VIS – TỔNG HỢP HƯỚNG DẪN CODE HỆ THỐNG

GAME REINFORCEMENT LEARNING

0. Link git hệ thống mẫu: AnnNguy3n/Splendor_v1.2.git
1. Thiết kế mô phỏng environment, actions và state
   a. Environment (env)
      - Chứa thông tin về tất cả các yếu tố có trong game. Có thể hiểu env giống như góc nhìn của người quản trò (trọng tài) đứng ngoài quan sát, có thể thấy được trạng thái của bàn chơi và trạng thái của toàn bộ người chơi.
      - Mô phỏng env dưới dạng một hoặc nhiều **array** để phù hợp với việc sử dụng thư viện "**python numba**" tối ưu thời gian chạy.
      - Thiết kế env như nào?
        Tìm hiểu và quan sát những người khác chơi game để bước đầu xác định các thông tin cần phải mô phỏng trong env.

Dưới dây là ví dụ khi thiết kế mô phỏng env của tựa game **Splendor**:

Ảnh minh họa bàn chơi của Splendor

Splendor có 4 người chơi, 100 thẻ bài trong đó có 90 thẻ thường được đánh dấu từ 0 đến 89, 10 thẻ quý tộc được đánh dấu từ 0 đến 9, 6 loại nguyên liệu trong đó có 5 nguyên liệu thường, mỗi loại có số lượng là 7 và 1 nguyên liệu đặc biệt (gold) có số lượng là 5. Có hai thành phần lớn cần phải mô phỏng trong Splendor, đó là bàn chơi (**Board**) và những người chơi (**Players**).

- **Board**:
    - Các thông tin bao gồm:
        - Nguyên liệu.
        - Các thẻ quý tộc.
        - Các thẻ cấp I, II, III đang mở trên bàn và các chồng thẻ úp tương ứng (ta quan tâm đến số lượng thẻ và thứ tự sắp xếp các thẻ trong những chồng thẻ úp).
        - Thông tin chi tiết về các thẻ (cả thẻ quý tộc và thẻ thường).

    - Mô phỏng các thông tin:
        - Nguyên liệu: array có độ dài 6. Các index từ 0 đến 5 lần lượt thể hiện số lượng của các loại nguyên liệu trên bàn chơi là “red”, “blue”, “green”, “black”, “white”, “gold”.
        - Thẻ quý tộc: array có độ dài 5, mỗi index thể hiện id của thẻ đang xuất hiện. Nếu vị trí nào không có thẻ (do đã được lấy trước đó) thì thay bằng giá trị -1.
        - Thẻ thường các cấp: 1 array có độ dài 4 thể hiện id của các thẻ đang xuất hiện. Ngoài ra, cần có một array lưu lại thứ tự các thẻ trong chồng thẻ úp.
        - Thông tin chi tiết về các thẻ: 2 array, một array lưu thông tin các thẻ thường (id từ 0 đến 89), một array lưu thông tin các thẻ quý tộc (id từ 0 đến 9).

- **Players**: 4 người chơi, thông tin của các người chơi sẽ bao gồm các thành phần giống nhau. Do đó chỉ cần thiết kế mô phỏng cho một người chơi, sau đó làm tương tự với những người chơi khác.
    - Các thông tin bao gồm:
        - Nguyên liệu.
        - Nguyên liệu vĩnh viễn.
        - Điểm.
        - Các thẻ đang úp (hold).

- Mô phỏng các thông tin:
  - Nguyên liệu: array có độ dài 6.
  - Nguyên liệu vĩnh viễn: array có độ dài 5.
  - Điểm: 1 giá trị thể hiện số điểm.
  - Các thẻ đang úp: 1 array có độ dài 3, mỗi index thể hiện id của thẻ đang úp. Nếu vị trí nào đó không có thẻ thì thay bằng giá trị -1.

Sau khi mô phỏng Board và Player, tổng hợp thành các array trong env. Ghi lại và giải thích ý nghĩa của từng index trong các array (ví dụ: xem file **design.txt** trong hệ thống mẫu).

b. Actions và game flow diagram

- **Actions**:
  - Gồm tất các cả các hành động mà Agent có thể thực hiện trong game.
  - Khi thiết kế, các actions được kí hiệu là các số tự nhiên tăng dần, bắt đầu từ 0 cho đến n-1, với n là "tổng số actions có thực hiện trong game".
- **Game flow diagram**: thể hiện mối quan hệ giữa **env** và **actions**, từ khi bắt đầu game đến lúc kết thúc game:
  - Với env đã biết thì Agent có thể thực hiện những actions nào.
  - Với mỗi action thì env sẽ thay đổi như thế nào.

Trước khi thiết kế actions, cùng tìm hiểu một chút về phase!

Phase là gì và tại sao lại dùng phase?

Trong Splendor, mỗi người chơi khi đến lượt có thể làm 3 hành động chính: lấy nguyên liệu (tối đa 3 nguyên liệu), úp thẻ nào đó (một trong các thẻ thường, xuất hiện trên bàn chơi) hoặc mua một thẻ nào đó (một trong các thẻ thường xuất hiện trên bàn chơi hoặc thẻ đang giữ). Với hành động lấy nguyên liệu hoặc úp thẻ, có thể dẫn tới việc người chơi phải trả nguyên liệu, do vậy số trường hợp là rất lớn (tổ hợp các trường hợp của nguyên liệu lấy và nguyên liệu trả), dẫn đến số actions cũng rất lớn, gây khó khăn cho việc học của Agent và cả thời gian chạy (rất mất thời gian trong bước tính toán các actions có thể thực hiện cho Agent). Do vậy, ta cần chia phase để giảm bớt tập actions xuống đáng kể ở những tựa game phức tạp.

Ý tưởng của chia phase?

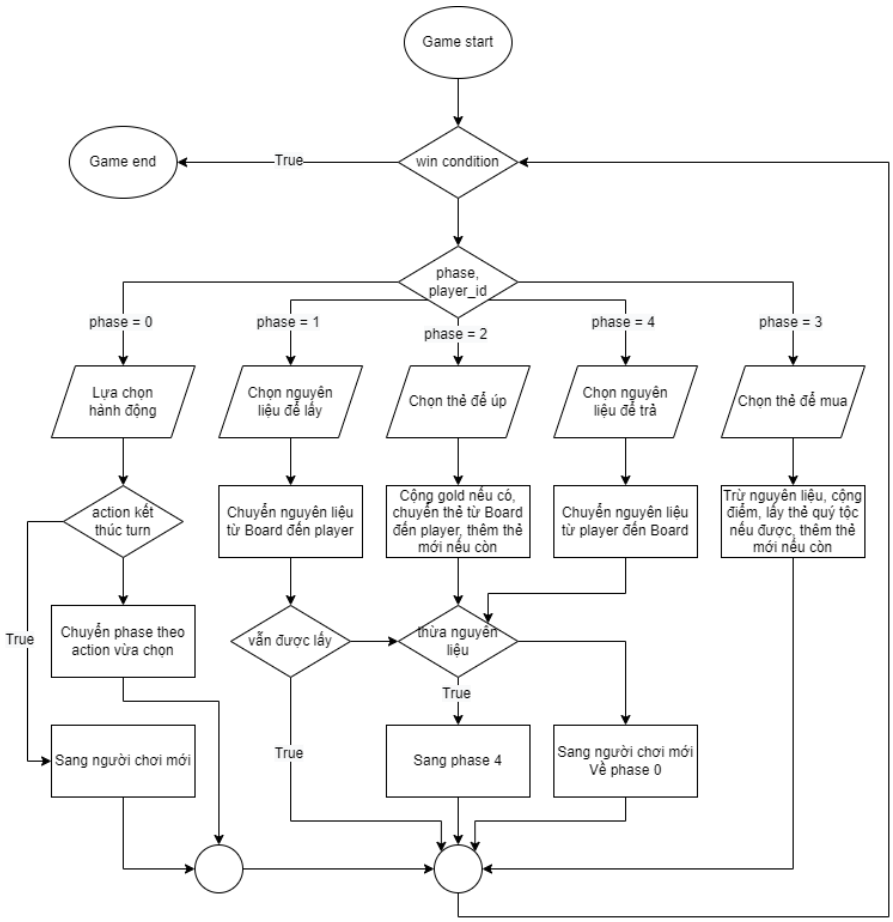
Trong tựa game có những hành động phức tạp, thực hiện chia nhỏ các hành động đó thành nhiều bước, tại mỗi bước sinh ra các hành động đơn giản hơn là một phần của hành động gốc. Toàn bộ quá trình thực hiện hành động gốc được coi là một phase.

Ví dụ: trong Splendor, hành động lấy nguyên liệu và trả nếu thừa (gom vào 1 action, phức tạp) có thể chia thành 2 phase tách biệt: phase lấy nguyên liệu và trả nguyên liệu. Ở phase lấy nguyên liệu, người chơi sẽ chọn 1 trong các nguyên liệu mà Board đang còn. Khi người chơi không thể lấy thêm nguyên liệu (hoặc người chơi chọn không lấy nữa) thì check xem người chơi có thừa nguyên liệu không, nếu thừa thì chuyển sang phase trả nguyên liệu. Ở phase trả nguyên liệu, người chơi chọn 1 trong các nguyên liệu hiện có để trả. Như vậy, thay vì thực hiện một action phức tạp, việc lấy nguyên liệu và trả nguyên liệu đồng thời được thay thế bằng các action đơn giản hơn rất nhiều.

Lưu ý 1.1: việc chia phase dẫn đến Agent phải action nhiều lần hơn, có thể khiến cho việc chạy một ván game lâu hơn.

Lưu ý 1.2: Trong quá trình thiết kế actions và game flow diagram, có thể phát sinh thêm các thông tin cần phải lưu lại trong env.

Dưới đây là thiết kế game flow diagram cho tựa game Splendor:



Splendor flow diagram (xem rõ hơn ở file **diagram.png** trong hệ thống mẫu)

Note: Ở phase 3, sau khi người chơi chọn thẻ để mua, hệ thống xử lí và sang người chơi mới (hình chữ nhật ở nhánh phase 3 phải nối vào ô “sang người chơi mới và về phase 0” chứ không phải nối vào hình tròn nhỏ).

Chi tiết về các actions và các thông tin phát sinh thêm cần phải lưu vào env, xem file **design_actions.txt** trong hệ thống mẫu.

Lưu ý 1.3: Các phase khác nhau có thể dẫn đến các action mang ý nghĩa giống nhau, do đó ta kí hiệu các action này cùng một giá trị. Lấy ví dụ: trong Catan, phase "chọn tài nguyên để trả ngân hàng" và phase "chọn tài nguyên để giao dịch với người chơi khác" mặc dù là 2 kiểu hành động khác nhau, nhưng vẫn được kí hiệu chung vì 2 kiểu hành động này đều có khả năng làm giảm lượng tài nguyên được chọn. Tuy nhiên với Splendor, khi người chơi chọn "úp thẻ" hoặc "mua thẻ", phase sau đó người chơi sẽ chọn "vị trí" thẻ muốn úp (hoặc mua). Dù cùng là chọn vị trí thẻ nhưng "úp thẻ" thì người chơi có thể được nguyên liệu "gold" và tăng số thẻ giữ, còn "mua thẻ" thì người chơi có thể mất nguyên liệu thường và được nguyên liệu vĩnh viễn. Do vậy, các action khi úp thẻ và mua thẻ không được kí hiệu chung giá trị do ý nghĩa khác nhau.

Lưu ý 1.4: Khi thiết kế actions cần chú ý đến tỉ lệ xuất hiện của các actions. Tránh:

- Để một action nào đó có tỉ lệ xuất hiện rất thấp.
- (Lỗi này ít người để ý): Một vài action tồn tại cũng nhiều action khác có liên quan tới nhau, khiến tỉ lệ chọn các action cũng thay đổi theo hướng không hợp lí. Ví dụ trong Splendor, thay vì cho các hành động lấy nguyên liệu ở đầu mỗi turn vào một phase, thì các hành động đó lại xuất hiện chung với hành động chọn lấy thẻ hoặc úp thẻ. Kết quả là hành động lấy thẻ hoặc úp thẻ xuất hiện cùng với nhiều action hơn, khiến tỉ lệ được chọn của 2 hành động này giảm:
    - Thiết kế chuẩn: 1. Chọn lấy nguyên liệu, 2. Chọn úp thẻ, 3. Chọn mua thẻ.
    - Thiết kế khiến tỉ lệ chọn hành động 2 và 3 bị giảm: 2. Chọn úp thẻ, 3. Chọn mua thẻ, 4. Lấy đỏ, 5. Lấy lam, 6. Lấy lục, 7. Lấy đen, 8. Lấy trắng.

c. State

- Là trạng thái của env mà Agent có thể quan sát được.
- Mô phỏng bằng duy nhất một numpy array, một chiều.
- Thiết kế như nào?
  Trực tiếp chơi game và ghi lại những thông tin mà người chơi có quyền được biết. Thực hiện che dấu một phần thông tin của env, sắp xếp lại các dữ liệu theo góc nhìn của người chơi. Điều chỉnh các **dữ liệu có tính chất phân loại**.

Ví dụ trong Splendor:

- Board: người chơi có thể thấy gần như toàn bộ thông tin trên bàn chơi, tuy nhiên người chơi không được nhìn thấy thông tin của các thẻ bài trong chồng bài úp.
- Players: Ngoại trừ thẻ úp, người chơi có thể nhìn thấy nguyên liệu, nguyên liệu vĩnh cửu, điểm của bản thân và người chơi khác, nhìn được người chơi khác đã úp mấy thẻ.

Lưu ý 1.5: Không cho các thông tin có ít giá trị vào state. Ví dụ trong Splendor, id thẻ là thông tin có ít giá trị đối với Agent, do đó thay vì truyền id thẻ, ta thay thế bằng thông tin chi tiết của thẻ tại vị trí tương ứng và truyền cho người chơi. Một ví dụ nữa, là thông tin về turn trong Splendor cũng không được truyền vào Agent.

Lưu ý 1.6: Về các dữ liệu có tính chất phân loại khi đưa từ env vào state:

Các dữ liệu được mô phỏng trong env có thể chia làm 2 loại: dữ liệu có tính chất phân loại và dữ liệu có tính chất đo lường.

- Dữ liệu có tính chất phân loại: là dữ liệu dùng để phân biệt một thứ gì đó trong game. Loại dữ liệu này chỉ có thể so sánh khác nhau, không thể so sánh độ lớn với nhau. Ví dụ: phase trong Splendor chỉ dùng để phân biệt trạng thái bàn chơi. Mặc dù phase có các giá trị khác nhau nhưng nói "phase 4 lớn hơn phase 1" là không đúng. Các phase khác nhau phải được mô phỏng với những tính chất công bằng khi truyền vào state.
- Dữ liệu có tính chất đo lường được: các dữ liệu có thể so sánh độ lớn nhỏ với nhau. Ví dụ: điểm của người chơi, điểm của thẻ, lượng nguyên liệu mỗi loại.

Đối với dữ liệu có tính chất đo lường được, có thể truyền ngay vào state. Nhưng đối với dữ liệu có tính chất phân loại, cần thực hiện như sau:

- Gọi k là số giá trị có thể có đối với dữ liệu đó (ví dụ: phase trong Splendor thì k = 5 do có 5 phase từ 0 đến 4).
- Thay vì truyền một giá trị vào state, ta truyền một array toàn 0 có độ dài k. Với giá trị dữ liệu là v, tại index v của array nói trên, đặt giá trị thành 1.

Lưu ý 1.7: State của agent ở các vị trí khác nhau phải có chung một cách biểu diễn:

Thông tin của người chơi nhận state luôn đứng đầu, sau đó lần lượt là thông tin của các người chơi liền sau. Giả sử sau đây là cách biểu diễn state cho người chơi ở vị trí đầu tiên (index 0):

[infor người 0] [infor người 1] [infor người 2] [infor người 3]

Nếu người 2 nhận state, thì state của người chơi 2 phải có dạng như sau:

[infor người 2] [infor người 3] [infor người 0] [infor người 1]

Xem file **env.py**, hàm **getAgentState** để hiểu về sự sắp xếp lại các dữ liệu bàn chơi.

2. Xây dựng các hàm trong hệ thống
    a. initEnv
        - Đầu vào: không.
        - Đầu ra: Một hoặc nhiều array (tùy thuộc vào thiết kế env trong phần **1a.**).
        - Trả về env ở trạng thái ban đầu.

    b. getAgentState
        - Đầu vào: env.
        - Đầu ra: array, float64.
        - Trả về state như đã thiết kế trong phần **1c.**

    c. getActionSize
        - Đầu vào: không.
        - Đầu ra: int64.
        - Trả về giá trị bằng tổng số action có thể thực hiện.

    d. getValidActions (đọc lưu ý bên dưới trước khi đi xây dựng hàm này)
        - Đầu vào: state.
        - Đầu ra: array có độ dài bằng getActionSize(), int64.
        - Trả về array, giá trị bằng 1 tại những action có thể thực hiện được, còn lại bằng 0.

    e. stepEnv
        - Đầu vào: env, action.
        - Đầu ra: env mới.
        - Trả về env ở trạng thái mới sau khi áp dụng action.

    f. getAgentSize
        - Đầu vào: không.
        - Đầu ra: int64.
        - Trả về giá trị bằng tổng số Agent tham gia đấu.

    g. checkEnded
        - Đầu vào: env.
        - Đầu ra: int64.
        - Trả về giá trị, -1 nếu chưa kết thúc game, từ 0 đến getAgentSize()-1 nếu có người chiến thắng (giá trị chỉ định người chiến thắng).

h. getReward
- Đầu vào: state.
- Đầu ra: int64.
- Trả về giá trị, 0 nếu chưa kết thúc game, 1 nếu kết thúc game và chiến thắng, -1 nếu kết thúc game và thua cuộc.

i. getStateSize
- Đầu vào: không.
- Đầu ra: int64.
- Trả về giá trị bằng độ dài của state.

j. one_game (không cần viết numba)

Nhận vào một list các Agent và file_per, khởi tạo và cho các Agent thi đấu, trả ra người chiến thắng và file_per (chi tiết: xem file **env.py**, hàm **run.py** trong hệ thống mẫu).

k. normal_main (không cần viết numba)

Nhận vào một list các Agent, số lần num_match và file_per. Cho các Agent đấu num_match trận và trả ra kết quả chiến thắng của các Agent, file_per.

l. numba_main_2

Hàm chạy cho một người chơi.

Hướng dẫn viết hàm numba_main_2:

Ví dụ với game splendor

Game này là game 4 người nên, đầu vào hàm n_game_numba sẽ có các file per: per1, per2, per3 và các hàm của các người chơi load từ hệ thống là p1, p2, p3

Có thể copy hết phía dưới các hàm, chỉ cần chỉnh sửa hàm one_game_numba theo cách code:

```
@njit()
def one_game_numba(p0, list_other, per_player, per1, per2, per3, p1, p2, p3):
```

```
env, lv1, lv2, lv3 = generate()
initEnv(env, lv1, lv2, lv3)

while env[154] <= 400:
    idx = env[154]%4
    player_state = getAgentState(env, lv1, lv2, lv3)
    list_action = getValidActions(player_state)

    if list_other[idx] == -1:
        action, per_player = p0(player_state,per_player)
    elif list_other[idx] == 1:
        action, per1 = p1(player_state,per1)
    elif list_other[idx] == 2:
        action, per2 = p2(player_state,per2)
    elif list_other[idx] == 3:
        action, per3 = p3(player_state,per3)



    if list_action[action] != 1:
        raise Exception('Action không hợp lệ')
    stepEnv(action, env, lv1, lv2, lv3)
    if checkEnded(env) != 0:
        break

turn = env[154]
for idx in range(4):
    env[154] = idx
```

```
        if list_other[idx] == -1:
            p_state = getAgentState(env, lv1, lv2, lv3)
            p_state[164] = 1
            act, per_player = p0(p_state, per_player)
    env[154] = turn
    winner = False
    if np.where(list_other == -1)[0] ==  (checkEnded(env) - 1): winner = True
    else: winner = False
    return winner,  per_player



@njit()
def random_Env(p_state, per):
    arr_action = getValidActions(p_state)
    arr_action = np.where(arr_action == 1)[0]
    act_idx = np.random.randint(0, len(arr_action))
    return arr_action[act_idx], per


@njit()
def n_game_numba(p0, num_game, per_player, list_other, per1, per2, per3, p1, p2,
p3):
    win = 0
    for _n in range(num_game):
        np.random.shuffle(list_other)
        winner,per_player  = one_game_numba(p0, list_other, per_player, per1,
per2, per3, p1, p2, p3)
        win += winner
```

```
    return win, per_player


import importlib.util, json, sys

from setup import SHOT_PATH


def load_module_player(player):

    return  importlib.util.spec_from_file_location('Agent_player',
f"{SHOT_PATH}Agent/{player}/Agent_player.py").loader.load_module()


@jit
def numba_main_2(p0, n_game, per_player, level, *args):
    list_other = np.array([1, 2, 3, -1])
    if level == 0:
        per_agent_env = np.array([0])
        return n_game_numba(p0, n_game, per_player, list_other, per_agent_env,
per_agent_env, per_agent_env, numbaRandomBot, numbaRandomBot, numbaRandomBot)
    else:
        env_name = sys.argv[1]
        if len(*args) > 0:
            dict_level =
json.load(open(f'{SHOT_PATH}Log/check_system_about_level.json'))
        else:
            dict_level = json.load(open(f'{SHOT_PATH}Log/level_game.json'))

        if str(level) not in dict_level[env_name]:
            raise Exception('Hiện tại không có level này')

        lst_agent_level = dict_level[env_name][str(level)][2]
        p1 = load_module_player(lst_agent_level[0]).Test
        p2 = load_module_player(lst_agent_level[1]).Test
        p3 = load_module_player(lst_agent_level[2]).Test
        per_level = []
        for id in range(getAgentSize()-1):
            data_agent_env =
list(np.load(f'{SHOT_PATH}Agent/{lst_agent_level[id]}/Data/{env_name}_{level}/Tra
in.npy',allow_pickle=True))
            per_level.append(data_agent_env)
```

```
        return n_game_numba(p0, n_game, per_player, list_other, per_level[0],
per_level[1], per_level[2], p1, p2, p3)
```

Lưu ý 2.1: **getValidActions** và **stepEnv** là hai hàm quan trọng nhất và thường tiêu tốn nhiều thời gian chạy của hệ thống, phần code của hai hàm này cũng thường nhiều nhất và cũng phức tạp nhất trong số tất cả các hàm. Do đó, ta cần chia nhỏ thành các phần để code và thiết kế các unit test cho từng phần. Sau đây là các bước làm:

- Liệt kê những trường hợp có thể xảy ra khi thực hiện getValidActions, chia nhỏ các trường hợp đến một mức độ chi tiết nhất định. Chú ý các trường hợp hiếm, đặc biệt.
- Với mỗi trường hợp đã liệt kê ở trên, code hàm getValidActions trong trường hợp đó.
- Khởi tạo env và gán các giá trị thích hợp để env trở thành trường hợp đang xét, chạy hàm getValidActions vừa code để kiểm tra xem kết quả trả về có đúng như logic khi thiết kế không (đây là bước tạo một unit test trường hợp đang xét).
- Thực hiện code hàm stepEnv cho trường hợp đang xét. Chạy stepEnv với env được khởi tạo ở bước liền trên và các action được trả ra ở hàm getValidActions. Kiểm tra xem env có thay đổi đúng theo logic khi thiết kế không
- Tiếp tục quay trở lại bước 2 để code các trường hợp khác, cho đến khi hết các trường hợp liệt kê ra.

Ví dụ trong Splendor, ta chia các trường hợp như sau để code:

- Phase 0:
    - Trường hợp chọn hành động (stepEnv)
    - Trường hợp chọn bỏ qua lượt (khi nào getValidActions có hành động này)
- Phase 1:
    - Trường hợp Board còn nguyên liệu
        - Trường hợp đã lấy 2 nguyên liệu cùng loại (stepEnv)
        - Trường hợp đã lấy đã 2 nguyên liệu khác loại (stepEnv)
    - Trường hợp Board hết nguyên liệu
- Phase 2:
    - Trường hợp úp thẻ trong chồng thẻ ẩn
    - Trường hợp úp thẻ trên bàn chơi
- Phase 3:
    - Trường hợp mua thẻ trên bàn
        - Trường hợp chồng thẻ ẩn còn (stepEnv)
        - Trường hợp chồng thẻ ẩn hết (stepEnv)
    - Trường hợp mua thẻ đang giữ (úp)

- Phase 4:
    - Trường hợp vẫn còn thừa nguyên liệu (stepEnv)
    - Trường hợp đã trả đủ số nguyên liệu thừa (stepEnv)

Các unit test ở hệ thống mẫu không lưu lại. Nhưng trong quá trình code hệ thống, cần lưu lại các unit test để check khi cần thiết.

Sau khi viết xong tất cả các hàm trong hệ thống. Thực hiện chạy khoảng 10000 lần hàm run với toàn bộ bot random để check bug. Xong bước này thì có thể gặp tester để thực hiện check luật chơi.

Lưu ý 2.2: Trong quá trình viết các hàm của hệ thống:

- Có thể thêm các thông tin vào cả env và state để tối ưu thời gian chạy (giả sử một thông tin nào đó cần truyền vào state nhưng ít khi bị thay đổi, tuy nhiên lại rất mất thời gian để tính toán, có thể lấy ví dụ là tính toán tỉ lệ trao đổi tài nguyên với ngân hàng của mỗi người chơi trong game Catan).
- Hạn chế tối đa dùng kiểu dữ liệu list. Thay vào đó, khởi tạo array có độ dài bằng số phần tử tối đa của list. Phần tử nào được thêm vào thì ta thay giá trị tại index tương ứng trong array bằng 1.
- Một số hàm trong numpy có tốc độ chậm (ví dụ như append, concatenate, choice) có thể thay thế bằng đoạn code khác giúp chạy nhanh hơn (gặp người hướng dẫn, sếp).

3. Check luật chơi

Thực hiện visualize bàn chơi bằng cách viết các lệnh in các thành phần của env, state ra màn hình theo dạng các dictionary thể hiện thông tin và bắt đầu check luật cùng tester.

4. Chạy các bài test và viết báo cáo về hệ thống

Các kết quả của các bài test dưới đây được lấy khi chạy hệ thống Splendor.
Tạo file system_test.py và chạy các đoạn code theo hướng dẫn dưới đây, sau đó ghi kết quả.

a. Tốc độ chạy trung bình 1000 trận với bot thường

```
from env import *
from time import time

def bot_random(state, temp_file, per_file):
    validActions = getValidActions(state)
    arrActions = np.where(validActions == 1)[0]
    choose_idx = np.random.randint(0, len(arrActions))
    return arrActions[choose_idx], temp_file, per_file

list_player = [bot_random] * getAgentSize()
```

```
t_ = time()
a, b = main(list_player, 1, [0])
print('load numba', time() - t_)

t_ = time()
a, b = main(list_player, 10000, [0])
print('1000 games', (time() - t_)/10)
```

Kết quả:

b. Chạy được 1 triệu trận trong 1 session colab
Import và chạy main với bot random 1 triệu trận trên Colab.

c. Chuẩn form (xem chuẩn đầu vào – đầu ra của hệ thống)

d. Đúng luật (đã check ở phần 3 – gặp tester)

e. Tối ưu hóa độ lớn state * getActionSize()
Tham khảo ý kiến của người hướng dẫn (hoặc sếp) để xem thiết kế đã tối ưu chưa.

f. Số lần truyền vào Agent trung bình một trận

```
from env import *

def bot_random(state, temp_file, per_file):
    per_file += 1
    validActions = getValidActions(state)
    arrActions = np.where(validActions == 1)[0]
    choose_idx = np.random.randint(0, len(arrActions))
    return arrActions[choose_idx], temp_file, per_file

list_player = [bot_random] * getAgentSize()

a, b = main(list_player, 10000, 0)
print(b / 10000)
```

Kết quả:

g. Tốc độ chạy các hàm mà Agent dùng

```
from env import *
from time import time

def bot_random(state, temp_file, per_file):
    for i in range(10):
        validActions = getValidActions(state)

    for i in range(5):
        getReward(state)

    arrActions = np.where(validActions == 1)[0]
    choose_idx = np.random.randint(0, len(arrActions))
    return arrActions[choose_idx], temp_file, per_file

list_player = [bot_random] * getAgentSize()

t_ = time()
a, b = main(list_player, 1, [0])
print('load numba', time() - t_)

t_ = time()
a, b = main(list_player, 10000, [0])
print('1000 games', (time() - t_)/10)
```

Kết quả:

h. Tổng số ván đấu = tổng số lần getReward khác 0 của người chơi

```
from env import *

def bot_random(state, temp_file, per_file):
    validActions = getValidActions(state)
    arrActions = np.where(validActions == 1)[0]
    choose_idx = np.random.randint(0, len(arrActions))

    reward = getReward(state)
    if reward != 0:
        if reward == -1:
            per_file[0] += 1
        else:
            per_file[1] += 1

    return arrActions[choose_idx], temp_file, per_file
```

```
list_player = [bot_random] * getAgentSize()

a, b = main(list_player, 1000, [0,0])
print(b)
```

Kết quả:

Lưu ý: Trường hợp ra không bằng nhau thì cần phải giải thích được là tại sao, tham khảo ý kiến của người hướng dẫn hoặc sếp để tìm cách khắc phục.

i. Giải thích ý nghĩa của các giá trị trong env, state và actions

Xem các file **design.txt**, **design_actions.txt**, **design_state.txt** trong hệ thống mẫu.

_____ Hết _____