Let's play BlackJack with Elaborate Methods

Yinghao Cai, Yiqing Wang, Chen Huang, Keyu Wang

***Abstract*—In this project, we explore various methodologies for achieving higher scores in the game of Blackjack. Concretely, Yinghao Cai tries the double-Q learning to train DQN, Keyv Wang tries Q-learning based on Monte-Carlo simulation, Yiqing Wang tries actor-critic algorithm, a policy-based method, and Chen Huang tries the mathematical formula and DQN. Following the experimentation phase, a tournament was conducted among the five separate models to determine the most successful method. Ultimately, the results indicated that Huang's approach yielded the highest score.**

I. Problem Statement

The game uses infinite pairs of playing cards to remove the king of size (that is, the probability of 1-10, J, Q and K being drawn is always 1/13). When calculating the sum of the points in the hand, J, Q and K count as 10 points, and Ace can count as 1 point or 11 points (if counting 11 points will not cause the sum to exceed 21, it will be counted as 11 points, otherwise it will be counted as 1 point), and the remaining number of cards corresponds to the numbers.

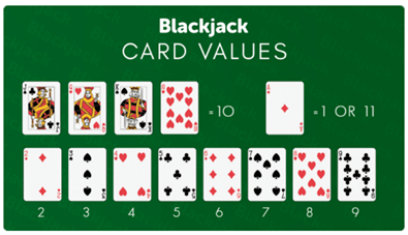

Fig. 1. Blackjack card values

At the beginning of each game, the banker and the leisure player draw two cards and show their first cards, and then enter the stage of leisure player touching cards. In the card-touching stage, the leisure player can keep touching cards, but if the sum of all the hands is greater than 21 points, the game will be lost directly and the banker will win. Unless you lose more than 21 points at this stage, the cards you touch at home are kept secret from the banker.

After the leisure player finishes playing cards, they enter the banker 's card-playing stage. The banker can keep playing cards, but if the sum of all the hands is greater than 21 points, the banker will lose the game directly and the idlers will win.

After the banker finished playing cards, both sides showed their hands. The one with the larger number of points wins, and the one with the smaller number loses. If the points are equal, record them. Both sides draw.

To sum up, here's the basic guide of how to play blackjack:

• The goal in blackjack is to beat the banker's hand without going over 21.

•You'll receive 2 cards at the beginning of each round, and we'll add up the values of these cards.

• Cards 2-10 have face value; King, Queen, Jack are worth 10; and Aces are either a 1 or an 11 — it's up to you to decide.

•The banker also draws two cards. The aim of the game is to beat his hand (have a higher hand) without going over 21.

•If you would like the banker to deal you another card, you tell him "hit".

• If you do not want to be dealt another card, you "stand".

II. Double-Q learning

In this part, I try DQN to predict the values of the action "Hit" and "Stand" and use double-Q learning.

A. Model

Given that the size of input and output is small, I choose MLP(multilayer perceptron) to fit the value function $f : R^S \to R^A$, in which S is the length of input and A is the number of available action. In BlackJack, $S = 3, A = 2$. In details, there are 3 layers in MLP and the sizes are 3×32, 32×32 and 32×2.

B. Algorithm

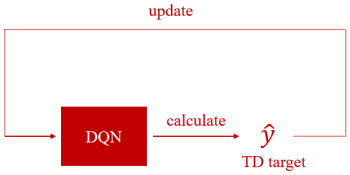

Fig. 2. Diagram of traditional DQN training

In traditional Q-learning to train DQN, only one network is used to select the next action and evaluate, figure 2. While double Q-learning is also a TD learning algorithm that addresses the overestimation bias problem in traditional Q-learning. It combines two Q-value networks to separate the action selection and evaluation steps, figure 3. By using one Q-function to choose the best action and another to estimate its value, double Q-learning reduces the overestimation of Q-values and provides more accurate action-value estimates. This technique improves the stability and performance of Q-learning algorithms by mitigating the overoptimistic value estimates and promoting better exploration-exploitation trade-offs.

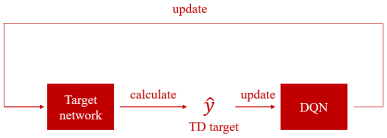

Fig. 3. Diagram of double-Q learning

1) Train: I use double Q-learning and experience replay to train the network. The training process and settings are as follow:

1) **Initialize the replay buffer**
In this step, the $\epsilon - greedy$ policy is adopted by the agent to interact with the environment and with tracts, the quadruples (s_i, a_j, r_j, s_{j+1}) are filled in the replay buffer. Experimentally, the size of the replay buffer and the number of tracts should be large enough for training. According to the analysis in [3], if DQN is used for Atari games, it is best to start experience playback to update DQN when $200,000$ quadruples are collected; if you use a better Rainbow DQN, you can start

when $80,000$ quadruples are collected Update DQNs. When the number of quadruples in the playback array is not enough, DQN only interacts with the environment without updating the DQN parameters, otherwise the experimental effect is not good. In my experiment, the size of the replay buffer and the number of initial tracts are both $20,000$.

2) **Forward**
Then I randomly choose a batch of training samples (in my experiment, batch size is 64) . For the samples, states are extracted and input into the DQN to calculate the values of the actions, "Hit" and "Stand". For the given quadruple (s_i, a_j, r_j, s_{i+1})

$$\hat{q}_j = Q(s_j, a_j; w) \tag{1}$$

and for the selected action a^*,

$$a^* = \text{argmax}_{a \in A} Q\left(s_{j+1}, a; \boldsymbol{w}\right) \tag{2}$$

in which w is the weights of DQN. Then for the TD target,

$$\tilde{y}_j = r_j + \gamma \cdot \hat{q}_{j+1} = r_j + \gamma \cdot Q(s_j, a^*; w^-) \tag{3}$$

in which $\gamma = 1$ and w^- is the weights of the target network. Then for TD error δ

$$\delta_j = \hat{q}_j - \tilde{y}_j \tag{4}$$

and loss L is

$$L = \frac{1}{2}\delta^2 \tag{5}$$

3) **Optimization**
The optimizer in this experiment is Adam [4]. The optimized parameters are just the weights in DQN. Hence, the parameters in DQN is updated through gradient descent,

$$w \leftarrow w - \alpha \cdot \nabla_w L \tag{6}$$

in which, α is the learning rate(in my experiment, $\alpha = 1e^{-6}$) and parameters of the target model are updated by averaging the parameters of DQN and the target model,

$$w^- \leftarrow \tau \cdot w^- + (1 - \tau) \cdot w \tag{7}$$

in which τ is the hyper parameter and $\gamma = 0.1$ in my experiment.

4) **Update the replay buffer.**
After updating the parameters of the DQN and the target model, the agent interacts with the environment and update part of the samples in the replay buffer. In my experiment, there are 20 samples are updated.

5) **Other settings**
The number of training epochs is 25 and in each epoch 750 steps are executed. The loss is recorded every 100 steps and the curve is shown in figure 4.

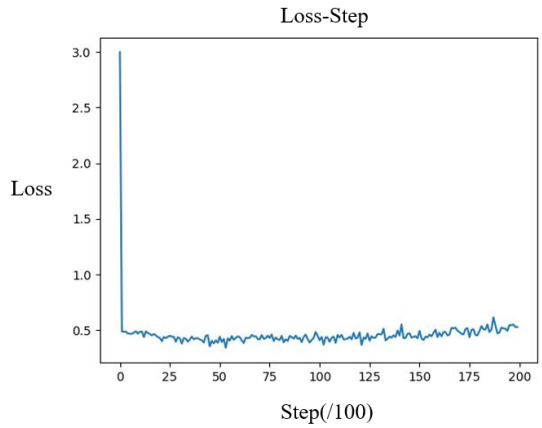

Fig. 4. Loss function change curve

2) Test: After each epoch, the agent plays BlackJack 1000 times as a player against the dealer with a simple policy and calculate the average score as the test score. The simple policy is hitting when the sum of total points is smaller than the threshold (17 by default). The result is shown in figure 5. Then I choose the model with best test score -0.088.

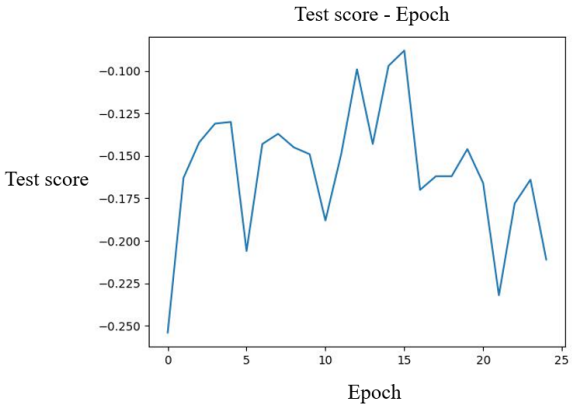

Fig. 5. Test score change curve

III. Monte-Carlo-simulation-based Q-learning

In this part, I try Monte-Carlo-based Q-learning to predict the values of the action "Hit" and "Stand".

A. Basic idea

Monte Carlo simulation is a numerical calculation method based on random sampling, which can be used to solve complex mathematical problems. In solving the Blackjack problem, the basic idea of Monte Carlo simulation is to simulate the game process of the casino by random sampling, so as to get the optimal strategy of the Blackjack problem.

Specifically, the basic steps of Monte Carlo simulation are as follows:

1. Define the problem: define the goal and constraints of the problem. In the Blackjack problem, the goal is to minimize the loss of gamblers, and the constraint is that only a certain amount of money can be bet in each game;

2. Random sampling: random sampling from the problem space, in the Blackjack problem, each time a gambler's bet amount and a card are extracted;

3. Simulation experiment: According to the question and sample information, simulate the game process in the casino, such as the betting, licensing and scoring process of gamblers in the Blackjack problem;
4. Statistical analysis: According to the results of simulation experiments, the winning percentage and average income of various strategies are counted, in the Blackjack problem, the winning percentage and average income of different betting strategies are counted;
5. Optimization strategy: According to the statistical analysis results, optimize the solution of the problem, in the Blackjack problem, choose the optimal betting strategy.

Through Monte Carlo simulation, we can simulate the stochastic process of Blackjack game, and thus get the optimal betting strategy.

B. State-action value function

Value iteration method and policy iteration method are the best strategies to solve the problem when the environmental model is known. These methods can be collectively called model-based methods. But in many cases, the model of the environment is unknown, and we don't know how to transfer between States, or even what all the state spaces look like. In this case, we can't use Bellman equation to solve the value function, and we can't get the best strategy from the value function:

$$\pi(s) = \arg\max_a \left(R_s^a + \gamma \sum_{'} P_a\left(s, s'\right) V\left(s'\right) \right)$$

We introduce state-value function $Q(s,a)$:

$$Q(s,a) = R_s^a + \gamma \sum_{s'} P_a\left(s, s'\right) V\left(s'\right)$$

$$\pi(s) = \arg\max_a Q(s,a)$$

By directly solving $Q(s,a)$, we can get the best strategy. Below we give a specific definition of $Q(s,a)$: State-action cost function under strategy π In the state, and immediately take action, and follow the strategy of π operation can get the cumulative expected reward, namely:

$$Q_\pi(s,a) = E_\pi\left(R_s^a + \gamma Q_\pi\left(s', a'\right) \mid S_t = s, A_t = a\right)$$

Although we don't know enough about environmental information, we can learn $Q(s,a)$ by interacting with the environment and generating a series of observation data. In the case of discrete state space and action space, a table called Q table can be established to store $Q(s,a)$ corresponding to state-action, horizontally indicating state and vertically indicating action. Through continuous iteration, the value of Q table is updated, and finally the decision is made by using Q table, according to $\pi(s) = \arg\max Q(s,a)$.

C. Monte-Carlo-based RL Method

The reinforcement learning method based on Monte Carlo does not need a known environmental model, but obtains multiple observation sequences through continuous interaction with the environment, and then uses the empirical average obtained from multiple observation sequences to estimate expectations and solve the cost function. Denote an observation as follows:

$s_0, a_0, r_0, s_1.a_1.r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$

The cumulative reward of t moment state s is:

$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-1} r_T$

Taking the state-action value function under the strategy as an example, under the strategy of π, the average of the cumulative reward of (s,a) generated by N interactions is calculated to approximate the expectation of cumulative reward:

$$Q_\pi(s,a) \approx avg(G_t), s.t. S_t = s, A_t = a$$

In the process of approximation function, the same state may appear repeatedly in a complete observation sequence, thus deriving two calculation methods:

First visit: Use the cumulative reward of (s,a) after the first visit to the state-action pair in each experimental observation sequence.

$$Q_\pi(s,a) = \frac{G_{11}(s,a) + G_{21}(s,a) + \cdots + G_{N1}(s,a)}{N(s,a)}$$

Every visit: The cumulative reward of (s,a) is obtained by using all the accessed states-actions in each experimental observation sequence.

$$Q_\pi(s,a) = \frac{G_{11}(s,a) + G_{12}(s,a) + \cdots + G_{21}(s,a) + \cdots + G_{N1}(s,a) + \cdots}{N(s,a)}$$

Next, we write the whole process of solving Blackjack problem by Monte Carlo.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$Q(s,a) \in R$ (arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Returns $(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

Generate an episode from S_0, A_0, following π : $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ $G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$$G \leftarrow \gamma G + R_{t+1}$$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$: Append G to Returns (S_t, A_t)

$Q\left(S_t, A_t\right) \leftarrow average\left(Returns\left(S_t, A_t\right)\right)$

$\pi\left(S_t\right) \leftarrow \arg\max_a Q\left(S_t, a\right)$

TableI shows a commonly used strategy table.

When the player has an "A", the best choice is shown in TableII.

IV. Actor-Critic Algorithm

In this section, we employ the Actor-Critic method [5] to play *BlackJack*. Specifically, feed-forward neural networks are chosen to represent both the actor deciding "Hit" or "Stand", and the critic evaluating current moves. Additionally, while model training, the resetting technique is applied to mitigate the effect of primacy bias [6].

A. Algorithm Design

In policy-based RL, the optimal policy is computed by manipulating policy directly, and value-based function implicitly finds the optimal policy by finding the optimal value function. Policy-based RL is effective in high dimensional & stochastic continuous action spaces, and learning stochastic policies. At the same time, value-based RL excels in sample efficiency and stability.

The main challenge of policy gradient RL is the high gradient variance. The standard approach to reduce the variance in gradient estimate is to use baseline function [2]. There are lots of concern about adding the based line will invite the bias in the gradient estimate. There is proof that baseline doesn't bring the basis to the gradient estimate.

Actor–critic methods are TD methods that have a separate memory structure to explicitly represent the policy independent of the value

TABLE I
A COMMONLY USED STRATEGY TABLE FOR BLACKJACK

Player	Dealer2	Dealer3	Dealer4	Dealer5	Dealer6	Dealer7	Dealer8	Dealer9	Dealer10	DealerA
≤ 11	H	H	H	H	H	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
≥ 17	S	S	S	S	S	S	S	S	S	S

TABLE II
THE STRATEGY TABLE FOR PLAYER WITH AN "A"

Player	Dealer2	Dealer3	Dealer4	Dealer5	Dealer6	Dealer7	Dealer8	Dealer9	Dealer10	DealerA
≤ 17	H	H	H	H	H	H	H	H	H	H
18	S	S	S	S	S	S	S	H	H	H
≥ 19	S	S	S	S	S	S	S	S	S	S

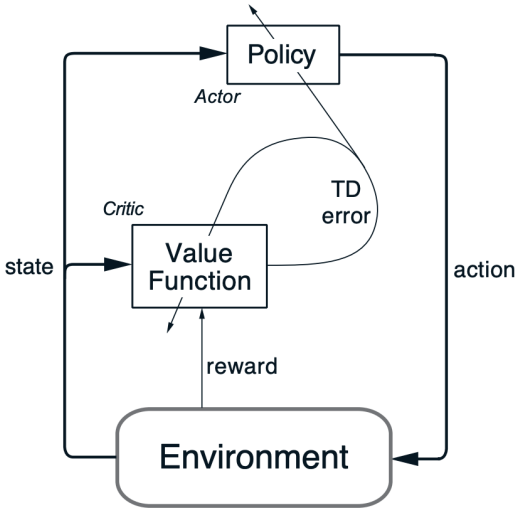

Fig. 6. Algorithm architecture for typical Actor-Critic methods.

function. The policy structure is known as the actor, because it is used to select actions, and the estimated value function is known as the critic, because it criticizes the actions made by the actor. Learning is always on-policy: the critic must learn about and critique whatever policy is currently being followed by the actor. The critique takes the form of a TD error. This scalar signal is the sole output of the critic and drives all learning in both actor and critic, as suggested by Fig.6.

In a simple term, Actor-Critic is a Temporal Difference(TD) version of Policy gradient [2]. It has two networks: Actor and Critic. The actor decided which action should be taken and critic inform the actor how good was the action and how it should adjust. The learning of the actor is based on policy gradient approach. In comparison, critics evaluate the action produced by the actor by computing the value function.

This type of architecture is in Generative Adversarial Network(GAN) where both discriminator and generator participate in a game [1]. The generator generates the fake images and discriminator evaluate how good is the fake image generated with its representation of the real image. Over time Generator can create fake images which cannot be distinguishable for the discriminator [1]. Similarly, Actor and Critic are participating in the game, but both of them are improving over time, unlike GAN [1].

Actor-critic is similar to a policy gradient algorithm called REINFORCE with baseline. Reinforce is the MONTE-CARLO learning that indicates that total return is sampled from the full trajectory. But in actor-critic, we use bootstrap. So the main changes in the advantage function.

From another perspective, actor–critic methods are the natural extension of the idea of gradient bandit methods to TD learning and to the full reinforcement learning problem. Typically, the critic is a state-value function. After each action selection, the critic evaluates the new state to determine whether things have gone better or worse than expected. That evaluation is the TD error Eq.8.

$$\delta_t = R_{t+1} + \gamma V_t\left(S_{t+1}\right) - V\left(S_t\right) \tag{8}$$

where V_t is the value function implemented by the critic at time t. This TD error can be used to evaluate the action just selected, the action At taken in state S_t. If the TD error is positive, it suggests that the tendency to select A_t should be strengthened for the future, whereas if the TD error is negative, it suggests the tendency should be weakened. Suppose actions are generated by the Gibbs softmax method Eq.9.

$$\pi_t(a \mid s) = \Pr\left\{A_t = a \mid S_t = s\right\} = \frac{e^{H_t(s,a)}}{\sum_b e^{H_t(s,b)}} \tag{9}$$

where the $H_t(s,a)$ are the values at time t of the modifiable policy parameters of the actor, indicating the tendency to select (preference for) each action a when in each state s at time t. Then the strengthening or weakening described above can be implemented by increasing or decreasing $H_t(S_t, A_t)$, for instance, by Eq.10.

$$H_{t+1}\left(S_t, A_t\right) = H_t\left(S_t, A_t\right) + \beta\delta_t \tag{10}$$

where β is another positive step-size parameter.

B. Simulation Results

In practice, the actor and critic are fulfilled through two independent FFNs. We adopted AdamW optimizer and applied the resetting [6] technique to avoid overfitting on early data points. Namely, given an agent's neural network, periodically reinitialize the parameters of its last few layers while preserving the replay buffer.

For better performance, we decide the hyperparameters through model selection, e.g., weight decay, network structure, activation methods. We set decay rate as $1e-4$, 3-layered FFN with ReLU activation. While training, we play BlackJack for 1000 times in one

TABLE III
THE STRATEGY LEARNT BY DQN (WITHOUT A)

Player	Dealer2	Dealer3	Dealer4	Dealer5	Dealer6	Dealer7	Dealer8	Dealer9	Dealer10	DealerA
$\leq$14	H	H	H	H	H	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	S	H	H	H	H
17	S	S	S	S	S	S	S	H	H	H
$\geq$ 18	S	S	S	S	S	S	S	S	S	S

TABLE IV
THE STRATEGY LEARNT BY DQN (WITH A)

Player	Dealer2	Dealer3	Dealer4	Dealer5	Dealer6	Dealer7	Dealer8	Dealer9	Dealer10	DealerA
$\leq$ 16	H	H	H	H	H	H	H	H	H	H
17	S	H	H	H	H	H	H	H	H	H
18	S	S	S	S	S	S	S	S	S	H
$\geq$ 19	S	S	S	S	S	S	S	S	S	S

epoch and accumulate losses to optimize actor and critic simutaneously. Training epoch is set as 10000 and model performance is evaluated through 10000 BlackJack rounds every 100 epochs. We carried out the training process with 100 different random seeds, and selected the best one with evaluation average reward being 0.122.

V. MATHEMATICAL FORMULA

The game BlackJack can be solved by Combinatorial Mathematics too. Since the number of poker is infinity, therefore the probability of getting a certain card is a constant number no matter what the state is, thus rendering listing all latent states and calculating the winning rate of each action possible. In practice, we can use the exhaustive method to find all possible outcome in different states and use statistics method to find the action yielding higher reward. After exploring all possible conditions, we can choose the action with the highest winning rate and generate the strategy table which ensures every step is the optimal action and guarantees the highest reward.

Then, we compare the strategy of DQN with that of math formula. TableIII and TableIV are the strategy learnt by the deep neural network. TableIII is the strategy picked by the player without an "A" and TableIV is that with an "A". To our surprise, the DQN algorithm outperforms the mathematical strategy which ought to be the optimal choice with a large margin. We try to explain the performance gap between the DQN algorithm and mathematical strategy by analysing the strategy table of them. From which, we can see that the strategy learnt by the DQN method uses a more aggressive method than the mathematical strategy, thus enabling agent with the DQN policy to gain more points than the mathematical strategy table. However, taking a more aggressive strategy also means a bigger chance to exceed 21 which will lose the game. As a result, we can see that both the DQN algorithm and the mathematical method stop asking for more poker once the total points get to 18. When the points of the player ranges from 14 to 17, the mathematical strategy will choose to add new poker only when the opponent shows a big poker. This can partly explain the depressing performance of the mathematical formula with the DQN method. First of all, the mathematical strategy is generated on the assumption that the dealer will stop asking for new pokers when its points is larger than 17. As a result, the player can also stop adding new cards with a point near 17 since if the score exceed 21, it will lose the game immediately. On the other hand, the DQN strategy, which is also trained on the dealer with a simple strategy that picks poker until the score get to 17, generates a different strategy from the mathematical formula thanks to the noise during the training. The DQN strategy not only shows competitive performance against the simple 17 strategy, but also surpasses the mathematical formula which can only deal with the simple 17 strategy. At last, We come to the conclusion that the mathematical formula is specially designed to cope with the simple 17 strategy and can only perform well in certain circumstances, the strategy learnt from the deep neural network, on the other hand, is adaptive to different strategies.

On the other side, both the DQN method and the mathematical strategy choose a similar strategy when the player has an "A". This is because the "A" can reduce to 1 if the total score exceeds 21, which allows the agent to try to get a higher score and can afford the price of exceeding 21. The similarity between the strategy learnt from the DQN and the mathematical formula also implies the effectiveness of the deep neural network in learning the value of each state.

VI. CONCLUSION

In the end, we fight against each other with our strategy and the score is shown in fig.7. The result shows that for our methods, Huang's method achieves the best score. Hence, his method is chosen to compete with other groups for us.

组名	平均得分
Agent17	372
Tester14	-10.333
Tester15	285.222
Tester16	474.333
Tester18	-26.444
DQN	388.222
Math	-39.333
A2C	83.6667
ddqn	-1527.3

Fig. 7. Result of out competion

REFERENCES

[1] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[2] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[3] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[4] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. 2015.

[5] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[6] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pages 16828–16847. PMLR, 2022.