

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
Khoa Công nghệ Thông tin



MÔN **CƠ SỞ TRÍ TUỆ NHÂN TẠO**

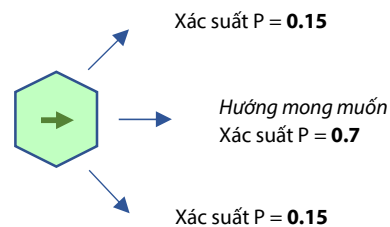
BÁO CÁO ĐỒ ÁN

DECISION MAKING

Sinh viên thực hiện
Vũ Công Thành (MSSV: 19120374)

Phát biểu bài toán

Bài toán Hex World là bài toán MDP đơn giản, trong đó chúng ta phải di chuyển trên các ô lục giác để đi đến trạng thái kết thúc (terminal states), mỗi ô trong bản đồ đại diện cho một trạng thái trong MDP (Markov Decision Process). Tại mỗi ô, agent có thể thực hiện 1 trong 6 hành động tương ứng với di chuyển sang các ô kề với 6 cạnh của ô hiện tại. Mỗi hành động đều có xác suất chuyển đổi giữa 2 trạng thái, cụ thể trong bài toán này sẽ là 70% chuyển đổi đúng hướng mong muốn, 30% còn lại chia đều cho 2 hướng kế bên hướng đang mong muốn. Chi phí để thực hiện một bước di chuyển là 1 đơn vị. Ngoài ra, khi cố gắng di chuyển sang một ô không tồn tại trên bản đồ thì sẽ bị trừ 1 chi phí, và vẫn đứng yên ở trạng thái hiện tại.



Theo mô tả bài toán từ [1], các khi đi vào các trạng thái kết thúc (terminal states) thì sẽ nhận được phần thưởng ở trạng thái tương ứng, sau đó được di chuyển đến một trạng thái cuối cùng. Tuy nhiên do trạng thái cuối cùng này không có thêm phần thưởng hay tác động gì đến bài toán, nên ta có thể xem là sau khi đi vào các trạng thái kết thúc thì agent đã hoàn thành được mục tiêu mà bài toán đặt ra.

Thách thức

- Bài toán này ít phổ biến (với hình dạng grid là hex) nên việc tìm kiếm môi trường có sẵn để thực nghiệm bài toán này cũng gặp nhiều khó khăn, do đó em đã tự xây dựng môi trường để agent có thể thực hiện các hành động trên đó.

Tổ chức mã nguồn

```
|   Algos.jl
|   Environments.jl
|   Main.jl
|
+---assets
|   QL_map1.gif
|   QL_map1_optimal.png
|   QL_map2.gif
|   QL_map2_optimal.png
|   QL_map3.gif
|   QL_map3_optimal.png
|   sarsa_map1.gif
|   sarsa_map1_optimal.png
|   sarsa_map2.gif
```

```

|      sarsa_map2_optimal.png
|      sarsa_map3.gif
|      sarsa_map3_optimal.png
|
\---maps
      map1.csv
      map2.csv
      map3.csv






```

Thư mục assets chứa các tài nguyên sinh ra trong quá trình chạy chương trình
 Thư mục maps chứa các bản đồ ở dạng file CSV
 File Algos.jl chứa các thuật toán huấn luyện
 File Environments.jl chứa môi trường của bài toán
 File Main.jl để gọi và thực thi các phương thức có trong Algos.jl và Environments.jl

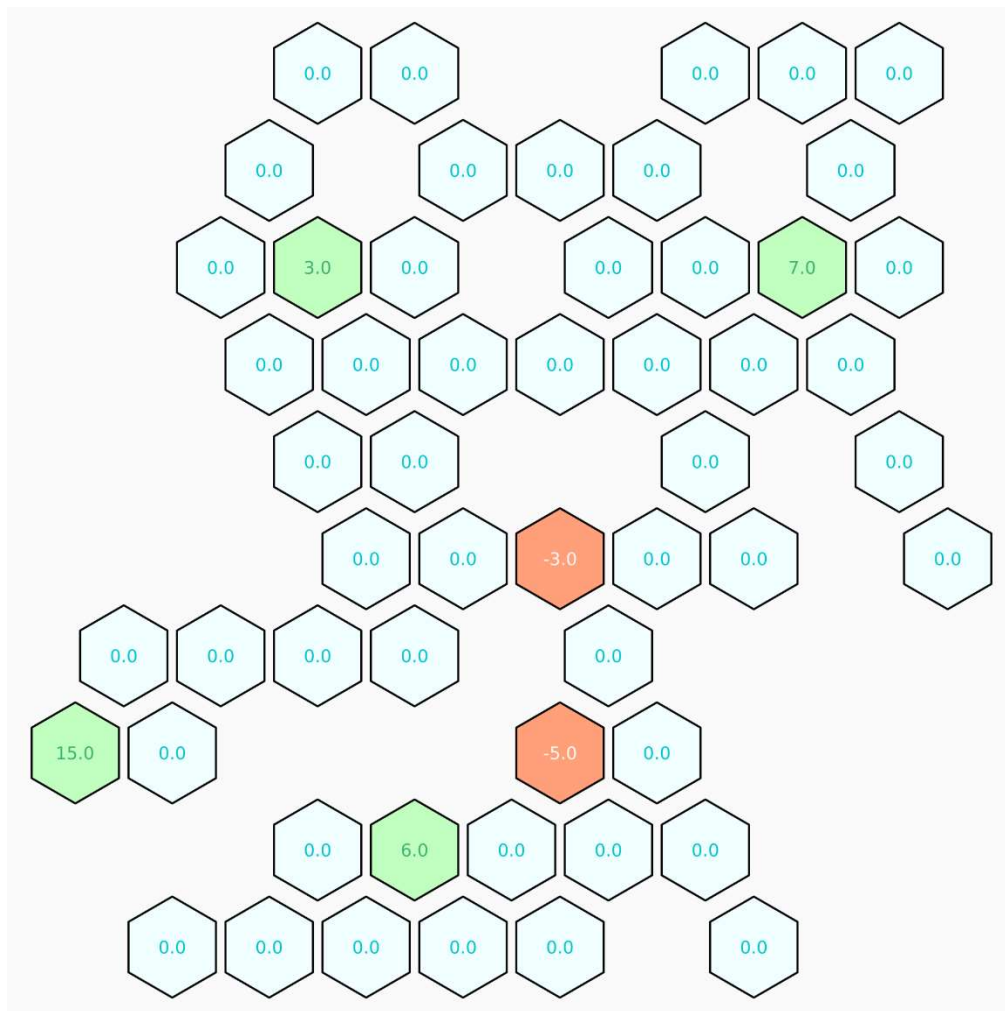
Mô hình hoá tính toán

Cấu trúc dữ liệu

HexEnv (môi trường bản đồ các ô lục giác)

- *Hệ toạ độ*: các ô trên bản đồ được đánh số bằng số thứ tự của nó theo hai trục Ox và Oy, chỉ số của mỗi ô có dạng (x, y), với mốc là ô nằm bên trái dưới cùng được đánh vị trí là (1, 1). Cách đánh chỉ số ô này tương tự với cách “odd-r” trong [2], điểm khác biệt là ô (1, 1) ở bên trái phía dưới thay vì bên trái phía trên. Ở cách đánh chỉ số này, những ô ở hàng chẵn (2, 4, 6, 8...) nằm lệch sang bên phải 0.5 đơn vị so với các ô ở hàng lẻ.
 Tập tin đầu vào của một bản đồ có dạng CSV và lần lượt có các cột là x, y, reward và terminal. Trong đó cột x, y lần lượt là vị trí theo trục Ox (chiều ngang) và Oy (chiều dọc) với vị trí gốc bên trái phía dưới là ô (1, 1), cột reward là phần thưởng đạt được khi vào trạng thái đó, cột terminal bằng 0 nếu trạng thái đó không phải là terminal state, ngược lại thì bằng 1.
- *Trực quan hoá*: các ô không phải trạng thái kết thúc được minh hoạ bằng màu xanh dương với các sắc độ khác nhau  , các ô trạng thái kết thúc có trọng số không âm được minh hoạ bằng màu xanh lá cây , các ô trạng thái kết thúc có trọng số âm được minh hoạ bằng màu đỏ , các ô biểu diễn một đường đi khi tìm đường đi từ một ô đến trạng thái kết thúc được minh hoạ bằng màu vàng . Các đường đi hoặc hướng đi tối ưu được biểu diễn bằng các dấu mũi tên.
- *Các hành động*: Có 6 hành động mà agent có thể thực hiện được đánh số từ 1 đến 6:
 1. ← Sang trái (left)
 2. ↖ Sang trái và lên trên (upper-left)
 3. ↗ Sang phải và lên trên (upper-right)
 4. → Sang phải (right)
 5. ↘ Sang phải và xuống dưới (lower-right)
 6. ↙ Sang trái và xuống dưới (lower-left)

Một bản đồ được render từ môi trường HexEnv:



Cấu trúc dữ liệu HexEnv bao gồm các thành phần:

Thuộc tính:

- `tiles::Matrix{Int64}`: Danh sách tất cả các ô trong bản đồ (bao gồm tọa độ, trọng số (reward) và trạng thái đánh dấu ô này có phải là trạng thái kết thúc hay không)
- `vizTiles::Matrix{Float64}`: Danh sách các ô trong bản đồ đã được chuẩn hoá tọa độ để dễ dàng trực quan hoá hơn (do đặc tính của hệ tọa độ có các ô ở hàng chẵn nằm chếch sang bên phải 0.5 đơn vị so với các ô ở hàng lẻ)
- `currentState::Vector{Int64}`: vị trí ô hiện tại

Phương thức (do Julia không hỗ trợ cài đặt phương thức bên trong struct nên các phương thức này được cài đặt bên ngoài và truyền vào đối tượng HexEnv làm tham số):

- `reset(env::HexEnv)`: Phương thức này sẽ lấy ngẫu nhiên một ô không phải trạng thái kết thúc trong danh sách tiles và gán nó làm currentState.
- Các phương thức `getNormalStates()`, `getNegRewardTerminals()`, `getPosRewardTerminals()` đều có 2 tham số `env::HexEnv` và `path::Any`. Các phương thức này lần lượt lấy ra các ô không phải

trạng thái kết thúc, trạng thái kết thúc có trọng số âm, trạng thái kết thúc có trọng số không âm từ danh sách vizTiles để trực quan hoá thành hình ảnh.

- Có 3 phương thức visualize() được overload:

- visualize(tile::Vector{Int64}; show = true):

Phương thức này để tô màu một ô trên bản đồ thành màu vàng, sử dụng để trực quan hoá quá trình huấn luyện

- visualize(env::HexEnv, startState::Vector{Int64}, qTable::Array{Float64, 3}; filename, show::Bool = true, prob::Bool = true)

Phương thức trực quan hoá đường đi từ startState đến trạng thái kết thúc có reward cao nhất. Tham số filename được truyền vào khi muốn lưu đường đi thành file hình ảnh, show là tham số Bool, khi có giá trị true thì bản đồ sẽ được in ra màn hình; prob là tham số Bool cho biết có sử dụng stochastic interaction cho các hành động chuyển đổi trạng thái hay không.

- visualize(env::HexEnv, path = nothing, directions = nothing, directionAlpha = nothing; show::Bool = true, lowDpi::Bool = false, optimalPolicy::Bool = false, filename = nothing)

Phương thức trực quan hoá đảm nhiệm chính việc render các đối tượng ra figure, phương thức này thường được gọi bởi các phương thức visualize khác. Khi được gọi trực tiếp và chỉ truyền vào tham số env nó sẽ vẽ ra một bản đồ cơ bản (bao gồm các ô nhưng chưa biểu diễn các tương tác trên đó). Khi được gọi bởi phương thức visualizeOptimalPolicy() nó sẽ vẽ ra bản đồ với các ô có các sắc độ đậm nhạt khác nhau để biểu diễn reward khi đi theo hướng tối ưu. Tham số lowDpi được sử dụng khi cần render ra màn hình trong quá trình huấn luyện, nếu bật tham số này thành true, hình ảnh sẽ được render với DPI thấp hơn, tăng hiệu năng trong lúc visualize.

- visualizeOptimalPolicy(env::HexEnv, qTable::Array{Float64, 3}; show = true, filename = nothing): Phương thức này sẽ tính toán sắc độ và các biểu tượng mũi tên biểu thị cho từng hướng đi tại mỗi ô, sau đó truyền vào hàm visualize với path là danh sách tất cả các ô không phải trạng thái kết thúc, directions, directionAlpha là hướng đi và sắc độ của các ô tương ứng
- step(env::HexEnv, action::Int64): Phương thức này được gọi để thực hiện chuyển đổi từ state hiện tại sang state kế tiếp với action được truyền vào. Trả về trạng thái tiếp theo, phần thưởng và một biến thông báo trạng thái hiện tại có phải trạng thái kết thúc hay không
- getNextAction(currentState::Vector{Int64}, epsilon, qTable; prob::Bool = true): lấy hành động tiếp theo bằng phương pháp Epsilon-Greedy

QLearning (cấu trúc dữ liệu để chứa các thông tin liên quan để huấn luyện và dự đoán bằng thuật toán Q-Learning)

Thuộc tính:

- qTable::Array{Float64, 3}: Thuật toán Q-Learning sử dụng value function là một bảng tra. Bảng tra này có kích thước là NxMxA với N là số cột, M là số dòng của bản đồ và A là số actions mà agent có thể thực hiện trên môi trường đó. Ban đầu bảng tra này sẽ được khởi tạo với giá trị random với phân phối đều trên khoảng từ -1 đến 0. Việc chọn giá trị khởi tạo random này có ưu thế hơn so với chọn giá trị 0 với những episode đầu tiên, nếu thuật toán Epsilon-Greedy đưa ra quyết định chọn action theo max action của bảng tra, khi đó nếu toàn bộ giá trị khởi tạo là 0 thì max action được chọn sẽ luôn là action có số thứ tự nhỏ nhất. Để đảm bảo tính ngẫu nhiên, khai phá các đường đi mới trong những episode đầu, ta cần khởi tạo bảng với các giá trị ngẫu nhiên.

- learningRate::Float64: tốc độ học trong quá trình huấn luyện
- discountFactor::Float64: tác động của discount factor đến quá trình huấn luyện sẽ làm giảm giá trị Q-table với những trạng thái kết thúc ở quá xa một trạng thái đang xét
- epsilon::Float64: thông số này để thuật toán Epsilon-Greedy quyết định xem nên khai phá đường mới (exploration) hay tiếp tục huấn luyện dựa theo những gì đã biết (exploitation)

Phương thức:

- train(agent::QLearning, episodes::Int64; visualizeEps = nothing, gifFile = nothing, prob::Bool = true)

Phương thức huấn luyện agent với thuật toán Q-Learning, có 2 tham số bắt buộc là agent và episodes là số lượng vòng cần huấn luyện. Tham số visualizeEps có thể truyền vào một số nguyên, sau mỗi số nguyên lần vòng đó sẽ trực quan hoá kết quả huấn luyện tại episode hiện tại. Nếu truyền thêm tham số gifFile sẽ trực quan hoá và lưu quá trình huấn luyện thành file GIF. Tham số prob chỉ định cách hành xử ở môi trường hoàn hảo (luôn chuyển đổi thành công từ trạng thái s sang trạng thái s' hoặc ở môi trường có xác suất chuyển đổi. Xác suất chuyển đổi mặc định là 70% chuyển đổi đúng ý định, và 30% chia đều cho 2 trạng thái kể với trạng thái mong muốn. Mặc định tham số prob là true

Sarsa (cấu trúc dữ liệu để chứa các thông tin liên quan để huấn luyện và dự đoán bằng thuật toán SARSA)

Thuộc tính:

Các thuộc tính và ý nghĩa của thuộc tính tương tự so với cấu trúc dữ liệu QLearning

Phương thức:

Phương thức train() tương tự như cấu trúc dữ liệu QLearning. Khác biệt ở quá trình huấn luyện thì phương thức train cho Sarsa sẽ sử dụng chung một chiến lược Epsilon-Greedy cho cả việc chọn hành động cũng như cập nhật Q-table

Phương pháp giải quyết

Bài toán Hex World là một bài toán MDP đơn giản, do không gian bài toán đặt ra là đi đến trạng thái đích với tổng phần thưởng (reward) là tối ưu nhất, đồng thời cân bằng giữa lợi ích tức thời và lợi ích lâu dài. Số lượng trạng thái với mỗi bản đồ là có hạn và thay đổi tùy theo bản đồ. Với mỗi trạng thái chỉ phụ thuộc vào trạng thái trước đó và xác suất chuyển đổi giữa chúng. Q-Learning và SARSA là 2 thuật toán dễ hiểu, dễ cài đặt, cả 2 thuật toán này đều là thuật toán model-free. Thuật toán model-free chỉ cần biết phần thưởng (reward) và trạng thái tiếp theo (next state) khi đã thực hiện nó mà không cần một hàm dự đoán reward và next state [3] vì vậy nên các thuật toán model-free tốt hơn model-based ở tốc độ tính toán và bộ nhớ chiếm dụng. Vậy nên 2 thuật toán này đã được chọn để khảo sát phương pháp giải quyết bài toán

Thuật toán Q-Learning

Thuật toán Q-Learning là một thuật toán Off-Policy, nó "học" chiến lược tối ưu (optimal policy) độc lập so với những hành động của agent, hay nói cách khác là nó có chiến lược hành động (behavior policy) khác với chiến lược ước lượng (estimate policy).

Công thức cập nhật giá trị Q-table với $Q(s,a)$ là giá trị khi thực hiện hành động a tại trạng thái s với tốc độ học (learning rate) α và chiết khấu (discount factor) γ :

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(R + \gamma \max_{a'} Q(s', a'))$$

Trong công thức này, chiến lược ước lượng được sử dụng là hàm $\max()$ lấy giá trị của action tốt nhất với vị trí s' , khác với chiến lược hành động (cụ thể trong đồ án này sử dụng chiến lược hành động là Epsilon-Greedy)

Thuật toán SARSA

Thuật toán SARSA là một thuật toán On-Policy, nó "học" chiến lược cận tối ưu (near optimal policy) phụ thuộc vào hành động của agent, nó có chiến lược hành động (behavior policy) giống với chiến lược ước lượng (estimate policy).

Công thức cập nhật giá trị Q-table với $Q(s,a)$ là giá trị khi thực hiện hành động a tại trạng thái s với tốc độ học (learning rate) α và chiết khấu (discount factor) γ , trong đó a' là hành động tiếp theo được thực hiện tại trạng thái s' với cùng chiến lược hành động với khi chuyển đổi trạng thái từ s sang s' bằng hành động a , cụ thể trong đồ án này là sử dụng chiến lược Epsilon-Greedy để chọn hành động chuyển đổi trạng thái:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(R + \gamma Q(s', a'))$$

Giải quyết tình trạng mắc kẹt ở cực trị địa phương (local optima)

Nếu tiếp cận bài toán với chiến lược hành động phụ thuộc hoàn toàn vào Q-table, ban đầu khi agent chưa được huấn luyện sẽ không có cơ sở để dựa vào Q-table và đưa ra các hướng di chuyển và chính vì thế rất dễ bị mắc kẹt tại các cực trị địa phương. Do đó, sử dụng một chiến lược khác để chọn hành động cho agent là cần thiết để có thể khai phá những hướng đi khác có thể đem lại nhiều lợi ích hơn. Cụ thể trong đồ án này sẽ tiếp cận bài toán với phương pháp Epsilon-Greedy. Với phương pháp này, ví dụ ta chọn $\epsilon = 0.4$, khi đó sẽ có 40% các hành động được chọn bằng cách chọn ngẫu nhiên, 60% còn lại dựa vào hành động tốt nhất hiện tại dựa vào Q-table. Tuy nhiên, việc khai phá hướng đi mới chỉ thực sự hiệu quả trong khoảng thời gian mới bắt đầu huấn luyện do lúc đó Q-table chưa đủ độ tin cậy để có thể dựa vào.

Chính vì thế, để tối ưu cho việc chọn khai phá (exploration) hay khai thác (exploitation), trong bài toán này có thể sử dụng phương pháp Decaying Epsilon-Greedy. Epsilon khi bắt đầu sẽ được khởi tạo một giá trị cao (gần với 1), ưu tiên khai phá những hướng đi mới, và sẽ giảm dần về 0 theo hàm tuyến tính đến khi đã huấn luyện được $\frac{1}{2}$ tổng số lượng episodes. Khi đó, ta có thể dựa hoàn toàn vào Q-table do các đường đã được khai phá đủ nhiều, lúc này ta chỉ khai thác những gì đã biết để tìm kiếm optimal policy.

Code

Hàm chọn hành động tiếp theo

Hàm chọn hành động sẽ dùng phương pháp Epsilon-Greedy (EG) để chọn ra hành động muốn đi tại trạng thái hiện tại. Sau đó, tùy vào cài đặt bật / tắt hiệu ứng ngẫu nhiên (stochastic effects) bằng tham số Bool, nếu hiệu ứng ngẫu nhiên tắt, nó sẽ đi đúng hành động tối ưu bởi EG; ngược lại, nếu bật

hiệu ứng ngẫu nhiên, nó sẽ chọn một số ngẫu nhiên trong khoảng từ 0 đến 1 theo phân phối đều (uniform distribution), action được chọn sẽ phụ thuộc vào số ngẫu nhiên này:



Nếu số ngẫu nhiên nằm trong khoảng A, action được chọn sẽ nằm kế bên trái của action tối ưu, nếu số ngẫu nhiên nằm trong khoảng B, action được chọn nằm kế bên phải của action tối ưu, còn lại khoảng C sẽ đi đúng action tối ưu

```
# get next action with stochastic effect
function getNextActionWithProb(action::Int64, prob::Float64)

    neighborsProb = (1 - prob)
    randNum = rand(Uniform(0., 1.))

    # |=====|=====|=====|
    #   A       B               C

    # if the random number in A
    if (randNum < neighborsProb / 2)
        action = (action - 1) % HEX_ACTIONS_NUM
        if (action == 0)
            action = HEX_ACTIONS_NUM
        end
        return action;
    # if the random number in B
    elseif (randNum < neighborsProb)
        action = (action + 1) % HEX_ACTIONS_NUM
        if (action == 0)
            action = HEX_ACTIONS_NUM
        end
        return action;
    # if the random number in C
    else
        return action;
    end

end;

# get next action
function getNextAction(currentState::Vector{Int64}, epsilon, qTable;
    prob::Bool=true)
    if rand(Uniform(0., 1.)) > epsilon
        if prob
```



```

        return getNextActionWithProb(findmax(qTable[currentState[1],
currentState[2], :])[2], MOVE_PROB)
    else
        return findmax(qTable[currentState[1], currentState[2], :])[2]
    end
else
    if prob
        return getNextActionWithProb(rand(1:HEX_ACTIONS_NUM), MOVE_PROB)
    else
        return rand(1:HEX_ACTIONS_NUM)
    end
end
end;

```

Hàm chuyển đổi giữa 2 trạng thái

```
ACTIONS = [-1 0; 0 1; 1 1; 1 0; 1 -1; 0 -1]
```

Hàm chuyển đổi trạng thái sẽ chuyển đổi từ trạng thái hiện tại sang trạng thái mới theo hành động đã được chọn từ bước trước. Trạng thái tiếp theo được xác định bằng cách cộng trạng thái hiện tại với các chỉ số trong tuple ACTIONS bên trên, các chỉ số này tương ứng với 6 hướng đã trình bày ở phần cấu trúc dữ liệu HexEnv. Do hệ toạ độ của hex world như trình bày ở phần phát biểu bài toán sẽ được đánh chỉ số bắt đầu từ (1, 1), Julia cũng là ngôn ngữ đánh chỉ số từ 1 (1-based indexing). Vậy nên các ô nằm trên hàng có chỉ số chẵn sẽ nằm lệch sang bên phải 0.5 đơn vị, cũng vì đó mà chỉ số của các ô này sẽ bị lệch phải 1 đơn vị sau khi cộng ACTIONS tương ứng, ta cần dời chỉ số các ô này sang trái 1 đơn vị. Với mỗi bước đi sẽ mất một chi phí là STEP_COST (mặc định là -1 do nếu chi phí di chuyển là 0 thì có thể agent sẽ bị mắc kẹt giữa 2 trạng thái và không thể thoát ra được, xảy ra khi các terminal state âm), với mỗi lần chuyển đổi ra ngoài biên sẽ bị mất một chi phí là OUT_OF_BOUNDS_PENALTY (mặc định là -1)

```
# transfer from one state to another
```

```
function step(env::HexEnv, action::Int64)
    nextState = env.currentState + ACTIONS[action, :]
    if (action != 1 && action != 4)
        nextState[1] -= abs((nextState[2] + 1) % 2)
    end
end
```

```
# check if the next state is in the list of tiles (available in the
environment)
```

```
position = matchRow(env.tiles[:, 1:2], nextState)
```

```
if (position != 0)
```

```
    reward = STEP_COST
```

```
    reward += env.tiles[position, 3]
```

```

    if (env.tiles[position, 4] == 1)
        return env, nextState, reward, true
    end

    return env, nextState, reward, false

else
    return env, env.currentState, OUT_OF_BOUNDS_PENALTY, false
end
end;

```

Hàm huấn luyện

Hàm huấn luyện sẽ chạy trên số lượng episodes được truyền vào, với mỗi episode hàm huấn luyện sẽ chọn một trạng thái ngẫu nhiên làm trạng thái khởi đầu và sẽ di chuyển từ trạng thái đó đến terminal state dựa theo Q-table hiện tại và cập nhật Q-table bằng công thức đã trình bày ở phần phương pháp giải quyết. Ngoài ra hàm này còn chứa các thành phần để trực quan hoá quá trình huấn luyện cũng như trả về các thông số để đánh giá quá trình huấn luyện như reward trung bình trên episode đã huấn luyện. Dưới đây là code của thuật toán Q-Learning, với thuật toán SARSA cũng tương tự, chỉ thay đổi phương trình cập nhật Q-table.

```

function train(agent::QLearning, episodes::Int64; visualizeEps = nothing, gifFile
= nothing, prob::Bool = true, samplingRate::Float64 = 0.001)

    print("\nStep cost: $(Environments.STEP_COST) \n")
    print("Out of bounds penalty: $(Environments.OUT_OF_BOUNDS_PENALTY) \n")

    EPSILON_DECAY_START_EP = 1
    EPSILON_DECAY_END_EP = episodes // 2;
    EPSILON_DECAY = agent.epsilon / EPSILON_DECAY_END_EP;
    SAMPLING_EP = trunc(Int, 1 / samplingRate);

    # accumulated reward
    y = [];
    accumulatedReward = 0;

    anim = Environments.Plots.Animation()

    if visualizeEps === nothing
        visualizeEps = episodes + 1
    end

    for episode in 1:episodes

```

```

if (episode % visualizeEps == 0)
    print("Visualizing episode $episode/$episodes\r");
    flush(stdout);
    if (gifFile != nothing)
        visualize(agent.env, show = false);
        Environments.Plots.frame(anim);
    else
        visualize(agent.env, lowDpi = true);
        sleep(0.5);
    end;
    visualizeEpsLabel(episode);
end;

terminal = false
agent = @set agent.env = Environments.reset(agent.env)

while (!terminal)

    nextAction = getNextAction(agent.env.currentState, agent.epsilon,
agent.qTable, prob = prob)

    env, nextState, reward, terminal = Environments.step(agent.env,
nextAction)
    accumulatedReward += reward;

    currentQValue = agent.qTable[env.currentState[1],
env.currentState[2], nextAction]

    newQValue = (1 - agent.learningRate) * currentQValue +
agent.learningRate *
        (reward + agent.discountFactor *
findmax(agent.qTable[nextState[1], nextState[2], :])[1])

    agent.qTable[env.currentState[1], env.currentState[2], nextAction] =
newQValue

    if (episode % visualizeEps == 0)
        if (gifFile != nothing)
            visualize(agent.env.currentState, show = false);
            Environments.Plots.frame(anim);
        else
            visualize(agent.env.currentState);
            sleep(0.1);
        end
    end
end

```

```

env = @set env.currentState = nextState

agent = @set agent.env = env

end

if (episode <= EPSILON_DECAY_END_EP && episode >= EPSILON_DECAY_START_EP)
    agent = @set agent.epsilon = agent.epsilon - EPSILON_DECAY
end

if (episode % visualizeEps == 0)
    if (gifFile === nothing)
        sleep(1);
    end;
end;

if (visualizeEps > episodes && episode % SAMPLING_EP == 0)
    push!(y, accumulatedReward / episode);
    print("Trained $episode/$episodes episodes\r");
    flush(stdout);
end

end;

print("\n");

if (gifFile !== nothing)
    print("Writing GIF\n");
    Environments.Plots.gif(anim, gifFile, fps = 10);
else
    if (visualizeEps <= episodes)
        print("Press any key to continue...\n");
        readline();
    end
end;

return y;
end;

```

Hàm lấy mẫu đánh giá

Hàm này dùng để chạy thử với các trạng thái bắt đầu ngẫu nhiên theo phân phối đều và trả về số lượng mẫu đánh giá tùy thuộc vào giá trị đầu vào của `samplingRate` (mặc định là 0.001 – 1 lần lấy mẫu mỗi 1000 lần chạy thử)

```

function evaluate(agent, times = 100000; samplingRate = 0.001)
    result = [];
    SAMPLING_TIME = trunc(Int, 1 / samplingRate);
    accumulatedReward = 0;
    print("\n");
    for time in 1:times
        agent = @set agent.env = Environments.reset(agent.env);
        accumulatedReward += Environments.visualize(agent.env,
agent.env.currentState, agent.qTable, show = false, render = false);
        if (time % SAMPLING_TIME == 0)
            push!(result, accumulatedReward / time);
            print("Evaluated $time/$times times\r");
            flush(stdout);
        end
    end
    return result;
end;

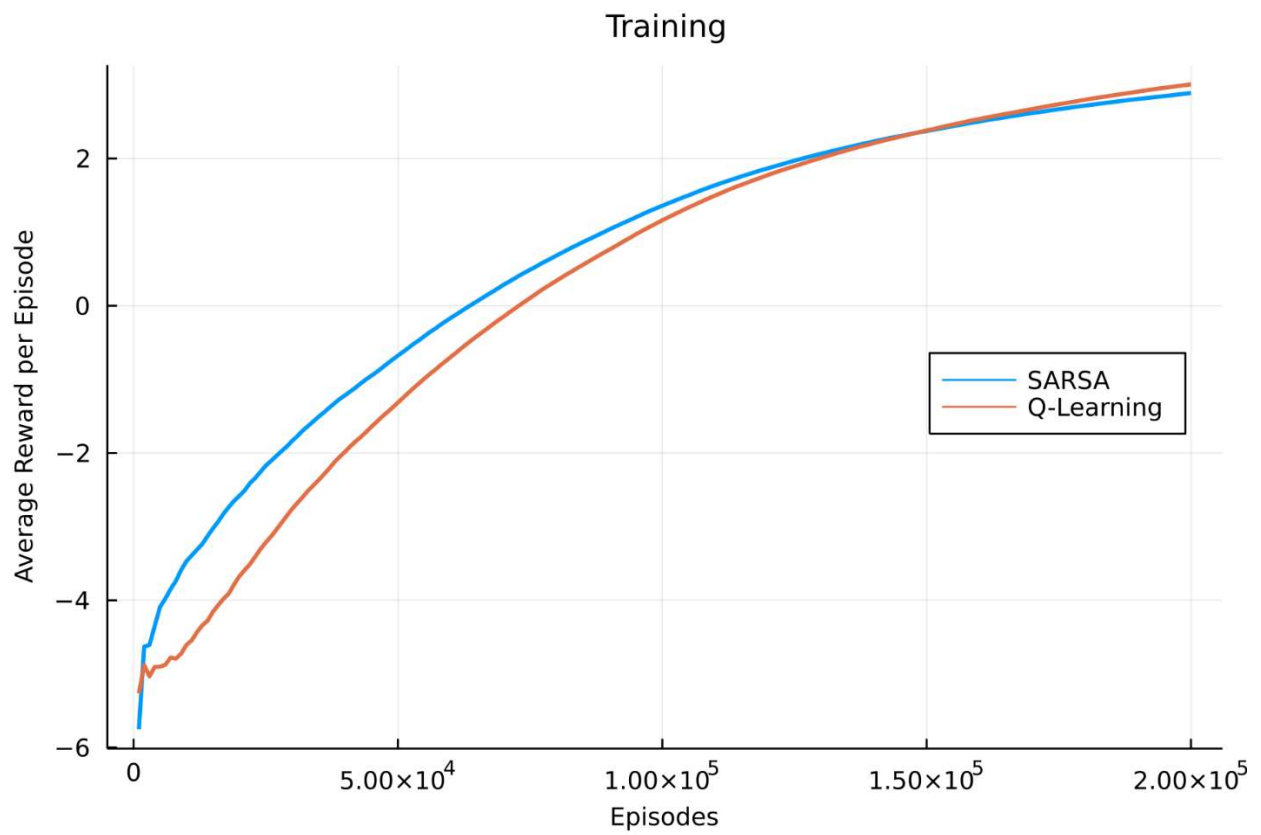
```

Phân tích

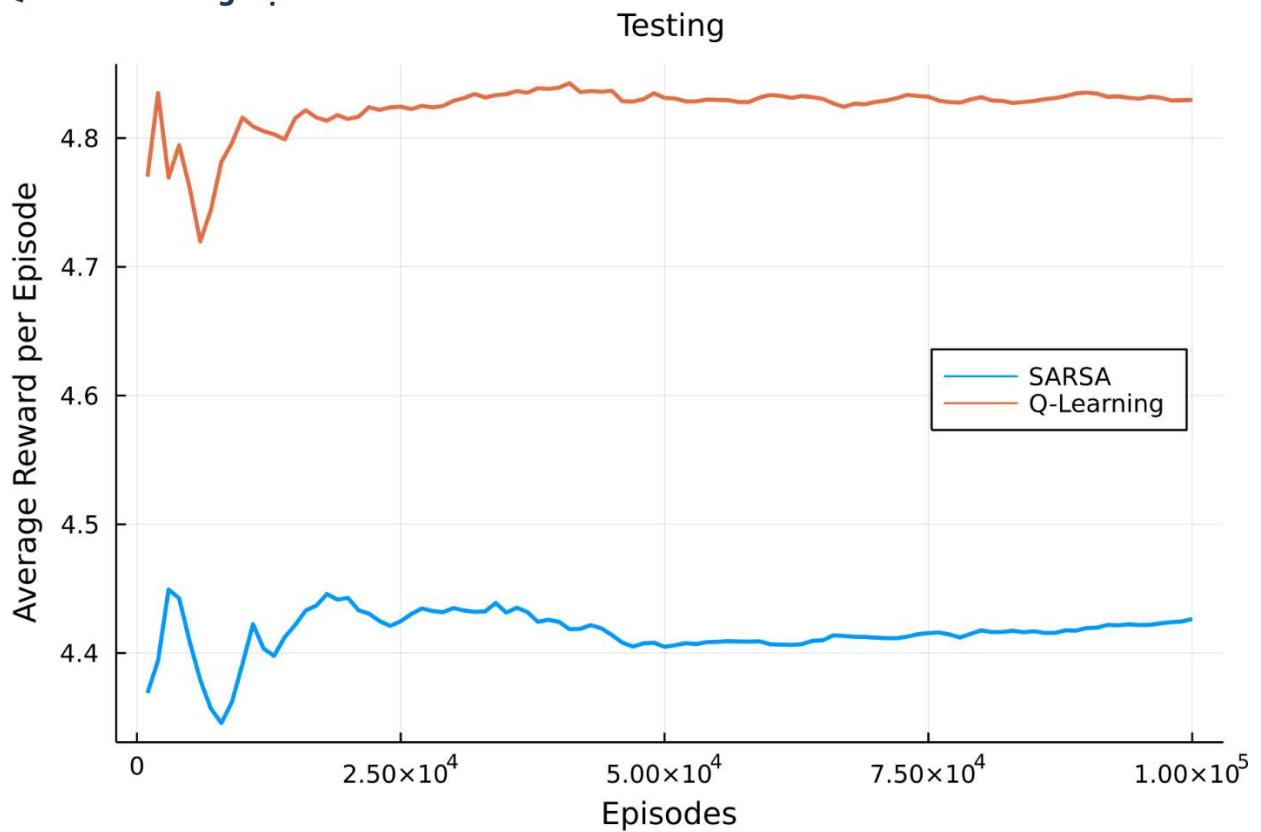
Các biểu đồ và phân tích dưới đây dựa trên kết quả của các agent trên bản đồ map2.csv đính kèm trong project. Cả 2 agent đều được huấn luyện 200,000 episodes với learning rate = 0.1, discount factor = 0.9, epsilon = 0.8 và có áp dụng Linear Decaying Epsilon trong khoảng từ episode đầu tiên đến một nửa số lượng tổng epsilon, phần còn lại sẽ được exploit với epsilon là 0.0

Quá trình huấn luyện

Biểu đồ bên dưới thể hiện phần thưởng (reward) trung bình trên số episode đã huấn luyện của 2 agent được huấn luyện bằng thuật toán Q-Learning và SARSA. Ở những episode đầu tiên, thuật toán Q-Learning tỏ ra kém hơn so với SARSA do SARSA thực hiện chiến thuật (policy) thận trọng hơn, hướng đến đường đi an toàn (nhưng có thể không tối ưu) so với Q-Learning luôn cố tìm chiến thuật tối ưu nhất. Do điểm phạt khi thực hiện chuyển đổi sang một trạng thái không tồn tại (ra ngoài biên) chỉ là 1 đơn vị, nên chênh lệch giữa Q-Learning và SARSA không nhiều. Nếu điểm phạt này tăng lên gấp nhiều lần (ví dụ như 100 đơn vị mỗi lần) thì đường biểu diễn của Q-Learning sẽ thấp hơn nhiều so với SARSA. Ở phần sau của đồ thị, Q-Learning dần chiếm lại ưu thế và vượt qua SARSA một chút, do Q-Learning tìm chiến thuật tối ưu, nên những phần thưởng về sau sẽ được cộng dồn nhiều hơn so với SARSA. Vậy nên thuật toán SARSA sẽ phù hợp hơn với các bài toán cần tránh phạm lỗi (do điểm phạt khi phạm lỗi lớn) nhưng không tốt bằng Q-Learning về tối ưu phần thưởng nhận được



Quá trình thử nghiệm



Quá trình thử nghiệm được tiến hành trên 100000 lần chơi khác nhau, với mỗi lần chơi vị trí bắt đầu sẽ được chọn ngẫu nhiên theo phân phối đều, đồ thị bên trên thể hiện reward trung bình sau các vòng chơi đã được thử nghiệm. Ở những vòng chơi đầu, đồ thị chưa ổn định do tính ngẫu nhiên của vị trí bắt đầu, sau khi đã thử nghiệm được nhiều vòng chơi thì đồ thị dần ổn định và Q-Learning cho thấy rõ lợi thế của một thuật toán tìm kiếm chiến lược tối ưu (optimal policy) thay vì chiến lược gần tối ưu (near optimal policy) như SARSA, agent được huấn luyện với Q-Learning cho kết quả tốt hơn so với SARSA. Vậy nên về mặt tối ưu reward thì Q-Learning là thuật toán tốt hơn

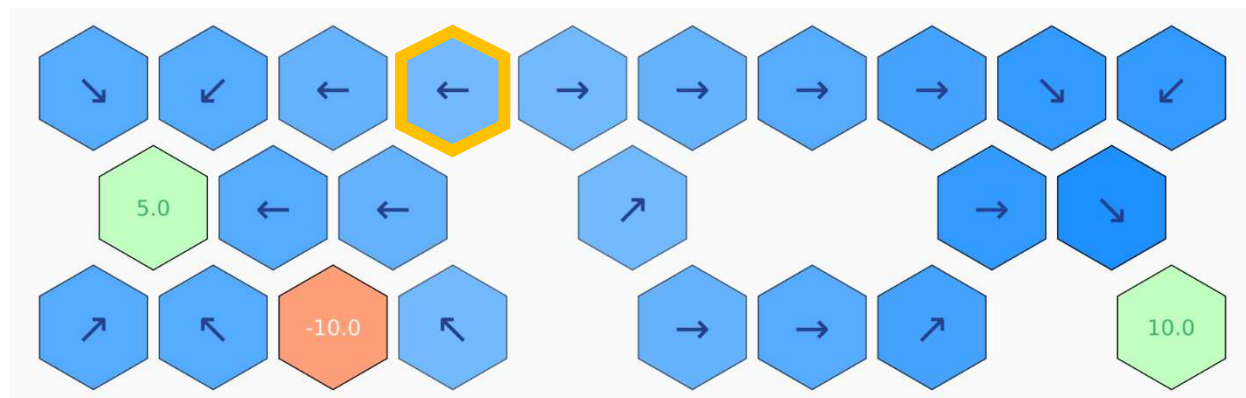
Tóm tắt kết quả

Cả 2 agent được huấn luyện với Q-Learning và SARSA đều được chạy thử trên 3 bản đồ khác nhau, trong đó có một bản đồ là ví dụ được cung cấp sẵn trong tài liệu [1], một bản đồ to hơn thể hiện trường hợp tổng quát và một bản đồ thể hiện trường hợp đặc thù (các terminal state đều có reward âm). Tất cả các bản đồ đều được huấn luyện với learning rate = 0.1, discount factor = 0.9 và epsilon ban đầu là 0.8, có sử dụng Linear Decaying cho epsilon với 1 nửa trong tổng số episodes

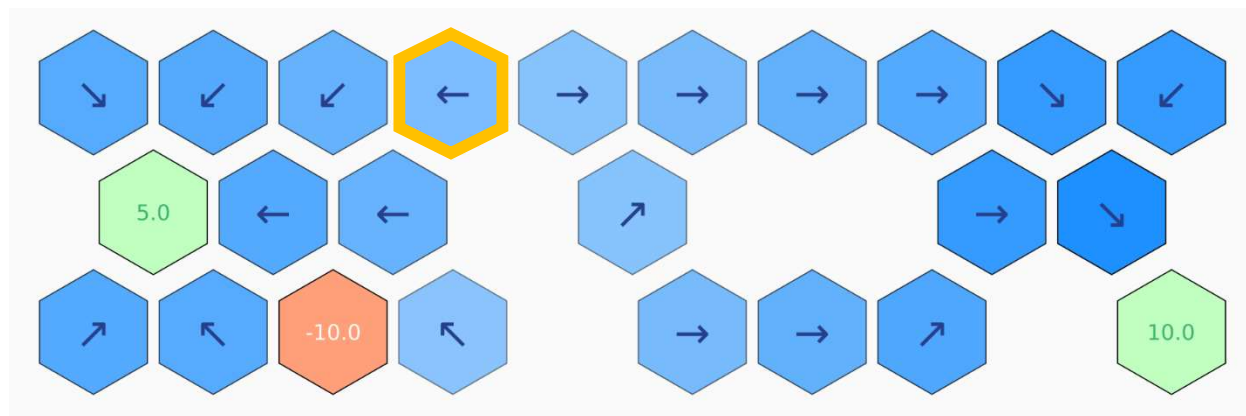
Bản đồ 1

Dưới đây là hình ảnh minh họa optimal policy sau khi huấn luyện 100,000 episodes của 2 agent:

Thuật toán Q-Learning



Thuật toán SARSA

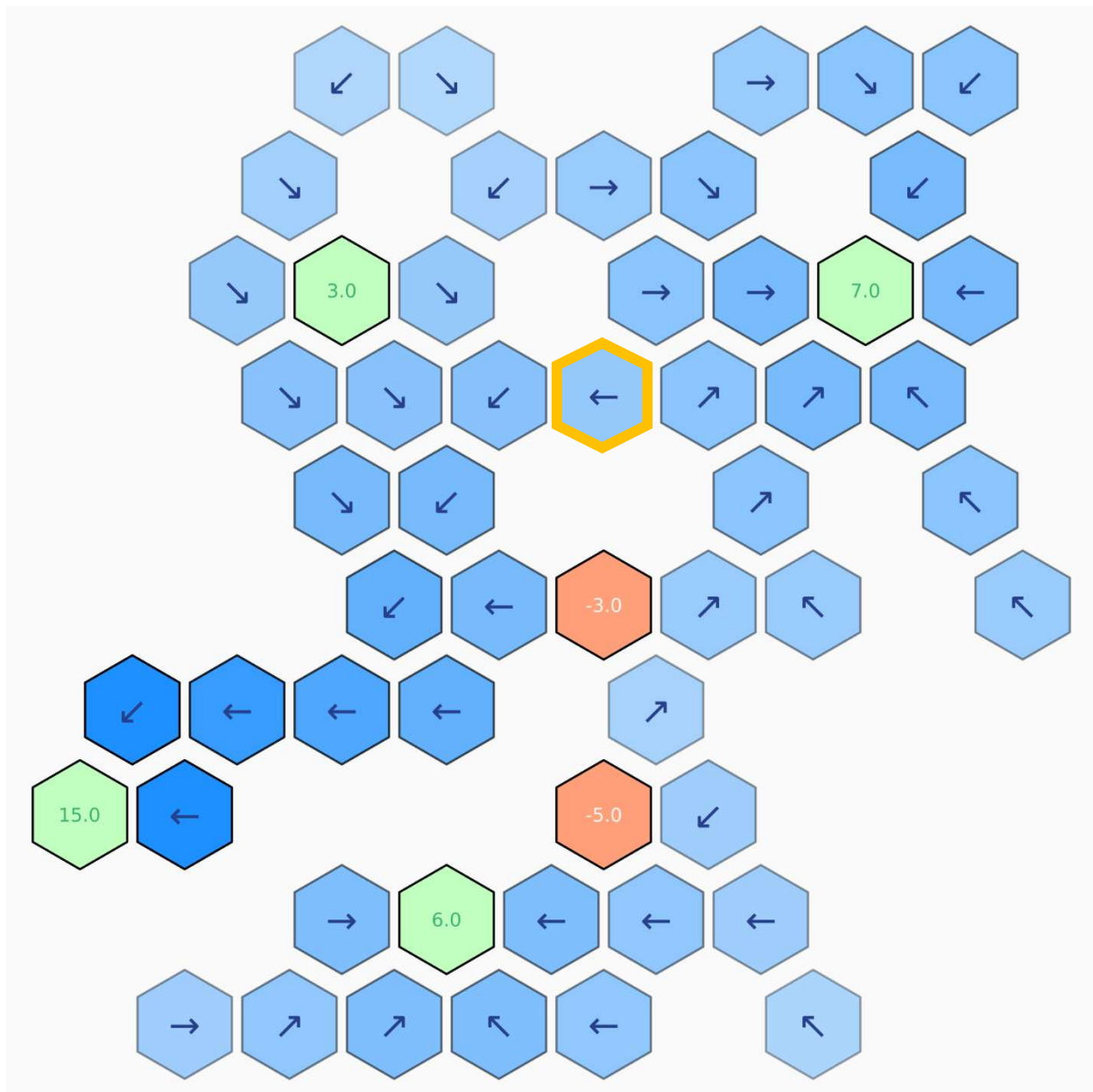


Ở bản đồ này, cả 2 thuật toán đều cho kết quả gần giống nhau (có một vài trạng thái lựa chọn hành động khác nhưng mang lại reward như nhau) và là kết quả tốt nhất có thể đạt được. Điểm cần lưu ý là ô trạng thái ở vị trí (4, 3) (vị trí thứ 4 của hàng thứ 3 – ô có viền màu vàng), tại trạng thái này khi di chuyển sang terminal state có reward 10 sẽ cho tổng reward cao hơn, tuy nhiên do trạng thái này cách quá xa nên nó ưu tiên chọn terminal state có reward 5 hơn do ảnh hưởng của discount factor. Trong trường hợp này cả 2 agent đều chọn lợi ích tức thời thay vì lợi ích lâu dài.

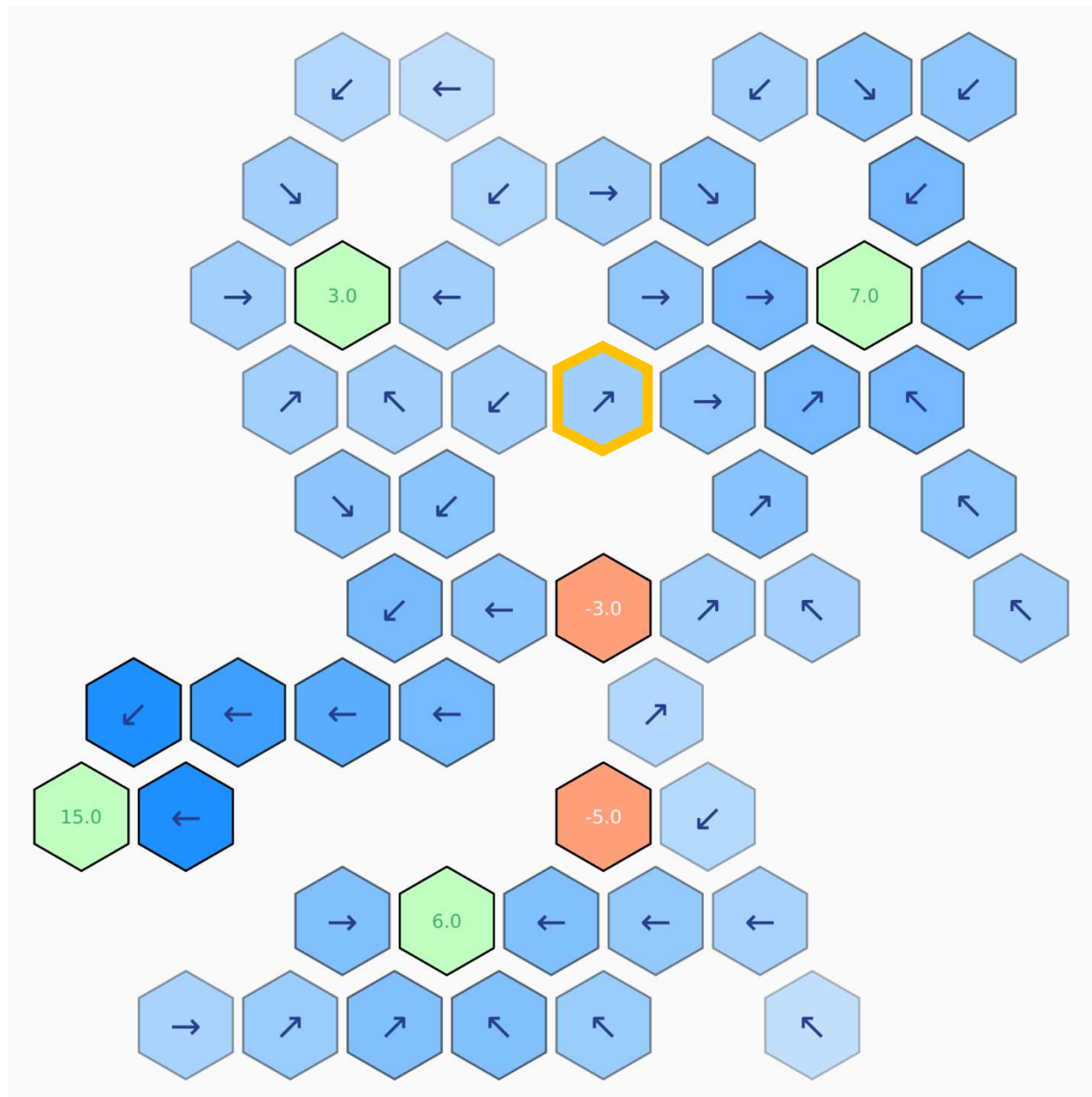
Bản đồ 2

Dưới đây là hình ảnh minh họa optimal policy sau khi huấn luyện 300,000 episodes của 2 agent:

Thuật toán Q-Learning



Thuật toán SARSA



Ở các ô trạng thái xung quanh terminal state có reward bằng 3, các trạng thái này có thể di chuyển xuống terminal state có reward bằng 15 để đạt reward cao nhất (giống như Q-Learning đã làm) nhưng SARSA lại không chọn được chiến thuật này mà lại ưu tiên trạng thái kết thúc gần hơn (và cũng vì thế mà an toàn hơn). Trường hợp tương tự xảy ra với ô được đánh dấu bằng viền màu vàng. Reward khi đi qua terminal state 15 cao hơn nhưng SARSA vẫn chọn terminal state an toàn hơn. Các trạng thái khác Q-Learning và SARSA cho kết quả tương tự nhau

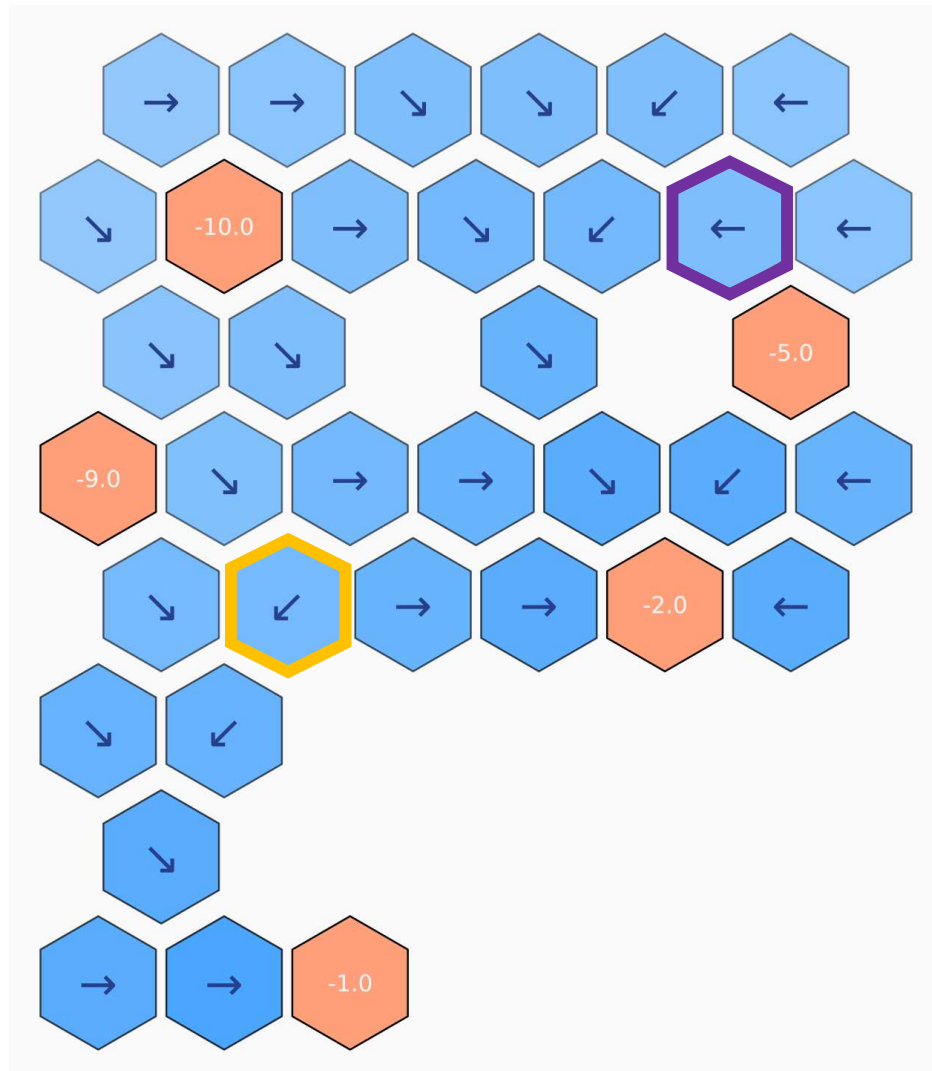
Bản đồ 3

Dưới đây là hình ảnh minh họa optimal policy sau khi huấn luyện 200,000 episodes của 2 agent:

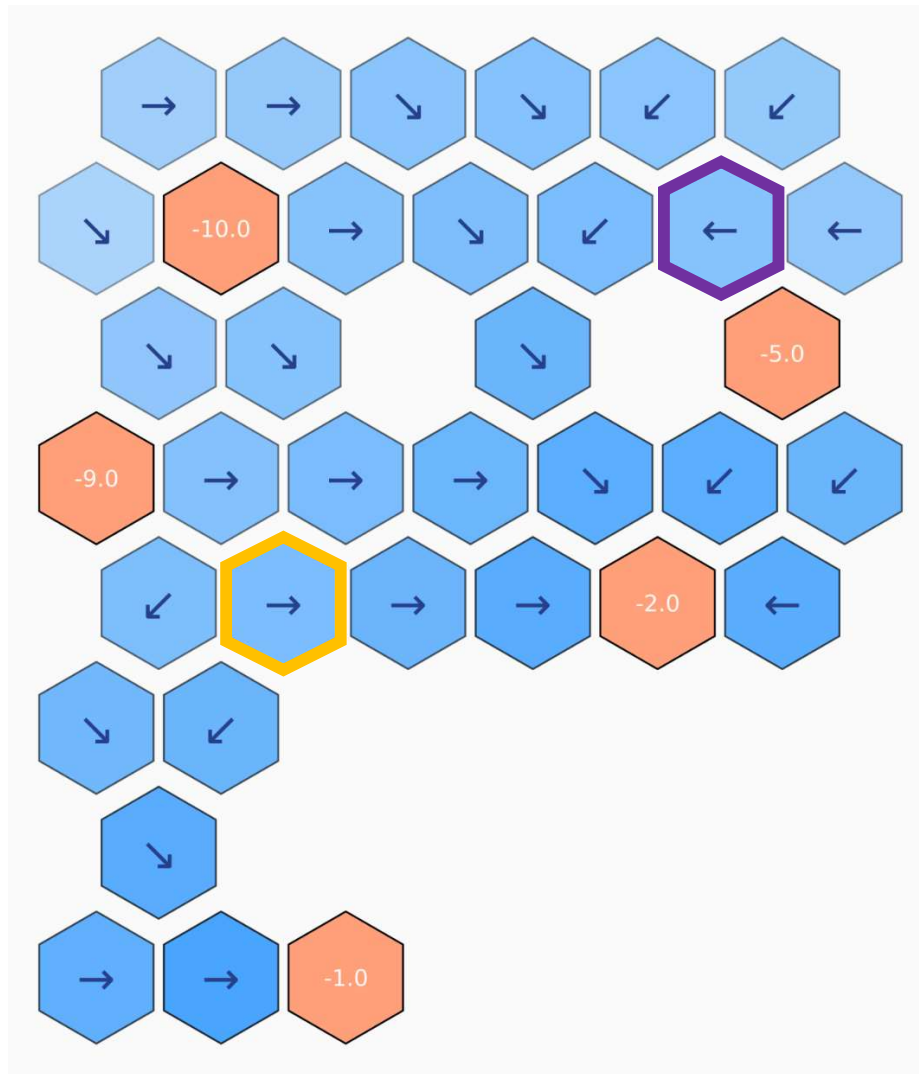
Ở 2 hình bên dưới, với trường hợp đặc biệt là terminal states toàn bộ đều là âm, nếu có trường hợp tại một trạng thái bắt đầu có 2 đường đi với số bước khác nhau nhưng có cùng reward (trạng thái có viền màu vàng), Q-Learning sẽ ưu tiên đi đường có số bước nhiều hơn (xa hơn). Còn đối với SARSA, trong bản đồ này các quyết định không thể xác định được chiến lược tổng quát do không có tính nhất quán, ở trạng thái viền màu vàng, SARSA quyết định đi đến terminal state có reward là -2 (gần hơn), nhưng với ô trạng thái có viền màu tím, SARSA lại quyết định đi đến terminal state có reward là -2 (xa hơn) thay vì đi đến terminal state có reward là -5 nằm kế bên (gần hơn, cũng có tổng reward đã trừ step cost là -6). Dù đã thử huấn luyện cả 2 agent với 1,000,000-3,000,000 episodes nhưng kết quả thu được không khả quan hơn so với khi huấn luyện 200,000 episodes

Cả 2 agent đều có những trường hợp ưu tiên terminal state ở xa hơn dù có terminal state gần hơn có cùng reward do ảnh hưởng của discount factor, càng xa thì reward càng tiến về gần 0, khiến cho những terminal state ở xa có Q-value lớn hơn (do là số âm gần 0 hơn)

Thuật toán Q-Learning

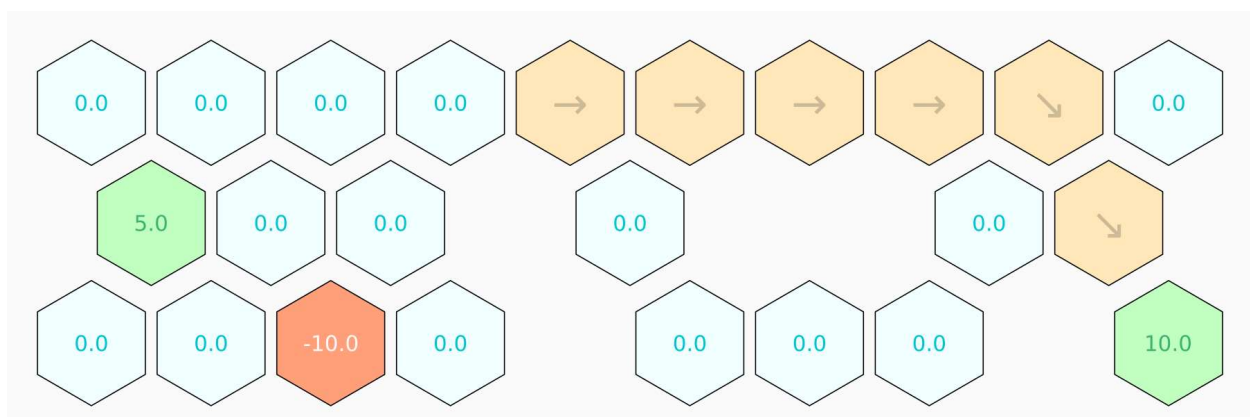


Thuật toán SARSA



Minh hoạ đường đi từ một trạng thái đến trạng thái kết thúc

Đây là hình ảnh minh hoạ một đường đi từ trạng thái xuất phát (5,3) đến trạng thái kết thúc, trong nhiều lần chạy thử có thể ra nhiều kết quả khác nhau do tính ngẫu nhiên khi chuyển đổi giữa 2 trạng thái



Tài liệu tham khảo

- [1] M. J. Kochenderfer, T. A. Wheeler and K. H. Wray, Algorithms for Decision Making.
- [2] "Hexagonal Grids," [Online]. Available: <https://www.redblobgames.com/grids/hexagons>. [Accessed 10 12 2021].
- [3] OpenAI, "Part 2: Kinds of RL Algorithms," [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html. [Accessed 10 12 2021].
- [4] A. Gupta, "SARSA Reinforcement Learning," [Online]. Available: <https://www.geeksforgeeks.org/sarsa-reinforcement-learning/>. [Accessed 10 12 2021].
- [5] C. Shyalika, "A Beginners Guide to Q-Learning," [Online]. Available: <https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c>. [Accessed 10 12 2021].
- [6] L. Panneerselvam, "Q – Learning Algorithm with Step by Step Implementation using Python," [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/04/q-learning-algorithm-with-step-by-step-implementation-using-python/>. [Accessed 10 12 2021].
- [7] V. H. T. Duong, "Intro to reinforcement learning: temporal difference learning, SARSA vs. Q-learning," [Online]. Available: <https://towardsdatascience.com/intro-to-reinforcement-learning-temporal-difference-learning-sarsa-vs-q-learning-8b4184bb4978>. [Accessed 10 12 2021].