CSE 546: REINFORCEMENT LEARNING

Final project: MARL with LUDO

Instructor: Prof. Alina Vereshchaka

By:

Nabeel Khan (#50419151)

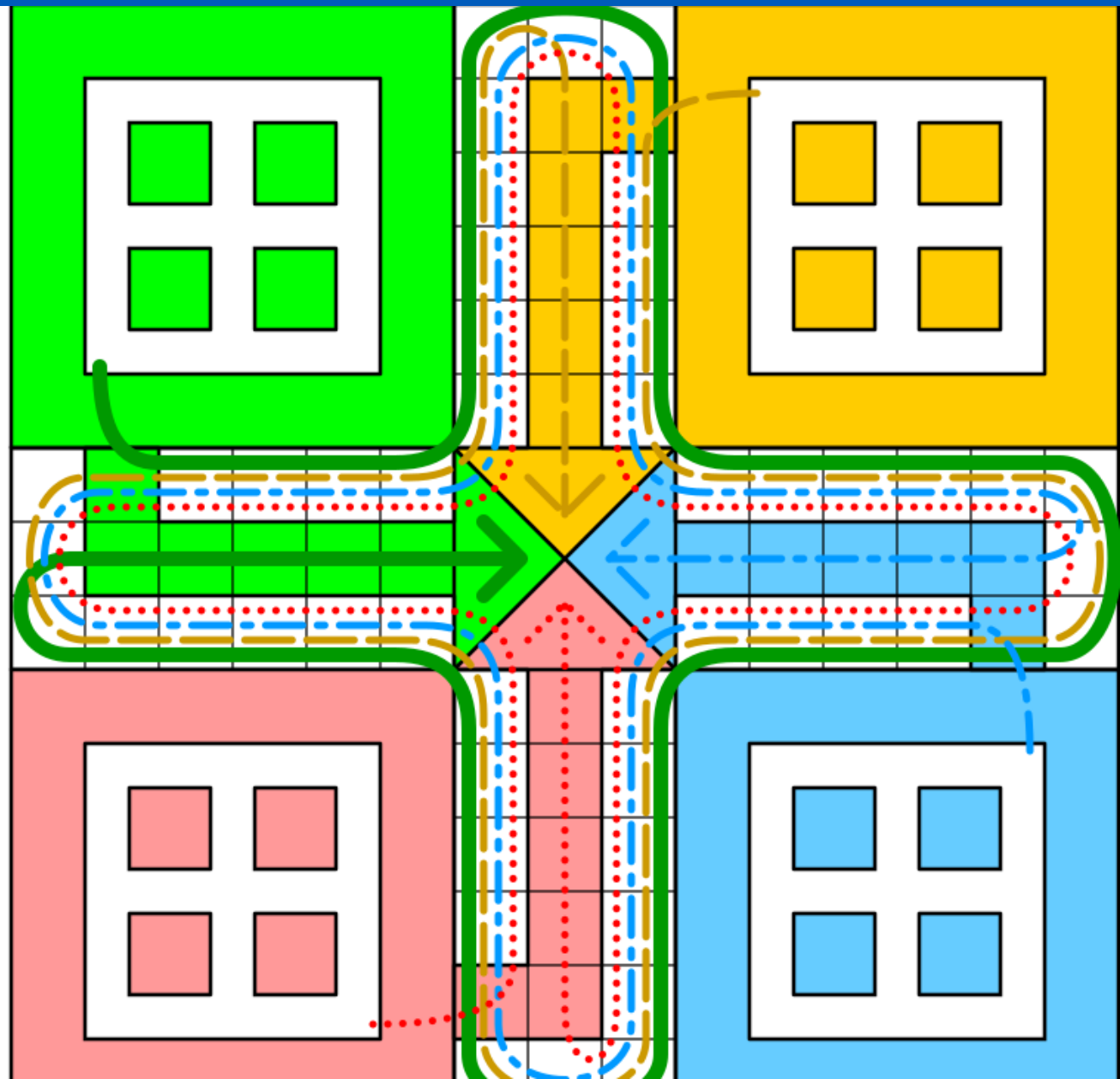
Vivek Vijay Wadegaonkar (#50372653)

Babar Ahmed Shaik (#50416252)

University at Buffalo The State University of New York

Ludo

Ludo is a 2 to 4 player strategic board game in which players race their four pieces from start to finish based on a single die roll.

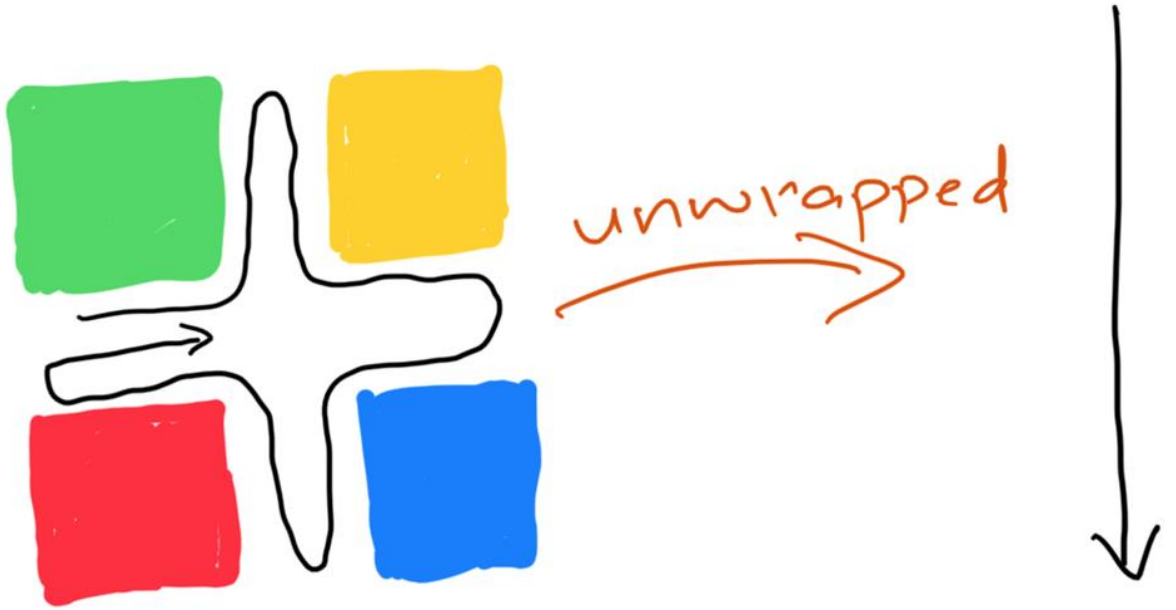


Objective

- We wanted to create a scalable multi agent environment for Ludo.
- To use Tabular methods with the downscaled version of Ludo, & actor critic for the full-scale version.
- Design the reward structures, to accelerate learning.

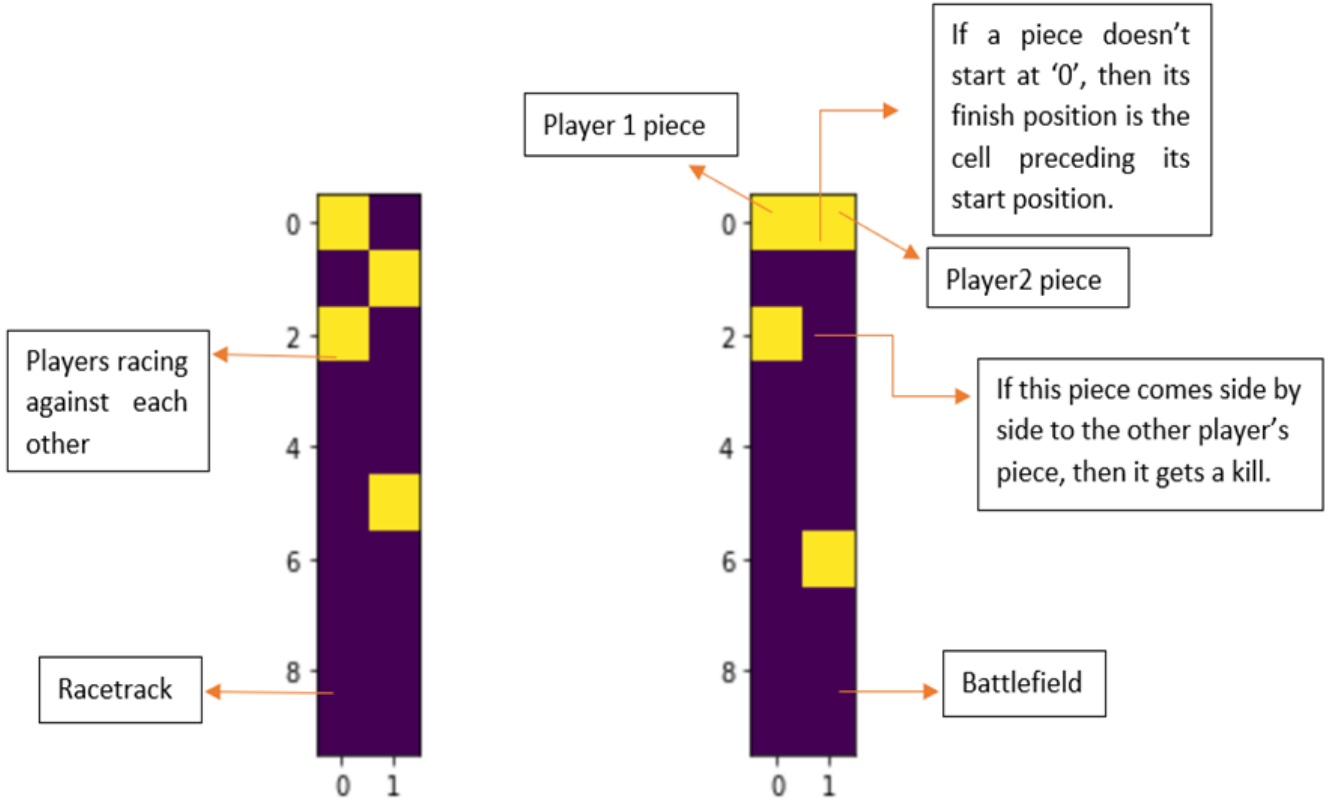
Simplified Ludo

- As Ludo has a circular racetrack, to simplify our environment, we unwrapped the Ludo board/racetrack to a one-dimensional.
- This made scaling down and tracking movements and kills a lot easier.
- This is a variation of Grid world.
- Simplification helped in reducing number of states to implement tabular methods.



Racetrack and Battlefield

- Visualization is done using 2 grids (Racetrack and Battlefield)
- Racetrack allows us to track progress, while battlefield gives the relative positions.
- Since Ludo is circular with different start positions, we incorporated shifts in the start positions accordingly.



Racetrack & Battlefield ground

Open AI Gym Code structure

- We have created ***reset, step, render*** functions and in addition to that we have created ***check_move, move*** and ***check_kill*** function to calculate moves for the player's pieces and check if any piece is getting killed by other player's piece.
- All safe spots on racetrack are at : [0, 5, 6, 11]
- In ***check_kill*** function first we check if the piece is at an unsafe location. If yes, then the pursuing another player's piece can kill it if it comes to the same position.
- Actual reward is received if the piece finishes the race.

Evaluation & Training

- For **all evaluations** we pitched all our agents against random agent for 1000 games and calculated the win% as the performance for the agents.
- We trained our agents using self-play, with analytical agent, and with random agent.

Q-Learning:

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

Q Learning States

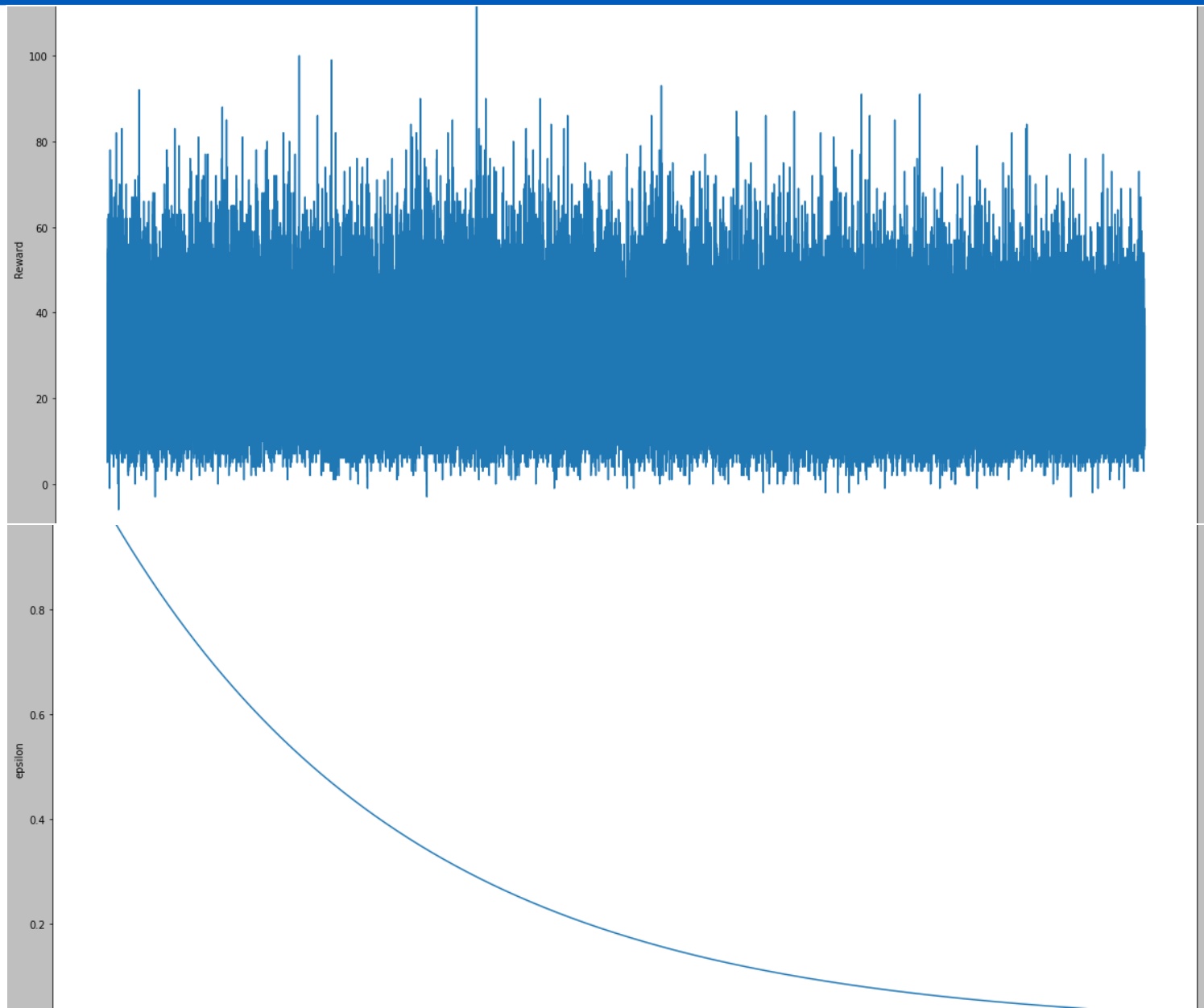
- State = $(length\ of\ board+2)n*(num\ of\ faces\ on\ dice)*(num\ of\ pices)$, where $n=num\ of\ players*num\ of\ pices$
- Small Env States = ((12^4)×3×2) used with Q Learning & evaluation of analytical agent
- Large Env States = ((42^16)×4×8) used with Actor critic & evaluation of analytical agent
- 3-piece variant = ((10^6)×2×3) used with Q Learning & evaluation of analytical agent

Q-Learning Agent (Self play)

- Here 2 versions of the agent are playing Ludo.
- Both versions utilize the same q-table.
- This is achieved by shuffling the state position vector in a manner such that the position vector is from the perspective of the player making move, and as if the player making move is player 1.

Results

- After training with Self play, Upon running for 100000 episodes with learning rate 0.2 and discount factor 0.9,
- We observed that the Q-agent was winning against random agent 60% of the time against random agent.
- It's not possible to win every time when we play against the random agent as there is randomness involved. (Max possible ~70%)



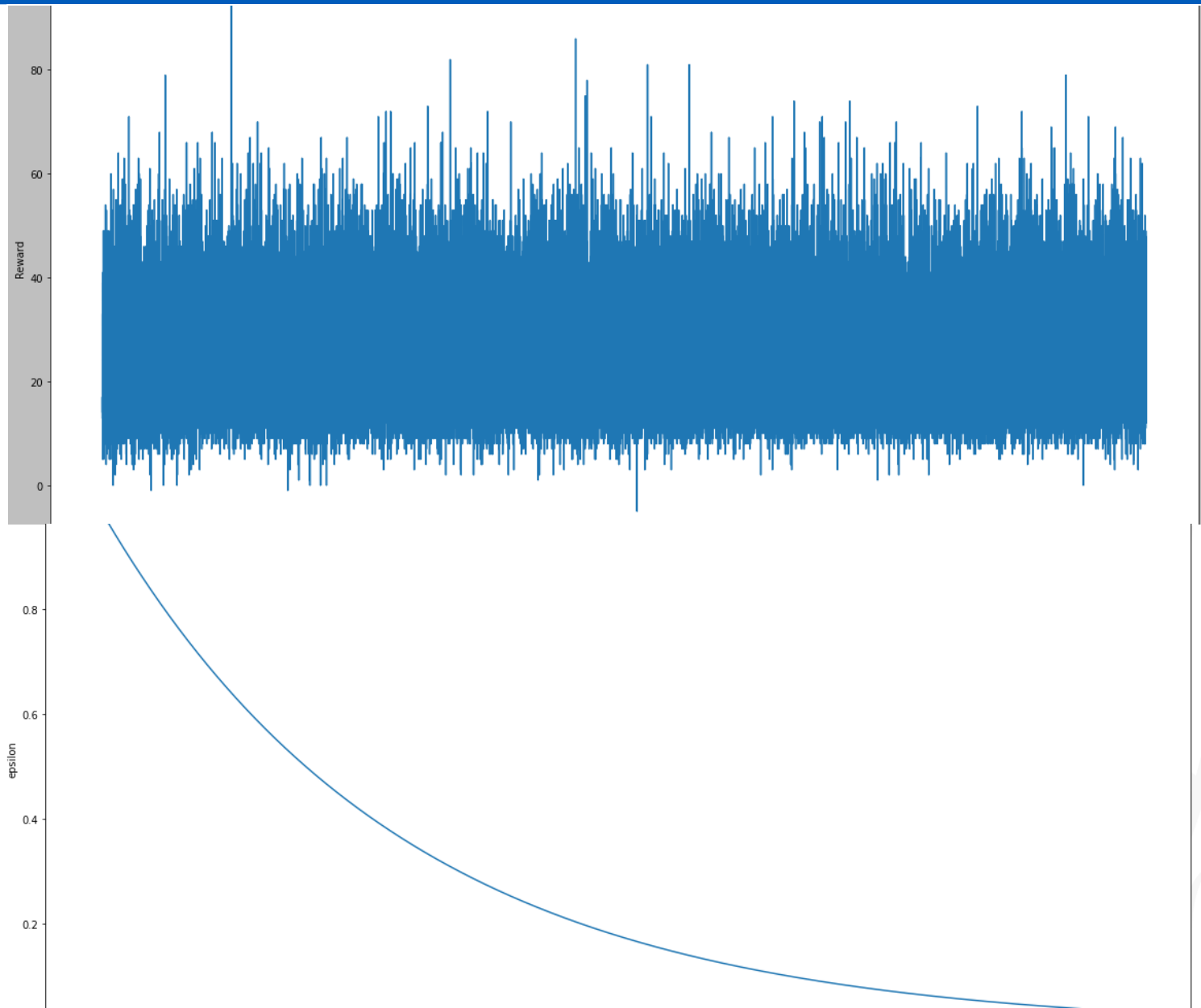
Analytical Agent

This is the rule-based version of the agent. We hardcoded the policy by calculating a score for each possible action and then choosing action with max score.

Results

After training the Q-learning Agent with the analytical agent,

When we pitched this Q learning agent vs Random agent, it won 57% of the times (in 1000 games)



Similarly,

When we pitted Analytical Agent with Random Agent, the Analytical 70% of the time.

On Large Environment,

- Analytical Vs Random – 98%
- After Training with Analytical

  Q Agent VS Random Agent– 57%
- After Training with Self

  Q Agent VS Random Agent – 59%
- After Training with Random

  Q Agent VS Random Agent – 62%

Actor Critic Algorithm

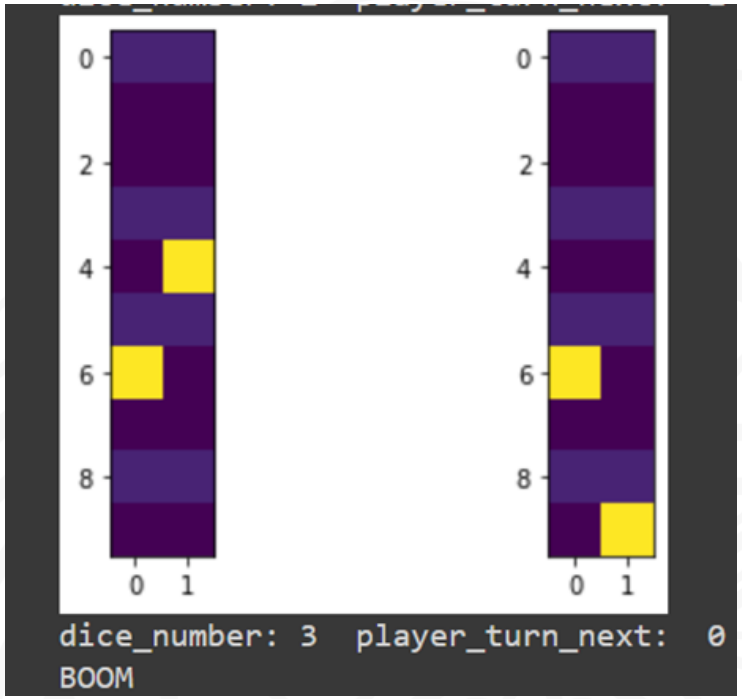
- We have used Advantage Actor-Critic (A2C) algorithm.
- Critic uses policy evaluation (e.g., MC or TD learning) to estimate $Q\pi(s, a)$, $A\pi(s, a)$ or $V\pi(s)$.
- Here, we have implemented Temporal Difference Advantage.

Parameters

- Discount factor = 0.99
- Num of episodes = 10000
- Actor Learning rate = 0.00001
- Critic learning rate = 0.0005
- Loss function = MSELoss
- Layer1 Size = (State size = 18 ,64)
- Layer2 Size (actor layer) = (64 ,128)
- Layer3 Size (critic layer) = (128,128)
- Layer4 Size (128, no of actions= 1)

Performance of Actor critic

- Analytical agent with mathematical rules, when pithed with Random Agent performed well achieving 99%-win score.
- Our Actor Critic agent's performance when trained with analytical agent achieved around 64-66% win score.



Here, the visualization shows when a piece kills enemy's piece, the environment displays a message 'BOOM'.

Summary

- Q-Learning agent performed better than Actor Critic agent when benchmarked with analytical agent.
- Overall win score achieved by Actor critic agent is more.
- While training and testing, we observed that as we increased the number of episodes for training, the performance of the model increased, and it achieved better win score.

Thank You