**Design Defense: Deep Q-Learning Pirate Agent**

**Human vs. Machine Problem Solving**

People typically solve mazes with a mix of perception, heuristics, and short-term planning. A person glances over the layout, spots corridors and dead ends, and then uses simple rules: “favor open corridors,” “avoid loops,” “work toward the bottom-right,” augmented by backtracking when a path fails. This combines a little look-ahead with a mental map that updates as progress is made.

The pirate agent, in contrast, does not plan with an internal map. It learns a value function Q(s,a) that estimates the long-term return from taking action a in state s. Those estimates are improved incrementally from experience using temporal-difference updates and the Bellman optimality target. Exploration is injected stochastically rather than guided by visual intuition (Sutton & Barto, 2018; Watkins & Dayan, 1992). In short, humans rely on structured reasoning and visual heuristics, whereas the agent relies on trial-and-error learning with credit assignment.

**How a Human Would Solve This Maze**

1. Scan the grid. Identify the goal at the bottom-right, notice blocked cells, and look for promising channels.
2. Pick a corridor. Start from the pirate’s cell and move along contiguous free cells that trend toward the goal.
3. Avoid revisits. Keep a mental set of visited cells to prevent loops; mark or remember junctions.
4. Backtrack if needed. When a dead end is reached, return to the last junction and try an alternative.
5. Iterate the goal. Repeat until reaching the treasure. This process is essentially depth-first search with heuristics; a careful solver might mimic breadth-first search or A\* if they mentally keep a frontier (Hart, Nilsson, & Raphael, 1968).

**What the Agent Actually Does**

The pirate agent uses deep Q-learning with experience replay. The 8×8 maze is flattened into a 64-value vector that encodes walls as 0.0, free cells as 1.0, plus marks for the pirate’s current cell and whether a cell has been visited. The action space has four moves left: up, right, down, and left, and the environment penalizes illegal choices such as stepping off the grid or into a wall. Rewards are shaped to favor short, clean routes: +1.0 for reaching the treasure, −0.04 per step, −0.25 for revisiting a cell, −0.75 for an invalid move, and early termination if total reward drops below a floor.

Policy selection is ε-greedy: with probability ε the agent explores by sampling a random valid move; otherwise it exploits by choosing the action with the highest current Q(s, a). During evaluation, invalid actions are masked so exploitation never selects an illegal move. Learning happens from tuples (s, a, r, s′, done) stored in a replay buffer; targets follow the Bellman update use r if the episode ended, or r + γ·maxₐ′ Q(s′, a′) otherwise. Mini-batches drawn from replay break up correlations and improve sample efficiency. The function approximator is a small MLP: input size 64, two Dense layers of 64 units with PReLU activations, and a 4-unit output for the action values, trained with Adam and mean-squared error. To keep CPU time reasonable, training waits for a short warm-up to fill the buffer, performs an update every N steps (for example, train\_interval = 8) with moderate batch sizes (around 32), and uses a brief curriculum that starts episodes at (0, 0) before switching to random free cells.

**Similarities and Differences**

Similarities. Both approaches improve with experience, avoid revisiting, and adapt when a path fails. Both balance trying what seems best versus trying something new.

Differences. The human relies on global structure and causal reasoning; the agent aggregates experience into scalar Q-values without an explicit map. Humans explore purposefully (e.g., “peek down that corridor”), whereas the DQN explores randomly via ε-greedy. Credit assignment for a human is immediate (“that turn boxed me in”); for the agent it propagates back through Bellman targets across many updates (Sutton & Barto, 2018).

**Purpose of the Intelligent Agent in Pathfinding**

The pirate agent supplies a general, adaptive opponent that can learn the maze dynamics from interactions rather than coded rules. That reduces game-specific hard-coding, allows quick retuning for new levels, and provides variability: even on the same map, exploration causes non-deterministic behavior that feels less scripted.

**Exploration vs. Exploitation**

Exploitation chooses the action with the highest current Q-estimate; exploration chooses a non-greedy action to gather new information. In practice, decaying ε from a higher value to a small floor yields a GLIE-style schedule greedy in the limit with infinite exploration (Sutton & Barto, 2018).

For this small, deterministic maze, a simple effective schedule is:

* Warm-up curriculum: ε≈0.6 for ~25 epochs from (0,0).
* Broadened training: ε≈0.3 decaying toward 0.05.

This maintains enough exploration to escape early suboptimal policies while converging toward stable greedy behavior. Too little exploration stalls; too much keeps the policy noisy (Mnih et al., 2015).

**Why Reinforcement Learning Works Here**

Reinforcement learning turns the sequential decision problem into value estimation. Rewards provide a shaping signal that makes shorter, legal paths more valuable than meandering or illegal movement. Temporal-difference learning propagates the “good news” of reaching the treasure back to earlier states, so the agent begins to prefer action sequences that line up corridors leading to the goal (Sutton & Barto, 2018). Experience replay amortizes learning by reusing past transitions, which is critical for data efficiency on CPU (Lin, 1992; Mnih et al., 2015).

**Algorithm Choice and Trade-offs**

Classical search algorithms, BFS or A\*, can guarantee a shortest path on a known static grid. A\* with an admissible heuristic (e.g., Manhattan distance) is optimal and typically much faster to compute a single path (Hart et al., 1968). However, those methods require an explicit model and do not learn; each new map restarts the search from scratch.

A DQN, by contrast, learns a policy that generalizes across starting cells and continues to improve with experience. That makes it a good fit for NPC behavior that must be learned rather than scripted. The trade-offs are training time, stability, and the need for careful engineering. Our minimalist DQN omits common stabilizers, target networks, Double DQN, and prioritized replay, which would usually speed convergence and reduce overestimation bias (Mnih et al., 2015; Hasselt, 2010; Schaul et al., 2016). These are clear avenues for future improvement.

**Implementation Summary: Deep Q-Learning for the Pirate**

* Model. MLP(64→64→64→4) with PReLU, Adam, MSE.
* State. Flattened grid with current pirate cell marked; visited set used for penalties and visualization.
* Rewards. +1 at treasure; −0.04 per step; −0.25 revisits; −0.75 invalid; episode loss if total reward drops below a floor.
* Policy. ε-greedy with action masking at evaluation to avoid picking illegal actions by mistake.
* Replay. FIFO buffer (e.g., 2,000 transitions). Mini-batches sampled to compute Bellman targets r+γmaxa′​Q(s′,a′).
* Training loop.
  1. Reset to a start cell (first curriculum from (0,0), then random free cells).
  2. Roll out until win/lose or step cap, storing (s,a,r,s′,done).
  3. Every train\_interval steps after warmup, sample a batch and fit.
  4. Decay ε, log win rate, and continue.
* Evaluation. Two checks: (a) a completion check that tries every free start cell, and (b) a playthrough from (0,0). Training stops early if the rolling window reaches 100% and the completion check passes.

Within the constraints of CPU-only training and a simple network, this design balances speed and learning stability. The agent reliably improves its win rate on the provided maze and, with additional stabilizers (target network, Double DQN), can be pushed to faster and more consistent 100% completion.

**References**

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136

Hasselt, H. V. (2010). Double Q-learning. *Advances in Neural Information Processing Systems*, 23, 2613–2621.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning, 8*(3–4), 293–321. https://doi.org/10.1007/BF00992699

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533. https://doi.org/10.1038/nature14236

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. *International Conference on Learning Representations (ICLR)*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3–4), 279–292. https://doi.org/10.1007/BF00992698