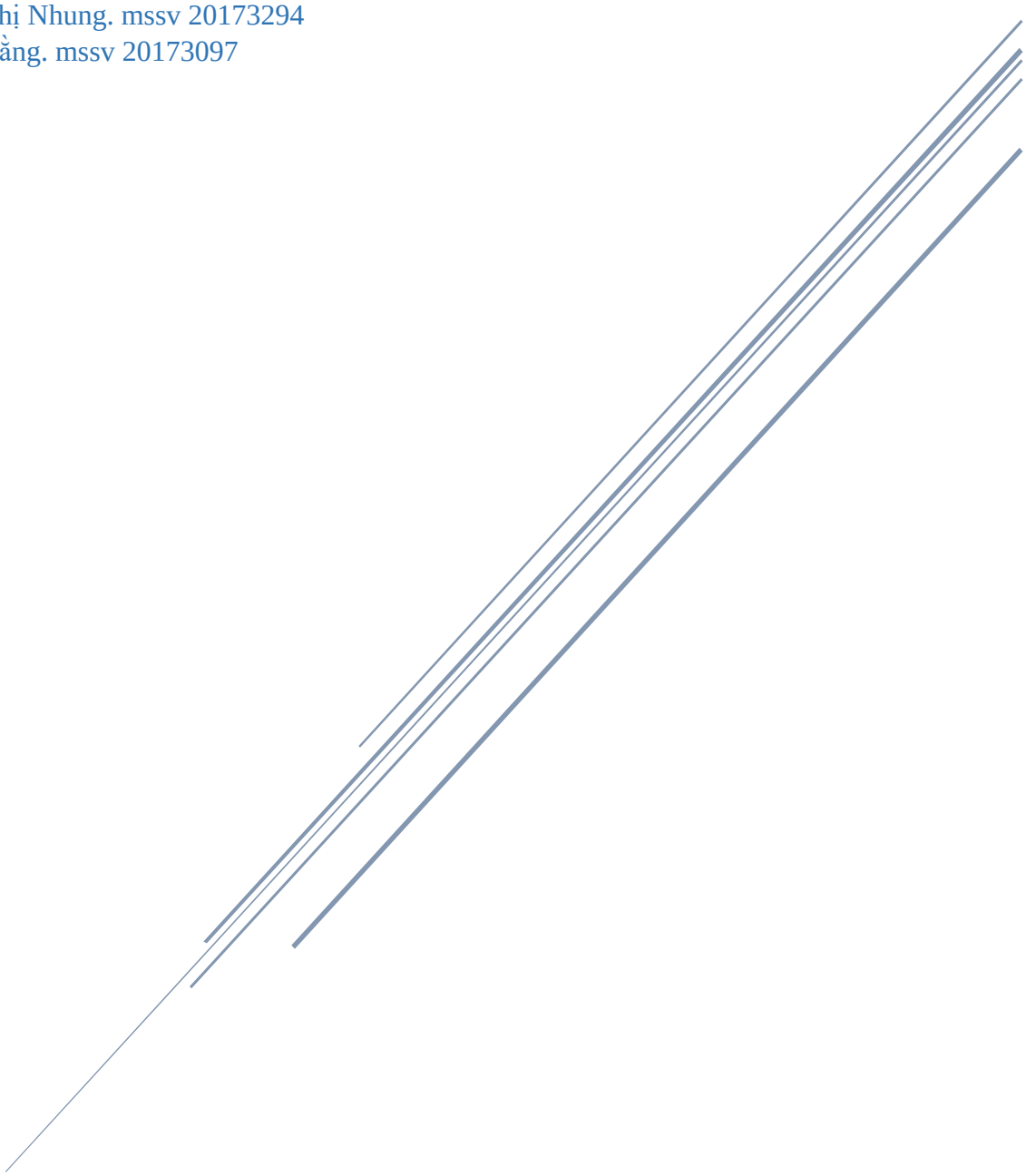
HÀM THÔNG MINH CHƠI GAME CUỘC CHIẾN LÃNH THỔ

BÁO CÁO BÀI TẬP LỚN NHẬP MÔN TRÍ TUỆ NHÂN TAO

Mã lớp: 111561
Giảng viên: ThS. Ngô Văn Linh

Thực hiện:
Ngô Việt Hoàng. mssv 20173142
Vũ Hồng Phúc. mssv 20173305
Nguyễn Thị Nhung. mssv 20173294
Bùi Thị Hằng. mssv 20173097



Đại học Bách Khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

LỜI MỞ ĐẦU

Cuộc thi Cuộc chiến lãnh thổ là một cuộc thi rất thú vị khi mà chúng em không chỉ được thử sức mình với một bài toán mới mẻ, được đem những kiến thức đã học trong môn Nhập môn Trí tuệ nhân tạo ra để áp dụng, mà còn là cơ hội cho chúng em được cọ sát với các đội khác cũng tham gia cuộc thi.

Khi thực hiện bài tập lớn này, nhóm em đã có sự phân công công việc như sau:

Ngô Việt Hoàng (nhóm trưởng): Xây dựng thuật toán khi game chưa phân vùng (Q - learning)

Vũ Hồng Phúc: Thiết kế thuật toán tổng quát, xây dựng target network và phần experience replay.

Nguyễn Thị Nhung: Khảo sát ban đầu, train target network, ghi nhận và biểu diễn kết quả thực nghiệm, cài đặt client, lập báo cáo.

Bùi Thị Hằng: Xây dựng thuật toán khi game đã phân vùng (DFS, BFS)

MỤC LỤC

CHƯƠNG 1. TỔNG QUAN VỀ DỰ ÁN VÀ CƠ SỞ LÍ THUYẾT

1.1 Một số khái niệm

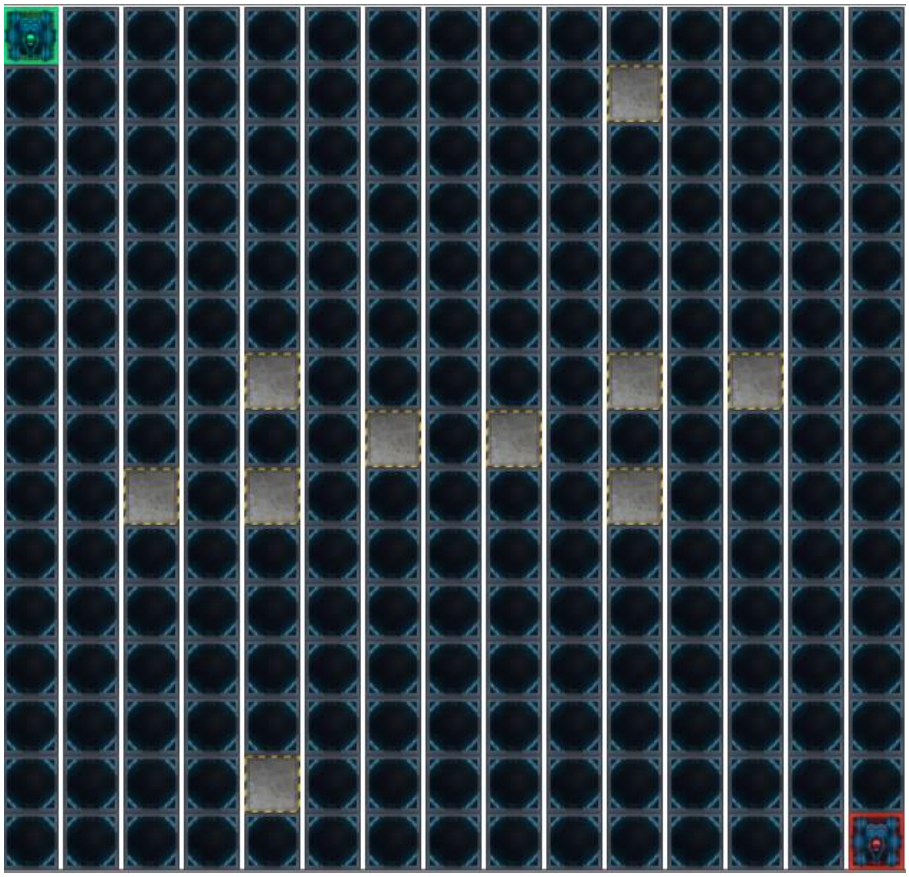
1.1.1 Game cuộc chiến lãnh thổ

Đây là một game thuộc loại game đối kháng, chơi trên một map có kích thước 15*15, map này được biểu diễn thành một ma trận số nguyên với mỗi ô sẽ mang giá trị biểu thị trạng thái của ô đó, tương ứng là:

- Ô còn trống (space): 0
- Ô có quân xanh đã đi qua (green): 1
- Ô có quân đỏ đã đi qua (red): 2
- Ô là vật cản (wall): 3

Map sẽ có 10 chướng ngại vật được đặt ngẫu nhiên, đối xứng nhau qua tâm. Mỗi đội chơi sẽ xuất phát từ một góc màn hình và di chuyển theo lượt. Mỗi lượt chỉ được đi tới 1 trong 3 ô lân cận vị trí hiện tại. Đội nào hết nước đi trước là đội thua cuộc

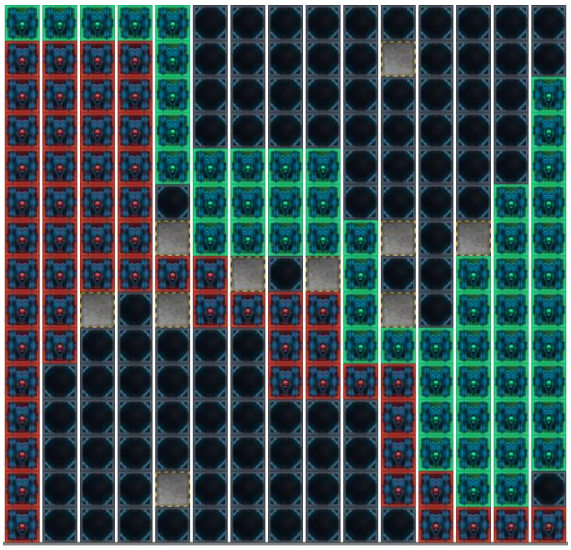
Nhiệm vụ của tác tử trong trò chơi này là nhận về map và vị trí hiện tại của mình và của đối thủ, trả về vị trí tiếp theo mà mình dự định sẽ đi.



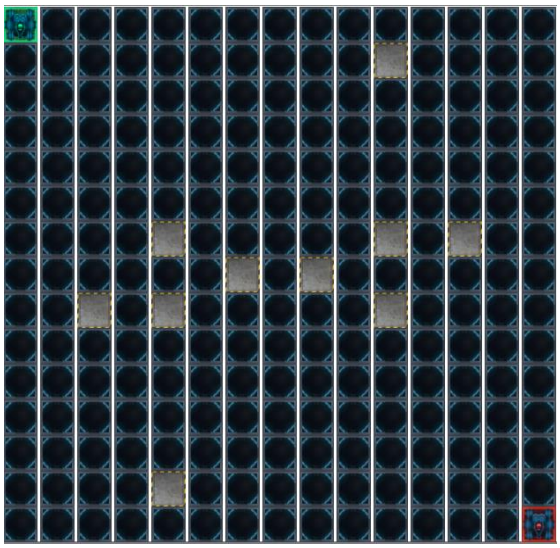
Hình 1. Trạng thái bắt đầu của game

1.1.2 Hai trạng thái của game

Game được chia thành 2 giai đoạn, chưa phân vùng và đã phân vùng (Xem hình dưới)



Hình 1. Game đã phân vùng



Hình 3. Game chưa phân vùng

1.2 Lí thuyết về học tăng cường và Deep Q-Network

Để tìm kiếm nước đi trong khi map chưa phân vùng, chúng em sử dụng thuật toán học tăng cường sâu (Deep Q-Network - DQN).

1.2.1 Một số khái niệm về học tăng cường

Agent (tác tử): đối tượng thực hiện nước đi trong game

Environment (môi trường): không gian mà agent tương tác

State (trạng thái): 1 trạng thái của môi trường giữa hai lần thực hiện nước đi liên tiếp của 2 đối thủ

Policy (chính sách): chiến thuật lựa chọn nước đi của agent để đạt được mục đích

Reward (phần thưởng): phần thưởng agent nhận lại được khi thực hiện một action a

Episode (tập): một chuỗi các cặp trạng thái, hành động từ khi bắt đầu tới khi kết thúc trò chơi (ở trong game này là đến khi map được phân vùng)

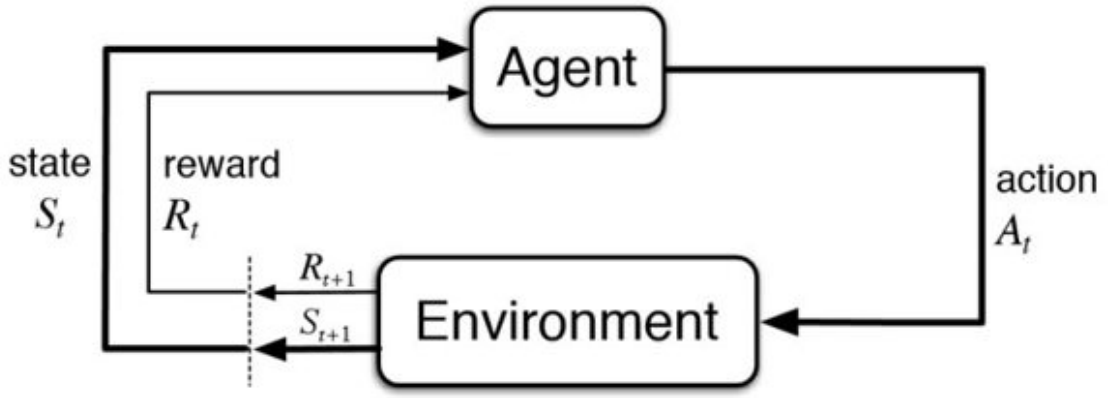
Reward function: Hàm đáng giá reward sau mỗi nước đi, cho biết độ tốt của hành động (trạng thái) tại một thời điểm.

Value function: Hàm dự đoán reward của toàn bộ quá trình hành động của agent, là tổng tất cả các reward mà agent kì vọng sẽ đạt được từ trạng thái hiện tại cho tới trạng thái kết thúc

Model (*Target network*): Một mạng neural dùng để mô phỏng lại quy luật của môi trường, đưa ra dự đoán reward sẽ nhận được nếu agent thực hiện một hành động a khi đang ở trạng thái s, từ đó giúp đưa ra hành động sẽ được thực hiện trên thực tế.

Học tăng cường nói chung và DQN nói riêng là một loại hình học máy theo cách thử và sai. Tác tử (agent) thực hiện hành động (action) để tương tác với môi trường (environment) và nhận lại từ môi trường một phần thưởng (reward) tương ứng với hành động vừa thực hiện.

Nhiệm vụ của tác tử là tìm ra chiến lược lựa chọn hành động (policy) để cực đại hóa phần thưởng tích lũy nhận được sau mỗi ván chơi



Hình 4. Sơ đồ quá trình tương tác giữa tác tử và môi trường

1.2.2 Thuật toán học tăng cường sâu (DQN)

1.2.2.1 Q – learning

Xét một chuỗi state - action từ trạng thái bắt đầu đến trạng thái kết thúc

$\mathcal{T} = (s_1, a_1, s_2, a_2, \ldots, s_T, a_T)$

Giả sử game đang ở trạng thái s_t ($t \in \{1, 2, \ldots, T\}$), xác suất agent thực hiện hành động a_t là $\pi(a_t|s_t)$, phần thưởng nhận được khi thực hiện hành động này là $r(a_t|s_t)$.

Ta gọi $Q_\pi(s_t, a_t)$ là giá trị value function **dự đoán** sẽ nhận được khi thực hiện hành động a_t tại state s_t:

$$Q_\pi(s_t, a_t) = E_\pi[G_t \mid S = s_t, A = a_t]$$

Trong đó $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ là tổng các reward nhận được từ trạng thái hiện tại (s_t) tới khi kết thúc. γ là discount rate, $0 < \gamma < 1$, một giá trị cho biết độ phụ thuộc của quyết định vào các reward trong tương lai, γ càng nhỏ thì quyết định của agent sẽ càng có tính "tham lam" hơn, tức là phụ thuộc nhiều hơn vào các nước đi ở gần mà ít quan tâm đến các nước đi ở xa.

Tại trạng thái s_t sẽ có nhiều action $a_{t1}, a_{t2}, \ldots a_{tn}$, ta sẽ lựa chọn action để trả về cho môi trường một cách tham lam, nghĩa là chọn action có giá trị $Q_\pi(s_t, a_t)$ lớn nhất (policy $\pi(a_t|s_t)=1$ nếu $a_t = \text{argmax}_{at}\, Q_\pi(s_t,a_t)$).

Sau khi thực hiện action a_t, môi trường sẽ trả lại cho ta một phần thưởng $r(a_t|s_t)$, do đó giá trị value function **thực tế** sẽ là

$$Q(s_t, a_t) = r(a_t|s_t) + \gamma * \max(Q_\pi(s_{t+1}, a_{t+1})) \qquad (1)$$

Trong đó $\max(Q_\pi(s_{t+1}, a_{t+1}))$ là value function kì vọng sẽ nhận được ở state $s + 1$, cũng với policy trên. Công thức (1) có thể chứng minh dựa trên phương trình Bellman, tuy nhiên trong khuôn khổ báo cáo này chúng em xin phép không đề cập.

Chúng ta mong muốn giá trị Q dự đoán và thực tế càng gần nhau càng tốt, do đó ta xác định một đại lượng biểu thị khoảng cách giữa hai giá trị này, gọi là Temporal Difference (TD)

$$TD(s_t, a_t) = Q(s_t, a_t) - Q_\pi(s_t, a_t)$$

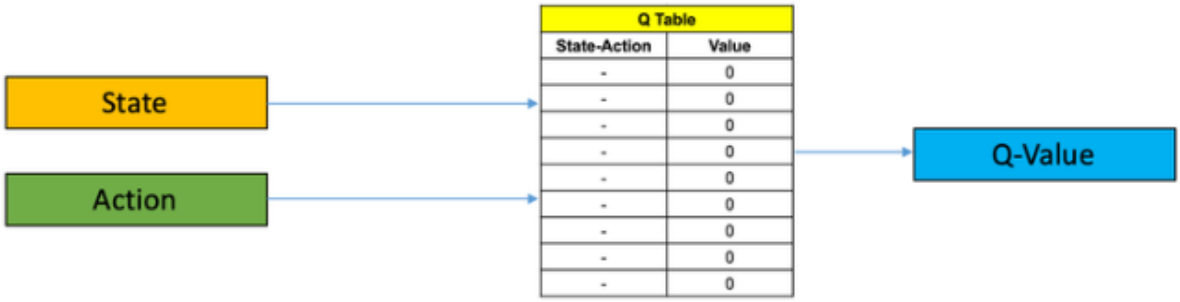
Nhiệm vụ của chúng ta lúc này là tìm cách tối ưu hóa giá trị TD này.

Trong thuật toán Q-learning truyền thống, người ta lưu giá trị Q ứng với các cặp state – action vào một bảng gọi là Q-table. Mỗi lần thực hiện hành động, người ta sẽ truy xuất vào bảng này để tìm các action tương ứng với state hiện tại và chọn action thích hợp. Sau khi thực hiện, tác tử nhận lại reward từ môi trường và cập nhật vào bảng giá trị mới của Q theo công thức:

$$Q_\pi(s_t, a_t) = Q_\pi(s_t, a_t) + \alpha * [r(a_t|s_t) + \gamma * \max(Q_\pi(s_{t+1}, a_{t+1})) - Q_\pi(s_t, a_t)]$$

Trong đó α là learning rate. Giá trị $Q_\pi(s_t, a_t)$ sẽ hội tụ dần về $Q(s_t, a_t)$ sau nhiều lượt chơi.

Nhược điểm lớn nhất của phương pháp này là gây tốn kém bộ nhớ và mất thời gian khi truy xuất bảng Q-table.



Hình 5. Bảng Q-table

1.2.2.2 Các phương pháp lựa chọn hành động

Trước khi trình bày về phương pháp DQN, em xin đề cập một số vấn đề khác của phương pháp Q-learning.

Chúng ta xác định hai khái niệm sau:

Exploitation (Khai thác): thực hiện hành động dựa trên kinh nghiệm trong quá khứ

Exploration (Khám phá): tìm kiếm hành động mới để tích lũy kinh nghiệm

Trong quá trình học, chúng ta luôn phải tìm cách cân bằng giữa khai thác và khám phá để đảm bảo agent vừa chọn được hành động tốt nhất dựa theo kinh nghiệm, cũng như học được những kinh nghiệm mới. Người ta đưa ra rất nhiều phương pháp lựa chọn hành động ở mỗi nước đi để cân bằng giữa 2 yếu tố này, như phương pháp greedy, ε – greedy, softmax, reinforcement comparison, pursuit,… Tuy nhiên trong khuôn khổ báo cáo này, chúng em chỉ xin trình bày hai phương pháp đơn giản và phổ biến nhất là greedy và ε – greedy

Phương pháp greedy (tham lam)

Trong phương pháp này, action được chọn luôn là action có giá trị Q(s, a) lớn nhất, do đó ta thấy ngay nhược điểm của nó là chỉ có khai thác mà không có khám phá.

Phương pháp ε – greedy

Trong phương pháp này, ta xác định một giá trị $0 < \varepsilon < 1$. Ta sẽ lựa chọn hành động ngẫu nhiên với xác suất ε, ngược lại với xác suất $1 - \varepsilon$ ta sẽ lựa chọn hành động có giá trị Q(s, a) lớn nhất. Phương pháp này đảm bảo được có cả khai thác và khám phá, tuy nhiên nhược điểm là mọi hành động sẽ được khám phá với xác suất như nhau, bất kể hành động đó đã được lựa chọn nhiều hay ít ở những lượt lựa chọn trước đó.

1.2.2.3 Phương pháp xấp xỉ giá trị Q bằng mạng neural

Như đã trình bày ở trên, phương pháp Q-learning truyền thống gây tốn tài nguyên rất lớn khi lưu trữ giá trị reward cho mỗi cặp state – action, do đó người ta tìm cách xấp xỉ giá trị reward này bằng một mạng neural, gọi là target network.

Target network có nhiệm vụ xấp xỉ giá trị hàm Q sao cho giá trị này của mạng càng gần giá trị thực tế càng tốt, do đó ta có thể chọn loss function là giá trị $TD^2(s, a)$ đã đề cập bên trên:

$$J(\theta) = TD_\theta^2(s, a) = (r(a_t|s_t) + \gamma * \max(Q_\theta(s_{t+1}, a_{t+1})) - Q_\theta(s_t, a_t))^2$$

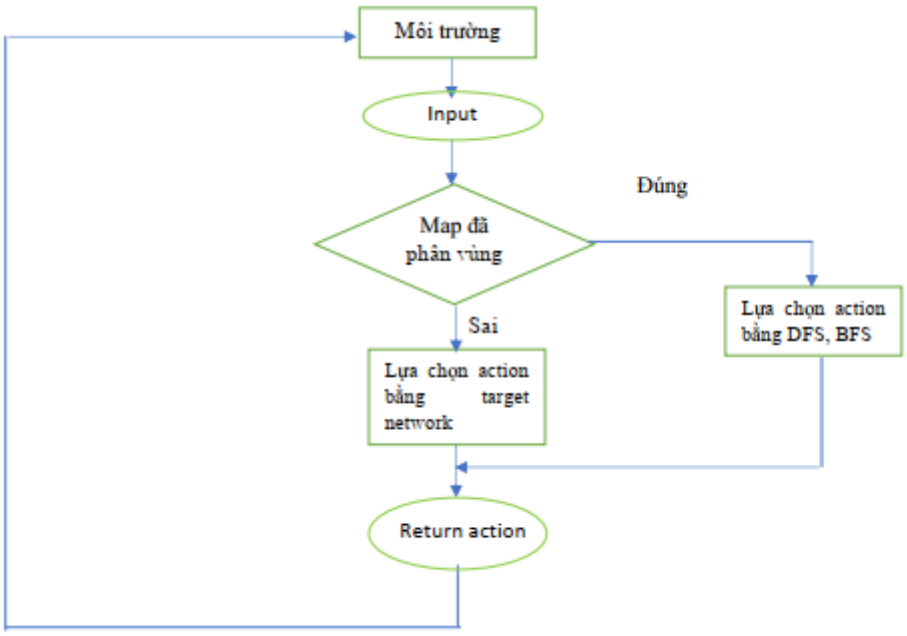
Trong đó θ là bộ tham số của target network. Ta sẽ tối ưu loss function bằng cách cập nhật bộ tham số này dựa trên reward nhận được sau mỗi hành động.

1.2.2.4 Experience replay

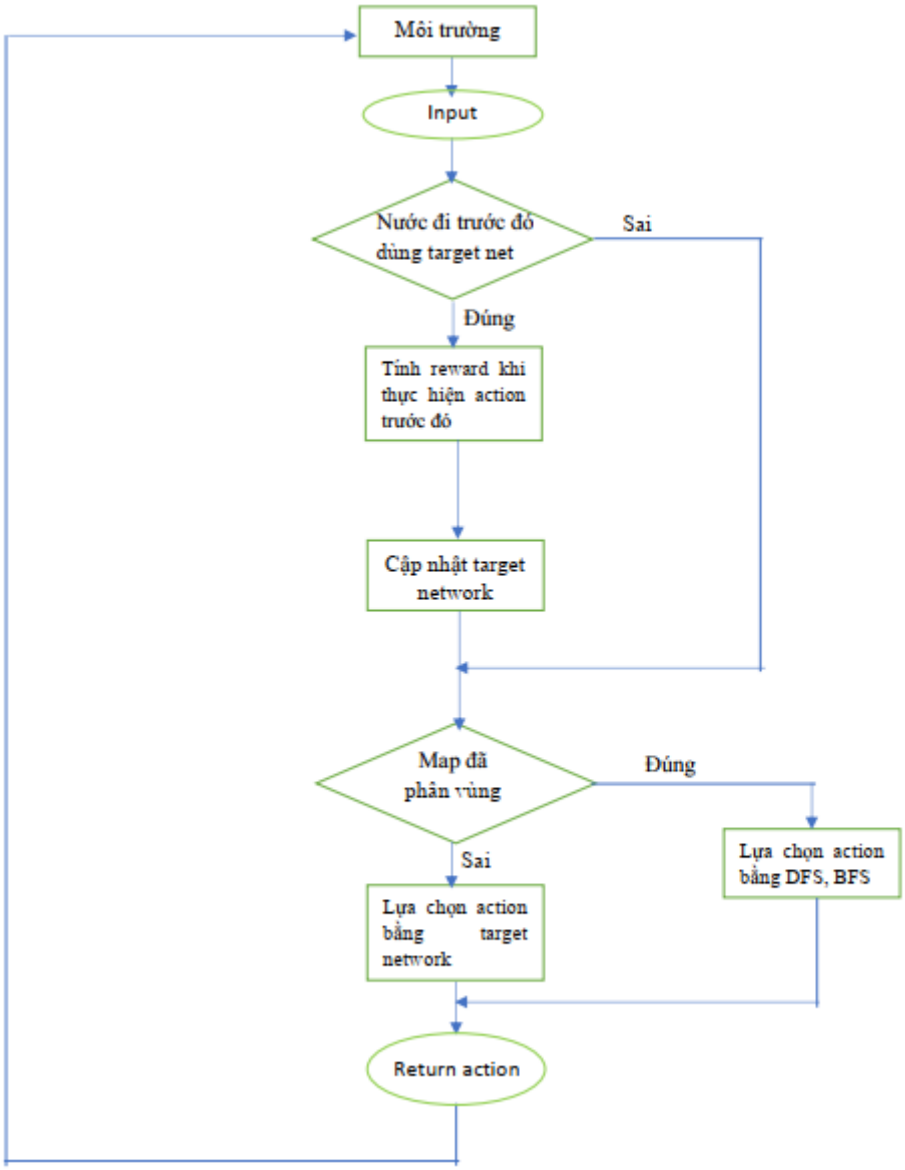
Một vấn đề đặt ra là khi train target network, ta sẽ truyền vào từng state liên tiếp nhau, những state này có rất ít điểm khác nhau (do sau mỗi action của tác tử, sự thay đổi của môi trường thường rất nhỏ). Điều này rất dễ gây nên hiện tượng overfitting. Để giải quyết vấn đề này, ta có thể tạo một replay buffer: một list các bộ (s_t, a_t, r, s_{t+1}) khác nhau, sau mỗi lượt chơi, thay vì lấy giá trị reward vừa nhận được để train target network, ta lưu giá trị này vào buffer, sau đó lấy ngẫu nhiên một số giá trị khác từ buffer ra để train.

1.3 Sơ đồ thiết kế thuật toán

Thuật toán khi chiến đấu



Thuật toán khi train



CHƯƠNG 2. CÁCH THỨC XÂY DỰNG CHƯƠNG TRÌNH

2.1 Trạng thái game chưa phân vùng

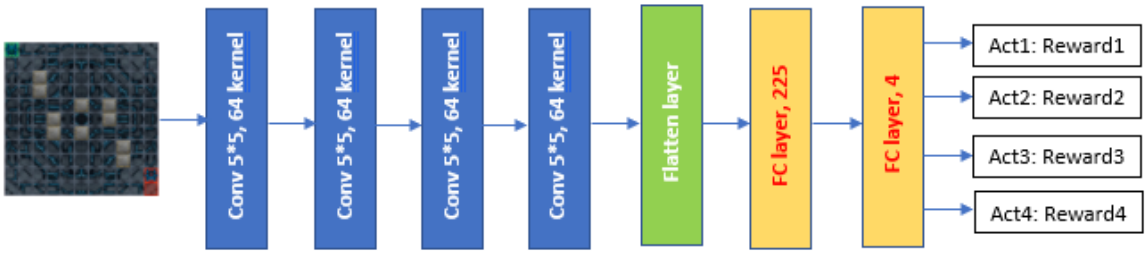
2.1.1 Tính reward nhận được sau mỗi action

Sau khi gửi nước đi đến môi trường, tác tử chờ đợi đối thủ thực hiện nước đi tiếp theo và chờ đợi môi trường phản hồi. Sau khi nhận được phản hồi của môi trường (bao gồm map hiện tại, tọa độ của tác tử và của đối thủ), tác tử sẽ tiến hành kiểm tra xem map đã phân vùng hay chưa. Nếu chưa phân vùng, reward = 0; nếu đã phân vùng, tiến hành đếm số ô trong vùng hiện tại của tác tử (gọi là m) và của đối thủ (gọi là e) bằng BFS, sau đó xét hiệu m – e. Nếu
m – e > 0, reward = 1, nếu m – e = 0, reward = 0.5, nếu m – e < 0, reward = -1.

2.1.2 Target network

2.1.2.1 Kiến trúc target network

Target network nhận đầu vào là state hiện tại và trả về đầu ra là vector 4 chiều ứng với reward nhận được nếu thực hiện di chuyển đến ô tương ứng trong nước đi tiếp theo. Kiến trúc của target network như hình vẽ dưới đây.



Hình 6. Kiến trúc Target network

2.1.2.2 Cập nhật target network

Ta phải tiến hành cập nhật target network trước khi đưa ra nước đi tiếp theo là do game Cuộc chiến lãnh thổ là game đối kháng, do đó việc đánh giá hiệu quả của nước đi (nhận reward) chỉ có thể thực hiện sau khi đối thủ cũng đã thực hiện nước đi của mình thì mới đem lại hiệu quả.

Việc train neural network được thực hiện theo phương pháp experience replay đã đề cập ở chương 1. Ta tạo 1 buffer có kích thước 50.000 phần tử để lưu các bộ (s, a, r, s'), khi train network, ta random 1 batch gồm 10 bộ bất kì từ buffer để làm input cho mạng.

Về việc lựa chọn tham số: γ được chọn giá trị cố định là 0.95, ε được thay đổi, trong 1000 ván đầu, ε được chọn bằng 0.8, đảm bảo agent học được càng nhiều trạng thái mới càng tốt, trong 3000 ván sau đó, ε được lấy bằng 0.5, từ ván thứ 4001 trở đi, khi agent đã học được tương đối nhiều kinh nghiệm, ε được chọn bằng 0.2.

2.1.2.3 Lựa chọn nước đi tiếp theo

Ta lựa chọn nước đi theo phương pháp ε – greedy. Mã giả:

```
int make_move(Map map, int x, int y, int xe, int ye) {
        int action = -1;
        int eps = 0.2;
        float a: a random float between 0 and 1;
        boolean check_action(Map map, int x, int y, action a);        // is action a feasible?
        float predict[4]: the predict vector at state s;
        if (a < eps) {
                do
                        action: a random int between 0 and 4
                while (!check_action(map, x, y, action));
        }
        else {
                float max = -10000;
                for (i from 0 to 4) {
                        if(check_action(map, x, y, i)) {
                                max = max(max, predict[i]);
                                action = i;
                        }
                }
        }
        return action;
}
```

2.2 Trạng thái game đã phân vùng

Khi game đã phân vùng, ta sẽ áp dụng thuật toán BFS và DFS để tìm đường đi. Lúc này việc đối thủ di chuyển thế nào không còn ảnh hưởng đến ta nữa, do đó mục đích của ta chỉ là đi được càng nhiều ô trong phân vùng của mình càng tốt.

Đầu tiên, với mỗi ô xung quanh vị trí hiện tại của tác tử, ta sử dụng DFS với độ sâu depth= 12, nếu như chưa đạt đến độ sâu này mà hết nước đi, ta sẽ trả về giá trị -depth, còn nếu đạt đến độ sâu này, ta sẽ trả về số ô trống còn lại trong phân vùng chứa vị trí có depth bằng 12 này (Đếm bằng BFS).

2.3 Thực nghiệm thuật toán

Chúng em đã tiến hành train target network với khoảng 20.000 ván đấu với đối thủ là bot được thầy cung cấp. Ưu điểm của việc lựa chọn đối thủ này so với việc tự chơi (self play) là target network sẽ học được nhanh hơn do đối thủ sử dụng thuật toán cắt tỉ alpha – beta tương đối điển hình và khá mạnh, tuy nhiên nhược điểm là dễ bị overfiting. Sau đây là kết quả thực nghiệm được chúng em trình bày dưới dạng sơ đồ (sau 1000 ván tỉ số được ghi nhận lại 1 lần)

TÀI LIỆU THAM KHẢO

Nguyễn Linh Giang, *Bài toán quyết định Markov với số bước hữu hạn và thuật toán học Q mờ,* Tạp chí Thông tin và Truyền thông, 2004

Châu Quang Mạnh, *Học tăng cường và quyết định Markov (Luận văn thạc sĩ)*, Trường Đại học Bách Khoa Hà Nội, 2009.

Wolfgang Konen, *Reinforcement Learning for Board Games: The Temporal Difference Algorithm*, Cologne University of Applied Sciences Steinm¨ullerallee 1, D-51643 Gummersbach, Germany, 2015

Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction,* The MIT Press, 2017

https://forum.machinelearningcoban.com/t/tutorial-implement-cac-thuat-toan-reinforcement-learning-ddpg-bai-1/3276

https://adventuresinmachinelearning.com/reinforcement-learning-tutorial-python-keras/

https://towardsdatascience.com/deep-reinforcement-learning-and-monte-carlo-tree-search-with-connect-4-ba22a4713e7a

https://viblo.asia/p/gioi-thieu-ve-hoc-tang-cuong-va-ung-dung-deep-q-learning-choi-game-cartpole-Az45bYy6lxY