**Introduction**

Bridge hand is a set of 13 card held by one player. The main goal of the following task is to predict a retrieved bridge hand, which could deliver more information than retrieving a single card alone. Doomed to be ignored in poker comparison between different hands taking into account their power, now the idea was extended to create learning frameworks and to attempt to solve one based on deep unfolded neural networks. Both approaches resulted in test network runs with similar efficiency.

**Bridge color implementation**

It has been already proven that computer can do better than human beings on the principle of math. Appearance of other player playing very complicated game makes even the most preeminent player lose some points. This is where learning frameworks start to play an essential role. Each single card retrieved bridge color (D) can compute, how many tricks each pair is able to collect depending on trump, but it takes some time while it delivers costs of all cards.

**Hand representation**

Each standard bridge problem contains a set of 13 cards of 4 suits (each suit has 13 cards). To choose a group of 13 cards from 52, to compute the number of tricks that each player can take with a partner in the range of 0-13, it is necessary to describe bridge hands by a set of numbers: 13 integer values from 0 to 3, which represents a number of cards in each suit. 1 is the input part of the hand, 2 is the other suit, 3 in the other suit, plus the trump. For example, a hand consisting of these cards: ♠KQJ75, ♦AKT6Q, ♣Q62, ♢5 is represented by the vector (3,2,2,0,0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0). Retrieving a bridge color, networks were able to compute trick multitudes less than thins IDES with average gain in hand 1.

**Problem description**

In standard bridge, there are four players in opposition. One hand is held by one player, after which the other players make decisions to pass or bid. The game is over when a trick which is won by the highest bid is taken. The bidding continues until the player who has made the best bid or who has been forced to do so by the rules. The optimal bidding strategy is to win the first trick, since each player is hoping to win the bidding war and to play a hand with a partner. Reaching the bidding is the major issue of this framework. A better understanding of the bidding war and high trick multitudes leads to less time for playing tricks and can result in getting less reward. The following code (Listing 1) shows the use of the environment with random agent actions without exploration training runs.

**Environment description**

A large number of algorithms working in bidding is not to end the game of bridge playing using reinforcement learning. In order to compare three agents, environment monitoring of the bidding was encoded in Python. The implemented environment has the following features:

- The environment is designed for four agents.
- The environment has both a graphical and console interface. The graphs option is well suited for demonstration purposes. However, for the algorithms development it is better to use the console interface.
- The environment is designed for four agents.

**Environment test**

Environment test has a random starting point with equal reward. Agent's score is calculated using simple formula: Agent score = 50 + 10 * (actual score - expected score). Agent score is punished or rewarded according to the difference between the actual score and the expected score. The plan is to use the OpenAI Gym library interface, which is an open source toolkit for developing and comparing reinforcement learning algorithms.

**Application of reinforcement learning to bridge bidding**

Reinforcement learning is based on an agent playing in trick by trick and receiving points or punishment from environment. When agent performs an action, it gets rewarded or punished. The reward is usually small and the total reward grows over time. Both advantages and disadvantages of computing for calculating of Q-values in the priority of an agent step by step. Instead of using the temporal-difference learning (Q-learning) proposed in 1989, we use a Deep Q-Network (DQN) which allows better performance in high-dimensional environments. The agent uses a neural network to map states to Q-values and acts according to the greedy action. The DQN is a deep neural network that learns to associate states to Q-values and acts according to the greedy action. The Q-values are updated using a target network that is frozen and not updated during training. This allows the agent to learn and generalize better.

**Reinforcement learning**

Reinforcement learning is a type of machine learning where an agent learns to make decisions in an environment to maximize some notion of reward. It is based on the idea of trial and error, where the agent explores the environment, collects data, and learns to make better decisions over time. The agent receives feedback in the form of a reward signal, which indicates how well it is doing. The reward signal is used to update the agent's policy, which is a function that maps states to actions. The goal of the agent is to maximize the expected cumulative reward over time. Reinforcement learning is widely used in various applications, such as video games, robotics, and autonomous vehicles.