Mini-Max Pursuit Planning for Detectives

1. Concept Overview

The **Mini-Max algorithm with alpha–beta pruning** is employed by the detective team (maximizing player) to anticipate Mr. X's best evasive reactions (minimizing player).
The search alternates detective multi-agent moves with Mr. X counter-moves, evaluating partial futures and selecting the option that minimizes Mr. X's maximum survivability. Scotland Yard constraints such as ticket budgets, reveal turns and transport availability are embedded into the transition model.

Utility heuristic:
State value combines (a) expected capture time derived from distances to the belief centroid of Mr. X, (b) coverage score measuring how many high-probability exits are controlled, (c) ticket pressure applied to Mr. X (forcing use of scarce tickets), and (d) coordination penalty when detectives cluster unnecessarily.

Planning pipeline:

1. **Preprocessing:**
   - Build the weighted transport graph (taxi, bus, underground, ferry edges).
   - Cache shortest-path distances between all relevant nodes (reused by heuristic evaluation).
   - Gather current ticket counts and reveal schedule from the game state.
2. **Search tree generation:**
   - Alternate Mr. X and detective layers; in the detective layer branch over the most threatening pawn first.
   - Limit depth using adaptive horizon (shorter when few tickets remain or reveal is imminent).
   - Apply move ordering heuristics: prioritize moves by Mr. X escape potential and detective coverage improvement.
   - Prune dominated joint moves: if detective A reaches node N faster than B with same tickets, discard B$\rightarrow$N.
3. **Evaluation:**
   - Combine the precomputed distances with the current belief distribution for Mr. X (generated by the front-search module).
   - Add bonus for reducing entropy of the belief map and occupying choke points with high branching factor.
   - Detect forced reveals: if a reveal occurs within the horizon, forecast the revealed node and reward states that place detectives on its perimeter.
   - Sample top-K belief particles (K=5–10) to represent Mr. X hypotheses and avoid combinatorial explosion.
4. **Alpha–beta pruning and caching:**
   - Maintain alpha/beta bounds to prune branches guaranteed to be dominated by better pursuit lines.
   - Use a transposition table keyed by (detective positions, Mr. X belief hash, ticket vector, reveal index) to memoize backed-up values.
5. **Action selection:**
   - Return the coordinated detective move set with the highest backed-up value.
   - Provide a confidence score (utility gap to the runner-up plan) for UI feedback.

2. Flowcharts

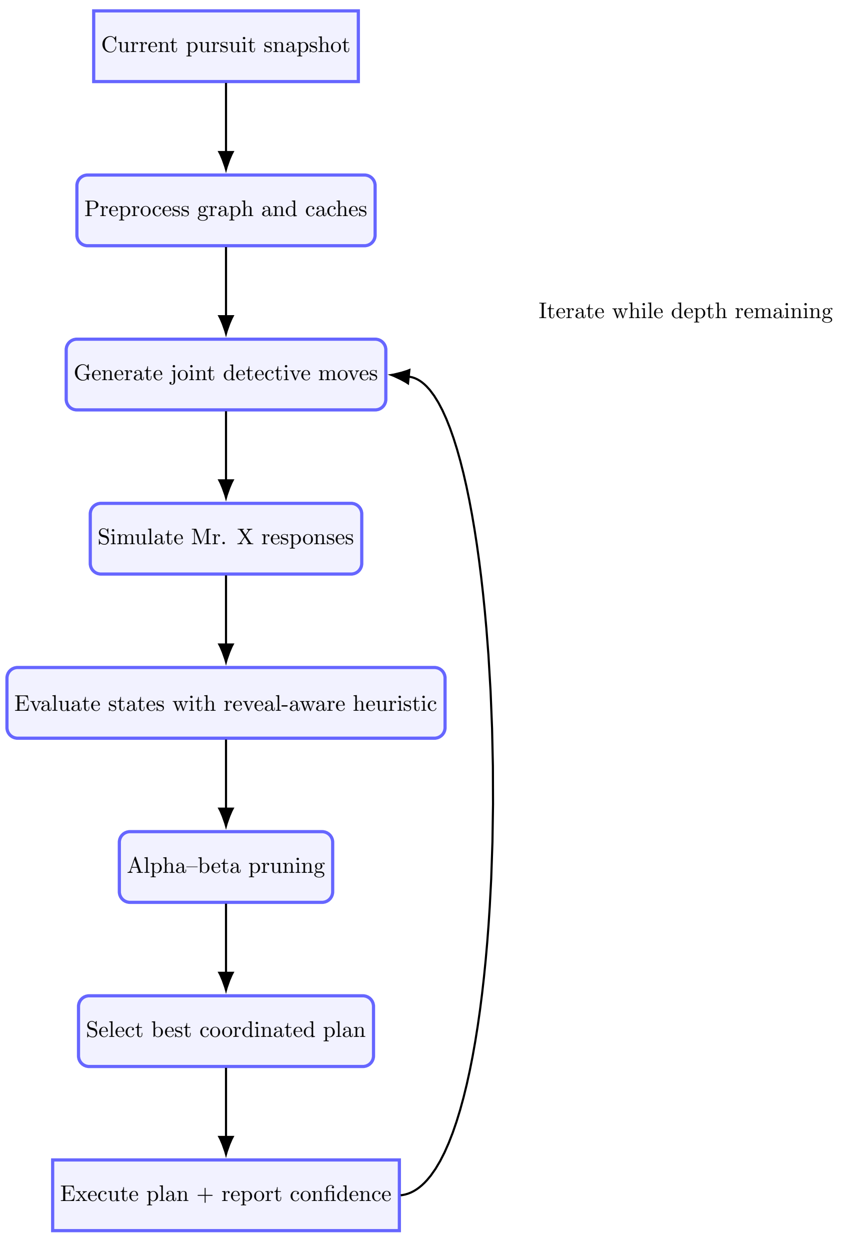
Current pursuit snapshot
Preprocess graph and caches
Iterate while depth remaining
Generate joint detective moves
Simulate Mr. X responses
Evaluate states with reveal-aware heuristic
Alpha–beta pruning
Select best coordinated plan
Execute plan + report confidence

Figure 1: Pipeline for detective-side Mini-Max planning

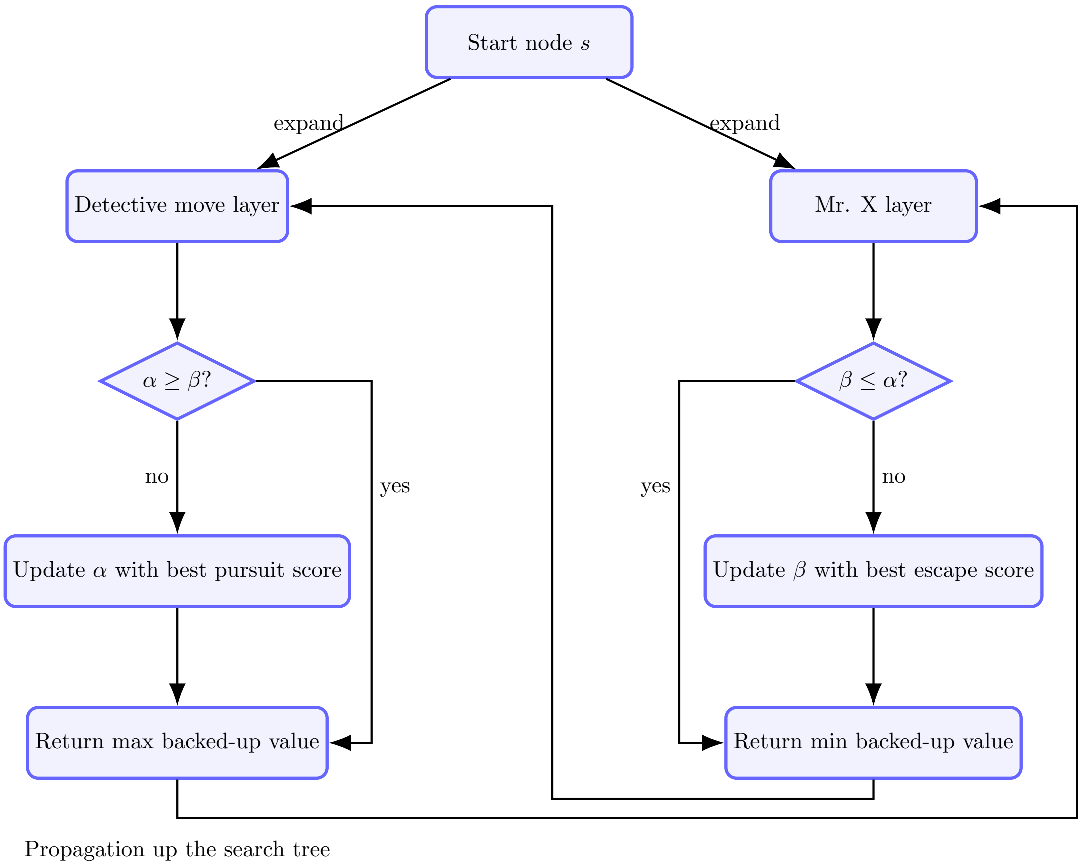
Start node s
expand
expand
Detective move layer
Mr. X layer
$\alpha \geq \beta$?
$\beta \leq \alpha$?
no
yes
yes
no
Update α with best pursuit score
Update β with best escape score
Return max backed-up value
Return min backed-up value
Propagation up the search tree

Figure 2: Recursive alpha–beta interaction between detective and Mr. X layers

3. Pseudocode

Algorithm 1: Mini-Max pursuit planning with alpha–beta pruning for detectives

```
Input: Detective positions $posDetectives$, belief distribution $beliefX$, transport graph $graph$, detective tickets
       $tickets_D$, estimated Mr. X tickets $tickets_X$, planning horizon $horizon$, reveal schedule
Output: Best coordinated detective move set for the current turn
Function Evaluate(state):
    $beliefCentroid \leftarrow$ weightedCentroid(state.beliefX);
    $distScore \leftarrow$ coverageRadius(state.detectives, beliefCentroid);
    $entropyDrop \leftarrow$ entropyReduction(state.beliefX, state.prevBelief);
    $ticketPressure \leftarrow$ ticketDifferential(state.ticketsX, state.ticketsD);
    $clumpingPenalty \leftarrow$ clusteringMetric(state.detectives);
    return $w_1 \cdot distScore + w_2 \cdot entropyDrop + w_3 \cdot ticketPressure - w_4 \cdot clumpingPenalty$;
Function NextStates(state, maximizing):
    if maximizing then
        return jointDetectiveMoves(state);
    particles $\leftarrow$ topKBeliefNodes(state.beliefX, K=8);
    responses $\leftarrow$ [];
    foreach hypothesis in particles do
        append evasiveMovesForMrX(state, hypothesis) to responses;
    end
    return mergeMrXResponses(responses);
Function MiniMax(state, depth, $\alpha$, $\beta$, maximizing):
    if depth = 0 or terminal(state) then
        return Evaluate(state);
    end
    if hash(state) in transpositionTable then
        return transpositionTable[hash(state)];
    end
    if maximizing then
        value $\leftarrow -\infty$;
        foreach child in NextStates(state, true) do
            value $\leftarrow$ max(value, MiniMax(child, depth-1, $\alpha$, $\beta$, false));
            $\alpha \leftarrow$ max($\alpha$, value);
            if $\alpha \geq \beta$ then
                break;
            end
        end
    else
        value $\leftarrow +\infty$;
        foreach child in NextStates(state, false) do
            value $\leftarrow$ min(value, MiniMax(child, depth-1, $\alpha$, $\beta$, true));
            $\beta \leftarrow$ min($\beta$, value);
            if $\beta \leq \alpha$ then
                break;
            end
        end
    end
    transpositionTable[hash(state)] $\leftarrow$ value;
    return value;
Initialize $bestValue \leftarrow -\infty$, $bestPlan \leftarrow None$;
Initialize state $\leftarrow$ current pursuit snapshot;
foreach plan in jointDetectiveMoves(state) do
    nextState $\leftarrow$ simulateDetectivePlan(state, plan);
    value $\leftarrow$ MiniMax(nextState, horizonDepth(state), $-\infty$, $+\infty$, false);
    if value > bestValue then
        bestValue $\leftarrow$ value, bestPlan $\leftarrow$ plan;
    end
end
Output bestPlan with confidenceMargin(bestValue, plans);
```