**TABLE OF CONTENTS**

* INTRODUCTION
* WORKING
* FUNCTIONAL REQUIREMENTS
* OUTPUTS
* CODE

**PROJECT TOPIC:** SUDOKU SOLVER

## INTRODUCTION:

## This report delves into the progression of solving techniques and software implementations used in creating Sudoku solvers, specifically focusing on a C++ and Qt-based solver.

A Sudoku solver is a program, algorithm, or tool that can automatically solve Sudoku puzzles. It's typically a piece of software designed to take a partially filled Sudoku grid m as input and compute the correct solution based on the puzzle's rules.

This report aims to provide a detailed look at this developed game application.

## WORKING:

## Initialization:

## The code starts by initializing a QApplication object to manage application-wide resources. A QWidget window is created with a title and a dull yellow background.

## UI Setup:

## QLabel widgets for the title and instructions are created and styled for user guidance.

## Input Grid Creation:

## A 4x4 grid of QLineEdit widgets is created for user input, each validated to accept numbers between 0 and 4. The input fields are styled with a dull yellow background.

## Button Creation:

## Three QPushButton widgets are created: "Solve", "Solve More", and "Exit". Initially, only the "Solve" button is visible.

## Layout Setup:

## A QGridLayout organizes the labels, input grid, and buttons within the window.

## Solve Button Functionality:

## When "Solve" is clicked, the title and instructions are hidden, and the Sudoku grid is populated from the input fields. Empty cells are identified, and possible values are determined. The solveSudoku function is called to solve the puzzle using a backtracking algorithm. The solution is displayed, and the "Solve More" and "Exit" buttons are shown.

## Solve More Button Functionality:

## Clicking "Solve More" resets the puzzle grid and shows the title and instructions again.

## Exit Button Functionality:

## The "Exit" button closes the application.

## Application Execution:

## The window is displayed using window.show(), and the QApplication event loop is started with app.exec().

## FUNCTIONAL REQUIREMENTS:

## Main Window:

## The application must have a main window titled "Sudoku Solver" with a dull yellow background, including a title and instructions label.

## Input Fields:

## A 4x4 grid of QLineEdit input fields validated to accept numbers between 0 and 4.

## Buttons:

## Three buttons: "Solve" to trigger Sudoku solving, "Solve More" to reset the grid, and "Exit" to close the application.

## Sudoku Solving:

## The application must read and validate user input, populate the grid, identify empty cells, and use a backtracking algorithm to solve the puzzle. Display the solution or show a warning if no solution exists.

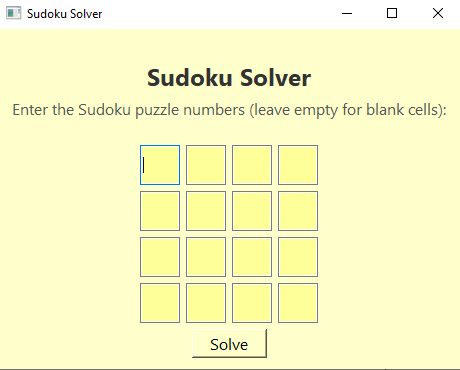
## Memory Management:

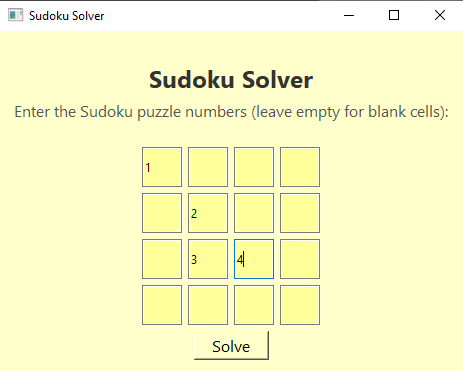
## The application must properly manage dynamically allocated memory to avoid memory leaks.

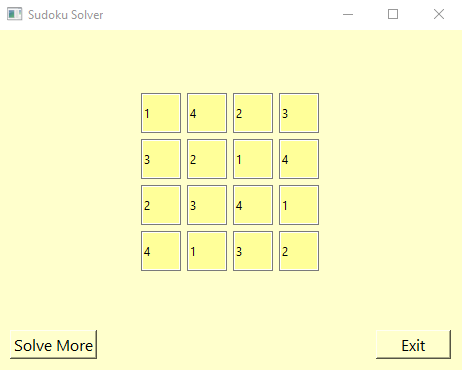
## User Interaction:

## An intuitive interface with clear labels and instructions. Handle invalid inputs gracefully by warning the user and preventing further processing until valid input is provided.

## OUTPUTS:







|  |
| --- |
| **CODE:**  #include <QApplication>  #include <QWidget>  #include <QLabel>  #include <QPushButton>  #include <QLineEdit>  #include <QGridLayout>  #include <QIntValidator>  #include <QMessageBox>  #include <QColor>  #include <iostream>  #include <vector>  using namespace std;  #define N 4  struct Node {  int data;  Node\* left;  Node\* right;  Node(int val) : data(val), left(nullptr), right(nullptr) {}  };  Node\* insert(Node\* root, int val) {  if (!root) {  return new Node(val);  }  if (val < root->data) {  root->left = insert(root->left, val);  }  else {  root->right = insert(root->right, val);  }  return root;  }  bool search(Node\* root, int val) {  if (!root) {  return false;  }  if (root->data == val) {  return true;  }  if (val < root->data) {  return search(root->left, val);  }  return search(root->right, val);  }  void printGrid(int grid[N][N]) {  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  cout << grid[i][j] << " ";  }  cout << endl;  }  }  bool isSafe(int grid[N][N], int row, int col, int num) {  if (num < 1 || num > N) {  return false;  }  for (int x = 0; x < N; x++) {  if (grid[row][x] == num || grid[x][col] == num) {  return false;  }  }  int startRow = row - row % 2;  int startCol = col - col % 2;  for (int i = 0; i < 2; i++) {  for (int j = 0; j < 2; j++) {  if (grid[i + startRow][j + startCol] == num) {  return false;  }  }  }  return true;  }  bool solveSudoku(int grid[N][N], vector<vector<Node\*>>& emptyCells) {  int row, col;  bool found = false;  for (row = 0; row < N; row++) {  for (col = 0; col < N; col++) {  if (grid[row][col] == 0) {  found = true;  break;  }  }  if (found) {  break;  }  }  if (!found) {  return true;  }  for (Node\* node : emptyCells[row \* N + col]) {  int num = node->data;  if (isSafe(grid, row, col, num)) {  grid[row][col] = num;  if (solveSudoku(grid, emptyCells)) {  return true;  }  grid[row][col] = 0;  }  }  return false;  }  int main(int argc, char\* argv[]) {  QApplication app(argc, argv);  QWidget window;  window.setWindowTitle("Sudoku Solver");  window.setStyleSheet("background-color: #FFFFCC;");  QLabel\* titleLabel = new QLabel("Sudoku Solver");  titleLabel->setStyleSheet("font-size: 24px; font-weight: bold; color: #333333; margin-top: 20px;");  QLabel\* instructionsLabel = new QLabel("Enter the Sudoku puzzle numbers (leave empty for blank cells):");  instructionsLabel->setStyleSheet("font-size: 16px; color: #555555; margin-bottom: 20px;");  QGridLayout\* gridLayout = new QGridLayout();  vector<vector<QLineEdit\*>> puzzleInputs(N, vector<QLineEdit\*>(N));  QColor dullYellow(255, 255, 153); // Dull Yellow  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  puzzleInputs[i][j] = new QLineEdit();  puzzleInputs[i][j]->setFixedSize(40, 40);  // Set validator to allow only numbers from 0 to 4  QIntValidator\* validator = new QIntValidator(0, N);  puzzleInputs[i][j]->setValidator(validator);  // Set background color to dull yellow  puzzleInputs[i][j]->setStyleSheet(QString("background-color: %1").arg(dullYellow.name()));  gridLayout->addWidget(puzzleInputs[i][j], i, j);  }  }  QPushButton\* solveButton = new QPushButton("Solve");  solveButton->setStyleSheet("font-size: 16px; background-color: #FFFFCC;"); // Dull Yellow  QPushButton\* solveMoreButton = new QPushButton("Solve More");  solveMoreButton->setStyleSheet("font-size: 16px; background-color: #FFFFCC;"); // Dull Yellow  solveMoreButton->setVisible(false);  QPushButton\* exitButton = new QPushButton("Exit");  exitButton->setStyleSheet("font-size: 16px; background-color: #FFFFCC;"); // Dull Yellow  exitButton->setVisible(false);  QGridLayout\* layout = new QGridLayout();  layout->addWidget(titleLabel, 0, 0, 1, 4, Qt::AlignCenter);  layout->addWidget(instructionsLabel, 1, 0, 1, 4, Qt::AlignCenter);  layout->addLayout(gridLayout, 2, 0, 1, 4, Qt::AlignCenter);  layout->addWidget(solveButton, 3, 1, 1, 2, Qt::AlignCenter);  layout->addWidget(solveMoreButton, 4, 0, 1, 1, Qt::AlignLeft);  layout->addWidget(exitButton, 4, 3, 1, 1, Qt::AlignRight);  window.setLayout(layout);  QObject::connect(solveButton, &QPushButton::clicked, [&]() {  titleLabel->hide();  instructionsLabel->hide();  int grid[N][N];  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  QString text = puzzleInputs[i][j]->text();  int num = text.isEmpty() ? 0 : text.toInt();  if (num > N) {  QMessageBox::warning(&window, "Invalid Input", "Please enter numbers from 0 to 4.");  return;  }  grid[i][j] = num;  }  }  vector<vector<Node\*>> emptyCells(N \* N);  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  if (grid[i][j] == 0) {  for (int num = 1; num <= N; num++) {  if (isSafe(grid, i, j, num)) {  emptyCells[i \* N + j].push\_back(new Node(num));  }  }  }  }  }  if (solveSudoku(grid, emptyCells)) {  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  puzzleInputs[i][j]->setText(QString::number(grid[i][j]));  }  }  solveButton->setVisible(false);  solveMoreButton->setVisible(true);  exitButton->setVisible(true);  }  else {  QMessageBox::warning(&window, "No Solution", "No solution exists for the provided Sudoku puzzle.");  }  for (auto& nodes : emptyCells) {  for (Node\* node : nodes) {  delete node;  }  }  });  QObject::connect(solveMoreButton, &QPushButton::clicked, [&]() {  titleLabel->show();  instructionsLabel->show();  solveButton->setVisible(true);  solveMoreButton->setVisible(false);  exitButton->setVisible(false);  for (int i = 0; i < N; i++) {  for (int j = 0; j < N; j++) {  puzzleInputs[i][j]->clear();  }  }  });  QObject::connect(exitButton, &QPushButton::clicked, [&]() {  window.close();  });  window.show();  return app.exec();  } |