**Deep Q-Learning for Maze Pathfinding: Human vs Machine Approaches**

Human problem-solving and machine learning approaches to tasks like maze navigation differ fundamentally. Humans solve mazes using cognitive maps and reasoning, whereas a learning agent uses computational algorithms to optimize a reward function. Cognitive science suggests that humans build an internal “cognitive map” of the spatial environment to guide navigation (Epstein et al., 2017). In contrast, a deep Q-learning agent relies on a neural network to estimate action values (Q-values) and learns a policy through trial-and-error interaction with the environment (Mnih et al., 2013). The human approach typically involves explicit reasoning, memory, and heuristics (e.g. recognizing patterns or dead ends, planning shortcuts), while the machine approach uses statistical function approximation and algorithmic exploration/exploitation balance. Because deep reinforcement learning (DRL) integrates deep neural networks with reinforcement learning, it can handle high-dimensional inputs and learn complex strategies without explicit programming of rules.

**Human vs Deep-Q Agent Maze-Solving Strategy**

When a person tries to find a treasure in a maze, they typically use a combination of visual perception, memory, and planning. For example, humans may form an internal map of the maze (landmarks, corridors, and dead ends) and plan a route to the treasure (Epstein et al., 2017). Empirical studies indicate that the human hippocampus and associated brain areas support map-like spatial codes, which are used together with frontal lobe mechanisms to plan routes during navigation (Epstein et al., 2017). In practice, a human pirate exploring the treasure-hunt maze might first scan visible passages, remember choices made, and apply strategies like always keeping one hand on a wall or backtracking when encountering dead ends. The human may recognize patterns (e.g. loops or symmetry) and use logical deduction (for instance, if one path leads to a dead end, try a different direction). Throughout this process, the human actively forms expectations about how far certain corridors lead and updates their plan when new information is obtained.

The machine agent (a deep Q-learning pirate) follows a different procedure based on reinforcement learning. Formally, this agent observes the state of the maze (for example, the locations of walls, the agent’s position, and the treasure) and feeds a numerical representation of that state into a neural network. The network (implemented as a Keras Sequential model with Dense layers, PReLU activations, etc.) outputs Q-values estimating the expected cumulative reward for each possible action (move up/down/left/right). At each time step, the agent selects an action. It explores by choosing a random action, otherwise it exploits by choosing the action with highest estimated Q-value (Xian et al., 2025). After executing the action, the agent receives a reward (e.g. positive if closer to treasure, negative for hitting a wall or taking a step) and transitions to a new state. These transitions (state, action, reward, next state) are stored in an experience replay memory. Periodically, the agent samples mini-batches from this memory and trains the neural network: it uses the Bellman equation to compute target Q-values (reward plus discounted max next-state Q) and fits the network weights to minimize prediction error. A separate target network (a periodically updated copy of the main network) is often used to stabilize learning (Xian et al., 2025). Over many episodes with decaying epsilon (e.g. annealing from 0.95 to 0.01 as in some studies by Xian et al, the agent gradually learns which sequences of moves yield the highest cumulative reward, effectively discovering the optimal path to the treasure.

Both humans and the deep Q-learning agent ultimately aim to find a route to the treasure, but they approach the task differently. A key similarity is that both can learn from experience: a human may remember which turns worked or failed, and the agent updates its Q-value estimates based on previous trial outcomes. Both may backtrack when encountering dead ends. However, the strategies differ in mechanism. Humans rely on conscious spatial reasoning and prior knowledge (for example, innate sense of direction or learned maze-solving tactics), whereas the agent uses statistical learning and optimization. Humans do not explicitly compute value functions or update equations; instead, they form a mental spatial representation and make decisions based on heuristic judgments and exploration. The machine, conversely, has no inherent understanding of space - it simply processes input features through the network to produce numeric Q-values. For example, Epstein et al. describe how animals (including humans) navigate using spatial codes in the brain, a process akin to a cognitive map, while a Q-learning agent represents “state-value” information implicitly in network weights.

In terms of outcome, both approaches aim for efficiency: humans often intuitively minimize path length when possible, and the agent learns to maximize expected reward (which may correlate with shortest path). Yet the agent may find non-intuitive paths if its reward structure or exploration yields them. The agent can also quantitatively refine its policy until convergence, whereas a human may settle on a “good enough” path.

**Exploration vs. Exploitation**

In reinforcement learning, *exploration* refers to trying new or random actions to discover their effects, while *exploitation* means choosing the best-known action to maximize reward. These must be balanced: too much exploration wastes time on suboptimal moves, too little may trap the agent in a local optimum. The epsilon-greedy strategy embodies this balance by selecting a random action with probability epsilon (exploration) and the highest-Q action with probability 1–epsilon (exploitation). In the treasure-maze context, ideal behavior is to start with high exploration (large epsilon) to map out the maze’s possibilities, then gradually reduce epsilon to consolidate the learned path. For example, practical implementations anneal epsilon from about 0.9–1.0 down to near 0.01 over training (Xian et al., 2025). The exact schedule depends on maze complexity: a very large, random maze might require more exploration to find the treasure, whereas a small maze could use less. A reasonable heuristic is to allow exploration (~20-30% of moves) in early training to discover reward signals, then shift to exploitation (~80-90% greedy) once an effective route has been identified. This balance is justified by the fact that the environment is stochastic each episode; some ongoing exploration (a small epsilon) must remain so the agent can adapt if the maze changes. Ultimately, the agent’s learning curve (see RL literature) is improved by a decaying epsilon schedule that transitions from exploration to exploitation as confidence in the value estimates grows (Xian et al., 2025).

Reinforcement learning formalizes the search for an optimal path via the maximization of expected cumulative reward. The central idea is the Bellman optimality principle: the value of an action in a state equals the immediate reward plus the best possible future value. Over many episodes, the learned Q-function converges (under ideal conditions) to the true optimal action values. In practical terms, each episode of the maze teaches the agent how good each move is, and the deep Q-network eventually encodes the optimal policy. This process helps determine the shortest (or most rewarding) path: if the reward is structured as negative cost for each step plus a large positive treasure reward, then maximizing cumulative reward encourages shorter routes to the goal.

**Algorithmic Solutions vs. Learning**

Traditional algorithms can also solve the maze pathfinding problem, offering a contrast to the learning-based approach. For example, the A\* search algorithm is a well-known graph traversal method that is *complete* and *optimal* under admissible heuristics, meaning it will find a shortest path if one exists. In a fully known maze, A\* or Dijkstra’s algorithm could efficiently compute the treasure path by systematically exploring grid nodes. These algorithms require knowledge of the maze graph and typically use a heuristic (like straight-line distance) to guide search. In contrast, the RL agent does not use any explicit heuristic or model. It learns indirectly which neighbors lead closer to the goal through reward feedback. Sutton and Barto note that a key difference is that RL algorithms do not assume knowledge of the exact environment model and can handle very large state spaces by function approximation. However, classical algorithms often guarantee an exact shortest path (if the problem size permits), whereas RL finds a policy that maximizes reward and may not explore all possible variations if it already finds a good solution. In summary, A\* and similar methods are algorithmic, model-based approaches that solve pathfinding through search, while the deep Q-learning agent is model-free and learns a policy through experience. Both are valid for maze solving, but they represent fundamentally different paradigms: one is planning with a global view, the other is learning from local feedback.

**Deep Q-Learning Implementation Details**

At the implementation level, deep Q-learning uses a neural network to estimate the Q-function. In our Keras setup, the Sequential model takes a representation of the maze state and feeds it through several Dense layers with non-linear activations (in this case PReLU, which is a parametric version of ReLU allowing a small learned negative slope) (*PReLU — PyTorch 2.7 Documentation*, n.d.). The final layer outputs one Q-value per possible action. During training, given a state, the network’s forward pass produces Q-values for each action. Action selection then uses the epsilon-greedy policy described above (often with epsilon decayed over time) (Xian et al., 2025).

When a batch of experience tuples is sampled from replay memory, the agent computes target values depending on whether it is a terminal or non-terminal state. The network is trained (via backpropagation with mean-squared error loss) to reduce the difference between its current estimate and the target. Over many updates, the network’s weights adjust so converges toward the true expected return of taking said action in said state.

Experience replay and the target network are crucial: replay breaks correlations by randomizing experience order, and the target network stabilizes learning by providing fixed targets for n steps (Xian et al., 2025). The target network might be updated every few hundred or thousand steps. Hyperparameters such as learning rate, discount factor (commonly <1 to prioritize near-term rewards), and optimizer (e.g. Adam) are tuned to ensure stable convergence. In summary, the deep Q-learning architecture uses the neural network to approximate the Q-values, observes states via the game’s grid input, selects actions epsilon-greedily, and updates the model by fitting to Bellman targets computed from stored experiences (Mnih et al., 2013, Xian et al., 2025).

**References**

Epstein, R. A., Patai, E. Z., Julian, J. B., & Spiers, H. J. (2017). The cognitive map in humans: spatial navigation and beyond. *Nature Neuroscience*, *20*(11), 1504–1513. <https://doi.org/10.1038/nn.4656>

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013, December 19). *Playing Atari with Deep Reinforcement Learning*. arXiv.org. <https://arxiv.org/abs/1312.5602>

Xian, Y., Ding, X., Jiang, X., Zhou, Y., Sun, J., Xue, D., & Lookman, T. (2025). Unlocking the black box beyond Bayesian global optimization for materials design using reinforcement learning. *Npj Computational Materials*, *11*(1). <https://doi.org/10.1038/s41524-025-01639-w>

*Sutton & Barto Book: Reinforcement Learning: An Introduction*. (2018). <http://incompleteideas.net/sutton/book/the-book.html>

*PReLU — PyTorch 2.7 documentation*. (n.d.). https://docs.pytorch.org/docs/stable/generated/torch.nn.PReLU.html