DESCRIPTION OF THE ALGORITHM

The algorithm used to solve the Tennis environment is the DDPG algorithm.

It is an off-policy algorithm for deep reinforcement learning and it is one of the first algorithms to allow to operate in environments with continuous actions.

In order to make this possible it uses the Actor Critic paradigm. In other words it contains 2 nets, one that acts (the Policy) and another one (the Critic) that evaluates the values of these actions.

More specifically, the Policy net is a net with input size the size of the observation space and output size the dimension of the action space . It selects, for a given state s, an action that our Policy "thinks" could be a reasonable behaviour in the presence of s.
On the other side the Critic takes the selected state s and action a and evaluates them giving back a value $Q(s, a)$ that is an estimate of the value of the couple (s, a).

During the training of the algorithm we play with the environment following the Policy, and at the same time we feed a buffer that stores experience, i.e. tuples of the form $(state, action, reward, next\_state, is\_done)$.
With a certain regularity this experience is batched and is used to improve the Critic in a DQN style (see the explanation in the value based repository for further details). The Policy, instead, uses the estimates from the Critic to improve its decision making.
To be more precise, for what concerns the Policy, if we batched just to tuples $(s_1, a_1, r_1, is\_done_1)$, $(s_2, a_2, r_2.is\_done_2)$ then in its updates the Policy would try to maximise

$$Q(s_1, a_1) + Q(s_2, a_2),$$

where $Q(s_1, a_1) + Q(s_2, a_2)$ are the outputs of the Critic.

When we reach a reasonable result the process is stopped. Notice that a noise to the output of the Policy can be added in order to improve exploration and that we used the same net to train both players at the same time.

DESCRIPTION OF THE NETS

The Policy net contains three linear layers with added ReLU functions. The hidden sizes contains respectively 256 and 128 nodes. The input size is the size of the observation space while the output size is the size of the action space.

The Critic net contains three layers. The first layer has input size the size of the observation space and hidden size 256. The second layer has input size 256 plus the size of the action space while an output size of 128 (we concatenate the action space and the previous hidden nodes and feed it to the second layer. We do this, instead of feeding action and state at the same time at the first layer in order to prevent the state space, which it is usually much bigger than the action space, of having a predominant factor). We then use a third layer with dimensions 128 and 1.

DESCRIPTION OF THE LEARNING PARAMETERS

$BUFFER\ SIZE = int(1e5)$
$BATCH\ SIZE = 128$
$GAMMA = 0.99$
$TAU = 1e-2$
$LR\ ACTOR = 1e-4$
$LR\ CRITIC = 1e-3$
$WEIGHT\ DECAY = 0$
$UPDATE\ FREQUENCY = 1$
$NUM\ UPDATES = 1$

Most of the parameters are self explanatory. Let's just indulge over
$UPDATE\ FREQUENCY$: which decides when update Policy and Critic. Which in our case is every time we collect a new piece of information (a tuple $(s, a, r, is\ done)$) $NUM\ UPDATES$: which decides how many times we update the nets.

TABLE

The table of the plotting rewards is included inside the Jupyter Notebook. The number of episodes to solve the environment varies between 300 episodes and 500 episodes.

FUTURE IDEAS

The most straightforward idea is to implement D4PG. One has to add prioritized experience replay and a distributed factor to the algorithm. Also, the learning part depends on the distributional algorithm of Bellamere.
Easier ideas are to investigate different architectures for nets and different hyperparameters which might be more optimal than ours.