Sample source code description: training and competition

During the competition, the following State information will be returned after an Action is performed:

- ✓ Information about competing Agents (***"playerId"***: Agent's ID, integer; ***"posx"***: Agent's X position, integer; ***"posy"***: Agent's Y position, integer; ***"score"***: Agent's amount of gold mined, integer; ***"energy""***: Agent's amount of remaining energy, integer; ***"lastAction"***: the last action, integer).
- ✓ Information about the remaining obstacles on the map (their position and the amount of energy that will be subtracted when an Agent passes by).
- ✓ Information about the remaining gold mines on the map (their position and the amount of gold).
- ✓ Map size (height and witdth)

Based on the returned State information, teams can decide their own training strategies, such as designing Reward Function and defining State Space. In the two sample source code (**Miner-Training-Local-CodeSample and Miner-Testing-CodeSample**) provided to teams (described below), we will give an example on designing Reward Function and defining State Space using 02 functions *get_state() and get_reward() respectively*. Below is an overview of the two sample source code provided for training and competition:

A. Source code for training - Miner-Training-Local-CodeSample

This is the sample source code used for training. The source code contains 02 major parts: Miner Game Environment and Deep reinforcement learning algorithm (Deep-Q learning - DQN). Figure 1 illustrates the information flow between programs.

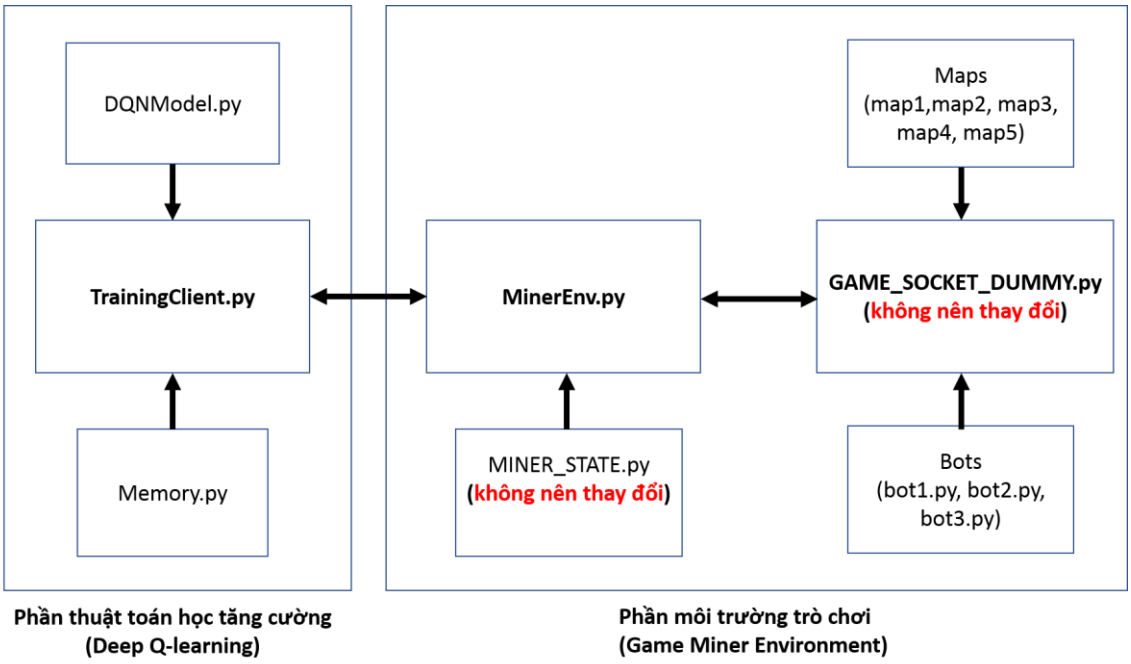

Figure 1: The information flow between programs in the sample source code used for training

Details of the two parts are as follows:

1. Miner Game Environment

The source code of Miner Game Environment is derived from the original source code of Miner Game on Codelearn system. It includes GAME_SOCKET_DUMMY.py, MINER_STATE.py, MinerEnv.py, Maps, and 03 Bots (bot1.py, bot2.py, bot3.py). Figure 2 illustrates the exchange process of map information and Agent's State information between MinerEnv.py and GAME_SOCKET_DUMMY.py. The details of programs are described below.

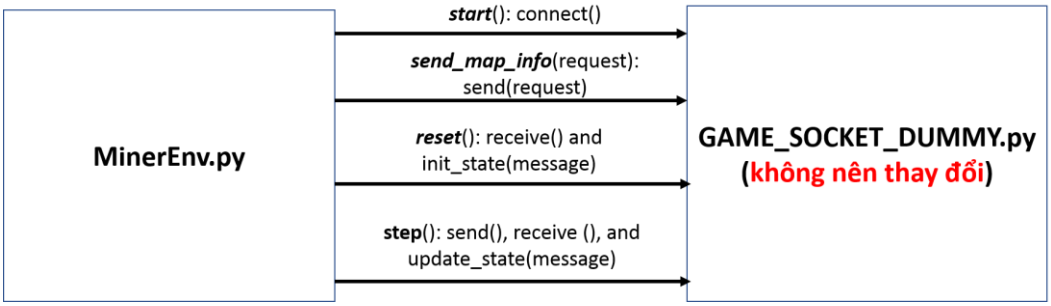

Figure 2: The information flow between MinerEnv.py and GAME_SOCKET_DUMMY.py during training (simulating the information flow between client and server)

a) **MinerEnv.py**: A program designed based on the general structure of the reinforcement learning environment to help teams access the main program (GAME_SOCKET_DUMMYpy) in a simple and convenient way. Some of the main functions in the program are as follows:

- ***start()***: a single-use function used to simulate the process of connecting to the server to start playing. During training, this function calls the ***connect()*** function in GAME_SOCKET_DUMMY.py to read 05 maps in Maps folder.
- ***send_map_info()***: a function used to select maps to train Agents.
- ***reset()***: a function used to initialize a map and a State for the Agent. This function calls the ***receive()*** function in GAME_SOCKET_DUMMY.py to get the map initial information saved in a json message, as well as the ***init_state(message)*** function in MINER_STATE.py to update the Agent's State with the map initial information.
- ***step()***: a function used to send an action to GAME_SOCKET_DUMMY.py and receive the changes of map information and Agent's State.
- ***get_state()***: a function provided as an example of defining a State of the Agent during training. Teams can overwrite this function to define a State that suits their training strategies.
- ***get_reward()***: a function provided as an example of defining a reward function of the Agent during training. Teams can overwrite this function to define a reward function that suits their training strategies.

b) **MINER_STATE.py (**teams should not change the source code in this program**)**: this program is a sample source code for saving the map information and the Agent's State received from GAME_SOCKET_DUMMY.py (which will be sent from the server in the actual competition). This program is designed to help teams manage their State information easily during training. Map and State classes, along with some main functions in these two classes, are as follows:

- **MapInfo** (Class): a class used to store all map information. This class includes max_x, max_y, maxStep, numberOfPlayer, golds (the current amount of remaining gold on the map), obstacles (the information about current obstacles on the map).
  + ***update (golds, changedObstacles):*** update the map information after each step.

- **State** (Class): a class that contains the States of the game (including the player's State and map).
  + ***init_state(data)***: a function used to initialize the map information and the Agent's State at the beginning of an episode in training (or a match in the actual competition).
  + ***update_state(data):*** a function used to update the State of the game after each step. The transferred data includes the map information and the Agent's State.

c) GAME_SOCKET_DUMMY.py (teams should not change the source code in this program**):** a program used to simulate gold miner game, including the process of transferring data (message) to the server. This program contains 07 classes: ObstacleInfo, GoldInfo, PlayerInfo, GameInfo, UserMatch, StepState and GameSocket. GameSocket is the main class and contains the following main functions:

- ***__init__ (host, port):*** a function used to initialize the environment. In this function, the purpose of host and port initialization is to simulate the connection on the server in the actual competition.
- ***init_bots():*** a function designed to assist the players to train Agent with bots. To specify a bot to participate in training, use the following command: ***self.bots = [Bot1(2), Bot2(3), Bot3(4)]***
- ***connect():*** a function used to simulate the connection from client to server. In training, the function will upload maps from the Maps folder to the environment.
- ***receive():*** a function used to simulate the action in which the client receives messages from the server. During training, if this function is called for the first time, it will return the map initial information and the Agent's initial State. In other cases, it will return the current map information and the Agent's current State.
- ***send():*** a function used to simulate the action in which the client sends messages to the server. During training, there are 2 types of messages from the client:
  + Action: an action for the next step, the data type is numeric.
  + Request: a request for parameters to initialize the game environment. The parameters include: map, init_x, init_y, init_energe, max_step. For example, request = "map1,1,2,100,150" means that the server will use the map information (gold, obstacles) from map1 in the Maps folder, the players will start from position (x = 1, y = 2) with an initial energy of 100, and the game will have a maximum of 150 steps.

d) Maps: The Maps folder. It containing 05 sample maps for training. Information on traps will be the same in these 05 maps, only the positions of gold mines and the amount of gold will change in the preliminary round. Teams may redesign these maps to suit their own training strategies. Teams need to pay attention to the followings when working with Maps:

- ✓ Each file in the Maps folder is considered a map, the filename is the map name.
- ✓ Each map is a matrix of integers with the following meanings:

0	Land
-1	Woods
-2	Trap
-3	Swamp
> 0	Gold

- ✓ Select a training map as follows:
  - Function to select a map in MinerEnv.py file: send_map_info(request)
  - Request structure: {map_name},{init_x},{init_y},{init_energy}

- For example, request = "map2,4,5,1000" means that map2 will be used for the match,the players will start from position (x = 4, y = 5) with an initial energy of 1000.

e) Bots (not the bots used in the preliminary round): 3 sample bots (bot1.py, bot2.py, and bot3.py) are provided to teams. Teams can create bots to suit their training strategies. The bots will be put into play in the game environment via GAME_SOCKET_DUMMY**.**py**.** You will need to declare the bots (*import Botx*) and initialize them (*init_bots()*). Some of the main functions in the bot source code are as follows:

- ***new_game(data)***: a function used to initialize the game environment (including initial map information and the initial state of the bots).
- ***new_state(data)*** : a function used to update the State received from the server.
- ***next_action*** : a function used to return an Action for the next step.

2. Deep reinforcement learning algorithm (Deep Q-learning)

In this section, the source code is written based on the Deep reinforcement learning algorithm (Deep Q-learning - DQN). The DQN algorithm has been introduced in a work of Mnih et al (*"Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533*). The source code contains the following program files: TrainingClient.py, DQNModel.py, and Memory.py.

a) TrainingClient.py: this program allows communication with the game environment. Some main points in this program are as follows:

i. Initialize parameters for the algorithm:

```
N_EPISODE = 10000 #The number of episodes for training
MAX_STEP = 1000 #The number of steps for each episode
BATCH_SIZE = 32 #The number of experiences used in each training session
MEMORY_SIZE = 100000 #The memory capacity to save experiences
SAVE_NETWORK = 100 # The number of episodes after which the DQN network will be saved
INITIAL_REPLAY_SIZE = 1000 #The number of experiences required to start training
INPUTNUM = 198 #The number of inputs for the DQN network
ACTIONNUM = 6 #The number of actions equivalent to the number of outputs of the DQN network
```

ii. Initialize the game environment:

```
minerEnv = MinerEnv(HOST, PORT)
minerEnv.start()
```

iii. Acquire the initial State of the Agent:

```
minerEnv.reset()
s = minerEnv.get_state()
```

iv. Perform an Action:

```
action = DQNAgent.act(s)
minerEnv.step(str(action))
```

v. Acquire the current State of the Agent and reward for the last Action, check the requirements to terminate the episode:

```
s_next = minerEnv.get_state()
reward = minerEnv.get_reward()
```

terminate = minerEnv.check_terminate()

vi. ***Train the DQN network (put some experiences from Memory to DQNAgent to start training):***

batch = memory.sample(BATCH_SIZE)

DQNAgent.replay(batch, BATCH_SIZE)

b) DQNModel.py: this source code is designed to allow the creation of deep learning models and model training functions. Some main points in this program are as follows:

- ***Initialize numeric parameters:***
  gamma = 0.99, #The discount factor
  epsilon = 1, #Epsilon - the exploration factor
  epsilon_min = 0.01, #The minimum epsilon
  epsilon_decay = 0.999,#The decay epislon for each update_epsilon time
  learning_rate = 0.00025, #The learning rate for the DQN network
- ***create_model():*** a function used to create a deep network. The network contains 02 hidden layers (each layer has 300 nodes, the activation function of these 02 hidden layers is ReLu) and an output layer (06 nodes corresponding to 06 Q-action values of 06 actions, the activation function is Linear).
- ***act(state):*** a function used to return an Action for the Agent at the State.
- ***replay(samples, batch_size):*** a function used to train the deep network with experiences from Memory file.
- ***update_epsilon():*** a function used to reduce epsilon (exploration factor).

c) Memory.py: this source code is used to store data (experiences) for training.

✓ ***Note***: *As the above source code is used for training, the game ends only when the map runs out of gold or the players are eliminated.*

B. Source code for competition -Miner-Testing-CodeSample

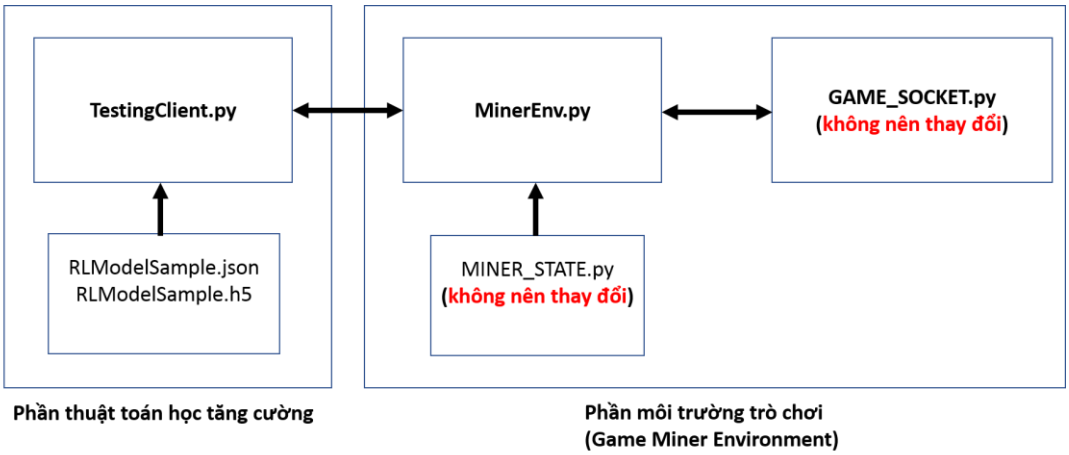

Figure 3: The information flow between programs in the sample source code used in the competition

✓ A source code designed for teams to use in official competitions.

✓ The difference between this and the source code provided for training (**Miner-Training-Local-CodeSample**) is that this source code uses GAME_SOCKET.py instead of GAME_SOCKET_DUMMY.py. GAME_SOCKET.py allows data transfer to the server.

- ✓ Information on the HOST and PORT of the server is taken from the environment variables when TestingAgent.py is executed.
- ✓ The other source code (MINER_STATE.py and MinerEnv.py) is similar to that provided for training (Miner-Testing-CodeSample).
- ✓ In the source code, a trained DQN model (RLModelSample.json, RLModelSample.h5) is provided as an example for uploading a model in the competition (Note: the model has not been fully trained to be able to compete). In particular, the json file stores the network parameters and the h5 file stores the network weight.