# Playing Tank War Game With Different Search and Learning Methods

**Yiyang He**
Shanghaitech University
heyy@shanghaitech.edu.cn

**Zixin Teng**
Shanghaitech University
tengzx@shanghaitech.edu.cn

**Luo Wang**
Shanghaitech University
wangluo@shanghaitech.edu.cn

**Qihan Xu**
Shanghaitech University
xuqh@shanghaitech.edu.cn

**Xiaoxia Zhang**
Shanghaitech University
zhangxx5@shanghaitech.edu.cn

## Abstract

The game Tank War, a simplified version of the famous game *Battle City*, is a popular multi-directional shooter video game worldwide that has its own interesting rules. Due to the fact that some of the walls in this 2-D game can be breakable and the interactions between gamer and enemy tanks are so diversified, it's not very easy to play and we want to use AI methods we have learned to dive into this game and test different models' performance. We tested some search methods like Expectimax and other deep models like DQN method. Eventually we find out that Expectimax method surpasses the baseline agent(random agent) significantly and showed good performance.

## 1 Introduction

*Tank War* is originated from the famous game *Battle City*, which is a representative *multi-directional shooter* video game. In *Battle City* players control a tank and shoot bullets to fight against enemies. Though *Battle City* has a multi-player mode and much more items that have special effect, our project will mainly focus on the one-player mode and a simplified version of the game, which is just our *Tank War*. This game has some features similar to *Pac-man*, such as maps made by grids, movements, and one player agent vs. multiple hostile agents. However, there are also influential differences, including flying bullets that has same speed as agents. The main factors for an agent are:

- *Goal*: **Win** if no opponent lives; **Lose** if the agent dies, or its *headquarter* (a special item on grid) is destroyed.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

- *Reward*: **Score increment** if the agent **wins** or **kills an enemy**, and **decrement** if the agent **loses** or **moves**.

- *Action*: The agent may **move one grid left, right, up or down,** as well as **firing or not in each movement**. "Firing" means placing a flying bullet on the grid in front of it, with the bullet direction same as its facing direction.

- *State*: Valid states include **the location of visible** enemies, bullets and the player, as well as **scores** the player gets. A common match of the game is shown in Figure1.


Figure 1: Screenshot of a common *Tank War* match

## 2 Related Work

[Ngoc et al., 2019] explores the application of Deep Reinforcement Learning to create intelligent agents, addressing challenges in complex environments. It introduces a lightweight workflow with a target map and a Multi-Policy Control System (MPCS) to manage agent behaviors in real time without human feedback. Additionally, the study develops a multi-agent game that is similar to our target game, Tank Battle, for examining DRL agent behaviors. Simulation results show MPCS-guided agents outperform others in mean total rewards and human-like behaviors in complex environments like Seaquest and Tank Battle. Their work showed the potential of reinforcement learning and deep models for tank battle games, which gave us a lot of inspiration.

[Yung et al., 2009] also built a tank-battle game environment and use the methodology of reinforcement learning for the NPCs (tanks). Apart from making the game more interesting by adding enhanced interactions with tanks, it uses a dynamic reward during the training process and get a better convergence speed. Notice that our project also needs to deal with the sparse reward problem, and this article further confirmed that these dynamic and game-specific strategies will probably bring benefits to us.

As for the original *Battle City* game, there is almost no formal research on reinforcement learning and deep learning for this version of the game. But our group members still find out that [JiaXing et al., 2018] tried to implement DQN models in a similar scenario. Their models can perform much better than random agent but still exist great gap toward the human level, which is the sign that our implementation may also meet great challenge. Considering in our simplified game version, models that easier than deep models still have the probability to gain an above-random level at least, we plan to start from some basic methods, and then use some reinforcement and deep models to test their performance and give our analysis.

# 3 Methodology

## 3.1 Expectimax Search Agent

Expectimax search is a decision-making algorithm commonly applied in the realm of gaming agents, providing a probabilistic extension to the classic minimax algorithm. In the context of Tank War, leveraging the expectimax search methodology offers a structured approach for agent decision-making in a stochastic and dynamic gaming environment.

Notice that Tank War's complexity necessitates a depth-limited expectimax search to manage computational resources effectively. Determining a suitable depth for the search involves striking a balance between computational efficiency and decision accuracy.

In tackling the challenges posed by the Tank War game's dynamic environment, our approach integrates discretization and an adjusted evaluation function within the Expectimax search framework. Discretization simplifies the problem space by confining agent and enemy movements to integer grid distances, facilitating the definition of the "nextState" and enabling more efficient algorithmic processing. Furthermore, our adjusted evaluation function introduces penalties based on distance metrics and proximity to enemy bullets, enhancing the agent's decision-making by incentivizing engagement with nearby adversaries while mitigating the risk of being overwhelmed by enemy fire. These strategic adjustments enhance the agent's adaptability and performance within the Tank War game environment.

Through the fusion of discretization and an adjusted evaluation function, our Expectimax algorithm provides a robust foundation for decision-making in the Tank War game. By strategically penalizing distant interactions and potential bullet exposure, the agent is incentivized to pursue advantageous positions and engage with nearby adversaries. This approach not only enhances the agent's ability to navigate the game's dynamic challenges but also fosters a more nuanced understanding of strategic gameplay elements. Overall, our methodology empowers the agent to make informed decisions, balancing risk and reward to achieve optimal outcomes in the complex and dynamic Tank War gaming environment.

**Algorithm 1** Evaluation Function for Tank War Expectimax Agent

**Require:** Agent position, Enemy positions
**Ensure:** Evaluation value for agent's action
Initialize evaluation value $E$ to current score S
**for** each enemy **do**
    Calculate distance $d$ between agent and current enemy
    **if** agent is far from all enemies **then**
        Add penalty to $E$ based on $d$
    **end if**
    **if** agent is in front of and close to current enemy **then**
        Add penalty to $E$ to avoid bullet proximity
    **end if**
**end for**
**return** $E$

## 3.2 Adapted A* Search

We propose to merge heuristic search with a greedy strategy, prioritizing local optimality over global path planning. This strategy resembles the use of heuristic functions in algorithms such as A*, guiding the search algorithm towards promising trajectories instead of exhaustively exploring all possible paths, thus expediting solution discovery. By assessing the distance to the nearest enemy tank and bullet at each step, the approach swiftly identifies promising actions, adapting to dynamic enemy movements by abandoning previous plans. This dynamic and agile decision-making accelerates gameplay efficiency.

In summary, the heuristic function design for Tank War primarily focuses on the proximity to the nearest enemy bullet, accounting for anticipated trajectories. Excessive proximity incurs a severe penalty, while closeness to the nearest enemy tank warrants a lesser penalty. Conversely, a positive

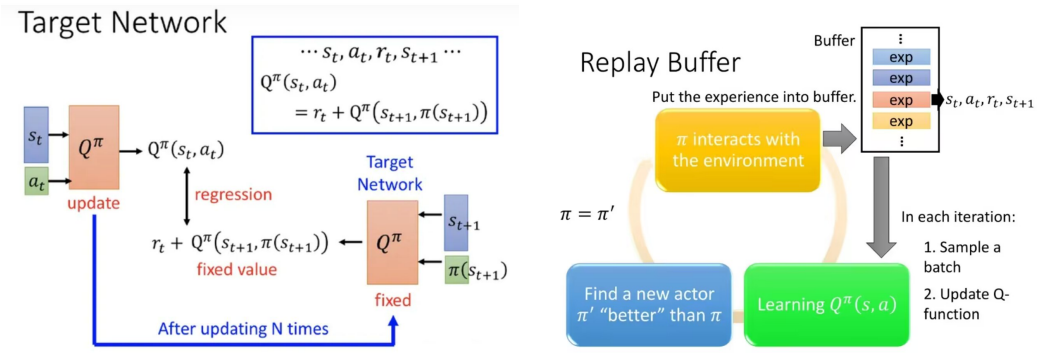


Figure 2: Target Network and Replay Buffer

reward is bestowed if neither condition is met, inversely proportional to the distance from the nearest enemy tank. Recognizing Tank War's departure from traditional search problems due to dynamic enemy movements, our novel approach employs only the heuristic function from A* search, integrating greedy principles to select the best initial action at each step, promising an alternative strategy.

### 3.3 Deep Q Network

DQN is a reinforcement learning algorithm that combines Q-Learning, which is what we have learned in class, with deep neural networks to let RL work for complex, high-dimensional environments, like video games, or robotics. It holds considerable potential and significance within the context of the Tank War game.

- **Learning Complex Strategies**: The multifaceted decision-making involved in Tank War, including movement, shooting, and evasion of enemy bullets, demands sophisticated strategies. DQN excels at learning and optimizing intricate decision-making strategies to adapt to the diverse scenarios presented in the game.
- **Adaptability and Generalization**: Due to potential variations in game maps and enemy tank configurations, DQN, through its learning process, enhances the adaptability of the agent, enabling it to generalize across different gaming scenarios effectively.
- **Optimization of Reward System**: The reward system in the game plays a pivotal role in the learning process. DQN facilitates the optimization of agent behavior by adjusting the weights of reward signals, encouraging successful actions leading to victory and enemy eliminations, while penalizing failures and suboptimal movements.

The composition and design of the two crucial model—Target Network and Replay Buffer are shown in Figure2. In the realm of DQN, the concept of a target network enhances learning stability by separating Q-value estimation into two networks: an inner network for current state Q-values and an outer network, termed the target network, for future state Q-values. This approach mitigates divergence risks and optimizes learning efficiency. Additionally, the integration of a replay buffer facilitates the storage and sampling of experience data, allowing for efficient sample reuse during Q-value updates. By retaining and reusing previously sampled experiences, the replay buffer significantly enhances sample utilization efficiency, contributing to improved learning performance in the DQN algorithm.

When designing the DQN agent, consideration is also given to the following factors:

- **State Representation**: The appropriate representation of the game state is crucial for DQN performance. We choose to include visible enemies, bullets, player position and map information in the state representation.

- **Action Space**: The actions that the agent can perform in the Tank War game, such as movement (left, right, up, down) and shooting, constitute the action space for the DQN agent.
- **Reward Function**: A well-designed reward function encourages the agent to learn favorable behaviors. Our reward function incentivizes the agent for victories and eliminations while imposing penalties for losses and suboptimal movements.
- **Deep Neural Network Structure**: Utilizing a deep neural network structure as the function approximator for Q-values allows the agent to learn the mapping between states and actions effectively.

In synthesizing these considerations, we anticipate the design of a superior-performing DQN agent within the Tank War game, thereby elevating the performance and adaptability of gaming intelligence through the lens of reinforcement learning principles.

# 4 Experiment

## 4.1 Model Settings

- **Reward Settings for all models**:

Table 1: Reward Settings

| Action | Reward |
|---|---|
| Destroy an Enemy | +100 |
| Destroy all Enemies | +500 |
| Living Penalty every 60 frames | -1 |
| Player death or headquarter destruction | -500 |

- **Expectimax Search Agent**:
  depth = 1; Evaluation function: Algorithm 1.
- **Adapted A* Search Agent**:

**Algorithm 2** Heuristic Function for Tank War

**Require:** Position of player tank and enemy tanks and bullets
**Ensure:** Heuristic score for each action
Initialize distance to nearest enemy bullet $d_b$
Initialize distance to nearest enemy tank $d_t$
**if** $d_b$ is too small **then**
    **return** Extreme penalty + $d_b$
**else if** $d_t$ is too small **then**
    **return** Moderate penalty + $d_t$
**else**
    **return** Positive reward - $d_t$
**end if**

- **DQN Agent**: Another complex map for network traning;

All Search Agents will be tested 100 times and gain average results. DQN Agents will be trained for over 5000 episodes.

## 4.2 Baseline

We will use random agent and human players' performance as our baseline. Notice that unlike previous works, for benchmark we do not always choose skilled players, but instead players of different game levels within the group.

Table 2: Parameters for Deep Reinforcement Learning

| Parameter | Value |
|---|---|
| Learning Rate | $1 \times 10^{-4}$ |
| Epsilon Greedy Start Probability | 0.5 |
| Epsilon Greedy Decay | Decreasing over time |
| Discount Factor (Gamma) | 0.99 |
| Experience Replay Buffer Capacity | 2048 |
| Batch Size | 64 |
| Number of Training Episodes | 5500 |
| Initial Learning Threshold | 0 |
| Importance Sampling Weight (Beta) | 0.4 |

### 4.3 Result

Table 3: Result of different models

| Model | Average Score | Highest Score | Win Rate |
|---|---|---|---|
| Expectimax | +706 | +982 | 0.82 |
| Adapted A* | +458 | +921 | 0.70 |
| Random | -280 | +905 | 0.12 |
| Human Players | +130 | +870 | 0.51 |

See DQN explanation below.

Following the refinement of the Adapted A* Search algorithm, while the overall game score remains unaltered, a notable enhancement is observed in computational efficiency, resulting in increased running speed. Upon conducting a comprehensive analysis of the outcomes, a noteworthy issue with the Adapted A* algorithm is identified: the agent frequently exhibits a tendency to linger in close proximity to enemy tanks to avoid excessive proximity, leading to instances of hovering near enemy tanks. Consequently, the agent incurs substantial penalties for survival.

While our DQN model can achieve high maximum values, its average scores and win rates notably lag behind human performance levels, marginally surpassing random agents and displaying substantial data fluctuations. In Tank War, DQN's performance limitations stem from two primary factors. Firstly, the environment's inherent stochasticity, characterized by randomly designed enemy tanks and dynamic terrain, poses challenges for accurately forecasting future state values, potentially impeding the agent's ability to learn stable strategies effectively. Secondly, algorithmic constraints, particularly in handling high-dimensional state spaces, present challenges for efficiently processing and learning from the extensive state space represented as 3x84x84 NumPy arrays. These collective limitations hinder DQN's performance in Tank War, underscoring the need for further research to address these challenges and enhance the algorithm's effectiveness in dynamic and stochastic environments. Notably, the DQN model can achieve maximum values exceeding 900, and with additional training, we anticipate further performance improvements.

Furthermore, after we get primary results from Table 3, we decided to put our focus on Expectimax method since it has better performance generally. It's observed that the win rate gradually decreases when predicting straight-line movements, possibly because the probability of enemies turning increases over time. Therefore, deeper predictions become less accurate. Additionally, using the Expectimax algorithm, where the agent selects Max and the enemy selects Expect, directly predicting straight-line movements disregards the enemy's decision-making process. This leads to the agent consistently choosing the action with the maximum expected payoff, resulting in a tendency towards risky behavior and increased susceptibility to being defeated, especially evident at greater depths.

We also set two different strategy like what Table 4 has listed. The Normal strategy involves solely avoiding the enemy's direct forward path. In contrast, the Conservative strategy, particularly represented by column C, inspired by previous insights, involves more cautious movement, avoiding areas surrounding the enemy. Although the Conservative strategy may result in fewer victories, it tends to yield higher average scores. This could be attributed to the cautious nature of the Conservative

strategy, where the agent avoids risky situations, potentially missing out on opportunities for quick victories.

In summary, the distinction between the Normal and Conservative strategies primarily lies in the evaluation function. The Conservative strategy, inspired by a recent insight, emphasizes cautious movements to avoid close encounters with enemies. Conversely, the Normal strategy, previously employed, focuses on simply avoiding direct confrontation with enemies.

Table 4: Further exploration results of different Expectimax Search

| | Mean | Win Rate | Max | Min | Depth | Strategy |
|---|---|---|---|---|---|---|
| Expectimax-1 | 684.4 | 0.8 | 971 | -422 | 1 | Normal |
| Expectimax-2 | 315.1 | 0.55 | 968 | -522 | 1 | Conservative |
| Expectimax-3 | 141.5 | 0.45 | 975 | -570 | 2 | Conservative |
| Expectimax-4 | 3.9 | 0.3 | 983 | -515 | 5 | Conservative |
| Expectimax-5 | 581.65 | 0.7 | 972 | -414 | 1 | Normal |
| Expectimax-6 | 154.3 | 0.4 | 974 | -512 | 2 | Normal |

Our initial algorithm, when assessing future states, assumed that enemies would continue moving in a straight line. While this holds true in the majority of cases, there is approximately a one-fifth probability that enemies will change direction. This unpredictability increases the vulnerability of our agent, leading to potential unexpected deaths. To address this issue, we made adjustments such that during the expectation process of enemy movements, different weights are assigned to various directions, followed by evaluating the goodness of states through a weighted average. Following the refinement of the algorithm, there is a notable improvement in the agent's effectiveness. For instance, as shown in the table above, Expectimax-1 demonstrates enhanced performance compared to the subsequent five results obtained from the algorithm that did not consider changes in direction. This improvement is evident in both the average scores and success rates.

## 5 Conclusion

We present a foundational architecture alongside practical algorithms for AI agents participating in Tank War. Based on an open-sourced Python implementation of *Battle-City* game, we conducted extensive exploration and exploitation, culminating in the implementation and evaluation of Expectimax and DQN models. Despite encountering various implementation challenges, we successfully navigated them to attain significant insights. However, it is noteworthy that while our DQN model may have fallen short of expectations, our Expectimax implementation yielded favorable results. Notably, we experimented with diverse parameters to enhance our models' efficacy.

The observed limitations of our deep learning model in Tank War are attributed to the game's inherent complexity and the hardware constraints of deep model training environments. Nonetheless, such outcomes were anticipated. As we advance in training depth and refine our models iteratively, deep reinforcement learning techniques exhibit potential for substantial advancements and promising adaptability to gaming scenarios. Additionally, the integration of fundamental search algorithms has enhanced our comprehension of the game dynamics, prompting our pursuit of synergizing various model strengths in future endeavors to attain superior outcomes.

## References

[1] Ngoc Duy Nguyen & Thanh Nguyen & Saeid Nahavandi, *Multi-agent behavioral control system using deep reinforcement learning* In *Neurocomputing*, Volume 359, 2019, Pages 58-68, ISSN 0925-2312.

[2] Yung-Ping Fang & I-Hsien Ting, *Applying Reinforcement Learning for Game AI in a Tank-Battle Game* In 2009 Fourth International Conference on Innovative Computing pp. 1031-1034, doi: 10.1109/ICICIC.2009.114.

[3] Jiaxing Geng & Junjie Dong & Zihuan Diao, *AI Agent for the Battle City* from Stanford University [Online; accessed 1-January-2024].

[4] Open Source Code of the pygame: *https://github.com/IronSpiderMan/TankWar* [Online; accessed 1-January-2024].

[5] Classic Battle City reference that help us dive into this game: *https://github.com/raitisg/battle-city-tanks* [Online; accessed 1-January-2024].

[6]Thanh Nguyen & Ngoc Duy Nguyen & Saeid Nahavandi, *Multi-Agent Deep Reinforcement Learning with Human Strategies* Deakin University, DOI: 10.1109/ICIT.2019.8755032