Reinforcement learning on Minesweeper

Group Members: Junhao Chen, Franklin Pu, Kabir Maharjan

This project explores how reinforcement learning can help an AI learn to play Minesweeper on its own. In Minesweeper, the player must reveal safe squares and mark mines based on the numbers shown on the board. The game involves hidden information, some unavoidable guessing, and decisions that can affect the game many steps later. Even though the rules are simple, the number of possible board states is huge, and the reasoning needed can be very complex, which is why Minesweeper has long been a challenging problem in AI research. In this project, we build our own Minesweeper environment and train agents using Q-Learning and a Deep Q-Network (DQN) to study how well these methods handle such a difficult reasoning task and what limitations they run into.

The project uses a complete Python Minesweeper implementation as its base environment. The game board is 8×8 and contains 10 mines which can be adjusted in Settings.py file (defined in Settings.py). Each cell is represented by a “Tile” object, containing state information such as cell type (space "/", number "C", mine "X", unknown "."), whether it is revealed, and whether it has a flag. The Board class is responsible for generating mines, calculating numerical hints, and handling the player's or agent's mining behavior, including connecting empty areas (recursive expansion). The Game class is used for the interaction mode of a normal player, while the core environment of reinforcement learning is encapsulated by the Env_Minesweeper class, which provides interfaces such as reset, step, state encoding, and win/loss determination, thus providing the agent with a more rigorous and structured interaction mode than human players. When we were making the Minesweeper Environment, because of laziness, we did not achieve the mechanic that all the mines should be placed AFTER the player’s first move so that the player can have an acceptable start. This might also affect the AI training output, making it difficult to have good output.

Minesweeper is really a partially observable decision problem. The agent can only see the squares that have already been revealed (either empty or a clue), while unrevealed squares might be mines or might contain important information. It cannot calculate every possibility for next move to optimize the reward and make decision. The size of the state space also grows exponentially with the board size, and for a 15×15 board it becomes almost impossible to handle.

A major challenge is that most actions do not give clear or immediate rewards. The agent only receives strong feedback when it hits a mine or reveals a large area of the board. This makes the learning process very noisy and hard to stabilize. Minesweeper also has randomness built in—for example, the classic "50/50 guessing" situations—meaning that even human experts can't always avoid mistakes. Because of all these factors, Minesweeper is a difficult environment for reinforcement learning.

To let the agent learn in this environment, we first implemented Q-Learning. However, because the state space is so large and a full Q-table cannot be stored, Q-Learning is only used as a baseline in our project. The action space is simplified to integers from 0 to 63, each representing a square on the board. If a square has not been revealed, the only action available is to dig it. If the square is already revealed, the environment immediately gives a small penalty of -0.1 (as in Env_Minesweeper.step) to discourage repeated invalid moves. This setup keeps the action space simple and allows the agent to focus on the main question: "which square should I dig next?"

To further handle high-dimensional state spaces, we tried to implement a Deep Q-Network (DQN). In the code (DQN_Agent.py), the input to DQN is an 8×8 integer matrix, which is constructed using Env_Minesweeper._get_observation().

- -1 indicates that the square has not been flipped.
- -2 indicates planting a flag (although the agent can only dig it up at present).
- 0 represents a blank tile.
- 1–8 indicate clue of how many mines are around it.
- 9 indicates a mine that has been revealed (in case of failure).

The network structure is a three-layer fully connected feedforward network (Flatten → 256 → 256 → 64), where each output corresponds to the Q-value of an action. Exploration employs an ε-greedy strategy, with ε gradually decreasing from 1.0 to 0.05 to balance random exploration and policy exploitation. The experience replay pool size is 50,000, and training is performed using random sampling with a batch size of 64; the target network is updated every 20 episodes (DQN_Agent.py).

The reward function is a crucial component of this project and directly determines the agent's learning behavior. Based on the implementation of Env_Minesweeper.step, the reward structure is as follows:

1. Repeatedly click on a revealed tile: reward = -0.1

   The goal is to punish meaningless actions.
2. Digging out a mine (Board.dig returns False): reward = -10, and the turn ends immediately.

   This penalty is severe and is intended to prevent the agent from blindly clicking. This is hard to set a good reward for this case. Although, we all know that in Minesweeper, when the player digging out a mine, game over. But the overall points might differ based on how many tiles you revealed already.
3. Successfully found a safe cell (empty "/" or clue "C"): reward = +1

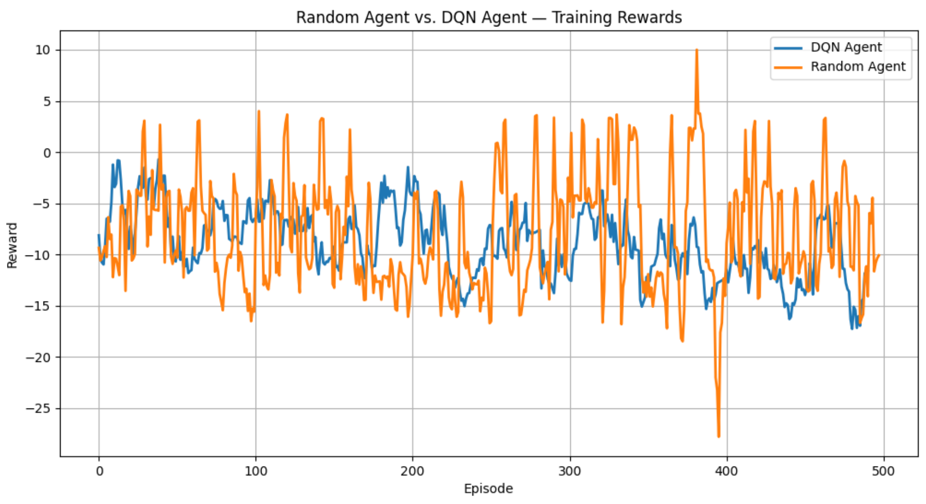
   The reward is equal regardless of whether the cell truly provides a large amount of information (such as multiple blanks) or is merely a single numerical prompt.
4. Successfully complete the entire chessboard (_check_win() == True): reward = +10, and end the turn.

Compared to many Minesweeper AIs, this reward structure is relatively simplified and does not reflect important strategic factors in Minesweeper such as "information gain" (e.g., revealing a 0 zone should be more critical than digging up a single digit) and "reducing uncertainty." Because the reward design is entirely based on "results" rather than "the magnitude of information value," the agent may be unable to infer which areas are high-value mining points from the reward signal. This is particularly evident in subsequent experimental results.

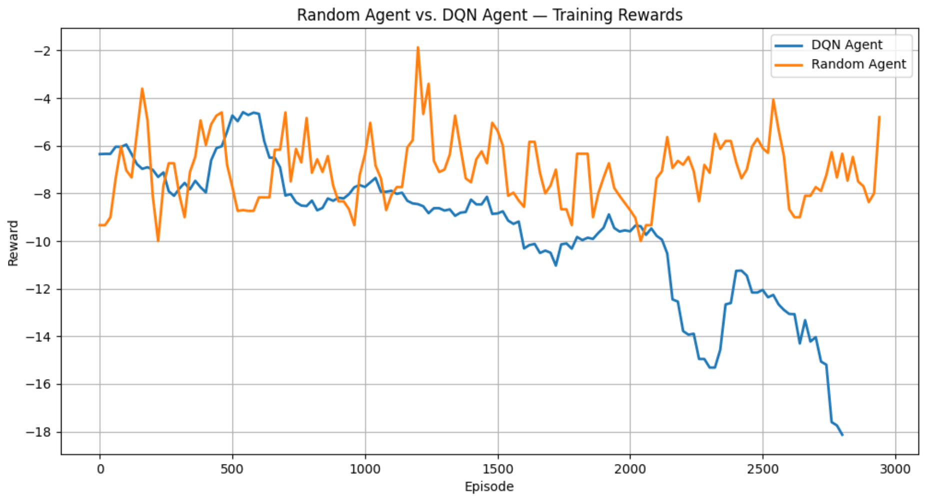
In our experiments, we compared the random policy with a DQN agent at two different difficulty levels: 15×15 (25 mines) and 8×8 (10 mines). In the 15×15 setting, the agent almost never obtained positive rewards, no matter how many training iterations we ran. The reward curve was dominated by large negative spikes, showing extremely high noise and no meaningful improvement, which made it clear that the environment was simply too hard for the model to learn anything useful. Because of this, we reduced the difficulty to an 8×8 board and trained the agent for 500 and 3000 episodes. After 500 episodes, DQN performed only slightly better than the random policy, and the curve still showed strong fluctuations without a clear positive trend. When we extended training to 3000 episodes, the issue became even more obvious: instead of converging, DQN's rewards gradually decreased over time and eventually fell below the random agent. This suggests that with our current network size, simplified observation space, and reward structure, DQN is unable to discover a stable long-term strategy. Instead, it tends to overfit to short-term patterns or misleading correlations in the environment, causing its behavior to degrade the longer it trains. This pattern is consistent with DQN's known weaknesses in environments that are highly stochastic, partially observable, or contain sparse and delayed rewards, all of which are characteristics of Minesweeper.

Graphs trained on 8*8 board and 10 mines:

500 episodes (plot every episode)



3000 episodes (plot every 20 episodes)



After analyzing the code logic, we believe this stems primarily from two points: First, the Minesweeper environment itself is partially observable. Secondly, the reward signal is too vague and lacks guidance for the inference chain. For example, digging up a "0 area" that provides a wealth of information only yields +1, which is no different from digging up a "3" or "4" cell with very little value; the agent cannot distinguish between "high-value mining" and "low-value mining" based on the reward. Furthermore, the frequent occurrence of the -10 penalty for stepping on a mine in high-density areas leads to the policy learning being dominated by negative rewards for a long time, making it difficult for the model to find a stable direction for improvement. Ultimately, the decline in rewards and policy degradation in DQN are to be expected.

Therefore, this project still has several limitations, including the enormous size of the state space, the difficulty of inference due to partial observability, the inadequacy of the reward design, and the relatively basic DQN network structure, which does not incorporate modern reinforcement learning techniques such as Double DQN, Dueling DQN, and priority experience replay. These factors collectively limit the final performance of the model.

Future improvements could include: designing reward functions with more Minesweeper semantics (e.g., based on information gain, reducing uncertainty, weighted rewards for consecutive open regions), using stronger deep network structures, gradually increasing the difficulty of the environment through a curriculum-based learning approach, and attempting to combine symbolic logic reasoning methods with deep reinforcement learning to compensate for DQN's lack of advanced logical reasoning capabilities.