Experiment No. 2B

Semester	T.E. Semester VI
Subject	ARTIFICIAL INTELLIGENCE (CSL 604)
Subject Professor In-charge	Prof. Avinash Shrivas
Assisting Teachers	Prof. Avinash Shrivas

Student Name	Deep Salunkhe
Roll Number	21102A0014
Lab Number	310A

Title:

Tic-Tac-Toe implementation

Theory:

Tic-Tac-Toe is a classic game that involves two players, usually denoted as 'X' and 'O', taking turns marking spaces in a 3x3 grid. The objective of the game is to get three of your symbols in a row, column, or diagonal. While Tic-Tac-Toe is a simple game, designing an AI to play it effectively involves implementing various strategies and algorithms.

In your implementation, you've created a simple AI that plays against the user. The AI employs a basic strategy to determine its moves, aiming to both defend against the user's potential winning moves and set up its own winning opportunities.

Ritch and Knight's Algorithm:

The Ritch and Knight's Algorithm is a strategic approach to playing Tic-Tac-Toe, named after its inventors. It involves analyzing the current board state and making moves based on certain priorities. The

algorithm considers blocking the opponent from winning, setting up its own winning moves, and creating opportunities for future wins.

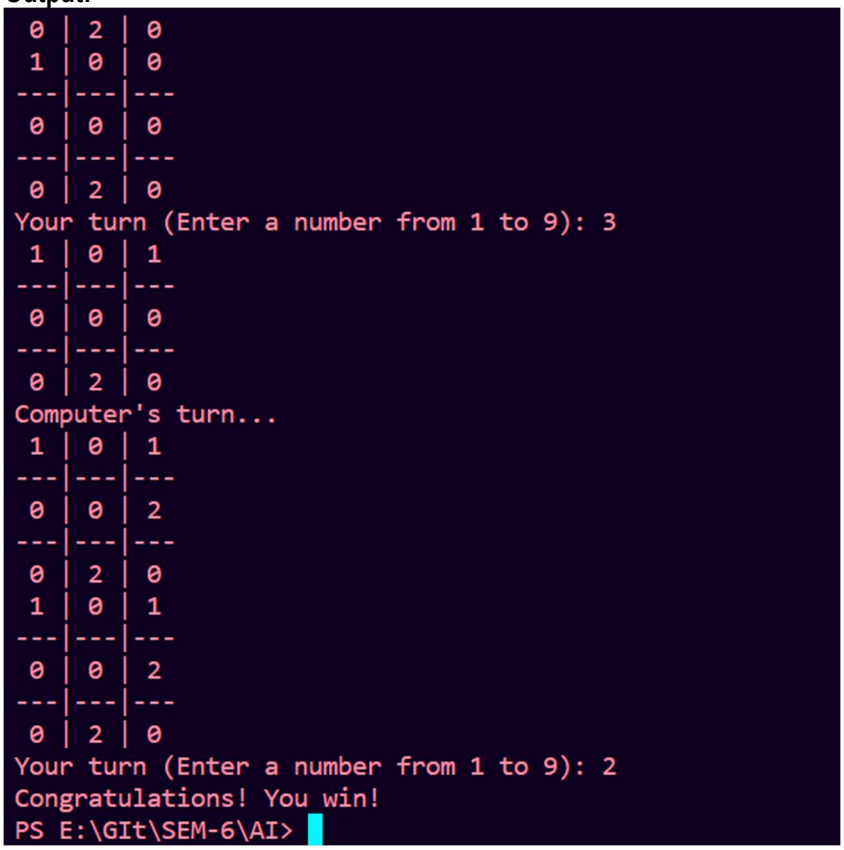
Program Code:

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <algorithm>
#include <ctime>
using namespace std;

// Function to display the Tic Tac Toe board
void displayBoard(const vector<int>& board) {
    cout << " " << board[0] << " | " << board[1] << " | " << board[2] << endl;
    cout << "---|---|---" << endl;
    cout << " " << board[3] << " | " << board[4] << " | " << board[5] << endl;
    cout << "---|---|---" << endl;
    cout << " " << board[6] << " | " << board[7] << " | " << board[8] << endl;
}

// Function to check if a player has won
bool checkWin(const vector<int>& board, int player) {
    vector<vector<int>> winPatterns = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}, {0, 3,
6}, {1, 4, 7}, {2, 5, 8}, {0, 4, 8}, {2, 4, 6}};
    for (const auto& pattern : winPatterns) {
        if (board[pattern[0]] == player && board[pattern[1]] == player &&
board[pattern[2]] == player) {
            return true;
        }
    }
    return false;
}

// Function to check if the board is full
bool boardFull(const vector<int>& board) {
    for (int cell : board) {
        if (cell == 0) {
            return false;
        }
    }
    return true;
}

// Function to make the computer play using the Rich and Knight's strategy
```

```
void computerPlay(vector<int>& board) {
    // Magic square for the "Rich and Knight's" strategy
    vector<int> magicSquare = {8, 1, 6, 3, 5, 7, 4, 9, 2};
    vector<int> availableMoves;

    // Find available moves
    for (int i = 0; i < 9; ++i) {
        if (board[i] == 0) {
            availableMoves.push_back(i);
        }
    }

    // Prioritize moves based on the magic square
    for (int i : magicSquare) {
        if (find(availableMoves.begin(), availableMoves.end(), i - 1) != availa-
bleMoves.end()) {
            board[i - 1] = 2; // Place computer's symbol (O) at the chosen posi-
tion
            return;
        }
    }
}

int main() {
    // Initialize the Tic Tac Toe board
    vector<int> board(9, 0); // 0 represents an empty cell

    // Randomly decide who plays first
    srand(time(0));
    bool playerTurn = rand() % 2 == 0;

    // Main game loop
    while (true) {
        // Display the board
        displayBoard(board);

        // Check if it's the player's turn
        if (playerTurn) {
            int move;
            cout << "Your turn (Enter a number from 1 to 9): ";
            cin >> move;

            // Validate the player's move
            if (move < 1 || move > 9 || board[move - 1] != 0) {
                cout << "Invalid move! Try again." << endl;
                continue;
            }
```

```
            // Update the board with the player's move
            board[move - 1] = 1; // Player's symbol (X) is represented by 1

            // Check if the player has won
            if (checkWin(board, 1)) {
                cout << "Congratulations! You win!" << endl;
                break;
            }

            // Check if the game is a draw
            if (boardFull(board)) {
                cout << "It's a draw!" << endl;
                break;
            }

            // Switch turns to the computer
            playerTurn = false;
        } else { // Computer's turn
            cout << "Computer's turn..." << endl;
            computerPlay(board);

            // Display the updated board
            displayBoard(board);

            // Check if the computer has won
            if (checkWin(board, 2)) {
                cout << "Computer wins! Better luck next time." << endl;
                break;
            }

            // Check if the game is a draw
            if (boardFull(board)) {
                cout << "It's a draw!" << endl;
                break;
            }

            // Switch turns to the player
            playerTurn = true;
        }
    }

    return 0;
}
```

Output:



Conclusion:

In this lab, we implemented a Tic-Tac-Toe game with a basic AI opponent using the Ritch and Knight's Algorithm. The AI showcases a fundamental understanding of the game's strategy by prioritizing defense and offense to secure a win or prevent a loss. Overall, this lab provided valuable insights into the implementation of AI .