GOAL FORMULATION

The goal of the gold miner agent is to reach the tile on the board containing the pot of gold, referred to hereafter as the "gold tile," without falling into a pit; this is the only goal of the nonrational agent. As part of its metric for rationality, the smart agent additionally aims to minimize the number of moves it takes to reach the gold tile.

PROBLEM FORMULATION

From its initial position at the upper left corner square (1, 1), the agent performs one of three possible actions to reach the gold tile. While searching, the agent interacts with four elements in its environment (i.e., the board): tiles with pits, tiles with beacons, the gold tile, and the size of the board itself.

The problem formulation for the nonrational agent is straightforward, as it only aims to arrive at the gold tile and avoid pits. On the other hand, the smart agent aims to arrive at the gold tile using the minimal number of moves, which necessitates the use of an optimal pathfinding algorithm.

AGENT STATES AND CONFIGURATIONS

Both versions of the agent have three states: the initial state, where it is located on the upper left-hand corner of the grid; the active state, where its three actions can be performed; and the terminal state, whereupon further actions can no longer be performed.

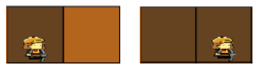
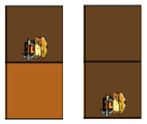
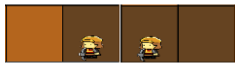
The terminal state itself can be further divided into two. In the *terminal success state*, the agent is able to reach the gold tile. On the contrary, in the *terminal fail state,* it either lands on a pit tile or determines that it is impossible to reach the gold tile with the given board configuration. Note that the former only applies to the nonrational agent while the latter, only to the rational. Symbolically,

State = (x, y, D, M), where x is the row number and y is the column number of the miner's position, D is the direction the miner faces[1], and M is the collection of its memorized tile details
Initial State = (1, 1, *right*, *NULL*). Refer to this footnote[2] as regards the nonrational agent.
Goal State = (x_G, y_G, *, *) where x_G and y_G represent the row and column numbers of the gold tile and * refers to any valid value

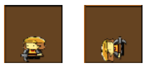
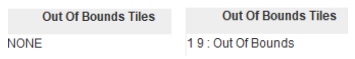
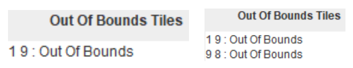
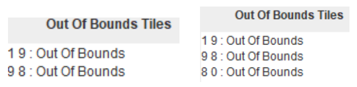
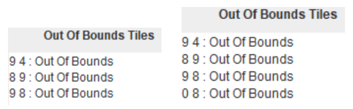
The agent itself has four configurations, corresponding to the four cardinal directions to which it can be facing: east (right), south (down), west (left), and north (up).

ACTIONS AND TRANSITION TABLE

Majority of the states and actions of both the nonrational and smart agents are similar; therefore, the states and operator table are presented in the context of the smart agent. In cases where the details of the nonrational agent differ, footnotes specifying these differences are included.

Name	Conditions	Transition	Illustration	Effect
Move [facing right]	D = *right* and $y + 1 \leq n$	$(x, y, right, M) \rightarrow (x, y + 1, right, M)$		It moves one tile to the right.
Move [facing down]	D = *down* and $x + 1 \leq n$	$(x, y, down, M) \rightarrow (x + 1, y, down, M)$		It moves one tile down.
Move [facing left]	D = *left* and $y - 1 \geq 1$	$(x, y, left, M) \rightarrow (x, y - 1, left, M)$		It moves one tile to the left.

[1] The nonrational agent does not memorize the information of the tiles it scans; hence, its memory can be regarded as having a constant value of *NULL*.
[2] Though the smart agent initially faces to the right, the direction of the initial state of the nonrational agent is randomized.

Move [facing up]	$D = up$ and $x - 1 \geq 1$	$(x, y, up, M) \rightarrow (x - 1, y, up, M)$		It moves one tile up.
Rotate [facing right]	$D = right$	$(x, y, right, M) \rightarrow (x, y, down, M)$		It turns to face down.
Rotate [facing down]	$D = down$	$(x, y, down, M) \rightarrow (x, y, left, M)$		It turns to face to the left.
Rotate [facing left]	$D = left$	$(x, y, left, M) \rightarrow (x, y, up, M)$		It turns to face up.
Rotate [facing up]	$D = up$	$(x, y, up, M) \rightarrow (x, y, right, M)$		It turns to face to the right.
Scan [facing right]	$D = right$	$(x, y, right, M \rightarrow (x, y, right, M +$ *east tile info*$)$ [3]	 Out Of Bounds Tiles NONE Out Of Bounds Tiles 1 9 : Out Of Bounds	Details about the tile to the right are memorized.
Scan [facing down]	$D = down$	$(x, y, down, M) \rightarrow (x, y, down, M +$ *south tile info*$)$ [3]	 Out Of Bounds Tiles 1 9 : Out Of Bounds Out Of Bounds Tiles 1 9 : Out Of Bounds 9 8 : Out Of Bounds	Details about the tile below are memorized.
Scan [facing left]	$D = left$	$(x, y, left, M) \rightarrow (x, y, left, M +$ *west tile info*$)$ [3]	 Out Of Bounds Tiles 1 9 : Out Of Bounds 9 8 : Out Of Bounds Out Of Bounds Tiles 1 9 : Out Of Bounds 9 8 : Out Of Bounds 8 0 : Out Of Bounds	Details about the tile to the left are memorized.
Scan [facing up]	$D = up$	$(x, y, up, M) \rightarrow (x, y, up, M +$ *north tile info*$)$ [3]	 Out Of Bounds Tiles 9 4 : Out Of Bounds 8 9 : Out Of Bounds 9 8 : Out Of Bounds Out Of Bounds Tiles 9 4 : Out Of Bounds 8 9 : Out Of Bounds 9 8 : Out Of Bounds 0 8 : Out Of Bounds	Details about the tile above are memorized.

DESCRIPTION AND ANALYSIS OF THE AGENT'S BEHAVIOR

This section describes the behavior of the gold miner agent. The *nonrational agent* decides on its actions randomly as long as it does not result in the miner going out of bounds. Formally, it employs a method from Java's `ThreadLocalRandom` class, a 48-bit-seed pseudorandom linear congruential generator. Meanwhile, the *smart agent* is characterized by its use of more sophisticated decision-making processes to minimize the number of actions, as well as optimize complexities, involving the following aspects:

Memorization. Minimizing the number of actions necessitates a trade-off in terms of space complexity. The miner keeps a record of the *(i)* visited tiles, *(ii)* scanned pits and *(iii)* out-of-bounds locations that have been scanned while it is situated on an edge tile. Assuming worst case, the first two records total to at most n^2 tiles whereas the last record contains at most $4n$ tiles (since the four edges of the board each have n out-of-bounds tiles). Thus, the space complexity for the memory is $O(n^2)$.

Exploration. Among established blind-search strategies, depth-first search (DFS) features the least number of backtracks. The movement restriction to the cardinal directions implies that breadth-first search must rely on backtracking to explore same-level nodes sequentially; iterative deepening requires returning to the origin per iteration. Since it is a strategy as opposed to a single-cost action, backtracking may incur a number of moves and rotations; thus, it is imperative to employ it sparingly.

[3] As the nonrational agent does not make use of memorization, the information about the scanned tiles is not added to its memory; therefore, although the nonrational agent can perform the scan action, this does not change its state.

Ergo, the agent's approach is akin to a modified DFS since a path is explored to the maximum depth. It starts at the leftmost corner and moves to the right (the only rational moves are to move to the right or down; right was chosen arbitrarily). It continues its unidirectional movement until either it scans a pit or it finds the tile in front in the record of visited squares (note that a visited square is not scanned anymore to avoid redundancy). It rotates to find an exit (i.e., a neighboring tile that is neither a pit nor a visited square) and moves in that direction. This continues until no such exit can be found; at this point, it is defined to have reached its *local maximum depth*, and backtracking ensues.

This rotation scheme involves some intricate optimizations. Generally, the miner rotates to face the direction of the first unvisited square. However, if the four neighboring tiles have been visited or scanned as pits, then the agent does not perform any rotation since this fact can already be determined by looking at the memory without any locomotive action. It proceeds immediately to backtracking.

Improvements. The smart agent's modified DFS strategy features two salient improvements to remedy certain weaknesses of the typical approach. First, it takes advantage of the record of visited tiles to guarantee termination even if the graph representing the board contains cycles. Since the board is already a finite search space, this solves the completeness issue of the typical DFS. Second, instead of consuming $O(bm)$ space (b is the branching factor and m is the maximum depth of the entire graph of the board) to record the sibling nodes in the fringe, the agent keeps a significantly smaller stack of selected visited tiles, referred from hereon as the *path stack*, which is also crucial in backtracking.

Backtracking. The strategy takes inspiration from Prolog's mechanism [1]. The miner pushes a record of the tiles it has visited onto the path stack. When the local maximum depth is reached, it retraces its path by checking if the tile at the top of the stack has an unvisited neighbor (the miner can determine this since it remembers all the visited tiles), popping it if has none, and moving to that square (ad hoc rotation is done, but scanning is not performed since the tile has already been scanned pre-backtracking). This cycle continues until the top of the stack is a tile with an unvisited neighbor, signaling the end of the backtracking. It proceeds to this unvisited neighbor and explores it to the local maximum depth. Therefore, at any point in the program, the space complexity of the path stack is only $O(m)$; in fact, it will not exceed the maximum depth of the graph of the board as the strategy entails popping tiles that will not be part of the final path — a significant upgrade over an $O(bm)$ fringe.

Beacon Usage. The rational agent takes advantage of the distance r returned by the beacon by searching for only r square tiles in the cardinal directions; a unidirectional search can be prematurely terminated upon encountering a pit. To taper moves and backtracks, it only checks a direction if there is at least one unvisited tile among the r tiles. The average number of rotations is minimized by having the agent traverse diametrically opposite directions first (e.g., up then down rather than up then right). Another optimization introduced deals with the case when another beacon is encountered. If its return value is 0 or larger than r, then the gold square is not in this cardinal direction and the unidirectional search is prematurely terminated, immediately switching to the next direction.

Error Handling. The agent is also capable of detecting configurations where it is impossible to reach the gold square: either the miner or the gold square is enclosed in a polygonal barricade of pits. A subset of the latter case (i.e., when the four neighboring tiles of the gold square are all pits) triggers the rational agent's worst-case performance since it has to search the entire space to conclude unreachability, clocking in $O(b^m)$ time complexity. Under the hood, attempting to pop from an empty path stack during backtracking implies the exhaustion of all valid paths without reaching the goal.

As a final note, the contents of the path stack upon reaching the terminal state indicate a direct path towards the gold square tile (i.e., excluding backtracks). However, since the pathfinding employs DFS, this is not guaranteed to be the shortest path; nevertheless, as explained, it is this strategy that results in the minimum number of actions — which is the specified metric for the agent's rationality.

[1] Pearce, A. (2016). *How Prolog works*. The University of Melbourne. https://people.eng.unimelb.edu.au/adrianrp/COMP90054/lectures/Prolog_How_Works.pdf