Dungeons&Dragons Combat Simulation Using Q-Learning and DQN

Riccardo Evangelisti
Course of Autonomous and Adaptive Systems
Master's Degree Computer Engineering
University of Bologna
riccardo.evangelist6@studio.unibo.it
evangericcardo00@gmail.com

Abstract

This work presents a simulation of Dungeons& Dragons combat using Q-learning and Deep Q-Network (DQN) algorithms. The environment models a simplified D&D battle scenario with two agents, Erik and the Mimic, each with distinct attributes and combat strategies. Erik has a high movement speed and a ranged attack, while the Mimic is slower but more powerful in close combat. In this setup, Q-learning and DQN are applied exclusively to Erik, while the Mimic follows fixed strategies, either random or rule-based, to serve as a consistent opponent. The simulation is framed as a Markov Decision Process, where episodes represent entire combats, and steps involve the individual actions that an agent can take.

1 Environment

1.1 Structure

The environment simulates a Dungeons&Dragons playground. A classic D&D battle mat consists of a top-down view of a background (e.g., a forest, a street, or a room) with a drawn grid that divides the mat into squares. Each side of a square on the mat corresponds to 5 feet (approximately 1.5 m) in the game. This project considers two medium-sized creatures (referred to as "agents" from now on), each occupying one square.

To maintain manageable complexity, the environment has a width of 6 squares and a height of 7 squares. Figure 1 shows a graphic representation.

1.2 Combat system

Here are the rules for a simplified version of the D&D 5e combat system. Combat is divided into turns. The "playing agent" refers to the agent whose turn is active. During each turn, the playing agent (i) can move a number of squares up to its movement speed in any direction within the playground's borders, and (ii) can attack a number of times up to its available attacks, provided the enemy is within the attack's range. When the playing agent ends its turn, play passes to the next agent. No actions can be taken during another agent's turn.

At the start of a player's turn, they restore their maximum number of attacks and movement speed.

Preprint. Under review.

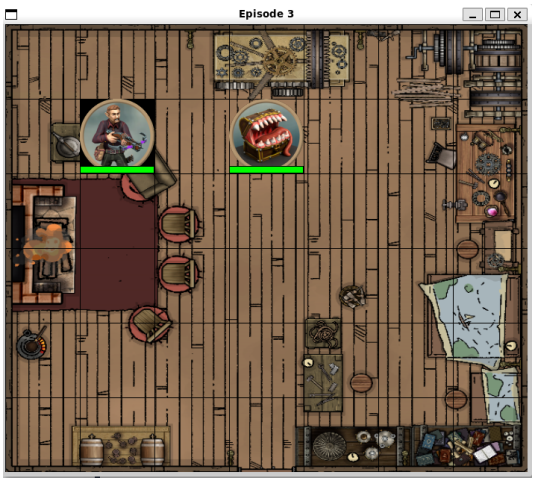

Figure 1: Simulated Environment.

Attribute	Erik	Mimic
Health Points	50	100
Movement Speed	5	2
Number of Attacks	1	2
Attack Damage	10	10
Attack Range	3	1

Figure 2: Agent Statistics.

1.3 Agents

Erik: A half-elf with high movement speed, equipped with a heavy crossbow, capable of long-range attacks. **Mimic**: A treasure chest that attempts to devour you when opened; slow in movement speed but high in health points and number of attacks.

The statistics are shown in Figure 2. The Mimic is a powerful creature that leaves no chance for its enemy when in close proximity. Erik has half the health points and fewer attacks but has the advantage of attacking from a distance.

2 Markov Decision Process

2.1 Episode, Step, Action Set

To adapt the simulation as a Markov Decision Process, an "episode" represents an entire combat between the two agents, ending with the death of either Erik or the Mimic. A single "step" occurs when the playing agent takes an action from the set of available actions. An action can only be taken if it is available, i.e., if the agent satisfies certain conditions.

The set of actions and their corresponding conditions are shown in Table 1.

Table 1: Available actions and conditions for the playing agent

Action	Availability Conditions
Movement UP, DOWN, RIGHT, LEFT, UP_RIGHT, UP_LEFT, DOWN_RIGHT, DOWN_LEFT	(i) The target position is not already occupied, (ii) The target position is within the playground borders, (iii) The playing agent has at least one square of movement left
Attack	(i) The enemy target is within the attack's range, (ii) The playing agent has at least one attack remaining
EndTurn	Always available

2.2 State

Given the configuration of the combat simulation, the "state" can be modeled as a tuple of values, each representing a specific aspect of the environment: x-coordinate of the playing agent, y-coordinate of

the playing agent, x-coordinate of the enemy agent, y-coordinate of the enemy agent, current HP of the playing agent, damage dealt to the enemy agent, remaining attack actions of the playing agent, remaining movement actions of the playing agent. In this way, the playing agent knows its own and the enemy's positions on the battlefield, what actions it can still perform, and the current status of its health points. Each agent does not know the enemy's total health points; instead, only the damage dealt is recorded.

2.3 Rewards

Two reward systems were considered:

- Simple Reward. The playing agent receives:

  +10 if the enemy is dead

  +4 if it dealt damage to the enemy

  +0 otherwise

- Advanced Reward. In addition to the Simple Reward, the playing agent receives:

  –5 if (i) it took the *EndTurn* action, (ii) still had attacks left, and (iii) an attack action was available (i.e., the enemy was within the attack's range).

  –3 if (i) it took the *EndTurn* action, (ii) still had attacks left, and (iii) had enough movement left to bring the enemy within attack range. This condition is calculated as the Chebyshev distance between the agents minus the maximum attack range, which must be less than or equal to the number of movement squares left.

The Simple Reward system directly incentivizes damage to the enemy. The Advanced Reward system aims to avoid wasting attack actions, which, in this simulation, are the only way to win the combat. It will be investigated whether the Advanced Reward penalizes the discovery of new patterns or strategies.

The reward is given immediately after an action is taken. When the playing agent strikes the final blow that ends the combat, it receives the maximum reward. Since the playing agent cannot lose the combat in response to its own action, no negative reward is given to the agent that loses the combat.

3 Models

3.1 Q-Learning

As stated in (1), Q-learning converges to the optimal action-values with probability 1, provided that all actions are repeatedly sampled in all states. This algorithm was chosen due to the simplicity of the simulation, which allows all action-state pairs to be repeatedly sampled over a large number of episodes.

It follows the implementation used in this study. To ensure exploration, a fixed epsilon-greedy policy is employed. The exploitation of the best action first filters the action-values corresponding to the current state. Next, it filters only the actions that are currently available to the playing agent. If no matching state-action pairs are found, a random action among the available options is returned. Otherwise, the action with the highest value among the filtered options is returned.

Filtering only the available actions is a necessary step, as almost all actions that an agent can take have certain conditions to meet (see Table 1). Without this filtering, the agent might choose actions that violate the rules of the environment (e.g., moving outside the map borders). Convergence is still guaranteed, as noted in (1) ("Notes", 1, p.287).

During the learning phase, the action-value for the current state is updated as follows:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left(R_t + \gamma \max_a [Q(S_{t+1}, a)] - Q(S_t, A_t) \right),$$

where R_t is the reward received after taking the action A_t in state S_t and transitioning to the next state S_{t+1}.

This approach ensures that no state-action pairs are saved for actions that cannot be taken in a given state. Thus, the action-values for the next state are computed only for the available actions, so the

$\max$ is calculated only among actions that can actually be taken. Without this, there would be a logical discrepancy, as the set of actions could differ from the actual available actions, and the $\max$ might select an action that would never be taken.

3.2 Deep Q-Network

The DQN algorithm (2) was chosen due to its simplicity in implementation. Following the approach of the original paper, the present study includes: a replay buffer, which "randomises over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution"; a target network, whose weights are periodically updated, "thereby reducing correlations with the target"(2).

The implementation adopted in this study follows this approach. Using the epsilon-greedy policy, exploitation for choosing the best action is performed by a forward pass through the main network, taking the state as input and producing the action-values as output. Among the available actions, the one with the maximum value is then selected.

During the learning phase, the tuple (current state S_t, action A_t, reward R_t, next state S_{t+1}, done, set of valid actions for the next state) is first stored in the replay buffer. A fixed-size batch is then sampled from the buffer, with the next states provided as input to the target network. The output is a set of vectors, each containing the values of all actions in the given next state.

Since the value function is approximated, the network always returns a vector containing values for all actions, even those that are not actually available in the given state. Therefore, during the calculation of the maximum value among the network's output, only the actions available in the given state (the "next states") are considered.

Once the maximum value for each of the next states in the sampled batch is computed, the target Q-values (the updated values of the actions taken in the "state" of the batch) are determined as follows: if the episode following the state is "done," the target Q-value equals the reward; if the episode is not done, the target Q-value is:

$$target_i = R_i + \gamma \cdot \max_a [\hat{Q}(S_{i+1}, a)] \ ,$$

where R_i is the reward obtained by taking action A_i in state S_i and transitioning to state S_{i+1}, and $\hat{Q}$ is the Q-value output from the target network.

The algorithm then computes the loss and updates the weights. First, the batch "states" are fed to the main network, which returns the Q-values; only the Q-values for the actions actually taken in each state of the batch are retained. Next, the loss (Mean Squared Error) is computed between the already calculated target Q-values and the Q-values obtained from the target and main networks, respectively. Then, the weights of the main network are updated through backpropagation.

Finally, at a fixed frequency, the weights of the main network are copied to the target network.

3.3 Others

To test the above algorithms, two additional strategies were used:

- Random strategy - Implemented as an epsilon-greedy policy with epsilon equal to 1.
- Rule-Based strategy - The simplest, yet powerful, strategy involves first positioning the agent to bring the enemy within range, and then taking the attack actions.

4 Experiments

It follows various tests that combines different strategies and hyperparameters fine tuning. If not specified, the results pertain to both Simple and Advanced Rewards, with no notable differences in the winning statistics or the learned strategy of play. If not specified, Erik has a static exploration rate of epsilon$= 0.05$.

First Experiments For both algorithms, starting with no prior knowledge, Erik wins almost all combats against Mimic with the Random strategy almost immediately, without developing any clear

play tactic [Experiments A, B]. In contrast, Erik loses 100% of combats against the Rule-Based strategy [Experiments C, D]. The objective is then to beat the Mimic that fully follows the Rule-Based strategy.

4.1 Q-Learning Training

Mimic starts with no movement or attacks Initially, Erik faces the Mimic whose number of attacks and movement speed are set to zero. Afterward, the Mimic is allowed to move, and only after some training is it able to attack. If the ability to attack were introduced all at once, Erik's winning rate would drop from 100% to 0% [Experiments E, F]. As shown in Figure 3, [Experiment G], the Mimic starts with no ability to move or attack (episodes 0 to 100k), then it deals half damage and has half the number of attacks (episodes 100k to 250k). Finally, it deals full damage but still has half the attacks (from episode 250k). As soon as the Mimic gains increased attack power, Erik stands no chance of winning, because he has not yet learned an effective strategy (for example, striking the Mimic and then moving away).

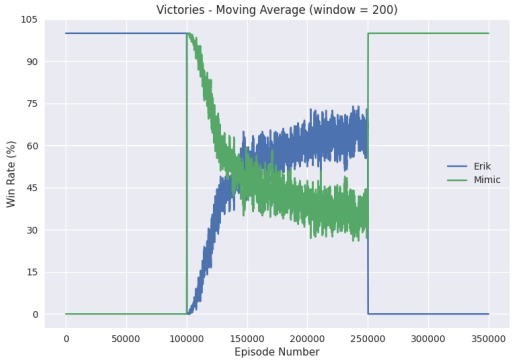

Figure 3: Victories over Episodes [Experiment G].

The approach described above may be wrong for various reasons. Since the winner's remaining HP is not a value factored into the reward system, whether Erik wins with full HP or near zero HP, the reward remains unchanged. Therefore, if in the first episodes the Mimic doesn't attack, Erik does not learn that his own Health Points are an important factor, and his strategy will remain to stay close to the enemy, because sooner or later he will receive the maximum reward anyway. It is necessary for the Mimic to start with full attacking power and the ability to move.

Mimic with high random action rate To allow Erik time to explore without immediately losing the combat, the Mimic starts with a very high epsilon (with full HP and all abilities), while Erik still uses a small, static epsilon [Experiment H]. As shown in Figure 4, Erik quickly converges near the maximum reward. However, when the Mimic's exploration rate changes (from 0.95 to 0.90 at episode 150k), Erik does not adapt. If the Mimic's exploration rate is further reduced, the same scenario repeats.

Additional tests were conducted using various combinations of exploration rate decay for both Erik and the Mimic [Experiments I, J, K, L, M, N]. The most promising strategy is to start with full randomness for both agents, allowing Erik the opportunity to explore. Then, the exploration rate is decreased linearly for Erik (from 1 to 0) and sub-linearly for the Mimic (from 1 to 0.5). As the Mimic becomes more powerful, Erik plays less randomly but at twice the speed. In this scenario, Erik starts to win more consistently when the Mimic plays with epsilon$= 0.7$. As shown in Figure 5, the total rewards begin to decrease, likely because Erik loses more quickly as episodes progress, thus receiving fewer positive rewards.

4.2 DQN Training

The DQN algorithm was primarily tested using the last approach previously described, due to the numerous forward passes through the network that slowed the training process. Therefore, it should be noted that these experiments were conducted with a smaller number of episodes.

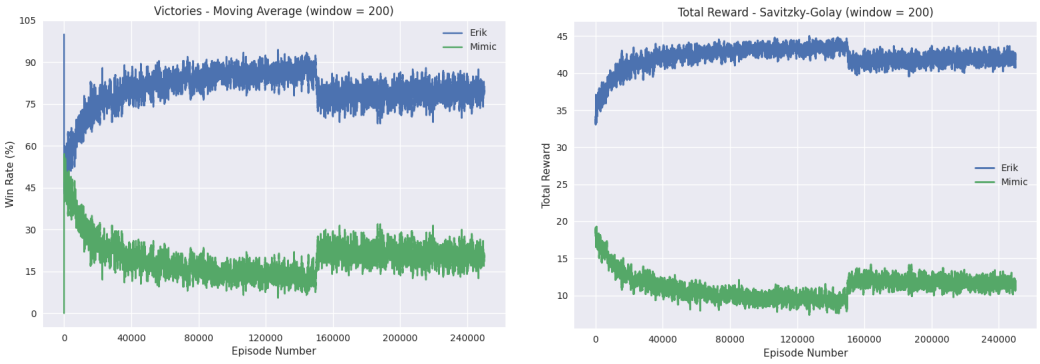

Figure 4: Victories and Total Rewards over Episodes [Experiment H].

Figure 5: Victories and Total Rewards over Episodes [Experiment L].

Considering the DQN network architecture and hyperparameters, different combinations were tried: replay buffer size, batch size for sampling from the buffer, target update frequencies, and the number of nodes and layers in the network [Experiments B, C, D]. With the exploration rate decreasing linearly for both Erik and the Mimic, there were not many changes compared to the Q-Learning results. Better outcomes were observed when training over more episodes.

5 Conclusions

In this research, significant results were not achieved with either algorithm. Although Erik is able to defeat the Mimic using a random strategy, he struggles to prevail when the random action rate is below 0.7.

Despite having lower statistics, Erik has the advantage of engaging in ranged combat. If he fully leveraged this ability, he could win consistently. However, his tendency to attack and then move randomly, limits his effectiveness. It was observed that Erik learns to stay close to the Mimic for effective attacks, following it when it moves away.

No significant difference was observed between the Advanced and Simple reward systems, likely because the behavior incentivized by the advanced system is eventually learned by the agent using the simple system.

Unfortunately, due to computational reasons, a limited number of experiments were conducted with the DQN algorithm, as its execution proved to be too slow.

An alternative reward strategy could involve assigning a victory reward inversely proportional to the winner's remaining HP or penalizing the agent for ending a turn near the opponent. However, these changes might overly constrain the exploration of new strategies.

6 Supplementary Material

The code and the final statistics of the tests are available at the GitHub repository.

References

[1] C. Watkins, P. Dayan "Technical Note: Q-Learning. Machine Learning", Machine Learning 8. 279-292 (1992) doi:10.1007/BF00992698

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland and G. Ostrovski, *et al.* "Human-level control through deep reinforcement learning", Nature **518** (2015) no.7540, 529-533 doi:10.1038/nature14236