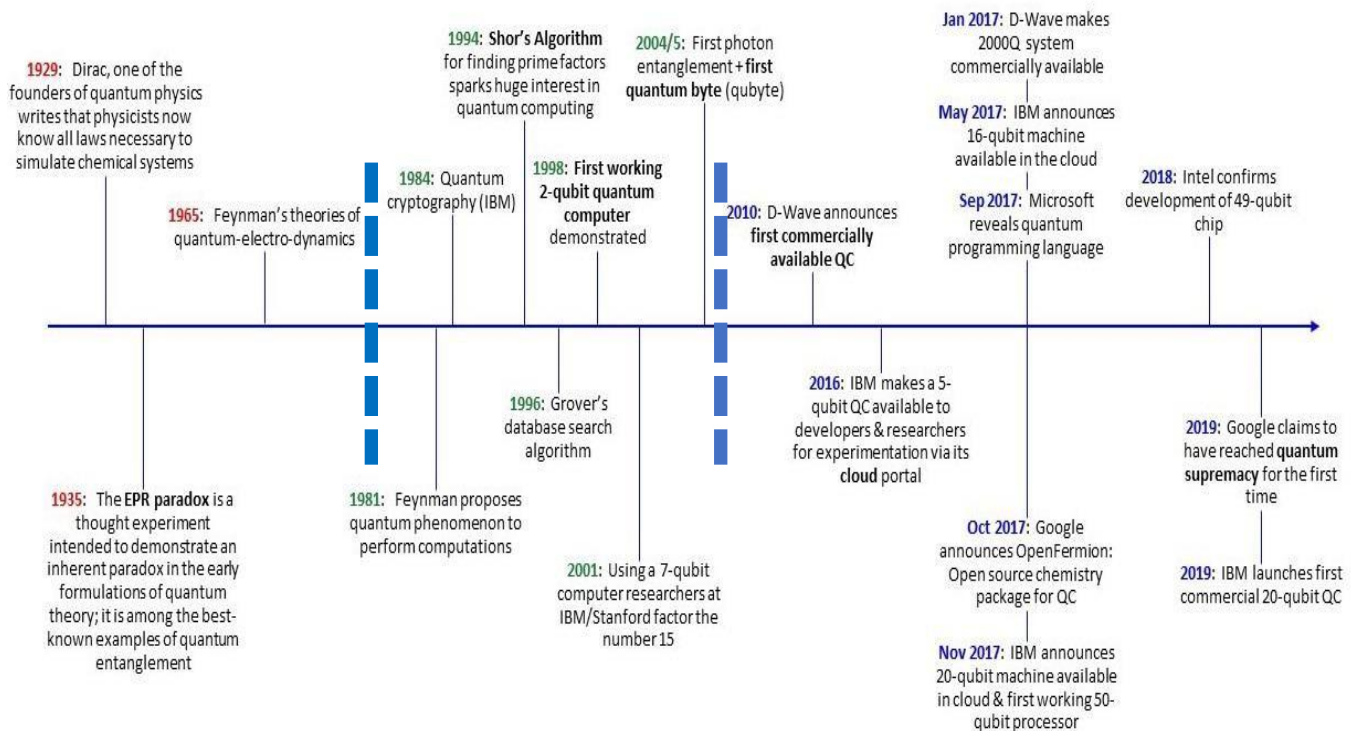# 1 INTRODUCTION

1.1 BRIEF ABOUT QUANTUM COMPUTING

The development of science and technology has led to significant advancements in civilization, with new methods emerging to exploit various physical resources like materials, forces, and energies. The history of computer development reflects years of technological progress, beginning with Charles Babbage's early concepts and culminating in the creation of the first computer by German engineer Konrad Zuse in 1941. This evolution involved a shift from gears to relays, valves, transistors, integrated circuits, chips, and beyond. Despite their compact size and increased speed, modern computers fundamentally operate the same way as their massive 30-ton predecessors, which utilized 18,000 vacuum tubes and 500 miles of wiring. Their core task remains to manipulate and interpret binary bits into meaningful computational results. Since 1950, the number of atoms required to store a single bit of memory has been decreasing exponentially. Gordon Moore's 1965 observation, known as Moore's Law, predicted that computer processing power would double every 18 months. If this trend continues, it suggests that bits of information will eventually be encoded by subatomic systems. A 1988 survey by Keyes supports this idea, showing a decreasing number of electrons needed to store a bit, pointing to the feasibility of atomic-scale computation in the near future.

The fundamental basis of quantum computation is Landauer's observation that all information is ultimately physical [1, 2]. Information, the 1's and 0's of classical computers, must inevitably be recorded by some physical system - be it paper or silicon. Which brings us to the key point. As far as we know today, all matter is composed of atoms - nuclei and electrons - and the interactions and time evolution of atoms are governed by the laws of quantum mechanics. Although the peculiarities of the quantum world may not seem readily apparent at first glance, a closer look reveals that applications of quantum mechanics are all around. As has been emphasized by Minsky, the very existence of atoms owes everything not to the chaotic uncertainties of classical mechanics, but rather to the certainties of quantum mechanics with the Pauli exclusion principle and well-defined and stable atomic energy levels! Indeed without our quantum understanding of the solid state and the band theory of metals, insulators and semiconductors, the whole of the semiconductor industry with its transistors and integrated circuits - and hence the computer on which I am writing this lecture - could not have developed.

1

1.2 Evolution of Quantum Theory & Quantum Technology

1929: Dirac, one of the founders of quantum physics writes that physicists now know all laws necessary to simulate chemical systems

1965: Feynman's theories of quantum-electro-dynamics

1935: The **EPR paradox** is a thought experiment intended to demonstrate an inherent paradox in the early formulations of quantum theory; it is among the best-known examples of quantum entanglement

1981: Feynman proposes quantum phenomenon to perform computations

1984: Quantum cryptography (IBM)

1994: **Shor's Algorithm** for finding prime factors sparks huge interest in quantum computing

1996: Grover's database search algorithm

1998: **First working 2-qubit quantum computer** demonstrated

2001: Using a 7-qubit computer researchers at IBM/Stanford factor the number 15

2004/5: First photon entanglement + **first quantum byte (qubyte)**

2010: D-Wave announces **first commercially available QC**

2016: IBM makes a 5-qubit QC available to developers & researchers for experimentation via its **cloud** portal

Jan 2017: D-Wave makes 2000Q system commercially available

May 2017: IBM announces 16-qubit machine available in the cloud

Sep 2017: Microsoft reveals quantum programming language

Oct 2017: Google announces OpenFermion: Open source chemistry package for QC

Nov 2017: IBM announces 20-qubit machine available in cloud & first working 50-qubit processor

2018: Intel confirms development of 49-qubit chip

2019: Google claims to have reached **quantum supremacy** for the first time

2019: IBM launches first commercial 20-qubit QC

1.3 QUANTUM V/S CLASSICAL

Quantum Computer

A quantum computer is a type of computer that leverages the principles of quantum mechanics to process information in a fundamentally different way from classical computers. Unlike classical computers, which use bits to represent data as either 0 or 1, quantum computers use quantum bits, or qubits, which can exist in a superposition of states. This means a qubit can represent both 0 and 1 simultaneously, enabling the computer to perform multiple calculations in parallel. Key phenomena in quantum mechanics that power quantum computers include:

- **Superposition**: Allows qubits to exist in multiple states at once, increasing computational capacity exponentially.

- **Entanglement**: Enables qubits to be correlated in such a way that the state of one qubit is directly related to the state of another, regardless of the distance between them.

- **Decoherence**: Refers to the loss of quantum states due to interaction with the environment, posing challenges in maintaining stable quantum computations.

Classical Computer

DEPARTMENT OF CSE, RYMEC, BALLARI

A classical computer operates using the principles of classical mechanics, relying on electronic circuits and gates to process information. Data in classical computers is represented as binary bits (0 or 1), and computations are performed by manipulating these bits through a series of logic gates. The main features of classical computers include:

- **Voltage-Based Logic**: Information is represented and processed through varying voltage levels, corresponding to binary states (on/off or 0/1).

- **Sequential Processing**: Tasks are executed in a linear, step-by-step manner, although modern classical computers employ parallelism through multiple processors or cores to enhance performance.

- **Deterministic Nature**: Every operation in a classical computer follows predictable and reproducible rules of classical mechanics.

Quantum Mechanics

• It deals with microscopic particles

• It is based on the Schrödinger equation

• In Planck's postulation, only discrete values of energy are emitted or absorbed – origin of "quantum"

• Because of Heisenberg's uncertainty principle and de Broglie hypothesis dual nature of matter (both particle & wave), the state of a system cannot be specified exactly

• It gives probabilities of finding particles at various locations in space

Classical Mechanics

• It deals with macroscopic particles

• It is based on Newton's laws of motion and Maxwell's electromagnetic wave theory

• Any amount of energy may be emitted or absorbed continuously

• The state of a system is defined exactly by specifying their positions and velocities

• The future state can be predicted with certainty

1.4 QUANTUM QBITS, GATES & CIRCUITS

Quantum vs. Classic Gates

Operator	Gate(s)		Matrix
Pauli-X (X)	—X—	—⊕—	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)	—Y—		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)	—Z—		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)	—H—		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)	—S—		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)	—T—		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)			$\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix}$
Controlled Z (CZ)			$\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&-1 \end{bmatrix}$
SWAP			$\begin{bmatrix} 1&0&0&0 \\ 0&0&1&0 \\ 0&1&0&0 \\ 0&0&0&1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)			$\begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0 \end{bmatrix}$

NOT gate — x — ▷o— y = NOT(x)

x	y
0	1
1	0

AND gate — x, y — z = (x) AND (y)

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

NAND gate — x, y — z = (x) NAND (y)

x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

OR gate — x, y — z = (x) OR (y)

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

NOR gate — x, y — z = (x) NOR (y)

x	y	z
0	0	1
0	1	0
1	0	0
1	1	0

XOR gate — x, y — z = (x) XOR (y)

x	y	z
0	0	0
0	1	1
1	0	1

Quantum Circuit

- A quantum circuit is a model for quantum computation in which a computation is a sequence of quantum gates with *n*-qubit register linked by "wires"

- The circuit has fixed "width" corresponding to the number of qubits being processed

Unlike classical circuits, the same number of wires is going throughout the circuit

DEPARTMENT OF CSE, RYMEC, BALLARI
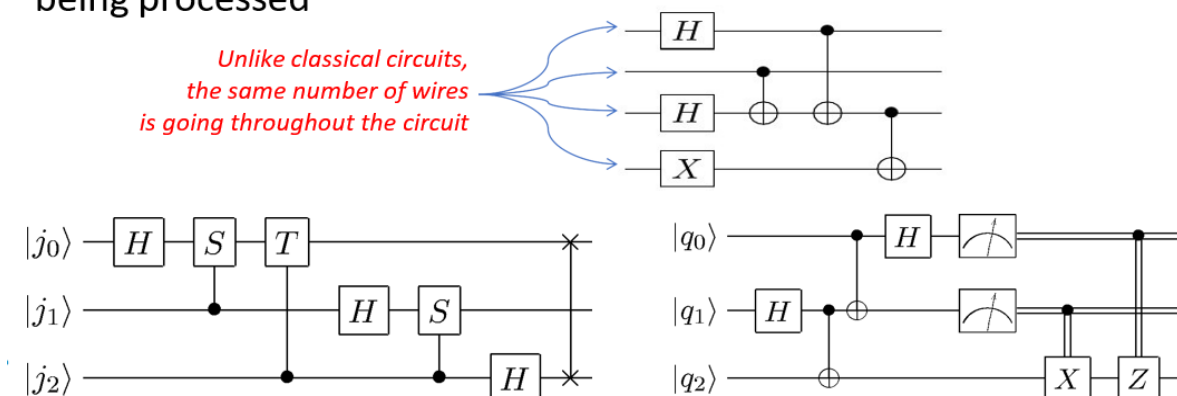
Single Qubit Gate

$$|0\rangle - \boxed{U} - \text{Any state} |\psi\rangle$$

- ## Pauli-X gate

$$|0\rangle \rightarrow |1\rangle, \quad |1\rangle \rightarrow |0\rangle$$

Dirac notation

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Matrix representation

$$-\boxed{X}-$$

Circuit representation

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \Longrightarrow \quad X\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \cdot 1 + 1 \cdot 0 \\ 1 \cdot 1 + 0 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \Longrightarrow \quad X\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \cdot 0 + 1 \cdot 1 \\ 1 \cdot 0 + 0 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

- Acting on pure states becomes a **classical *NOT*** gate

$$|0\rangle - \boxed{X} - |1\rangle$$
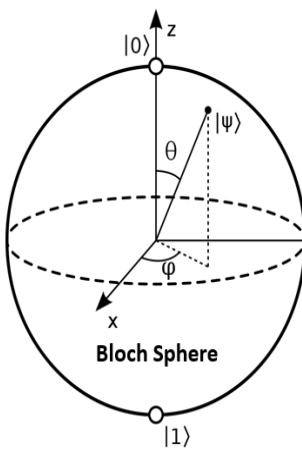
Single Qubit Gates

Dirac notation   *Matrix representation*   *Circuit representation*

Pauli Y − Gate $|0\rangle \rightarrow i|1\rangle, \quad |1\rangle \rightarrow -i|0\rangle$ $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ $-\boxed{Y}-$

…another gate with no classical equivalence

Pauli Z − Gates: $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ $-\boxed{Z}-$

$$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

Bloch Sphere

$$-\boxed{S}- \qquad -\boxed{\pi/8}-$$

Phase   **π/8 (T) gate**

$$-\boxed{S}-\boxed{S}-$$

||

$$-\boxed{Z}-$$

$$P^2 = Z$$

32

DEPARTMENT OF CSE, RYMEC, BALLARI

1.5 ALGORITHM'S

Algorithms Used in the Project

1. Sudoku Validation Algorithm

- Purpose: Ensures that each move made by the player adheres to Sudoku rules.

- Steps:

    1. Row Validation: Checks if the number already exists in the selected row.

    2. Column Validation: Checks if the number already exists in the selected column.

    3. Sub-grid Validation: Verifies if the number exists in the corresponding 2x2 sub-grid.

- Time Complexity: $O(n)O(n)$, where $n=4n = 4$ (for a 4x4 Sudoku grid).

2. Random Prefill Algorithm

- Purpose: Randomly prefill cells on the Sudoku board while maintaining the validity of the board.

- Steps:

    1. Iterate through the board's cells.

    2. Use a random number generator to decide whether to fill a cell.

    3. If chosen, generate a random number between 1 and 4.

    4. Validate the number using the Sudoku validation algorithm before placing it.

- Time Complexity: $O(n2)O(n^2)$, where nn is the size of the grid (4 in this case).

3. Quantum Circuit Representation Algorithm

- Purpose: Encodes the player's moves into a quantum circuit for visualization and potential quantum computation.

- Steps:

    1. Each cell in the Sudoku board is mapped to a corresponding quantum qubit.

    2. When the player places a number on the board, a quantum operation (X gate) is applied to the corresponding qubit.

    3. Quantum circuits are drawn and saved as images for analysis or reporting.

- Time Complexity: Depends on the number of moves ($O(m)$, where m is the number of valid moves made by the player).

4. Undo Mechanism Algorithm

- Purpose: Allows the player to undo their last move within a set limit.
- Steps:
    1. Record the player's move (row, column, and number) before placing it.
    2. If the player chooses to undo:
        - Reset the corresponding cell to 0.
        - Remove the corresponding quantum operation from the quantum circuit.
    3. Decrease the undo limit by 1.
- Time Complexity: $O(1)$ for undoing a move.

5. Box Completion Check Algorithm

- Purpose: Identifies when a 2x2 sub-grid (box) is completed correctly.
- Steps:
    1. Extract all numbers in the current 2x2 sub-grid.
    2. Sort the numbers and compare them to the expected sequence (1 to 4).
    3. If they match, declare the box as completed.
- Time Complexity: $O(k)$, where k is the size of the box (4 for 2x2 grids).

6. Timer Algorithm

- Purpose: Measures the time taken by the player to complete the game.
- Steps:
    1. Record the start time using the time.time() function.
    2. Record the end time when the game is completed.
    3. Compute the elapsed time by subtracting the start time from the end time.
- Time Complexity: $O(1)$.

7

Famous Quantum Algorithms

Algorithms	Classical steps	quantum logic steps
Fourier transform e.g.: - Shor's prime factorization - discrete logarithm problem - Deutsch Jozsa algorithm	$N \log(N) = n\, 2^n$ $N = 2^n$ - n qubits - N numbers	$\log^2(N) = n^2$ - hidden information! - Wave function collapse prevents us from directly accessing the information
Search Algorithms	N	$\sqrt{N}$
Quantum Simulation	c^N bits	kn qubits

1.6 ADVANTAGES & DISADVANTAGES

Advantages of Quantum Computing:

1. **Speed**: Solves complex problems faster than classical computers (e.g., factoring large numbers).

2. **Parallelism**: Processes multiple calculations simultaneously using qubits.

3. **Optimization**: Improves efficiency in optimization problems, like logistics or machine learning.

4. **Cryptography**: Breaks traditional encryption but also enables quantum-safe encryption.

5. **Simulations**: Accurately models quantum systems in chemistry and material science.

Disadvantages of Quantum Computing:

1. **Cost**: Expensive to build and maintain due to cooling and isolation requirements.

2. **Error Rates**: Susceptible to noise and decoherence, leading to errors.

3. **Complexity**: Difficult to program and understand due to quantum principles.

4. **Scalability**: Challenging to increase qubit count while maintaining stability.

5. **Security Risks**: Threatens existing cryptographic systems if quantum supremacy is achieved.

2 PROJECT

2.1 ABOUT PROJECT

This project explores the application of quantum computing in solving Sudoku puzzles and other simple games, leveraging quantum algorithms, qubits, and quantum circuits. Games such as Sudoku, Quantum Maze Solver, Quantum Chess, Quantum Battleship, and Quantum Rock-Paper-Scissors can benefit from the unique capabilities of quantum computing. These problems often involve complex constraints and optimization, making them computationally intensive for classical methods as complexity increases.

Quantum computing, with its ability to process vast combinations of states simultaneously through quantum parallelism, offers a novel and efficient approach to solving such problems. By encoding game constraints into quantum circuits, this project demonstrates how quantum algorithms can provide solutions faster than classical algorithms in certain scenarios, especially for larger or more intricate puzzles and games.

Key aspects of the project include:

1. **Quantum Algorithms**:
   - Implementation of algorithms such as Grover's algorithm for searching solutions or quantum annealing techniques to optimize constraint satisfaction.
   - Applying these algorithms to diverse games like Quantum Chess for strategic decision-making or Quantum Rock-Paper-Scissors for probabilistic outcome prediction.

2. **Qubits and Superposition**:
   - Utilizing the unique properties of qubits (superposition and entanglement) to represent and process multiple game states concurrently.
   - In games like Quantum Memory Game, qubits enable matching quantum states with unprecedented efficiency.

3. **Quantum Circuits**:
   - Designing quantum gates and circuits to enforce game constraints, such as row, column, and block uniqueness in Sudoku or grid placement rules in Quantum Battleship.
   - Customizing circuits for probabilistic games like Quantum Dice Roller or Quantum Coin Flip, where quantum randomness plays a pivotal role.

2.2 INTRODUCTION OF PROJECT

Quantum computing represents a revolutionary approach to solving computational problems by harnessing the principles of quantum mechanics, such as superposition, entanglement, and quantum parallelism. This project integrates these advanced concepts to address the challenges of solving Sudoku puzzles and other simple games by leveraging quantum algorithms, qubits, and quantum circuits.

Sudoku, a widely recognized logic-based game that involves filling a 9×9 grid with numbers while satisfying specific constraints, is classified as an NP-complete problem. Its computational complexity increases exponentially with its size and difficulty, posing significant challenges for classical computing methods that rely on sequential or heuristic-based techniques. Quantum computing offers a fundamentally different paradigm, enabling the processing of multiple possibilities simultaneously. By utilizing qubits, quantum gates, and quantum circuits, this project explores how quantum algorithms can efficiently solve Sudoku puzzles.

2.3 PROBLEM STATEMENT

Sudoku

Sudoku is a widely popular logic-based puzzle, but solving it computationally becomes increasingly challenging as puzzle complexity grows. Classical computing methods for solving Sudoku, such as brute force or heuristic algorithms, can be computationally expensive and time-consuming, especially for larger grids or puzzles with multiple solutions. The problem lies in efficiently solving Sudoku puzzles while adhering to the constraints:

1. Each row, column, and 3×3 sub-grid must contain unique numbers from 1 to 9.

2. The solution must satisfy all constraints simultaneously without conflicts.

This project addresses the challenge by leveraging quantum computing, which uses the principles of superposition, entanglement, and parallelism to process and explore multiple puzzle states simultaneously. The goal is to design and implement quantum algorithms using qubits and quantum circuits to solve Sudoku puzzles in a way that outperforms or complements classical approaches. Through this, the project seeks to demonstrate the potential of quantum computing in solving combinatorial and constraint satisfaction problems effectively.

Quantum Tic-Tac-Toe

Tic-Tac-Toe, a simple and widely played game, involves two players placing X and O marks on a 3×3 grid. While it is computationally simple for classical computers, optimizing strategies or evaluating all possible outcomes in a more advanced or quantum version can become complex, especially if enhanced with probabilistic or quantum strategies. The problem lies in evaluating different game states and predicting outcomes while considering multiple possible

strategies and moves. Classical algorithms focus on examining all possible game states, but quantum computing could offer advantages in exploring these states simultaneously, leveraging superposition and quantum parallelism.

2.4 PROBLEM DESCRIPTION

This project explores the use of quantum computing to solve a variety of simple games, including Sudoku, Tic-Tac-Toe, Quantum Maze Solver, Battleship, Chess, Rock-Paper-Scissors, Hangman, Memory Game, and Dice Roller. These games often involve combinatorial problems, optimization, or randomness, which are computationally challenging for classical algorithms, especially as the complexity increases.

For Sudoku, the challenge is to solve the puzzle by ensuring that each row, column, and sub-grid contains unique numbers, a problem that grows exponentially with size and difficulty. Classical methods like brute force or backtracking are often inefficient. Quantum computing offers a solution by using qubits to represent multiple possible solutions simultaneously, enhancing efficiency through superposition and quantum parallelism.

Similarly, for games like Tic-Tac-Toe, Battleship, and Chess, quantum algorithms can evaluate multiple possible game states at once, optimizing strategy and decision-making. Quantum randomness in games like Coin Flip, Rock-Paper-Scissors, and Dice Roller offers true randomness, unlike classical random number generators.

This project aims to demonstrate how quantum algorithms, leveraging quantum properties like superposition, entanglement, and parallelism, can improve the performance and efficiency of these games compared to classical approaches.

DEPARTMENT OF CSE, RYMEC, BALLARI

2.5 IMPLEMENTATION

Implementation of Quantum Hardware

- Liquid-state NMR
- NMR spin lattices
- Linear ion-trap spectroscopy
- Neutral-atom optical lattices
- Cavity QED + atom
- Linear optics
- Nitrogen vacancies in diamond
- Electrons in liquid He
- Superconducting Josephson junctions
  - charge qubits; flux qubits; phase qubits
- Quantum Hall qubits
- Coupled quantum dots
  - spin, charge, excitons
- Spin spectroscopies, impurities in semiconductors
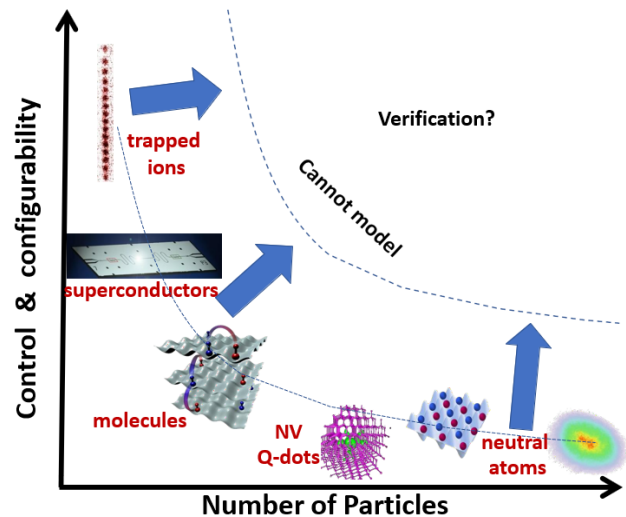
2.6 ADVANTAGE'S

Advantages of Using Quantum Theory and Algorithms Instead of Classical Computers

1. **Parallelism Through Superposition**:

   Quantum computing leverages the principle of superposition, allowing qubits to represent multiple states simultaneously. This enables quantum algorithms to process many potential solutions at once, significantly speeding up computations compared to the sequential nature of classical algorithms.

2. **Efficient Problem Solving**:

   Quantum algorithms, such as Grover's search algorithm, provide quadratic speed-ups for search problems, while others like Shor's algorithm achieve exponential speed-ups for certain tasks like factorization. These efficiencies make quantum computing particularly advantageous for NP-complete and optimization problems, like solving Sudoku.

3. **Reduction in Computational Complexity**:

Classical methods for constraint satisfaction problems, such as Sudoku solving, often involve exponential time complexity as the problem size increases. Quantum algorithms can reduce the complexity by exploring large solution spaces more effectively.

4. **Enhanced Optimization Capabilities**:

Quantum systems are naturally suited for optimization problems. Techniques like quantum annealing and Variational Quantum Eigensolvers (VQE) allow quantum computers to find optimal or near-optimal solutions to problems that are computationally expensive for classical methods.

5. **Scalability for Complex Problems**:

As problem complexity increases, classical computing struggles with memory and processing power requirements. Quantum computers, by encoding and processing vast amounts of information in qubits, are better suited for tackling large-scale problems.

6. **Potential for Breakthrough Applications**:

Quantum computing opens the door to solving problems previously thought to be infeasible, such as protein folding, financial modeling, and advanced cryptography. In the context of your project, it demonstrates how quantum techniques can revolutionize approaches to combinatorial puzzles.

7. **Exploration of Quantum Mechanics in Practical Applications**:

Developing a Sudoku solver using quantum algorithms showcases how theoretical concepts of quantum mechanics (superposition, entanglement, interference) can be applied to real-world problems, paving the way for future innovations.

8. **Simultaneous Constraint Satisfaction**:

Quantum circuits can enforce multiple constraints (like Sudoku's row, column, and block rules) in a single computation step, making them highly effective for problems with complex interdependent rules.

DEPARTMENT OF CSE, RYMEC, BALLARI

2.7 PROBLEM'S QUANTUM PROJECT

Problems with Quantum Computers

- Decoherence
  - Quantum system is extremely sensitive to external environment, so it should be safely isolated
  - It is hard to achieve the decoherence time that is more than the algorithm running time
- Error correction (requires more qubits!)
- Physical implementation of computations
- New quantum algorithms to solve more problems
- Entangled states for data transfer
- Running time increases exponentially with matrix size

3 REQUIREMENTS

3.1 SOFTWARE REQUIREMENTS

1. Operating System:

   o Windows 10/11, macOS 10.13 or later, or Linux-based distributions (e.g., Ubuntu 18.04 or later).

2. Python Interpreter:

   o Python 3.7.1 (or compatible version).

3. Libraries and Frameworks:

   o Qiskit (latest version compatible with Python 3.7.1).

   o NumPy (latest stable version).

   o Matplotlib (latest stable version).

4. Development Tools:

- o IDE/Text Editor: VSCode, PyCharm, or Jupyter Notebook for coding and testing.
- o Package Manager: pip for installing required Python libraries.

5. Quantum Computing Backend (Optional):

- o IBM Quantum Experience account for accessing real quantum computers (if needed).

6. Dependencies:

- o Python libraries: Ensure qiskit, numpy, and matplotlib are installed and functional using pip.

7. Other Software:

- o Image Viewer: To view saved quantum circuit diagrams.
- o PDF or Word Processor: For creating project documentation.

3.2 HARDWARE REQUIREMENTS

1. Processor:

- o A modern multi-core processor (e.g., Intel i5/i7 or equivalent) is recommended for handling computation-heavy tasks in Python and quantum simulations.
- o Quantum computations for games can also be run on cloud-based quantum systems (e.g., D-Wave or IBM Quantum), making cloud processing power crucial.

2. RAM:

- o Minimum 8 GB of RAM is recommended for smooth execution of Python programs, handling datasets, and simulating quantum algorithms for games like Sudoku, Quantum Maze Solver, or Quantum Chess.
- o For more complex games involving larger datasets or advanced quantum circuits, 16 GB or more is ideal.

3. Graphics Card (GPU):

- o While not strictly necessary for quantum algorithms themselves, a good GPU (e.g., NVIDIA with CUDA support) is beneficial for running simulations, rendering visualizations, or implementing machine learning components for games like Quantum Memory Game or Quantum Battleship.

4. Storage:

- o Minimum 20 GB of free storage is required for installing libraries, Python environments, and saving game-related project files.

- o Larger game data, visualization results, or simulation outputs may require additional storage space.

5. Quantum Hardware (Optional):

- o D-Wave Quantum Computer: For running quantum algorithms on actual hardware, access to a quantum annealing system like the D-Wave quantum computer is needed. This can be accessed via D-Wave's cloud platform (Leap).

- o IBM Quantum System: For running gate-based quantum algorithms, access to IBM Quantum's cloud platform is necessary.

3.4 QUANTUM PROGRAMMING LANGUAGE

Quantum Programming

- There is already a number of programming languages adapted for quantum computing
- The purpose of quantum programming languages is to provide a tool for researchers, not a tool for programmers
- QCL is an example of such language
- IBM QISKit (Quantum Information Science Kit) is another example

Name	Tagline	Programming language	Licence	Supported OS
Cirq	Framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits.	Python	Apache-2.0	Windows, Mac, Linux
Cliffords.jl	Efficient calculation of Clifford circuits in Julia.	Julia	MIT	Windows, Mac, Linux
dimod	Shared API for Ising/quadratic unconstrained binary optimization samplers.	Python	Apache-2.0	Windows, Linux, Mac
dwave-system	Basic API for easily incorporating the D-Wave system as a sampler in the D-Wave Ocean software stack.	Python	Apache-2.0	Linux, Mac
FermiLib	Open source software for analyzing fermionic quantum simulation algorithms.	Python	Apache-2.0	Windows, Mac, Linux
Forest (pyQuil & Grove)	Simple yet powerful toolkit for writing hybrid quantum-classical programs.	Python	Apache-2.0	Windows, Mac, Linux
OpenFermion	The electronic structure package for quantum computers.	Python	Apache-2.0	Windows, Mac, Linux
ProjectQ	An open source software framework for quantum computing.	Python, C++	Apache-2.0	Windows, Mac, Linux
PyZX	Python library for quantum circuit rewriting and optimisation using the ZX-calculus.	Python	GPL-3.0	Windows, Mac, Linux
QGL.jl	A performance orientated QGL compiler.	Julia	Apache-2.0	Windows, Mac, Linux
Qbsolv	Decomposing solver that finds a minimum value of a large quadratic unconstrained binary optimization problem by splitting it into pieces.	C	Apache-2.0	Windows, Linux, Mac
Qiskit Terra & Aqua	Quantum Information Science Kit for writing experiments, programs, and applications.	Python, C++	Apache-2.0	Windows, Mac, Linux
Qiskit Tutorials	A collection of Jupyter notebooks using Qiskit.	Python	Apache-2.0	Windows, Mac, Linux
Qiskit.js	Quantum Information Science Kit for JavaScript.	JavaScript	Apache-2.0	Windows, Mac, Linux
Qrack	Comprehensive, GPU accelerated framework for developing universal virtual quantum processors.	C++	GPL-3.0	Linux, Mac
Quantum Fog	Python tools for analyzing both classical and quantum Bayesian networks.	Python	BSD-3-Clause	Windows, Mac, Linux
Quantum++	A modern C++11 quantum computing library.	C++, Python	MIT	Windows, Mac, Linux
Qubiter	Python tools for reading, writing, compiling, simulating quantum computer circuits.	Python, C++	BSD-3-Clause	Windows, Mac, Linux
Quirk	Drag-and-drop quantum circuit simulator for your browser to explore and understand small quantum circuits.	JavaScript	Apache-2.0	Windows, Mac, Linux
reference-qvm	A reference implementation for a Quantum Virtual Machine in Python.	Python	Apache-2.0	Windows, Mac, Linux
ScaffCC	Compilation, analysis and optimization framework for the Scaffold quantum programming language.	C++, Objective C, LLVM	BSD-2-Clause	Linux, Mac
Strawberry Fields	Full-stack library for designing, simulating, and optimizing continuous variable quantum optical circuits.	Python	Apache-2.0	Windows, Mac, Linux
XACC	eXtreme-scale Accelerator programming framework.	C++	Eclipse PL.-1.0	Windows, Mac, Linux
XACC VQE	Variational quantum eigensolver built on XACC for distributed, and shared memory systems.	C++	BSD-3-Clause	Windows, Mac, Linux

DEPARTMENT OF CSE, RYMEC, BALLARI

Quantum Programming

- **QCL** (**Q**uantum **C**omputation **L**anguage)

```
/* Remove "//" if starting interpreter with -n option */
// extern operator H(qureg q);

                                              C-like syntax

procedure FlipCoin() {
  qureg q[1]; int x;
                                    allows combining of
                                      quantum and
  reset;                              classical code
  H(q);
  measure q, x;
  if x == 1  { print "Heads"; }
  if x == 0  { print "Tails"; }
  reset;
}
```

http://tph.tuwien.ac.at/~oemer/qcl.html

Quantum Programming

- **QISKit** (**Q**uantum **I**nformation **S**cience **Kit**)

Qiskit is an open-source framework for quantum computing. It provides tools for creating and manipulating quantum programs and running them on prototype quantum devices on IBM Quantum Experience over Cloud-based access
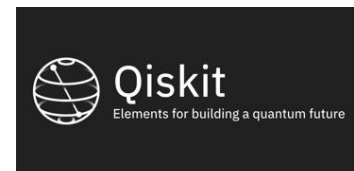

```
In [7]: from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
        from qiskit.tools.visualization import circuit_drawer
        import numpy as np

        qr = QuantumRegister(2)
        cr = ClassicalRegister(2)
        qp = QuantumCircuit(qr,cr)

        qp.rx( np.pi/2,qr[0])
        qp.cx(qr[0],qr[1])

        qp.measure(qr,cr)

        circuit_drawer(qp)
```

```
In [1]: from qiskit import *

In [2]: qr = QuantumRegister(2)
        cr = ClassicalRegister(2)

In [3]: c = QuantumCircuit(qr, cr)    # c = QuantumCircuit(2,2)
```
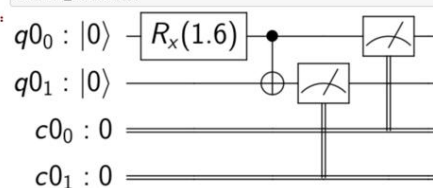
4 SOURCE CODE

```python
from qiskit import QuantumCircuit, execute, Aer

import numpy as np

import time

def print_board(board):

print("\nCurrent Sudoku Board:")

print("+-------+-------+-------+")

for i in range(9):

row = " | ".join(" ".join(str(x) if x != 0 else "." for x in board[i][j:j+3]) for j in range(0, 9, 3))

print(f"| {row} |")

if (i + 1) % 3 == 0:

        print("+-------+-------+-------+")

def is_valid_move(board, row, col, num):

if num in board[row]:

return False

if num in [board[x][col] for x in range(9)]:

return False

start_row, start_col = 3 * (row // 3), 3 * (col // 3)

for i in range(3):

for j in range(3):

        if board[start_row + i][start_col + j] == num:

        return False

return True

def get_move():

while True:

try:

        row = int(input("Enter the row number (1-9): ")) - 1

        col = int(input("Enter the column number (1-9): ")) - 1

        num = int(input("Enter the number to place (1-9): "))

        if 0 <= row < 9 and 0 <= col < 9 and 1 <= num <= 9:

        return row, col, num
```

DEPARTMENT OF CSE, RYMEC, BALLARI

```python
        else:
            print("Invalid input. Please enter values in the correct range.")
    except ValueError:
            print("Invalid input. Please enter numerical values.")
player_name = input("Enter your name: ")
scoreboard = {player_name: 0}
undo_limit = 3
final_circuit = QuantumCircuit(81, 81)
while True:
board = np.zeros((9, 9), dtype=int)
circuit = QuantumCircuit(81, 81)
for i in range(9):
for j in range(9):
        if np.random.rand() < 0.2:
        num = np.random.randint(1, 10)
        if is_valid_move(board, i, j, num):
                    board[i][j] = num
print_board(board)
winner = None
turn = 0
start_time = time.time()
while turn < 81:
print(f"{player_name}'s turn. Undo remaining: {undo_limit}.")
while True:
        row, col, num = get_move()
        if board[row][col] == 0 and is_valid_move(board, row, col, num):
        board[row][col] = num
        circuit.x(row * 9 + col)  # Represent move in quantum circuit
        break
        else:
        print("Invalid move. Try again.")
print_board(board)
if undo_limit > 0:
```

DEPARTMENT OF CSE, RYMEC, BALLARI

```python
            undo_choice = input("Do you want to undo your move? (yes/no): ").strip().lower()
            if undo_choice == 'yes':
            board[row][col] = 0  # Undo the move
            undo_limit -= 1
            print("Move undone.")
            continue
    if np.all(board):
            end_time = time.time()
            elapsed_time = end_time - start_time
            print(f"Congratulations {player_name}! You completed the Sudoku in
            {elapsed_time:.2f} seconds.")
            scoreboard[player_name] += 1
            winner = player_name
            break
    turn += 1
    if not winner:
    print("No winner this time!")
    final_circuit.compose(circuit, inplace=True)
    print("\nCurrent Scoreboard:")
    for player, score in scoreboard.items():
    print(f"{player}: {score}")
    choice = input("Do you want to play another game? (yes/no): ").strip().lower()
    if choice != 'yes':
    # Save final combined circuit diagram
    final_filename = "final_image.png"
    final_circuit.measure(range(81), range(81))
    final_circuit.draw(output='mpl').savefig(final_filename)
    print(f"Final combined circuit diagram saved as {final_filename}.")
    print("Thank you for playing! Final Scores:")
    for player, score in scoreboard.items():
            print(f"{player}: {score}")
    break
```

5 SNAPSHOT'S

```
PS C:\Users\srinu\Desktop\tic tac toe\sudoku> python s4
Enter your name: srinu

Current Sudoku Board:
+-------+-------+-------+
| . 2 . | . . . | . 3 . |
| . 5 . | . 9 . | . . . |
| . . . | . . 8 | . . . |
+-------+-------+-------+
| . . 8 | . . . | . . 5 |
| . . . | . . . | 6 2 . |
| . . 2 | 4 1 . | . . . |
+-------+-------+-------+
| 9 . . | . . . | . . . |
| . 8 . | . . . | . . . |
| . . . | . . 5 | . . . |
+-------+-------+-------+
srinu's turn. Undo remaining: 3.
Enter the row number (1-9): 1
Enter the column number (1-9): 1
Enter the number to place (1-9): 1
```

```
Current Sudoku Board:
+-------+-------+-------+
| 1 2 . | . . . | . 3 . |
| . 5 . | . 9 . | . . . |
| . . . | . . 8 | . . . |
+-------+-------+-------+
| . . 8 | . . . | . . 5 |
| . . . | . . . | 6 2 . |
| . . 2 | 4 1 . | . . . |
+-------+-------+-------+
| 9 . . | . . . | . . . |
| . 8 . | . . . | . . . |
| . . . | . . 5 | . . . |
+-------+-------+-------+
Do you want to undo your move? (yes/no): no
srinu's turn. Undo remaining: 3.
Enter the row number (1-9): |
```

DEPARTMENT OF CSE, RYMEC, BALLARI

```
Current Sudoku Board:
+-------+-------+-------+
| 1 2 4 | .  .  . | .  3  . |
| 8 5 3 | .  9  . | .  .  . |
| 6 9 . | .  .  8 | .  .  . |
+-------+-------+-------+
| .  .  8 | .  .  . | .  .  5 |
| .  .  . | .  .  . | 6  2  . |
| .  .  2 | 4  1  . | .  .  . |
+-------+-------+-------+
| 9  .  . | .  .  . | .  .  . |
| .  8  . | .  .  . | .  .  . |
| .  .  . | .  .  5 | .  .  . |
+-------+-------+-------+
Do you want to undo your move? (yes/no): no
srinu's turn. Undo remaining: 3.
Enter the row number (1-9): 3
Enter the column number (1-9): 3
Enter the number to place (1-9): 7
```

```
Current Sudoku Board:
+-------+-------+-------+
| 1 2 4 | .  .  . | .  3  . |
| 8 5 3 | .  9  . | .  .  . |
| 6 9 7 | .  .  8 | .  .  . |
+-------+-------+-------+
| .  .  8 | .  .  . | .  .  5 |
| .  .  . | .  .  . | 6  2  . |
| .  .  2 | 4  1  . | .  .  . |
+-------+-------+-------+
| 9  .  . | .  .  . | .  .  . |
| .  8  . | .  .  . | .  .  . |
| .  .  . | .  .  5 | .  .  . |
+-------+-------+-------+
Do you want to undo your move? (yes/no): no
Congratulations! One box is completed!
Circuit diagram saved as sudoku_circuit_box_0_0.png.
srinu's turn. Undo remaining: 3.
Enter the row number (1-9):
```

DEPARTMENT OF CSE, RYMEC, BALLARI

```
+-----+-----+
| 4 3 | . . |
| 1 . | 3 . |
+-----+-----+
| 3 . | . 4 |
| . . | . . |
+-----+-----+
Do you want to undo your move? (yes/no): no
srinu's turn. Undo remaining: 3.
Enter the row number (1-4): 2
Enter the column number (1-4): 2
Enter the number to place (1-4): 2

Current Sudoku Board:
+-----+-----+
| 4 3 | . . |
| 1 2 | 3 . |
+-----+-----+
| 3 . | . 4 |
| . . | . . |
+-----+-----+
Do you want to undo your move? (yes/no): no
Congratulations! One box is completed!
Circuit diagram saved as sudoku_circuit_box_0_0.png.
srinu's turn. Undo remaining: 3.
Enter the row number (1-4): |
```


Fig:- Citcuit Diagram

DEPARTMENT OF CSE, RYMEC, BALLARI
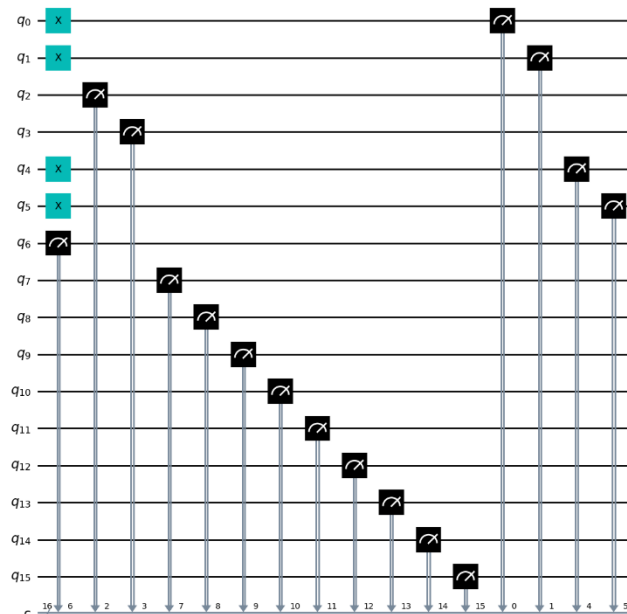
6 CONCLUSION

Conclusion

In this project, we explored the application of quantum computing techniques to solve Sudoku puzzles, a representative example of NP-complete problems, and extended the approach to other simple games. Using quantum algorithms and quantum circuits, we demonstrated how quantum computing can enhance the efficiency of solving combinatorial and constraint-based problems, leveraging the principles of superposition, entanglement, and quantum parallelism.

Through the use of quantum tools such as D-Wave's SAPI, qbsolv, and Python libraries, we encoded game constraints into quantum systems, utilizing quantum annealing and other quantum optimization methods to find valid solutions. While classical algorithms are commonly used for solving such problems, quantum computing holds the potential for faster and more scalable processing, especially for larger and more complex game scenarios.

The project highlights the versatility of quantum computing in addressing optimization and constraint satisfaction challenges across a wide range of games. Examples include **Quantum Sudoku, Tic-Tac-Toe etc.** Each of these games can benefit from the unique capabilities of quantum computing, such as quantum parallelism and probabilistic state exploration, to provide novel and efficient solutions.

This research underscores the transformative potential of quantum computing, not only for solving traditional puzzles like Sudoku but also for innovating in the realm of simple games. Additionally, it opens pathways for broader applications in areas such as cryptography, machine learning, and large-scale optimization problems. As quantum hardware continues to evolve, the scope and practicality of quantum computing in game development and other computational domains are expected to grow, making this an exciting field for future exploration.

DEPARTMENT OF CSE, RYMEC, BALLARI

7 REFERENCE'S

References:

1. Python Documentation:
   - Python 3.7.1 official documentation for language features, standard library functions, and modules.
   - URL: https://docs.python.org/3.7/

2. Qiskit Documentation:
   - Comprehensive guide for using Qiskit to create and simulate quantum circuits.
   - URL: https://qiskit.org/documentation/

3. NumPy Documentation:
   - Reference for NumPy functions used for numerical computations and random number generation.
   - URL: https://numpy.org/doc/stable/

4. Matplotlib Documentation:
   - Official documentation for visualization and saving quantum circuit diagrams.
   - URL: https://matplotlib.org/stable/contents.html

5. IBM Quantum Experience:
   - IBM's cloud-based quantum computing platform for simulating and running quantum circuits.
   - URL: https://quantum-computing.ibm.com/

6. Sudoku Rules and Gameplay:
   - General reference for the rules of Sudoku, adapted for the 4x4 grid format.
   - URL: https://sudoku.com/

DEPARTMENT OF CSE, RYMEC, BALLARI

STUDENT DETAILS

1. **Name: M Srinivas Reddy**
2. **USN: 3VC21CS091**
3. **Phone No: 8050621229**
4. **Email-ID: srinu.cse.rymec@gmail.com**
5. **Permanent Address: Havambhavi Rama Nagar Bellari**

1. **Name: K Sree Deepa**
2. **USN: 3VC21CS077**
3. **Phone No: 9513403890**
4. **Email-ID: sreedeepa.cse.rymec@gmail.com**
5. **Permanent Address: Vidhya Nagar, Bathri Road Bellari**

Appendix A

Appendix A: Quantum-Based Sudoku Game Explanation

This appendix explains the quantum-based approach implemented in the Sudoku project described in the main chapters. The game combines traditional Sudoku logic with quantum mechanics concepts to enhance the puzzle-solving experience and introduce an educational perspective on quantum computation.

A.1 Quantum Circuit Representation

The project utilizes a quantum circuit to represent the state of the Sudoku board. Each cell in the 4x4 Sudoku grid is mapped to a qubit in the quantum circuit. The following steps describe the quantum representation process:

1. Board Encoding into Qubits:

   o Each cell is uniquely indexed to correspond to a qubit. For example, the cell in the first row and first column maps to qubit 0, the second cell to qubit 1, and so on.

   o When a valid number is placed in a cell, an X gate is applied to the corresponding qubit, flipping its state to indicate that the cell has been filled.

2. Quantum Measurement:

   o For every completed 2x2 sub-grid, the circuit is measured, and the results are visualized. This process demonstrates how quantum mechanics concepts can be applied to classical problems, such as Sudoku.

3. Final Circuit Composition:

   o At the end of the game, all individual circuits representing intermediate states are combined into a final circuit. This final circuit encapsulates the entire gameplay process, which can then be visualized and analyzed.

A.2 Sudoku Logic Integration

The classical Sudoku logic is seamlessly integrated with quantum operations. Key steps include:

1. Validation of Moves:

   o A move is valid if the number does not already exist in the same row, column, or 2x2 sub-grid. This ensures adherence to Sudoku rules.

   o The is_valid_move() function implements this logic by checking the constraints for the given move.

28

2. Random Initialization:

   o At the start of the game, the board is partially filled with random numbers, ensuring that the initial setup respects Sudoku constraints.

3. Score Management:

   o Players earn points for completing sub-grids and the entire board. The integration of quantum circuit visualization adds a unique layer to the scoring system.

A.3 Mathematical Framework for Quantum Representation

The quantum aspect of the project can be mathematically interpreted as follows:

1. Qubit Mapping:

   o The board of size 4×4 requires 16 qubits. Each qubit represents the binary state of its corresponding cell:
   $$q_{ij} = \begin{cases} 1 & \text{if cell } (i,j) \text{ is filled} \\ 0 & \text{if cell } (i,j) \text{ is empty} \end{cases}$$

2. Quantum Gate Application:

   o An X gate flips the state of a qubit from 0 to 1, representing the placement of a number in the Sudoku grid.

3. Circuit Measurement:

   o The measurement of qubits in the completed sub-grid allows the visualization of the circuit, illustrating how quantum gates interact over the gameplay.

A.4 Algorithmic Perspective

The algorithm follows these steps:

1. Initialization:

   o Create an empty 4×4 board.

   o Randomly pre-fill some cells using a uniform distribution while ensuring constraints are met.

2. Player Interaction:

   o Accept player input to fill cells.

   o Validate moves using classical constraints and update the quantum circuit.

3. Quantum Circuit Updates:

   o Apply quantum gates to represent valid moves.

DEPARTMENT OF CSE, RYMEC, BALLARI

- For completed sub-grids, measure and save the quantum circuit visualization.

4. Game Completion:

- Combine all individual circuits into a final quantum circuit representing the entire gameplay.

Appendix B

Grid 1

	3	1	6	7		4			9
			8	3					
8	2								1
	7	4			8	1	6		
	8		6						4
9		2						7	3
4	9			5	7	2	3		
2			9		5		7		
7		3	2		6			1	

Grid 2

2				9	4			1
	6						3	
1	5		6		7			
	1	2	5	4	8	6		7
	8	9	2		6	1	4	3
	4	7					5	
4			9					
		6	5	3	2			
2					9		6	

Grid 3

	1				9	2	8
4		1	8	5		6	
	7	8				4	
						6	
2	5		4		8		1
6	1	3			7	5	
8		2		9	6		5
7		5		1		3	
1	9	6	7	5		8	

Grid 4

	6	8		2	4	9		
3	4		5	1			8	7
1		7	3	8		5		
	8	5		3	1	4	7	
2	1	9			7			
		4						2
4	7	3	6			9	1	
					4			
8				3			6	

Grid 5

				7	5		4	9
4	5	6	9				1	8
			6			7		
9	8		2					
2		7	1		8			
		3			5	8		7
4	2			6	7	1	8	
5				2			7	4
3			5	1		9		2

Grid 6

		4	2					6
	7	9		5		2		
				1				
4			7	1				
2		3						
9	8		5					
				4		9		
		3				5		
	2			7	4			

Grid 7

		1	2	3				
					7			
	8					1		
		7		5	6			
6						4	2	
2			8					
4	3					6		
	6	9	8					
			5			3		

Grid 8

			8			1	
			7			8	2
			5	3	6		
	8			9			
7		6					
2				9	4		
							5
3		5			8		
6				7		3	

Grid 9

		1			9		
				4	8		
8		7	3				
	2		9	1			
4			2	6			
	6						
	5				4		
3	4				1		
		9	6	2			

Grid 10

					3	4	
	9		5				6
2	5						
1	2				7		
5		4	3				
7							
6		1	9			4	
			6			3	
2			8				

Grid 11 (repeat of Grid 6)

		4	2					6
	7	9		5		2		
				1				
4			7	1				
2		3						
9	8		5					
				4		9		
		3				5		
	2			7	4			

Grid 12 (repeat of Grid 7)

		1	2	3				
					7			
	8					1		
		7		5	6			
6						4	2	
2			8					
4	3					6		
	6	9	8					
			5			3		

Grid 13 (repeat of Grid 8)

			8			1	
			7			8	2
			5	3	6		
	8			9			
7		6					
2				9	4		
							5
3		5			8		
6				7		3	

Grid 14 (repeat of Grid 9)

		1			9		
				4	8		
8		7	3				
	2		9	1			
4			2	6			
	6						
	5				4		
3	4				1		
		9	6	2			

Grid 15 (repeat of Grid 10)

					3	4	
	9		5				6
2	5						
1	2				7		
5		4	3				
7							
6		1	9			4	
			6			3	
2			8				

DEPARTMENT OF CSE, RYMEC, BALLARI