CSINTSY MCO1: Gold Miner

Members:
Jadie, Joshue Salvador A
Lopez, Angel L.
Ponce, Andre Dominic H.

Agent Goal:
Given a rectangular grid of $n \times n$ squares, where $8 \leq n \leq 64$, this forms a mining area that contains four possible objects: (1) the miner, (2) the pot of gold, (3) a pit, and (4) a beacon. The main goal is **to produce a state where the miner agent finds a path to the golden tile.** This goal is to be achieved by two types of agents: **random** and **smart**, which differ in terms of intelligence level:

The random agent does not use a formulated goal strategy, and its behavior is to move and execute actions randomly around the mining area - which has a chance of reaching the goal state, but can also produce a resulting state wherein the agent falls into the pit.

The smart agent, on the other hand, uses a formulated goal strategy. It analyzes the states produced on each valid move done all the while avoiding pits in order to reach the goal state. The smart miner agent also aims to minimize the number of actions performed in searching for the goal state - the smaller the actions done, the more rational it is.

Problem:
The main problem for Gold Miner is to have the miner agent reach the gold tile with the least amount of actions possible. In cases that the gold tile is not accessible, the miner agent must also be able to discern this as well.

Given the board state with varying configurations, the miner agent should be able to navigate it successfully using only the partial information it gains as it traverses from tile to tile. With all these considerations at hand, the miner must be able to: (1) avoid pits, (2) check beacons, and (3) reach the gold tile.

In the context of the random miner agent, the problem is still solvable, but it is less likely to solve it when compared to the smart miner agent - mainly due to its behavior. Hence, the problem is better tackled by the smart miner agent.

State transitions and algorithms:
Given that the agent's actions are only limited to *move*, *rotate*, and *scan*, the algorithms for each agent type is explained below:

For the **random agent**, its algorithm involves the use of a **random number generator** to determine which action will be executed. Depending on the number generated, the agent will either move, rotate, or scan - with each action having a ⅓ chance of occuring. For each action performed, the current position of the agent will be checked whether it is on the gold tile or on a pit tile. If this is the case, then the search shall end immediately as these are the cases where the agent has achieved or failed to reach the goal state. Since the agent has a random behavior, and has no intelligence whatsoever, it will produce different state sequences and transitions even if it uses the same initial board state every time.

For the **smart agent**, its algorithm involves the utilization of **two data structures**: (1) a *list of visited tiles*, and (2) a *stack for backtracking*, as well as **two data**: (1) a *target object*, and (2) a *target direction*. These are used for each move performed during the search in order to determine which next move will benefit the agent the most - such that it will reach the goal state as quickly as possible.

On how the algorithm works as a whole, the target object data will be used to determine if the agent will move immediately towards its current direction, or rotate and scan all valid directions at its current state. If the object targeted is the gold, then the agent shall immediately move towards the direction targeted as to where the gold is, else, the agent will instead proceed to analyzing which directions can it rotate to and scan for objects to be targeted based on precedence levels: gold has the highest precedence followed by beacon then null (no object), and lastly, pit. For this case, if the agent has not found any direction that benefits it in reaching the goal state - such as if all corresponding tiles from each valid direction scanned are pits or do not have any objects at all, then the agent will proceed to move towards its current direction instead. However, if the current direction's corresponding tile is also a pit, then the agent proceeds to backtrack. For each backtrack process, the stack list is always checked - if there are no more tiles left in the stack, then it implies that there are no more moves left for the miner agent. With this, the search shall end immediately. Before each move to be executed by the agent, its current position is added immediately to the list of visited tiles. The list is then used for the next move to determine which directions are to be analyzed - avoiding tiles that were already visited.

In cases where the agent lands on a beacon tile, the distance returned from the beacon will be analyzed. If the returned distance $d > 0$, which implies that the gold tile is on one of the beacon's axes, then the distance will be used to find and target the gold tile in all directions

(see Appendix for tabular presentation of state transition).

Process:
Given that the behavior of the random agent is inconsistent, the smart agent will be emphasized here instead. Sample states are illustrated below to show the efficacy of the smart agent's algorithm.

For the initial board state, the miner will always be placed at the coordinates (1, 1) at the beginning - which is at the upper left-hand corner of the mining area. For the other objects in the grid, these can be placed anywhere as long as: (1) no two objects are on the same tile, and (2) its row and column values are within the boundary: $1 \leq row, column \leq n$. Figure 1 shows an example initial board state where size $n = 8$.

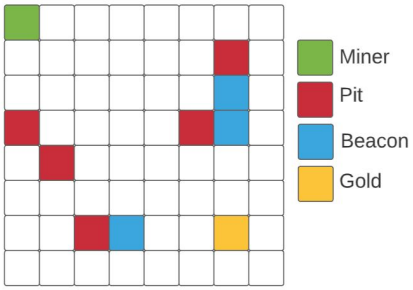

Fig 1. Sample initial board state

Based on Figure 1: (1) the agent is at coordinates (1, 1), (2) pits can be found at (4, 1), (5, 2), (7, 3), (4, 6), and (2, 7), (3) beacons are at (7, 4), (4, 7), and (3, 7), and (4) the gold is at (7, 7).

If the smart agent's algorithm is to be applied to the sample board state, the resulting path towards the goal state is shown in Figure 2.

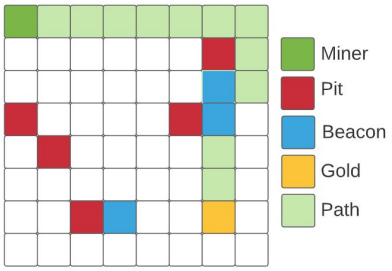

Fig 2. Achieved goal state from smart agent's path

From coordinates (1, 1) to (1, 8), it is observed that the smart agent continues to move at its current direction (right) as no gold or beacon were found per scan of each move. Previous directions were not rotated to and scanned as well, as they were already visited according to the list of visited tiles. On (1, 8), the only valid direction to rotate and scan with was the bottom direction as analyzed, therefore the agent rotated down and moved. On (3, 8), a beacon was found on the left of the agent therefore it was targeted, then the agent moved in that direction. With the given case of when the agent lands on a beacon, the beacon returns a distance of $d > 0$ as there is a gold tile below it. The returned distance was used, so the agent was able to target the gold tile immediately. It moved towards it directly without the need for rotating and scanning anymore the other directions with its corresponding unvisited tiles. The goal state was then reached.

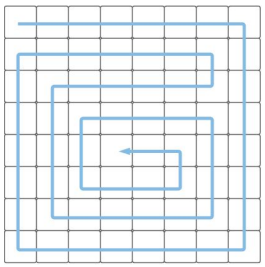

Fig 3. Smart agent's resulted path from empty board state

With the algorithm for the smart agent, if given an empty board state as seen on Figure 3 above, the smart agent will do a spiral traversal throughout the board starting from the outer edges towards the center. This is to make sure that for some edge cases - such as if the gold tile is located at the outer edges of the board with size $n > 30$, it will be covered and reached efficiently by the smart agent.

Performance of the smart agent's algorithm:

Cost-wise, the smart agent's algorithm requires a lot of rotations and scans in order for it to successfully determine which among the available moves will it benefit from the most in reaching the goal state, but it is always guaranteed that the number of moves made are small and acceptable enough for any board size from the given range $8 \leq n \leq 64$.

Appendix

(x is the current row position, y is the current column position)

Name	Condition	Transition	Effect
Move	In current direction: Up: x > 0 Down: x < N Left: y > 0 Right: y < N	In current direction: Up: (x, y) -> (x-1, y) Down: (x, y) -> (x+1, y) Left: (x, y) -> (x, y-1) Right: (x, y) -> (x, y+1)	Moves to another tile based on current direction
Rotate	---	Current direction -> chosen direction	Rotates to the chosen direction
Scan	In current direction: Up: x > 0 Down: x < N Left: y > 0 Right: y < N	In current direction: Up: Scan (x-1, y) Down: Scan (x+1, y) Left: Scan (x, y-1) Right: Scan (x, y+1)	Scans and returns the object found
Move Random Miner [Random agent only]	---	Based on number generated: <= 33 -> Move > 33 && <= 66 -> Rotate > 66 -> Scan	Executes basic action (Move, Rotate, Scan) based on random number generated.
Set Object Scanned [Smart agent only]	While targeted object is not gold, scan all valid directions	If scanned object is gold -> target object and direction immediately If scanned object is beacon and no target object yet -> target beacon and direction If scanned object is null (no object) and no target yet -> target direction	Scans all valid directions to target and move to the most beneficial tile to reach goal state.
Set Gold Target [Smart agent only]	When current tile position contains beacon and returned distance value of beacon > 0	For each valid direction: Up: Scan (x-distance, y) Down: Scan (x+distance, y) Left: Scan (x, y-distance) Right: Scan (x, y+distance) If scanned object is gold -> target object and direction immediately	Sets the target object and direction to the goal state (gold) based on the beacon's distance value.
Backtrack [Smart agent only]	If size of backtrack stack is > 0	Current tile position -> previous tile position	Backtracks to the previous tile position

Transition Table