Monte-Carlo Tree Search with Temporal-Difference Learning: TD-UCT

Rawan Abdulelsadig 35324987

***Abstract*— TD-UCT is a variation of the on-policy planning algorithm MCTS which combines UCB1 and TD learning when performing the tree policy of the tree search, this is done as an attempt to improve the performance of MCTS. TD-UCT uses a linear combination of the estimated Q-value of a state-action pair obtained using TD learning and the estimated Q-value defined as the mean backed-up reward, the linear combination is controlled using a variable w_{td}. In this paper, this method was investigated for the optimal value of w_{td} when applied to *gym's* Frozen Lake game. It was found that $w_{td} = 1$ gives the best performance where all episodes were won when using 600 simulations per move and above, while the average number of steps before winning an episode was found to reach 6.26 ± 0.027 steps at 1000 simulations per move (where 6 is the minimum number of steps to reach the goal).**

I. Introduction

Monte-Carlo methods generally aim to obtain the expected value of a random number by random simulations and observing the results, this method relays on the law of large numbers. Monte-Carlo Tree Search MCTS is a well known on-policy planning algorithm for finding the best decisions to take by randomly sampling from the decision space and building a search tree using the aggregated results of those simulations [1], it is typically used when there is no prior knowledge and it is not possible to obtain complete information about the environment and it is found to attain optimal behavior in spite of that [2].

MCTS consists of four main phases: selection, expansion, simulation and back-propagation. Starting at the root node, the selection phase is where an optimal child selection policy (tree policy) is used recursively until reaching a non-terminal expandable leaf node, then the expansion phase is where the selected node (or multiple selected nodes) is expanded according to the available actions, the rollout phase (also known as the out-of-tree phase) where a default policy is followed until reaching a terminal state many times and obtaining a reward (or rewards along the way) each time and aggregating them, the out-of-tree policy is typically set to randomly select an action at each node of the search tree, the reward obtained is then back-propagated along the path of the expanded nodes [1].

Temporal difference learning is an off-policy reinforcment learning algorithm that allows for gradually learning the utilities of the different states by repeated exposure to the environment and observations of the rewards. It was derived from the Bellman equation which then led to the blooming of most sophisticated reinforcement learning algorithms such as DQNs.

II. Related Work

The most critical phase of MCTS is the selection phase, the quality of the tree policy directly impacts the number of simulation trials needed and hence the time needed to arrive to the optimal solution. All tree policies should aim to minimize the regret; which is the expected loss due to not taking the best action [1]. Selecting nodes along the tree can be thought of as a multi-armed bandit problem [3], a popular policy used in classic reinforcement learning algorithms is the ϵ-greedy policy, where taking the best-valued action is assigned a high probability $1 - \epsilon$ while choosing a random action is assigned a lower probability ϵ [4], the value of ϵ can also be set to decay to 0 across the selection path. Upper Confidence Bounds applied to Trees (UCT) uses the policy UCB1 as a solution to the multi-armed bandit problem, which was found to give a probability of selecting the optimal action that converges to 1 as the number of simulation trials grows to infinity [3]. Many studies have introduced variations of the upper confidence bound policy such as UCB2, UCB1-Normal and UCB1-tuned [5], there is a bayesian take to the multi-armed bandit problem which is based on certain assumptions of the prior distributions across the actions, although it is slower than UCT it was claimed to outperform it [1], Thompson sampling can also be used as a bayesian apporach, the advantage is that it does not require a prior [6] however, there are no enough studies that examine this approach when applied to MCTS. These methods are only a few of the tree policies that can be used in the selection phase.

Different approaches to the default policy performed in the rollout phase were also examined in order to dramatically improve the estimated values of the states and hence improving performance of MCTS, taking uniformly random actions and observing the results is the typical approach, other approaches could include hand-designed role-based rollout policies and heuristics [1], it was stated that a default policy with appropriate domain knowledge can dramatically outperform a uniform random out-of-tree policy [7], some approaches attempt to learn the default policy for even better results [8].

After the search tree's growth is terminated for a decision to be made, the optimal action can be decided based on the estimated reward of taking it and the number of times it was visited during simulation. *Max child* is the the action leading to the highest estimated reward, *Robust child* is the most visited action, *Max-Robust child* is the action that has the highest estimated reward and the most visited during

simulation while *Secure child* is the child action with the highest lower confidence bound [1].

Incorporating TD learning into the basic MCTS framework was found to yield some significant improvements to the well known and widely used tree policy UCB1 [9], the tree search in this case is then called TD-UCT; this work aims to investigate this method when used in Gym's Frozen Lake game which involves stochasticity in the outcomes of actions performed at each step.

III. Methodology

A. The Algorithm: TD-UCT

The basic UCT algorithm utilizes the UCB1 formula to decide which choice of a child node (which action to take) during the tree policy, it also uses random plays as default policy and Max-child as the action to be taken at the end of the tree search (UCB1 with $C = 0$), the UCB1 policy is defined as:

$$UCB1: \underset{v' \in children\ of\ v}{argmax} \frac{Q(v')}{N(v')} + C\sqrt{\frac{2\ln N(v)}{N(v')}} \tag{1}$$

Where $\frac{Q(v')}{N(v')}$ is the mean estimated reward (Q-value) of the child node v', $N(v')$ is the number of times the child node v' was visited and $N(v)$ is the number of times the parent node v was visited. C is a weighting parameter that specifies the importance of the confidence bound $\sqrt{\frac{2\ln N(v)}{N(v')}}$ compared to the Q-value, $C = 1$ indicates that both clauses are equally important.
Algorithm 1 shows the UCT algorithm in detail.

TD learning is used to improve the estimation of the Q-value of any node v, the basic TD learning update is defined as:

$$V_{td}(s,a) \longleftarrow (1-\alpha)V_{td}(s,a) + \alpha(R(s) + \gamma \underset{a' \in A}{max} V_{td}(s',a')) \tag{2}$$

Where A is the set of possible actions at state s, α is the learning rate and γ is the discount factor which takes into account how far away is the leaf node from the root node, allowing the effect of the estimated long-term reward to degrade as the distance from the root increases.
In MCTS tree, a node v is associated with a pair of state and a certain action taken at that state, hence a state that can be arrived to from two different paths has two different tree nodes, and therefore, $V(s,a) \equiv V(v)$ which gives the update:

$$V_{td}(v) \longleftarrow (1-\alpha)V_{td}(v) + \alpha(R(s) + \gamma \underset{v' \in children\ of\ v}{max} V_{td}(v')) \tag{3}$$

The term $\left(\underset{v' \in\ children\ of\ v}{max} V_{td}(v')\right)$ is en estimation of the long-term "goodness" of node v which is compatible with the objective of the rollout phase of MCTS, and following the default policy until completion then observing the reward. This is why it is suitable to use the estimated reward obtained from the rollout phase which is denoted by Δ to replace this term giving us the following update:

$$V_{td}(v_{leaf}) \longleftarrow (1-\alpha)V_{td}(v_{leaf}) + \alpha(R(s) + \gamma\Delta) \tag{4}$$

In order to take into account the number of steps that were taken in the rollout phase into the estimation of the value of node v, where node v_{leaf} is the leaf node from which the default policy was started; the discount factor γ is set to decay

Algorithm 1: UCT [1]

Result: An action to take at state s_0.

Function `MCTSearch`(s_0)**:**
- $v_0 \longleftarrow$ MCTSNode(s_0)
- **while** *within computational budget* **do**
  - $v_1 \longleftarrow$ TreePolicy(v_0)
  - $\Delta \longleftarrow$ DefaultPolicy(v_1)
  - Backup(v_1, Δ)
- **end**
- **return** BestChild(v_0,0)

Function `TreePolicy`(v)**:**
- **while** *s(v) is non-terminal* **do**
  - **if** *v not fully expanded* **then**
    - **return** Expand(v)
  - **else**
    - $v \longleftarrow$ BestChild(v , C_p)
  - **end**
- **end**
- **return** v

Function `Expand`(v)**:**
- choose $a \in$ untried action from A(s(v))
- $v' \longleftarrow$ MCTSNode($s(v)$, a)
- **return** v'

Function `BestChild`$(v,\ C)$**:**
- **return** $\underset{v' \in children\ of\ v}{argmax} \left(\frac{Q(v')}{N(v')} + C\sqrt{\frac{2\ln N(v)}{N(v')}}\right)$

Function `DefaultPolicy`(v)**:**
- **while** $s(v)$ *is non-terminal* **do**
  - choose $a \in A(s(\text{v}))$ uniformly at random
  - $Reward \longleftarrow$ TakeAction($s(v)$, a)
- **end**
- **return** $Reward$

Function `Backup`$(v,\ \Delta)$**:**
- **while** v *is not null* **do**
  - $N(v) \longleftarrow N(v) + 1$
  - $Q(v) \longleftarrow Q(v) + \Delta$
  - $v \longleftarrow$ parent of v
- **end**

according to this number of steps, the update is then:

$$V_{td}(v_{leaf}) \longleftarrow (1-\alpha)V_{td}(v_{leaf}) + \alpha(R(s) + \gamma^S \Delta) \quad (5)$$

Following the update of the leaf node v_{leaf}, the updates of the parent nodes during backpropagation are defined as:

$$V_{td}(v) \longleftarrow (1-\alpha)V_{td}(v) + \alpha(R(s) + \gamma V_{td}(v_{child})) \quad (6)$$

When making a decision about the best child node to choose; the UCB1 formula is modified to include a linear combination of $V_{td}(v)$ and $\frac{Q(v)}{N(v)}$ governed by an additional parameter w_{td} which determines the amount of both quantities by having a value between 0 and 1, this linear combination is used as the mean estimated reward while the confidence bound term is kept as it is. The maximization objective when choosing the best child at node v is then as follows:

$$\underset{v' \in children\ of\ v}{argmax} \left(w_{td} V_{td}(v') + (1-w_{td}) \frac{Q(v')}{N(v')} + C\sqrt{\frac{2\ln N(v)}{N(v')}} \right) \quad (7)$$

Algorithm 2 below shows the modified UCT procedures that incorporates TD learning to UCT search.

Algorithm 2: TD-UCT Modifications

Function `DefaultPolicy`(v)**:**
- define S as the number of steps to the terminal node
- $S \longleftarrow 0$
- **while** $s(v)$ *is non-terminal* **do**
  - choose $a \in A(s(\text{v}))$ uniformly at random
  - $Reward \longleftarrow$ TakeAction($s(v)$, a)
  - $S \longleftarrow S + 1$
- **end**
- **return** $Reward, S$

Function `Backup`$(v,\ \Delta,\ S,\ \alpha,\ \gamma)$**:**
- **while** v *is not null* **do**
  - $N(v) \longleftarrow N(v) + 1$
  - $Q(v) \longleftarrow Q(v) + \Delta$
  - **if** v *is a leaf node* **then**
    - $V_{td}(v) \longleftarrow (1-\alpha)V_{td}(v) + \alpha(Reward + \gamma^S \Delta)$
  - **else**
    - $V_{td}(v) \longleftarrow (1-\alpha)V_{td}(v) + \alpha(Reward + \gamma V_{td}(v_{child}))$
  - **end**
  - $v \longleftarrow$ parent of v
- **end**

Function `BestChild`$(v,\ C,\ w_{td})$**:**
- **return** $\underset{v' \in children\ of\ v}{argmax} \left(w_{td} V_{td}(v') + (1-w_{td}) \frac{Q(v')}{N(v')} + C\sqrt{\frac{2\ln N(v)}{N(v')}} \right)$

B. Experiments

Algorithm 2 shows 4 parameters to tune α, γ, w_{td} and C, C was fixed to the value of 1 with the intuition of needing equal proportions of exploration and exploitation when evaluating the possible moves, the focus of the experiments was to find the optimal values of the other three parameters first, then evaluating the performance of the best linear combination ($0 < w_{td} < 1$) against using only UCB1 ($w_{td} = 0$) or only TD learning with the confidence bound ($w_{td} = 1$) as the number of simulations per move increases.
The evaluation measures used are the number of game episodes won out of the total number of episodes played using each parameter combination, and the average number of steps taken until reaching the goal (per "won" episode).

In order to find the best values of α, γ and w_{td} for the game, 500 episodes were played where 500 simulation iterations per move was performed, then the following experiment were done:

1) α: α was investigated for its optimal value when there is no linear combination of V_{td} and Q ($w_{td} = 1$) since this parameter only influences the TD part of the estimation and observing its pure effect on the quality of estimation is the main concern, γ was set to 0.95 to have a slight discount factor since this value is typically used. α was then set to vary from 0.001 to 0.5 (with unequal spaces) and the performance was then evaluated.

2) γ: γ was investigated for its optimal value when there is no linear combination ($w_{td} = 1$) since this parameter only influences the TD part of the estimation as well. α was set to the best value observed from the results of experiment III-B1. γ was then set to vary from 0.05 to 0.95 and the performance was evaluated.

3) w_{td}: After obtaining the best values of γ and α for an effective TD Q-value estimation, the optimal linear combination was then explored by varying w_{td} between 0.0 and 1.0, the performance was then evaluated.

The optimal values of α, γ and w_{td} are then used to evaluate its performance against using a pure UCT policy and a pure TD policy (aided with the confidence bound term) as the number of simulations per move vary from 100 to 1000, the results are then recorder and statistically evaluated.

C. The Game: Frozen Lake

Frozen lake https://gym.openai.com/envs/FrozenLake-v0 is an environment provided in *OpenAI's gym* package that was developed for training and testing reinforcement learning algorithms in python, winning frozen lake is about reaching a goal and avoiding thin ice (holes) in the ground. However, moving around involves uncertainty regarding the next state due to the ground being slippery; choosing to move to a certain direction has only 0.333% chance of actually moving to that direction, this makes MCTS a tempting algorithm of choice since it was

originally developed for Markov decision processes (MDPs) where reaching a next state has a certain probability.
Frozen Lake's grid world is either 4×4 or 8×8, the number of thin-iced locations are almost doubled in the 8×8 option which makes it even harder to solve. In this work, the dimensions of the grid world were set to be 4×4 in order to avoid the need for a more complex default policy, the minimum number of steps to reach the goal in this grid world is 6 steps.

IV. Results and Discussion

A. Finding the optimal values for the parameters:

In all of the following experiments; the number of wins was generally 500/500 in almost all of the trials due to the choice of the number of simulations per move (500 simulations per move), however the mean number of steps before winning was varying depending on the parameter of concern.

Starting with the results of experiment III-B1 shown in figure 1.

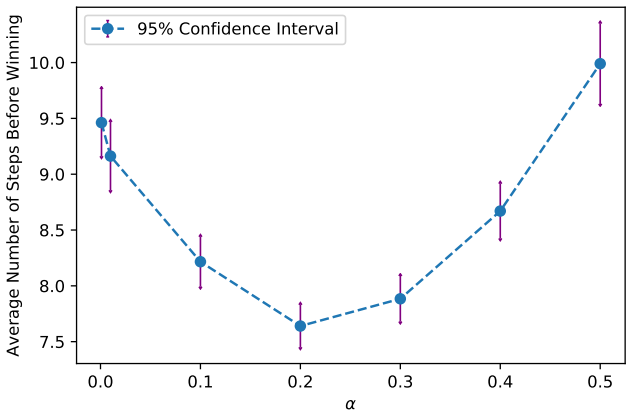

Fig. 1. The average number of steps before winning an episode as the value of α varies, showing the 95% confidence intervals.

It is clear that the value of $\alpha = 0.2$ results in the minimum number of steps before wining for this experiment set-up (7.64 ± 0.097).

Now using the results obtained and attempting to move further down by varying the value of γ as specified in III-B2, the number of steps before winning was varying as shown in figure 2.

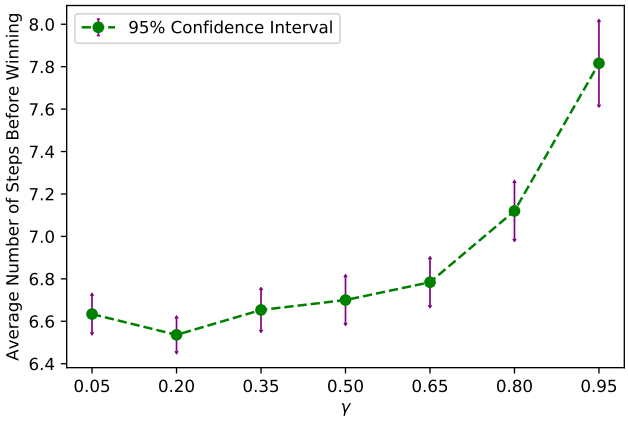

Fig. 2. The average number of steps before winning an episode as the value of γ varies, showing the 95% confidence intervals.

It also shows that as the value of γ was decreasing, the average number of steps before wining also decreased. $\gamma = 0.2$ seems to be the best value for this experimental set-up resulting in an average number of steps before wining equal to 6.536 ± 0.039.

Utilizing the previous results ($\alpha \longleftarrow 0.2$ and $\gamma \longleftarrow 0.2$) and varying w_{td} as described in III-B3, the results were found to be as shown in figure 3.

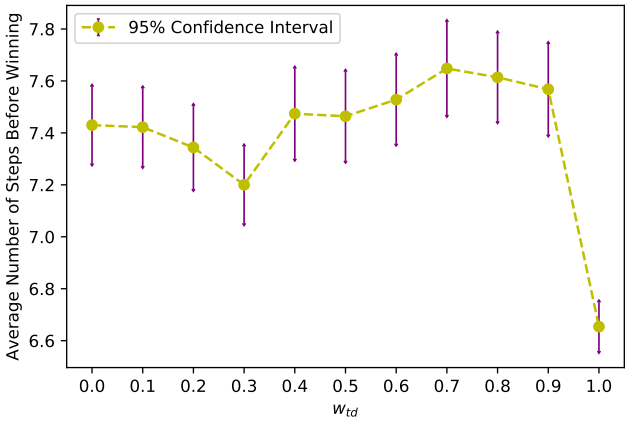

Fig. 3. The average number of steps before winning an episode as the value of w_{td} varies, showing the 95% confidence intervals.

The resulting graph shows some irregular changes to the average number of steps before winning as the value of w_{td} changes between 0 and 1. Considering the values $0 < w_{td} < 1$; $w_{td} = 0.3$ seems to give a local minimum value of 7.2 ± 0.076.

However, $w_{td} = 1$ seems to result in a significantly lower average number of steps before winning (6.654 ± 0.048), this result can suggest that a state's Q-value estimation using TD learning ($V_{td}(v)$) solely is more effective than using a linearly constructed estimation value from $\frac{Q(v)}{N(v)}$ and $V_{td}(v)$, this might be due to the fact that the choice of the parameters α and γ were optimized for the value $w_{td} = 1$, however the

learning process of $V_{td}(v)$ for each state is independant of the resulting combined estimation and hence it is updated in the same manner regardless of the value of w_{td}, therefore this explanation is unlikely to be correct.

B. Statistically evaluating the performance of TD-UCT against UCT:

After finding the optimal values of the parameters and assigning those values $\alpha \longleftarrow 0.2, \gamma \longleftarrow 0.2$ and $w_{td} \longleftarrow 0.3$, the resulting search TD-UCT policy is compared with the original UCT policy ($w_{td} = 0$) and with $w_{td} = 1$ as well since it was found to outperform them in figure 3.
The number of won episodes out of 500 episodes and the average number of steps before winning as the number of simulations per move varies between 100 and 1000 was recorded and shown in figures 4 and 5, respectively.

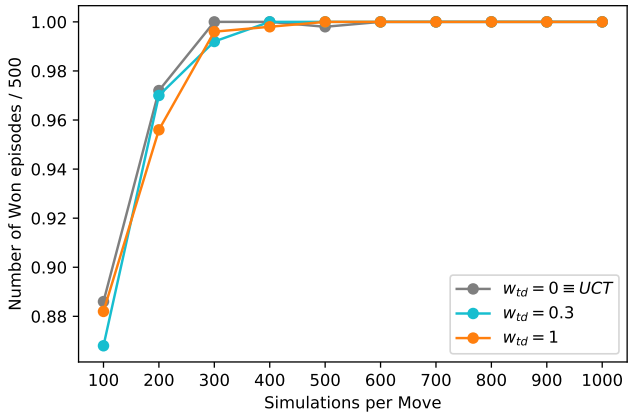

Fig. 4. The number of won episodes out of 500 as the number of simulations per move varies, showing the 95% confidence intervals.

Starting from 300 simulations per move, most episodes were won. While all game episodes were won and the goal was successfully reached when the number of simulations per move was 600 and above in all set-ups.

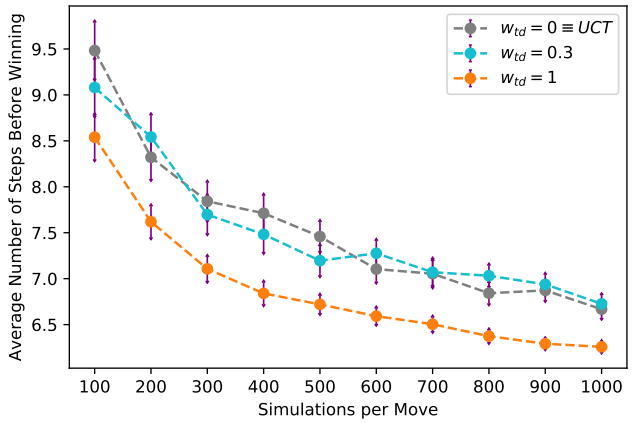

Fig. 5. The average number of steps before winning an episode as the number of simulations per move varies, showing the 95% confidence intervals.

It is clear to see that $w_{td} = 1$ consistently outperforms $w_{td} = 0.0$ and $w_{td} = 0.3$, a paired t-test confirms this finding as well giving a p-value of 9.44×10^{-7} and 1.09×10^{-7}, respectively, which are well below 0.05; suggesting a significant difference (at 5% level) between their performance regarding the average number of steps before winning. A paired t-test between $w_{td} = 0.0$ and $w_{td} = 0.3$ gave a p-value of 0.653 which is greater than 0.05 showing that there is no significant difference (at 5% level) in their performance regarding the average number of steps before winning, and that they have almost the same effect on the choices of moves in the game.

V. Conclusion

MCTS and TD learning are two of the most widely used reinforcement learning algorithms, which led to the emergence of a vast amount of variations that attempt to improve their performance.
This paper investigated combining the two techniques by incorporating TD learning to the tree policy of a MCTS that uses UCB1 multi-armed bandit policy, resulting in TD-UCT tree search. Parameters were investigated and tuned to find the best combination when trying to win *gym's* Frozen Lake game. TD-UCT involves a possible linear combination of the Q-value estimated using TD learning and the Q-value estimated as the mean simulated reward at each state, the intuitive expectation of the results was that the linear combination of the optimized TD learning and the mean simulated reward would result in an average number of steps before winning lower than using each of them separately; however the results showed that using only the TD learned Q-value estimation along with the confidence bound of the UCT policy (i.e. $w_{td} = 1$) is the best set-up for this game giving 6.26 ± 0.027 steps before winning (where 6 is the minimum number of steps required to reach the goal).

References

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[4] T. Vodopivec, S. Samothrakis, and B. Ster, "On monte carlo tree search and reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 60, pp. 881–936, 2017.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[6] A. Slivkins, "Introduction to multi-armed bandits," *arXiv preprint arXiv:1904.07272*, 2019.

[7] D. Silver and G. Tesauro, "Monte-carlo simulation balancing," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 945–952, 2009.

[8] S. Gelly, Y. Wang, O. Teytaud, M. U. Patterns, and P. Tao, "Modification of uct with patterns in monte-carlo go," 2006.

[9] T. Vodopivec and B. Šter, "Enhancing upper confidence bounds for trees with temporal difference values," in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, IEEE, 2014.

ACKNOWLEDGEMENTS

Some of the resources used to inspire and guide this work were:

- David Silver's lectures: https://www.davidsilver.uk/teaching/
- DeepLizard's reinforcement learning course: https://deeplizard.com/learn/playlist/PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv
- The github repository: https://github.com/yandexdataschool/Practical_RL