**GitHub Link:** [**https://github.com/kevinausrain/reinforment-learning-work**](https://github.com/kevinausrain/reinforment-learning-work)

As this project is firstly developed as common project, while I find Jupyter notebook is required when submitting, so this file



Is all you need to use, but following content explains core parameters of this project which is worthy to browse

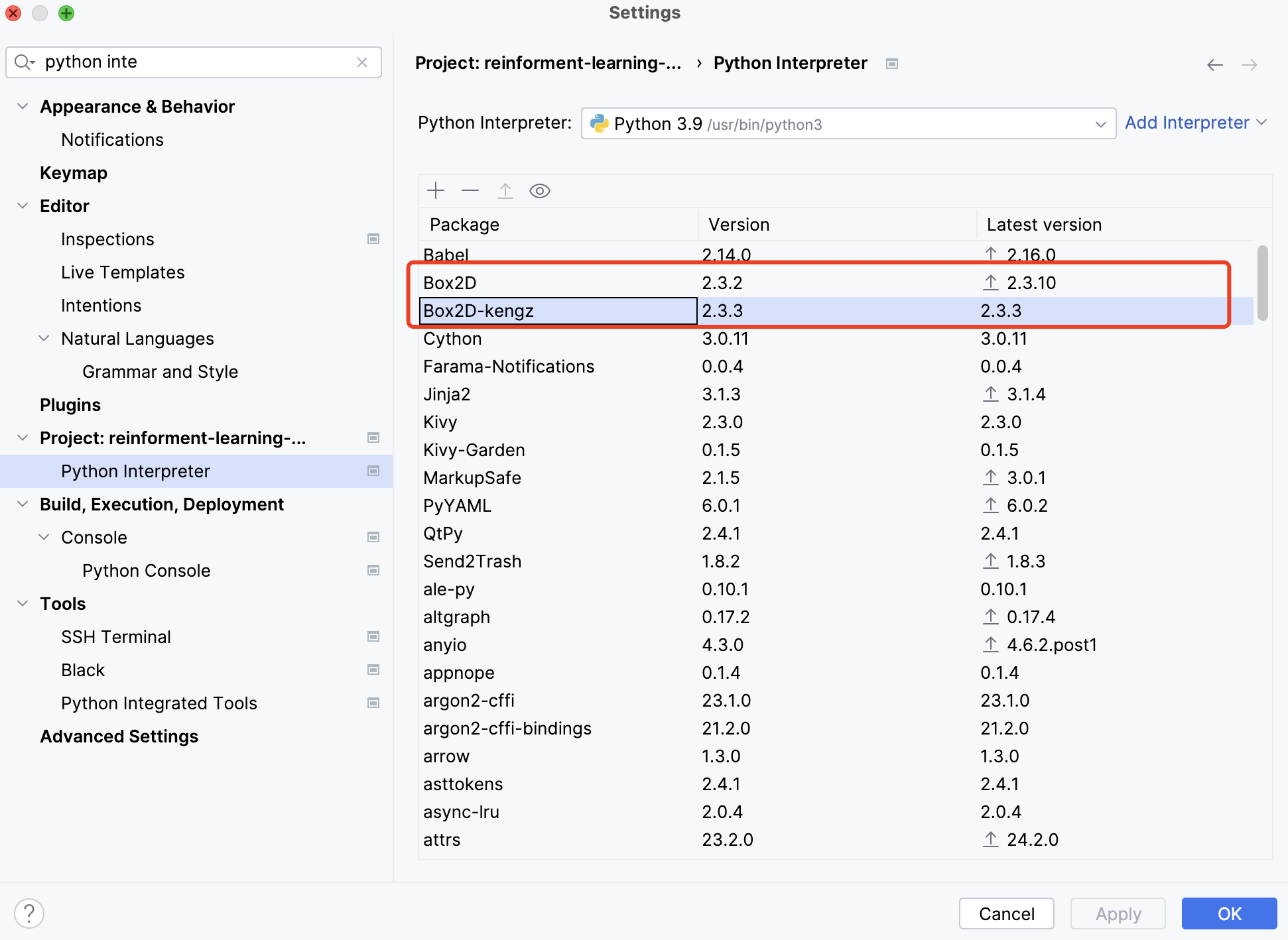
To run environment in Box2D on macOS, the following package are required for enable 2D graphic:

brew install gcc@9

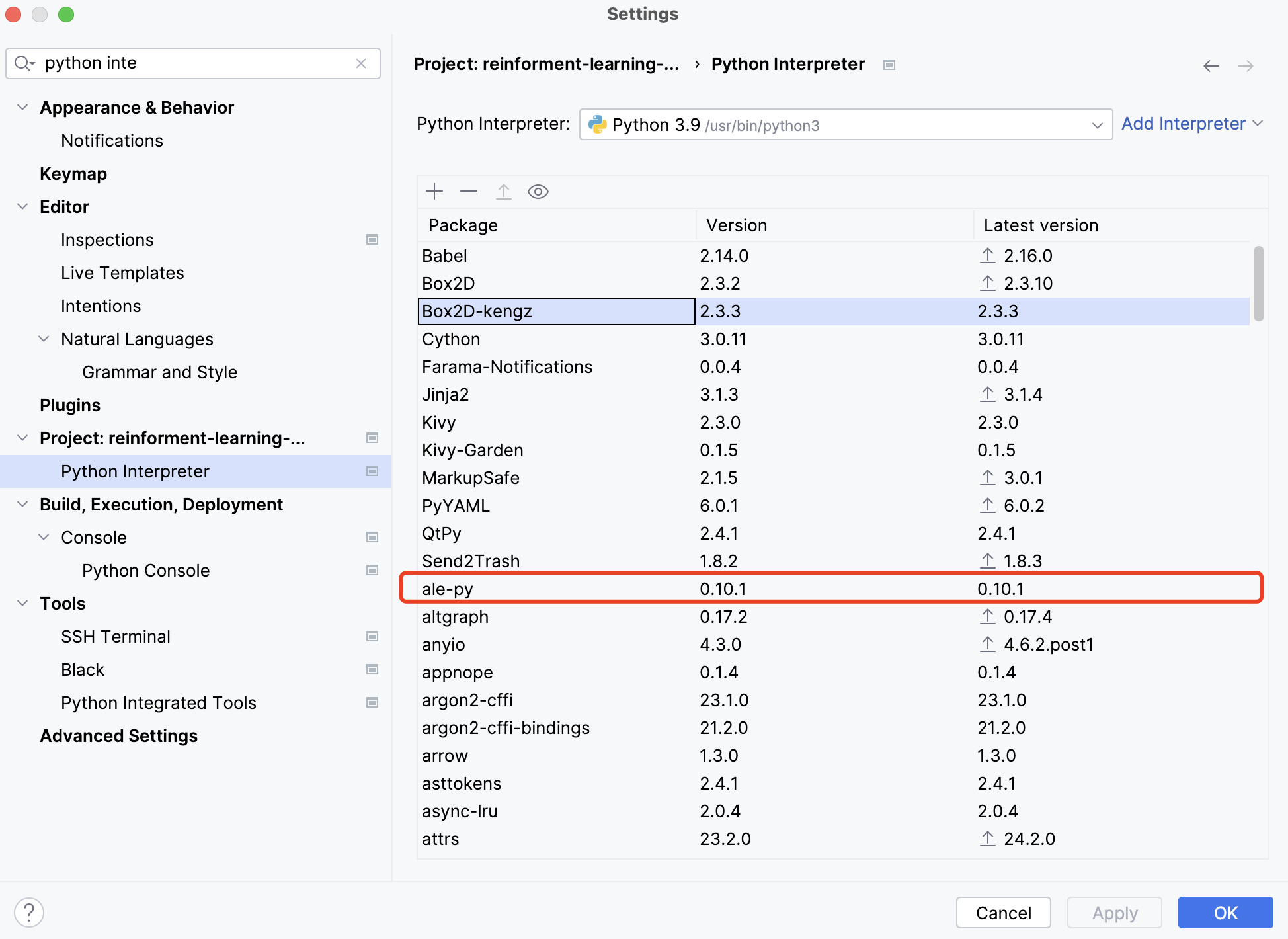
brew install swig



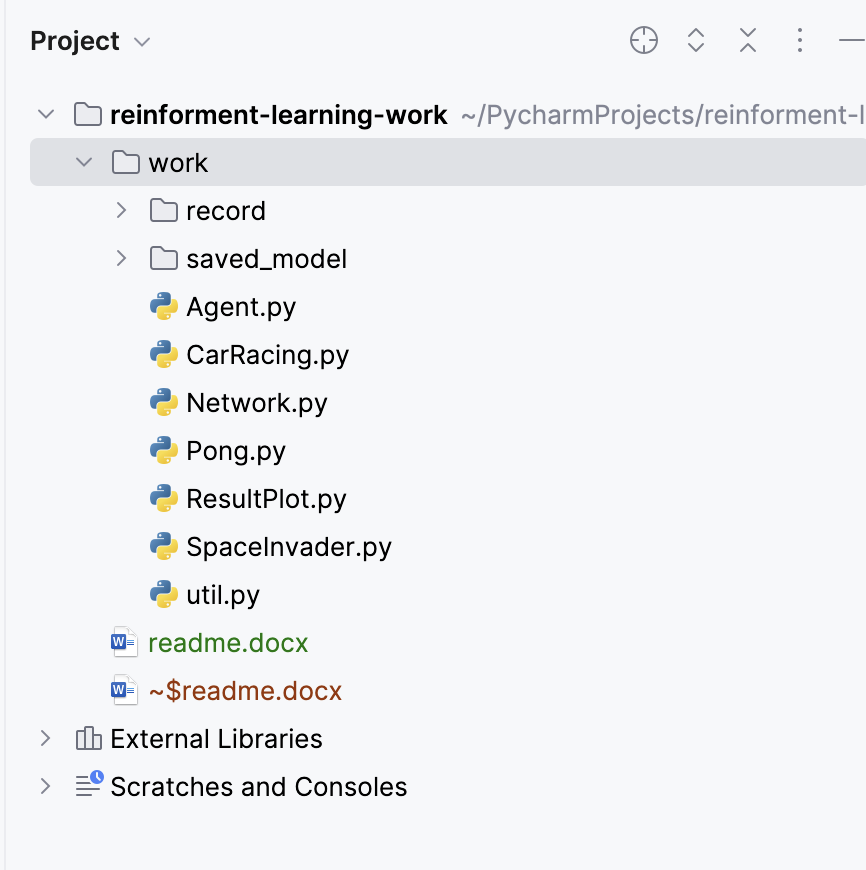
And following python packages are required for Box2D through “pip install”:



And following python packages are required for Atari through “pip install”:



Following are main code files:



Agent.py implements three algorithms: DQN, REINFORCE and Actor-Critic. The core implementation code in agent.py is mainly based on implementation code from Practicals with only minor updates (such as adding the function of writing training information into text file), so it is easy to read and understand.

CarRacing.py, Pong.py and SpaceInvader.py are files to initialize, train and evaluate agents by calling classes in Agent.py for three games respectively.

ResultPlot.py is to use matplotlib to visualize training outcome by figures

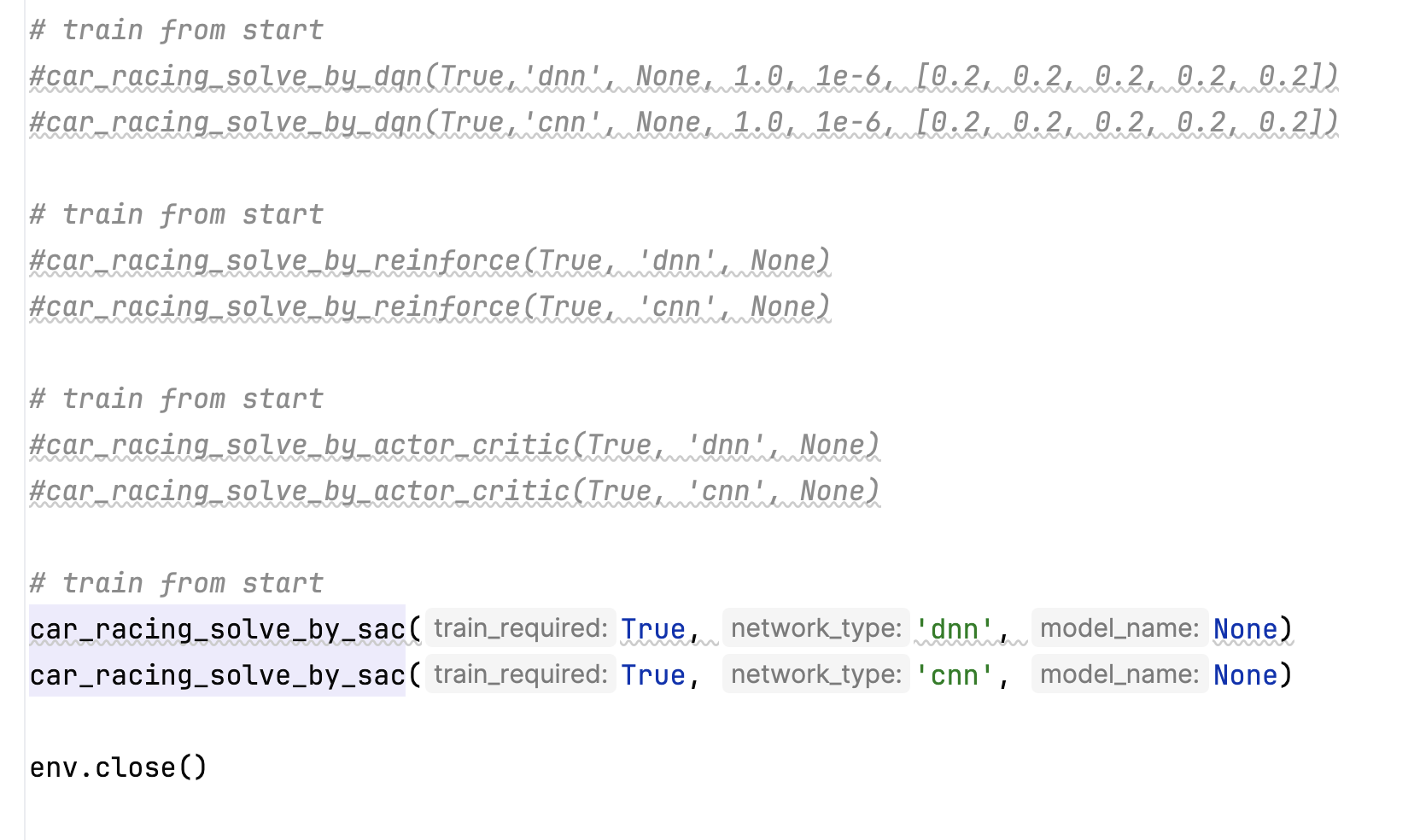
Util.py includes some tool functions for other files, such as image pre-process and text formation.

Training information text files and trained models are saved in “record” and “saved\_model” directories respectively.



In CarRacing.py, Pong.py and SpaceInvader.py, core parameters for agent are given in dictionary, such as:

1. network structure
2. learning rate
3. gamma (discount)
4. epsilon (e-greedy)
5. epsilon min (minimum value of epsilon after continuous decay)
6. decay speed (how much to reduce epsilon after each step)
7. max steps
8. stack frame num (how many consecutive frames are regarded as one state)
9. skip frame num (how many consecutive frames are given the same action of first frame every, for example, frame 1-4 given action of frame 1, frame 5-8 given action of frame 5 and so on)
10. use skip frame (if skipping frame trick is used, True of False)
11. replay buffer (size of experience replay buffer)
12. minibatch size (batch size for DQN)
13. target update (interval of steps for target Q-net sync to Q-net)
14. preferable action probs (when explore with greedy, the probability distribution of actions)
15. initial weight required (whether weights of network require to be randomly initialized)
16. network type (use ‘cnn’ or ‘dnn’)
17. type (‘policy’ network or ‘value’ network, which is used with ‘normalize prob required’ for deciding whether adding softmax layer at last)
18. target entropy (used for SAC)
19. normalize prob required (whether apply softmax to last layer of policy network, True or False, default False)



In CarRacing.py, Pong.py and SpaceInvader.py, there are four methods to train and evaluate three algorithms:

1. xx\_solve\_by\_dqn(): train and evaluate with DQN, parameters:
2. train\_required: True for training model, False for directly evaluate model
3. model\_name: can work with train\_required=False to load already trained model to directly evaluate, or work with train\_required=True so that training can continue based on previously trained model instead of nothing
4. greedy: define epsilon, this is useful when start to train from previously trained model, in this case, suppose previous model has already been trained for 500,000 steps, then then greedy can be set to 0.5 instead of 1.0 for training from nothing
5. decay\_speed: as it explained in previous paragraph
6. preferable\_action\_probs: as it explained in previous paragraph
7. network type: as it explained in previous paragraph

Also, you can add more parameters to be editable into function call if necessary

1. xx\_solve\_by\_reinforce(): train and evaluate with REINFORCE, parameters with the same name are similar with those for solve\_by\_dqn()
2. xx\_ solve\_by\_actor\_critic(): train and evaluate with Actor-Critic, parameters with the same name are similar with those for solve\_by\_dqn()
3. xx\_solve\_by\_sac(): train and evaluate with SAC, parameters with the same name are similar with those for solve\_by\_dqn()

As training requires incredible amount of time, environments for three games are all executed with graphic interface disabled.