In this homework, you are required to implement *deep reinforcement learning* algorithms such as DQN and its variants in a **real-world environment**. Read through this file before investing too much time in coding.

1 Environment

We adopt the Tencent Kaiwu Platform as our environment. Please register your account with your SJTU email, your real name and your student ID. Enter the experiment homepage and you will see the experiment environment. The environment, named *Back to the Realm*, is a maze quest in the game *Honor of Kings*. You will train agents to collect treasure and navigate to the destination in a maze shown in Figure 1. Please refer to the guidebook for full information of this environment and the documentation for how to develop and train an agent in the environment. Below is a simplified introduction based on the guidebook and the documentation.

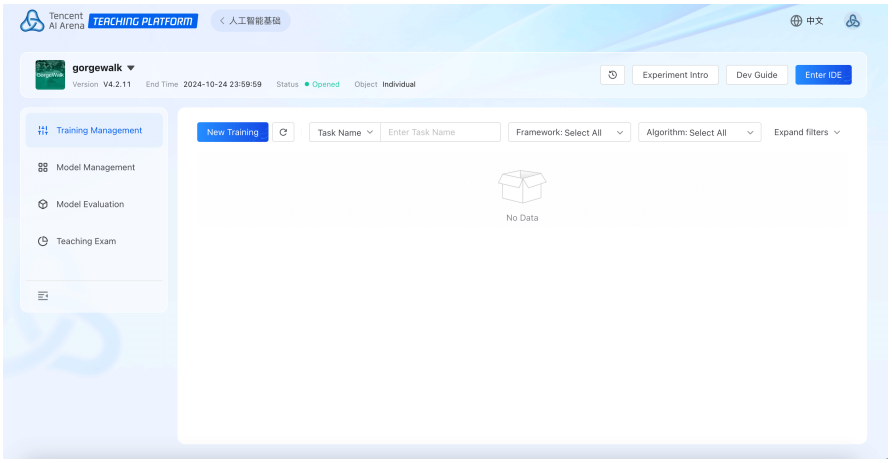
Figure 1: The enviornment visualization.

The yellow boxes are treasures that provide score rewards, and the blue little guy in the middle denotes a periodically reappearing acceleration buff. Your agent is required to navigate from the red marker in the bottom-left corner to the blue marker in the top-right corner while collecting treasures for a higher score. Apart from the scores from treasures, the faster your agent gets to the destination, the higher its score will be. The final evaluation score will be the sum of the treasure scores and the destination score. However, if your agent fails to reach the destination in a predefined maximum number of steps, it will receive zero score, no matter how many treasures it collects.

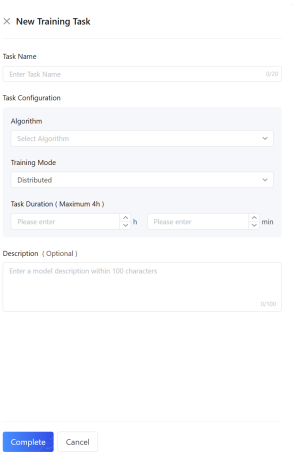
1.1 Coding and Training

After you enter the training platform, you will see Figure 2a. Click the "Enter IDE" button (top-right) and wait a few seconds before you see a `vscode`-like online code editor. There will be initial codes in three folders `diy`, `dqn` and `target_dqn`. These codes are provided by the platform, and they should work properly (`diy` is a folder with blank implementation for full customization, but we do not recommend starting from it).

After you have modified the code, run and debug your code as you did in your local `vscode` client. For detailed instructions, please refer to the documentation. Having confirmed that your code works, you can submit a training task by clicking the "New Training" button in the training homepage (Figure 2a). For the



(a) The training homepage.



(b) The new task page.

Figure 2: Coding and training.

algorithm selection bar, choose the folder you want to train your agent with (e.g. after you implemented DQN by modifying the `agent.py` file in folder `target_dqn`, you should choose "Target DQN").

1.2 Visualization and Submission

To evaluate a model, select "Model List" in a training item and click "Submit Model" for the checkpoint you want to test. Wait a few seconds until the model checkpoint is shown in the "Model Management" page and checked successfully.

Now you can submit an evaluation with a custom task configuration. Whatever the task configuration is in your training code, you are required to submit your evaluation results with a task configuration of 8 random treasures, 2000 steps, and 5 evaluation tasks. The evaluation results can be exported as a video, an example of which is provided in the homework directory (obviously, it is not an ideal demonstration due to not choosing the shortest route and not taking the acceleration buff).

Please notice that we have access to your training codes and your training records. Do not directly copy others' monitoring logs or model checkpoints.

2 Homeworks

1. **(Implementation of the DQN, Target DQN and Double DQN algorithms)** In this part, you will modify `agent.py` to implement DQN and its variants. Please replace the `target_dqn/agent.py` in the code environment with our provided `agent.py` and fill in the blanks that we have left incomplete. You are required to implement:

   - Definition of your own target Q network.
   - Calculation of the target Q values of DQN, Target DQN and Double DQN.
   - The training loop which backwards the Q value loss function for these DQN variants.

   In principle, we want you to switch between different variants of DQN algorithm through the parameter `algo_name` in line 37 of `agent.py`. For example, `algo_name='DQN'` and `algo_name='TARGET_DQN'` stands for the vanilla DQN and target DQN algorithms, respectively. You should implement vanilla DQN, target DQN and double DQN (in fact, the implementations are highly similar). Each block of code should take fewer than 20 lines.

2. **(Implementation of Dueling DQN)** In `target_dqn/model/model.py`, there is a predefined model class that will be instantiated as the backbone Q-networks during training and evaluation. In the

previous task we have maintained the model class unchanged. Now you are required to modify the model architecture to implement the Dueling DQN algorithm.

3. **(Experiments and Questions)** Having implemented these DQN variants, run experiments in the Honor of Kings environment now! You should run at least four trials to compare the performance (again, as is stated in Section 1.1, whatever the actual algorithm is adopted, in "New Training" page you should always choose "Target DQN"):

   - Vanilla DQN.
   - Target DQN.
   - Double DQN.
   - Dueling (vanilla/Target/Double) DQN.

   Finally, summarize you observation of the training trajectories of these algorithms and compare them with each other. For example, do you find double DQN preventing overestimation of Q-values as is described in the textbook?

4. **(Further Exploration)** **There is no bonus for anything described below (or beyond),** but we encourage everyone who is interested to explore.

   There are several factors contributing to the performance of learned agent:

   - **Training environment configuration**. The default configuration is set in line $34 - 44$ in `target_dqn/train_workflow`, with point 2 as the starting point, 1 as the destination, 8 random treasures and an allowed maximum number of steps 2000 (as is in our required evaluation configuration). You can modify it to see how the agent generalizes to different tasks.
   - **Reward definition**. If you are wondering why we choose `target_dqn` as the base folder, this is the reason. You can train a vanilla DQN selecting "DQN" as the algorithm in the "New Training" page and compare its performance to the vanilla DQN in `target_dqn` folder. Then you will see how the reward formulation impacts your learning algorithm. See line $50 - 220$ in `target_dqn/feature/definition.py` for the implementation of reward function.

3 Submission and Further Notes

In your submission, you should provide:

- A `agent.py` and a `model.py` file, indicating your implementation of DQN variants.
- A text file (markdown, pdf, e.t.c) summarizing the experiment results. You should include necessary screenshots of the training monitor or evaluator to make your observation and claims credible.

Note 1: Do not modify the attribute name `self.model` of the model. It is used to interact with the environment in class `LearningAlgo`. The interfaces of the environment requires a `self.model` instance to sample actions.

Note 2: In principle, all experiments except the vanilla DQN should be converging well within two hours of training. In the training monitor page, the `env/score` item should reflect its convergence, and do not rely too much on `algorithm/value_loss` to judge convergence.