# Stacker Class Documentation

**Class: Stacker**

**Overview**

The **Stacker** class is designed for a ladder game where an AI agent moves through a grid, handling obstacles, picking up blocks, and searching for gold. The class includes various methods for pathfinding, decision-making, and map exploration, tailored to the specific mechanics of the game.

**Constructor**

* **constructor(agentX: number, agentY: number)**
  + Initializes the **Stacker** instance with the starting position of the agent.
  + **Parameters**:
    - **agentX**: X-coordinate of the agent's starting position.
    - **agentY**: Y-coordinate of the agent's starting position.

**Properties**

* **EMPTY**, **WALL**, **BLOCK**, **GOLD**: Constants representing different types of cells on the game map.
* **carryingBlock**: Boolean indicating whether the agent is currently carrying a block.
* **currentLevel**: The current elevation level of the agent.
* **currentPos**: The current position of the agent.
* **goldPos**: The position of the gold, if known.

**Methods**

**findPathToTarget(start, goal, gameMap)**

* A\* pathfinding algorithm to find the shortest path from the start to the goal.
* **Parameters**:
  + **start**: The starting position.
  + **goal**: The goal position.
  + **gameMap**: The game map grid.
* **Returns**: An array of coordinates representing the path from start to goal.

**isVisible(start, end, gameMap)**

* Implements Bresenham's line-of-sight algorithm to determine if there is a clear line of sight between two points.
* **Parameters**:
  + **start**: The starting coordinate.
  + **end**: The ending coordinate.
  + **gameMap**: The game map grid.
* **Returns**: **true** if there is a clear path, **false** otherwise.

**updateGoldPosition(currentPos, gameMap)**

* Updates the known position of the gold if it's within visibility range.
* **Parameters**:
  + **currentPos**: The current position of the agent.
  + **gameMap**: The game map grid.

**decideNextMove(cell, gameMap)**

* Decision-making function that determines the next move based on the current state.
* **Parameters**:
  + **cell**: The current cell information.
  + **gameMap**: The game map grid.
* **Returns**: A string representing the next move (**"left"**, **"right"**, **"up"**, **"down"**, **"pickup"**, or **"drop"**).

**findNearestBlock(gameMap)**

* Uses BFS to find the closest block to the agent.
* **Parameters**:
  + **gameMap**: The game map grid.
* **Returns**: The coordinates of the nearest block or **null** if none are found.

**explore(gameMap, currentPos)**

* Explores the game map to find unexplored areas or new targets.
* **Parameters**:
  + **gameMap**: The game map grid.
  + **currentPos**: The current position of the agent.
* **Returns**: The next move direction based on exploration.

**getNextMoveFromPath(pathToTarget, currentPos)**

* Translates the next step in a path to a move action.
* **Parameters**:
  + **pathToTarget**: The path to follow.
  + **currentPos**: The current position of the agent.
* **Returns**: A string representing the next move.

**shouldPickupBlock(cell)**

* Checks if there's an adjacent block that the agent should pick up.
* **Parameters**:
  + **cell**: The current cell information.
* **Returns**: **true** if the agent should pick up an adjacent block, **false** otherwise.