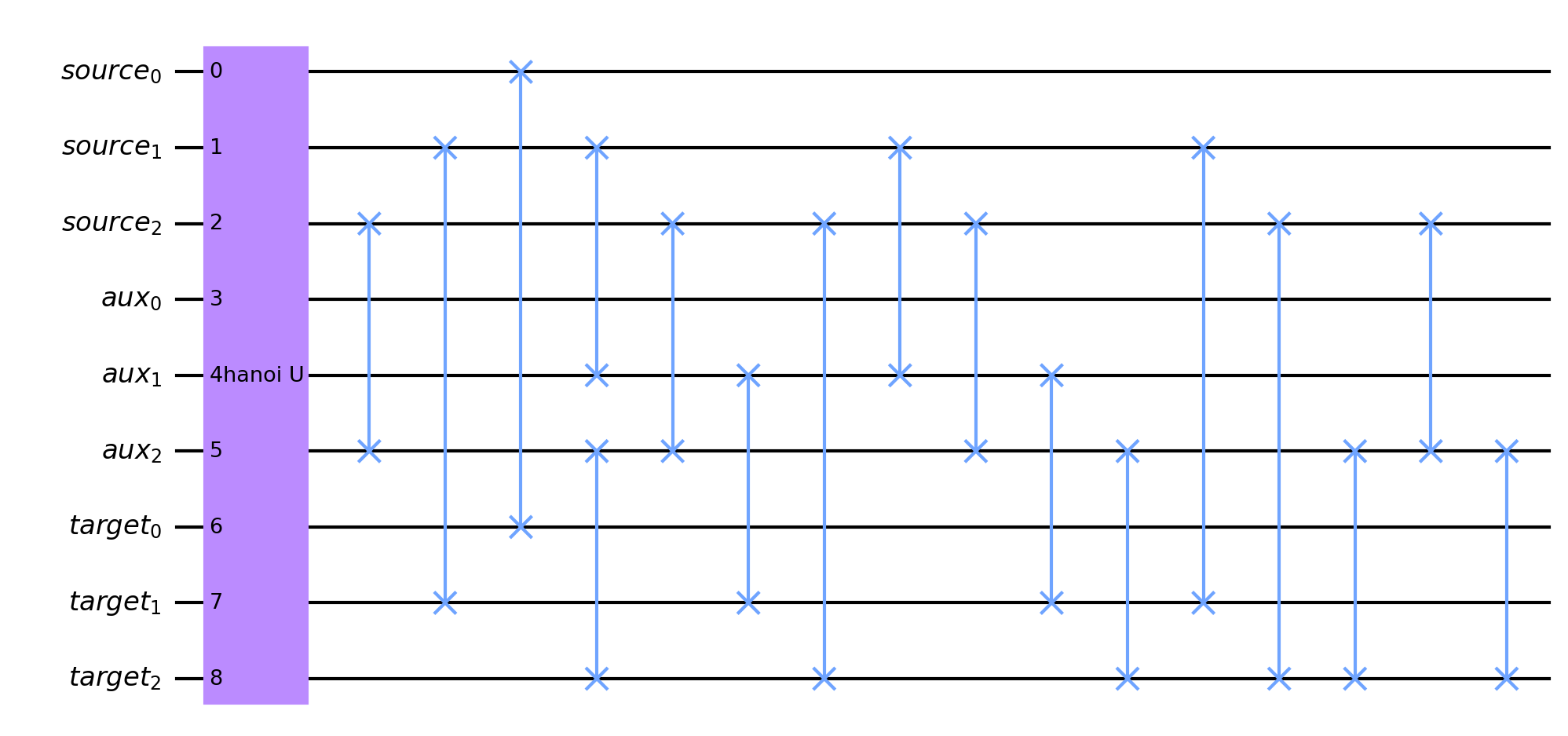
Além disso, por ser um algoritmo rodando em hardware quântico, este está suscetível a erros, sendo necessário a mitigação deles durante a execução.

#### 1.2.3. Torres de Hanoi

O terceiro projeto pensando aqui, foi as torres de Hanoi. Esse clássico problema de recursão foi também traduzido para uma versão quântica. Nesse caso, nenhum efeito quântico maior, como superposição, entanglement, etc. Foi explorado, aqui tentamos implementar as torres usando os quantum oracles e se baseando puramente na versão/resolução clássica.

Para esse problema, as torres foram embutidas como números binários dentro de um oracle. Com os dados preparados, foi feito um algoritmo clássico que traduz os movimentos que seriam feitos em um algoritmo recursivo clássico, para um circuito quântico usando os gates SWAP.

### Torre de hanoi com 4 discos



Com isso, é feita a manipulação bit a bit, e a matriz unitária encodada dentro do oracle é modificada a cada iteração.

Nesse caso, não há diferenças entre a versão clássica ou quântica, apenas a introdução de erros que deve ser levada em consideração aqui ao usar em um hardware quântico real.

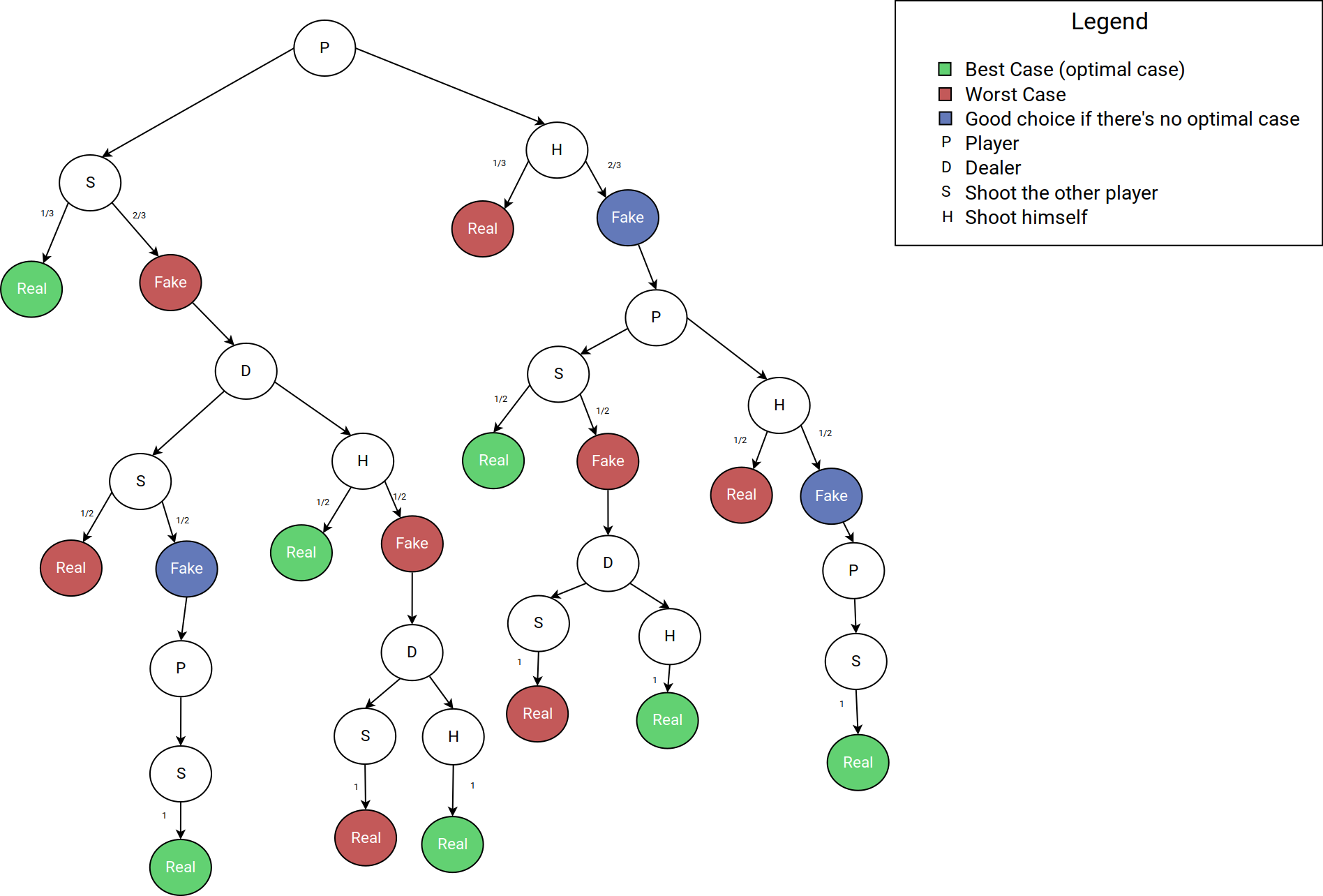
#### 1.2.4. Buckshot Roulette

Por fim, o último projeto implementado foi uma versão quântica do jogo Buckshot Roulette. Esse jogo, desenvolvido por [Mike Klubnika](https://mikeklubnika.itch.io/buckshot-roulette), cria uma nova versão da infame roleta russa. Neste, o jogador joga contra um demônio, tendo como objetivo ficar com mais vidas do que do que ele, caso o jogador perca todas as suas vidas o jogo reinicia.

Neste projeto, foi levado em conta apenas a primeira rodada do jogo, pois a medida que o jogo avança, power-ups são adicionados, o que acaba dificultando a análise que, nesse caso, não tinha como objetivo ser muito aprofundada no jogo, apenas como isso poderia ser traduzido para um circuito quântico.

Na primeira rodada, a arma possui 3 balas no total, sendo 2 falsas e 1 real. O jogador começa jogando, caso ele atire nele mesmo, se for real ele perde a rodada, se for falsa ele joga novamente; caso ele escolha atirar no dealer(demônio), se for real ele ganha a rodada, caso seja falsa o dealer joga. Com essa dinâmica em mente, o jogo foi modelado da seguinte forma:

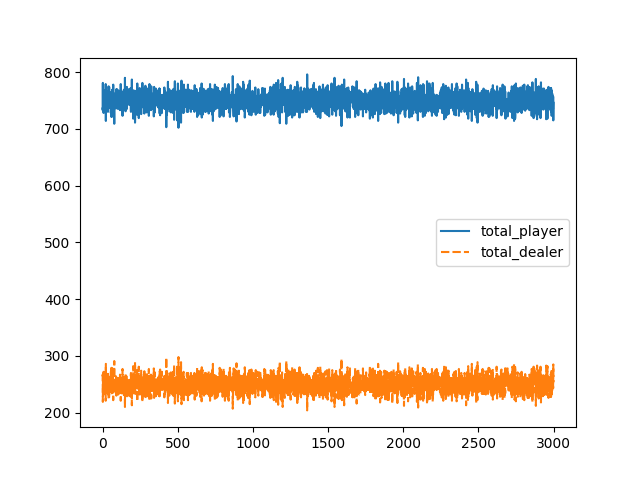
### Modelagem do round usando a estrutura de árvore



*Obs: o dealer, por padrão no jogo, tem 50% de chance de atirar ou não, sempre, não possuindo qualquer inteligência baseada em jogadas anteriores.*

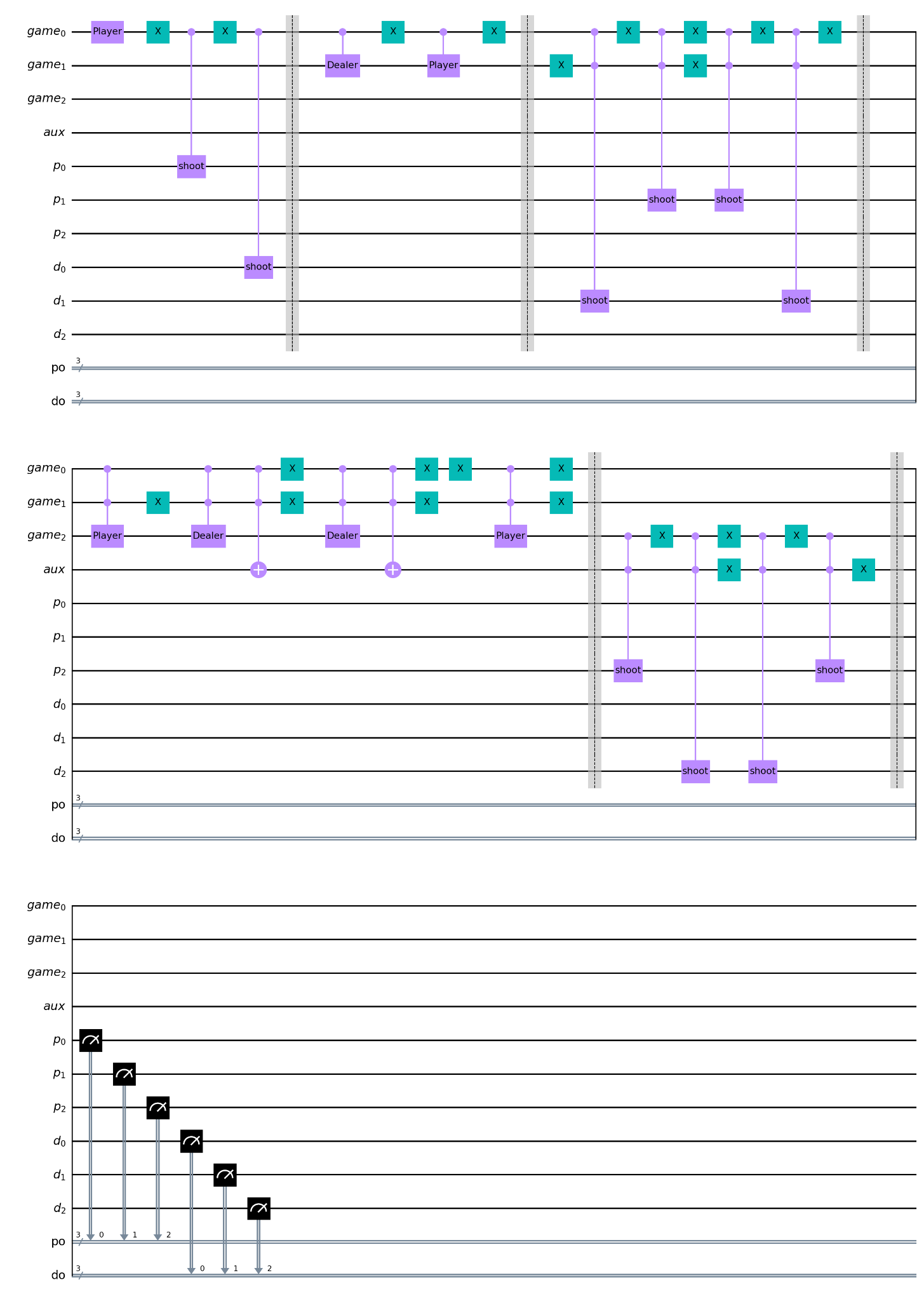
Com a modelagem feita, primeiro foi testado implementado uma simulação clássica seguindo a estratégia acima.

### Melhor resultado do round

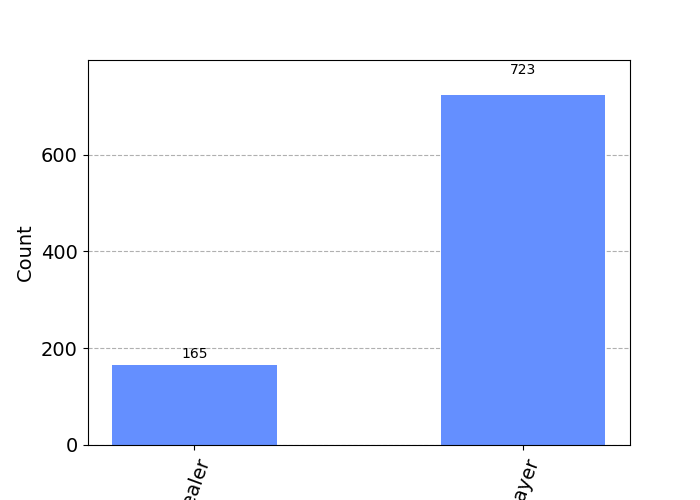


Seguindo a melhor estratégia (player começa atirando no dealer), o player ganha ~70% das vezes.

Traduzindo para a computação quântica, foi implementado um circuito que simula o round e foram adicionados um oracle para o dealer (estratégia dele), e um oracle para o player(estratégia do jogador), além disso, dentro do oracle do jogador, foi embutido um parâmetro para tentar maximizar os ganhos dele usando algoritmos variacionais

Circuito do jogo

Após simular múltiplas vezes o jogo, o resultado foi o seguinte:



Com isso, foi possível ver que o circuito obteve o mesmo desempenho da versão clássica. Sendo assim, é possível implementar algoritmos quânticos para game theory e obter o optimum value.

Novamente, esse teste se equipara à versão clássica, mas dessa vez tomamos proveito das rotações da Bloch Sphere para implementar as estratégias, o que em certos games pode ser um avanço perante a versão clássica.

## 2. Conclusão

A partir do que foi exposto, pode se ver que o projeto está sendo encaminhado para um passo além, já possuindo implementações concretas com dados, estando no estágio final do desenvolvimento.

### 2.1. Visão do projeto

Com o exposto, é possível ver que nem tudo possui ganhos ao usar algoritmo quânticos, mas ainda assim podemos tomar proveito de certos efeitos/técnicas e conseguir equiparar a versão clássica em áreas externas ao domínio da computação quântica.

### 2.2. Próximos passos

Para os próximos passos, é previsto finalizar o último projeto de QRAM e em seguida iniciar a escrita de um artigo científico mostrando os resultados obtidos nessa pesquisa.