Mô Tả Mã Nguồn Mẫu: Huấn Luyện và Thi đấu

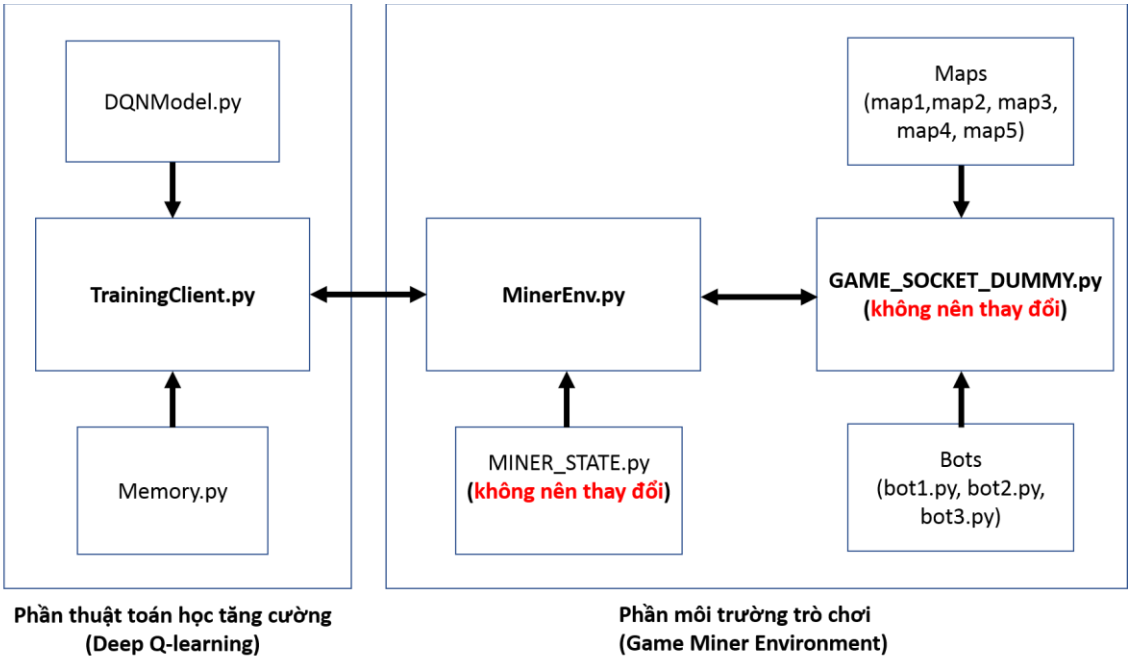
Trong quá trình thi đấu, thông tin trạng thái (State) được trả về sau khi thực hiện hành động (Action) bao gồm:

- ✓ Thông tin của những agent đang thi đấu (***"playerId"***: tên định danh của agent, kiểu integer; ***"posx"***: vị trí theo tọa độ x của agent, kiểu integer; ***"posy"***: vị trí theo tọa độ y của agent, kiểu integer; ***"score"***: số vàng agent đào được, kiểu integer; ***"energy"***: số năng lượng còn lại của agent, kiểu integer; ***"state"***: lưu action vừa thực hiện, kiểu integer).
- ✓ Thông tin các vật cản còn lại trên bản đồ (vị trí và số năng lượng sẽ bị trừ khi một agent đi qua).
- ✓ Thông tin số bãi vàng còn lại trên bản đồ (vị trí và số vàng).
- ✓ Kích thước của bản đồ (chiều cao và độ rộng).

Từ thông tin State được trả về trên, các đội chơi quyết định chiến lược huấn luyện riêng như thiết kế hàm thưởng (Reward Function) và định nghĩa không gian State (State Space). Trong hai mã nguồn mẫu (**Miner-Training-Local-CodeSample và Miner-Testing-CodeSample**) được cung cấp cho các đội chơi (được mô tả ở phía dưới),chúng tôi đưa ra một ví dụ về việc thiết kế Reward Function và định nghĩa State Space lần lượt trong 02 hàm *get_state() và get_reward()*. Dưới đây là mô tả tổng quan về hai mã nguồn được cung cấp cho việc huấn luyện và thi đấu:

A. Mã nguồn cho huấn luyện - Miner-Training-Local-CodeSample

Đây là mã nguồn (source code) mẫu được sử dụng cho quá trình huấn luyện tại máy của các đội chơi. Mã nguồn bao gồm 02 phần chính: môi trường trò chơi đào vàng (Miner Game Environment) và thuật toán học tăng cường (Deep-Q learning -DQN). Hình 1 cung cấp cái nhìn trực quan về luồng trao đổi thông tin giữa các chương trình.



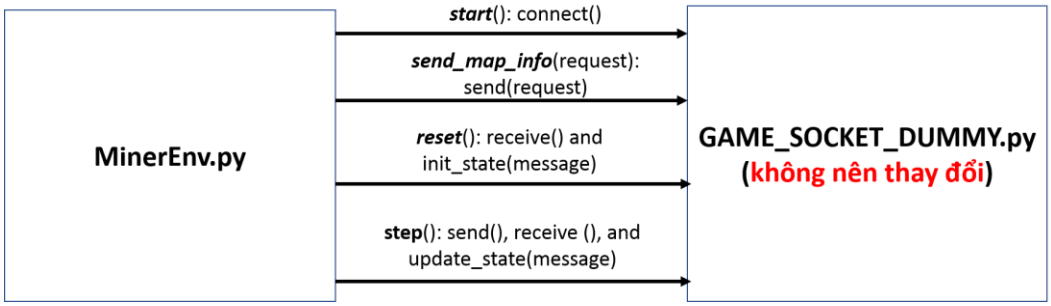
Hình 1: Luồng trao đổi thông tin giữa các chương trình trong mã nguồn mẫu

được sử dụng trong Huấn luyện.

Chi tiết của hai phần như sau:

1. *Phần môi trường trò chơi đào vàng (Miner Game Environment).*

Mã nguồn của môi trường được lấy từ mã nguồn gốc của trò chơi đào vàng (Miner Game) trên hệ thống Codelearn. Mã nguồn bao gồm: GAME_SOCKET_DUMMY.py, MINER_STATE.py, MinerEnv.py, Maps, và 03 Bots (bot1.py, bot2.py, bot3.py). Hình 2 mô tả quá trình trao đổi về thông tin bản đồ và trạng thái của agent giữa MinerEnv.py và GAME_SOCKET_DUMMY.py. Mô tả chi tiết các chương trình được miêu tả phía dưới:



Hình 2: Luồng trao đổi thông tin giữa MinerEnv.py và GAME_SOCKET_DUMMY.py được mô phỏng giữa client và server trong Huấn luyện.

a) **MinerEnv.py**: Chương trình được thiết kế theo cấu trúc chung của môi trường học tăng cường (Reinforcement learning environment) cho phép các đội chơi truy cập tới chương trình chính (GAME_SOCKET_DUMMY.py) đơn giản và thuận tiện. Một số hàm chính trong chương trình như sau:

- ***start()*** : hàm dùng một lần duy nhất với mục đích mô phỏng lại quá trình kết nối tới server để bắt đầu chơi. Trong Training, hàm này gọi tới hàm ***connect()*** trong GAME_SOCKET_DUMMY.py để đọc 05 maps trong tệp Maps.
- ***send_map_info()*** : hàm được sử dụng nhằm mục đích chọn map để huấn luyện agent.
- ***reset()*** : hàm được sử dụng nhằm mục đích khởi tạo map và state cho agent. Hàm này sẽ gọi tới hàm ***receive()*** trong GAME_SOCKET_DUMMY.py để lấy thông tin ban đầu của map được lưu trong một message định dạng json, và hàm ***init_state(message)*** trong MINER_STATE.py để cập nhật thông tin ban đầu của map tới state của agent.
- ***step()***: hàm được sử dụng nhằm mục đích gửi một hành động (action) tới GAME_SOCKET_DUMMY.py, và sẽ nhận về thông tin bản đồ (map) và trạng thái (state) của agent thay đổi.
- ***get_state()*** : hàm được cung cấp như một ví dụ cho việc định nghĩa một trạng thái (state) của agent cho quá trình huấn luyện. Các đội chơi tùy vào chiến lược huấn luyện riêng có thể viết lại hàm để đưa ra được state của agent phù hợp.
- ***get_reward()*** : hàm được cung cấp như một ví dụ cho việc định nghĩa một hàm thưởng (reward function) của agent cho quá trình huấn luyện. Các đội chơi tùy vào chiến lược huấn luyện riêng có thể viết lại hàm để đưa ra được hàm reward function cho phù hợp.

b) **MINER_STATE.py (**Các đội chơi không nên thay đổi mã nguồn trong chương trình**)**: Chương trình là mã nguồn mẫu cho việc lưu thông tin bản đồ (map) và trạng thái (state) của agent nhận được từ GAME_SOCKET_DUMMY.py (trong thi đấu sẽ nhận từ server). Chương trình được thiết kế giúp cho các đội chơi quản lý state dễ dàng trong quá trình huấn luyện. Hai lớp (class) map và state cùng với một số hàm chính trong hai lớp như sau:

- **MapInfo** (Class): Là lớp dùng cho việc lưu trữ toàn bộ thông tin của bản đồ. Bao gồm: max_x, max_y, maxStep, numberOfPlayer, golds: số vàng còn lại trên map tại thời điểm hiện tại, obstacles: thông tin các vật cản hiện tại trên bản đồ.
  + ***update (golds, changedObstacles):*** cập nhật lại thông tin bản đồ sau mỗi bước (step).
- **State** (Class): là lớp chứa trạng thái (state) của trò chơi (bao gồm trạng thái của người chơi và bản đồ).
  + ***init_state(data)***: hàm khởi tạo thông tin của bản đồ (map) và trạng thái (state) của agent tại thời điểm bắt đầu một episode trong training (hay một trận đấu trong thi đấu).
  + ***update_state(data):*** hàm cập nhật trạng thái (state) của trò chơi sau mỗi lượt chơi (step). Data được truyền vào bao gồm thông tin bản đồ và trạng thái của agent.

c) GAME_SOCKET_DUMMY.py (Các đội chơi không nên thay đổi mã nguồn trong chương trình**) :** Chương trình mô phỏng lại trò chơi đào vàng bao gồm cả quá trình truyền nhận dữ liệu (message) tới máy chủ (server). Chương trình bao gồm 07 lớp (class) : ObstacleInfo, GoldInfo, PlayerInfo, GameInfo, UserMatch, StepState,và GameSocket. Trong đó, lớp GameSocket là lớp chính gồm những hàm chính sau:

- ***__init__ (host, port):*** hàm được sử dụng nhằm mục đích khởi tạo môi trường. Trong hàm này, khởi tạo host và port nhằm mục đích mô phỏng việc tạo kết nối trên server nếu trong thi đấu.
- ***init_bots():*** hàm được thiết kế để hỗ trợ người chơi huấn luyện agent với bot. Để chỉ định bot tham gia vào huấn luyện, sử dụng dòng lệnh sau : ***self.bots = [Bot1(2), Bot2(3), Bot3(4)]***
- ***connect():*** hàm này mô phỏng hành động connect từ client đến server. Trong huấn luyện, hàm sẽ tải các bản đồ từ tệp Maps lên môi trường.
- ***receive():*** hàm này mô phỏng hành động client nhận message từ server. Trong huấn luyện, hàm sẽ trả về thông tin bản đồ ban đầu cũng như trạng thái ban đầu của agent nếu lần đầu được gọi, và sẽ trả về thông tin bản đồ hiện tại cũng như trạng thái hiện tại của agent.
- ***send():*** hàm này mô phỏng hành động client gửi message lên server. Trong huấn luyện, sẽ có 2 kiểu message từ client:
  + Action: là hành động cho step tiếp theo và là kiểu số
  + Request: là yêu cầu các thông số cho việc khởi tạo môi trường trò chơi. Các thông số bao gồm: (map, init_x, init_y, init_energe, max_step). Ví dụ: request = "map1,1,2,100,150" sẽ hiểu là server sẽ sử dụng thông tin bản đồ (vàng, vật cản) theo map1 trong tệp Maps, khởi tạo vị trí ban đầu cho người chơi tại ô (x = 1, y = 2), với năng lượng ban đầu là 100, và trận đấu có tối đa 150 lượt chơi (step).

d) Maps: Thư mục Maps chứa 05 bản đồ (map) mẫu phục vụ cho việc huấn luyện. Thông tin về bẫy trong 05 bản đồ là được giữ nguyên, và chỉ thay đổi vị trí vàng và số lượng vàng trong vòng sơ loại. Đội chơi có thể thiết kế lại những bản đồ này cho phù hợp với chiến lược huấn luyện riêng. Một số nội dung cần chú ý khi các đội chơi làm việc với Maps như sau:

✓ Mỗi file trong thư mục Maps được xem là một bản đồ, tên của file (filename) được xem là tên của bản đồ (map name).

✓ Mỗi bản đồ là một ma trận các số nguyên với ý nghĩa như sau:

0	Đất
-1	Rừng
-2	Bẫy
-3	Đầm lầy
> 0	Vàng

✓ Chọn bản đồ trong huấn luyện như sau:

- Hàm chọn bản đồ trong file MinerEnv.py : send_map_info(request)
- Cấu trúc của request: {map_name},{init_x},{init_y},{init_energy}
- Ví dụ: request = "map2,4,5,1000" - là trận đấu sẽ sử dụng map2, người chơi (players) xuất phát từ tọa độ: x = 4, y = 5 với năng lượng (energy) được khởi tạo là 1000.

e) Bots (không phải các bots được dùng trong vòng sơ loại): 03 bots (bot1.py, bot2.py, và bot3.py) được cung cấp mẫu cho các đội chơi. Các đội chơi có thể tạo những bot theo chiến lược huấn luyện riêng. Các bots sẽ được đưa vào trong môi trường trò chơi trong GAME_SOCKET_DUMMY.py**.** Bao gồm 02 bước: khai báo bots (*import Botx*) và khởi tạo bot (*init_bots()*). Một số hàm chính trong những mã nguồn bot như sau:

- ***new_game(data)***: hàm thực hiện khởi tạo môi trường trò chơi (bao gồm thông tin bản đồ và trạng thái ban đầu cho bot).
- ***new_state(data)*** : hàm cập nhật trạng thái (state) nhận được từ server.
- ***next_action*** : hàm trả về một hành động (action) cho bước (step) tiếp theo.

2. Phần thuật toán học tăng cường (Deep Q-learning)

Trong phần này, mã nguồn được viết theo thuật toán học tăng cường sâu (Deep Q-learning - DQN). Nguồn thuật toán được giới thiệu trong nghiên cứu của Mnih et al (*"Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533*). Mã nguồn thuật toán bao gồm những file chương trình sau: TrainingClient.py, DQNModel.py, và Memory.py.

a) TrainingClient.py: đây là mã nguồn của thuật toán DQN cho phép giao tiếp với phần môi trường trò chơi. Trong chương trình này, một số phần cần chú ý như sau:

i. Khởi tạo các tham số cho thuật toán:

```
N_EPISODE = 10000 #Số episode cho huấn luyện)
MAX_STEP = 1000   #Số bước (step) cho mỗi episode
BATCH_SIZE = 32   #Số trải nghiệm (experiences) được sử dụng cho môi lần huấn luyện.
MEMORY_SIZE = 100000 #Kích thước bộ nhớ lưu những trải nghiệm
SAVE_NETWORK = 100# Sau số episode này, mạng DQN sẽ được lưu lại.
INITIAL_REPLAY_SIZE = 1000 #Số trại nghiệm cần phải có trong bộ nhớ để bắt đầu huấn luyện.
INPUTNUM = 198 #Số đầu vào cho mạng DQN
ACTIONNUM = 6  #Số hành động tương đương số đầu ra của mạng DQN.
```

ii. Khởi tạo môi trường trò chơi:

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

iii. Lấy trạng thái (state) ban đầu của agent:

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

iv. Thực hiện hành động (action):

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

v. ***Lấy trạng thái mới của agent, phần thưởng (reward) cho hành động vừa thực hiện, và kiểm tra điều kiện kết thúc episode:***

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

vi. ***Huấn luyện mạng DQN (lấy một số trải nghiệm (experiences) từ Memory, và đưa vào DQNAgent để huấn luyện):***

```
batch = memory.sample(BATCH_SIZE)
DQNAgent.replay(batch, BATCH_SIZE)
```

b) DQNModel.py: mã nguồn này được thiết kế cho phép tạo những mô hình học sâu (deep learning models) và những hàm huấn luyện mô hình. Trong chương trình này, một số phần cần chú ý như sau:

- ***Khởi tạo tham số học:***

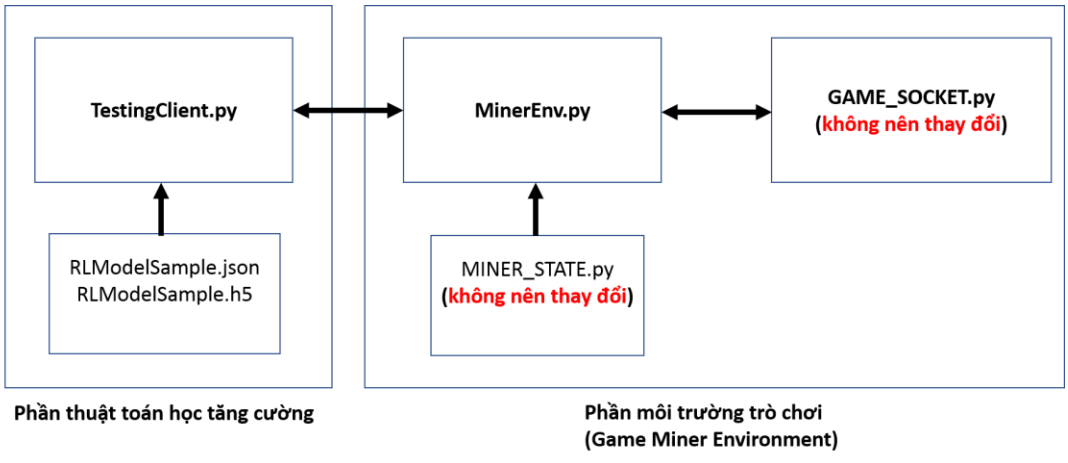
```
gamma = 0.99, #The discount factor
epsilon = 1, #Epsilon - the exploration factor
epsilon_min = 0.01, #The minimum epsilon
epsilon_decay = 0.999,#The decay epislon for each update_epsilon time
learning_rate = 0.00025, #The learning rate for the DQN network
```

- ***create_model():*** hàm được sử dụng để tạo một mạng sâu (deep network). Mạng chứa 02 lớp ẩn (hidden layers), mỗi lớp có 300 nút (node), hàm kích hoạt (activation) của hai lớp ẩn là ReLu, và một lớp đầu ra có 06 nút (node) tương ứng với 06 giá trị Q-action của 06 actions với hàm kích hoạt là Linear.
- ***act(state):*** hàm trả về hành động cho agent tại trạng thái state.
- ***replay(samples,batch_size):*** hàm được sử dụng để huấn luyện mạng sâu từ những trải nghiệm (experiences) lấy ra từ Memory file.
- ***update_epsilon():*** hàm thực hiện việc giảm epsilon hay giảm tham dò (exploration).

c) Memory.py: mã nguồn được sử dụng để lưu dữ liệu (experiences) cho việc huấn luyện

✓ ***Chú ý***: *Mã nguồn trên được dùng cho việc huấn luyện nên trận đấu kết thúc chỉ khi hết vàng trên bản đồ (map) hoặc người chơi bị loại (khi hết vàng hoặc ra khỏi bản đồ).*

B. Mã nguồn thi đấu -Miner-Testing-CodeSample



Hình 3: Luồng trao đổi thông tin giữa các chương trình trong mã nguồn mẫu được sử dụng trong thi đấu.

- ✓ Mã nguồn được thiết kế cho các đội chơi sử dụng để tham gia cuộc thi chính thức.
- ✓ Điểm khác với mã nguồn cung cấp cho phần huấn luyện (**Miner-Training-Local-CodeSample**) đó là sử dụng GAME_SOCKET.py thay cho GAME_SOCKET_DUMMY.py. Chương trình GAME_SOCKET.py cho phép truyền nhận dữ liệu tới máy chủ (server).
- ✓ HOST và PORT của server được lấy từ biến môi trường khi run TestingAgent.py
- ✓ Những mã nguồn còn lại (MINER_STATE.py và MinerEnv.py) được giữ nguyên như trong mã nguồn được cung cấp cho huấn luyện (Miner-Testing-CodeSample).
- ✓ Trong mã nguồn, một DQN model (RLModelSample.json, RLModelSample.h5) đã được huấn luyện được cung cấp để làm ví dụ cho việc tải model lên thi đấu (Chú ý: model trên chưa được huấn luyện đầy đủ để có thể thi đấu). Trong đó, file định dạng json lưu tham số mạng và file định dạng h5 lưu trọng số của mạng.