1999

A visualization of a quantum mechanical search algorithm

Terrance Bing-Parks
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Bing-Parks, Terrance, "A visualization of a quantum mechanical search algorithm" (1999). *Master's Theses*. 1795.

DOI: <https://doi.org/10.31979/etd.qhcx-nmh5>
https://scholarworks.sjsu.edu/etd_theses/1795

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

A VISUALIZATION
OF A
QUANTUM MECHANICAL SEARCH ALGORITHM

A Thesis
Presented to
The Faculty of the Department of Mathematics and Computer Science
San Jose State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Terrance Bing-Parks

May 1999

UMI Number: 1394507

UMI Microform 1394507

Copyright 1999, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized
copying under Title 17, United States Code.

UMI

300 North Zeeb Road
Ann Arbor, MI 48103

© 1999

Terrance Bing-Parks

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF
MATHEMATICS AND COMPUTER SCIENCE

Walter W. Kirchherr

Dr. Walter Kirchherr

David J. Hayes

Dr. David Hayes

Sami Khuri

Dr. Sami Khuri

APPROVED FOR THE UNIVERSITY

William Fisher

ABSTRACT

A VISUALIZATION
OF A
QUANTUM MECHANICAL SEARCH ALGORITHM

by Terrance Bing-Parks

In this thesis work, the conceptual foundations of quantum computers and quantum computing are explored. The notational and mathematical prerequisites of quantum computing are developed. A computer program that simulates a quantum algorithm developed by Lov K. Grover is implemented.

Grover [8, 9, 10] has shown that we can find an item in an unstructured search space of size N with a probability of success that is at least 50% in $O(\sqrt{N})$ steps. Boyer et al. [2] refined Grover's original analysis and demonstrated that Grover's search algorithm can achieve a success rate that is at least $1 - 1/N$ in approximately $\frac{\pi}{4}\sqrt{N}$ steps. The program presented in this work provides a visualization of the abstract representations and mathematical derivations of Grover's search algorithm.

The program is written in Java Version 1.1.4. User interface documentation describes how to run the animation. There is also documentation that describes the implementation of the program and verifies the program's correctness.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Dr. Walter Kirchherr, for inspiring me to think big thoughts, for his guidance throughout this work, and for his patience.

I would like to thank Dr. David Hayes for being an excellent academic advisor, for his encouraging words, and for taking the time to participate on my committee.

I would like to thank Dr. Sami Khuri for teaching me how to visualize this thesis, for his valuable suggestions on how to improve this work, and for taking the time to participate on my committee.

I would like to thank Eric Christopher for helping me refine this work.

I would like to thank Cathy Bautista for her friendship and support through the good dreams and the nightmares.

Contents

CHAPTER 1 INTRODUCTION	1
CHAPTER 2 MATHEMATICAL BACKGROUND	2
2.1 INTRODUCTION	2
2.2 OBSERVABLES	2
2.3 THE STATE OF THE SYSTEM.....	3
2.4 BRA AND KET NOTATION	5
2.5 SCHRÖDINGER EQUATION	5
2.6 THE SUPERPOSITION PRINCIPLE.....	6
2.7 KRONECKER PRODUCT	8
CHAPTER 3 QUANTUM COMPUTING.....	9
3.1 INTRODUCTION	9
3.2 QUBITS	9
3.3 QUANTUM GATES	11
3.4 QUANTUM COMPUTATION.....	12
CHAPTER 4 GROVER'S ALGORITHM.....	14
4.1 INTRODUCTION	14
4.2 ABSTRACT OF THE PROBLEM THAT GROVER'S ALGORITHM ADDRESSES.....	14
4.3 QUANTUM OPERATIONS	15
4.4 THE UNITARY MATRIX M	15
4.5 THE ROTATION TRANSFORM	16
4.6 ALGORITHM FOR EXHAUSTIVE SEARCH STARTING FROM THE 0 STATE	17
4.7 INVERSION ABOUT THE AVERAGE.....	18
CHAPTER 5 USER GUIDE FOR THE ANIMATION	24
5.1 OVERVIEW OF THE ANIMATION	24
5.2 SYSTEM REQUIREMENTS	26
5.3 INSTALLING THE ANIMATION	27
5.4 RUNNING THE ANIMATION	27
5.4.1 INTRODUCTION	27
5.4.2 THE ANIMATION SPEED SELECTION BOX	28
5.4.3 THE QUBIT SELECTION BOX	28
5.4.4 THE RUN BUTTON	28
5.4.5 THE TRACE BUTTON.....	30
5.4.6 THE DOCUMENTATION BUTTON.....	31
CHAPTER 6 TRACE OF THE ANIMATION'S EXECUTION	33
6.1 INTRODUCTION	33
6.2 THE NUMBER OF ITERATIONS OF Q TO PERFORM.....	34
6.3 THE START STATE	35
6.4 INITIAL APPLICATION OF THE W (WALSH-HADAMARD TRANSFORM).....	35
6.5 APPLICATION OF I_i	36
6.6 SECOND APPLICATION OF W	37

6.7 APPLICATION OF $-I_s$	39
6.8 FINAL APPLICATION OF W	40
6.9 SUMMATION	42
CHAPTER 7 VERIFICATION OF THE ANIMATION'S CORRECTNESS	44
7.1 INTRODUCTION	44
7.2 TWO QUBITS	44
7.3 THREE QUBITS	45
7.4 FOUR AND FIVE QUBITS	46
CHAPTER 8 CONCLUSION	55
REFERENCES	57
APPENDIX A MATHEMATICAL DEFINITIONS	60
A.1 VECTOR SPACES	61
A.2 EIGENVECTORS AND EIGENVALUES	62
A.3 LINEAR OPERATORS	62
A.4 HERMITIAN OPERATORS	62
A.5 UNITARY MATRICES	62
A.6 HILBERT SPACE	62
APPENDIX B GROVER_ANIMATION.JAVA	63
APPENDIX C GROVER_ANIMATION.HTML	119
APPENDIX D VERIFICATION.TXT	121

List of Figures

FIGURE 1	THE PRINCIPLE OF SUPERPOSITION	6
FIGURE 2	SYSTEM BEFORE APPLYING INVERSION ABOUT THE AVERAGE	19
FIGURE 3	SYSTEM AFTER APPLYING INVERSION ABOUT THE AVERAGE.....	20
FIGURE 4	SYSTEM AFTER INVERSION OF THE TARGET STATE	22
FIGURE 5	THE SYSTEM AFTER APPLYING INVERSION ABOUT THE AVERAGE	22
FIGURE 6	PROBABILITY OF SUCCESS VERSUS THE NUMBER OF ITERATIONS OF EQUATION 17.....	23
FIGURE 7	THE AMPLITUDES OF A 2-QUBIT QUANTUM SYSTEM AFTER APPLYING W	24
FIGURE 8	THE AMPLITUDES OF A 2-QUBIT QUANTUM SYSTEM AFTER APPLYING I_1	25
FIGURE 9	THE AMPLITUDES OF A 2-QUBIT QUANTUM SYSTEM AFTER ALL ITERATIONS OF Q	26
FIGURE 10	THE ANIMATION'S INITIAL SCREEN	27
FIGURE 11	THE INITIAL DIALOG BOX.....	29
FIGURE 12	SCREEN SHOT OF THE ANIMATION IN RUN MODE.....	29
FIGURE 13	SCREEN SHOT OF THE ANIMATION IN TRACE MODE.....	31
FIGURE 14	THE ANIMATION'S DISPLAY AFTER THE INITIAL APPLICATION OF W	36
FIGURE 15	THE ANIMATION'S DISPLAY AFTER APPLICATION OF I_1	37
FIGURE 16	THE ANIMATION'S DISPLAY AFTER THE SECOND APPLICATION OF W	39
FIGURE 17	THE ANIMATION'S DISPLAY AFTER APPLICATION OF I_1	40
FIGURE 18	THE ANIMATION'S DISPLAY AFTER THE FINAL APPLICATION OF W	41
FIGURE 19	THE ANIMATION'S DISPLAY AFTER ALL STEPS OF THE ALGORITHM ARE COMPLETE	42
FIGURE 20	THE ANIMATION'S PREDICTION FOR LOCATION 2	44
FIGURE 21	ANIMATION'S PREDICTION FOR LOCATION 5	45

Chapter 1 Introduction

Computers are physical devices; therefore, the laws of physics can be used to describe their behavior [18]. Hence, as computer circuits are reduced to the atomic and molecular level, their behavior must be described by the rules of quantum mechanics [18, 20]. Quantum computing is a new approach to the discipline of computer science. It explores the implications of the merging of quantum mechanics, computers, and computing [23].

Today only a handful of primitive quantum computers exists. They perform trivial calculations (like adding $1 + 1$) or a single task [23]. However, there has been some progress on the theoretical level. Lov K. Grover has developed an algorithm that uses quantum mechanical principles to find a particular record in an unsorted database. In a database containing N records, Grover's algorithm can find a particular record in $O(\sqrt{N})$ steps. This compares to the $O(N)$ steps that would be required to find the record on a classical computer [8, 9, 10]. Grover's algorithm is of particular interest because it has been successfully demonstrated on a physical device [4].

This thesis will provide an introduction to the concepts and notations used in quantum computing and then give concrete meaning to these ideas by developing an animation of Grover's search algorithm.

Chapter 2 Mathematical Background

2.1 Introduction

Quantum theory is used to describe the nature and interaction of things at the molecular, atomic, and particle level [20]. We can mathematically model quantum theory using special types of vector spaces called Hilbert spaces.¹ This chapter gives an overview of basic quantum mechanical concepts and notations that are used in quantum computing. This work concentrates on the mathematical structure of quantum theory from the matrix point of view.² A more thorough accounting of these ideas can be found in [11].

2.2 Observables

When we measure quantities such as position or momentum in quantum systems, we can never be completely sure about the accuracy of our measurements. The act of measuring a quantum system disturbs the system and introduces uncertainty as to the measurement [20]. Therefore, in quantum theory, we call physical quantities of the system, like position or momentum, *observables*.

In the matrix representation of quantum mechanics, we represent observables with Hermitian operators in Hilbert space. If the Hermitian operator A admits eigenvectors, then the observable only has a discrete number of possible values. Namely, the eigenvalues of A . If the Hermitian operator A does not admit eigenvectors, the observable has a continuum of possible values [11]. Although Hilbert space is linear

¹ See Appendix A for definitions of vector spaces and Hilbert spaces.

² An accessible introduction to quantum theory that includes wave mechanics and experimental results can be found in [19].

vector space over the field of complex numbers [23], the eigenvalues of Hermitian operators are real numbers [11, 20]. Thus, the values obtained from the Hermitian operator will produce sensible values [20].

In this work, we are only concerned with quantum systems that when measured are capable of having only a limited number of possible values. That is, quantum systems whose Hermitian operators admit eigenvectors. We require this restriction because we need a stable physical environment in which to store bits [23].

2.3 The State of the System

We say that a system is in a given state when we have as much information as possible about the system's motion [20]. In the matrix representation of quantum mechanics, the state of a physical system is represented by a vector ψ in Hilbert space [11]. The elements of this vector are complex numbers. Over time, this system is deterministic unless the system is somehow disturbed by external factors, such as measurement. We are restricting our focus to systems whose Hermitian operators admit eigenvectors; therefore, the eigenvalues of the eigenvectors of the system's operator are the only values that we can obtain when we measure the system [11].

In classical mechanics, if we are given enough information about a system, we can predict what the resulting state of the system should be. Hughes [11, p. 61] showed that we could view the state of a system as being a two-value function,

$$W(A, \Delta). \tag{1}$$

In Equation 1, Δ is some subset of the real numbers, and A is a Hermitian operator. This function returns true or false to the question—“Does the observable represented by A have a value in Δ ?”

However, in quantum mechanics, there is always some uncertainty as to the state of the system. The Heisenberg Uncertainty Principle tells us that we cannot exactly measure certain pairs of observables at the same time. For example, we cannot simultaneously measure both the position and momentum of a particle [20]. Let Q represent the position operator for a particle and P represent the momentum operator of that particle. If we knew the position and momentum of the particle, the particle would simultaneously have to be in an eigenstate of Q and P [20]. This would only be possible if Q and P commute. Operators Q and P are said to commute if and only if $QP - PQ$ is the zero matrix [23]. However, in the matrix approach to quantum mechanics Q and P do not commute [20].

This is illustrated by Heisenberg’s famous equation [13]

$$QP - PQ = \frac{ih}{2\pi}. \quad (2)$$

In Equation 2, Q and P are as previously defined, $i = \sqrt{-1}$, and h is Planck’s constant which is $h \cong 6.626 \times 10^{-27} \text{ g} \cdot \text{cm}^2 / \text{s}$. Thus, there is a non-zero difference between QP and PQ [20]. Equation 2 is a specific instance of the Heisenberg Uncertainty Principle.

In general, we cannot be certain of the state of a quantum system. Therefore, we cannot generally know beforehand the result of Equation 1 for a quantum system. Instead of knowing if some observable, as given by the Hermitian operator A , is within Δ ,

we only know the probability that A will yield one of the eigenvalues of the eigenvectors that A admits [11].

For example, let ϑ be an eigenvector with corresponding eigenvalue α . The probability that a measurement of some observable, as given by some Hermitian operator A , conducted on a system in state ψ will yield result α is $|\langle\psi, \vartheta\rangle|^2$ [15].

2.4 Bra and Ket Notation

A state vector in the matrix representation of quantum mechanics can be written with a special notation called a *ket* vector. For our purposes, the ket vector $|\psi\rangle$ can be viewed as a column vector. Likewise, the *bra* vector $\langle\psi|$ can be viewed as a row vector. Using this notation, we can compactly represent the product of a row and column vector by $\langle\psi|\psi\rangle$.

We call $\langle\psi|\psi\rangle$ a *bracket* [23]. If X represents a scalar, then the notation $X|\psi\rangle$ represents the column vector ψ multiplied by X .

2.5 Schrödinger Equation

If a quantum system is not disturbed, the system is governed by the Schrödinger equation, which is

$$i\hbar \frac{\partial |\psi(t)\rangle}{\partial t} = H(t)|\psi(t)\rangle, \quad \hbar = 1.054 \times 10^{-34} \text{ Js}, \quad i = \sqrt{-1}. \quad (3)$$

$H(t)$ in Equation 3 is called the Hamiltonian operator. This operator is related to the total energy of the system. When the system is unobserved, the system will undergo smooth continuous change that is governed by Equation 3. However, when the quantum

system is disturbed (e.g., by measurement), it will then jump unpredictably to one of the system's eigenstates [23].

2.6 The Superposition Principle

In the matrix approach to quantum theory, we call normalized vectors in Hilbert space a *pure state* [11]. For our purposes, we follow the principle of superposition in terms of vectors representing pure states. Hughes [11, p. 92] states this principle as follows:

“If v_1 and v_2 represent possible pure states of a system, then any vector $v_3 = c_1 v_1 + c_2 v_2$ such that $|v_3| = 1$ also represents a possible pure state of that system.”

Superposition is a distinguishing feature of quantum mechanical systems as opposed to deterministic systems. For example, Figure 1 shows a 2-dimensional system where there are two possible outcomes. Those outcomes are represented by the axes outcome 1 and outcome 2, respectively.

Figure 1. The principle of superposition.

In a deterministic system, the normalized vectors σ and τ in Figure 1 represent pure states, but the third vector υ cannot. However, in a quantum mechanical system, vector υ

can also be a pure state if $\upsilon = (\frac{1}{\sqrt{2}})\sigma + (\frac{1}{\sqrt{2}})\tau$ [11].

Moreover, suppose that some observable represented by Hermitian operator A has two possible values which can be observed at time t — say either α or β . If the probability of α is p_1 and the probability of β is p_2 such that $p_1 + p_2 = 1$, the system is in superposition [15]. That is [15],

$$\psi = (\sqrt{p_1}|\alpha\rangle + \sqrt{p_2}|\beta\rangle). \quad (4)$$

Consequently, the principle of superposition allows a quantum system to exist in a blend of possible states [23]. Thus, if ψ can exist in any one of its eigenstates $|\psi_1\rangle, \dots, |\psi_n\rangle$, then it can exist in the superposed state [23]

$$|\psi\rangle = \sum_{i=1}^n w_i |\psi_i\rangle, \quad (5)$$

where w_i is a weighting coefficient.

Although the coefficient w_i in Equation 5 appears to correspond to the probability of the system being in a given eigenstate, the coefficient w_i is a complex number; therefore, $|w_i|^2$ may not be between zero and one. Hence, the w_i coefficient must be rescaled. After rescaling the w_i coefficient in Equation 5, we have [23]:

$$Probability(|\psi_i\rangle) = \frac{|w_i|^2}{\sum_{i=1}^n |w_i|^2}. \quad (6)$$

The weighting coefficient w_i , is known as a *probability amplitude* or simply an *amplitude*. When we deal with normalized state vectors, we can simplify Equation 6 to

$$Probability(|\psi_i\rangle) = |w_i|^2. \quad (7)$$

For a simple 2-state quantum system like Figure 1, we can think of vectors in superposition, like vector ψ in Figure 1, as rotations to various positions in the subspace spanned by the eigenvectors of the Hermitian operator that represent some observable. We refer to these rotations as the *phase* of the vector.

2.7 Kronecker Product

The matrix representation of quantum mechanics and Grover's algorithm involve a significant amount of matrix multiplication. The Kronecker product is particularly well suited for the types of operations that we need to perform. In general, the Kronecker product of matrices $A = (a_{ij})_{m,n}$ and $B = (b_{ij})_{p,q}$ ($A \otimes B$) is the $mp \times nq$ matrix that is obtained from A when we replace every element a_{ij} with the block $a_{ij}B$ [14]. For example,

$$\begin{aligned} \begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} &= \begin{bmatrix} a \times \begin{bmatrix} c \\ d \end{bmatrix} \\ b \times \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}. \end{aligned} \quad (8)$$

Chapter 3 Quantum Computing

3.1 Introduction

This chapter explores how quantum theory can be used for computational applications. In this work, we restrict our explorations to simple 2-state quantum systems that can be used to model classical bits. An example of such a quantum system is the spin- $\frac{1}{2}$ particle [23]. This particle exists in one of only two possible states — spin-up or spin-down. Thus, we could use the spin-up state to represent the binary value of 0 and the spin-down state to represent the binary value of 1. In fact, we can use any 2-state quantum system to represent the binary values of 0 and 1. We call such representations of bits qubits [23]. We develop these ideas further in the following sections.

3.2 Qubits

A qubit represents a vector in 2-dimensional complex vector space relative to a certain basis. We generally choose the basis $\{|0\rangle, |1\rangle\}$ [21]. Thus, we can represent the 0 bit by the vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and the 1 bit by the vector $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Since qubits can be placed in superposition, we would assume that a single qubit could be encoded to hold an infinite amount of information. Although we can place a qubit in superposition, the information contained in the superposition can only be extracted by measurement. Yet, once we measure the qubit, the qubit is forced either to state $|0\rangle$ or to state $|1\rangle$. Moreover, these qubits cannot be cloned (copied) so we cannot measure a qubit

in two different ways [21]. Therefore, a qubit can only be encoded to hold the same amount of information as a classical bit [21]. The key to quantum computing is to somehow skew the state vectors of the system so that we can exploit the properties of superposition, but still get the desired results when we measure.

The potential power of quantum computing can be seen by examining how the state spaces of qubits combine. Rieffel et al. [21] showed that in classical computing individual state spaces of n particles combine by means of the cartesian product. In contrast, in quantum systems, individual state spaces of n particles combine by means of the Kronecker product.

For example, if V and W are two state vectors in a classical system, the dimension of the state space $V \times W$ is as follows [21]:

$$\dim(V \times W) = \dim(V) + \dim(W). \quad (9)$$

In contrast, if X and Y are two state vectors in a quantum system, the dimension of the state space of $X \otimes Y$ is as follows [21]:

$$\dim(X \otimes Y) = (x_1 \otimes y_1, \dots, x_n \otimes y_n) = \dim(X) \times \dim(Y). \quad (10)$$

In sum, classical state spaces combine additively, and quantum state spaces combine multiplicatively. Thus, a simulation of a quantum system is computationally expensive since we must keep track of state spaces that grow multiplicatively [21]. This is particularly relevant for us since this work seeks to simulate a quantum algorithm on a

classical computer. For this reason, the simulation implemented by this work only uses a restricted range of qubits.³

3.3 Quantum Gates

A quantum gate is simply a specification of how the logical qubits from the input will be outputted as logical qubits. Quantum gates exchange information with each other by sharing the same physical qubit or by field interactions [23]. As we have seen (see Section 2.5), unobserved quantum systems are governed by Schrödinger's equation.

For our purposes, we assume that the Hamiltonian in Schrödinger's equation (Equation 3) is time independent and that our quantum computer is always started with its memory register in the state $|\psi(0)\rangle$. Given these conditions, the general solution to Schrödinger's equation is given by [23]:

$$|\psi(t)\rangle = e^{-i\hat{H}t/\hbar}|\psi(0)\rangle = \hat{U}(t)|\psi(0)\rangle. \quad (11)$$

We call $\hat{U}(t) \equiv e^{-i\hat{H}t/\hbar}$ in Equation 11 the evolution operator; moreover, $\hat{U}(t)$ is a unitary matrix [23].

Since $\hat{U}(t)$ is unitary, operators that evolve our quantum system must preserve orthogonality [21]. That is, these operators must be reversible [23]. Unitary transformations preserve orthogonality of quantum systems. Thus, quantum gates must be unitary. Intuitively, unitary transformations are rotations in the complex vector space [11, 23].

³ The program implemented by this work accepts 2 to 5-qubits as input.

Since quantum gates are unitary transformations, we can describe quantum gates by unitary matrices [23]. For example, one of the most important quantum gates (transforms) is the Walsh-Hadamard (W-H) transform. The W-H transform is [9]

$$W = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (12)$$

When Equation 12 is applied to n qubits, it creates a superposition of all 2^n possible states. Moreover, this result can also be viewed as the binary representation of the numbers 0 to $2^n - 1$ [21].

3.4 Quantum Computation

Rieffel et al. [21] have shown that there exists a set of quantum gates that can achieve any desired quantum transformation. Moreover, Rieffel et al. [21] also have shown that any computable function $\gamma(x)$ that can be performed on a classical computer can be computed with some combination of quantum gates. We call a set of quantum gates that implement $\gamma(x)$ a quantum gate array U_γ . Following the notations used by Rieffel et al. [21], U_γ is

$$U_\gamma |x, y\rangle \rightarrow |x, y \oplus \gamma(x)\rangle. \quad (13)$$

In Equation 13, the symbol $\oplus$ denotes the bitwise exclusive-OR. Since $\gamma(x) \oplus \gamma(x) = 0$, U_γ is unitary ($U_\gamma U_\gamma^T = I$) [21]. $\gamma(x)$ can be computed by applying U_γ to $|x, 0\rangle$ [21].

The real power in quantum computing occurs when we apply U_γ to input that is in superposition. Since U_γ is a linear transformation, U_γ is applied to all basis vectors in

the superposition simultaneously. Therefore, $U\gamma$ will generate a superposition of the results in a single application. As a result, we can compute multiple values of $\gamma(x)$ with a single application of $U\gamma$ [21]. This is called *quantum parallelism*. Quantum parallelism allows us to simultaneously work with 2^n states; however, this requires only n qubits.

Although we can compute multiple values of $\gamma(x)$ with a single application of $U\gamma$, we can only read one value when we measure the system (see Section 2.5). Thus, we must somehow skew quantum parallelism so that when we measure we get the desired results. Grover describes an elegant method for amplifying a desired state in an unstructured quantum search space [8, 9, 10].

The remainder of this thesis is devoted to describing Grover's algorithm and then discussing the implementation of an animation of Grover's algorithm.

Chapter 4 Grover's Algorithm

4.1 Introduction

In [9], Grover presented a framework for the design and analysis of quantum mechanical algorithms. Two algorithms were presented in [9]: one concerned solving search problems by quantum mechanical methods, and the other dealt with calculating the mean and median of statistical distributions by quantum mechanical methods. This thesis implements an animation of the search algorithm discussed in [9].

We know that quantum computers can do anything that a classical computer can [1]. However, the driving force behind quantum computing is the belief that quantum computers can do things that classical computers cannot [7]. Grover presents an algorithm that makes the probability that a system reaches a certain target state increase quadratically. This is done by continued iterations of certain quantum mechanical operations [8, 9, 10].

4.2 Abstract of the Problem that Grover's Algorithm Addresses

We are given a function $f(x)$, where $x = 1, 2, \dots, N$, that is known to be non-zero at exactly one value. The problem is to find the value where $f(x)$ is non-zero. It is clear that a classical computer would on average take $N/2$ steps to find the value where $f(x)$ is non-zero [8, 9, 10].

Zalka [24] performed a detailed analysis of Grover's algorithm and proved that Grover's algorithm provides the best possible performance of any quantum algorithm for an unstructured search. Grover's algorithm uses the W-H transform (see Section 3.3) and

a transform to selectively invert certain states to find the value at which $f(x)$ is non-zero in $O(\sqrt{N})$ steps [8, 9, 10].

4.3 Quantum Operations

Quantum computers depart from their classical counter parts in their use of bits. The bits (i.e., the qubits) used by quantum computers are in superposition. That is, the system described by a quantum computer does not start out in a particular state. Rather, the state of the system is distributed over various states with a certain probability of being in a particular state. As the system evolves, there is a certain probability that the system will be in any particular state [8, 9, 10].

Grover [8, 9, 10] showed that we need an amplitude matrix and a transition matrix to track the evolution of the system. The entries in the amplitude matrix are complex; moreover, the square of the absolute values of the entries in this matrix gives the probability that the system will be in any given state. The transition matrix's entries are also complex.

The essence of Grover's algorithm is to apply two elementary unitary operations – the W-H transformation and a selective rotation of the amplitudes of certain states [8, 9, 10]. These two transforms are applied in such a way that we magnify the probability of being in a state containing the value that we are searching for, and at the same time decrease the probability of measuring all other states [8, 9, 10].

4.4 The Unitary Matrix M

A primary operation in quantum computing is to apply a unitary transform to a single qubit such that the qubit is placed in superposition. Grover [9] calls this unitary matrix M , which is

$$M \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (14)$$

If we apply M to the qubit that represents state 0, $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, the qubit is transformed into the superposition $\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$. Likewise, after applying M to the qubit which represents the state 1, $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$, we get the superposition $\left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right)$. For example, applying M to $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is expressed mathematically by:

$$\begin{aligned} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \end{aligned} \quad (15)$$

A system that contains n qubit has $N = 2^n$ basis states. We can change the state of the system by applying Equation 14 to each qubit in the n qubit system. The matrix resulting from this operation is of size $2^n \times 2^n$. This is the transition matrix, and it contains the superposition of all possible states with amplitudes of $\pm 2^{-\frac{n}{2}}$. In general usage, this transform is called the W-H transform, but Grover [9] calls this transform $\mathcal{W}$.

4.5 The Rotation Transform

Grover [9] uses a second transform to selectively rotate the phase of the amplitude of certain states. For a 2-state system, the transform is

$$\begin{bmatrix} e^{i\phi_1} & 0 \\ 0 & e^{i\phi_2} \end{bmatrix}, \quad (16)$$

with $i = \sqrt{-1}$ and ϕ being some real number.

Equation 16 is the selective rotation matrix. This matrix allows us to design a circuit that will selectively invert the amplitudes of all states where $f(x) = 1$, while leaving the amplitude in the other states unchanged [9].

4.6 Algorithm for Exhaustive Search Starting from the 0 State

I_x denotes the transform that inverts the amplitude in a single state x [9]. I_x is the diagonal matrix with all diagonal terms equal to 1 except the xx term –

which is equal to -1 . Suppose that there are $N = 2^n$ states to be searched, starting from the 0 state. We want to reach the target state t that contains $f(x)$ such that $f(x)$ is non-zero. First, we apply the W transform from Equation 14 to the start state (i.e., the 0 state). Second, we repeatedly apply

$$Q = -I_0 W I_t W \quad (17)$$

to the superposition of the start state. Lastly, we apply a single application of W to the system. Grover [9] showed that we can reach the target state with certainty in $O(\sqrt{N})$ steps.

Grover [9] demonstrated this by showing that the operator Q transforms any superposition of vectors v_{start} and $(W^{-1}v_{tar})$ into another superposition of two vectors (note: tar denotes the target state). Grover [9] has proved that this transform preserves

the 2-dimensional vector space spanned by the two vectors. Moreover, Grover [8, pp. 3-4] proved that starting with v_{start} and after x repetitions of Q , we get the superposition

$$a_s v_{start} + a_t W^{-1} v_{tar}. \quad (18)$$

In Equation 18, $a_s = \cos(2x|W_{tar, start}|)$ and $|a_t| = |\sin(2x|W|)|$. Hence, if we apply the transform Q , x times, where $x = \frac{\pi}{W_{tar, start}}$, we get the superposition $W^{-1} v_{tar}$. If we apply

W to this result, we get v_{tar} . Therefore, in $O\left(\frac{1}{W_{tar, start}}\right)$ steps, we can start with the start state s and reach the target state with certainty [9].

4.7 Inversion about the Average

Repeated application of the Q operator is the key to Grover's search algorithm [21]. We can rearrange Equation 17 by shifting the negative sign and parentheses so that iterations of Q can be viewed as repeated applications of

$$-W I_0 W \quad (19)$$

and

$$I_t. \quad (20)$$

Moreover, Grover [9] demonstrated that the sequence described by Equation 19 is what is known as *inversion about the average*.

Inversion about the average is a unitary transform. This transform when applied to a particular state vector will either decrease or increase the vector's amplitude. If the

amplitude of the state vector was above the average amplitude of the state vectors in the system, the transform will decrease the amplitude of the state vector. After application of the transform, the vector will be as much below the average as it was above the average before the transform was applied. Conversely, if the amplitude of the state vector was below the average amplitude of the state vectors in the system, the inversion about the average transform will increase the amplitude of the state vector. After application of the transform, the vector will be as much above the average as it was below the average before the transform was applied [10].

Grover showed that applying inversion about the average will increase the amplitude of a target state if the following initial conditions are met: all of the state vectors have equal amplitude; all state vectors, except the target state, have a positive phase; and the target state has a negative phase.

Figure 2 and Figure 3 show how inversion about the average works. In Figure 2 and Figure 3, the target location is location 3.

Figure 2. System before applying inversion about the average.

Figure 3. System after applying inversion about the average.

As can be seen in Figure 2 and Figure 3, inversion about the average increases the amplitude of the target state (location 3) and decreases the amplitudes of the other states.

Grover [8, 9, 10] showed that there exists some number X , which is less than $\sqrt{N}$, that gives the optimal number of times to apply Equation 17 to the start state. That is, if we apply Equation 17 X times to the start state followed by a single application of W , we will measure the target state with a probability greater than 50%. Boyer et al. [2] refined Grover's original analysis and demonstrated that we can do better than a 50% probability of success.

Boyer et al. showed that the optimal number of iterations of Grover's algorithm is $(\pi - 2\theta)/4\theta$ [2]; however, when 2^n is large, this is approximately

$$\frac{\pi}{4} \sqrt{2^n}. \quad (21)$$

This is because $\theta \approx \sin \theta = 1/\sqrt{2^n}$ when θ is small [2].

After performing the number of iterations required by Equation 21, the probability of finding the target location is close to 100% when n is large. The probability of success is at least [2]

$$1 - \frac{1}{2^n}. \quad (22)$$

Moreover, Boyer et al. [2] have shown that performing a few too many iterations of Equation 17 will decrease the probability of finding the target state. If we apply a single iteration of Equation 17 to a system after we have already performed the optimum number of iterations of Equation 17, the average amplitude of the system will become negative [3].

Thus, if the average amplitude is negative, the amplitude of the target state will decrease after being negated and then inverted about the average. This can be illustrated by performing Grover's algorithm on the system represented by Figure 3. Figure 3 shows a system in which we have the optimum amplitude of the target state (i.e., an amplitude of 1). Before we apply inversion about the average to the system represented by Figure 3, we negate the amplitude of the target state. As a result, the average amplitude of the system will become $-\frac{1}{4}$. This is illustrated in Figure 4.

Figure 4. System after inversion of the target state.

After we apply inversion about the average, the probability of finding the target state would fall from 100% to 25%. This is illustrated in Figure 5.

Figure 5. The system after applying inversion about the average.

In Grover's algorithm, the probability of success is periodically related to the number of iterations of Equation 17 performed [3]. This periodic behavior is due to the finite size of the system and the unitarity of Equation 17 [4]. The periodic nature of Grover's algorithm is shown in Figure 6. This figure shows the relationship between the number of iterations of Equation 17 performed and the probability of finding the target state (i.e., the probability of success) for a 5-qubit system. Figure 6 shows that the probability of success initially peaks after four iterations of Equation 17, falls, and then peaks again after thirteen iterations of Equation 17.

Figure 6. Probability of success versus the number of iterations of Equation 17.

Chapter 5 User Guide for the Animation

5.1 Overview of the Animation

The animation of Grover's search algorithm (hereinafter referred to as "the animation") implemented in this thesis work provides a visual representation of Grover's algorithm. The animation mimics a search for a unique item in an unsorted database by randomly selecting a target location (the target state in Grover's algorithm) and then applying Equation 17 X times as required by Equation 21.

Since the heart of Grover's algorithm is the rotation about the average operation, the animation dynamically shows how the amplitudes of the all locations change as each unitary operator in Equation 17 is applied during each of the required number of iterations. Figure 7 and Figure 8 illustrate this.

Figure 7. The amplitudes of a 2-qubit quantum system after applying W .

Figure 8. The amplitudes of a 2-qubit quantum system after applying I_t .

In Figure 7 and Figure 8, the percentages that are shown for each location are calculated by using Equation 6. These percentages reflect the probability of measuring the respective state (i.e., location) at a given time.

The animation can be run in two modes, run or trace. Run mode is non-interactive. The user simply watches the results of the transforms on the system. On the other hand, trace mode allows the user to control the timing of the application of the transforms. Whichever mode is selected, the animation terminates after the required number of transforms has been performed. When this has occurred, the animation reports the results of the session. This is shown in Figure 9.

Figure 9. The amplitudes of a 2-qubit quantum system after all iterations of Q .

In this chapter, the references to searches performed by the animation refer to the animation's isolation of the target state. A complete listing of the source code for the animation (*grover_animation.java*) can be found in Appendix B.

5.2 System Requirements

The animation is written in the Java programming language, version 1.1, and it is designed to be executed on any Java 1.1 compatible World Wide Web Browser.⁴ The animation can also be run from a program called an *appletviewer*. The appletviewer program is included in the Java Development Kit (JDK).⁵ If the animation is executed on an appletviewer, HTML links to the program's documentation are not viewable (see Section 5.4.6). These links are only viewable when the animation is run on a Browser.

⁴ A World Wide Web Browser (Browser) is a software program that reads and interprets HTML (HyperText Markup Language) pages.

⁵ The JDK can be downloaded free of charge from: <http://www.java.sun.com> or <http://www.javasoft.com>.

The animation's Graphical User Interface (GUI) requires a mouse or other interactive pointing device.

5.3 Installing the Animation

The animation can be installed by copying the entire contents of the diskette in the back cover of this thesis into a target directory on your system. Once this is done, the animation can be executed by starting the Browser on your system, and then opening the HTML page entitled *grover_animation.html* that is in the root directory of the enclosed diskette.

5.4 Running the Animation

5.4.1 Introduction

When you first start the animation, you will see the following screen:

Figure 10. The animation's initial screen.

The top section of the screen shown in Figure 10 is the prompting area. The prompting area gives the user information about the current state of the animation. At the bottom of the screen shown in Figure 10 is a rectangular area that contains the following

items (from left to right): 1) an animation speed selection box; 2) a qubit selection box; 3) the Run button; 4) the Trace button; and 5) the Documentation button. Each of these items is explained in the following sections.

5.4.2 The Animation Speed Selection Box

This selection box is used to vary the speed of the animation. There are three options: 1) slow animation; 2) medium animation; or 3) fast animation. The speed of the animation can be changed at anytime while the animation is running.

5.4.3 The Qubit Selection Box

This selection box is used to determine the maximum number of locations that will be searched. There are four possible selections: 1) 2-qubits; 2) 3-qubits; 3) 4-qubits; or 4) 5-qubits. If n represents the number of qubits selected, 2^n locations will be searched. The number of qubits cannot be changed while a search is in progress.

5.4.4 The Run Button

The Run button starts the animation in run mode. However, before the animation can begin its very first search, the user must select the number of qubits. If the number of qubits is not initially selected, an error message is displayed in the prompting area. This message asks the user to first select the number of qubits. Once the initial number of qubits is selected, the animation uses this initial number as the default number of qubits for all future searches until the number of qubits is changed.

After the number of qubits has been selected, pressing the Run button causes a dialog box to appear. This dialog box tells the user what the target location is and what the predicted probability of finding that location is. This is illustrated in Figure 11.

Figure 11. The initial dialog box.

After the user clicks the OK button in the dialog box, the program begins the search and the animation of Grover's algorithm. Run mode is a non-interactive mode of operation. While the animation is applying the various transforms, the prompting area simply reports which location is being isolated. The user can only watch the lengths (i.e., probabilities) of the various locations change until all iterations of Equation 17 are completed. This is illustrated in Figure 12.

Figure 12. Screen shot of the animation in run mode.

The animation will continue applying the various transforms until all steps of the algorithm are complete. At this time, the results are reported (see Figure 9). The animation can be paused during run mode by pressing the Trace button.

5.4.5 The Trace Button

The Trace button starts the animation in trace mode. However, before the animation can begin its very first search, the user must select the number of qubits. If the number of qubits is not initially selected, an error message is displayed in the prompting area. This message asks the user to first select the number of qubits. Once the initial number of qubits is selected, the animation uses this initial number as the default number of qubits for all future searches until the number of qubits is changed.

After the initial number of qubits is selected, the user can apply a transform from Equation 17 to the system by clicking the Trace button. As in run mode, a probability dialog box is shown at the beginning of the search (see Figure 11).

Trace mode is an interactive mode of operation. The user must press the Trace button to apply each of the various transforms of Grover's algorithm. The prompting area reports which location is being isolated and tells the user how to continue (by pressing the Trace button or Run button). There is also a dialog box that appears when the animation is in trace mode. This dialog box tells the user what transform has just been applied. This is illustrated in Figure 13.

Figure 13. Screen shot of the animation in trace mode.

Trace mode can be exited by pressing the Run button. This will place the animation in run mode (therefore, the animation will continue the iterations of Equation 17 in a non-interactive manner). Moreover, the Trace button can be used to pause a search that is in run mode.

Unless the Run button is pressed, the animation will apply a single transform from Equation 17 each time the user clicks the Trace button until all steps of the algorithm are complete. When all steps have been performed, the results are reported (see Figure 9). If the user continues to click the Trace button after all steps of algorithm are completed, a new search is begun in trace mode at a randomly chosen location.

5.4.6 The Documentation Button

Pressing the Documentation button opens up an HTML page called *QNotes.html*. If you are using an appletviewer, you will not be able to view this HTML page. The

appletviewer program cannot display HTML pages. *QNotes.html* contains a modified version of this thesis work, and thus serves as documentation for the animation.

Chapter 6 Trace of the Animation's Execution

6.1 Introduction

This chapter walks through a trace of the animation for a 2-qubit quantum system. Screen shots of the animation will be present along with an explanation of why the screen looks as it does at each stage of the animation. The underlying analysis presented in this chapter applies equally well to 3, 4, and 5-qubit systems. We shall choose location two (i.e., state vector $|01\rangle$, since our state vectors are numbered from 0 to 3) as our target state.

Since qubits combine using the Kronecker product (see Section 3.2), we use the following notations to describe the state vectors in our 2-qubit system:

$$1.) \quad |00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}; \quad (23)$$

$$2.) \quad |01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}; \quad (24)$$

$$3.) \quad |10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}; \text{ and} \quad (25)$$

$$4.) \quad |11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \quad (26)$$

In addition, we use the following notations to describe the transforms used in our 2-qubit system:

$$1.) \quad W = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}; \quad (27)$$

$$2.) \quad I_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \text{ and} \quad (28)$$

$$3.) \quad -I_s = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}. \quad (29)$$

6.2 The Number of Iterations of Q to Perform

The first issue that the animation must address is the number of times that we must apply the transform Q from Equation 17 to the initial state vector. Equation 21 tells us that we must apply $\frac{\pi}{4}\sqrt{2^2} \cong 1.6$ iterations of Q for a 2-qubit system. Boyer et al. showed that applying too many iterations of Q causes the probability of finding the target state to

drop precipitously [2]. For a 2-qubit system, the animation avoids performing too many iterations of Q by applying only a single application of Q to the start state.

6.3 The Start State

Grover proved that we could chose any basis state in our system as the start state [9]. However, if we use a start state that is not $|00\rangle$, we lose rotation about the average as a means for explaining the behavior of the algorithm [9]. Therefore, the animation uses $|00\rangle$ as the start state.

We can imagine that the register used to hold the results of the transforms on the start state is a 4×1 matrix called *Current_State*. Hence, when we start the animation, we initialize *Current_State* to hold $|00\rangle$. That is,

$$Current_State = |00\rangle. \quad (30)$$

6.4 Initial Application of the W (Walsh-Hadamard Transform)

Equation 17 tells us that the first transform that we must apply to the *Current_State* register is the Walsh-Hadamard transform. Thus, after applying W from Equation 27 to the *Current_State* register, we have

$$Current_State = |00\rangle \xrightarrow{W} \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle. \quad (31)$$

Our new current state is a superposition of all the four possible states that the system can be in. Each state in this superposition has a positive phase and amplitudes of $\frac{1}{2}$. The

asterisk under state $|01\rangle$ in Equation 31 (and subsequent equations) denotes that this our target state. Equation 6 tells us that the probability of measuring any state vector in the *Current_State* register is

$$\frac{\left| \frac{1}{2} \right|^2}{1} = \frac{1}{4}. \quad (32)$$

Figure 14 shows how the animation graphically represents Equation 31.

Figure 14. The animation's display after the initial application of W .

6.5 Application of I_t

The next step in the algorithm requires us to apply I_t , as given in Equation 28 to the *Current_State* register. I_t simply inverts the phase of the target state, $|01\rangle$. That is,

$$\text{Current_State} = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \xrightarrow{I_t} \frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle. \quad (33)$$

Figure 15 shows how the animation graphically displays Equation 33.

Figure 15. The animation's display after application of I_t .

6.6 Second Application of W

After we have applied I_t , we must again apply the W transform, as given in Equation 27, to our system. Unlike our initial application of W to the start state, the *Current_State* register is not a single vector, but rather it is in superposition. Hence, this application of W will send each of the vectors currently in superposition to another superposition, and then combine the resultant vectors to form the new current state. Mathematically, this is represented by:

Current_State =

$$\frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \xrightarrow{W}$$

$$\text{a. } \frac{1}{2}|00\rangle \xrightarrow{W} \frac{1}{4}|00\rangle + \frac{1}{4}|01\rangle + \frac{1}{4}|10\rangle + \frac{1}{4}|11\rangle$$

$$\text{b. } -\frac{1}{2}|01\rangle \xrightarrow{W} -\frac{1}{4}|00\rangle + \frac{1}{4}|01\rangle - \frac{1}{4}|10\rangle + \frac{1}{4}|11\rangle$$

$$\text{c. } \frac{1}{2}|10\rangle \xrightarrow{W} \frac{1}{4}|00\rangle + \frac{1}{4}|01\rangle - \frac{1}{4}|10\rangle - \frac{1}{4}|11\rangle$$

$$\text{d. } \frac{1}{2}|11\rangle \xrightarrow{W} \frac{1}{4}|00\rangle - \frac{1}{4}|01\rangle - \frac{1}{4}|10\rangle + \frac{1}{4}|11\rangle \xrightarrow{\text{combine a-d}}$$

$$\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle. \quad (34)$$

Thus, the absolute values of the amplitudes of the state vectors in the *Current_State* register are still the same ($\frac{1}{2}$). However, the phases of the state vectors are all positive, except for state vector 1. Figure 16 shows how the animation, with a slight rounding error (24.9% instead of 25% for state vectors 0, 2, and 3), graphically displays the results of Equation 34.

Figure 16. The animation's display after the second application of W .

6.7 Application of $-I_s$

The next step in the algorithm requires us to apply $-I_s$ from Equation 29 to the *Current_State* register. $-I_s$ simply inverts the phases of all of state vectors, except the start state ($|00\rangle$). Mathematically this is

$$\text{Current_State} = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \xrightarrow{-I_s} \frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle. \quad (35)$$

Figure 17 shows how the animation displays the results of Equation 35.

Figure 17. The animation's display after application of I_t .

6.8 Final Application of W

After we have applied $-I_s$, we have completed our single application of Q from Equation 17 to the start state. We now need to apply a final application of W , as given in Equation 27, to the *Current_State* register to increase the amplitude of the target state [9]. Since the *Current_State* register is in superposition (unlike our initial application of W), the current application of W will send each of the vectors in superposition to another superposition. The resultant superposition of vectors will then combine to form the final state of the system.

Mathematically, this is represented as follows:

Current_State =

$$\frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle \xrightarrow{W}$$

$$\text{a. } \frac{1}{2}|00\rangle \xrightarrow{W} \frac{1}{4}|00\rangle + \frac{1}{4}|01\rangle + \frac{1}{4}|10\rangle + \frac{1}{4}|11\rangle$$

$$\text{b. } -\frac{1}{2}|01\rangle \xrightarrow{W} -\frac{1}{4}|00\rangle + \frac{1}{4}|01\rangle - \frac{1}{4}|10\rangle + \frac{1}{4}|11\rangle$$

$$\text{c. } \frac{1}{2}|10\rangle \xrightarrow{W} \frac{1}{4}|00\rangle + \frac{1}{4}|01\rangle - \frac{1}{4}|10\rangle - \frac{1}{4}|11\rangle$$

$$\text{d. } -\frac{1}{2}|11\rangle \xrightarrow{W} -\frac{1}{4}|00\rangle + \frac{1}{4}|01\rangle + \frac{1}{4}|10\rangle - \frac{1}{4}|11\rangle \xrightarrow{\text{combine a-d}}$$

$$0|00\rangle + 1|01\rangle + 0|10\rangle + 0|11\rangle. \quad (36)$$

Thus, the absolute value of the amplitudes of the final state will all be 0, except for the target state. The amplitude of the target state is now 1 and has a positive phase. Figure 18 shows how the animation graphically displays the results of Equation 36.

Figure 18. The animation's display after the final application of W .

Thus, after applying Equation 6 to each state vector in the *Current_State* register, we see that will find the target state with 100% certainty. Figure 19 shows the animation's display after all steps of Grover's algorithm have been performed.

Figure 19. The animation's display after all steps of the algorithm are complete.

6.9 Summation

If we look at Figure 15 and Figure 18 along with Equation 33 and Equation 36, we see that the operation $-WI_0W$ is indeed the inversion around the average as claimed by Grover [9]. From Equation 33, we see that after inverting the target state, the average of the amplitudes of the state vectors is:

$$\frac{\left(\frac{1}{2} - \frac{1}{2} + \frac{1}{2} + \frac{1}{2}\right)}{4} = \frac{1}{4}. \quad (37)$$

Since the state vectors $|00\rangle$, $|10\rangle$, and $|11\rangle$ all have an amplitude of $\frac{1}{2}$, they are each $\frac{1}{4}$ above the average before inversion about the average (see Equation 33). Thus, after inversion about the average, these vectors should be $\frac{1}{4}$ below the average. That is, they should have amplitudes of 0 after inversion about the average.

Moreover, our target vector, $|01\rangle$, is $\frac{3}{4}$ below the average before the inversion about the average (see Equation 33). Therefore, after inversion about the average, it should be $\frac{3}{4}$ above the average. That is, state vector $|01\rangle$ should have an amplitude of 1.

After applying the operators $-WI_0W$, Equation 36 shows that the amplitudes of all the states are as predicted by inversion about the average. That is, $|00\rangle$, $|10\rangle$, and $|11\rangle$ each has an amplitude of 0, and $|01\rangle$ has an amplitude of 1 (see Equation 36). Thus, with a single application of Equation 17, we could find an item that is in one of four locations.

Chapter 7 Verification of the Animation's Correctness

7.1 Introduction

To verify the correctness of the animation, the steps of the algorithm were hand traced for 2 and 3-qubits using the Maple[®] Release 5. Moreover, this chapter verifies the animation's correctness for 4 and 5-qubits by using the results of the hand trace and an analysis of the Java source code.

A complete listing of the Maple session (*verification.txt*) saved as text can be found in Appendix D. The original Maple session (*verification.mws*) can be found in the diskette in the back cover of this thesis. In this chapter, the number of steps performed by the animation refers to the number of iterations of Equation 17 that are applied.

7.2 Two Qubits

For 2-qubits, the animation predicted that the target location could be found in one step with a probability of 100%. Figure 20 shows this prediction.

Figure 20. The animation's prediction for location 2.

Using Maple, the values for the transforms in Equation 17 for a 2-qubit system were calculated and used to compute the probability of finding a target location. The Maple hand trace predicted the same values as the animation.⁶

7.3 Three Qubits

For 3-qubits, the animation predicted that the target location could be found in two steps with a probability of 94.5%. Figure 21 shows this prediction.

Figure 21. Animation's prediction for location 5.

Using Maple, the values for the transforms in Equation 17 for a 3-qubit system were calculated and used to compute the probability of finding a target location. The Maple hand trace predicted the same values as the animation.⁷

⁶ See Appendix D at page 124.

⁷ See Appendix D at page 128.

7.4 Four and Five Qubits

A hand trace for 4 and 5-qubits would involve multiplication of matrices of sizes 16 by 16 (256 entries), and 32 by 32 (1024 entries). Thus, instead of a hand trace of Grover's algorithm for 4 and 5-qubits, we can verify the animation's correctness for these inputs by analyzing the source code and using the results of the 2 and 3-qubit hand trace.

We can assume that the functions used to implement Equation 17 are correct for 2 and 3-qubits since a hand trace produced the same results as the animation. The program uses a Java class called *matrix* to implement two key matrix operations – multiplication and calculation of the Kronecker product.⁸ Function prototypes (with comments) of these two key methods are given below.

The prototype of the function *multiply*, which is used to implement matrix multiplication, is as follows:

```
matrix multiply (matrix A, matrix B);9

/**
Function:    multiply
Purpose:    Computes the product of two matrices.
In:         A: a matrix object.
            B: a matrix object.
Out:        A matrix object that is the product of A and B.
Requires:   The number of columns in matrix A equals the number of
            rows in matrix B. If this requirement is not met, a
            matrix error is thrown.
*/
```

The prototype of the function *direct*, which is used to implement calculation of the Kronecker product, is as follows:

⁸ See Appendix B at page 111 for a listing of the class *matrix*'s data members and methods.

⁹ See Appendix B at page 114 for the complete implementation of this function.

```

matrix direct (matrix A, matrix B);10

/**
Function:    direct
Purpose:     Computes the Kronecker product of two matrices.
In:          A: a matrix object.
              B: a matrix object.
Out:         A matrix object that is the direct product of A and B.
Requires:    None.

*/

```

The correctness of the functions *multiply* and *direct* does not depend on the number of qubits entered by the user. This is because the logic in these two functions works with matrix objects. The hand trace showed that these two functions correctly manipulate matrices to produce the correct product of two matrices and the correct Kronecker product. Thus, as long as the correct matrices are given as input, these functions will produce the correct output for 4 and 5-qubits since the number of qubits does not affect the underlying logic of these two functions.

The Java class *grover_animation* uses the class *matrix* to implement Equation 17 by implementing the following methods: *make_I_sub_T*, *make_I_sub_O*, *make_W*, and *make_init_reg*.

The function *make_I_sub_T* is used to implement the variable I_t in Equation 17. This is a simple function whose implementation is as follows:

```

/**
Function:    make_I_sub_T
Purpose:     Makes a diagonal matrix of size: 2^states X 2^states
              where all the diagonals are 1, except the element

```

¹⁰ See Appendix B at page 112 for the complete implementation of this function.

```

        _special_loc, (_special_loc is the target state)
        whose value is -1.
In:      numStates: The number of qubits selected by the user.
Out:     A diagonal matrix of size: 2^states X 2^states
        where all the diagonals are 1, except the element
        _special_loc, whose value is -1.
Requires: None.
*/

matrix make_I_sub_T( int numStates)
{

    matrix C = make_identity(numStates);

    C.set_element(_special_loc, _special_loc, -1.0);

    return C;

} //end of function

```

Make_I_sub_T passes the number of qubits selected by the user (*numStates*) to the function *make_identity*. The *make_identity* function makes an identity matrix of size $numStates \times numStates$ by means of a simple *for loop* iteration construct (i.e., a Java *for* looping construct).¹¹ The hand trace for 2 and 3-qubits showed that the function *make_identity* must make the correct size matrix for 2 and 3-qubits.

Because no logic in the *make_identity* function changes for 4 and 5-qubits, the *make_identity* function will also make the correct size identity matrix for 4 and 5-qubits. The *make_identity* function is the only logical operation in the *make_I_sub_T* function that makes use of the number of qubits. Hence, follows that *make_I_sub_T* will make the correct I_i for 4 and 5-qubits.

¹¹ See Appendix B at p. 84 for a listing of the *make_identity* function.

The function *make_I_sub_O* is used to implement the variable $-I_0$ in Equation 17.

This is a simple function whose implementation is as follows:

```

/**
Function:    make_I_sub_O
Purpose:    Makes a diagonal matrix of size: 2^states X 2^states
            where all the diagonals are 1, except the element
            0,0 whose value is -1.
Date:       February 27, 1999
In:         numStates: The number of bits selected by the user
Out:        Makes a diagonal matrix of size: 2^states X 2^states
            where all the diagonals are 1, except the element
            0,0 whose value is -1.
Requires:    None.
*/

matrix make_I_sub_O( int numStates)
{
    matrix C = make_identity(numStates);

    C.set_element(0,0, -1.0);

    return C;
} //end of function

```

This function (like *make_I_sub_T*) passes the number of qubits selected by the user (*numStates*) to the function *make_identity*. We have shown that the *make_identity* function produces the correct output for 4 and 5-qubits. Since the *make_identity* function is the only logical operation in the *make_I_sub_O* function that makes use of the number of qubits, it follows that *make_I_sub_O* will make the correct $-I_0$ for 4 and 5-qubits.

The function *make_W* is used to implement the variable W in Equation 17. This is a simple function whose implementation is given below:

```

/**
Function:    make_W
Purpose:    Applies the matrix  $M = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ 
            to numStates qubits to make the Walsh-Hadamard
            transform.
Date:       March 12, 1999
In:         numStates: The number of bits selected by the user
Out:        a matrix that represents the Walsh-Hadamard
            transform.
Requires:   None.

*/

```

```

matrix make_W(int numStates)
{
    matrix M = new matrix(2,2);
    //M = 1/sqr(2) * | 1 1 |
    //                | 1 -1 |

    M.set_element(0,0,.707106781);
    M.set_element(0,1,.707106781);
    M.set_element(1,0,.707106781);
    M.set_element(1,1,-.707106781);

    matrix temp1 = new matrix (4,4);

    matrix temp2 = new matrix(8,8);

    matrix temp3 = new matrix(16,16);

    matrix temp4 = new matrix(32,32);

```

```

switch(numStates)
{
    /* no breaks needed because we return out of each case */
    case 2:
    {
        temp1 = M.direct(M, M);

```

```

return temp1;

}

case 3:
{
temp1 = M.direct(M, M);

temp2=temp1.direct(M, temp1);
return temp2;
}

case 4:
{
temp1 = M.direct(M, M);

temp2=temp1.direct(M, temp1);

temp3 = temp2.direct(M, temp2);

return temp3;
}

default:
{
temp1 = M.direct(M, M);

temp2=temp1.direct(M, temp1);

temp3 = temp2.direct(M, temp2);

temp4 = temp3.direct(M, temp3);

return temp4;
}

} //end of switch

} //end of function

```

The *switch* statement in this function implements the Kronecker product of

$M \otimes M \dots \otimes M$, where M is as defined in Equation 14. *Case 2* and *case 3* correspond to 2-qubits ($M \otimes M$) and 3-qubits ($M \otimes M \otimes M$), respectively. The hand trace for 2 and 3-qubits verified that these two *case* statements must be correct since the animation used *make_W* to produce the correct results.

Case 4 corresponds to 4-qubits and is simply $M \otimes$ *case 3*. This corresponds to $M \otimes M \otimes M \otimes M$. We know that *case 3* produces the correct output and that the function used to implement the Kronecker product (*direct*) produces the correct results. Therefore, it follows that *case 4* will also produce the correct result.

The default *case* statement corresponds to five qubits, and is simply $M \otimes$ *case 4*. This corresponds to $M \otimes M \otimes M \otimes M \otimes M$. It follows that this will produce the correct results for 5-qubits since *case 4* produces the correct results.

The function *make_init_reg* is used to implement the initial register (start state) of the quantum system. This is a simple function whose implementation is given below:

```

/**
Function:    make_init_reg
Purpose:    Makes a column matrix of size: 2^states X 1
             where all the diagonals are 0, except the element
             0,0 whose value is 1.
In:         numStates: The number of bits selected by the user
Out:        makes a column matrix of size: 2^states X 1
             where all the diagonals are 0, except the element
             0,0 whose value is 1.
Requires:    None.

*/
matrix make_init_reg(int numstates)
{
matrix zero = new matrix(2,1);

zero= make_zero();

```



```

matrix temp1 = new matrix(4,1);
matrix temp2 = new matrix(8,1);
matrix temp3 = new matrix (16,1);
matrix temp4 = new matrix (32,1);

```

```

switch(numstates)
{
case 2:
{

    temp1 = zero.direct(zero,zero);
    return temp1;
}

case 3:
{

    temp1 = zero.direct(zero,zero);
    temp2 = zero.direct(zero, temp1);
    return temp2;
}

case 4:
{

    temp1 = zero.direct(zero,zero);
    temp2 = zero.direct(zero, temp1);
    temp3 = zero.direct(zero, temp2);
    return temp3;
}
default:
{

    temp1 = zero.direct(zero,zero);
    temp2 = zero.direct(zero, temp1);
    temp3 = zero.direct(zero, temp2);
    temp4 = zero.direct(zero,temp3);
    return temp4;
}

} //end of switch

```

```
}//end of function
```

The *switch* statement in *make_init_reg* implements the Kronecker product of $0 \otimes 0 \dots \otimes 0$, where 0 is $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. *Case 2* and *case 3* correspond to 2-qubits ($0 \otimes 0$) and 3-qubits ($0 \otimes 0 \otimes 0$), respectively. The hand trace for 2 and 3-qubits verified that these two *case* statements must be correct since the animation produced the correct results using the function *make_init_reg*.

Case 4 corresponds to 4-qubits, and is simply $0 \otimes$ *case 3*. This corresponds to $0 \otimes 0 \otimes 0 \otimes 0$. We know that *case 3* produces the correct output and that the function used to implement the Kronecker product (*direct*) produces the correct results. It, therefore, follows that *case 4* will also produce the correct result.

The default *case* statement corresponds to 5-qubits, and is simply $0 \otimes$ *case 4*. This corresponds to $0 \otimes 0 \otimes 0 \otimes 0 \otimes 0$. It follows that this will produce the correct results for 5-qubits since *case 4* produces the correct results.

The function *the_algo* uses all the above-mentioned functions to implement Grover's algorithm.¹² We have shown that the *the_algo* produces the correct results for 2 and 3-qubits as verified by the Maple hand trace. In addition, we have shown that the functions used to implement *the_algo* will produce the correct results for 4 and 5-qubits. Therefore, we can conclude that the function *the_algo* and, hence, the animation is correct for all of its input (i.e., 2, 3, 4, and 5-qubits).

¹² The implementation of the function *the_algo* is given on page 99 of Appendix B.

Chapter 8 Conclusion

This thesis work has presented an introduction to the mathematical and theoretical foundations of quantum computing. It used Grover's search algorithm to demonstrate one of the key ideas in quantum computing – increasing the probability of measuring a desired state. A program was developed that provided a visualization of Grover's algorithm.

There are significant technical problems in building a practical quantum computer, such as making sure that the environment does not corrupt the quantum state and preserving the coherence of the quantum system [4, 21]. However, Isaac et al. [4] have shown that it is possible to overcome these problems by using quantum error correction and techniques to control decoherence.

In [4], Isaac et al. experimentally demonstrated that Grover's quantum search algorithm could successfully find data in one of four possible locations in a single step, instead of the 2.25 steps that would be required on a classical computer. This experimental result is in accordance with the result obtained by the program developed in this thesis work (see Section 6.9).

Thus, as Grover's algorithm demonstrates, quantum computing gives us a new tool for looking at fundamental computer science problems and questions. Quantum computing may be the key to answering the question of whether $P = NP$ [21].

The references to this thesis provide a good starting place for further investigation. In particular, the site, <http://xxx.lanl.gov/archive/quant-ph>, provides a rich collection of articles on quantum computing.

References

- [1] Benioff, Paul (1980). The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines. *Journal of Statistical Physics*, Vol. 22, pp. 563-591.
- [2] Boyer, Michel, Gilles Brassard, Peter Høyer, and Alain Tapp (1996). Tight Bounds on Quantum Searching. Los Alamos preprint archive, <http://xxx.lanl.gov/archive/quant-ph/9605034>.
- [3] Butler, Michael, and Pieter Hartel (1998). Reasoning about Grover's Quantum Search Algorithm using Probabilistic wp. Los Alamos preprint archive, <http://xxx.lanl.gov/archive/quant-ph/9810066>.
- [4] Chuang, Isaac L., Neil Gershenfeld, and Mark Kubinec (1998). Experimental Implementation of Fast Quantum Searching. *Physical Review Letters*, Vol. 80, No. 15, pp. 3408-3411.
- [5] Cunningham, Steve, and Roger Hubbard (eds.) (1992). Interactive Learning Through Visualization: The Impact of Computer Graphics in Education. Springer-Verlag, New York.
- [6] Deutsch, David (1985). Quantum Theory, the Church-Turing Principle, and the Universal Quantum Computer. *Proceedings of the Royal Society of London*, Vol. A 400 (1985), pp. 97-117.
- [7] Feynman, Richard (1982). Simulating Physics with Computers. *International Journal of Theoretical Physics*, Vol. 21, Nos. 6/7, pp. 467-488.
- [8] Grover, Lov K. (1996). A Fast Quantum Mechanical Algorithm for Database Search. *Proceedings 28th Annual Symposium on the Theory of Computing*, pp. 212-219.
- [9] Grover, Lov K. (1997). A Framework for Fast Quantum Mechanical Algorithms. Los Alamos preprint archive, <http://xxx.lanl.gov/archive/quant-ph/9711043>.
- [10] Grover, Lov K. (1997). Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Physical Review Letters*, Vol. 78, No. 2, pp. 325-328.
- [11] Hughes, Richard I. G. (1989). The Structure and Interpretation of Quantum Mechanics. Harvard University Press, Cambridge, MA.

- [12] Jauch, Josef Maria (1968). *Foundations of Quantum Mechanics*. Addison-Wesley Publishing Company, Reading, MA.
- [13] Jordan, Thomas F. (1986). *Quantum Mechanics in Simple Matrix Form*. John Wiley & Sons, New York.
- [14] Kirchherr, Walter W. (1993). NEPS Operations on Cordial Graphs. *Discrete Mathematics*, Vol. 115, p. 204.
- [15] Kirchherr, Walter W. (1994). *Quantum Computing*. Lecture Notes, Department of Mathematics and Computer Science, San Jose State University, San Jose, CA.
- [16] Lemay, Laura, and Rogers Cadenhead (1998). *SAMS Teach Yourself Java 1.2*. Sams Publishing, Indianapolis, IN.
- [17] Lloyd, Seth (1993). A Potentially Realizable Quantum Computer. *Science*, Vol. 261, pp. 1569 – 1571.
- [18] Lloyd, Seth (1997). Quantum-Mechanical Computers. *Scientific American, The Solid-State Century, Special Issue*, Vol. 8, No.1, pp. 98-104.
- [19] Pohl, Herbert (1967). *Quantum Mechanics for Science and Engineering*. Prentice-Hall, Inc., Englewood Cliffs, NJ.
- [20] Polkinghorne, John Charlton (1984). *The Quantum World*. Longman Group Limited, New York.
- [21] Rieffel, Eleanor, and Wolfgang Polak (1998). An Introduction to Quantum Computing for Non-Physicist. Los Alamos preprint archive, <http://xxx.lanl.gov/archive/quant-ph/9809016>.
- [22] van der Linden, Peter (1998). *Just Java and Beyond*, Third Edition. Sun Microsystems Press, Palo Alto, CA.
- [23] Williams, Colin P., and Scott H. Clearwater (1998). *Explorations in Quantum Computing*. Springer-Verlag, New York.
- [24] Zalka, Christof (1997). Grover's Quantum Search Algorithm is Optimal. Los Alamos preprint archive, <http://xxx.lanl.gov/archive/quant-ph/9711070>.

Appendices

Appendix A Mathematical Definitions

A.1 Vector Spaces

A vector space is a mathematical structure that is defined over some field F . For our purposes, the field that we are concerned with is the set of complex numbers. We call the elements of F , scalars. Hughes [11, p. 41] defines a vector space $\mathcal{V}$ as follows:

$\mathcal{V}$ is called a vector space over F if $\mathcal{V} = \langle V, +, \bullet, 0 \rangle$, where V is a nonempty

set whose elements are called vectors;

$+$ is an operation which takes any pair of vectors and yields a vector (that is, $+$ is a binary operation on V);

$\bullet$ is an operation which takes a scalar and a vector and yields a vector;

0 is a member of V (the zero vector);

for all u, v , and w in V , the following identities hold:

$$\dots \quad (u + v) + w = u + (v + w)$$

$$\dots \quad u + v = v + u$$

$$\dots \quad v + 0 = v$$

$$\dots \quad a \bullet (b \bullet v) = (a \bullet b) \bullet v$$

$$\dots \quad (a + b) \bullet v = a \bullet v + b \bullet v$$

$$\dots \quad a \bullet (v + u) = a \bullet v + a \bullet u$$

$$\dots \quad 0 \bullet v = 0$$

$$\dots \quad 1 \bullet v = v.$$

A.2 Eigenvectors and Eigenvalues

An eigenvector of matrix A exists if $A\varphi = \lambda\varphi$, where φ is a vector and λ is a scalar. λ is called an eigenvalue and φ is called an eigenvector.

A.3 Linear Operators

An operator A in the some vector space V is called linear if for all vector u and v and any scalar a , $A(v + u) = Av + Au$ and $A(av) = a(Av)$.

A.4 Hermitian Operators

We say that a linear operator A on a vector space V is Hermitian if, for all vectors u , and v , $\langle u | Av \rangle = \langle Au | v \rangle$ [11].

A.5 Unitary Matrices

A unitary operator is of the form $A^T A = AA^T = I$.

A.6 Hilbert Space

The vector space that we will use to model quantum behavior is called a Hilbert space. Hughes [11, p.55] defines a Hilbert space as a vector space in which the inner product for vectors is defined, and which is *complete*. A vector space is complete if “any converging sequence of vectors in the space converges to a vector in the space” [11, p. 55]. All finitely dimensional vector spaces are complete [11]. In this work, we are only concerned with finitely dimensional vector spaces; therefore, all such vector spaces are Hilbert spaces.

In this work, we use a vector in Hilbert space to model the representation of the physical state of a quantum system.

Appendix B grover_animation.java

/**

Program: grover_animation.java

Purpose: Animation of Lov K. Grover's algorithm for exhaustive search starting from the 0 state.

Compile: javac grover_animation.java

Run: appletviewer grover_animation.html

Java Version: JDK1.1.4

Copyright:

Copyright (c) 1999
 Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

Presented to
 The Faculty of the Department of Mathematics and
 Computer Science
 San Jose State University
 San Jose, California

In Partial Fulfillment of the Requirements for the
 Degree Master of Science

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. The authors make no representation about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

*/

```

// import necessary packages
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.math.*;
import java.util.Random;
import java.net.*;

/**
    Class:      grover_animation
    Date:      March 12, 1999
    Purpose:   Animates Lov K. Grover's algorithm for exhaustive search
               starting from the 0 state.

    Author:    Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

*/

public class grover_animation extends Applet
implements Runnable, ActionListener, ItemListener

{
    //data members

    int _xStart = 5; // x coordinate location to start drawing base line from
    int _xEnd = 495; // x coordinate location to stop drawing base line
    int _LineLength = _xEnd - _xStart; // length of the base line

    int _yStart = 205; // y coordinate location of the base line

    int _Y1_offset = -55; //used to draw base line 1 when _total_states =32

    int _Y2_offset = 75; //used to draw base line 2 when _total_states =32
    int _x;      /* holds current x coordinate when iterating through
                  the column states

```

```

        */
int _y;      /* holds current y coordinate when iterating through
              the column states
              */

int _special_loc = 2; // target location

int _randValue;      // value imputed by the user to be found

int _offset;      //offset used to space line lengths of target states

double _length_offset = 150.00;

double _scale = 75; // value used multiply lengths by

int _states;      // the number of states selected by the user
int _total_states; //  $2^{\text{\_states}}$ 
length_array_elements _length_array[]; /*The lengths and value of
                                         the items stored in each
                                         state vectors (column vectors).
                                         */

int _turns;      // number of steps used in the algorithm

int _max_turns; // the maximum number of turns that the algorithm runs

double _predicted_prob; //predicted probability

// flags

boolean _animateFlag = false; //start animating when true
boolean _drawLineFlag = false; // draw histogram when true
boolean _clearFlag = false; // clear the screen when true

boolean _okToRun = false; /* Flag used to make sure states have been
                           selected.
                           */

```

```

boolean _goodToGo = false; // used to break of the while loop in run

boolean _TraceFlag = false; //used to suspend the animation

boolean _running = false; // used to tell if the algorithm is in running mode

//double buffering variables

Image  _offScreenImg;

Graphics _offscreen;

//Thread for animation

Thread  _runner;

int  _speed = 2000; //value used to pause threads (in sleep function)
int  _traceSpeed = 0; //need very fast tread speed (little sleep time) for
    // tracing
int  _oldSpeed = _speed;    //holds the current speed before tracing

int  _clearTime = 150; //time use synchronize dialog clear and screen output
//layout variables.

Panel _north = new Panel();

Panel _south = new Panel();

Panel _init_dialog_panel = new Panel();

// the trace dialog

Dialog _dialog;

//the initiating dialog

Dialog _init_dialog;

Component _dialog_component;

```

```
//text used in the trace dialog
```

```
Label _dialog_label1 = new Label();
Label _dialog_label2 = new Label();
Label _dialog_label3 = new Label();
Label _dialog_label4 = new Label();
```

```
//text used in the initialization dialog
```

```
Label _init_dialog_label1 = new Label();
Label _init_dialog_label2 = new Label();
Label _init_dialog_label3 = new Label();
Label _init_dialog_label4 = new Label();
```

```
//button used in the initialization dialog
```

```
Button _ok = new Button("OK");
```

```
int gap = 10;
int gap2 = 25;
FlowLayout _flo = new FlowLayout(FlowLayout.LEFT, gap2, gap);
GridLayout _upNorth = new GridLayout(2, 1, gap, gap);
GridLayout _dialogGrid = new GridLayout(4, 1);
GridLayout _dialogGrid2 = new GridLayout(5, 1);
```

```
Label _prompt = new Label("
algorithm
");
Label _prompt2 = new Label("
qubits to begin.
");
```

Animation of Grover's Search

Please select the number of

```
Button _run = new Button("Run");
```

```
Button _trace = new Button("Trace  ");
```

```
Button _link = new Button("Documentation");
```



```
Choice _state_options = new Choice();
```

```
Choice _speed_options = new Choice();
```

```
URL _qNotes;
```

```
/**
```

```
Function:          init
```

```
Purpose:            Initializes the applet.
```

```
Date:            March 12, 1999
```

```
In:              None.
```

```
Out:            Applet variables are initialized.
```

```
Requires:        None.
```

```
Author:          Terrance Bing-Parks:
```

```
Advisor:
```

```
Dr. Walter Kirchherr
```

```
Department of Mathematics and Computer Science
```

```
San Jose State University
```

```
San Jose, California
```

```
*/
```

```
public void init()
```

```
{
```

```
//Create the dialog window (is initially invisible).
```

```
// used during tracing.
```

```
Component _dialog_component = this.getParent();
```

```
//Recursively find the parent of the applet.
```

```
while (_dialog_component !=null && !(_dialog_component instanceof Frame))
```

```
    _dialog_component = _dialog_component.getParent();
```

```
// the trace dialog
```

```
_dialog =new Dialog ((Frame) _dialog_component, "Transform Applied", false);
```

```
_dialog.setBounds(65,0,330,140);
```

```
_dialog.setLayout(_dialogGrid);
```

```
_dialog.add(_dialog_label1);
```

```
_dialog.add(_dialog_label2);
```

```
_dialog.add(_dialog_label3);
```

```
_dialog.add(_dialog_label4);
```

```
// the initiating dialog
```

```
_init_dialog =new Dialog ((Frame) _dialog_component, "Initializing The System", false);
```

```
_init_dialog.setBounds(150,150,250,200);
```

```
_init_dialog.setLayout(_dialogGrid2);
```

```
_init_dialog_panel.add(_ok);
```

```
_init_dialog.add(_init_dialog_label1);
```

```
_init_dialog.add(_init_dialog_label2);
```

```
_init_dialog.add(_init_dialog_label3);
```

```
_init_dialog.add(_init_dialog_label4);
```

```
_init_dialog.add(_init_dialog_panel);
```

```
setLayout( new BorderLayout());
```

```
setBackground(Color.white);
```

```
//North Panel
```

```
_north.setLayout(_upNorth);
```

```
_north.setBounds(400, 100, 0,0);
```

```
_north.setBackground(Color.yellow);
```

```
_south.setBackground(Color.blue);
```

```
//add components to the panel
```

```
_north.add(_prompt);
```

```
_north.add(_prompt2);
```

```
//South Panel
_south.setLayout(_flo);
_state_options.addItem("2- qubits");
_state_options.addItem("3- qubits");
_state_options.addItem("4- qubits");
_state_options.addItem("5- qubits");

_speed_options.addItem("Slow Animation");

_speed_options.addItem("Medium Animation");

_speed_options.addItem("Fast Animation");

_south.add(_speed_options);

//Add Listeners.
_state_options.addItemListener(this);

_speed_options.addItemListener(this);

_run.addActionListener(this);

_trace.addActionListener(this);

_ok.addActionListener(this);

_link.addActionListener(this);

//Add components to the panel.
_south.add(_state_options);
_south.add(_run);
_south.add(_trace);
_south.add(_link);
```

```

//Add Panels.
add("North", _north);
add("South", _south);

//initialize double buffering variables

_offScreenImg = createImage(getSize().width, getSize().height);

_offscreen = _offScreenImg.getGraphics();

//Load the HTML URL object.

try
{
    _qNotes = new URL(getCodeBase(), "Documentation/QNotes.html");
}

catch (MalformedURLException e)
{
    System.out.println("Bad URL: " + _qNotes);
}

} //end of init()

//implementation of the abstract class listeners

/**
Function:      actionPerformed
Purpose:      Callback for the buttons.
Date:         March 12, 1999
In:           theEvent: ActionEvent object that tracks changes in
               the buttons.
Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science

```

San Jose State University
San Jose, California

```

*/

public void actionPerformed(ActionEvent theEvent)
{

if(theEvent.getSource() == _link)
    {
        execute_link();
    }
if(_okToRun) //make sure the user has selected the number of states
    {
        if(theEvent.getSource() == _run) //run selected
            {
                execute_run();
            } //end of if

        else if (theEvent.getSource() == _trace )
            {
                execute_trace();
            } //end of else if

        else if(theEvent.getSource() == _ok)
            {
                execute_ok();
            }

    } //end of if

else
    {
        // states have not been entered
        _prompt.setText("Please select the number of qubits and enter a number");
        _prompt.setText("");
    }

} //end of function

```

```

/**
Function:      execute_run
Purpose:      Low level function used by actionPerformed to implement callbacks for
               the Run button.
Date:         March 12, 1999
In:           None
Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

*/
void execute_run()
{
    _run.setEnabled(false); // Disable the run button until execution complete
                           // or trace button hit.

    _dialog.setVisible(false); //hide the dialog box
    _state_options.setEnabled(false);
    _prompt.setText("Animation of Grover's Search algorithm");

    // reset the old speed
    _speed = _oldSpeed;

    // set animation flags
    _goodToGo = true;
    _drawLineFlag = true;

    //reset the label
    _trace.setLabel("Trace");

    if(_TraceFlag)
    {
        try
        {
            _runner.resume();
        } //end try
    }
}

```

```

        catch(Exception ie) {};

        _prompt.setText("
"+(_special_loc+1)+".");
        _prompt2.setText("");
    }

    else if(!_TraceFlag) //restart
    {
        //start threads
        stop();
        start();
    }

    if(_running)
    {
        _prompt.setText("
"+(_special_loc+1)+".");
        _prompt2.setText("");
    }

    _running = true;
    _TraceFlag = false;

} //end of function

```

Isolating location

Isolating location

```

/**
Function:      execute_trace
Purpose:      Low level function used by actionPerformed to implement callbacks for
               the Trace button.
Date:         March 12, 1999
In:           None.
Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

```

```

*/
void execute_trace()
{
    //Enable the run button.
    _run.setEnabled(true);

    //Show the dialog box.

    _dialog.setVisible(true); //show the dialog box
    _dialogToFront();
    _state_options.setEnabled(false);

    if(_TraceFlag)
    {
        try
        {
            _runner.resume();
        } //end try

        catch(Exception ie) {System.out.println("resume error");}
    } //end if flag

    else
    {
        //change the current speed
        _speed = _traceSpeed;
        _TraceFlag = true;
        _trace.setLabel("Tracing...");

        //animate
        _drawLineFlag = true;
        _goodToGo = true;

        if (_running)
        {
            _prompt.setText("Isolating location "+(_special_loc+1)+" using trace
mode. Press the Tracing... button ");
            _prompt2.setText("to step through the isolation. Press the Run button to
run at normal speed.");

            try

```



```

        {
            _runner.suspend();
        } //end try

        catch(Exception ie) {}

    } //end if
else
    {
        stop();
        start();
    } //end else

} //end else

} //end of function

/**
Function:      execute_ok
Purpose:      Low level function used by actionPerformed to implement callbacks for
               the ok button.
Date:         March 20, 1999
In:           None.
Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

*/
void execute_ok()
{

    _init_dialog.setVisible(false); //hide the _init_dialog box

    //enable the trace options
    _trace.setEnabled(true);

```

```

        if(!_TraceFlag)
            _run.setEnabled(true);

    try
    {
        _runner.resume();
    }//end try

    catch(Exception ie) {}

}

/**
Function:      execute_link
Purpose:      Low level function used by actionPerformed to implement callbacks for
               the Documentation button.
Date:         March 20, 1999
In:           None.
Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

*/
void execute_link()
{
    if(_qNotes!= null)
        getAppletContext().showDocument(_qNotes);
}

/**
Function:      itemStateChanged
Purpose:      Call back for the comboboxes.
Date:         February 27, 1999
In:           e: ItemEvent object that tracks changes in
               the comboboxes.
Out:

```

Requires: None.
 Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

*/

```
public void itemStateChanged(ItemEvent e)
{
    if(e.getSource() == _state_options) // state selected
    {
        switch(_state_options.getSelectedIndex())
        {
            //begin switch
            case 0: //2 states
            {
                _total_states = 2*2;
                _states =2;
            }
            break;

            case 1: // 3 states
            {
                _total_states = 2*2*2;
                _states =3;
            }
            break;

            case 2: // 4 states
            {
                _total_states = 2*2*2*2;
                _states =4;
            }
            break;

            case 3: // 5 states
            {
                _total_states = 2*2*2*2*2;
                _states =5;
            }
            break;
        }
    }
}
```

```

        }//end switch

        _okToRun = true; // set flag ok to run the program
    }//end of if

else if(e.getSource()==_speed_options)
{
    switch(_speed_options.getSelectedIndex())
    { //begin switch

        case 0: // slow
        {
            _speed = 2000;
            //save the current speed
        } //end case 0
        break;

        case 1: //medium
        {
            _speed= 1000;
        } //end case 1
        break;

        case 2: //fast
        {
            _speed = 500;
        } //end case 2
        break;
    } //end switch

    _oldSpeed = _speed;
} //end else

} //end of function

```

//the threads

/**

Function: start
 Purpose: Initializes the thread, _runner.
 Date: February 27, 1999

In: None.
 Out: The thread, `_runner`, is initialized.
 Requires: None.
 Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

*/

```

public void start()
{
  if(_runner == null)
  {
    _runner = new Thread(this);
    _runner.start();
  } //end if
} //end of function

```

/**

Function: stop
 Purpose: Stops the thread, `_runner`.
 Date: February 27, 1999
 In: None.
 Out: The thread, `_runner`, is stopped.
 Requires: None.
 Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

*/

```

public void stop()
{
  if(_runner != null)
    _runner = null;
} //end of function

```

```

/**
Function:      run
Purpose:      Defines the action of the thread, _runner.
Date:         February 27, 1999
In:           None.
Out:          The thread, _runner, is activated.
Requires:     None.
Author:       Terrance Bing-Parks:
              Advisor:
              Dr. Walter Kirchherr
              Department of Mathematics and Computer Science
              San Jose State University
              San Jose, California

*/

public void run()
{

    //Get the current thread of execution.
    Thread thisThread = Thread.currentThread();

    //If we should begin animating, then do so.
    if(_goodToGo == true)
    {
        while(_runner == thisThread) /* This is true as long as the
                                     thread, _runner is not null.
                                     */
        {
            the_algo(_states); // The top level function call in the
                               // in Grover's algorithm.
            _goodToGo = false; // stop animating

            break;
        } //end of while
    } //end of if
} //end of run

```

/**

Function: make_one

Purpose: Makes the column vector: $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Date: February 27, 1999

In: None.

Out: A matrix that represents the column vector: $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Requires: None.

Author: Terrance Bing-Parks:

Advisor:

Dr. Walter Kirchherr

Department of Mathematics and Computer Science

San Jose State University

San Jose, California

*/

matrix make_one()

```
{
    matrix C = new matrix(2,1);
    C.set_element(0,0,0.0);
    C.set_element(1,0,1.0);
    return C;
}
```

/**

Function: make_zero

Purpose: Makes the column vector: $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

Date: February 27, 1999

In: None.

Out: A matrix that represents the column vector: $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

Requires: None.

Author: Terrance Bing-Parks:

Advisor:

Dr. Walter Kirchherr

Department of Mathematics and Computer Science

San Jose State University

San Jose, California

*/

```
matrix make_zero()
{
    matrix C = new matrix(2,1);
    C.set_element(0,0,1.0);
    C.set_element(1,0,0.0);
    return C;
}
```

/**

Function: make_identity
 Purpose: Makes an identity matrix of size: $2^{\text{states}} \times 2^{\text{states}}$.
 Date: February 27, 1999
 In: numStates: The number of bits selected by the user.
 Out: A matrix that represents makes an identity matrix of
 size: $2^{\text{states}} \times 2^{\text{states}}$.
 Requires: None.
 Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

*/

```
matrix make_identity(int states)
{
```



```

// matrix that is to be returned
matrix C = new
matrix((int)Math.pow(2.0,(double)states),(int)Math.pow(2.0,(double)states));

//Make the identity matrix.
for(int i =0; i<C._rows; i++)
{
    for(int j =0; j<C._columns; j++)
    {
        if(i!=j)
            C._values[i][j]=0.0;
        else
            C._values[i][j]=1.0;
    }
}
return C;
} //end of function

```

```

/**
Function:      make_I_sub_T
Purpose:      Makes a diagonal matrix of size: 2^states X 2^states
               where all the diagonal are 1, except the element
               _special_loc (_special_loc is the target state)
               whose value is -1.

Date:         February 27, 1999
In:           numStates: The number of bits selected by the user.
Out:          A diagonal matrix of size: 2^states X 2^states
               where all the diagonal are 1, except the element
               _special_loc, whose value is -1.

Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

```

```

*/

```

```

matrix make_I_sub_T( int numStates)
{

    matrix C = make_identity(numStates);

```

```
C.set_element(_special_loc, _special_loc, -1.0);
```

```
return C;
```

```
//end of function
```

```
/**
```

```
Function:    make_I_sub_O
```

```
Purpose:      Makes a diagonal matrix of size:  $2^{\text{states}}$  X  $2^{\text{states}}$   
              where all the diagonal are 1, except the element  
              0,0 whose value is -1.
```

```
Date:       February 27, 1999
```

```
In:         numStates: The number of bits selected by the user.
```

```
Out:        Makes a diagonal matrix of size:  $2^{\text{states}}$  X  $2^{\text{states}}$   
              where all the diagonal are 1, except the element  
              0,0 whose value is -1.
```

```
Requires:   None.
```

```
Author:     Terrance Bing-Parks:
```

```
Advisor:
```

```
Dr. Walter Kirchherr
```

```
Department of Mathematics and Computer Science
```

```
San Jose State University
```

```
San Jose, California
```

```
*/
```

```
matrix make_I_sub_O( int numStates)
```

```
{
```

```
matrix C = make_identity(numStates);
```

```
C.set_element(0,0, -1.0);
```

```
return C;
```

```
//end of function
```

```
/**
```

```
Function:    make_W
```

```
Purpose:      Applies the matrix  $M = 1/\text{sqr}(2) * \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ 
```

$\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix}$

to numStates qubits to make the Walsh-Hadamard transform.

Date: March 12, 1999

In: numStates: The number of bits selected by the user

Out: a matrix that represents the Walsh-Hadamard transform.

Requires: None.

Author: Terrance Bing-Parks:
Advisor:
Dr. Walter Kirchherr
Department of Mathematics and Computer Science
San Jose State University
San Jose, California

*/

```
matrix make_W(int numStates)
{
    matrix M = new matrix(2,2);
    //M = 1/sqr(2) *  $\begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix}$ 
    //
    M.set_element(0,0,.707106781);
    M.set_element(0,1,.707106781);
    M.set_element(1,0,.707106781);
    M.set_element(1,1,-.707106781);

    matrix temp1 = new matrix (4,4);

    matrix temp2 = new matrix(8,8);

    matrix temp3 = new matrix(16,16);

    matrix temp4 = new matrix(32,32);
```

```
switch(numStates)
{
    /* No breaks need because we return out of each case. */
    case 2:
```

```

{
temp1 = M.direct(M, M);

return temp1;
}

case 3:
{
temp1 = M.direct(M, M);

temp2=temp1.direct(M, temp1);
return temp2;
}

case 4:
{
temp1 = M.direct(M, M);

temp2=temp1.direct(M, temp1);

temp3 = temp2.direct(M, temp2);

return temp3;
}

default:
{
temp1 = M.direct(M, M);

temp2=temp1.direct(M, temp1);

temp3 = temp2.direct(M, temp2);

temp4 = temp3.direct(M, temp3);

return temp4;
}

} //end of switch

} //end of function

```

/**

Function: make_init_reg

Purpose: Makes a column matrix of size: 2^{states} X 1
 where all the diagonal are 0, except the element
 0,0 whose value is 1.

Date: March 12, 1999

In: numStates: The number of bits selected by the user

Out: Makes a column matrix of size: 2^{states} X 1
 where all the diagonal are 0, except the element
 0,0 whose value is 1.

Requires: None.

Author: Terrance Bing-Parks:

Advisor:

Dr. Walter Kirchherr

Department of Mathematics and Computer Science

San Jose State University

San Jose, California

*/

matrix make_init_reg(int numstates)

{

matrix zero = new matrix(2,1);

zero= make_zero();

matrix temp1 = new matrix(4,1);

matrix temp2 = new matrix(8,1);

matrix temp3 = new matrix (16,1);

matrix temp4 = new matrix (32,1);

switch(numstates)

{

case 2:

{

temp1 = zero.direct(zero,zero);

return temp1;

}

case 3:

```

{

    temp1 = zero.direct(zero,zero);
    temp2 = zero.direct(zero, temp1);
    return temp2;
}

case 4:
{

    temp1 = zero.direct(zero,zero);
    temp2 = zero.direct(zero, temp1);
    temp3 = zero.direct(zero, temp2);
    return temp3;
}
default:
{

    temp1 = zero.direct(zero,zero);
    temp2 = zero.direct(zero, temp1);
    temp3 = zero.direct(zero, temp2);
    temp4 = zero.direct(zero,temp3);
    return temp4;
}

} //end of switch

} //end of function

/**
Function:      iterate
Purpose:      Applies the quantum transform Q
               (See Lov K. Grover: "A framework for fast quantum
               mechanical algorithms.", p. 5).
Date:         March 12, 1999
In:           None.
Out:          Matrix that represents the quantum transform Q.
Requires:     None.
Author:       Terrance Bing-Parks:

```

Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

```

*/
matrix iterate( int numStates)
{

    matrix Io = make_I_sub_O(numStates);
    matrix It = make_I_sub_T(numStates);
    matrix W = make_W(numStates);

    matrix R = new matrix(_total_states,1);
    boolean draw = true;
    double percentage = 0.0;
    R= make_init_reg(_states);

    for (_turns =0; _turns<_max_turns; _turns++)
    {
        _dialog_label1.setText("Quantum step number: "+(_turns+1));

        try
        {
            R = R.multiply(W,R);

            if(_turns==0)
            {

                _dialog_label2.setText("Applied the Walsh-Hadamard transform to the start");

                _dialog_label3.setText("state (location 1).");

                _dialog_label4.setText("");
            }
        }
    }
}

```

```

    }
    else
    {

        _dialog_label2.setText("Applied the Walsh-Hadamard transform to the");

        _dialog_label3.setText("results of the previous application of the transform");

        _dialog_label4.setText("that negated all states, except the start state (location 1).");

    }

    get_lengths(R,draw);

}
catch(matrix_error e){System.out.println("Matrix Multiply error");}

try
{
    R = R.multiply(It,R);

    _dialog_label2.setText("Applied a transform to invert the target state");

    _dialog_label3.setText("(location "+(_special_loc+1)+") to the results of the previous
application");

    _dialog_label4.setText("of the Walsh-Hadamard transform.");

    get_lengths(R,draw);
}
catch(matrix_error e){System.out.println("Matrix Multiply error");}

try
{
    R = R.multiply(W,R); //rotate about the average

    _dialog_label2.setText("Applied the Walsh-Hadamard transform to the");

    _dialog_label3.setText("results of the previous application of the transform ");

```



```

        _dialog_label4.setText("that inverted the target state (location
"+(_special_loc+1)+").");

        get_lengths(R,draw);
    }
    catch(matrix_error e){System.out.println("Matrix Multiply error");}
    try
    {

        R = R.multiply(Io,R);

        R.scalar_mult(-1.0);

        _dialog_label2.setText("Applied the transform to invert all states, except the");

        _dialog_label3.setText("start state (location 1), to the results of the previous");

        _dialog_label4.setText("application of the Walsh-Hadamard transform.");

        get_lengths(R,draw);

    }
    catch(matrix_error e){System.out.println("Matrix Multiply error");}

}

return R;
}

/**
Function:    get_lengths
Purpose:    Puts the dynamic lengths of the column vectors
            in the quantum gate R in the array

```

length_array and then draw these lengths in the applet.

Date: March 12, 1999

In: R: A matrix.
draw: A flag that tells us whether or not to draw to the device context.

Out: None.

Requires: None.

Author: Terrance Bing-Parks:
Advisor:
Dr. Walter Kirchherr
Department of Mathematics and Computer Science
San Jose State University
San Jose, California

```

*/
void get_lengths(matrix R, boolean draw)
{
    R.calculate_column_values();
    for(int i=0; i<_total_states; i++)
    {
        _length_array[i]._length= (R._the_columns[i].length)*
                                   (R._the_columns[i].sign);
    }//end for

    if(draw)
    {
        //Prepare to draw.
        _drawLineFlag = true;

        //Draw.
        repaint();

        try
        {
            Thread.sleep(_speed);
        }
        catch(InterruptedException ie) {System.out.println("Thread error");}

        if( _TraceFlag )
        {
            try
            {

```

```

    _runner.suspend();

    }//end try
    catch(Exception ie) {};
} //end if
}
} //end of function

/**
Function:      initialize_the_algo
Purpose:      Initializes the variables need to animate
               Lov K. Grover's search algorithm.

Date:         March 11, 1999
In:           None.
Out:          The variables need to animate
               Lov K. Grover's search algorithm are initialized.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

*/

void initialize_the_algo()
{

    _clearFlag= true;
    repaint();

    //Calculate the number of iterations that we must perform.
    _max_turns= (int)((Math.PI/4.0)*Math.sqrt((double)_total_states));

    //Calculate the minimum probability of success.
    double prob_temp = 1.0 - (1.0/(double)_total_states);
    int temp =(int) (1000.00*prob_temp);

    _predicted_prob = ((double)temp)/10.00;

```

```

_turns =0;

_prompt.setText("");

_prompt2.setText("");
Random r = new Random();

_length_array = new length_array_elements[_total_states];

for(int index=0; index<_total_states;index++)
    _length_array[index] = new length_array_elements();

_special_loc = (int)(r.nextDouble()* _total_states);

    //Put some random values in the numbers.
    for(int i=0; i<_total_states; i++)
        _length_array[i]._stored_val = (int) (r.nextDouble()*1000);

        _length_array[_special_loc]._stored_val = _randValue;

//deal with the dialog
if (_max_turns>1)
{
    _init_dialog_label1.setText("The target location is location
"+(_special_loc+1)+".");
    _init_dialog_label2.setText("We predict that this location can be isolated");
    _init_dialog_label3.setText("with a probability of at least "+ _predicted_prob +
"% in "+ _max_turns);
    _init_dialog_label4.setText("quantum steps. Press OK to begin.");
}
else
{
    _init_dialog_label1.setText("The target location is location
"+(_special_loc+1)+".");
    _init_dialog_label2.setText("We predict that this location can be isolated");
    _init_dialog_label3.setText("with a probability of at least "+ _predicted_prob +
"% in "+ _max_turns);
    _init_dialog_label4.setText("quantum step. Press OK to begin.");
}

```

```

// Disable the trace button.

    _trace.setEnabled(false);

    _run.setEnabled(false);

    _init_dialog.setVisible(true);

/* Suspend until the dialog is visible. */
try
{
    _runner.suspend();
}

catch(Exception ie) {}

    if(_TraceFlag == false)
        _prompt.setText("Isolating location
"+(_special_loc+1)+".");

    else
    {
        _prompt.setText("Isolating location "+(_special_loc+1)+" using trace mode.
Press the Tracing... button ");
        _prompt2.setText("to step through the isolation. Press the Run button to run at
normal speed.");
    }

}

/**
Function:    find_percentage
Purpose:    Finds the maximum length of the column vectors

```

in the array lenght_array.

Date: March 7, 1999
 In: None.
 Out: The percentage of the length of the target state as compared to the total length of all the states.
 Requires: None.
 Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

```

*/

double find_percentage(int location)
{
    double sum = 0.0;

    for(int i=0; i<_total_states; i++)
    {
        double temp = _length_array[i]._length;
        sum += Math.pow(Math.abs(temp),2);
    }//end for

    double spec_length = Math.pow(Math.abs(_length_array[location]._length ), 2);

    double val = (spec_length/sum);
    Double testIfInfinite = new Double(val);

    int temp =(int) (1000.00*val);

    double returnVal= ((double)temp)/10.00;

    if (testIfInfinite.isInfinite())
        return 100.00;

    else
    {
        if(returnVal>100.00)
            return 100.00;

        return returnVal;
    }

```

```
//end of find_maxHeight
```

```
/**
```

```
Function:    the_algo
```

```
Purpose:      Top level implementation the Lov K. Grover's
              quantum search algorithm and animation
              (See Lov K. Grover: "A framework for fast quantum
              mechanical algorithms.", p.5).
```

```
Date:       March 7, 1999
```

```
In:         None.
```

```
Out:        Matrix which represents the quantum transform  $Q * W$ .
```

```
Requires:   None.
```

```
Author:     Terrance Bing-Parks:
```

```
Advisor:
```

```
Dr. Walter Kirchherr
```

```
Department of Mathematics and Computer Science
```

```
San Jose State University
```

```
San Jose, California
```

```
*/
```

```
matrix the_algo (int numStates)
```

```
{
```

```
    //the W-H transform
```

```
    matrix W = make_W(numStates);
```

```
    initalize_the_algo();
```

```
    _clearFlag= true;
```

```
    repaint();
```

```
//Hiding the dialog messes up the screen, so wait for awhile before resuming.
```

```
try
```

```
{
```

```
    Thread.sleep(_clearTime);
```

```
}
```

```
catch(InterruptedException ie) {System.out.println("Thread error");}
```

```
boolean draw = true;
```

```
//Iterate the required number of times.
```

```
matrix itresults = iterate(numStates);
```

```
    _dialog_label2.setText("Applied the Walsh-Hadamard transform to the");
```

```
    _dialog_label3.setText("results of the previous application of the transform");
```

```
    _dialog_label4.setText("that inverted all states, except the start state (location 1).");
```

```
try{
itresults= W.multiply(W,itresults);
}
catch(matrix_error e){System.out.println("Matrix Multiply error");}
```

```
get_lengths(itresults,draw);
```

```
double percentage= find_percentage(_special_loc);
```

```
/* See if _special location height equals the max height (or -.01% of the
max height. */
```

```
if(percentage>= 50.00) //
{
    if(_turns ==1)
    {
        _prompt.setText("Isolation completed! The predicted probability of success was
at least "+ _predicted_prob+ "%. The value");
        _prompt2.setText("will be found at location " + (_special_loc +1)+" in "+_turns+"
quantum step with a probability of "+percentage+ "%." );
    }
    else
    {
        _prompt.setText("Isolation completed! The predicted probability of success was at
least "+ _predicted_prob+ "%. The value");
        _prompt2.setText("will be found at location " + (_special_loc +1)+" in "+_turns+"
quantum steps with a probability of "+percentage+ "%." );
    }
}
} //end if
```



```

    }

//reset flags

if(!_TraceFlag) //reset
    _TraceFlag=false;

    _state_options.setEnabled(true);

    _running = false;

//Clear the dialog labels and hide the dialog box.

    _dialog_label1.setText("");
    _dialog_label2.setText("");
    _dialog_label3.setText("");
    _dialog_label4.setText("");

    _dialog.setVisible(false);

//Enable the run button.
_run.setEnabled(true);

return itresults;

} //end of function

// for double buffering

/**
Function:      update
Purpose:      Overridden function used to help minimize flickering.
Date:         February 27, 1999
In:           g: a Graphics context
Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
              Advisor:
              Dr. Walter Kirchherr
              Department of Mathematics and Computer Science

```

San Jose State University
San Jose, California

```

*/
public void update(Graphics g)
{
    paint(_offscreen);

    //Draw the image to the real screen.
    g.drawImage(_offScreenImg, 0, 0, this);

} //end of function


/**
Function:      paint
Purpose:      Overridden function used to draw the
               dynamic lengths of the column vectors on the applet.
Date:         February 27, 1999
In:           g: A Graphics context.
Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

*/

public void paint(Graphics g)
{

    if(_clearFlag == true)

    {
        g.clearRect(0,0, getSize().width, getSize().height);
    }

```

```

    _clearFlag= false;
    return;
}

```

```

if(_drawLineFlag == true)
{
    drawPicture(g);
    _drawLineFlag=false;
}
} //end of function

```

```

/**
Function:      drawPicture
Purpose:      Function used to clear the applet and draw the
               dynamic lengths of the column vectors on the applet.
Date:         March 7, 1999
In:           g: A Graphics context.
Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

*/

```

```

void drawPicture(Graphics g)
{
    _offset = _LineLength/_total_states;
    _x = _xStart;

    boolean gold;

    g.clearRect(0,0, getSize().width, getSize().height);

```

```

    if(_total_states<=16)
    {
//Draw line.
    g.drawLine(_xStart,_yStart,_xEnd,_yStart);

//Draw numbers.
    for(int j=0; j<_total_states; j++)
    {
        g.drawString(""+ (j+1), _x+2,_yStart + 10 );

        if(j== _special_loc)
        {
            g.setColor(Color.red);

            g.drawString("*", _x+2,_yStart + 25 );

            g.setColor(Color.black);
        }//end if
        _x += _offset;
    }//end for

    _x = _xStart;

//Draw histogram.
    for(int i=0; i<_total_states; i++)
    {
        if(_length_array[i]._length<0.0)
            g.setColor(Color.red);

        g.drawLine(_x,_yStart, _x,_yStart - (int)(_scale*_length_array[i]._length));

        if( i==_special_loc)
            gold = true;
        else
            gold = false;
        drawPercents(g, i , _x, _yStart - (int)(_scale*_length_array[i]._length),gold);
    }
}

```

```

    g.setColor(Color.black);
    _x += _offset;
}
}

else if(_total_states==32)
{

int Y1 = _yStart + _Y1_offset;
int Y2 = _yStart + _Y2_offset;

_offset = _LineLength/16; //use this offset for all other values

//Draw the first line.
//drawline 1
g.drawLine(_xStart,Y1,_xEnd,Y1);

//Drawline 2.
g.drawLine(_xStart,Y2,_xEnd,Y2);

//Draw numbers: first set.
for(int j=0; j<_total_states/2; j++)
{
g.drawString(""+ (j+1), _x+2,Y1 + 10 );

if(j== _special_loc)
{
g.setColor(Color.red);

g.drawString("*", _x+2,Y1 + 25 );

g.setColor(Color.black);
} //end if
_x += _offset;
} //end for

_x=_xStart;

//Draw numbers: first set.
for(int k=_total_states/2; k<_total_states; k++)
{
g.drawString(""+ (k+1), _x+2,Y2 + 10 );

```

```

if(k== _special_loc)
{
    g.setColor(Color.red);

    g.drawString("*", _x+2,Y2 + 25 );

    g.setColor(Color.black);
} //end if
_x += _offset;
} //end for

_x=_xStart;

//Draw histogram, first set.
for(int l=0; l<_total_states/2; l++)
{
    if( _length_array[l]._length<0.0)
        g.setColor(Color.red);
    g.drawLine(_x,Y1, _x,Y1 - (int)(_scale*_length_array[l]._length));

    if( l==_special_loc)
        gold = true;
    else
        gold = false;
    drawPercents(g,l, _x, Y1 - (int)(_scale*_length_array[l]._length),gold);

    g.setColor(Color.black);
    _x += _offset;
}

_x=_xStart;

//Draw histogram, second set.
for(int m=_total_states/2; m<_total_states; m++)
{
    if( _length_array[m]._length<0.0)
        g.setColor(Color.red);

    g.drawLine(_x,Y2, _x,Y2 - (int)(_scale*_length_array[m]._length));

```

```

if( m==_special_loc)
    gold = true;
else
    gold = false;
drawPercents(g,m , _x, Y2 - (int)(_scale*_length_array[m]._length),gold);

    g.setColor(Color.black);
    _x += _offset;
}

_x=_xStart;

} //end if

} //end of function

```

```

/**
Function:      drawPercents
Purpose:      Function used to clear the applet and draw the
               dynamic lengths of the column vectors on the applet.
Date:         March 7, 1999
In:           g: A Graphics context.
               location: The location to write percentages over.
               x: The x coordinate of the location to write percentages over.
               y: The y coordinate of the location to write percentages over.
               gold: Flag that tells if location is the target location.

Out:          None.
Requires:     None.
Author:       Terrance Bing-Parks:
               Advisor:
               Dr. Walter Kirchherr
               Department of Mathematics and Computer Science
               San Jose State University
               San Jose, California

*/

```

```

void drawPercents(Graphics g, int location, int x, int y, boolean gold)
{
    double percent = 0.0;
    int xOffset = -5;
    int yOffset;

    if(_length_array[location]._length<0.0)
    {
        if (gold==true)
            yOffset =15;
        else
            yOffset =20;
    }
    else
    {
        if (gold==true)
            yOffset = -5;
        else
            yOffset = -5;
    }
    //Set the color.
    if(!gold)
        g.setColor(Color.blue);
    else
        g.setColor(Color.red);

    double printVal =find_percentage(location);

    //Draw the number.
    g.drawString(""+printVal+"%",x+ xOffset, y + yOffset);

    //Reset the color.
    g.setColor(Color.black);

}

```



```
//destroy offscreen image
```

```
/**
```

```
Function:    destroy
```

```
Purpose:      Overridden function used to destroy the  
             graphic context, _offscreen.
```

```
Date:       February 27, 1999
```

```
In:         None.
```

```
Out:        None.
```

```
Requires:   None.
```

```
Author:     Terrance Bing-Parks:
```

```
Advisor:
```

```
Dr. Walter Kirchherr
```

```
Department of Mathematics and Computer Science
```

```
San Jose State University
```

```
San Jose, California
```

```
*/
```

```
public void destroy()
```

```
{  
    _offscreen.dispose();  
}
```

```
}//end of class grover_animation
```

```
/**
```

```
Class:      matrix_error
```

```
Date:       February 27, 1999
```

```
Purpose:      Object thrown by the class matrix when  
             there is a multiplication error.
```

```
Author:     Terrance Bing-Parks:
```

```
under direction of Dr. Walter Kirchherr of
```

```
Department of Mathematics and Computer Science
```

```
San Jose State University
```

```
San Jose, California
```

```
*/
```

```
class matrix_error extends Exception
```

```
{}
```

```

/**
    Class:      column_element
    Date:       February 27, 1999
    Purpose:    Class that statically stores the lengths and sign
                of column vectors in a matrix.
    Author:     Terrance Bing-Parks:
                Advisor:
                Dr. Walter Kirchherr
                Department of Mathematics and Computer Science
                San Jose State University
                San Jose, California

```

```

*/

```

```

class column_element
{
    public double length;
    public double sign;
}

```

```

/**
    Class:      length_array_elements
    Date:       February 27, 1999
    Purpose:    Class that dynamically stores the length and value
                of column vectors in a matrix as it is operated on
                quantum gates.
    Author:     Terrance Bing-Parks:
                Advisor:
                Dr. Walter Kirchherr
                Department of Mathematics and Computer Science
                San Jose State University
                San Jose, California

```

```

*/

```

```

class length_array_elements
{

```

```

    double _length;
    int _stored_val;
}

/**
    Class:      matrix
    Date:       February 27, 1999
    Purpose:    Class which make matrix objects that has methods which
                allows for:
                1) the computing of the product of two matrices;
                2) the computing of the direct product of two matrices;
                3) the computing of the product of a scalar and a matrix.

    Author:     Terrance Bing-Parks:
                Advisor:
                Dr. Walter Kirchherr
                Department of Mathematics and Computer Science
                San Jose State University
                San Jose, California

*/

class matrix
{
    //data members
    int _rows;      // the number of rows in the matrix
    int _columns;   // the number of columns in the matrix

    double _values[ ][ ]; //array that stores the values in the matrix

    column_element _the_columns[ ]; // the length and sign of the column vectors in
                                    // matrix

    //Begin function definitions.

    /**
    Function:    matrix
    Purpose:    Constructs matrix elements.
    Date:       February 27, 1999
    In:         rows: The number of rows that are to be in the matrix.
                columns: The number of columns that are in the matrix.
    */

```

Out: A matrix object.
 Requires: None.
 Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

*/

```
matrix (int rows, int columns)
{
  _values = new double[rows][columns];
  _rows = rows;
  _columns = columns;

  _the_columns = new column_element[rows];

  for(int i=0; i<rows;i++)
    _the_columns[i] = new column_element();

} //end of constructor
```

/**

Function: direct
 Purpose: Computes the direct product of two matrices.
 Date: February 27, 1999
 In: A: A matrix.
 B: A matrix.
 Out: A matrix object that is the direct product of A and B.
 Requires: None.
 Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

*/

matrix direct (matrix A, matrix B)

```

{
int As_Rows; // index value that holds the current row in matrix A
int As_Column; // index value that holds the current column in matrix B

int row_displacement; /* offset used to put the product of elements
                        B[i][j] * multiplier in the correct row of the
                        output matrix
                        */
int column_displacement; /* offset used to put the product of elements
                           B[i][j] * multiplier in the correct column of the
                           output matrix
                           */

double multiplier; // Each i value of matrix A becomes a multiplier.

//Create the matrix to be returned.
matrix C = new matrix(A._rows* B._rows,A._columns* B._columns);

//Compute the direct product.

for(As_Column = 0; As_Column<A._columns; As_Column++)

{ //1
for(As_Rows = 0; As_Rows<A._rows; As_Rows++)
{ //2
multiplier = A._values[As_Rows][As_Column];

for(int row = 0; row<B._rows; row++)

{ //3
for( int column =0; column<B._columns; column++)
{ //4
if(B._values[row][column]== -0.0)
B._values[row][column]= 0.0;

double temp = multiplier *B._values[row][column]; /* current value to
                                                    be stored.
                                                    */

//Compute the correct output column.
if(As_Column>0)

```

```

        column_displacement = ((As_Column + 1 * B._columns) - 1);

    else
        column_displacement = 0;

    if(As_Rows > 0)
        row_displacement = ((As_Rows + 1 * B._rows) - 1);
    else
        row_displacement = 0;

    //Store the current value.
    C._values[row + row_displacement][column + column_displacement] = temp;

} //4

} //3

} //2

} //1

return C;

} //end direct

```

```

/**
Function:    multiply
Purpose:    Computes the product of two matrices.
Date:       March 7, 1999
In:         A: A matrix.
           B: A matrix.
Out:        A matrix object that is the product of A and B.
Requires:   The number of columns in matrix A equals the number of
           rows in matrix B. If this requirement is not met, a
           matrix error is thrown.
Author:     Terrance Bing-Parks:
           Advisor:
           Dr. Walter Kirchherr
           Department of Mathematics and Computer Science
           San Jose State University
           San Jose, California

*/

```

```

matrix multiply(matrix A, matrix B) throws matrix_error
{
    int Bs_Columnn; // index that holds the current column in matrix B
    double sum = 0.0; //holds the sum of row A_i and column B_i

    // Create the matrix that is to be returned.
    matrix C = new matrix(A._rows, B._columns);

    // Keep to see if multiplication is possible.
    if (A._columns != B._rows)
        throw new matrix_error();

    int N = A._columns;

    for(int i =0; i<B._columns; i++)
        for(int j =0; j<A._rows; j++)
        {
            C._values[j][i]=0.0;
            for(int k =0; k<A._columns; k++)
                C._values[j][i] += A._values[j][k] *B._values[k][i];
        }

    return C;

} // end of function

/**
Function:      scalar_mult
Purpose:      Computes the scalar product.
Date:         February 27, 1999
In:           val: A scalar.
Out:          A matrix object that has all of its elements multiplied by
              val.
Requires:     None.

```

Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California

*/

```
void scalar_mult(double val)
{
    //Compute the scalar product.
    for(int i=0; i<_rows; i++)
    {
        for(int j=0; j<_columns; j++)
        {
            if(_values[i][j]!=0.0)
                _values[i][j] *=val;
        }
    }
}
```

}//end of print

/**

Function: calculate_column_values
 Purpose: Computes the sign and length of the column vectors in
 a matrix.

Date: February 27, 1999
 In: None.
 Out: The sign and lengths of the column vectors in
 the implicit object (matrix) are stored in there
 representative spaces in the array _the_columns.

Requires: None.

Author: Terrance Bing-Parks:
 Advisor:
 Dr. Walter Kirchherr
 Department of Mathematics and Computer Science
 San Jose State University
 San Jose, California


```

*/

void calculate_column_values()
{

    for (int i = 0; i<_rows; i++)
    {
        double temp = _values[i][0];
        _the_columns[i].length = Math.abs(temp);
        if(temp>=0.0)
            _the_columns[i].sign = 1.0;
        else
            _the_columns[i].sign = -1.0;
    }

    }//end of function

/**
Function:    print()
Purpose:    Prints the elements the matrix (used for debugging).
Date:       February 27, 1999
In:         None.
Out:        The elements of a matrix are printed to standard output.
Requires:   None.
Author:     Terrance Bing-Parks:
            Advisor:
            Dr. Walter Kirchherr
            Department of Mathematics and Computer Science
            San Jose State University
            San Jose, California

*/

void print()
{
    for(int i =0; i<_rows; i++)

        {
            for(int j =0; j<_columns; j++)
            {
                System.out.print(" "+ _values[i][j]+" ");
            }
        }
}

```

```
    }  
    System.out.println("");  
  }  
} //end of print
```

```
void set_element(int row, int col, double val)  
{  
  _values[row][col] = val;  
}
```

```
} //end of class matrix
```

Appendix C grover_animation.html

```
<HTML>  
<HEAD>  
<TITLE> Quantum Computing Animation </TITLE>  
</HEAD>  
<BODY>
```

```
<APPLET CODE = "grover_animation.class" CODEBASE ="Algorithm/java_exe/"  
WIDTH =550 HEIGHT = 410>  
</APPLET>  
</BODY>  
</HTML>
```

Appendix D verification.txt

```

# Verification of grover_animation.java
> # for 2-qubits and 3-qubits.
> with(linalg):
Warning, new definition for norm
Warning, new definition for trace
> # Verification for 2-qubits.
> # Define the start state.
> Start_state := matrix(4,1,[1,0,0,0]);

                                [1]
                                [ ]
                                [0]
Start_state := [ ]
                                [0]
                                [ ]
                                [0]

> # Define the Walsh-Hadamard Transform
> # for 2-qubits.
> W_2a := matrix(4,4,[[1,1,1,1],[1,-1,1,-1],[1,1,-1,-1],[1,-1,-1,1]]);

                                [1      1      1      1]
                                [      ]
                                [1     -1      1     -1]
W_2a := [      ]
                                [1      1     -1     -1]
                                [      ]
                                [1     -1     -1      1]

> # Define the scalar to multiply W_2 by.
> scal := (1/(sqrt(2)))*(1/(sqrt(2)));

                                scal := 1/2

> # Now multiply W_2 times the scal.
> W2 := matrix (4,4);

                                W2 := array(1 .. 4, 1 .. 4, [])

> W2 := scalarmul(W_2a,scal);

                                [1/2     1/2     1/2     1/2 ]
                                [      ]
                                [1/2     -1/2     1/2     -1/2]
W2 := [      ]
                                [1/2     1/2     -1/2     -1/2]
                                [      ]
                                [1/2     -1/2     -1/2     1/2 ]

> # Let the initial target state be location two.
> # Define the target state matrix.
> I_t2 := matrix(4,4, [[1,0,0,0],[0,-1,0,0],[0,0,1,0],[0,0,0,1]]);

                                [1      0      0      0]
                                [      ]

```

```

I_t2 :=
[0  -1  0  0]
[
[0  0  1  0]
[
[0  0  0  1]

> # Define the I sub zero matrix.
> I_o2a := matrix(4,4, [[-1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]);

I_o2a :=
[ -1  0  0  0]
[
[ 0  1  0  0]
[
[ 0  0  1  0]
[
[ 0  0  0  1]

> I_o2 := matrix(4,4);

I_o2 := array(1 .. 4, 1 .. 4, [])

> I_o2 := scalarmul(I_o2a, -1);

I_o2 :=
[1  0  0  0]
[
[0  -1  0  0]
[
[0  0  -1  0]
[
[0  0  0  -1]

> # Define Q.
> Q2 :=matrix(4,4, multiply(I_o2,W2,I_t2,W2));

Q2 :=
[1/2  1/2  -1/2  1/2 ]
[
[-1/2  -1/2  -1/2  1/2 ]
[
[1/2  -1/2  -1/2  -1/2]
[
[-1/2  1/2  -1/2  -1/2]

> # Apply 1 iteration for 2-qubits.
> two_qubits_result := matrix (4,1, multiply(W2,Q2,Start_state));

two_qubits_result :=
[0]
[ ]
[1]
[ ]
[0]
[ ]
[0]

> # As we see from the results in the variable, two_qubit_results, all
the

```

```

> # amplitudes are all zero, except for the target location.
> Sum_of_ampl2:= 0 + 0 + (1^2) +0;

        Sum_of_ampl2 := 1

> target_length2 := (1)^2;

        target_length2 := 1

>

> # Calculate the probability of finding the target -- location two.
> prob_of_finding_taget2 := target_length2/Sum_of_ampl2;

        prob_of_finding_taget2 := 1

> # Thus, we see that we have a 100% probability of finding the target
location.
>
>
> # Now, let us calculate the probability of finding an item in a 3-
qubit system.
> # Verification for 3-qubits.
> # Define the start state.
> Start_state3 := matrix(8,1,[1,0,0,0,0,0,0,0]);

        [1]
        [ ]
        [0]
        [ ]
        [0]
        [ ]
        [0]
Start_state3 := [ ]
        [0]
        [ ]
        [0]
        [ ]
        [0]
        [ ]
        [0]

> # Define the Walsh-Hadamard Transform
> # for 3-qubits.
> W_3a := matrix(8,8,[[1,1,1,1,1,1,1,1],[1,-1,1,-1,1,-1,1,-1],[1,1,-1,-
1,1,1,-1,-1],[1,-1,-1,1,1,-1,-1,1],[1,1,1,1,-1,-1,-1,-1],[1,-1,1,-1,-
1,1,-1,1],[1,1,-1,-1,-1,-1,1,1],[1,-1,-1,1,-1,1,1,-1]]);

        [1      1      1      1      1      1      1      1]
        [      ]
        [1     -1      1     -1      1     -1      1     -1]
        [      ]
        [1      1     -1     -1      1      1     -1     -1]
        [      ]
        [1     -1     -1      1      1     -1     -1      1]

```



```

W_3a := [
[1      1      1      1     -1     -1     -1     -1]
[
[1     -1      1     -1     -1      1     -1      1]
[
[1      1     -1     -1     -1     -1      1      1]
[
[1     -1     -1      1     -1      1      1     -1]
]
]

> # Define the scalar to multiply W_3a by.
> scal2 := (1/(sqrt(2)))* (1/(sqrt(2)))* (1/(sqrt(2)));

          scal2 := 1/4 sqrt(2)

> # Now, multiply W_3a times scal2.
> W3 := matrix (8,8);

          W3 := array(1 .. 8, 1 .. 8, [])

> W3 := scalarmul(W_3a,scal2);

W3 :=

[1/4 sqrt(2) , 1/4 sqrt(2) , 1/4 sqrt(2) , 1/4 sqrt(2) ,
1/4 sqrt(2) , 1/4 sqrt(2) , 1/4 sqrt(2) , 1/4 sqrt(2)]
[1/4 sqrt(2) , - 1/4 sqrt(2) , 1/4 sqrt(2) , - 1/4 sqrt(2) ,
1/4 sqrt(2) , - 1/4 sqrt(2) , 1/4 sqrt(2) , - 1/4 sqrt(2)]
[1/4 sqrt(2) , 1/4 sqrt(2) , - 1/4 sqrt(2) , - 1/4 sqrt(2) ,
1/4 sqrt(2) , 1/4 sqrt(2) , - 1/4 sqrt(2) , - 1/4 sqrt(2)]
[1/4 sqrt(2) , - 1/4 sqrt(2) , - 1/4 sqrt(2) , 1/4 sqrt(2) ,
1/4 sqrt(2) , - 1/4 sqrt(2) , - 1/4 sqrt(2) , 1/4 sqrt(2)]
[1/4 sqrt(2) , 1/4 sqrt(2) , 1/4 sqrt(2) , 1/4 sqrt(2) ,
- 1/4 sqrt(2) , - 1/4 sqrt(2) , - 1/4 sqrt(2) , - 1/4 sqrt(2)
]
[1/4 sqrt(2) , - 1/4 sqrt(2) , 1/4 sqrt(2) , - 1/4 sqrt(2) ,
- 1/4 sqrt(2) , 1/4 sqrt(2) , - 1/4 sqrt(2) , 1/4 sqrt(2)]
[1/4 sqrt(2) , 1/4 sqrt(2) , - 1/4 sqrt(2) , - 1/4 sqrt(2) ,
- 1/4 sqrt(2) , - 1/4 sqrt(2) , 1/4 sqrt(2) , 1/4 sqrt(2)]
[1/4 sqrt(2) , - 1/4 sqrt(2) , - 1/4 sqrt(2) , 1/4 sqrt(2) ,

```

```

- 1/4 sqrt(2) , 1/4 sqrt(2) , 1/4 sqrt(2) , - 1/4 sqrt(2)]

> # Let the initial target state be location five.
> # Define the target state matrix.
> I_t3 := matrix(8,8,
[[1,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0],[0,0,1,0,0,0,0,0],
[0,0,0,1,0,0,0,0],[0,0,0,0,-
1,0,0,0],[0,0,0,0,0,1,0,0],[0,0,0,0,0,0,1,0],[0,0,0,0,0,0,0,1]]);

      [1      0      0      0      0      0      0      0]
      [
      [0      1      0      0      0      0      0      0]
      [
      [0      0      1      0      0      0      0      0]
      [
      [0      0      0      1      0      0      0      0]
I_t3 := [
      [0      0      0      0      -1     0      0      0]
      [
      [0      0      0      0      0      1      0      0]
      [
      [0      0      0      0      0      0      1      0]
      [
      [0      0      0      0      0      0      0      1]

> # Define the I sub zero matrix.
> I_o3a := matrix(8,8,[[
1,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0],[0,0,1,0,0,0,0,0],
[0,0,0,1,0,0,0,0],[0,0,0,0,1,0,0,0],[0,0,0,0,0,1,0,0],[0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,1]]);

      [-1     0      0      0      0      0      0      0]
      [
      [ 0      1      0      0      0      0      0      0]
      [
      [ 0      0      1      0      0      0      0      0]
      [
      [ 0      0      0      1      0      0      0      0]
I_o3a := [
      [ 0      0      0      0      1      0      0      0]
      [
      [ 0      0      0      0      0      1      0      0]
      [
      [ 0      0      0      0      0      0      1      0]
      [
      [ 0      0      0      0      0      0      0      1]

> I_o3 := matrix(8,8);

      I_o3 := array(1 .. 8, 1 .. 8, [])

> I_o3 := scalarmul(I_o3a, -1);

      [1      0      0      0      0      0      0      0]
      [

```

```

      [0      -1      0      0      0      0      0      0]
      [
      [0      0      -1      0      0      0      0      0]
      [
      [0      0      0      -1      0      0      0      0]
      [
      [0      0      0      0      -1      0      0      0]
      [
      [0      0      0      0      0      -1      0      0]
      [
      [0      0      0      0      0      0      -1      0]
      [
      [0      0      0      0      0      0      0      -1]
I_o3 :=

> # define Q;
> Q3 :=matrix(8,8, multiply(I_o3,W3,I_t3,W3));

      [3/4 ,  -1/4 ,  -1/4 ,  -1/4 ,  1/4 ,  1/4 ,  1/4 ,  1/4]
      [
      [1/4 ,  -3/4 ,  1/4 ,  1/4 ,  -1/4 ,  -1/4 ,  -1/4 ,  -1/4]
      [
      [1/4 ,  1/4 ,  -3/4 ,  1/4 ,  -1/4 ,  -1/4 ,  -1/4 ,  -1/4]
      [
      [1/4 ,  1/4 ,  1/4 ,  -3/4 ,  -1/4 ,  -1/4 ,  -1/4 ,  -1/4]
Q3 := [
      [-1/4 ,  -1/4 ,  -1/4 ,  -1/4 ,  -3/4 ,  1/4 ,  1/4 ,  1/4]
      [
      [-1/4 ,  -1/4 ,  -1/4 ,  -1/4 ,  1/4 ,  -3/4 ,  1/4 ,  1/4]
      [
      [-1/4 ,  -1/4 ,  -1/4 ,  -1/4 ,  1/4 ,  1/4 ,  -3/4 ,  1/4]
      [
      [-1/4 ,  -1/4 ,  -1/4 ,  -1/4 ,  1/4 ,  1/4 ,  1/4 ,  -3/4]

> # Apply 2 iterations for 3-qubits.
> three_qubits_itertion1 := matrix (8,1, multiply(Q3,Start_state3));

      [3/4 ]
      [
      [1/4 ]
      [
      [1/4 ]
      [
      [1/4 ]
three_qubits_itertion1 := [
      [-1/4]
      [
      [-1/4]
      [
      [-1/4]
      [
      [-1/4]

> three_qubits_result:= multiply(W3,Q3,three_qubits_itertion1);

      [- 1/16 sqrt(2)]

```

```

                                [
                                [- 1/16 sqrt(2)]
                                [
                                [- 1/16 sqrt(2)]
                                [
                                [- 1/16 sqrt(2)]
                                [
three_qubits_result := [ 11
                                [ -- sqrt(2)
                                [ 16
                                [
                                [- 1/16 sqrt(2)]
                                [
                                [- 1/16 sqrt(2)]
                                [
                                [- 1/16 sqrt(2)]

                                %1 := [- 1/16 sqrt(2)]

>
> # Get the sum of all the amplitudes.
> Sum_of_ampl3:= ((abs(-1/16 * sqrt(2)))^2)+ ((abs(-1/16 *
sqrt(2)))^2) + ((abs(-1/16 * sqrt(2)))^2) +((abs(-1/16 *
sqrt(2)))^2)+((abs(-1/16 * sqrt(2)))^2)+((abs(-1/16 *
sqrt(2)))^2)+((abs(-1/16 * sqrt(2)))^2)+((abs(11/16 * sqrt(2)))^2);

                                Sum_of_ampl3 := 1

> # target_length3 is the length of our target location -- location
five.

> target_length3 := ((abs(11/16 * sqrt(2)))^2);

                                121
target_length3 := ---
                                128

> # Calculate the probability of finding the target location.
> prob_of_finding_taget3 := eval(target_length3/Sum_of_ampl3);

                                121
prob_of_finding_taget3 := ---
                                128

> # Thus, we see that we have a 121/128 or .945 probability of finding
the target location after two iterations.
>
> # End of verification.
>

```

NOTE TO USERS

The diskette is not included in this original manuscript. It is available for consultation at the author's graduate school library.

UMI

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved