Universidade Federal Fluminense

Instituto de Computação

Simulando Algoritmos Quânticos em um Computador Clássico

Igor Valente Blackman

NITERÓI - RJ

2017

Universidade Federal Fluminense
Instituto de Computação
Departamento de Ciência da Computação

Igor Valente Blackman

Simulando Algoritmos Quânticos em um Computador Clássico

NITERÓI - RJ
2017

IGOR VALENTE BLACKMAN

SIMULANDO ALGORITMOS QUÂNTICOS EM UM COMPUTADOR CLÁSSICO

Trabalho submetido ao Curso de Bacharelado em Ciência da Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Profa. Dra. Karina Mochetti de Magalhães

Niterói - RJ

2017

Dedico este trabalho aos meus pais e a toda minha família que me apoiaram desde sempre e sem eles não seria o que sou e onde cheguei.

Ao curso de Ciência da Computação na Universidade Federal Fluminense, aos professores e aos colegas de faculdade que conheci ao longo desses anos.

Agradecimentos

À professora Karina Mochetti, pela orientação, revisão, cobrança, paciência e coragem de me aceitar como orientando.

Aos meus pais que me ajudaram no que era possivel para eles, sempre me incentivando não importando o que fosse.

À minha irmã...bom, esquece.

Aos meus amigos Pumbas pelo incentivo, auxílio e testes sempre que necessário não importando a hora.

Resumo

A construção de um computador quântico terá impactos profundos em várias áreas, tais como Física, Matémática e, especialmente, a Ciência da Computação. As dificuldades práticas em se construir um computador quântico são muitas, logo não é possível prever quando ou mesmo se um computador quântico será construído. Apesar disso, a teoria na área da computação quântica vem evoluindo fortemente e para auxiliar na análise desses algoritmos pode ser essencial a criação de um simulador quântico de fácil acesso. Esse simulador utilizaria um computador clássico para realizar os trabalhos de um computador quântico, simulando o entrelaçamento e a sobreposição em bits clássicos e realizando uma estimativa de tempo gasto num computador quântico. O objetivo desse trabalho é realizar a implementação desse simulador de algoritmos quânticos.

Palavras-chave: quântica, circuito, simulador quântico, aplicação web.

Abstract

The construction of a quantum computer will have deep impacts in several areas, such as Physics, Mathematics, and especially Computer Science. There are several practical difficulties in building a quantum computer, so it is not possible to predict when or even if a quantum computer will be built. Nevertheless, the theory in the Quantum Computing field is evolving strongly and to assist in the analysis of these algorithms can be essential to create a quantum simulator. This simulator would be implemented in a classic computer, performing the work of a quantum computer, emulating the superposition and entanglement in classical bits with an estimate of efficiency. Therefore, the goal of this work is to implement a simulator for quantum algorithms.

Keywords: quantum, quantum simulator, circuit, web application.

Sumário

Lista de Figuras

Capítulo 1

Introdução

A computação quântica aproveita a possibilidade de superposição de estados das partículas subatômicas [1]. Em um computador normal, que chamaremos de clássico, um bit é um simples pedaço de informação que possui apenas dois estados, 0 ou 1. Para representar tal informação um computador deve conter algum tipo de sistema físico com dois estados distintos para associar aos valores como, por exemplo, um switch que pode estar fechado (1) ou aberto (0) [16].

Computadores quânticos, por sua vez, utilizam bits quânticos ou "qubits", que também possuem dois estados, entretanto, diferente do bit clássico, qubits podem armazenar muito mais informação já que eles podem existir em qualquer sobreposição desses valores, 0, 1, ou ambos ao mesmo tempo [1].

No mundo da mecânica quântica os eventos são governados por probabilidades. Um átomo radioativo, por exemplo, pode enfraquecer emitindo um elétron, ou não. É possível preparar um experimento de tal forma que exista uma chance exata de 50% de que um dos átomos de um pedaço de material radioativo caia em um determinado tempo [10].

Pode-se tomar como exemplo o experimento teórico do "Gato de *Schrödinger*". São colocados um gato, um frasco de veneno e uma fonte de radiação em uma caixa selada. Se um monitor interno (ex. Contador Geiger) detecta radiação, o frasco é quebrado soltando o veneno, matando o gato. No mundo quântico existe uma chance de 50% de o gato estar morto, e sem olhar dentro da caixa não podemos dizer se o gato está vivo ou morto. Segundo a teoria, nenhuma das duas possibilidades tem qualquer realidade a menos que seja observada [10]. Isso se equivale para o qubit, ele pode ser 0 ou 1 ao mesmo tempo mas quando ele é observado ele deixa de ser um bit quântico e passa a ser um bit clássico,

ou seja, só pode ser 0 ou 1.

Outra representação de um qubit pode ser uma esfera imaginária, ilustrada na Figura 1.1, onde um bit clássico pode estar apenas em um dos pólos da esfera e o qubit, bit quântico, pode estar em qualquer ponto da esfera. Assim qubits podem armazenar uma quantidade muito maior de informação que um bit clássico.

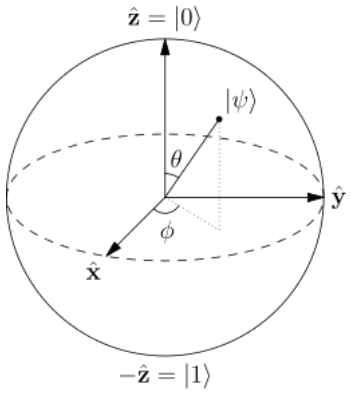

Figura 1.1: Esfera de Bloch

1.1 Motivação e Objetivo

A computação quântica é uma área de pesquisa muito relevante, pois utiliza elementos de áreas importantes como a Física, a Matemática e a Engenharia. Apesar de ainda não ser possível prever quando ou mesmo se um computador quântico será construído, caso isto ocorra serão gerados grandes impactos em todas essas áreas. Em um computador quântico, os sistemas não devem ter interações físicas que não estejam sob o controle total e completo do programa. Diferente de como ocorre em um computador clássico, interações físicas não controladas introduziriam interrupções potencialmente catastróficas na operação de um computador quântico, o que resulta em uma dificuldade na construção do mesmo.

Entretanto, os benefícios que podem ser gerados com a construção de um computador quântico vão muito além do que os pesquisadores podem imaginar. Para os cientistas da computação, o mais impressionante na computação quântica é a eficiência alcançada por seus algoritmos, algo não possível na teoria clássica da complexidade computacional. O tempo de execução de algumas tarefas em um computador quântico cresce de forma muito mais lenta com o tamanho da entrada do que em um computador clássico, como exemplo temos o algoritmo clássico de busca que possui complexidade O(N) e o algoritmo

quântico de busca, conhecido por Algoritmo de Grover, possibilita que esse método de busca chegue a complexidade $O(\sqrt{N})$ [15].

Neste trabalho apresentamos a implementação de um simulador de circuitos quânticos que utiliza um computador clássico para executar os algoritmos quântico. Assim, características quânticas como o entrelaçamento e a sobreposição serão realizadas em bits clássicos o que resultará num gasto de tempo maior. Por isso, o simulador também gera uma estimativa de tempo gasto a partir das portas quânticas, emulando o que seria obtido por um computador quântico caso ele seja construído; as portas e o tempo gasto em cada uma são dados pelo usuário. O simulador é uma ferramenta online e dispõe de arraste das portas quânticas para formação dos circuitos com o intuito de facilitar a interação com o usuário.

1.2 Revisão Bibliográfica

Nesta sessão mostramos alguns simuladores que existem e quais suas vantagens e desvantagens.

1.2.1 Quantum - Davy Wybiral

Existe um simulador online desenvolvido por Davy Wybiral de Austin Texas, *Quantum* ou como está no projeto dele do Git *Quantum Circuit Simulator* [31]. O projeto foi disponibilizado no Git em Janeiro de 2017 [30]. Davy Wybiral utiliza o Canvas para gerar toda a parte gráfica dos circuitos e menus, e javascript para toda a parte de cálculo. O simulador não possui nenhum tipo de documentação e tem uma interface bem simples com um menu superior onde existem algumas opções e uma área com as portas quânticas que podem ser adicionadas nos circuitos e ter a quantidade dos mesmos alterada, como pode ser visto na Figura 1.2.

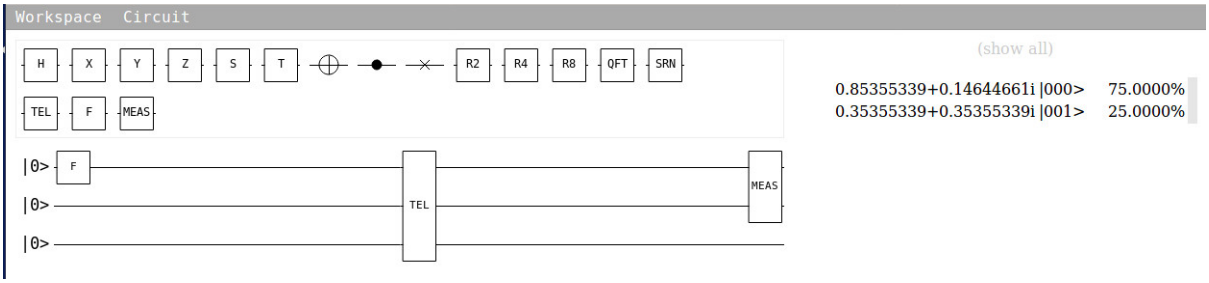

Figura 1.2: Simulador quântico de Davy Wybiral

A montagem do circuito funciona com a seleção de uma das portas e após isso basta clicar nos circuitos para adicionar a porta selecionada. Pode-se deletar uma porta clicando com o botão direito. Nos casos de portas de 2-bits é necessário que haja uma ação de clique e arraste onde o clique seleciona a posição do bit de controle e o largar selecionar a porta. Existe também a possibilidade de arrastar com uma porta de 1-bit selecionada o que adiciona a mesma em diversos circuitos.

Para efetuar o cálculo do circuito basta pressionar a tecla Enter ou selecionar no menu a opção *Evaluate*. O resultado é mostrado ao lado direito com todas as possibilidades possíveis e suas respectivas probabilidades.

1.2.2 IBM Q experience

IBM *Quantum experience* (IBM Q experience) tem como objetivo ajudar no aprendizado sobre o mundo quântico lendo o guia de usuário da ferramenta, realizando seus próprios experimentos, simulando-os e executando-os no processador quântico através do IBM Cloud [29].

O IBM Quantum Experience é composto por:

- um conjunto de tutoriais que vão desde simples experimentos com únicos qubits a experimentos mais complexos com múltiplos qubits;
- o *Quantum Composer*, uma interface gráfica para criação de circuitos quânticos, os quais eles chamam de *quantum score*;
- um simulador que testa seus *quantum scores*;

- acesso a um processador quântico que roda no laboratório da IBM *Quantum Computing*, onde os seus quantum scores serão executados;
- a *quantum community* onde *quantum scores*, ideias e experiências são compartilhadas e discutidas.

No *Quantum Composer*, interface gráfica para construção de circuitos quânticos, logo de inicio existe a opção de executar em um processador quântico "real" ou em um processador quântico customizado. Na segunda opção as portas podem ser colocadas em qualquer lugar enquanto na primeira a topologia é definida pelo dispositivo físico no qual está sendo executado no laboratório.

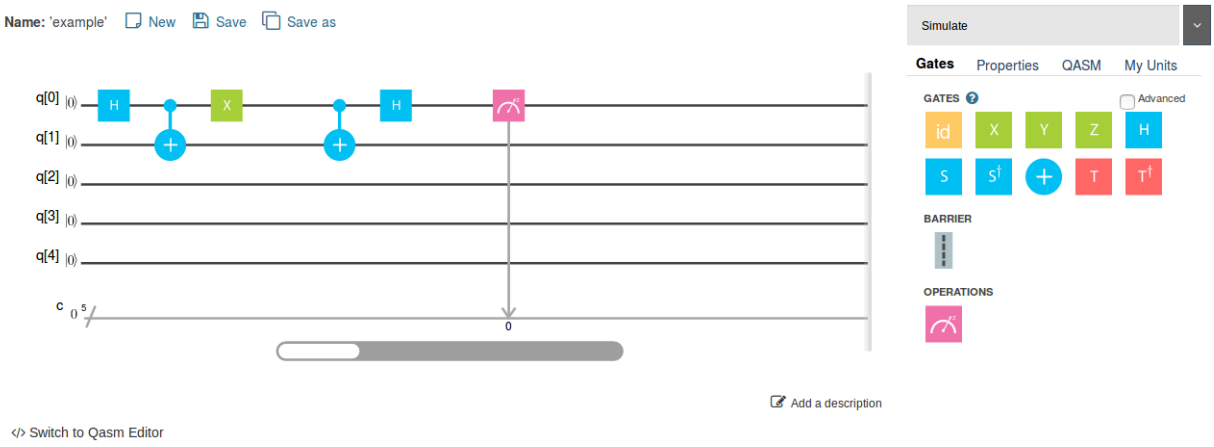

Figura 1.3: Interface gráfica *Quantum Composer* do IBM Q Experience

Como pode-se notar na Figura 1.3 a interface é composta pela área do circuito na parte esquerda da tela com cada linha sendo equivalente a um qubit. Já na parte direita existem as portas que podem ser arrastadas para montar o circuito, e caso o usuário clique em alguma porta dentro do circuito uma explicação sobre a mesma é mostrada nesta área.

1.2.3 Quirk - Craig Gidney

O projeto teve seu início em Março de 2014, mas não teve commits após o mês de criação. Em Novembro do mesmo ano, 2014, voltou a ter commits e entre Março de 2016 e Novembro de 2016 teve a maior concentração de commits, 647 dos 1006 totais. O projeto ainda vem sendo atualizado sendo o commit mais recente do dia 29 de Abril de 2017 [7].

Craig Gidney, desenvolvedor do Quirk, explica que decidiu criar o simulador porque ficou interessado em computação quântica e pelo fato de ter lido "*Media for Thinking the Unthinkable*" de Bret Victor onde ele menciona benefícios do *feedback* imediato em relação ao entendimento e produtividade. Craig diz não ter achado nenhum simulador de circuitos quânticos que passasse essa experiência de "manipulação direta" [8]. Foi desenvolvido utilizando Webgl. Na Figura 1.4 pode-se ver uma captura de tela do simulador.

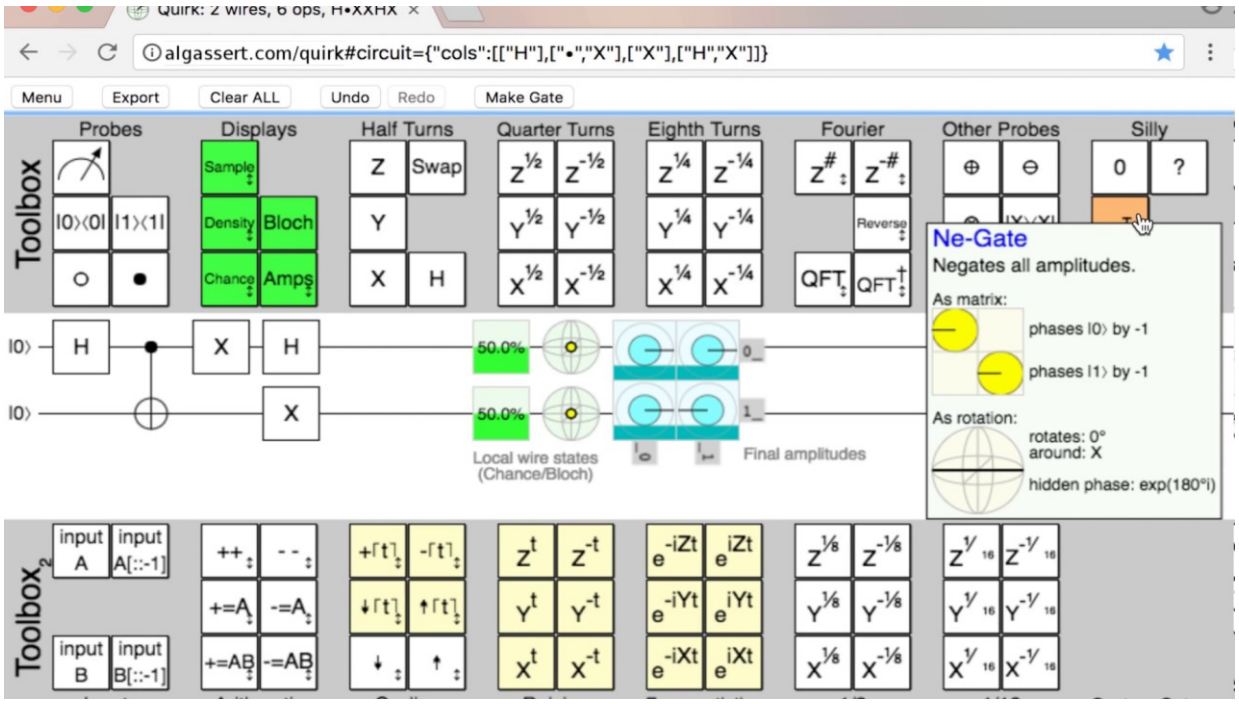

Figura 1.4: Simulador quântico Quirk de Craig Gidney

Craig faz comparações com alguns simuladores como, por exemplo, o já mencionado de Davy Wybiral e o IBM *Quantum Experience*. Ele também menciona alguns outros como *Quantum Computing Playground* e Microsoft's LIQ Ui|>.

O simulador Quirk é composto por um menu que possui opções de carregar alguns exemplos de circuitos já prontos, exportar o circuito montado, desfazer ou refazer alguma alteração e inclusive criar porta. Ele possui uma área com as opções de portas a serem utilizadas no circuito, formas de mostrar o resultado e mais diversas opções. Algumas vantagens desse simulador são: as portas são colocadas da forma "arrastar e soltar", o resultado é mostrado a todo momento não havendo a necessidade de se apertar um botão para isso, e é possível adicionar mais bits ao circuito dinamicamente com o arrastar de uma das portas para além da linha do último bit.

1.3 Organização do Texto

Este trabalho está divido da seguinte forma: a Seção 2 são apresentados alguns tópicos sobre circuito quântico que são utilizados ao longo do projeto; na Seção 3 é apresentada a escolha das ferramentas utilizadas para o desenvolvimento do simulador, expondo a teoria e os frameworks existentes atualmente; a Seção 4 apresenta como a solução foi desenvolvida e como resultado o código gerado; e por final na Seção 5 são apresentadas as conclusões e trabalhos futuros.

Capítulo 2

Circuito Quântico

2.1 Circuitos digitais

Os circuitos lógicos clássicos são formados por portas como AND, OR e NOT que recebem uma sequência de valores binários 0 e 1. Ao passar pelas portas é gerada uma saída que representa a entrada transformada pelo circuito, como ilustrado na Figura 2.1.

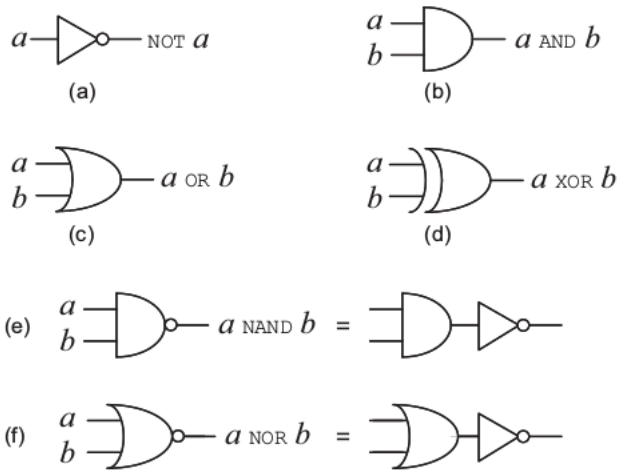

Figura 2.1: Exemplos de portas clássicas

2.2 Qubit

Dentro do espaço vetorial bidimensional os únicos vetores clássicos são os dois vetores ortogonais $|0\rangle$ e $|1\rangle$, visto que esses são os únicos dois estados que um bit clássico pode ter. Já um bit quântico, ou seja, um qubit possui um estado associado $|\Psi\rangle$ que pode ser

qualquer vetor unitário dentro do espaço vetorial bidimensional abrangido por $|0\rangle$ e $|1\rangle$ sobre os números complexos, ou seja, um vetor composto de dois números complexos que chamaremos de a e b, como pode se notar em (2.1).

$$|\Psi\rangle = a\,|0\rangle + b\,|1\rangle = \begin{pmatrix} a \\ b \end{pmatrix}. \tag{2.1}$$

Como a única exigência de $|\Psi\rangle$ é ser um vetor unitário dentro do espaço vetorial complexo, a condição de normalização se aplica a ele, tendo assim (2.2)

$$|a|^2 + |b|^2 = 1, \tag{2.2}$$

O estado $|\Psi\rangle$ define uma superposição dos estados $|0\rangle$ e $|1\rangle$ com amplitudes a e b. Se a for 1, por exemplo, temos o caso especial onde o estado do qubit é igual a $|0\rangle$. Respectivamente, temos o estado $|1\rangle$ caso b seja 1.

Ao ser medido os elementos $|a|^2$ e $|b|^2$ são, respectivamente, a probabilidade de que o qubit seja $|0\rangle$ e $|1\rangle$, logo

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad\qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

2.3 Entrelaçamento

Assim como o estado geral de um único qubit é qualquer superposição normalizada dos dois possíveis estados clássicos (2.1), o estado geral de $|\Psi\rangle$ nos possibilita associar com dois qubits em qualquer superposição dos quatro estados clássicos,

$$|\Psi\rangle_2 = a\,|00\rangle + b\,|01\rangle + c\,|10\rangle + d\,|11\rangle = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \tag{2.3}$$

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Assim os valores das quatro amplitudes devem seguir a condição de normalização, onde

$$|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1 \tag{2.4}$$

Generalizando para n qubits o estado geral pode ser qualquer superposição dos 2^n estados clássicos, com amplitudes cujo quadrado somam a unidade:

$$|\Psi\rangle = \sum_{0 \leq x < 2^n} \alpha_x \, |x\rangle_n$$
$$\sum_{0 \leq x < 2^n} |\alpha_x|^2 = 1.$$

No contexto de computação quântica, o conjunto de 2^n estados clássicos, isto é, todos os possíveis produtos tensores de n qubits $|0\rangle$ e $|1\rangle$ é chamado de base clássica. Os estados que caracterizam n bits clássicos, estados de base clássica, são um conjunto bem limitado dos estados de n qubits, os quais podem ser qualquer superposição com coeficientes complexos desses estados de base clássica.

Se tivermos dois qubits, um no estado $|\Psi\rangle = a' \, |0\rangle + b' \, |1\rangle$ e outro no estado $|\Phi\rangle = c' \, |0\rangle + d' \, |1\rangle$, então o estado de 2-qubits $|\Psi\rangle$ do par é o produto tensorial dos estados individuais,

$$|\Psi\rangle_2 = |\Psi\rangle \otimes |\Phi\rangle = a'c' \, |00\rangle + a'd' \, |01\rangle + b'c' \, |10\rangle + b'd' \, |11\rangle = \begin{pmatrix} a' \cdot c' \\ a' \cdot d' \\ b' \cdot c' \\ b' \cdot d' \end{pmatrix} \tag{2.5}$$

Com (2.5) e a condição de normalização temos

$$\begin{aligned} |a' \cdot c'|^2 + |a' \cdot d'|^2 + |b' \cdot c'|^2 + |b' \cdot d'|^2 \\ |a'|^2 \cdot |c'|^2 + |a'|^2 \cdot |d'|^2 + |b'|^2 \cdot |c'|^2 + |b'|^2 \cdot |d'|^2 \\ |a'|^2 \cdot (|c'|^2 + d'^2) + |b'|^2 \cdot (|c'|^2 + |d'|^2) \\ |a'|^2 \cdot 1 + |b'|^2 \cdot 1 \\ |a'|^2 + |b'|^2 = 1 \end{aligned}$$

Note que um estado geral de 2-qubit (2.3) é da forma especial (2.5) se e somente se $a' \cdot d' = b' \cdot c'$. Como as quatro amplitudes seguem a condição de normalização (2.4), essa relação não se mantém, e o estado geral de 2-qubit é diferente do estado geral de 2 bits

clássicos, ele não é um produto (2.5) de dois estados de 1-qubit. O mesmo é verdade para estados de n-qubits. Enquanto o estado geral de n bits clássicos é um dos 2^n produtos de $|0\rangle$s e $|1\rangle$s, o estado geral de n-qubits é a superposição desses 2^n produtos e não pode ser expresso como um produto de qualquer conjunto de estados de 1-qubit. Esse não produto de estados de dois ou mais qubits é chamado de entrelaçamento.

2.4 Portas quânticas

Assim como um computador clássico é formado de circuitos com portas lógicas, um computador quântico também é formado por circuitos. Porém, este último possui portas quânticas [20], que definem as operações básicas que podemos realizar em qubits. Essas portas podem ser representadas graficamente através de circuitos, como visto na Figura 2.2, ou matematicamente por matrizes, como visto na equação (2.6).

$$|\Psi'\rangle = U\,|\Psi\rangle \tag{2.6}$$

Figura 2.2: Porta quântica

Uma porta quântica deve sempre ser representada por uma matriz unitária pois ao multiplicar o vetor pela matriz a norma do vetor deve ser mantida.

2.4.1 Identidade

A operação Identidade quântica mantém os valores iniciais do qubit, representada graficamente pela Figura 2.3 e pela matriz (2.7). Ela é trivial e não realiza qualquer modificação no qubit.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.7}$$

Figura 2.3: Porta quântica Identidade

$$I\left|\Psi\right\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 \cdot a + 0 \cdot b \\ 0 \cdot a + 1 \cdot b \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} = \left|\Psi'\right\rangle$$

2.4.2 NOT

Considerando as portas de um único bit, a única porta clássica é a porta NOT a qual faz com que os bits 0 e 1 se invertam. Imagine o mesmo processo que recebe o estado $|0\rangle$ e retorna o estado $|1\rangle$, e vice versa. Este processo seria um bom candidato para a porta quântica análoga a NOT clássica. Entretanto, especificando a ação sobre os estados $|0\rangle$ e $|1\rangle$ não nos diz nada sobre o que aconteceria sobre a superposição dos mesmos. Na verdade, a porta quântica NOT age linearmente, ou seja, recebe o estado $a\left|0\right\rangle + b\left|1\right\rangle$ e retorna $b\left|0\right\rangle + a\left|1\right\rangle$, assim invertendo as amplitudes do qubit. A porta quântica NOT pode ser representada graficamente pela Figura 2.4 e um outro modo conveniente de representá-la é em forma de matriz (2.8).

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.8}$$

Figura 2.4: Porta quântica NOT

Com o estado $|\Psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix}$, podemos aplicar (2.8) e teremos

$$X\left|\Psi\right\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 0 \cdot a + 1 \cdot b \\ 1 \cdot a + 0 \cdot b \end{pmatrix} = \begin{pmatrix} b \\ a \end{pmatrix} = \left|\Psi'\right\rangle$$

2.4.3 Y e Z

Diferentemente da computação clássica, na qual só existe uma porta de um bit, na computação quântica existem várias portas de um único qubit. Como, por exemplo, as operações Y e Z, que são representadas graficamente pelas Figuras 2.5 e 2.6, e matricialmente por (2.9) e (2.10) respectivamente. Essas operações são análogas a operação NOT se pensarmos na representação do estado de um qubit como uma esfera, ilustrado pela esfera de

Bloch na Figura 1.1. A operação NOT é equivalente a rotação da esfera no eixo X em π radianos e análogamente, as operações Y e Z são rotações de π radianos nos eixos Y e Z, respectivamente.

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.9}$$

Figura 2.5: Porta quântica Y

$$Y\,|\Psi\rangle = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 0 \cdot a - i \cdot b \\ i \cdot a + 0 \cdot b \end{pmatrix} = \begin{pmatrix} -bi \\ ai \end{pmatrix} = |\Psi'\rangle$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.10}$$

Figura 2.6: Porta quântica Z

$$Z\,|\Psi\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 \cdot a + 0 \cdot b \\ 0 \cdot a - 1 \cdot b \end{pmatrix} = \begin{pmatrix} a \\ -b \end{pmatrix} = |\Psi'\rangle$$

2.4.4 Hadamard

A porta Hadamard é uma operação fundamental na computação quântica e não possui uma operação clássica equivalente. Ela pode ser representada pela Figura 2.8 ou pela matriz (2.11). Essa porta transforma os qubits bases $|0\rangle$ e $|1\rangle$ para um estado de sobreposição onde $|0\rangle$ e $|1\rangle$ possuem a mesma probabilidade. Se consideramos a esfera de Bloch ela seria equivalente a realizar essa operação sobre ela, isto é uma rotação da esfera no eixo Y de 90^o e em seguida uma rotação no eixo X de 180^o, como é mostrado na Figura 2.7.

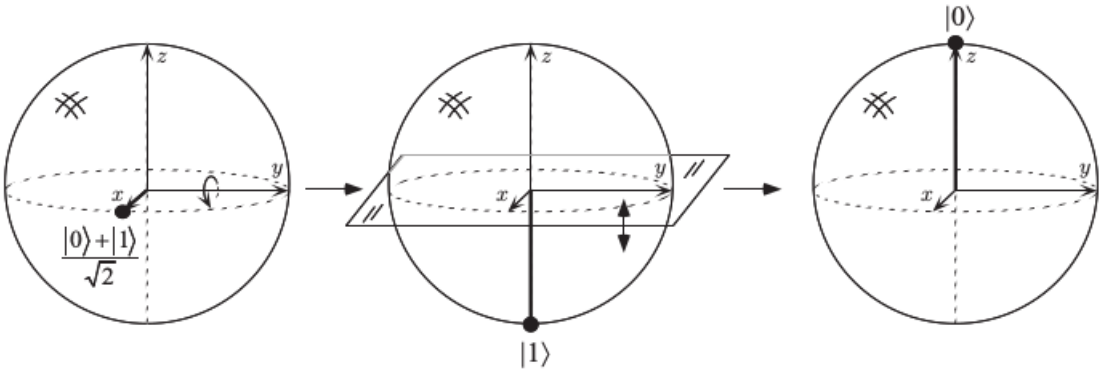

Figura 2.7: Esferas de Bloch da operação Hadamard

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.11}$$

Figura 2.8: Porta quântica Hadamard

Assim, para um qubit de probabilidade $|a|^2$ e $|b|^2$ teríamos um qubit resultante com probabilidades $\frac{(a+b)^2}{2}$ e $\frac{(a-b)^2}{2}$

$$H|\Psi\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \cdot a + 1 \cdot b \\ 1 \cdot a - 1 \cdot b \end{pmatrix} = \begin{pmatrix} \frac{a+b}{\sqrt{2}} \\ \frac{a-b}{\sqrt{2}} \end{pmatrix} = |\Psi'\rangle \tag{2.12}$$

2.4.5 CNOT

A operação CNOT é uma porta de dois qubits conhecidos como controle e alvo, representada pela matriz (2.13) ou pela Figura 2.9, na qual a linha de cima representa o qubit de controle e a de baixo o qubit alvo. A porta CNOT funciona da seguinte maneira: dados dois qubits, $|x\rangle$ e $|y\rangle$, a matriz C mantém o valor de $|y\rangle$ caso $|x\rangle$ seja igual a $|0\rangle$ e caso $|x\rangle$ seja igual a $|1\rangle$ ela inverte o valor de $|y\rangle$, como é mostrado em (2.14).

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.13}$$

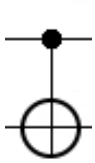

Figura 2.9: Porta quântica CNOT

Assim, para dois qubits entrelaçados, ela inverterá as amplitudes referentes aos qubits $|10\rangle$ e $|11\rangle$:

$$C\,|\Psi\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 1 \cdot a + 0 \cdot b + 0 \cdot c + 0 \cdot d \\ 0 \cdot a + 1 \cdot b + 0 \cdot c + 0 \cdot d \\ 0 \cdot a + 0 \cdot b + 0 \cdot c + 1 \cdot d \\ 0 \cdot a + 0 \cdot b + 1 \cdot c + 0 \cdot d \end{pmatrix} = \begin{pmatrix} a \\ b \\ d \\ c \end{pmatrix} = |\Psi'\rangle \qquad (2.14)$$

2.4.6 Medição

Outra porta importante é a de medição, que comumente é representada por um "medido", como pode ser observado na Figura 2.10. Essa operação converte um único qubit no estado $|\Psi\rangle = a\,|0\rangle + b\,|1\rangle$ em um bit clássico probabilístico, o qual é 0 com uma probabilidade de $|a|^2$, ou 1 com uma probabilidade $|b|^2$. Para fazer essa diferenciação entre qubit e bit clássico em um circuito lógico quântico, um traço representa um qubit e um traço duplo um bit clássico.

Esta é uma porta trivial em um circuito clássico, normalmente nem representada, sendo uma das maiores diferenças entre circuitos quânticos e clássicos.

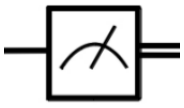

Figura 2.10: Porta quântica de medição

2.5 Operações em qubits entrelaçados

Dados dois qubits $|\Psi_1\rangle$ e $|\Psi_2\rangle$ podemos fazer a operação A em $|\Psi_1\rangle$ e B em $|\Psi_2\rangle$ como C se ambos estiverem entrelaçados formando um 2-qubit (2.15).

$$C = A \otimes B \tag{2.15}$$

O produto tensorial pode ser expandido para matrizes em geral.

$$A \otimes B = \begin{pmatrix} a_{11} \cdot B & a_{12} \cdot B & \dots & a_{1n} \cdot B \\ a_{21} \cdot B & a_{22} \cdot B & \dots & a_{2n} \cdot B \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} \cdot B & a_{n2} \cdot B & \dots & a_{nn} \cdot B \end{pmatrix}$$

Para qualquer operação em $|\Psi_1\rangle$ seja transformada em uma operação para um 2-qubit $|\Psi_{12}\rangle$, basta realizar o produto tensorial com a identidade em $|\Psi_2\rangle$

$$C = A \otimes I \tag{2.16}$$

$$\begin{aligned} A \otimes I(|\Psi_1\rangle \otimes |\Psi_2\rangle = \\ (A\,|\Psi_1\rangle) \otimes (I\,|\Psi_2\rangle) = \\ A\,|\Psi_1\rangle \otimes |\Psi_2\rangle = \\ A(|\Psi_1\rangle \otimes |\Psi_2\rangle) \end{aligned}$$

Agora no caso de duas operações transformadas, dadas as operações A em $|\Psi_1\rangle$ e B em $|\Psi_2\rangle$, temos $C = A\,|\Psi_1\rangle \otimes I\,|\Psi_2\rangle$ e $B_{12} = I\,|\Psi_1\rangle \otimes B\,|\Psi_2\rangle$.

Assim,

$$\begin{aligned} (A\,|\Psi_1\rangle \otimes I\,|\Psi_2\rangle)(I\,|\Psi_1\rangle \otimes B\,|\Psi_2\rangle) = \\ AI\,|\Psi_1\rangle \otimes IB\,|\Psi_2\rangle = \\ A\,|\Psi_1\rangle \otimes B\,|\Psi_2\rangle \end{aligned}$$

2.6 Código Superdenso

Apesar de uma quantidade infinita de informações serem necessárias para especificar o estado $|\Psi\rangle = a\,|0\rangle + b\,|1\rangle$ de um único qubit, não existe um modo no qual alguém que tenha adquirido um qubit possa descobrir o valor de suas amplitudes. Se Alice prepara

um qubit no estado $|\Psi\rangle$ e o envia para Bob, tudo que ele pode fazer é aplicar uma transformação unitária à sua escolha e fazer a medição do qubit, chegando ao valor 0 ou 1. O máximo que Alice pode comunicar para Bob enviando um único qubit é um único bit de informação.

E para esse problema existe o código superdenso, seu protocolo consiste de 3 usuários. Alice, que deseja enviar 2 bits clássicos para Bob utilizando somente um qubit e Eve que fará a inicialização do sistema. Usaremos algumas das portas citadas nas seções 2.4.1 a 2.4.6 para explicar o código superdenso.

Abaixo, temos os passos do protocolo Superdenso.

1. Eve inicializa um qubit de 2-qubits $|\Psi\rangle = |\Psi'\Psi''\rangle$ com zero;
2. Eve usa a porta Hadamard em $|\Psi'\rangle$;
3. Eve usa a porta CNOT em $|\Psi\rangle$;
4. Eve envia $|\Psi'\rangle$ para Alice e $|\Psi''\rangle$ para Bob;
5. Alice deseja enviar dois bits clássicos $|ab\rangle$ para Bob;
6. Alice modifica o qubit $|\Psi'\rangle$ para $U|\Psi'\rangle$ da seguinte maneira:
   - Se $|ab\rangle = |00\rangle$, então U = I.
   - Se $|ab\rangle = |01\rangle$, então U = X.
   - Se $|ab\rangle = |10\rangle$, então U = Z.
   - Se $|ab\rangle = |11\rangle$, então U = XZ.
7. Alice envia $U|\Psi'\rangle$ para Bob;
8. Bob usa a porta CNOT em $|\Psi\rangle$;
9. Bob usa a porta Hadamard em $U|\Psi'\rangle$;
10. Bob faz a medição dos dois qubits, colapsando-os para bits clássicos $|ab\rangle$.

A seguir, exemplificaremos o protocolo.

Na inicialização do sistema, Eve inicializa um qubit de tamanho dois $|\Psi\rangle = |\Psi'\Psi''\rangle$ com zero.

$$|\Psi\rangle = |00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Após isso, Eve aplica a operação Hadamard em $|\Psi'\rangle$

$$H \otimes I = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & -\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix}$$

$$H \otimes I = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

$$(H \otimes I)\,|\Psi\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$$

e usa a porta CNOT em $|\Psi\rangle$.

$$C\,|\Psi\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

Em seguida Eve envia $|\Psi'\rangle$ para Alice e $|\Psi''\rangle$ para Bob.

Alice que deseja enviar dois bits clássicos $|ab\rangle$ para Bob, ao receber o qubit $|\Psi'\rangle$ de Eve aplica uma transformação U em $|\Psi'\rangle$ da seguinte forma:

- Se $|ab\rangle = |00\rangle$, então U = I.

$$U = I \otimes I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(I \otimes I)\,|\Psi\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

- Se $|ab\rangle = |01\rangle$, então U = X.

$$U = X \otimes I = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$(X \otimes I)\,|\Psi\rangle = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} \tag{2.17}$$

- Se $|ab\rangle = |10\rangle$, então U = Z.

$$U = Z \otimes I = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

$$(Z \otimes I)\,|\Psi\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

- Se $|ab\rangle = |11\rangle$, então U = XZ.

$$U = XZ \otimes I = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

$$(XZ \otimes I)\,|\Psi\rangle = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

Depois de aplicada a transformação, Alice envia $U\,|\Psi'\rangle$ para Bob. Para continuar o exemplo usaremos $|ab\rangle = |01\rangle$ (2.17).

Bob aplica a porta CNOT em $|\Psi\rangle$

$$C\,|\Psi\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix},$$

em seguida usa a porta Hadamard em $U\,|\Psi'\rangle$ que recebeu de Alice.

$$H \otimes I = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

$$(H \otimes I)\,|\Psi\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Após tudo isso Bob faz a medição dos dois qubits, passando estes para bits clássicos $|ab\rangle$

$$|\Psi\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$|\Psi\rangle = 0 \cdot |00\rangle + 1 \cdot |01\rangle + 0 \cdot |10\rangle + 0 \cdot |11\rangle$$

$$|ab\rangle = |01\rangle .$$

A Figura 2.11 ilustra o código Superdenso.

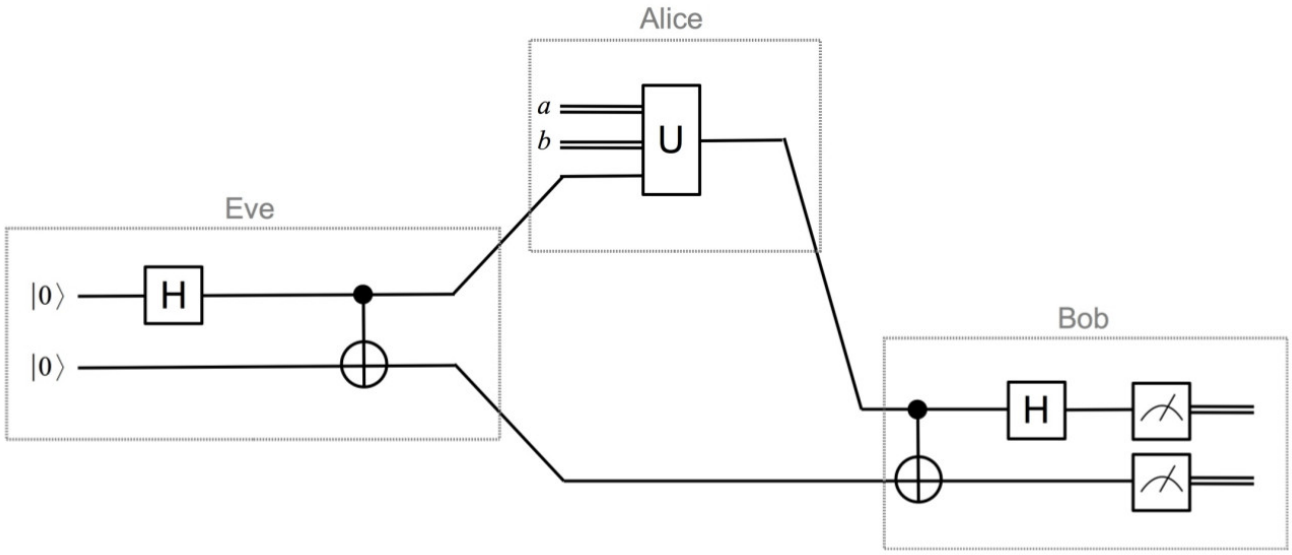

Figura 2.11: Código Superdenso

Capítulo 3

Escolha das ferramentas

Neste capítulo falaremos sobre as opções e escolhas de linguagem, biblioteca e frameworks para desenvolvimento do projeto.

3.1 Linguagem

Python foi a escolha por possuir todos os requisitos necessários para o desenvolvimento do projeto, frameworks de desenvolvimento web e bibliotecas de circuitos quânticos. Além disso, ela é vastamente utilizada em diversas aplicações e empresas como YouTube, Google, Industrial Light e Magic [6] o que adiciona mais valor ao seu aprendizado.

3.2 Biblioteca

Tendo em vista que escolhemos Python como linguagem, precisamos definir alguma biblioteca para auxílio com os cálculos de cicuito quântico. A seguir falamos sobre algumas opções encontradas e qual foi a escolhida.

3.2.1 PyQu

PyQu é um módulo de extensão para Python 3 com o objetivo principal de providenciar um completo conjunto de tipos de dados e funções para simular computação quântica com uma sintaxe pura. Entretanto o último commit feito foi há 7 anos o que nos fez entender que o projeto foi descontinuado, fazendo com que descartássemos essa opção [24].

3.2.2 Qitensor

Qitensor, outro módulo que essencialmente é um pacote para Numpy [21], utiliza semânticas mais úteis para mecânica quântica de dimensões finitas de muitas partículas. Infelizmente essa biblioteca também não recebia atualizações há mais de um ano [26].

3.2.3 Qubiter

Qubiter tem como intuito ajudar no designer e simulação de circuitos quânticos em computadores clássicos. É um projeto gêmeo do Quantum Fog e eles esperam que um dia haja uma interação entre o Quantum Fog e o Qubiter para realizar algumas tarefas, como compilação e simulação quântica [27].

3.2.4 QuTiP

QuTiP teve sua primeira versão em Julho de 2011 [18]. É um software open-source para simular as dinâmicas de um sistema quântico aberto que utiliza os pacotes numéricos Numpy, Scipy e Cython. Teve sua primeira versão em Julho de 2011. Possui também um output gráfico provido pela Matplotlib. QuTiP tem como objetivo providenciar simulações numéricas eficientes e amigáveis para o usuário de um variado número de Hamiltonianas comumente achadas em uma vasta quantidade de aplicações físicas. Está disponível sem custos para Linux, Mac OSX e Windows, sendo esta última com algumas restrições [19].

3.2.5 Escolhida

QuTip foi a escolhida por possuir pacotes numéricos e também output gráfico, mesmo que, este último não tenha sido utilizado no projeto. Além de possuir a melhor documentação dentre as bibliotecas pesquisadas, com exemplos e tutoriais, ele possui funcionalidades que podem auxiliar no desenvolvimento e em trabalhos futuros.

3.3 Framework Web

Frameworks web são projetados para dar suporte no desenvolvimento de sites dinâmicos, aplicações web e web services, auxiliando no trabalho de tarefas comuns como, controle de sessão, validação de dados, acesso ao banco de dados, templates, tratamento de requisições

e respostas HTTP. Dentre os frameworks web existentes para Python avaliamos neste trabalho Flask, Pyramid e Django.

3.3.1 Flask

Flask é o framework mais recente encontrado, foi criado em meados de 2010. Evoluiu muito a partir da análise de frameworks anteriores e tem se destacado em pequenos projetos. A sua comunidade é pequena comparada a do Django mas é ativa em listas de e-mail e no IRC [2].

3.3.2 Pyramid

O Pyramid foi criado a partir do projeto Pylons [23] e recebeu o nome de Pyramid somente no ano de 2010, mesmo tendo sua primeira release em 2005. É um framework tão maduro quanto o Django e disponibiliza mais flexibilidade para desenvolvedores que possuem projetos onde o caso de uso não se encaixe muito bem nos padrões do mapeamento objeto-relacional, que nada mais é do que a representação das tabelas do banco de dados através de classes onde cada resgistro é representado como uma instancia da classe [25]. Pode-se dizer que Pyramid é o framework mais flexível dentre os estudados. Apesar disso, pareceu muito mais complexo do que nosso projeto necessitava. Além disso, ele possui a comunidade mais inativa dentre os frameworks pesquisados.

3.3.3 Django

Django foi lançado em 2005. É um framework com foco em grandes aplicações que inclui dezenas de extras para lidar com as tarefas comuns do desenvolvimento web, como por exemplo autenticação e administração de conteúdo, além de auxiliar também com questões de segurança como SQL injection com seus querysets, evitar ataques Cross Site Scripting e fornecer proteção a Clickjacking [5]. Sem dúvida o Django é o framework mais popular e a lista de sites que o utilizam é impressionante, tendo mais de 5000 páginas [28]. Para sites que possuem requisitos comuns, Django segue padrões bem sensatos tornando-o uma escolha bem popular entre aplicações web de porte médio ou grande.

3.3.4 Comunidade

Django possui uma quantidade enorme de perguntas no StackOverflow se comparado com os outros dois Frameworks, Django 144.000, Flask 16.600, Pyramid 1.900 (Maio 2017) [22]. Já no Github Flask e Django têm números semelhantes e Pyramid 10 vezes menos, Flask possui um pouco mais de 27.200 estrelas, Django pouco mais de 25.900 e Pyramid 2.300 [9]. Esses números evidenciam a populariedade e o suporte oferecido por cada framework.

3.3.5 Escolhido

O Flask seria uma boa opção entretanto escolhemos o Django por:

- possuir uma grande comunidade ativa;
- diversas ferramentas já inclusas, evitando o tempo de reprogramação de funções comuns;
- ser facilmente escalável, pensando em uma futura continuação deste Projeto Final
- ser uma ferramenta muito utilizada no mercado, trazendo mais benefícios com o aprendizado.

Capítulo 4

Desenvolvimento e Resultados

O simulador deveria ter uma área com inputs de tempo, portas e o circuito. Primeiramente, definimos a interface geral, determinando onde seria localizada cada funcionalidade. Como um dos objetivos do projeto é simular o tempo que um computador quântico levaria para executar o circuito, decidimos por colocar os inputs em local de destaque, ficando assim à esquerda e a cima. Resolvemos que as portas ficariam também a cima, pois ao carregar a página o circuito estaria vazio e assim, ela se diferenciaria dos inputs ficando à direita. Por final, precisamos de uma área para mostrar o resultado do circuito que foi deixado abaixo de tudo pois ela seria dependente das anteriores. Também adicionamos abaixo do circuito, botões para exemplificar o código superdenso, descrito na seção 2.6 para facilitar os testes e uso inicial do simulador. Na Figura 4.1 pode ser visto uma captura de tela do simulador, que pode ser acessado em `https://demo-quantum.herokuapp.com/simulador/`.

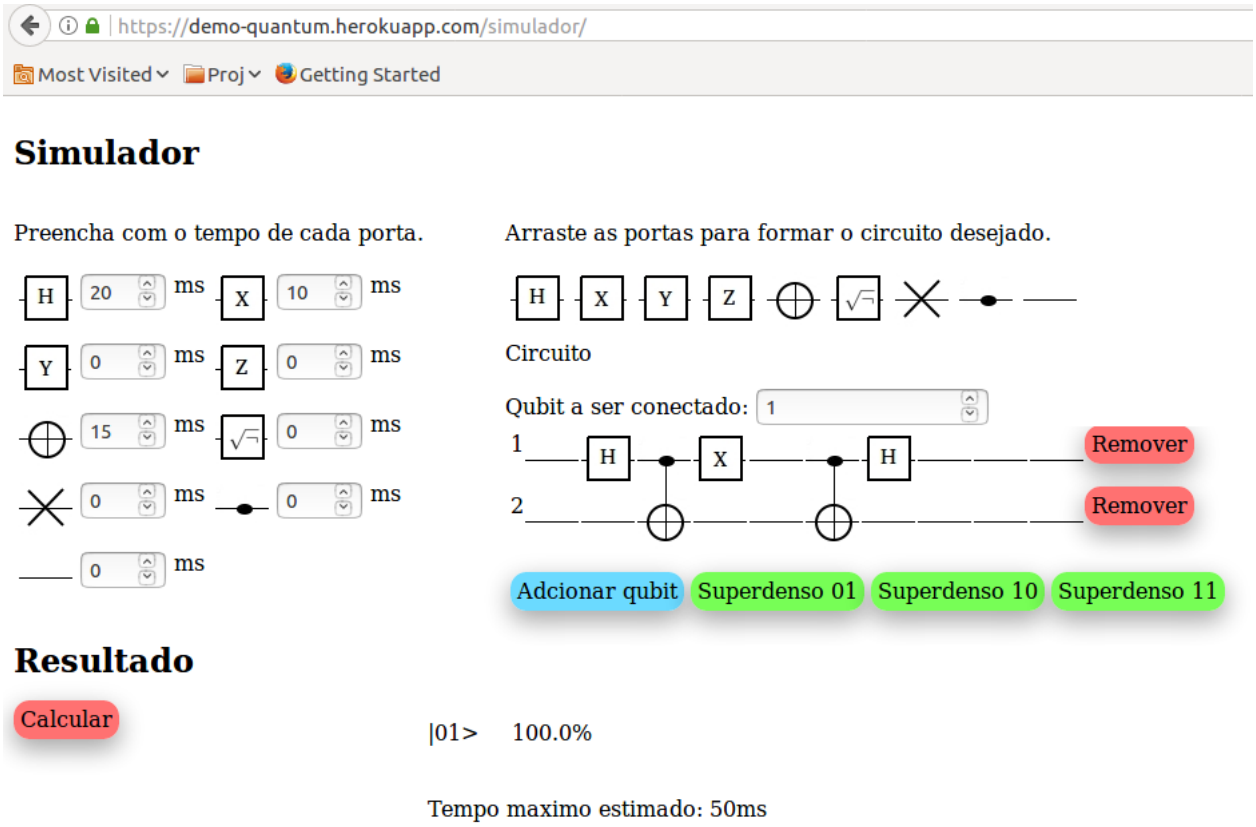

Figura 4.1: Nosso simulador circuito quântico online

4.1 Arquitetura

Escolhemos o padrão Model-View-Controller (MVC) para o projeto já que é o modelo padrão seguido pelo Django e que faz muito sentido para o desenvolvimento de aplicações online.

Esse estilo arquitetural é composto de três elementos: modelo, visão e controle, ilustrados na Figura 4.2. O modelo é composto dos elementos que representam os principais conceitos do domínio, se pensarmos na orientação a objetos, o modelo seriam as classes do sistema e que normalmente são persistidas em bancos de dados, mas não possuem nenhum acesso de fora do sistema. Já a visão é composta por elementos de interface com o usuário, podendo ser uma interface gráfica, linha de comando ou uma API que manipulam elementos do modelo. E por último, o controle são os elementos responsáveis pela orquestração, que manipulam uma visão e executam os casos de uso.

No MVC uma visão conhece seu modelo mas o modelo não conhece sua visão, os

controles conhecem suas visões mas a visão não conhece seu controle. E as entidades do modelo não conhecem ninguém a não ser as outras entidades do modelo.

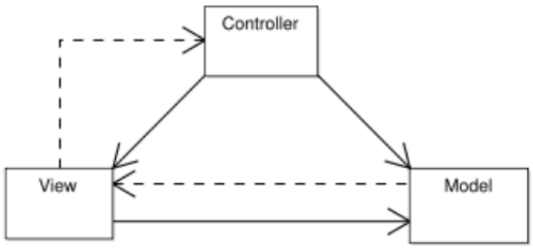

Figura 4.2: Diagrama MVC [17]

Vantagens do MVC:

- responsabilidades bem definidas por componente
- modularização da aplicação, o que traz independência
- fácil manutenabilidade
- alta reutilização
- possibilita desenvolvedores trabalharem simultaneamente

O Django possui uma particulares, ele utiliza o padrão Model View Controller (MVC), entretanto possui nomes peculiares para seus componentes. Ele renomeia a *view* para *template* e o *controller* para *view*, o que gera confusão e reclamações de muitos usuários. No próprio site do Django [5] existe uma pergunta sobre isso onde eles respondem que *view* representa os dados que são apresentados ao usuário, não necessariamente como é apresentado. Então para eles a *view* representa qual informação é vista e *template* como ela é vista [3].

4.2 Drag and Drop

Como o simulador deveria ser amigável para o usuário decidimos utilizar Drag & Drop (D&D) para a criação/edição do circuito. Resolvemos que o simulador deveria ter uma área com as portas quânticas disponíveis para montar o circuito quântico e a partir dessa área o usuário poderia arrastar as portas para formar o circuito. Entretanto os D&D

convencionais apenas possibilitavam o arrastar de um elemento para um outro container, ou seja, o elemento deixava de existir no container original e era adicionado no final. O que queríamos era que o elemento inicial continuasse existindo onde estava e que uma cópia do mesmo fosse criada e adicionada ao container alvo, por isso foi necessário implementar o nosso próprio.

Sendo assim utilizamos Javascript(JS) e JQuery [13], uma biblioteca de JS, para criar uma cópia do elemento que sofreu o evento de Drag e movemos a cópia até o destino do Drop. Ao realizar o Drop existe uma checagem para excluir o elemento existente e adicionar a cópia. Nos materiais encontrados sobre D&D o elemento utilizado pelo Javascript para capturar o elemento do drop era o ev.target o que gera alguns problemas já que esse target não traz especificamente o elemento com o evento de drop atrelado, podendo trazer uma das divs que o encapsulam ou algum outro elemento mais interno fazendo com que não houvesse uma consistência do retorno. Para resolver esse problema decidimos utilizar currentTarget ao invés do target.

Quando pensamos nas portas de 2 qubits nos deparamos com outro problema, o de a seleção do qubit de controle. A escolha foi ter um input que serviria de alvo do segundo qubit, fazendo o tratamento para quando o alvo era igual ao bit do Drop ou caso o alvo fosse maior que o número de bits. No primeiro caso é solto um alerta com a mensagem de que não é possível arrastar a porta para o mesmo bit alvo da conexão, e no segundo caso o alvo se torna o último qubit do circuito ou o primeiro, caso o valor seja negativo.

4.3 Cálculo via AJAX

AJAX, ou JavaScript e XML assíncronos, não é uma linguagem e sim uma combinação de técnicas que utilizam ferramentas já embutidas do browser para realizar requisições ao servidor e receber respostas do mesmo de forma assíncrona, sem a necessidade de recarregar a página inteira, apenas parte dela.

A escolha do AJAX para realização do cálculo se deve ao simples fato de que o simulador tem o intuito de ser amigável ao usuário e dessa forma utilizando AJAX não existe a necessidade de recarregar a página ou ser direcionado para uma outra, o que é uma funcionalidade muito útil para o nosso sistema.

Primeiramente, existe o tratamento do circuito para um objeto de fácil acesso

à aplicação onde decidimos utilizar um JSON (JavaScript Object Notation - Notação de Objetos JavaScript) que é uma formatação de dados de fácil leitura e escrita para pessoas e de fácil interpretação e geração por computadores. JSON é composto por duas estruturas: uma coleção de pares nome/valor e uma lista ordenada de valores.

Por serem estruturas de dados comuns a todas as linguagens, de formato para troca de dados para nosso sistema foi decidido como uma boa opção [14].

Após isso colocamos os elementos (as portas pertencentes ao circuito) em um array de arrays. Também colocamos a quantidade de qubits que o circuito possui, as conexões entre as portas e enviamos para o servidor.

No servidor, por estarmos utilizando o Django, o tratamento de requisição e respostas já é realizado [4]. Dentro da view, que é considerado um controle pelo Django, bastou receber o objeto Request, fazer a leitura do JSON que foi enviado pelo AJAX e trabalhar em cima dele utilizando a biblioteca QuTip para realizar o cálculo.

A parte do cálculo do tempo, por não necessitar da biblioteca ou qualquer outra informação por parte do servidor, pode ser feita na própria página pelo JavaScript, pois as únicas informações necessárias são os inputs de tempo de cada porta e quais portas formavam o circuito.

Ao receber o retorno do servidor, o AJAX adiciona os resultados, tanto do cálculo do circuito quanto do cálculo do tempo, na área do HTML de resultados da página.

4.4 Usando QuTip

Utilizar a biblioteca QuTip se mostrou um desafio maior do que inicialmente previsto. Entretanto, após o período de familiarização inicial, ela se tornou uma excelente ferramenta. QuTip possui um objeto, o QubitCircuit, que representa um programa ou algoritmo quântico que mantém uma sequência de portas (chamados *Gates*). O QubitCircuit possui uma função add_gate com os seguintes parâmetros:

- gate - Nome da porta, dentro de uma lista aceitável. RX, RY, RZ, CRX, CRY, CRZ, SQRTNOT, SNOT, PHASEGATE, CPHASE, CNOT, CSIGN, BERKELEY, SWAPalpha, SWAP, ISWAP, SQRTSWAP, SQRTISWAP, FREDKIN, TOFFOLI, GLOBALPHASE;
- targets - Uma lista com os qubits alvos;

- controls - Uma lista com os qubits de controle;
- arg_value - Parâmetro para a transformação, do tipo Float;
- arg_label - Label para representação da porta.

Dependendo da porta desejada, alguns dos parâmetros são obrigatórios. Por exemplo, para adicionar a porta Hadamard basta qcircuit.add_gate("SNOT", targets=[0]), onde os parâmetros obrigatórios são *gate* e *targets*.

Após montarmos o objeto QubitCircuit com as portas passadas no Request, deve-se calcular a matriz de propagação utilizando a função QubitCircuit.propagators() que faz esse cálculo. Ela retorna uma lista com os passos em forma de matrizes unitárias operando da esquerda para a direita. Com essa lista é possível chamar a função gate_sequence_product() que retorna a matriz unitária final.

O objeto gerado pela função gate_sequence_product() é então multiplicado pelo vetor coluna dos qubits passados no input que inicialmente são sempre iguais a $|0\rangle$. Essa multiplicação gera um novo vetor coluna com as probabilidades dos valores finais para o circuito e é nesse vetor que é feito uma análise para retirada legível dos resultados que finalmente são retornados para o AJAX.

4.5 Hospedagem - Heroku

Como o simulador tem o objetivo de ser online utilizamos a plataforma da nuvem Heroku para a hospedagem. O Heroku é um PaaS (Platform as a Service) que faz com que o desenvolvedor não tenha que se preocupar com infraestrutura tendo assim foco total no desenvolvimento, sem precisar gerenciar, fazer a manutenção e garantir a segurança do servidor [11]. Ele possui uma opção grátis que dá suporte a diversas linguagem sendo uma delas Python, o que fez dela uma boa escolha.

A configuração do projeto para rodar no Heroku não é complicada. Basta colocar o projeto no Git, preparar um ambiente virtual com todas as dependências necessárias, gerar um arquivo texto chamado "requirements.txt" com essas dependências, o que pode ser feito com o comando "pip freeze > requirements.txt" e adicionar esse arquivo ao diretório do projeto. Devemos também criar um outro arquivo "runtime.txt" para especificar a versão do Python sendo utilizada. É necessário criar um "Procfile" que serve para declarar

o comando que será executado para começar a aplicação. No caso do nosso projeto colocamos “web: gunicorn quantum-circuit-simulator.wsgi –log-file -”.

Após essas configurações básicas é necessário criar uma conta no Heroku e instalar o Heroku CLI. Com a instalação do Heroku CLI podemos executar o comando “heroku login” e entrar com as credenciais de login. Após isso, criar uma aplicação com o comando “heroku create nome-aplicacao”, adicionar todas as alterações feitas no projeto e realizar o deploy da aplicação no Heroku “git push heroku master”.

Durante o deploy o Heroku detecta a linguagem, faz a instalação dos requisitos que estão no arquivo “requirements.txt” e roda o comando no Procfile.

Com o deploy da sua aplicação feito basta acessar a url da aplicação acessando sua conta do Heroku ou executando o comando “heroku open” [12].

4.6 Resultados

O projeto alcançou o objetivo traçado ao criar um simulador web de circuito quântico. O código gerado se encontra no GitHub: `https://github.com/iblackman/quantum-circuit-simulator`.

Na Figura 4.3 podemos ver uma parte do código HTML produzido, que cria a área dos inputs de tempo das portas, as portas a serem arrastadas, o circuito e a parte onde é mostrado o resultado.

```
<!-- load static 'e utilizado para carregar o caminho dos arquivos static -->
{% load static %}
<!DOCTYPE HTML>
<html>
    <head>
        <link rel="stylesheet" href="{% static 'simulador/css/' %}style.css" />
        <script src="{% static 'simulador/js/' %}main.js"></script>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
    </head>
    <body>
        <div class="row">
                <h2>Simulador</h2>
            <div class="left">
                <p>Preencha com o tempo de cada porta.</p>
                <!-- cria todas as portas passadas como parametro na view como variavel arrPortas -->
                <div style="">
                        {%for porta in arrPortas %}
                                <div id="porta{{ porta }}" class="porta-tempo">
                                        <img src="{% static 'simulador/media/' %}{{ porta }}.jpg" width="40" height="40">
                                        <input type="number" min="0" name="input-{{ porta }}" alt="porta {{ porta }}" required="required" value="0"/>
                                        <span>ms</span>
                                </div>
                        {% endfor %}
                </div>
            </div>
            <div class="right">
                <p>Arraste as portas para formar o circuito desejado.</p>
                    <!-- cria todas as portas passadas como parametro na view como variavel arrPortas -->
                <div class="drag-portas">
                        {%for porta in arrPortas %}
                                <div id="porta{{ porta }}" class="porta">
                                  <img src="{% static 'simulador/media/' %}{{ porta }}.jpg" draggable="true" ondragstart="drag(event)" id="drag{{ porta }}"
                                </div>
                        {% endfor %}
                </div>

                <div >
                        <p><br>Circuito<br></p>
                    <label for="link-to-circuit">Qubit a ser conectado: </label>
                    <input type="number" min=1 name="link-to-circuit" id="link-to-circuit" required="required" value="1"/>
                </div>

                <div class="container-outer" id="container-outer">
                        <div id="circuitos" class="circuitos">
                                <div class="c-container-original" id="container-original">
                                <!-- Cria um circuito q aceita o drop das portas com o tamanho passado pela view -->
                        <span>#</span>
                                {% for i in numMaxPortas %}
                                        <div name="c-porta{{ i }}" class="c-porta" ondrop="dropCopy(event)" ondragover="allowDrop(event)" value="{{ i }}">
                                                <!-- utiliza a imagem da portaDefault passada como parametro pela view -->
                                                <img src="{% static 'simulador/media/' %}{{ portaDefault }}.jpg" draggable="true" ondragstart="drag(event)"

                                        </div>
                                {% endfor %}
                                        <a class="button red" onclick="remove(event)">Remover</a>
                                </div>
                        </div>
                </div>

                <!-- Ajusta a width da div circuitos dependendo do numero maximo de portas na div container-original -->
                <script>
                    adjustCircuitWidth();
                </script>

                <div class="b-novo-circuito" >
                        <a class="button blue" onclick="duplicate()">Adcionar qubit</a>
                    <a class="button green" onclick="superdenseX()">Superdenso 01</a>
                    <a class="button green" onclick="superdenseZ()">Superdenso 10</a>
                    <a class="button green" onclick="superdenseXZ()">Superdenso 11</a>
                </div>
            </div>
        </div>
        <div class="row">
            <h2>Resultado</h2>
            <div class="left">
                <a class="button red" onclick="calcular()">Calcular</a>
            </div>
            <div class="right resultado-circuito">
                <div class="resultado">

                </div>
                <div class="matrix" style="display: none">

                </div>
                <a class="button blue show" onclick="showMatriz(this)" style="display: none">Mostrar matriz</a>
                <a class="button blue hide" onclick="hideMatriz(this)" style="display: none">Esconder matriz</a>
            </div>
        </div>
    </body>
</html>
```

Figura 4.3: Parte do HTML

Já na Figura 4.4 podemos ver uma parte do código Javascript com a função de *Drop* criada, como descrito na seção 4.2.

```
16  //drop modificado para criar uma copia do objeto que foi arrastado
17  function dropCopy(ev) {
18      ev.preventDefault();
19      //variavel de controle para voltar ao estado inicial
20      var reset = false;
21      //armazena valor anterior para caso seja cancelada a acao
22      var prevValue = $(ev.currentTarget).find("img").attr("value");
23      var preNodeCopy = $(ev.currentTarget).find("img").clone();
24      //limpa div
25      ev.currentTarget.innerHTML = '';
26
27      var data = ev.dataTransfer.getData("text");
28      var nodeCopy = document.getElementById(data).cloneNode(true);
29
30      nodeCopy.id = "newId"; /* We cannot use the same ID */
31      nodeCopy.setAttribute('draggable', false);
32      ev.currentTarget.appendChild(nodeCopy);
33
34      var nodeCopyValue = $(nodeCopy).attr("value");
35
36      var col = getElemntCol(ev.currentTarget);
37      var lin1 = getElementLin(ev.currentTarget);
38      //caso tenha q ligar
39      if(nodeCopyValue == "not" || nodeCopyValue == "sw" || nodeCopyValue == "sr") {
40          var lin2 = getCircuitToLink();
41          var pos1 = $(ev.currentTarget).position();
42
43          if(lin1 == lin2) {
44              reset = true
45              alert("Porta arrastada para o mesmo qubit da conexao, por favor mude um dos dois.");
46          } else {
47              var pos2 = $(getCircuitElement(lin2, col)).position();
48              //desenha linha
49              var valConex = lin1+"-"+lin2+"-"+col;
50              connect($(ev.currentTarget), getCircuitElement(lin2, col), "black", 1, valConex);
51              placePoint(lin2, col);
52          }
53      }
54      //caso tentou ligar com a mesma linha do circuito cancela
55      if(reset){
56          ev.currentTarget.innerHTML = '';
57          ev.currentTarget.appendChild(preNodeCopy[0]);
58      }
59
60      //se a antiga porta era uma de bit duplo deve limpar sua conexao
61      if(prevValue == "not" || prevValue == "sw" || prevValue == "sr"){
62          //tem q apagar conexao e ponto
63          limparConexao(lin1, col, 0);
64      }else if(prevValue == "point"){
65          //tem q apagar conexao e porta
66          limparConexao(lin1, col, 1);
67      }
68
69  }
70  //adiciona um novo circuito, que 'e apenas uma copia do circuito original
71  function duplicate() {
72      var original = document.getElementById('container-original');
73      var clone = original.cloneNode(true);
74
75      //para aumentar o tamanho da div para mostrar o novo circuito
76      increaseContainer($('.c-container-original').outerHeight());
77
78      var divCircuitos = document.getElementById('circuitos');
79      var index = divCircuitos.childElementCount - 1;
80      clone.id = 'container' + index;
81      clone.className = 'c-container';
82      divCircuitos.appendChild(clone);
83      //ajusta numeracao
84      renumeraCircuitos();
85  }
```

Figura 4.4: Parte do Javascript

Finalmente, na Figura 4.5 está sendo ilustrado uma parte do código Python que faz o cálculo do circuito quântico montado, gerando o resultado do qubit final.

```python
# funcao de ajax para processar o circuito
@csrf_exempt  # para receber os dados por post
def calcular(request):
  print("****** Inicio ******")
  json_data = json_loads_byteified(request.body)

  numBit = json_data['len']

  connections = prepareConnections(json_data['connections'].iteritems())
  ###
  #  execution using qubitcircuit
  ###
  qcircuit = QubitCircuit(numBit)

  #arrumando ordem dos dados para inserir os gates corretamente
  #os circuitos estavam vindo em ordem estranha
  circuitList = []
  for key, gates in json_data['data'].iteritems():
    circuitList.insert(int(key),gates)


  for i in range(0,len(circuitList[0])):
    for j in range(0,numBit):
      gate = circuitList[j][i]
      if gate != "line":
        if(connections.has_key(str(i))):
          connDict = connections.get(str(i))
          auxKey = j + 1
          targetAux = connDict.get(str(auxKey))
          if targetAux != None:
            target = j
            control = int(targetAux)-1
            if gate == "not":
              qcircuit.add_gate("CNOT", targets=[target], controls=[control])
            elif gate == "sw":
              qcircuit.add_gate("SWAP", targets=[target,control])
            elif gate == "sr":
              qcircuit.add_gate("SQRTNOT", targets=[target], controls=[control])
        else:
          if gate == "x":
            qcircuit.add_gate("RX", j, arg_value=-pi)
          elif gate == "y":
            qcircuit.add_gate("RY", j, arg_value=pi)
          elif gate == "z":
            qcircuit.add_gate("RZ", j, arg_value=pi)
          elif gate == "h":
            qcircuit.add_gate("SNOT", j)


  result = resultInitTransp(numBit)

  qInput = initializeInput(numBit)


  U0 = gate_sequence_product(qcircuit.propagators())

  qFinal = U0*qInput

  msg = resultToText(numBit, qFinal.full())
  msg += "<br/>"
  #coloquei isso prq nao quero trazer o resultado com o numero complexo, entao o result ja e suficiente
  msg =  ""
  if(U0 == 1):
    msg += htmlify_matrix(qeye(2**numBit))
  else:
    msg += htmlify_matrix(U0)
  ###Fim teste

  prepareResultJSON(result)
  data = {
    'msg' : msg,#str(htmlify_matrix(U0)),
    'result' : resultToTextSimple(numBit, qFinal.full())
  }
  print("**** Fim *****")
  #return json.dumps(data)
  return JsonResponse(data)
```

Figura 4.5: Parte do programa em Python

Capítulo 5

Conclusão e Trabalhos Futuros

O trabalho teve como objetivo a implementação de um simulador de circuitos quânticos que executa algoritmos quânticos em um computador clássico, recebendo alguns valores de entrada para estimar o tempo gasto por cada porta quântica. Assim, o simulador consegue emular o que seria obtido em um computador quântico caso ele seja construído emulando sobreposição de estados e entrelaçamento. O simulador dispões de algumas funcionalidades como arraste das portas para a construção do circuito, criação de circuitos já preparados, além de disponibilidade online. Isso tudo com o intuíto de facilitar a interação com o usuário.

O projeto providenciou um enorme aprendizado em duas áreas que não estão relacionadas e que não são obrigatórias na grade curricular do aluno, Computação Quântica e Desenvolvimento Web. Na parte do desenvolvimento web tivemos a oportunidade de estudar a linguagem de programação Python e um de seus Frameworks para desenvolvimento web, o Django, e realizar deploy na plataforma na nuvem Heroku. E quanto a parte de computação quântica estudamos a biblioteca QuTip para realização dos cálculos além dos algoritmos quânticos e seu funcionamento.

Como trabalhos futuros pode-se implantar novas portas, melhora do sistema de Drag and Drop principalmente no caso das portas de dois qubits, melhora no design do simulador com a ideia de torná-lo mais atrativo ao usuário, e alterar a forma como os resultados são mostrados para facilitar o entendimento do usuário. O sistema pode ser expandido, adicionando novas funcionalidades como leitura de um circuito a partir de algum tipo de arquivo, e a possibilidade de salvar um circuito já montado.

Referências Bibliográficas

[1] Abigail Beall. Quantum computing: what is it and how does it differ to classical computing | wired uk. http://www.wired.co.uk/article/quantum-computing-explained, May 2017.

[2] Ryan Brown. Django vs flask vs pyramid: Choosing a python web framework. https://www.airpair.com/python/posts/django-flask-pyramid, May 2017.

[3] Django. Django afirma ser um framework mvc, mas ele chama o controller de <view> e a view de <template>. por que não usar os nomes padrões? https://docs.djangoproject.com/pt-br/1.11/faq/general/, June 2017.

[4] Django. Request and response objects. https://docs.djangoproject.com/en/1.11/ref/request-response/, June 2017.

[5] Django. The web framework for perfectionists with deadlines | django. https://www.djangoproject.com/, June 2017.

[6] Python Software Foundation. Quotes about python | python.org. https://www.python.org/about/quotes/, May 2017.

[7] Craig Gidney. Git project quirk. https://github.com/Strilanc/Quirk, April 2017.

[8] Craig Gidney. My quantum circuit simulator: Quirk. http://algassert.com/2016/05/22/quirk.html, April 2017.

[9] GitHub. Github. https://github.com/, May 2017.

[10] John Gribbin. *In Search of Schrodinger's Cat: Quantum Physics And Reality*. Random House Publishing Group, May 2011.

[11] Heroku. Cloud application platform | heroku. https://www.heroku.com/, June 2017.

[12] Heroku. Getting started on heroku with python. https://devcenter.heroku.com/articles/getting-started-with-python, June 2017.

[13] The jQuery Foundation. Jquery. https://jquery.com/, June 2017.

[14] JSON. Introducing json. http://www.json.org/, June 2017.

[15] Grover L.K. *A fast quantum mechanical algorithm for database search.* Proceedings, 28th Annual ACM Symposium on the Theory of Computing, May 1996.

[16] N. David Mermin. *Quantum Computer Science, An Introduction.* Cambridge University Press, Cornell University, New York, USA, September 2007.

[17] Leonardo Gresta Paulino Murta. Utilizando servlets e jsp em conjunto. http://www2.ic.uff.br/~leomurta/past/2015.1/dw/aula13.pdf, June 2017.

[18] P.D. Nation and J.R. Johansson. Qutip - change log. https://github.com/qutip/qutip-doc/blob/master/changelog.rst, May 2017.

[19] P.D. Nation and J.R. Johansson. Qutip - quantum toolbox in python. http://http://qutip.org/, April 2017.

[20] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, Cornell University, New York, USA, 2000.

[21] Numpy. Numpy. http://www.numpy.org/, June 2017.

[22] Stack Overflow. Tags - stack overflow. https://stackoverflow.com/tags, May 2017.

[23] Pylons Project. Pylons framework (deprecated) | pylons project. http://pylonsproject.org/about-pylons-framework.html, May 2017.

[24] Pyqu. pyqu - python project - developer fusion. www.developerfusion.com/project/42536/pyqu/, June 2017.

[25] Pyramid. Welcome to pyramid, a python web framework. https://trypyramid.com/, May 2017.

[26] Qitensor. qitensor for quantum information in python. http://www.stahlke.org/dan/qitensor, June 2017.

[27] Qubiter. Python tools for reading, writing, compiling, simulating quantum computer circuits. https://github.com/artiste-qb-net/qubiter, June 2017.

[28] Django sites. Latest additions :: Djangosites.org - powered by django. https://www.djangosites.org/, June 2017.

[29] IBM Quantum team. Ibm q - quantum experience. https://quantumexperience.ng.bluemix.net/, July 2017.

[30] Davy Wybiral. Git project quantum circuit simulator. https://github.com/wybiral/quantum, April 2017.

[31] Davy Wybiral. Quantum circuit simulator. http://www.davyw.com/quantum/, April 2017.