Tic Tac Toe Solution Plan

1. Understanding the Problem

- The game is played on a 3x3 grid.
- Two players participate, one player is X and the other is O.
- Players take turns to place their symbol on the grid.
- The game ends when there's a winner or a draw.
- The game is considered a draw if all the squares are filled and there's no winner.
- Characteristics of Tic Tac Toe:
  - Futile game: No winning strategy exists for either player if both players play optimally.
  - Zero-sum game: If one player wins, the other player loses. In a draw, both players have the same score.
  - Perfect information: All previous moves are known to both players when making a decision.
  - There are 8 winning combinations (3 horizontal, 3 vertical, 2 diagonal).
  - Brute force is feasible as there are 255,168 possible games, but there are better and faster ways to solve the game, such as depth-first search.
  - Space complexity is O(b^m) where b is the branching factor and m is the maximum depth of the game tree.

2. Planning a Solution - GRAPH SEARCH

- Search all possible moves and choose the optimal move.
- Use a heuristic to choose the optimal move. If there is a winning move, take it; otherwise, take a move that doesn't allow the opponent to win.
- Use a game tree, which is a directed graph with the following elements:
  - Root node: The initial state of the game.
  - Nodes: The states of the game.
  - Edges: The moves.
  - Leaf nodes: The terminal states of the game.

3. Solve

- Use the Minimax algorithm for computer's optimal moves:
  - Search the game tree and evaluate the best outcome for each move.
  - The computer acts as the max player and the human acts as the min player.
  - The computer will choose the move that maximizes the score.
  - The human will choose the move that minimizes the score.
  - The score is calculated by a heuristic function that returns a score for a given state of the game.

4. Test

- Test for computer win.
- Test for human win.
- Test for draw.
- Test for optimal moves.
- Test for invalid moves.
- Test the game loop.

Glossary of Terms

Problem Solving

The process of finding solutions to difficult or complex issues.

Algorithm Design

The process of designing a set of instructions to solve a problem.

Design Paradigms

A general approach to solving problems.

Branching Factor

The number of child nodes a node has.

Depth

The number of edges from the root node to a node.

Ply

A move made by one player.

Minimax

A decision rule used for minimizing the possible loss in a worst-case scenario.

Heuristic

A function that ranks alternatives in search algorithms at each branching step, based on available information, to decide which branch to follow.

Zero-Sum Game

A situation in which the sum of the payoffs to all players is zero, indicating that one player's gain comes at the expense of another player's loss.

Depth Limit

The maximum depth of the game tree that can be searched.

Game Theory

The study of mathematical models of strategic interaction among rational decision-makers.

Markov Property

A property stating that the future is independent of the past given the present.

Test Cases

Verifies the functionality of a particular feature of a software application.

Automated Testing

The use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.

Edge Case

A problem or situation that occurs only at an extreme (maximum or minimum) operating parameter that could cause a program to crash.

Quality Assurance

A way of preventing mistakes in software products and avoiding problems when delivering solutions or services.

Useful Algorithms

minimax algorithm / depth doesn't change in our version of tic-tac-toe

```
def max-value(state,depth):
    if (depth == 0): return value(state)
    v = -infinite
    for each s in SUCCESSORS(state):
        v = MAX(v,min-value(s,depth-1))
    return v

def min-value(state,depth):
    if (depth == 0): return value(state)
        v = infinite
    for each s in SUCCESSORS(state):
        v = MIN(v,max-value(s,depth-1))
    return v
```

Thougts on minimax

- The minimax algorithm is a recursive algorithm for choosing the next move in an n-player game, usually a two-player, zero-sum game.
- A value is associated with each position or state of the game.
  - For example, a positive value may be assigned to a winning position, a negative value to a losing position, and zero to a neutral state.
  - These value are from the perspective of the maximizing player.
- The maximizing player aims to choose a position with the maximum value, while the minimizing player aims to choose a position with the minimum value.

---

alpha-beta pruning / not useful for tic-tac-toe

```
a = best score for max-player (helen)
b = best score for min-player (stavros)
initially, we call max-value(initial, -infinite, infinite, max-depth)

def max-value(state, a, b, depth):
    if (depth == 0): return value(state)
    for s in SUCCESSORS(state):
        a = max(a, min-value(s,a,b,depth-1))
        if a >= b: return a \\ this ia a cutoff point
    return a

def min-value(state, a, b, depth):
    if (depth == 0): return value(state)
    for s in SUCCESSORS(state):
        b = min(b,max-value(s,a,b,depth-1))
        if b <= a: return b \\ this is a cutoff point
    return b
```

Thoughts on alpha-beta pruning

- Alpha-beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree.
- It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move.
- Aplha-beta pruning does not affect the final decision of the minmax algirithm -it comes to the same conclusion, but more efficiently.

Pseuo-code for Tic-Tac-Toe

```
Initialize an empty 3x3 board as a 2D array or matrix

Display the empty board

Repeat until there is a winner or a draw:
    Ask the computer to make a move
        - Determine the best move for the computer using an algorithm like minimax
        - Place the computer's mark (X) on the board at the chosen position
        - Display the updated board

    Check if the computer has won
        - Perform a check to see if the computer's mark appears in any winning combination o
        - If yes, display "Computer wins!" and end the game

    Check if the board is full
        - Iterate through the board to check if all positions are filled
        - If yes, display "Draw!" and end the game

    Ask the player to make a move
        - Get the desired position from the player
        - Check if the position is valid and available on the board
        - Place the player's mark (O) at the chosen position on the board
        - Display the updated board

    Check if the player has won
        - Perform a check to see if the player's mark appears in any winning combination on
        - If yes, display "You win!" and end the game

    Check if the board is full
        - Iterate through the board to check if all positions are filled
        - If yes, display "Draw!" and end the game
```

```
End the game
```