A* + Greedy Movement Algorithm for Detectives

1. Verbal Explanation

The **A* + Greedy Movement Algorithm** estimates the next move for each detective based on a probability map of Mr. X's possible locations, weighted by the distance to the nearest detective, number of turns since the last reveal, and ticket constraints.

Core idea: Detectives select targets that are most likely to contain Mr. X. The probability for each node increases with distance from the nearest detective. When multiple nodes have the same maximum probability, each detective moves to the closest node (tie-breaker). A* search is used to compute the shortest legal path to the target, considering available tickets for both detectives and Mr. X.

Safety condition: All moves must respect legal ticket usage.

1. **Initialization:**
   - Start from current positions of all detectives.
   - Obtain last known position of Mr. X, number of turns since he was revealed, and ticket states.
   - Generate all reachable positions for Mr. X given ticket usage and turns since last reveal.
2. **Probability Map:**
   - Assign base probability to each reachable node (uniform or weighted by path count).
   - Adjust probability for each node according to distance from nearest detective:

$$P(node) = P_{base}(node) \cdot (1 + \alpha \cdot distanceToClosestDetective)$$

   - Normalize probabilities so their sum equals 1.
3. **Target Selection and Tie-Breaker:**
   - For each detective, select the node(s) with maximum probability.
   - If multiple nodes tie, choose the node closest to the detective (tie-breaker).
4. **Path Planning and Move Execution:**
   - Use A* search to compute shortest legal path to the target, considering ticket availability.
   - Move the detective along the first step of the path.

2. Flowcharts

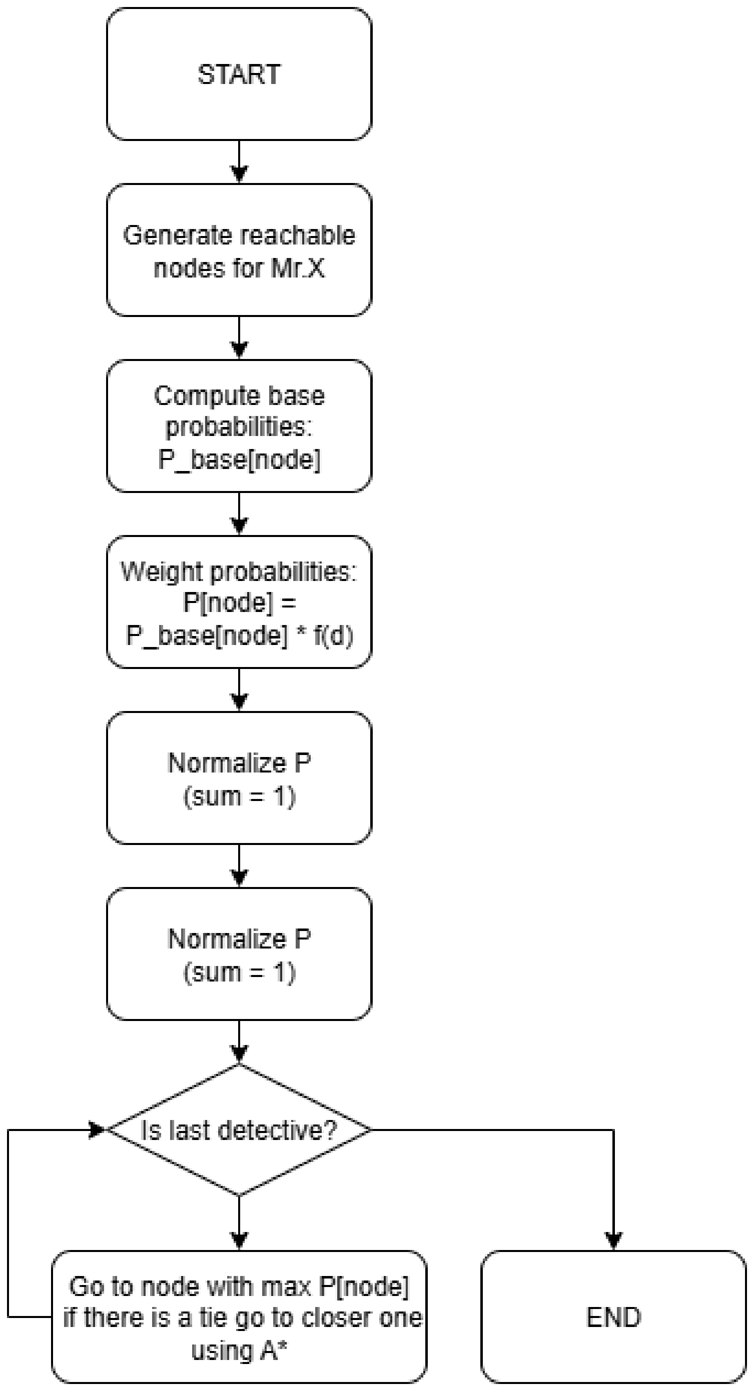
START
Generate reachable nodes for Mr.X
Compute base probabilities: P_base[node]
Weight probabilities: P[node] = P_base[node] * f(d)
Normalize P (sum = 1)
Normalize P (sum = 1)
Is last detective?
Go to node with max P[node] if there is a tie go to closer one using A*
END

Figure 1: Overall A* + Greedy Flowchart for Detectives with Distance-Weighted Probability

3. Pseudocode

Algorithm 1: A* + Greedy Algorithm for Detectives with Distance-Weighted Probability, Tie-Breaker, Turn and Ticket Constraints

```
Input: posDetectives, lastKnownPos, turnsSinceReveal, graph, detectiveTickets, mrXTickets, parameter $\alpha$
Output: Next move for each detective
Function GenerateReachableNodes(lastKnownPos, turnsSinceReveal, graph, mrXTickets):
    reachable ← {lastKnownPos};
    for t ← 1 to turnsSinceReveal do
        nextReachable ← ∅;
        foreach node in reachable do
            foreach ticket in availableTickets(mrXTickets, t) do
                foreach neighbor in neighbors(node, ticket) do
                    add neighbor to nextReachable;
                end
            end
        end
        reachable ← nextReachable;
    end
    return reachable;
Function ComputeProbabilityMap(lastKnownPos, turnsSinceReveal, graph, posDetectives, mrXTickets):
    reachableNodes ← GenerateReachableNodes(lastKnownPos, turnsSinceReveal, graph, mrXTickets);
    assign base probability $P_{base}[node]$ (uniform or path-count weighted);
    foreach node in reachableNodes do
        $d \leftarrow \min_{det \in posDetectives} distance(det.position, node)$;
        $P[node] \leftarrow P_{base}[node] \cdot (1 + \alpha \cdot d)$;
    end
    normalize $P$ so that $\sum P[node] = 1$;
    return $P$;
Function GenerateDetectiveMoves(detective, graph, detectiveTickets):
    moves ← legal moves from detective.position using available tickets;
    return moves;
probMap ← ComputeProbabilityMap(lastKnownPos, turnsSinceReveal, graph, posDetectives, mrXTickets);
foreach detective in posDetectives do
    moves ← GenerateDetectiveMoves(detective, graph, detectiveTickets[detective]);
    maxProb ← max(probMap.values());
    candidates ← nodes with $P(node) = maxProb$;
    if length(candidates) ¿ 1 then
        target ← candidate closest to detective.position // tie-breaker: nearest node
    end
    else
        target ← candidates[0]
    end
    path ← AStar(graph, detective.position, target, detectiveTickets[detective]);
    move detective along first step of path;
end
```