Designing Intelligent Agents Project

Conrad Mwaura

May 9, 2025

1 Introduction

Minesweeper is a logic-based puzzle game where players uncover tiles on a grid, trying to avoid hidden mines and using numerical clues to deduce their locations. The goal is to clear all non-mine tiles without triggering any mines. The question is **"How do different models compare in accuracy and efficiency when solving the game of Minesweeper?"**

2 Literature Review

The difficulty of applying reinforcement learning to Minesweeper is due to its large state and action space. Minesweeper can be framed as a constraint satisfaction problem [4], where the goal is to deduce safe moves by satisfying logical constraints imposed by revealed numbers. However, these constraints are often ambiguous, forcing the agent to make probabilistic decisions rather than purely logical inferences. These challenges impact the decision-making and learning process of the agents due to hidden information and inter-cell dependencies; the agent must be able to gather and process the observed board state as logical constraints to identify safe moves, often with incomplete information. Another challenge comes from delayed rewards; the agent has to complete entire episodes to evaluate the value of early actions, resulting in slower convergence and computational cost [5]. Also, the biggest reward (win or loss) is given at the end of the game, making it difficult to interpret what actions were positive, which may result in the agent learning suboptimal moves.

A paper in Stanford CSS29 researched different models and how various factors affected the agents' performance [3]. These models included Simplified Q-Learning, Global Probability Regression, and Local Classification. Their results showed that board size and mine density significantly affect the model's performance during testing and training, attributed to the increase in board state permutations. A higher board size and mine density increase the game's complexity, leading to more frequent guessing, reduced accuracy, and longer training times for the agent to learn effective strategies.

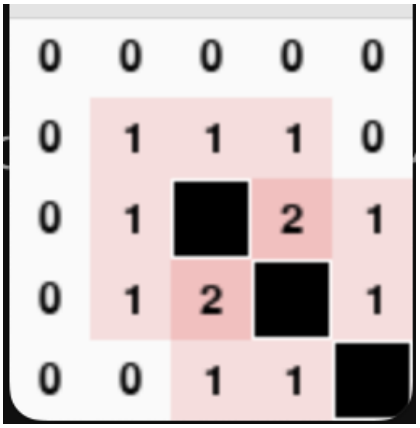
A wide range of results was observed from different models. Models trained over several hundred thousand episodes showed high results, such as a Deep

Q-Learning model achieving 16.87% win rate on an 8×8 intermediate game [2]. When hardware limitations are present, the number of training episodes must decrease to remain feasible. This reduction in training opportunities requires changes to the model architecture, such as simplifying network depth, modifying exploration strategies, or balancing efficiency over effectiveness. Under similar computational constraints, a more efficient model achieved a higher win rate of 45% on a smaller 5×5 grid, illustrating how performance is tightly coupled not only to model design, but also to the scale of the game environment and the available computational budget.

3 Methodology

3.1 Environment

The agents need to use an environment to interact. The program uses an OpenAI gym environment for Minesweeper found online [6], simulating the classic minesweeper game for reinforcement learning. It was chosen because of the observation space representation, simplicity of code, and GUI design.



(a) GUI version of observation space

```
Action: (3, 4), Reward: 1
[[ 0  0  0  0  0]
 [ 0  1  1  1  0]
 [ 0  1 -1  2  1]
 [ 0  1  2 -1  1]
 [ 0  0  1  1 -1]]
```

(b) 2D array of observation

Figure 1: Example of how the 2D Array observation space translates as a GUI

It has an observation space of a 2D grid (as seen in Figure 1), where an agent can interact with individual tiles using an (x,y) coordinate. After an action has been taken, the environment returns a reward that has to be evaluated. An unrevealed tile is -1, a revealed tile is represented by the number of bombs surrounding it.

The environment's implementation initially faced a challenge due to the pygame library, which handles the rendering and graphical interface. Pygame's screen could not be directly copied, requiring the creation of a new method to handle deep copies of the game states without copying the GUI, where only

necessary data was copied. Additionally, the option to reveal all surrounding tiles when a tile with zero bombs is uncovered was made optional, providing the flexibility to disable this feature and control the degree of exploration during the agent's training (see 3.2)

3.2 Monte Carlo Prediction

One of our models uses Monte Carlo to predict the tile to reveal. Monte Carlo methods refer to a class of algorithms for decision-making in Markov Decision Processes (MDPs) that rely on sampling-based evaluation of future outcomes rather than exact computation. In particular, they estimate the value of a decision node by recursively simulating future actions (rollouts) and propagating average returns up the decision tree [1].

Algorithm 1 Monte Carlo Training for Minesweeper (3×3)

```
Initialize empty dictionary to store state-action rewards
for each mine count from 1 to 8 do
    Create a 3×3 Minesweeper environment with the given mine count
    for each episode do
        Reset the environment and get the initial observation
        while game is not over do
            Find all valid (unopened) moves
            for each valid move do
                Simulate the move in a copy of the environment
                Record the resulting reward
            end for
            Choose best move (or random with some probability)
            Apply the move to the real environment
            Update reward estimates for the current state
        end while
    end for
end for
```

A pseudocode version of the algorithm training is shown above. The agent repeatedly plays in a 3×3 Minesweeper environment with different mine densities. For each unique board state, it simulates all valid moves using a copy of the environment to estimate the expected reward. The reward values are stored in the dictionary associated with the board states, allowing for quick access during decision making. An ϵ-greedy strategy balanced exploration and exploitation during training to avoid overfitting to earlier experiences. This process allows the agent to encounter and evaluate almost all possible 3×3 board configurations and have an action associated with the highest reward. As a result, a comprehensive policy is formed in the form of a lookup table, where each board state maps directly to an optimal or near-optimal action based on empirical experience.

A 3×3 board was chosen to reduce the state space, instead of a variable board size. By restricting training, the grid becomes small enough to explore its state space almost exhaustively, allowing the agent to see and learn from nearly every possible state. Furthermore, a surrounding tile is only affected by its immediate 8-cell mine neighbourhood, making local reasoning sufficient for many decisions. When the model is running on a test grid, it will sequentially look through each valid move and evaluate the reward repeatedly until all moves have been looked at.

The option to automatically reveal all surrounding tiles when a zero is uncovered was enabled in this setup. While this behavior aligns with how traditional Minesweeper is played, it introduced a limitation during training: not all possible board permutations were being explored. Since multiple tiles could be revealed in a single action, the agent was exposed to fewer unique state transitions, potentially reducing the diversity of experience collected. This may have limited the agent's ability to learn optimal responses for less frequently encountered configurations.

3.3 Deep Q-Learning

4 Evaluation

To understand how well an agent has performed, three performance metrics were used: win rate, the average reward per episode, and the average percentage of the board revealed. Win rate alone is not a sufficient metric, as an agent might lose most games yet still demonstrate intelligent decision-making. Winning requires the agent to make several correct moves in a row, some of which may involve unavoidable guessing due to incomplete information.

5 Bibliography

References

[1] M. C. Fu. "A tutorial introduction to Monte Carlo Tree Search". In: *2020 Winter Simulation Conference (WSC)*. 2020.

[2] *Github repository for AI Minesweeper with Q Learning and Deep Q Learning by ChunTheBear.*

[3] Ryan Silva Luis Gardea Griffin Koontz. "Training a Minesweeper Solver". In: *CS 229*. 2015.

[4] A. Mehta. *Reinforcement Learning For Constraint Satisfaction Game Agents*. 2020.

[5] Muhammad Hamza Sajjad. "Neural Network Learner for Minesweeper". In: *Loughborough University*. 2022.

[6] Jeffrey Yao. *gym-minesweeper: OpenAI Gym environment for Minesweeper*. Accessed: 2025-05-09. 2019. URL: `https://github.com/Jeffreyyao/gym-minesweeper`.