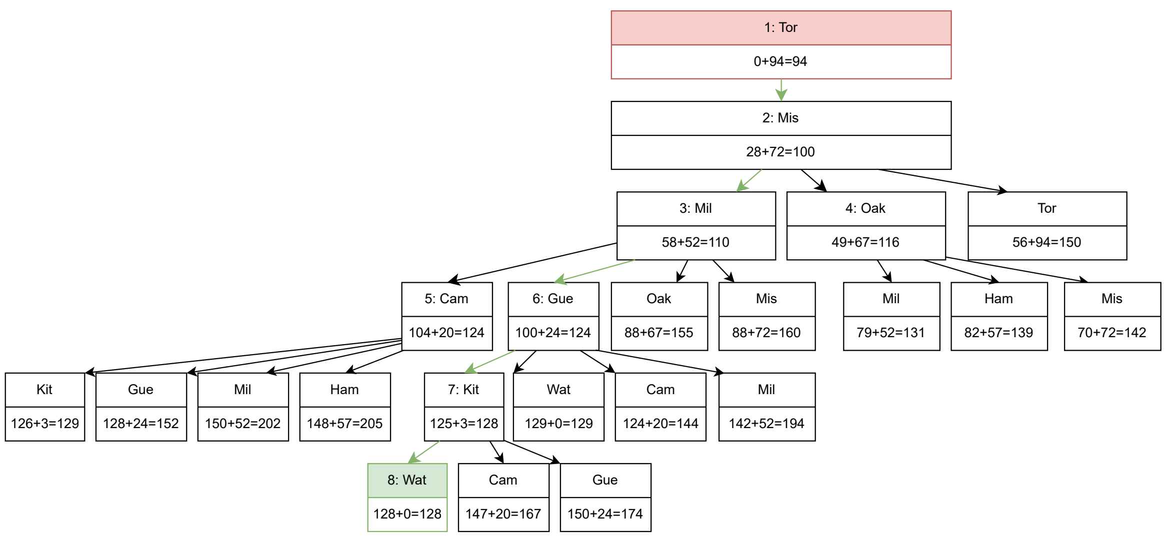
University of Waterloo
CS 486, Winter 2024
Assignment 1

Problem 1

a) The Euclidean distance to the destination is an admissible heuristic function because it never overestimates the actual shortest path cost, as the shortest possible path between two cities is the straight line (Euclidean) distance between them, thus the only possible case is the Euclidean distance equals or **underestimates** the shortest path cost.

b) The Euclidean distance to the destination is a consistent heuristic function because it satisfies the monotone restriction: **heuristic estimate of the path cost is always less than or equal to the actual cost**. As previously stated the Euclidean distance could only equals or **underestimates** the shortest path cost.

c) Decision Tree:



Problem 2

a) minimax:

```
if depth == 0 or node.state.is_terminal:
    # Base case: evaluate the state using the heuristic function
    node.value = heuristic_fn(node.state, max_role)
    return node.value
if node.state.turn == max_role:
    # Maximizing player
    max_eval = float('-inf')
    for move in node.state.get_legal_moves():
        # Create a successor node
        successor_state = deepcopy(node.state)
        successor_state.advance_state(move)
        successor_node = MinimaxNode(successor_state)
        # Recursively call minimax on the successor
        eval = minimax(successor_node, depth - 1, max_role, heuristic_fn)
        # max_eval is the maximum evaluation value for the player.
        max_eval = max(max_eval, eval)
        # Store successor node and its value
        node.successors[move] = successor_node
    # Assign the maximum value found to the current node
    node.value = max_eval
    return max_eval
else:
    # Minimizing player
    min_eval = float('inf')
    for move in node.state.get_legal_moves():
        # Create a successor node
        successor_state = deepcopy(node.state)
        successor_state.advance_state(move)
        successor_node = MinimaxNode(successor_state)
        # Recursively call minimax on the successor
        eval = minimax(successor_node, depth - 1, max_role, heuristic_fn)
        # min_eval is the minimum evaluation value for the player.
        min_eval = min(min_eval, eval)
        # Store successor node and its value
        node.successors[move] = successor_node
    # Assign the minimum value found to the current node
    node.value = min_eval
    return min_eval
```

b) The game always terminates in less than 10 seconds (around `0.04s` to `0.05s`) on my laptop for `RandomPlayer` vs. `MinimaxHeuristicPlayer` with depth `2`.

The first depth number i encountered for the game not terminating within 10 seconds is `depth=6`, which takes around `34s` on my Laptop.

c) `my_heuristic`:

```
    #If the state is terminal, give the true evaluation
    if state.is_terminal:
        if state.winner == '':
            return 0
        elif state.winner == max_role:
            return 100
        else:
            return -100
    #If the state is not terminal, give the heuristic evaluation
    score_board = [
        [1, 2, 3, 3, 2, 1],
        [2, 4, 6, 6, 4, 2],
        [3, 6, 9, 9, 6, 3],
        [4, 8, 12, 12, 8, 4],
        [3, 7, 10, 10, 7, 3],
        [2, 4, 6, 6, 4, 2],
        [1, 2, 3, 3, 2, 1],
        ]
    score = 0
    # Calculate the score for each piece based on its distance from the center column
    for col in range(state.num_cols):
        for row in range(state.num_rows):
            if state.board[col][row] == max_role:
                # Assign higher score to pieces closer to the center
                # The maximum score is given to the center column and decreases as it moves away from the center
                score += score_board[col][row]
            else:
                score -= score_board[col][row]
    return score
```

`my_heuristic` evaluates the board state by calculating a score based on the pieces' location on the board. Each spot on the board is given a score, with higher scores for spots closer to the centre. In the end, it returns the score as the difference between the total scores of the two players.

I believe `my_heuristic` will outperform `three_line_heur` because my heuristic function assign scores based on board positions, more precisely based on the pieces' distance from the centre of the board. The closer a piece is to the centre, the higher its score, as closer a piece is to the centre, higher the chance that it will contribute to a connect-4, or stop a connect-4 attempt for the opponent. And `three_line_heur` only based scores on number of 3-in-a-row, without considering how easy is that 3-in-a-row to become an actual connect-4.

d) Diagram (each run 20 games):

Agent 1	Agent 2	Wins	Draws	Losses
Minimax Player, depth 3 three_line_heur	RandomPlayer	20	0	0
Minimax Player, depth 3 my_heuristic	RandomPlayer	20	0	0
Minimax Player, depth 5 three_line_heur	Minimax Player, depth 2 three_line_heur	16	1	3
Minimax Player, depth 5 my_heuristic	Minimax Player, depth 2 my_heuristic	20	0	0
Minimax Player, depth 5 my_heuristic	Minimax Player, depth 5 three_line_heur	15	0	5

Problem 3

1) We will denote the i'th Queen's coordinate in the form (X_i, Y_i). And the domain for them is $1 \leq X_i \leq N$ and $1 \leq Y_i \leq N$ for all $i \in \{1, 2, \ldots, N\}$.

And the constraints are:

- Row Constraint: $X_i \neq X_j$ for all $i \neq j$ where $i, j \in \{1, 2, \ldots, N\}$.
- Column Constraint: $Y_i \neq Y_j$ for all $i \neq j$ where $i, j \in \{1, 2, \ldots, N\}$.
- Diagonal Constraint: $|X_i - X_j| \neq |Y_i - Y_j|$ for all $i \neq j$ where $i, j \in \{1, 2, \ldots, N\}$.

2) We will denote the i'th Octopi's coordinate in the form (X_i, Y_i). And the domain for them is $1 \leq X_i \leq N$ and $1 \leq Y_i \leq N$ for all $i \in \{1, 2, \ldots, N\}$.

And the constraints are:

- Row Constraint: $X_i \neq X_j$ for all $i \neq j$ where $i, j \in \{1, 2, \ldots, N\}$.
- Column Constraint: $Y_i \neq Y_j$ for all $i \neq j$ where $i, j \in \{1, 2, \ldots, N\}$.
- Block Constraint: $\left\lfloor \frac{X_i - 1}{M} \right\rfloor \neq \left\lfloor \frac{X_j - 1}{M} \right\rfloor$ or $\left\lfloor \frac{Y_i - 1}{M} \right\rfloor \neq \left\lfloor \frac{Y_j - 1}{M} \right\rfloor$ for all $i \neq j$ where $i, j \in \{1, 2, \ldots, N\}$.