

A. ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



Đồ án thực hành:

Thuật toán ra quyết định

Cơ sở trí tuệ nhân tạo – 19TN

TP. Hồ Chí Minh, tháng 12 năm 2021

MỤC LỤC

A. Thông tin nhóm.....	1
B. Kiến thức chung	2
C. 5 bài toán.....	3
I. Hex world	3
1. Phát biểu bài toán.....	3
2. Thách thức.....	4
3. Thực nghiệm.....	4
4. Đánh giá kết quả	15
5. Tài liệu tham khảo.....	15
II. Rock-Paper-Scissors.....	16
1. Phát biểu bài toán.....	16
2. Thách thức.....	17
3. Thực nghiệm cho phương pháp Fictitious Play	17
4. Thực nghiệm cho phương pháp Linear Regression	25
5. Đánh giá kết quả	32
6. Tài liệu tham khảo.....	33
III. Traveler's Dilemma	34
1. Phát biểu bài toán.....	34
2. Thách thức.....	34
3. Thực nghiệm.....	35
4. Đánh giá kết quả	43
5. Tài liệu tham khảo.....	44
IV. Predator-Prey Hex World	46

1. Phát biểu bài toán.....	46
2. Thách thức.....	46
3. Thực nghiệm.....	47
4. Đánh giá kết quả	59
5. Tài liệu tham khảo.....	61
V. Multi-Caregiver Crying Baby.....	62
1. Phát biểu bài toán.....	62
2. Thách thức.....	62
3. Thực nghiệm.....	62
4. Đánh giá kết quả	71
5. Tài liệu tham khảo.....	71
D. Tóm tắt kết quả	72
I. Điểm mạnh.....	72
II. Điểm yếu.....	72

B. Thông tin nhóm

Nhóm gồm 5 thành viên

MSSV	Sinh Viên
19120156	Nguyễn Thị Hiền Vi
19120160	Đàm Thị Xuân Ý
19120173	Đinh Minh Bảo
19120491	Đặng Thái Duy
19120727	Võ Hoàng Vũ

C. Kiến thức chung

Discount factor (hệ số chiết khấu)

- Hệ số chiết khấu dùng để kiểm soát mức độ trong việc đưa ra quyết định, phần thưởng trong hiện tại có giá trị hơn phần thưởng trong tương lai như thế nào, nhân tố quyết định mức độ ảnh hưởng trong tương lai của hành động.
 - Kí hiệu: γ
 - Tập xác định: $\gamma \in [0, 1]$
 - Nếu $\gamma=0$: agent chỉ quan tâm phần thưởng hiện tại
 - Nếu $\gamma>1$: agent nhận càng nhiều thông tin về phần thưởng trong tương lai trong quá trình ra quyết định của nó
- Giá trị của hệ số chiết khấu trong 5 bài toán như sau

Bài toán	γ
Hex World	0.9
Rock-Paper-Scissors	0.9
Traveler's Dilemma	1
Predator-Prey Hex World	0.9
Multi-Caregiver Crying Baby	0.9

D. 5 bài toán

I. Hex world

1. Phát biểu bài toán

- Giới thiệu bài toán Hex World:

- Hex World bao gồm các ô hình lục giác đều, mỗi ô có thể có chung cạnh nhiều nhất với 6 ô khác liền kề, những ô này gắn liền nhau tạo thành một bản đồ.
- Chúng ta cần phải duyệt qua tất cả các ô trong bản đồ này, mỗi ô của bản đồ đại diện cho một trạng thái hay còn gọi là vị trí hiện tại của tác nhân.
- Từ một trạng thái tác nhân có thể di chuyển theo 6 hướng (hướng theo 6 đỉnh của lục giác) để chuyển đổi trạng thái
- Quá trình chuyển đổi trạng thái, tạo ra một hành động là hoàn toàn ngẫu nhiên
- Tại một số ô nhất định sẽ có phần thưởng riêng đặc biệt được quy định bởi 1 giá trị số, đây là tham số tác động chính đến quá trình di chuyển của tác nhân.
- Các thuật ngữ sử dụng thay thế tương ứng:

Agent	Tác nhân
State	Trạng thái
Action	Hành động
Reward	Phần thưởng

- Nội dung bài toán Hex World:

- Với tập dữ liệu đầu vào bao gồm
 - Tập các trạng thái S (state) quy định tất cả các trạng thái tồn tại trong bản đồ Hex World
 - Tập các phần thưởng R (reward) quy định giá trị phần thưởng tương ứng với mỗi trạng thái trong bản đồ Hex World. Mỗi ô chỉ có duy nhất một giá trị phần thưởng, tổng số phần thưởng không lớn hơn tổng số trạng thái trong tập trạng thái S (state)

- Vấn đề cần giải quyết: Quá trình quyết định cách mà tác nhân thực hiện các hành động ở mỗi một trạng thái để nhận được phần thưởng nhiều nhất có thể (bao gồm việc tránh các phần thưởng âm trong bản đồ)

2. Thách thức

- Tìm hiểu và xây dựng được mô hình giải quyết bài toán hợp lý, cho kết quả thực nghiệm đúng đắn.
- Kiến thức về MDP, Q-learning.
- Ngôn ngữ Julia khá mới, sử dụng thông thạo các package cần nhiều thời gian.
- Không có nhiều mô hình để thực nghiệm, chỉ sử dụng mô hình MDP cho bài toán
- Tài liệu về MDP HexWorld không nhiều, chỉ tham khảo trong sách Algorithm for Decision Making
- Tối ưu hóa phương pháp

3. Thực nghiệm

3.1. Mô hình hóa tính toán

a. Markov Decision Process (MDP)

- Giới thiệu: MDP là mô hình xác định cách một tác nhân thực hiện các hành động tuần tự từ các trạng thái trong môi trường của nó, được ảnh hưởng bởi phần thưởng và mang tính ngẫu nhiên trong quá trình chuyển đổi trạng thái.
- Cấu trúc

Biến	Kiểu dữ liệu	Mô tả
S	Vector{Tuple{Int,Int}}	State - Tập các trạng thái mà agent có thể đứng
A	Vector{Int}	Action - Tập các hành động mà agent có thể đi
$T(s' s,a)$	Array{Float64,3}	Hàm chuyển tiếp - Transition Input: s', s, a Output: Xác suất khi agent chuyển từ trạng thái s sang s' bằng hành động a
$R(s,a)$	Array{Float64,2}	Hàm phần thưởng - Reward Input: s,a

		Output: Phần thưởng agent nhận được khi chuyển từ trạng thái s bằng hành động a
γ	Float64	Hệ số chiết khấu - Discount factor

- Các hàm sử dụng

Biến	Kiểu dữ liệu	Mô tả
π	Vector{Int}	Chính sách - Policy Tập sự lựa chọn hành động ở mỗi trạng thái trong bản đồ
U	Vector{Int}	Tính thiết thực - Utility Tổng các phần thưởng có thêm hệ số chiết khấu

b. Discrete MDP

- Giới thiệu: Discrete MDP là mô hình MDP bao gồm:
 - Không gian trạng thái: các đỉnh trạng thái rời rạc.
 - Không gian hành động: tập hữu hạn
- Cấu trúc

Biến	Kiểu dữ liệu	Mô tả
$T(s' s,a)$	Array{Float64,3}	Hàm chuyển tiếp - Transition Input: s', s, a Output: Xác suất khi agent chuyển từ trạng thái s sang s' bằng hành động a
$R(s,a)$	Array{Float64,2}	Hàm phần thưởng - Reward Input: s,a Output: Phần thưởng agent nhận được khi nhảy từ trạng thái s bằng hành động a
γ	Float64	Hệ số chiết khấu - Discount factor

c. HexWorld

- Giới thiệu: Tập input của HexWorld bao gồm: tập tọa độ và tập phần thưởng
- Cấu trúc

Biến	Kiểu dữ liệu	Mô tả
<i>hexes</i>	Vector{Tuple(Int,Int)}	Tập các ô mà agent có thể đứng
<i>rewards</i>	Array{Float64,2}	Tập phần thưởng quy định phần thưởng tương ứng ở mỗi ô (nếu có)
<i>p_intended</i>	Float64	Xác suất chuyển trạng thái theo dự định
<i>p_bumpBorder</i>	Float64	Hệ số nhảy biên
γ	Float64	Hệ số chiết khấu - Discount factor

d. HexWorldMDP

- Giới thiệu: Với tập HexWorld ta mô hình hóa theo DiscreteMDP cho ra cấu trúc HexWorldMDP
- Cấu trúc

Biến	Kiểu dữ liệu	Mô tả
<i>hexes</i>	Array{Float64,3}	Tập tọa độ - Tập các ô mà agent có thể đứng
<i>mdp</i>	Array{Float64,2}	Cấu trúc DiscreteMDP
<i>reward</i>	Float64	Reward - Phần thưởng tương ứng ở mỗi ô

- Cách hàm sử dụng
 - Hàm `hex_neighbors(cell i,j)`: trả về 6 ô liền kề với ô (i,j)
 - Hàm khởi tạo `HexWorldMDP(struct HexWorld)`: từ các dữ liệu của cấu trúc HexWorld hàm này thực hiện tính toán Reward và Transition để trả về DiscreteMDP
 - Hàm tính Reward: tổng các giá trị phần thưởng được giới hạn bởi hệ số discount factor

$$R = \sum_{t=1}^{\infty} \gamma^{t-1} r_t$$
 trong bài HexWorld ta sẽ chọn $\gamma=0.9$

3.2. Phương pháp giải quyết

Mục đích: Đạt được **optimal policy**

Thực hiện: Sự kết hợp giữa **policy evaluation** và **policy improvement**

a. Các ký hiệu

$\pi(s)$	Policy trả về hành động tại trạng thái s
$U^\pi(s)$	Value function tương ứng với trạng thái s

b. Policy evaluation

Chúng ta sẽ đánh giá chính sách bằng **Value function** U^π theo 2 cách như sau:

Cách 1: Sử dụng vòng lặp

- Giá trị U: Với mỗi trạng thái s, hành động a, sử dụng để tính **lookahead**
 - $n=1 \Rightarrow U_1^\pi(s) = R(s, \pi(s))$
 - $n=k+1 \Rightarrow U_{k+1}^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' | s, \pi(s)). U_k^\pi(s')$
- Lặp k_max lần giá trị U: trả về **Value function** U^π là dãy giá trị **U** tương ứng với chuỗi hành động được xét ở mỗi trạng thái, tương ứng **S** \rightarrow **U**

$$U^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' | s, \pi(s)). U^\pi(s')$$

- Convergence: luôn đảm bảo hội tụ (được chứng minh bởi contraction mapping)

Cách 2: Sử dụng ma trận - Không sử dụng vòng lặp (One-step learning)

- Tại đây, ta sử dụng ma trận để đánh giá chính xác **Value function** U^π

$$U^\pi = R^\pi + \gamma T^\pi U^\pi$$

$$\Leftrightarrow U^\pi = (1 - \gamma T^\pi)^{-1} R^\pi$$

c. Policy improvement

Tại đây chúng ta sẽ sử dụng **Value function** U^π kết hợp với **Policy** π .

- Đầu tiên, ta khởi tạo bằng **Policy** π bằng **argument max (argmax)**

$$\pi(s) = \text{argmax}(U^\pi)$$

- Ta gọi đây là greedy policy, từ hàm này sẽ chọn được một hành động tối ưu cho mỗi trạng thái

d. Optimal policy

Nếu U là “**optimal value function**” thì ta sẽ có được một “**optimal policy**”

- **Optimal value function**

- **Q-function** hay còn gọi là **action value function**, hàm này trả về giá trị mong đợi khi thực hiện một trạng thái s và hành động a

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s' | s, a) \cdot U^\pi(s')$$

- Ta được Value function $U(s) = \max Q(s, a)$

- **Policy improvement và Optimal value function**

$$\pi(s) = \operatorname{argmax} Q(s, a)$$

e. Policy Iteration

- Sử dụng vòng lặp cho đến khi optimal policy hội tụ tại một giá trị. Sau mỗi vòng lặp chính sách sẽ được cải thiện và về một giá trị, đây là mục đích và kết quả cuối cùng cho bài toán.
- Ý tưởng:

```

for k = 1:M.k_max
    U = policy_evaluation(P, pi)           #lặp k_max vòng (đủ lớn cho bài toán)
    pi' = ValueFunctionPolicy(P, U)       #optimal value function
    if all(pi(s) == pi'(s) for s in S)    #policy improvement
        break                             #kiểm tra điều kiện hội tụ
    end
    pi = pi'
end
return

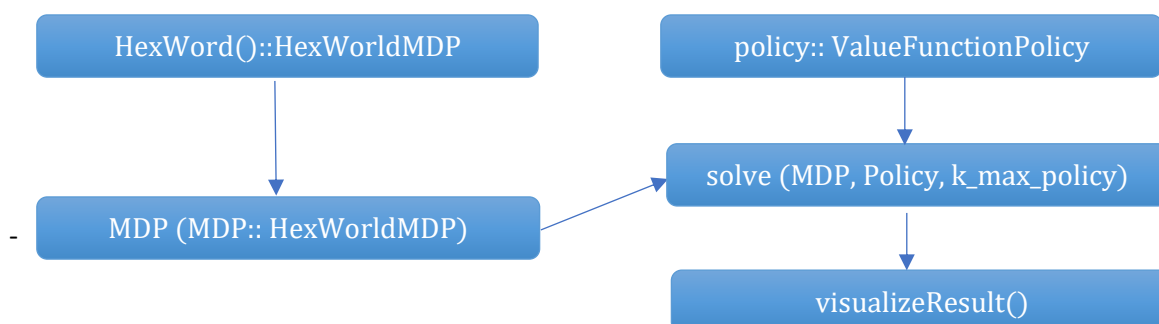
```

f. Lý do chọn phương pháp

- Mô hình MDP đặc thù cho bài toán quyết định, Discrete MDP phù hợp với yêu cầu và không gian bài toán HexWorld đặt ra
- Cài đặt luôn dẫn đến quyết định đúng tại giá trị hội tụ (đã được chứng minh)
- Phương pháp kinh điển của mô hình MDP

3.3. Flow code

a. Flow chính

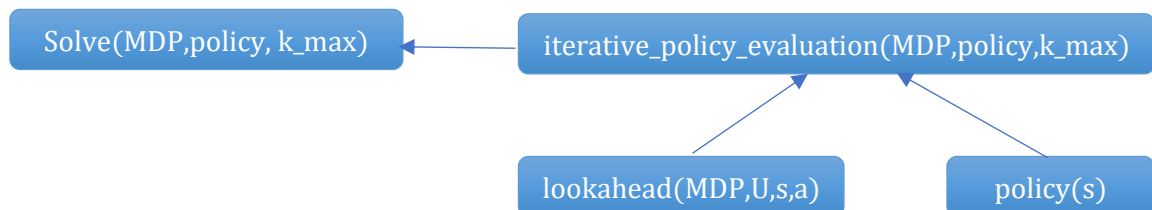


- Hàm **HexWorld()**: Khởi tạo HexWorldMDP từ tập dữ liệu {State, Reward }
- Hàm **MDP(MDP::HexWorldMDP)**: Khởi tạo MDP từ HexWorldMDP
- **Policy::ValueFunctionPolicy(MDP,U)**: Khởi tạo từ MDP và U (trong đó U được khởi tạo bằng một mảng các phần tử 0), đây là mảng lưu chính sách quyết định hành động tại mỗi trạng thái.
- Hàm **ValueFunctionPolicy(MDP,U)**: Gọi hàm greedy() trả về các U tương ứng
- Hàm **Solve(mdp::MDP,policy,k_max)**: Trả về là 1 mảng các hành động (1/2/3/4/5/6) cho mỗi mỗi trạng thái trong bản đồ.
- Hàm **visualizeResult()**: In kết quả là bản đồ các trạng thái tại mỗi trạng thái là một hướng đi

b. Hàm hỗ trợ



- Hàm **lookahead(MDP,U,s,a)** : Trả về giá trị U tương ứng với trạng thái s khi thực hiện hành động a trong MDP
- Hàm **greedy(MDP,U,s)**: Trả về giá trị hành động s và giá trị u được chọn cho trạng thái s bằng cách tìm giá trị U lớn nhất trong khi thực hiện 1 trong 6 hành động -> findmax(lookahead())
- Hàm **policy(s)**: Trả về hành động được quyết định tại trạng thái s sau khi gọi hàm hỗ trợ greedy()



- Hàm **iterative_policy_evaluation (MDP,policy,k_max)**: Trả về là 1 mảng U sau khi lặp k_{max} lần với policy tương ứng với 2 hàm hỗ trợ là lookahead() và policy(s)
- Function **Solve(MDP,policy,k_max)**: Trả về là 1 mảng các hành động là giá trị sau khi hội tụ khi lặp k_{max} lần. Sau mỗi vòng lặp thứ k hàm iterative_policy_evaluation() được gọi và cải tiến policy

3.4. Visualize

a. Cách thực hiện

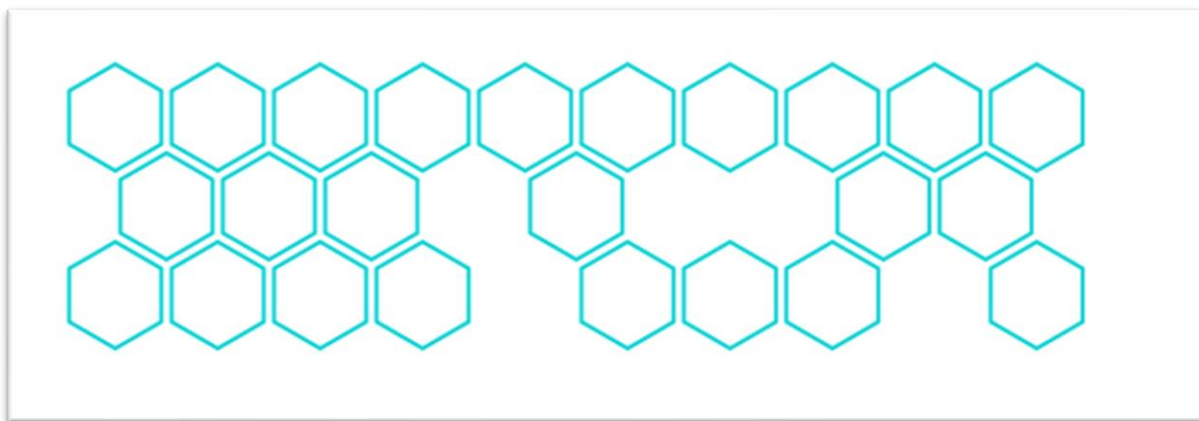
- Sử dụng package Luxor, Colors của Julia

- Chi tiết các hàm thực hiện được comment trong code

b. Qui ước

b.i. Bản đồ Hex World

Hex World bao gồm các ô hình lục giác đều, mỗi ô có thể có chung cạnh nhiều nhất với 6 ô khác liền kề, những ô này gắn liền nhau tạo thành một bản đồ, dưới đây là hình ảnh minh họa một bản đồ gồm 24 ô ngẫu nhiên.



b.ii. Xác định ô (State)

Các ô trong bản đồ hiển thị theo 2 chỉ số (i,j)

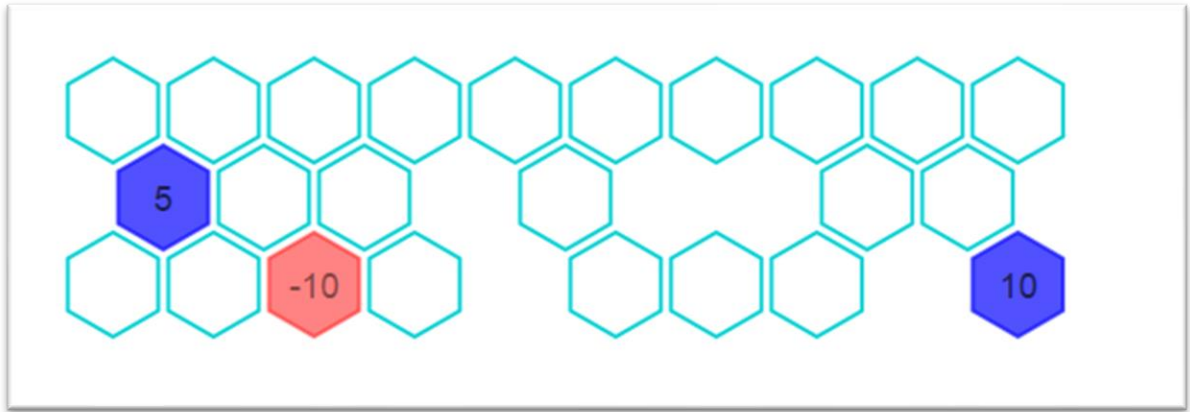
- Mỗi hàng cùng chỉ số $j \Rightarrow i$ tăng dần từ trái sang phải
- Mỗi cột hợp với phương ngang một góc 60 độ có cùng chỉ số $i \Rightarrow j$ tăng dần từ dưới lên



b.iii. Phần thưởng (Reward)

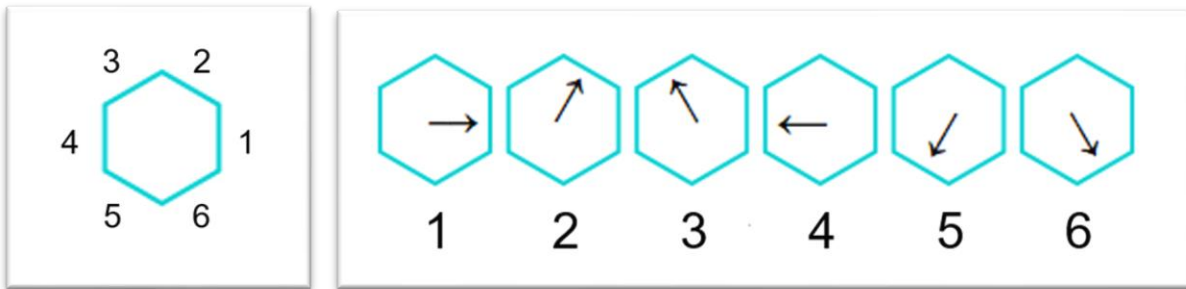
Trong bản đồ Hex World, các ô có phần thưởng sẽ được tô màu tương ứng:

- Màu xanh: Điểm thưởng dương
- Màu đỏ: Điểm thưởng âm



b.iv. Hướng đi (Action)

Một tác nhân có thể đi theo 6 hướng để đi đến 6 ô hàng xóm liền kề trạng thái hiện tại của nó được qui ước như sau, các mũi tên dùng để chỉ hướng đi tương ứng.



c. Khởi tạo

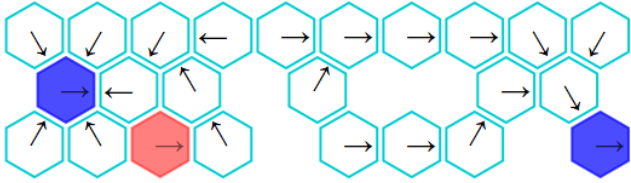
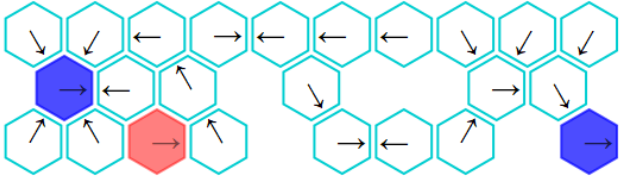
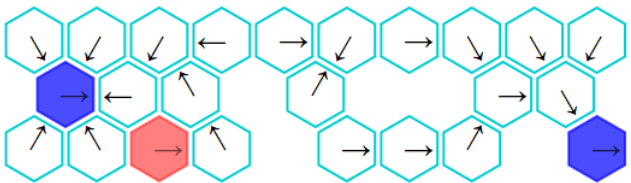
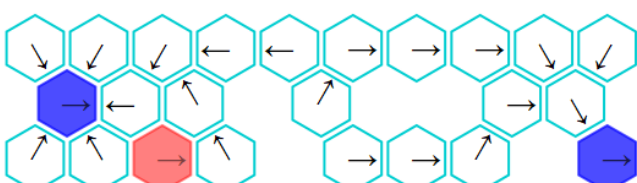
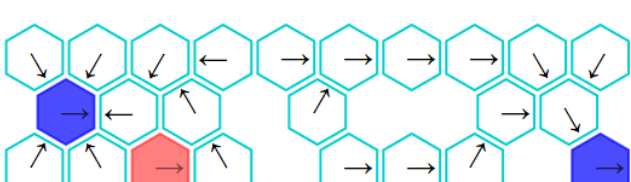
Ta sẽ khởi tạo một bản đồ Hex World bao gồm:

Tập trạng thái	Tập phần thưởng
$(0,0), (1,0), (2,0), (3,0), (0,1), (1,1), (2,1), (-1,2),$	$(0,1) \Rightarrow 5.0$
$(0,2), (1,2), (2,2), (3,2), (4,2), (5,2), (6,2), (7,2),$	$(2,0) \Rightarrow -10.0$
$(8,2), (4,1), (5,0), (6,0), (7,0), (7,1), (8,1), (9,0)$	$(9,0) \Rightarrow 10.0,$



d. Quá trình hội tụ

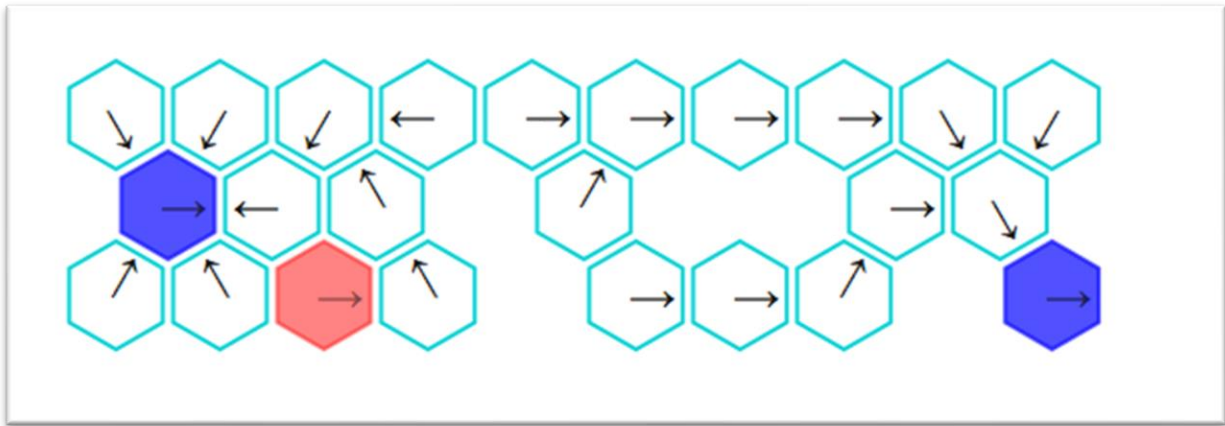
k_max	k	Policy sau mỗi vòng lặp
10	1	
	2	
	3	

	4	
	≥ 5	Dừng
100	1	
	2	
	3	
	4	
	≥ 5	Dừng

3.5. Phân tích

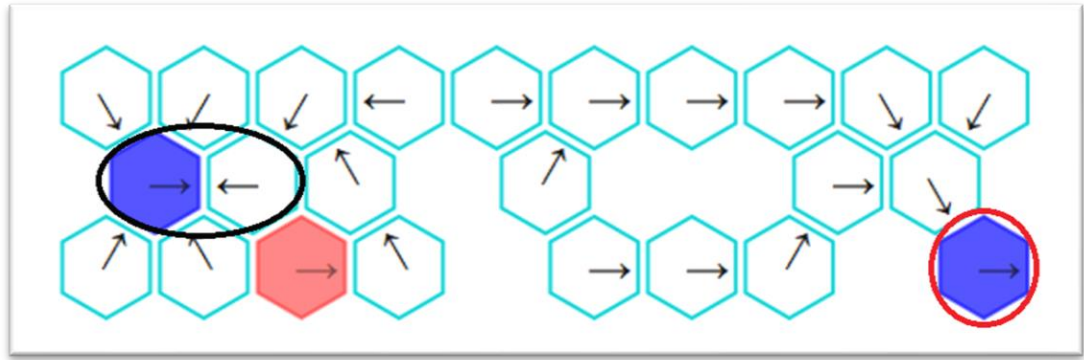
a. Kết quả đạt được

Sau đây là kết quả của tập input được khởi tạo ở 3.4.c, kết quả chỉ ra hướng đi được quyết định cho tác nhân ở bất kì trạng thái nào trong bản đồ



b. Phân tích kết quả

- Đánh giá mức độ giải quyết vấn đề: thành công.
- Độ tốt của kết quả:
 - Tác nhân không chuyển sang trạng thái rỗng (ô ngắt quãng trong bản đồ) nhờ vào hệ số quy định nhảy biên tại trạng thái hiện tại khi khởi tạo MDPHexWorld từ HexWorld. Vì vậy tập hành động tại một ô trạng thái đã được tối giản đi.
 - Các hành động được quyết định cho tác nhân đều tránh xa các ô màu đỏ và hướng dẫn đến các ô màu xanh nhờ vào hàm giá trị reward
 - Ở vòng lặp đối với $k_{\max}=10$ hay $k_{\max}=100$ thì PolicyIteration chỉ lặp 4 vòng vì đã nhận ra nó đã hội tụ vì vậy thuật toán sẽ không lặp lại những vòng vô nghĩa.
- Bất cập:
 - Xét về cách nhìn khi thực hiện trò chơi, nếu cho tác nhân thực hiện hành động theo kết quả trong bản đồ này thì sẽ có hiện tượng lặp lại vòng chạy vô cùng tại 2 ô khoanh tròn màu đen mà không bao giờ đến được ô có phần thưởng lớn nhất tại ô khoanh tròn màu đỏ trong hình dưới đây



4. Đánh giá kết quả

4.1. Điểm tốt

- Mô hình MDP được áp dụng thành công và cho ra kết quả khả quan và chính xác.
- Chính sách quyết định luôn hướng tác nhân đi đến ô nhận được phần giá trị phần thưởng lớn nhất.
- Visualization sáng sủa, dễ nhìn, giúp chúng ta dễ hình dung ra được hướng đi.

4.2. Hạn chế

- Chưa thực hiện quá trình cấu hình số vòng lặp để chắc chắn bài toán hội tụ được riêng biệt cho mỗi tập dữ liệu.
- Khi thực hiện bài toán Hex World hành động chỉ được đánh giá là tốt khi nó ở một trạng thái, vì vậy ta cần phát triển bài toán theo hướng nhìn xung quanh, nhìn toàn bộ bản đồ chứ không chỉ là “lookahead” được đề xuất trong phương pháp này.

5. Tài liệu tham khảo

- Sách Algorithm for Decision Making
- CS 188 Lecture 8: MDPs II <https://www.youtube.com/watch?v=S0jvKddQdUU>
- Discrete MDP
https://web.stanford.edu/class/cme241/lecture_slides/DiscreteVSContinuous.pdf
- Julia x MDPs <https://juliaacademy.com/courses/decision-making-under-uncertainty-with-pomdps-jl/lectures/34881770>
- Package Luxor Julia <https://juliagraphics.github.io/Luxor.jl/v0.8.5/>

II. Rock-Paper-Scissors

1. Phát biểu bài toán

- Trò chơi rock-paper-scissors (còn gọi là kéo búa bao hay oẳn tù tì) là một bài toán rất phổ biến trên toàn thế giới.
- Khi chơi, người chơi đưa ra lựa chọn bằng tay, có thể có 2 hoặc nhiều người chơi. Xét trường hợp đơn giản nhất là có 2 người chơi, mỗi người chọn 1 trong 3 loại hành động rock-paper-scissors thông qua hình dạng của bàn tay:
 - Scissors: 2 ngón tay tạo hình chữ V.
 - Rock: gồng tay lại hình nắm đấm.
 - Paper: xòe cả bàn tay ra (ngửa hoặc sấp đều được).
- Luật chơi:
 - Người chơi ra scissors sẽ thắng paper, rock thắng scissors và paper lại thắng rock, điều này tạo ra một vòng tròn khắc chế lẫn nhau.
 - Nếu cả hai đều ra giống nhau, kết quả được tính là hoà và tiếp tục chơi đến khi có một trong hai chiến thắng.
- Bảng biểu diễn các kết quả có thể có giữa hai người, dòng thứ (i,j) thể hiện kết quả của dòng i khi đấu với cột j .

Ví dụ: dòng $(1,2)$ thể hiện rằng rock sẽ thua khi đấu với paper

	Rock	Paper	Scissors
Rock	Hoà	Thua	Thắng
Paper	Thắng	Hoà	Thua
Scissors	Thua	Thắng	Hoà

- **Bài toán:** mặc dù tỉ lệ thắng/thua cho mỗi hành động là như nhau, việc thắng và thua cũng là ngẫu nhiên, song bài toán cần giải quyết là tìm ra chiến lược hợp lý nhất dựa trên các tình huống thực tế; từ đó đưa ra quyết định tối ưu dẫn đến chiến thắng.
- Nhóm đề cập và giải quyết bài toán bằng 2 phương pháp:
 - Xây dựng và giải quyết bài toán dựa trên phương pháp Fictitious Play (phương pháp chính).
 - Xây dựng và giải quyết bài toán dựa trên phương pháp Linear Regression.

2. Thách thức

- Tìm hiểu và xây dựng được một mô hình giải quyết bài toán hợp lý, cho kết quả thực nghiệm đúng đắn.
- Tìm được chiến lược hợp lý cho bài toán mang tính ngẫu nhiên, khi mà xác suất thắng thua và hoà bằng nhau như rock-paper-scissor không phải là việc dễ dàng.
- Cần tìm hiểu thêm một số thư viện để có thể visualize hiệu quả.
- Khó ra quyết định việc lựa chọn trong nhiều phương pháp mà sách đề cập cũng như nhiều nguồn tài liệu github.
- Cần tích hợp và sử dụng khá nhiều thư viện, cần có sự tìm hiểu và lựa chọn sử dụng hợp lý.
- Cần thời gian làm quen với ngôn ngữ Julia.

3. Thực nghiệm cho phương pháp Fictitious Play

3.1. Cấu hình bài toán

- Bài toán gồm 2 agents $I = \{1, 2\}$ và tập action $A = A^1 \times A^2$ với mỗi $A^i = \{\text{rock, paper, scissors}\}$
- Trò chơi có thể chơi một lần hoặc lặp lại nhiều lần với số ván xác định
- Phần thưởng cho các agent được mô tả cụ thể ở bảng sau:

	Rock	Paper	Scissors
Rock	0,0	-1,1	1,-1
Paper	1,-1	0,0	-1,1
Scissors	-1,1	1,-1	0,0

3.2. Mô hình hóa tính toán

- Rock-Paper-Scissors là một Simple Game có cấu trúc như sau:

Biến	Kiểu dữ liệu	Mô tả
γ	Float	Chiết khấu (discount factor) Nhóm chọn $\gamma = 0.9$

\mathcal{I}	Int	Tập các agents Gồm agent 1 và 2
\mathcal{A}	Vector{Vector{Int}}	Tập các actions của agents Gồm 2 mảng, mỗi mảng bao gồm 3 phần tử Rock, Paper, Scissors (dạng symbol)
R	method	Tính toán reward cho action a Trả về mảng gồm 2 rewards tương ứng của 2 agents

3.3. Phương pháp giải quyết

a. Giới thiệu phương pháp Fictitious Play:

- Fictitious Play là phương pháp mà các agents sẽ mô phỏng đóng vai của đối thủ, dựa vào kinh nghiệm từ những lượt chơi trước để tìm ra policy của họ, tiến hành phỏng đoán được nước đi của đối thủ, từ đó cập nhật policy của chính mình và đưa ra best response để tối đa hóa lợi ích nhận được. Fictitious Play không đảm bảo chắc chắn sẽ hội tụ về một cân bằng Nash.
- Cụ thể đối với bài toán này, nhóm sẽ mô phỏng việc chơi game rock-paper-scissors giữa 2 agents và play với số lần lớn. Tại mỗi iteration, phương pháp đánh giá dựa trên số lần ra quyết định mỗi action của đối thủ (action counts), dùng maximum likelihood estimates với policy đối thủ. Từ ước lượng đó, tính toán đưa ra chiến lược tốt nhất có thể (best response), và cập nhật cho policy hiện tại.
- Không có một cân bằng Nash xác định cho bài toán này, bởi bất kỳ chiến thuật xác định nào cũng đều có thể bị khắc chế dễ dàng bởi một action khác. Ví dụ, nếu agent 1 chọn rock thì sẽ bị đánh bại nếu agent 2 chọn paper, tương tự cho các cặp action khác.
- Để tính maximum likelihood estimate, agent i lưu lại số lần agent j (đối thủ) sử dụng action a^j trong bảng Dictionary $N^i(j, a^j)$. Chẳng hạn:

Dict(rock \Rightarrow 33, paper \Rightarrow 40, scissors \Rightarrow 27)

- Agent i tính best response dựa trên việc giả sử rằng mỗi agent j đều tuân theo một stochastic policy như sau:

$$\pi^i a^j \propto N^i(j, a^j) \quad (2.3.1)$$

- Để trực quan dễ hiểu cho cách tính best response, ta xem xét ví dụ sau: từ N^i ta tính được xác suất mỗi action của đối thủ như sau rock $\Rightarrow 0.4$, paper $\Rightarrow 0.25$, scissors $\Rightarrow 0.35$

Đối thủ Agent	Rock (0.4)	Paper (0.25)	Scissors (0.35)	Utility
Rock	0	-1	1	0.1
Paper	1	0	-1	0.05
Scissors	-1	1	0	-0.15

Mặc dù rock đang dẫn đầu về tần suất ra action của đối thủ, thế nhưng xem xét khả năng xuất hiện cũng như reward có thể nhận được của từng action, ta sẽ chọn action **rock** vì nó mang lại utility cao nhất

b. Fictitious Play cho bài toán có cấu trúc như sau:

Biến	Kiểu dữ liệu	Mô tả
\mathcal{P}	SimpleGame	Được đề cập ở 3.2
i	Int	Agent index (1 hoặc 2)
N	Vector{Dict}	Dùng để count số lần ra từng action của từng agents Là mảng gồm 2 Dict ứng với 2 agents, mỗi Dict gồm keys là 3 actions và values là 3 counts tương ứng
π	SimpleGamePolicy { p }	Policy hiện tại, sẽ được dùng để ra quyết định cho ván tiếp theo Là một Dict chỉ chứa duy nhất 1 keys là action ai có xác suất bằng 1 (Dict(ai \Rightarrow 1.0))

c. Lý do lựa chọn phương pháp:

- Dễ theo dõi, mô phỏng được quá trình chơi từng ván, khả năng ra quyết định (policy), số quyết định cho mỗi action ra sao, tỉ lệ thắng
- Fictitious Play là phương pháp dựa trên học tập (learning-based approach), vừa mô phỏng chơi game vừa học và thích ứng với cách chơi của đối thủ, phù hợp với bài toán, nó có thể lặp

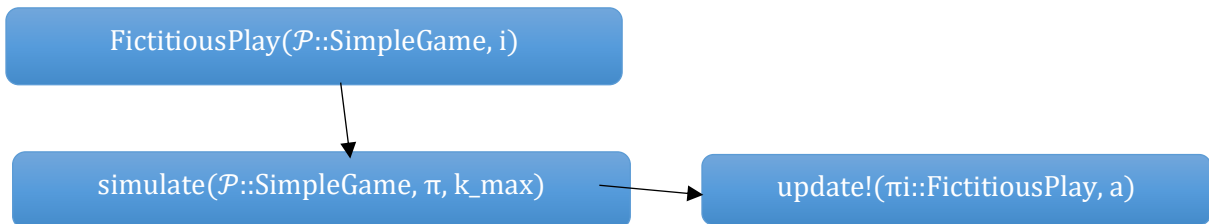
lại với số lượng lớn lần chơi, tính toán được best response cho policy và thực hiện action có utility tối đa nhất tương ứng. Nếu đối thủ ra action ngẫu nhiên, phương pháp cũng thích ứng và tìm cách sao cho đảm bảo không bất lợi hơn.

- Phương pháp phù hợp cho bài toán với cả người hoặc máy chơi, trong khi đó Iterated Best Response là dành cho máy, còn Hierarchical Softmax dành cho agent là người chơi.
- Phương pháp được thử nghiệm với bộ dữ liệu lớn cho kết quả tốt, có thể visualization và dễ theo dõi, code đơn giản.

3.4. Code

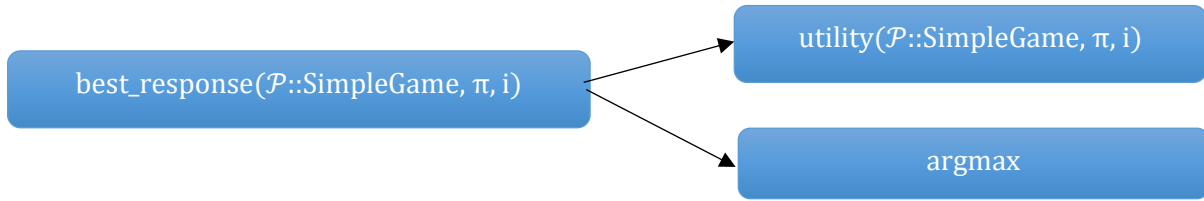
- Tham khảo và sử dụng lại từ sách đã được [trích dẫn](#), nhóm tìm hiểu nhiều phương pháp, từ đó lựa chọn phương pháp Fictitious, sau đó phân tích những chức năng, dòng lệnh và rồi áp dụng vào bài toán.

a. Luồng đi chính:



- **simulate($\mathcal{P}::\text{SimpleGame}$, π , k_max):**
Mô phỏng việc chơi Simple Game \mathcal{P} với π chứa 2 FictitiousPlay tương ứng với 2 agent 1, 2. Mô phỏng thực hiện k_max iterations. Trong mỗi iteration, chọn action a từ current policy π cho cả hai agents, sau đó gọi đến **update!($\pi::\text{FictitiousPlay}$, a)** để cập nhật N^i , tính toán best response cho ván tiếp theo và cập nhật lại π cho agent i
- **FictitiousPlay($\mathcal{P}::\text{SimpleGame}$, i):**
Khởi tạo struct FictitiousPlay cho agent thứ i tương ứng trong SimpleGame \mathcal{P} . Quy ước lúc ban đầu, mỗi action được khởi tạo N^i với count là 1 và Policy có xác suất của mỗi action là $1/3$.

b. Các hàm hỗ trợ:



- **best_response(P::SimpleGame, π, i)**: giúp agent i ra quyết định, trả về SimpleGamePolicy với action a_i là action có utility cao nhất được tính từ policy π
- **utility(P::SimpleGame, π, i)**: $\pi[i]$ là SimpleGamePolicy(Dict($a_i \Rightarrow 1.0$)), trong khi $\pi[j]$ ($j \neq i$) là SimpleGamePolicy(Dict(action \Rightarrow probability)), hàm sẽ trả về utility cho policy $\pi[i]$ của agent i . Utility (độ hữu ích) được tính theo công thức:

$$\sum_{a \text{ in } joint(\mathcal{A})} R(a)[i] * p(a)$$

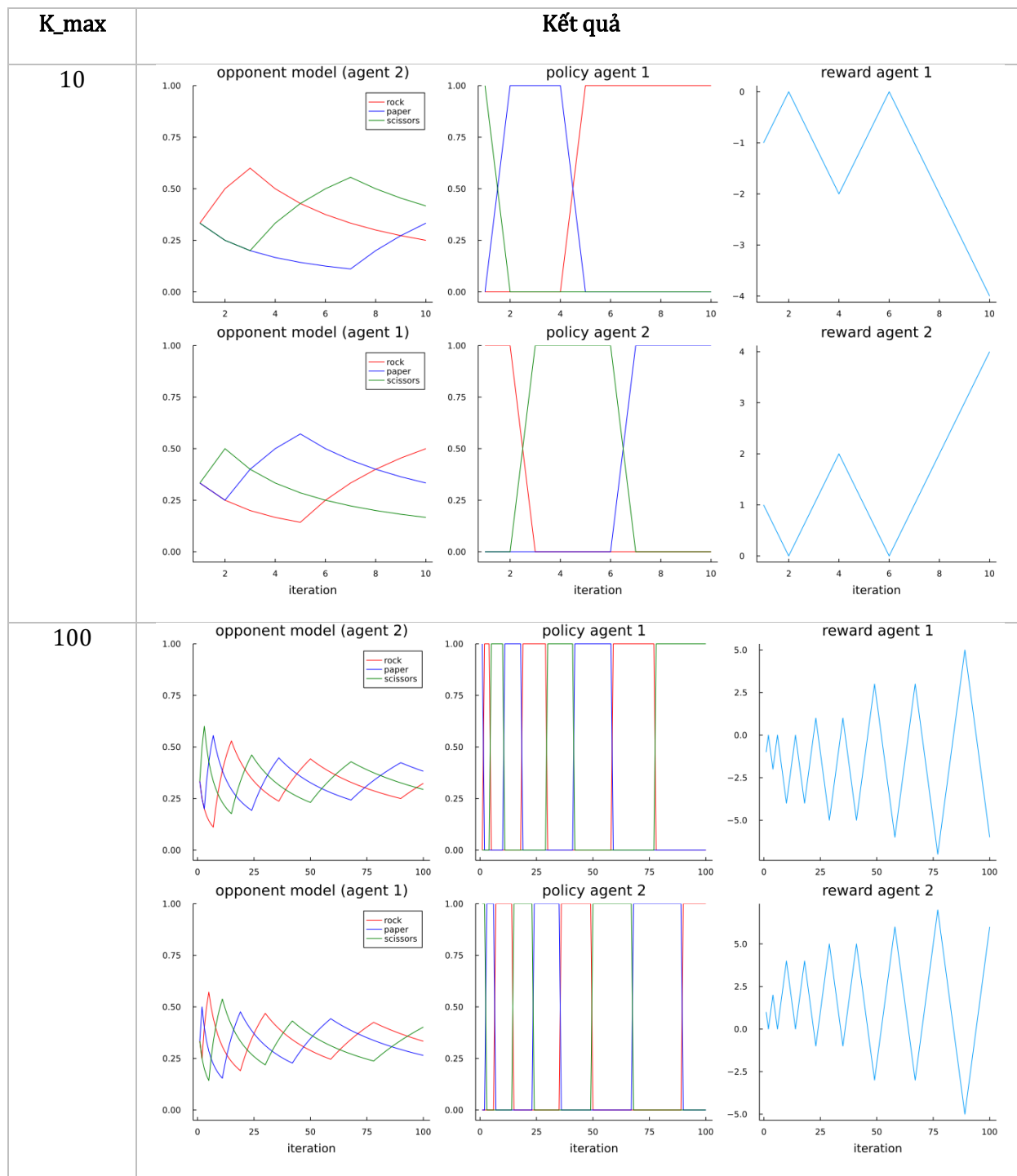
Trong đó:

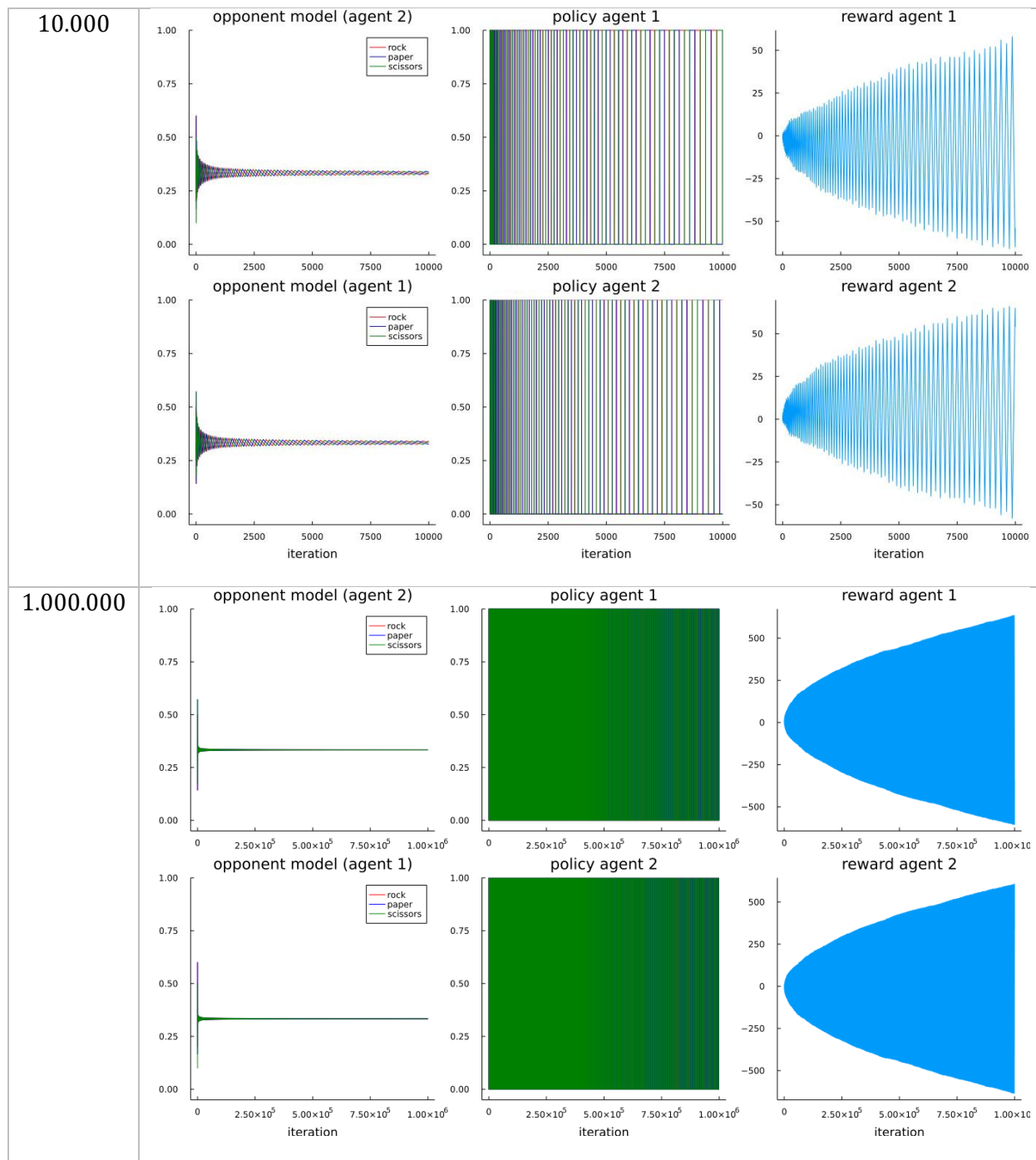
- + $R(a)[i]$ trả về reward tương ứng của agent i khi cả 2 agents chọn action a (thuộc joint action space).
- + $p(a)$: xác suất xảy ra action a .
- **argmax**: chọn lựa và trả về action có utility cao nhất dựa trên function **utility** và actions space của agent i

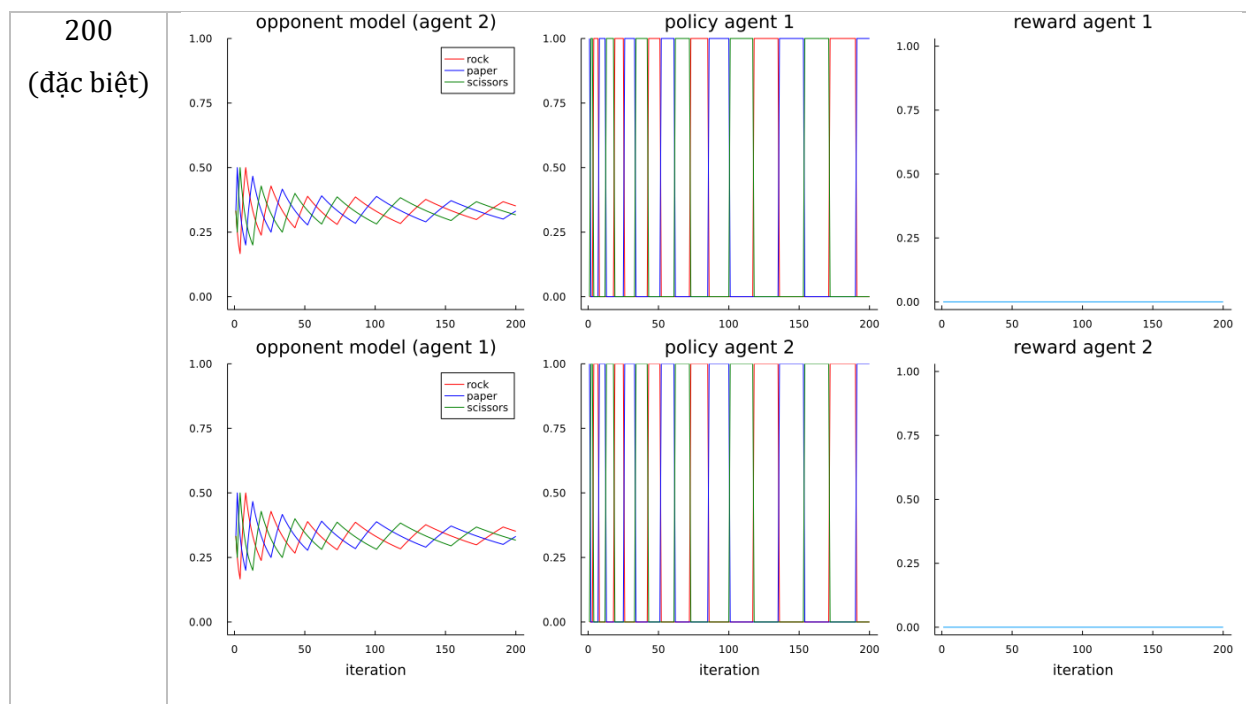
3.5. Phân tích

a. Kết quả đạt được

- Ta sẽ thực hiện thử nghiệm với các trường hợp có tham số `k_max` khác nhau và quan sát kết quả nhận được, kết quả thống kê theo từng iteration của mỗi agent được visualize như sau:
 - Opponent model: thống kê xác suất ra quyết định của đối thủ đối với từng action
 - Policy: thống kê chính sách của agent qua mỗi iteration
 - Reward: số điểm thưởng mà agent nhận được ở mỗi iteration







b. Phân tích kết quả

- Tỷ lệ reward/k_max giảm dần khi k_max tăng ($4/10 \Rightarrow 5/100 \Rightarrow 50/10.000 \Rightarrow 500/1.000.000$)
- \Rightarrow Điều này chứng tỏ cả hai agents đều đã dần dần học và thích nghi được với cách chơi của đối thủ.
- Với mọi k_max, reward của agent 1 và agent 2 luôn có sự cân bằng, cả 2 agents luôn không để thua quá nhiều ván liên tiếp. Do đó kết quả chung cuộc agent nào thắng có thể sẽ phụ thuộc vào k_max(iterations).
- Khả năng thắng của 2 agents đều không có sự chênh lệch nhiều do đều học theo cùng một phương pháp.
- Với k_max đủ lớn, mô hình dần hội tụ về tỷ lệ rock = paper = scissors = 1/3, điều này thể hiện được tính đúng đắn của mô hình và phương pháp vì vốn dĩ đây là trò chơi khả năng ra 3 action là ngẫu nhiên.
- **Trường hợp đặc biệt:** reward của cả 2 agents ở mọi thời điểm đều bằng 0. Nguyên nhân do cả 2 agent đều học theo cùng 1 phương pháp, nên khả năng ra quyết định giống nhau là không hiếm gặp

4. Thực nghiệm cho phương pháp Linear Regression

4.1. Cấu hình bài toán

- Gồm một người và một máy, trò chơi diễn ra trong thời gian thực khi người chơi sẽ chọn một trong ba hành động Rock, Paper hoặc Scissors. Dựa trên lịch sử ra quyết định của người chơi, máy sẽ dự đoán đưa ra lựa chọn tối ưu để chiến thắng.
- Dự đoán dựa vào phương pháp Linear Regression.
- Đếm số lần thắng của mỗi người chơi, từ đó nhìn ra được kết quả chung cuộc.
- Trận đấu sẽ kết thúc sau 20 iteration (lần đấu).

4.2. Mô hình hóa tính toán

- Mỗi hành động rock, paper, scissor sẽ tương ứng với giá trị 1, 2, 3
- Để lưu lịch sử của người chơi, ta sử dụng một DataFrame sao cho ứng với row thứ i sẽ là lastResponse, secondToLastResponse, thirdToLastResponse tính từ round thứ i, đồng thời lưu lại correctAiResponse (tức là nếu ra action này thì AI sẽ chiến thắng).
- Lúc ban đầu, chuẩn bị tập dataset dùng cho training cho AI để khởi động như sau:

8x4 DataFrame				
Row	lastResponse Int64	secondToLastResponse Int64	thirdToLastResponse Int64	correctAiResponse Int64
1	1	1	2	2
2	3	1	1	1
3	2	3	1	3
4	2	2	3	3
5	3	2	2	1
6	1	3	2	2
7	1	1	3	2
8	1	1	1	2

- Dùng Linear model để xử lý, thiết lập mối quan hệ giữa lastResponse, secondToLastResponse, thirdToLastResponse với correctAiResponse
- Người chơi nhập input vào: chọn 1 trong 3 action tương ứng với các số 1, 2, 3.

Rock	1
Paper	2
Scissors	3
last_response	lần chơi gần nhất

second_to_last_response	lần chơi gần nhất thứ 2
third_to_last_response	lần chơi gần nhất thứ 3
ai_victory	số lần thắng của AI
user_victory	số lần thắng của người chơi

- Một Victory Dictionary: tra cứu action này sẽ bị đánh bại bởi action tương ứng nào

Action	Action thắng
Rock	Paper
Paper	Scissors
Scissors	Rock

4.3. Phương pháp giải quyết

a. Giới thiệu phương pháp Linear Regression:

- Một trong những thuật toán cơ bản nhất (và đơn giản nhất) của Machine Learning. Đây là một thuật toán Supervised learning (Học có giám sát).
- Hồi quy chính là một phương pháp thống kê để thiết lập mối quan hệ giữa một biến phụ thuộc và một nhóm tập hợp các biến độc lập, cụ thể ở đây là correctAiResponse phụ thuộc vào lastResponse, secondToLastResponse, thirdToLastResponse

b. Lý do lựa chọn phương pháp:

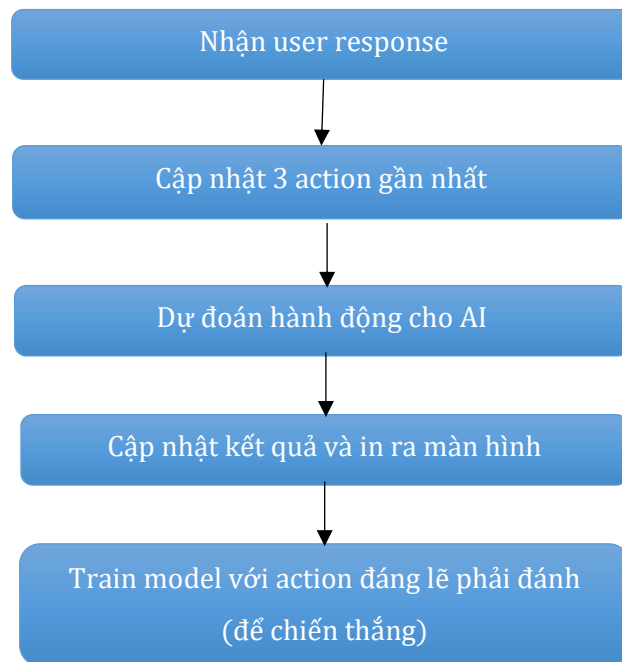
- Vì AI theo dõi lịch sử ra quyết định xuyên suốt từng round nên có thể dự đoán được xu hướng hoặc tính cách của người chơi (mỗi người sẽ có cách chơi riêng mình, không ai giống ai). Hơn nữa, tâm lý người chơi thường không nhớ được nhiều, đa phần họ thường phân tích kết quả của 3 lần chơi gần nhất để đưa ra quyết định cho lần chơi hiện tại, cho nên thiết lập mối quan hệ như vậy là khá hợp lý. Từ mối quan hệ này, AI tiến hành dự đoán người chơi sẽ tiếp tục có những quyết định như thế nào trong tương lai và đưa ra phương án phản đòn tốt nhất. Đây là phương pháp linh hoạt và có hiệu quả thật sự trong thời gian thực, vừa chơi vừa train model, vừa học.
- Mang tính thực tế cao.

- Nhóm gợi ý một cách tiếp cận khác cho bài toán Rock-Paper-Scissors dưới góc nhìn agent đối thủ là con người và trực tiếp chơi.

4.4. Code

Tham khảo từ github đã được [trích dẫn](#), đây là thành quả mà nhóm đã nghiên cứu và thể hiện dưới dạng ngôn ngữ Julia.

a. Luồng đi chính:



- Nhận user response:
`user_response = parse{Int, readline()}`
- Cập nhật 3 action gần nhất:
`third_to_last_response = second_to_last_response`
`second_to_last_response = last_response`
`last_response = user_response`
- Dự đoán hành động cho AI:
`ai_response_raw = AiPredict(last_response, second_to_last_response, third_to_last_response)`
`ai_response = ProcessPrediction(ai_response_raw)`
- Cập nhật kết quả và in ra màn hình:

```
result = VictoryCounter(user_response, ai_response)
```

```
.....
```

```
println("Score: AI - $ai_victory , You - $user_victory")
```

- Train model với action đáng lẽ phải đánh (để chiến thắng): gọi hàm TrainModel() với DataFrame đã được cập nhật từ:

```
push!(df,[last_response,second_to_last_response,third_to_last_response,correct_ai_response])
```

b. Một số hàm phụ trợ:

- **AiPredict(data_1, data_2, data_3)**: từ 3 action của 3 lần gần nhất mà người chơi ra, dùng Linear Regression dự đoán được giá trị tương ứng
- **ProcessPrediction(data)**: tính toán và xử lý giá trị predict trả về tương ứng với 3 action
- **TrainModel()**: train lại model với DataFrame đã được cập nhật
- **VictoryCounter(user, ai)**: tính toán giữa action của user và ai và cập nhật số lượt thắng tương ứng.

4.5. Phân tích kết quả đạt được và Ý nghĩa

a. Kết quả đạt được

- Người chơi sẽ tham gia bằng cách nhập 1 trong 3 lựa chọn (từ 1 đến 3) để ra quyết định.

```
ROCK-PAPER-SCISSORS GAME WITH AI PREDICTION
(1: Rock, 2: Paper, 3: Scissors) => You choose: █
```

- Sau mỗi ván, hiển thị thông báo kết quả của ván đó cùng tổng số điểm(số lần chiến thắng) của 2 bên:

```
ROCK-PAPER-SCISSORS GAME WITH AI PREDICTION
(1: Rock, 2: Paper, 3: Scissors) => You choose: 1
Union{Missing, Float64}[1.8610354223433252]
AI says: Rock
You say: Rock
--> DRAW

Round: 1
Score: AI - 0 , You - 0
-----
(1: Rock, 2: Paper, 3: Scissors) => You choose: █
```

Ví dụ trên là cả AI và người chơi đều ra Rock, kết quả hoà ở round 1.

- Lặp lại trò chơi với 20 rounds, nhóm thử nghiệm có lúc đánh theo chiến thuật, đôi lúc đánh ngẫu nhiên và nhận được kết quả như sau:

Giải thích bảng kết quả

- Mỗi hành động ứng với một con số tương ứng:
ROCK là 1, PAPER là 2, SCISSORS là 3
- Cặp lựa chọn (Người, AI) tương ứng với hai hành động của người và máy trong 1 round, chẳng hạn (1,1) là người và máy đều chọn ROCK
- Để đánh giá một round xem Người như thế nào với AI, ta chia thành 3 mức:
Thắng: ô được tô màu **đỏ**
Hoà: ô được tô màu **xanh**
Thua: ô được tô màu **đen**

Ví dụ:

1) Tại Round thứ 1, (Người, AI) lựa chọn (1,1), nghĩa là chọn (ROCK, ROCK) nên kết quả là Người hoà với AI hoà (tô màu **xanh**)

2) Tại Round thứ 2, (Người, AI) lựa chọn (2,1), nghĩa là chọn (PAPER, ROCK) nên kết quả là Người thắng AI (tô màu **đỏ**)

(Người, AI) Round	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									

16									
17									
18									
19									
20									

Kết quả chung cuộc:

```
(1: Rock, 2: Paper, 3: Scissors) => You choose: 2
Union{Missing, Float64}[2.162162162162162]
  AI says: Paper
  You say: Paper
  --> DRAW
```

```
Round: 20
Score: AI - 11 , You - 4
-----
```

Vậy ở lần chơi này, AI thắng với kết quả áp đảo là 11-4

Trường hợp đặc biệt: khi đã hoà một lần rồi nhưng vẫn liên tục ra quyết định đó, AI vẫn sẽ tiếp tục hoà với người chơi. Nhưng sau một lượng tương đối số lần như vậy, AI sẽ học được cách chiến thắng. Nhận thấy rằng predicted value của AI ngày càng gần với action cần phải đánh, tuy nhiên nó tăng khá chậm. Sau đó, nếu đột ngột đổi chiến thuật sang action chiến thắng thì AI thua.

```
ROCK-PAPER-SCISSORS GAME WITH AI PREDICTION
(1: Rock, 2: Paper, 3: Scissors) => You choose: 1
Union{Missing, Float64}[1.8610354223433252]
  AI says: Rock
  You say: Rock
  --> DRAW

Round: 1
Score: AI - 0 , You - 0
-----

(1: Rock, 2: Paper, 3: Scissors) => You choose: 1
Union{Missing, Float64}[1.9108391608391624]
  AI says: Rock
  You say: Rock
  --> DRAW

Round: 2
Score: AI - 0 , You - 0
-----

(1: Rock, 2: Paper, 3: Scissors) => You choose: 1
Union{Missing, Float64}[1.9343629343629334]
  AI says: Rock
  You say: Rock
  --> DRAW

Round: 3
Score: AI - 0 , You - 0
-----

(1: Rock, 2: Paper, 3: Scissors) => You choose: 1
Union{Missing, Float64}[1.9480651731160896]
  AI says: Rock
  You say: Rock
  --> DRAW
```

- Không phải lúc nào AI cũng chiến thắng, nhưng tỉ lệ thua cũng không cao.

b. Ý nghĩa của kết quả

- Mặc dù trong những nỗ lực đánh chiến thuật lúc đầu, nhận thấy rằng dần dần AI đã trở nên “thông minh” hơn, ra những quyết định hợp lý và áp đảo người chơi.
- Khi đánh ngẫu nhiên vào lúc sau, AI đã học và đoán được những hành động không để thua đối thủ (thắng hoặc hoà) với tỉ lệ khá ổn.
- Nhóm đã thử nghiệm với khá nhiều ván đấu và đưa ra kết luận như sau:
 - Càng về sau, AI sẽ càng học thông minh hơn và người chơi càng bất lợi. Thông thường, từ những ván đấu thứ 10, AI sẽ win nhiều hơn người chơi.

- Mô hình cho kết quả khả quan trong phần lớn trường hợp (AI hoà hoặc thắng).
 - Khi theo dõi 3 lần gần nhất, sẽ đoán được tính cách, chiến thuật hay thói quen của đối phương.
 - Mô hình không phải lúc nào cũng cho kết quả cao, cũng có đôi lúc đánh thua nếu người chơi ra ngẫu nhiên hoặc hiểu được cách đoán của AI để phản đòn.
- Việc hoà liên tục chứng tỏ AI cũng không học “thông minh lắm”, nhưng hoà là một kết quả có thể chấp nhận được.
 - AI không phải lúc nào cũng chiến thắng, điều này là tất yếu do vốn dĩ không thể tìm được phương pháp nào chắc thắng cho bài toán ngẫu nhiên như thế này. Tuy nhiên, AI đã làm tốt trong việc không để thua nhiều, nhất là khi đối thủ đánh theo chiến thuật thì tỉ lệ thắng AI sẽ cao hơn đánh ngẫu nhiên.

5. Đánh giá kết quả

5.1. Phương pháp Fictitious Play

- Đánh giá mức độ giải quyết vấn đề: thành công.
- Độ tốt của kết quả:
 - Theo dõi được các thông số như opponent model, policy, reward qua visualization. Mô phỏng được quá trình chơi và ra quyết định.
 - Khảo sát số lần chơi từ nhỏ đến lớn, có cái nhìn tổng quát và đầy đủ
 - Kết quả đúng đắn như kỳ vọng bài toán, biết cách phân tích và giải thích.
- Bất cập:
 - Hiện nhóm chưa có điều kiện để test policy này khi chơi với **người** ở số lần k_{\max} lớn và với nhiều người khác nhau.
 - Thường có tình huống cả 2 đều hoà nhau ở các ván do ra quyết định giống nhau.

5.2. Phương pháp Linear Regression

- Đánh giá mức độ giải quyết vấn đề: thành công.
- Độ tốt của kết quả:
 - Mô phỏng được quá trình chơi và ra quyết định của mỗi ván trong thời gian thực (real-time).
 - Kết quả đúng đắn như kỳ vọng bài toán, biết cách phân tích và giải thích.

- Kết quả ổn định, AI đã làm tốt trong việc không để thua nhiều, nhất là khi đối thủ đánh theo chiến thuật thì tỉ lệ thắng AI sẽ cao hơn đánh ngẫu nhiên.
- Bất cập:
 - Chưa có nhiều thời gian để nghiên cứu với số lần chơi lớn
 - Trường hợp hoà nhau liên tục chưa được giải quyết, phương pháp linear model train theo 3 tham số được đề cập cho bài toán này là chưa tối ưu

6. Tài liệu tham khảo

<https://algorithmsbook.com/files/dm.pdf>

<https://algorithmsbooks.github.io/DecisionMakingProblems.jl/>

<https://medium.com/m/globalidentity?redirectUrl=https%3A%2F%2Ftowardsdatascience.com%2Fintroduction-to-fictitious-play-12a8bc4ed1bb>

<https://viblo.asia/p/linear-regression-hoi-quy-tuyen-tinh-trong-machine-learning-4P856akRIY3>

<https://github.com/justCodeThings/rockpaperscissors>

[Tutorial · Luxor \(juliagraphics.github.io\)](#)

[Overview · Plots \(juliaplots.org\)](#)

III. Traveler's Dilemma

1. Phát biểu bài toán

- Giới thiệu: Traveler's Dilemma, hay còn gọi là thể tiến thoái lưỡng nan của lữ khách là một trò chơi có tổng khác không, trong đó hai người chơi cố gắng tối đa hóa phần thưởng của riêng họ mà không quan tâm đến người kia. Có các bài toán tương tự như Prisoner's Dilemma (thể tiến thoái lưỡng nan của người tù) hay Diner's Dilemma (thể tiến thoái lưỡng nan của thực khách), ... Đặc trưng các trò chơi này là chúng thể hiện “nghịch lý của sự hợp lý”, khi mà việc đưa ra các quyết định một cách phi logic hay ngây thơ thường dẫn đến một kết quả tốt hơn trong bài toán Lý thuyết trò chơi.
- Nội dung trò chơi: Bài toán đặt ra một kịch bản trong đó một hãng hàng không làm mất hai vali giống hệt nhau của hai hành khách khác nhau. Người quản lý hàng không sẵn sàng đền bù cho họ và nói với hai hành khách viết một con số từ 2 đến 100 \$ giá trị của vali mà không cho người kia biết.

$$Xem \begin{cases} a, a_{-i} \text{ lần lượt là con số ghi xuống của hành khách 1 và 2} \\ r_i, r_{-i} \text{ lần lượt là số tiền nhận về của hành khách 1 và 2} \end{cases}$$

Số tiền mỗi hành khách nhận về sẽ phụ thuộc vào con số đó với những điều kiện sau đây:

Trường hợp	r_i	r_{-i}
$a_i = a_{-i}$	a_i	a_i
$a_i < a_{-i}$	$a_i + 2$	$a_i - 2$
$a_i > a_{-i}$	$a_{-i} - 2$	$a_{-i} + 2$

- Xu hướng con người sẽ chọn giá trị trong khoảng từ \$97 đến \$100. Tuy nhiên ngược lại với điều đó, có một giá trị cân bằng Nash cho bài toán này là \$2. Phần tới đây sẽ đề cập rõ hơn về cân bằng Nash cũng như đưa ra thực nghiệm để khảo sát và đánh giá rõ hơn cho nhận định trên.

2. Thách thức

- Tìm hiểu và xây dựng được một mô hình giải quyết bài toán hợp lý, cho kết quả thực nghiệm đúng đắn.

- Việc tìm hiểu các bài toán có liên quan đến Traveler's Dilemma để đưa ra được đánh giá chung cho dạng toán này khá phức tạp
- Nguồn tài liệu cho các giải thuật khá hạn chế nên việc chốt lọc thông tin và đề xuất giải pháp ban đầu gặp khó khăn
- Các phương pháp có trong sách được đề cập khá cụ thể, nhưng cần thời gian để đưa ra nhận định và chọn xem giải thuật nào là phù hợp và tốt nhất trong số đó

3. Thực nghiệm

Như đã phát biểu ở phần 3.1, kết quả của con người có xu hướng lựa chọn là \$97 đến \$100, trong khi đó cân bằng Nash lại chỉ ra kết quả là \$2. Vậy thì lý do mà lại có hai kết quả khác nhau như trên là do đâu?

- Thứ nhất, việc con người lựa chọn số tiền trong khoảng lớn (\$97 đến \$100) là bởi vì họ không thật sự muốn phải thắng thua hết mình với người kia, tức là không thật sự xem xét kĩ đối phương chọn bao nhiêu để nâng lợi ích nhận được của mình lên cao hơn. Đơn giản ta chỉ nghĩ việc chọn số tiền ở mức cao thì lợi nhuận thu về ta cũng sẽ cao mà thôi.
- Thứ hai, ta cần phải hiểu về cân bằng Nash của Lý thuyết trò chơi. Cân bằng Nash (Nash Equilibrium) là trạng thái hai hoặc nhiều người tham gia trò chơi bất hợp tác (noncooperative game) không có động cơ để đi chệch khỏi chiến lược ban đầu của họ sau khi xem xét chiến lược của các đối thủ. Có thể nói cân bằng Nash sẽ tính toán tất cả các chiến lược có thể để nâng lợi ích của bản thân lên mức tối đa so với đối thủ.

Đặt trong trường hợp bài toán Traveler's Dilemma này, nếu 2 hành khách cùng chọn \$100 thì giá trị 2 bên nhận cùng là \$100. Tuy nhiên A thấy rằng nếu viết $\$99 < \100 của B, khi đó số tiền A nhận được là $\$99 + \$2 = \$101$, tốt hơn nhiều so với \$100 ban đầu, và khi đó B sẽ chỉ còn $\$100 - \$2 = \$98$. Và tương tự B cũng sẽ suy đoán A sẽ chọn \$99, nên chọn \$98 để nâng lợi ích lên $\$98 + \$2 = \$100$, tốt hơn nhiều so với \$98 khi này. Điều này sẽ lặp lại liên tục đến khi cả hai bên cùng chọn \$2 (giá trị thấp nhất có thể lựa chọn). Đó cũng chính là giá trị cân bằng Nash cho bài toán này, khi mà cả hai hành khách đều muốn nâng tối đa lợi ích của bản thân so với đối thủ.

Từ đó ta đã hiểu được tại sao lại có hai kết quả khác nhau như vậy, và ở đây nhóm cũng đưa ra hai thực nghiệm tương ứng với hai trường hợp trên để rút ra được nhận xét ứng với từng thực nghiệm:

- Thực nghiệm 1: Tính toán cho trường hợp 2 người chơi là máy (vì máy luôn muốn tính ra giá trị tốt nhất có thể nên sẽ đi tìm cân bằng Nash của bài toán)

- Thực nghiệm 2: Tính toán cho trường hợp 2 người chơi là con người (có sự lưỡng lự trong việc đưa ra quyết định, có thể xem là 1 biến số đại diện cho việc xác suất đưa ra lựa chọn)

3.1. Mô hình hóa tính toán

- Tương tự như bài toán Rock-Paper-Scissors, Traveler's Dilemma cũng dựa trên mô hình cấu trúc Simple Game, đây là một mô hình cơ bản cho phương pháp suy luận nhiều người chơi (multiagent), trong đó:
 - Chiết khấu (discount factor) γ : Với γ đủ lớn sẽ ảnh hưởng đến quyết định của những người chơi có tiếp tục thực hiện chính sách được đề ra hay không. Trong bài toán này, ta sẽ chọn mặc định γ là 0.9
 - \mathcal{I} : tập hợp các agents, ở đây gồm 2 người chơi là 2 hành khách
 - \mathcal{A} : tập hợp các hành động (action) của agents, mỗi người chơi sẽ có thể chọn giá trị từ \$2 đến \$100. \mathcal{A} trả về 1 mảng gồm 2 mảng con chứa các hành động của mỗi người chơi
 - R : tập hợp reward (điểm thưởng) tương ứng với mỗi action a của lần lượt 2 agents, công thức tính của hàm điểm thưởng được đề cập trước đó trong phần 3.1
- Bên cạnh đó, ta cũng cần xây dựng một chính sách chung cho bài toán SimpleGame này. Có nhiều cách để xây dựng nên chính sách, nhưng trong đây nhóm đưa ra một cấu trúc đơn giản cho SimpleGamePolicy như sau:
 - p : một dictionary ánh xạ từng actions ra các giá trị xác suất (probability) lựa chọn actions đó. Các actions là key và các probabilities là value tương ứng.
 - Constructor cho SimpleGamePolicy: Ta có thể xây dựng nhiều phương thức khởi tạo cho chính sách như truyền vào đối số p là 1 Generator hay Dictionary; từ 1 actions a^i nào đó hoặc là từ 1 chính sách ban đầu π để tính toán xác suất ứng với action của nó.

3.2. Phương pháp giải quyết

a. Thực nghiệm 1: Tính toán cho trường hợp 2 người chơi là 2 máy

a.i. Mô hình lựa chọn

- Ở đây ta lựa chọn mô hình để giải quyết là **Iterated Best Response**
- Cấu trúc:
 - k_max : số lần lặp của giải thuật

- π : chính sách ban đầu

a.ii. Lý do

- Mặc dù trong tài liệu có đề cập đến mô hình **NashEquilibrium** cũng như thuật toán kèm theo để tính ra cân bằng Nash của trò chơi, nhưng việc thực hiện tính toán có thể tốn kém trong việc thực thi, nên ta dùng một cách tiếp cận thay thế khác là áp dụng việc lặp đi lặp lại các phản hồi tốt nhất (best responses) trong trò chơi lặp lại này. Và trong mô hình **Iterated Best Response**, ta sẽ xoay vòng ngẫu nhiên giữa các agents và lần lượt giải quyết các chính sách phản hồi tốt nhất của từng agents đó. Với một số lần lặp lớn, quá trình này có thể hội tụ về trạng thái cân bằng Nash.

a.iii. Phương pháp

- Dựa trên hàm đánh giá là **best_response**
- Best response của 1 agent i đối với chính sách của các agent π^{-i} khác là chính sách π^i thỏa:

$$U^i(\pi^i, \pi^{-i}) \geq U^i(\pi^{i'}, \pi^{-i}), \forall \pi^{i'} \neq \pi^i$$

- Nói cách khác, best response của 1 agent là chính sách khi mà không khuyến khích agent đó thay đổi chính sách của mình so với bộ chính sách cố định của các agent khác. Có thể có đồng thời nhiều best response khác nhau.
- Nếu ta giới hạn bản thân trong các chính sách xác định (deterministic policies), khi đó phản hồi xác định tốt nhất đối với các chính sách của đối thủ sẽ dễ dàng tính toán được ra. Ta chỉ cần lặp lại tất cả các actions của agent i và trả về action tối đa hóa lợi ích nhận được:

$$\operatorname{argmax} U^i(a^i, \pi^{-i}), \text{ với } a^i \in \mathcal{A}^i$$

a.iv. Phương hướng xây dựng

- **Iterated Best Response** sẽ duyệt qua hết tất cả các agent và áp dụng best response của nó so với các agent khác. Thuật toán bắt đầu với chính sách được khởi tạo ban đầu và kết thúc sau k_{\max} lần lặp. Để thuận tiện cho việc xây dựng, ta tạo một constructor có tham số đầu vào là SimpleGame trong đó chính sách ban đầu của mỗi agent thực hiện hành động nào được chọn một cách ngẫu nhiên và đồng nhất với nhau.

b. Thực nghiệm 2: Tính toán cho trường hợp 2 người chơi là con người

b.i. Mô hình lựa chọn

- Ở đây ta lựa chọn mô hình để giải quyết là **Hierarchical Softmax**
- Cấu trúc:

- λ : độ chính xác (giá trị càng cao thì việc đưa ra quyết định của con người sẽ càng chuẩn xác hơn)
- k : cấp bậc
- π : chính sách ban đầu

b.ii. Lý do

- Khách quan: Một lĩnh vực được gọi là lý thuyết trò chơi hành vi nhằm mục đích mô hình hóa với các tác nhân tương ứng là con người. Khi xây dựng các hệ thống ra quyết định phải tương tác với con người, việc tính toán cân bằng Nash không phải lúc nào cũng hữu ích. Con người thường không chơi theo chiến lược cân bằng Nash. Có nhiều lý do cho việc đó:
 - Thứ nhất, có thể không rõ nên áp dụng điểm cân bằng nào nếu có nhiều điểm cân bằng khác nhau trong trò chơi
 - Thứ hai, đối với các trò chơi chỉ có một điểm cân bằng, con người khó có thể tính toán được điểm cân bằng Nash này vì những lý do hạn chế về mặt nhận thức
 - Thứ ba, ngay cả khi con người tính toán ra điểm cân bằng Nash, họ có thể nghi ngờ đối thủ có thể thực hiện được phép tính đó hay không
- Chủ quan: Dạng bài toán lý thuyết trò chơi hành vi là khá hay trong lĩnh vực này, khi mà nhân tố con người đóng vai trò quyết định đến kết quả của bài toán. Gần như không biết được xác suất việc con người thực hiện các hành động như thế nào nên việc đề ra một giải thuật cho nó khá khó khăn, có thể phải tổng hợp từ một lượng dữ liệu lớn từ thực tế để đưa ra một đánh giá gần đúng nhất có thể. Trong các mô hình và phương pháp nhóm tìm hiểu được trong sách, thì **Hierarchical Softmax** là một cách tiếp cận dễ hiểu và dễ dàng để cài đặt. Phương pháp tiếp cận này mô hình hóa hợp lý độ sâu của agent theo cấp bậc $k \geq 0$. Một agent cấp k sẽ chọn các hành động tiếp theo theo chiến lược cấp $k-1$ trước đó

b.iii. Phương pháp

- Dựa trên hàm đánh giá là **softmax response**
- Ta dùng softmax response để mô hình hóa cách mà agent i lựa chọn hành động tương ứng. Như đã đề cập trước đó, con người thường không có xu hướng tối ưu hóa hoàn toàn lợi ích mong đợi. Nên nguyên tắc cơ bản của mô hình softmax response là các agent (thường là con người) có nhiều khả năng mắc lỗi trong quá trình tối ưu hóa hơn khi những lỗi đó ít gây tổn kém hơn. Với độ chính xác $\lambda \geq 0$, mô hình này lựa chọn action a^i theo công thức:

$$\pi^i(a^i) \propto \exp(\lambda U^i(a^i, \pi^{-i}))$$

- Khi $\lambda \rightarrow 0$, agent sẽ không nhận thấy sự khác biệt trong lợi ích họ nhận được, và do đó sẽ lựa chọn các hành động một cách ngẫu nhiên và đồng nhất. Ngược lại, khi $\lambda \rightarrow \infty$, chính sách sẽ hội tụ về một best response có thể xác định được. Tham số λ có thể được học từ các dữ liệu trong thực tế, ví dụ như ước tính khả năng xảy ra tối đa của hành động...

b.iv. Phương hướng xây dựng

- Mô hình **Hierarchical Softmax** với độ chính xác λ và cấp bậc k . Mặc định, nó sẽ bắt đầu với một chính sách chung ban đầu chỉ định xác suất đồng nhất cho mọi hành động riêng lẻ. Lúc đầu, các agent cấp 0 sẽ thực hiện các hành động một cách ngẫu nhiên. Đến lần tiếp theo, agent cấp 1 giả định những đối thủ khác áp dụng chiến lược cấp 0 và sẽ lựa chọn hành động theo softmax response với độ chính xác λ . Cứ như thế, agent cấp k sẽ chọn lựa các hành động theo mô hình softmax của những người chơi khác ở cấp độ $k - 1$.

3.3. Code

- Trong source code, nhóm đã có comment giải thích về cách chạy, phương pháp cũng như mục tiêu, kết quả của những hàm, dòng lệnh. Trong báo cáo này, nhóm sẽ mô tả chung về luồng đi chính, cũng như một số hàm quan trọng.

a. Các hàm quan trọng

- function **reward(simpleGame::Travelers, i::Int, a)**: dùng để tính toán điểm thưởng cho agent i trong game, a là tập hợp các hành động tương ứng của agent đó. Cụ thể, công thức tính toán điểm thưởng r cho agent i dựa trên bảng sau, trong đó $noti$ là agent còn lại:

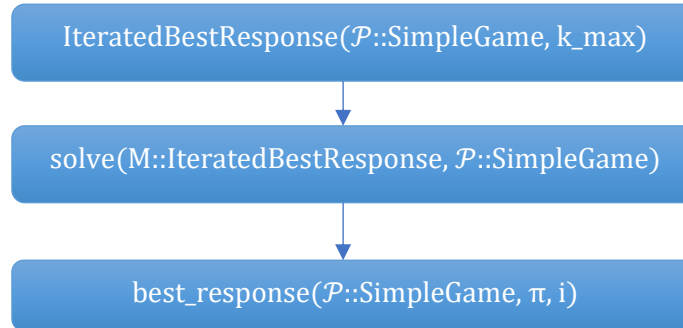
Trường hợp	r
$a[i] = a[noti]$	$a[i]$
$a[i] < a[noti]$	$a[i] + 2$
$a[i] > a[noti]$	$a[noti] - 2$

- function **utility(\mathcal{P} ::SimpleGame, π , i)**: dùng để tính toán ra được lợi ích liên quan đến việc thực hiện chính sách π trong trò chơi \mathcal{P} từ góc nhìn của agent i .
- function **best_response(\mathcal{P} ::SimpleGame, π , i)**: trong trò chơi SimpleGame \mathcal{P} , ta có thể tính toán được best response xác định cho agent i trong khi xem rằng các agent khác cũng đang thực hiện chính sách trong π .

- function **softmax_response**($\mathcal{P}::\text{SimpleGame}$, π , i , λ): đối với trò chơi SimpleGame \mathcal{P} và cho một agent cụ thể i , ta có thể tính toán chính xác được softmax response π^i trong khi các agent khác cũng đang thực hiện các chính trong π . Khác với best response, tính toán này yêu cầu chỉ định tham số cho độ chính xác λ .

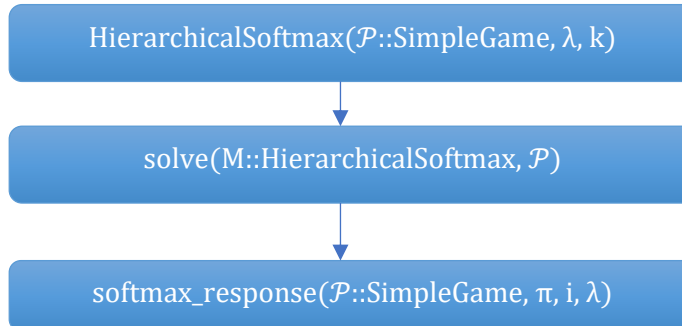
b. Luồng đi chính

- Thực nghiệm 1:



Trước hết, ta khởi tạo 1 đối tượng **IteratedBestResponse** để thử nghiệm với số lần lặp k_{\max} hợp lý (thường > 100 để có thể tính toán về cân bằng Nash một cách chính xác). Tiếp đó ta dùng hàm **solve** để thực hiện giải thuật, trong đó từng vòng lặp sẽ gọi đến hàm **best_response** để liên tục cập nhật lại chính sách mới cho trò chơi.

- Thực nghiệm 2:



Trước hết, ta cũng khởi tạo 1 đối tượng **HierachicalSoftmax** để thử nghiệm với các tham số là độ chính xác λ và cấp bậc k . Tiếp đó ta dùng hàm **solve** để thực hiện giải thuật, trong đó từng vòng lặp sẽ gọi đến hàm **softmax_response** để liên tục cập nhật lại chính sách mới cho trò chơi.

3.4. Phân tích

a. Thực nghiệm 1

a.i. Kết quả thử nghiệm

Ở đây sẽ đưa ra các thử nghiệm cho từng trường hợp với tham số k_{\max} khác nhau và quan sát kết quả nhận được, kết quả trả về là 1 Dictionary có dạng $\text{Dict}(\text{action} \Rightarrow \text{probability})$

k_{\max}	Agent 1	Agent 2
0	$\text{Dict}(a \Rightarrow 0.(01)),$ với $a \in [2,100]$	$\text{Dict}(a \Rightarrow 0.(01)),$ với $a \in [2,100]$
10	$\text{Dict}(87 \Rightarrow 1.0)$	$\text{Dict}(87 \Rightarrow 1.0)$
50	$\text{Dict}(47 \Rightarrow 1.0)$	$\text{Dict}(47 \Rightarrow 1.0)$
80	$\text{Dict}(17 \Rightarrow 1.0)$	$\text{Dict}(17 \Rightarrow 1.0)$
100	$\text{Dict}(2 \Rightarrow 1.0)$	$\text{Dict}(2 \Rightarrow 1.0)$
1000	$\text{Dict}(2 \Rightarrow 1.0)$	$\text{Dict}(2 \Rightarrow 1.0)$
≥ 10000	$\text{Dict}(2 \Rightarrow 1.0)$	$\text{Dict}(2 \Rightarrow 1.0)$

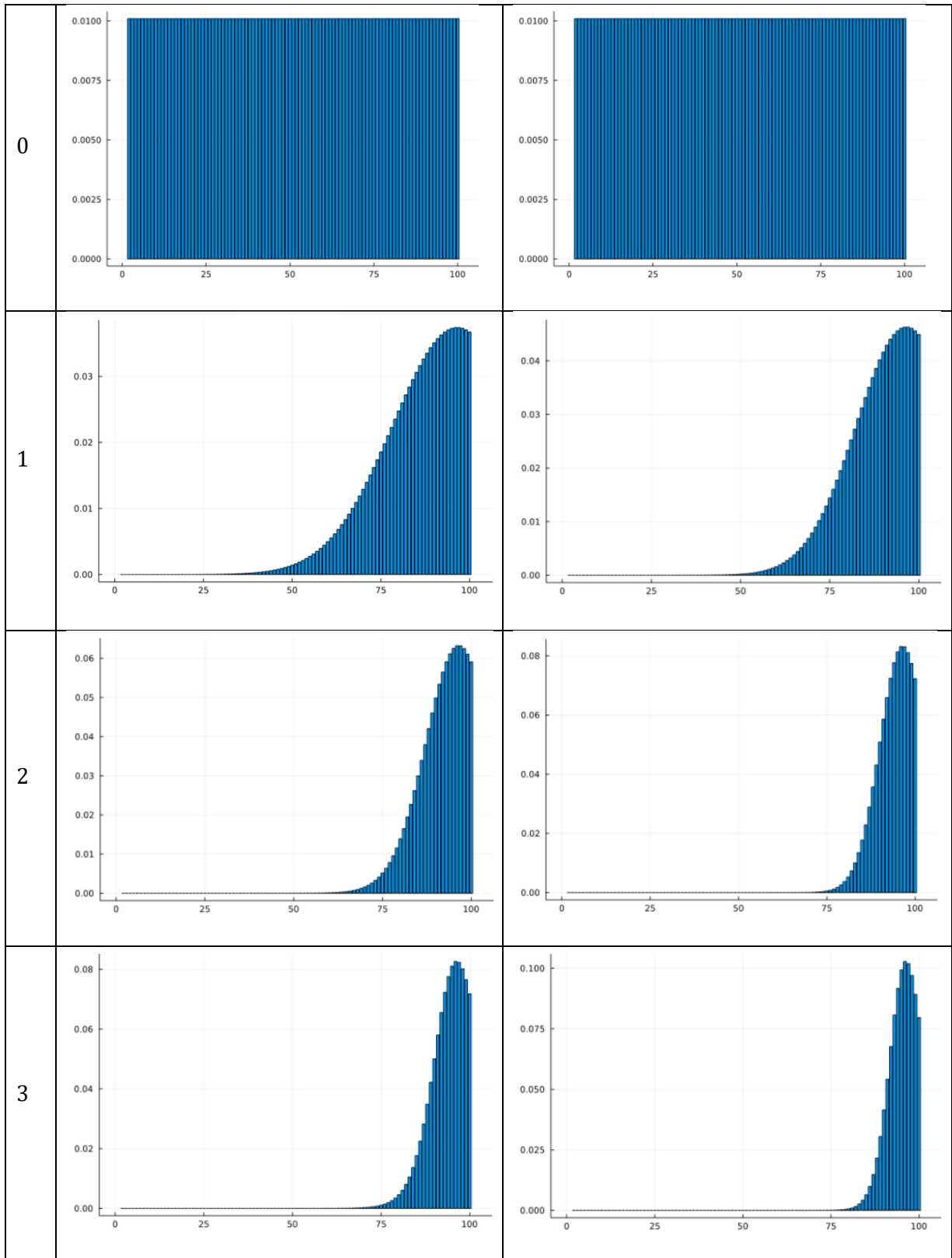
a.ii. Ý nghĩa

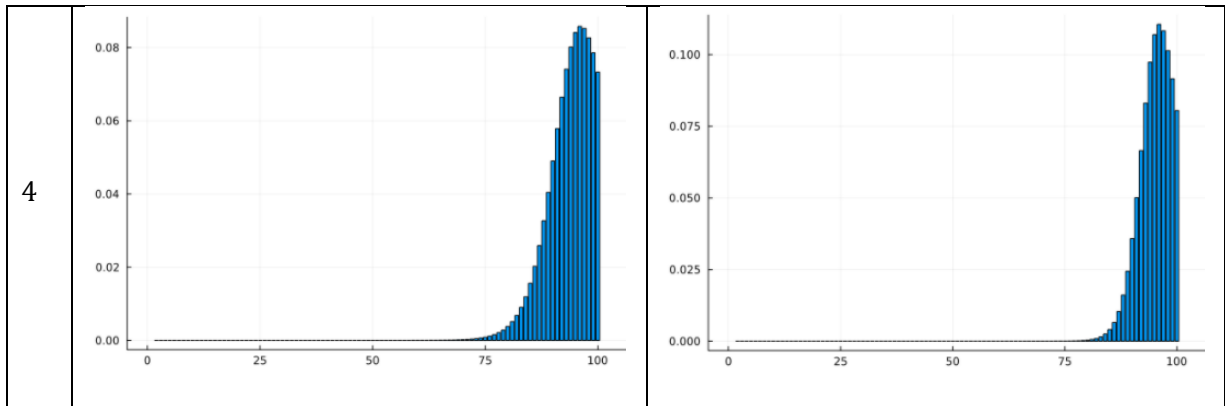
- Kết quả trả về trong từng trường hợp đều giống nhau về hành động cũng như xác suất lựa chọn hành động đó cho cả 2 agent.
- Với $k_{\max} = 0$, xác suất lựa chọn các action là ngẫu nhiên và đồng nhất với nhau $= \frac{1}{99}$
- Với $k_{\max} > 0$, kết quả luôn chỉ trả về Dictionary 1 ($\text{action} \Rightarrow \text{probability}$) duy nhất, tức là tại lần chơi k_{\max} đó 2 agent luôn đánh ra kết quả là action đó với xác suất 100%.
- Khi k_{\max} tăng dần, kết quả lựa chọn action của cả 2 agent ngày càng giảm dần và chạm tới mức thấp nhất là 2, đó cũng chính là cân bằng Nash của bài toán Traveler's Dilemma.

b. Thử nghiệm 2**b.i. Kết quả thử nghiệm**

Ở đây sẽ đưa ra các thử nghiệm cho từng trường hợp với các tham số λ và k khác nhau và quan sát kết quả nhận được, kết quả trả về sẽ được visualize bằng thư viện Plots minh họa cho xác suất lựa chọn hành động của hai agent.

k	$\lambda = 0.3$	$\lambda = 0.5$
-----	-----------------	-----------------





b.ii. Ý nghĩa

- Kết quả trả về trong từng trường hợp đều giống nhau về hành động cũng như xác suất lựa chọn hành động đó cho cả 2 agent.
- Với $k_{\max} = 0$, xác suất lựa chọn các action là ngẫu nhiên và đồng nhất với nhau $= \frac{1}{99}$
- Khi k_{\max} tăng dần, kết quả lựa chọn action của cả 2 agent có xu hướng tăng dần về phía 97 đến 100, đó cũng chính là dự đoán ban đầu được đề ra cho dạng toán lý thuyết trò chơi hành vi của Traveler's Dilemma với các agent là con người.

4. Đánh giá kết quả

4.1. Thực nghiệm 1

- Mức độ giải quyết vấn đề: Bằng cách áp dụng mô hình Iterated Best Response, ta đã chỉ ra được lý do tại sao cân bằng Nash của bài toán đạt tại \$2, khi mà ta muốn tối đa hóa lợi ích đạt được so với đối thủ.
- Độ tốt của kết quả:
 - Mô hình có thể tính toán ra chính xác được cân bằng Nash với số lần lặp k_{\max} đủ lớn.
 - Kết quả tính toán ra khá nhanh so với các thuật toán khác, với số lần lặp lớn (> 10000) Iterated Best Response vẫn tính toán ra được trong thời gian ngắn.
- Bất cập:
 - Thuật toán Iterated Best Response nói rằng quá trình thực thi không luôn luôn hội tụ về cân bằng Nash, tùy từng bài toán áp dụng mà có đảm bảo việc đó hay không. May thay, trong trường hợp bài toán ta đang xét thì việc hội tụ này là khả thi.

- Bên cạnh đó, thuật toán cũng không biết cụ thể nên thực hiện tối thiểu bao nhiêu lần lặp để tính ra đúng chính xác cân bằng Nash. Đa phần trong nhiều bài toán, người thực hiện phải quan sát các chu kỳ vòng lặp để tìm ra điều đó.
- k_{\max} cần phải đạt giá trị tầm 100 trở lên thì mới tính ra được chính xác cân bằng Nash.

4.2. Thực nghiệm 2

- Mức độ giải quyết vấn đề: Bằng cách áp dụng mô hình Hierarchical Softmax, ta cũng chỉ ra được tại sao trong bài toán lý thuyết trò chơi hành vi thì con người có xu hướng chọn giá trị trong khoảng \$97 đến \$100, bởi vì nhân tố agent là con người với độ chính xác λ sẽ đóng vai trò quyết định đến kết quả phủ quyết của bài toán.
- Độ tốt của kết quả:
 - Mô hình cho thấy một cái nhìn tổng quan các lựa chọn hành động của agent. Bằng cách visualize hình ảnh, ta nhận thấy xác suất lựa chọn hành động của người chơi có sự biến chuyển khác nhau.
 - Cấp độ k cho bài toán không cần quá lớn, từ $k \geq 10$ thì các kết quả sau đó đều gần như là giống nhau. Vì vậy nên ta có thể thực nghiệm được nhiều trường hợp λ khác nhau mà không phải chạy quá lâu.
- Bất cập:
 - Thuật toán Hierarchical Softmax chạy khá lâu, gần như với $k > 1000$ thì phải đợi trong một khoảng thời gian mới đạt được kết quả.
 - Tham số λ có thể xem như quyết định đến toàn bộ thuật toán. Tuy nhiên gần như không biết chính xác giá trị này là bao nhiêu trong lý thuyết trò chơi hành vi. Ta chỉ có thể tìm hiểu từ tập dữ liệu thực tế cũng như mối liên hệ giữa λ và k .
 - Ta có thể sử dụng một thuật toán tìm λ và k để tối ưu hóa khả năng thực hiện hành động. Tuy nhiên việc tối ưu hóa này thường không thể thực hiện được bằng cách phân tích. Thay vào đó, ta sẽ áp dụng các phương pháp số học để thực hiện việc này như là dùng cách tiếp cận Bayes.

5. Tài liệu tham khảo

- Sách Algorithm for Decision Making
- <https://vietnambiz.vn/the-tien-thoai-luong-nan-cua-lu-khach-travelers-dilemma-la-gi-noi-dung-lien-quan-20191114174423475.htm>

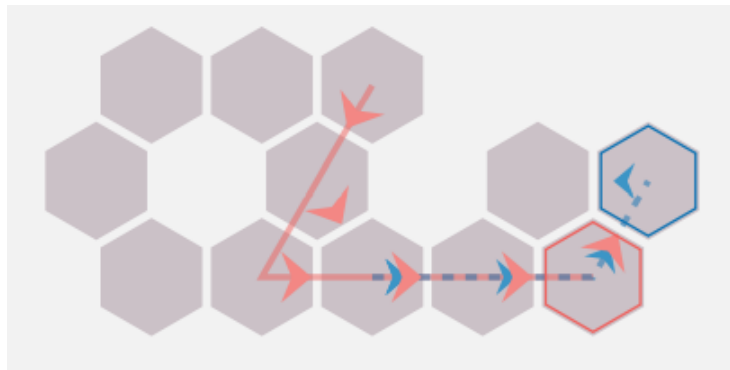
- <https://vietnambiz.vn/can-bang-nash-nash-equilibrium-la-gi-ung-dung-trong-thi-truong-doc-quyen-nhom-20191030140829824.htm>
- <https://github.com/daslerpc/Iterated-Travelers-Dilemma>
- <https://github.com/algorithmsbooks/DecisionMakingProblems.jl>
- <https://algorithmsbooks.github.io/DecisionMakingProblems.jl/>

IV. Predator-Prey Hex World

1. Phát biểu bài toán

1.1. Phát biểu bài toán

- Predator-prey hex world là bài toán được mở rộng từ bài toán hex world, bao gồm nhiều agents tham gia, được chia thành 2 nhóm chính là **predator** (động vật ăn thịt hay kẻ săn mồi) và **prey** (con mồi)
- Kẻ săn mồi cố gắng bắt con mồi càng nhanh càng tốt và ngược lại, con mồi cố gắng tránh khỏi những kẻ săn mồi càng lâu càng tốt. Dưới đây là hình vẽ minh họa bài toán đuổi bắt này trong hex world, predator có màu đỏ và màu xanh là prey. Các agents có thể di chuyển theo 6 hướng của hex cell.



1.2. Luật chơi

- Nếu predator và prey vào chung 1 ô, tức là con mồi đã bị bắt. Khi đó, prey sẽ bị phạt, đồng thời xuất hiện ngẫu nhiên tại một ô khác trong các hex cells. Đương nhiên, predator sẽ nhận được phần thưởng tương ứng khi đã bắt được con mồi. Các agents trong trò chơi di chuyển đồng thời.
- Predator di chuyển có thể nhận được hình phạt đơn vị (-1 điểm chẳng hạn), prey di chuyển có thể không bị trừ tùy vào quy ước bài toán, nhưng khi bị bắt sẽ phạt nặng (ví dụ -100 điểm).

2. Thách thức

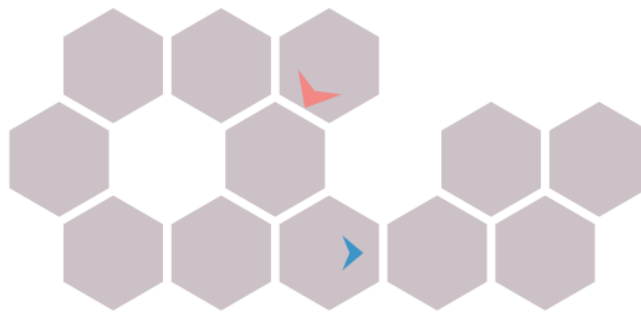
- Giải quyết bài toán cần có kiến thức và tìm hiểu tổng hợp về các nội dung như Hex World, Discrete MDP, Simple Game, việc cài đặt khá phức tạp.
- Bài toán sử dụng Discrete MDP vì xung quanh 1 cell không có đầy đủ 6 cell khác cho 6 actions mà phải kiểm tra các trường hợp có thể đi được hay không, đụng tường...

- Vì các agents di chuyển liên tục nên policy rất phức tạp và dài, việc hiểu và phân tích, mô hình hoá khá khó khăn.
- Việc phân tích và hiểu hết tất cả đoạn code trong tài liệu tham khảo khá là tốn thời gian do có những xử lý phức tạp cô đọng lại bằng 1 dòng dài.
- Ngoài sách ra thì hầu như rất ít tài liệu tham khảo, bài viết phân tích về bài toán này, nhất là cài đặt bằng Julia, một ngôn ngữ mới với cộng đồng support chưa được đông đảo lắm, dẫn đến nhiều bug phải tự nghiên cứu và thử nghiệm bằng tay.

3. Thực nghiệm

3.1. Cấu hình bài toán

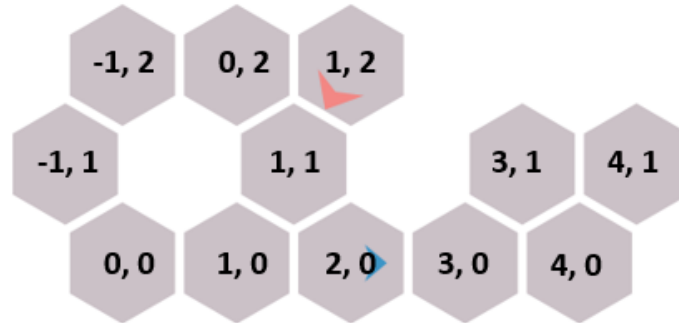
- Bài toán gồm 2 agents: predator và prey di chuyển trong bản đồ hex world như sau



- Trạng thái của mỗi agent là vị trí của nó trong hex map.
- Tại mỗi hex cell, agent có thể di chuyển theo 6 hướng là hướng east, north east, north west, west, south west và south east để đi đến ô liền kề
- Nếu predator và prey đứng ở cùng một hex cell, prey sẽ bị bắt và bị phạt 100 điểm (-100), và predator sẽ được thưởng 10 điểm (+10). Cả hai sẽ bị phạt 1 điểm cho mỗi lần di chuyển (-1), và không được phép di chuyển ra bên ngoài hex map

3.2. Mô hình hóa tính toán

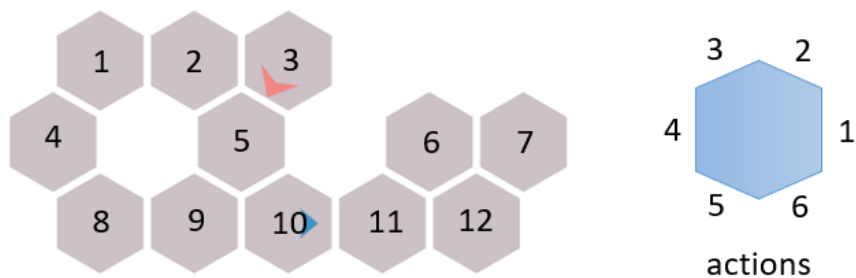
a. Bản đồ Hex-world và cách xác định 1 cell:



- Mô hình hoá bản đồ hex-world: các hex cell được đánh chỉ số theo thứ tự hàng tăng dần từ dưới lên trên và cột tăng dần từ trái qua phải theo hướng hợp với phương ngang một góc 60 độ, lần lượt là :

$(-1, 2), (0, 2), (1, 2), (-1, 1), (1, 1), (3, 1), (4, 1), (0, 0), (1, 0), (2, 0), (3, 0), (4, 0)$ (3.3.1)

- Cũng theo thứ tự này, state s sẽ được đánh số từ trên xuống dưới, từ trái qua phải. Tại một hex cell, ta có 6 action di chuyển lần lượt theo các hướng east, north east, north west, west, south west, south east tương ứng với 1, 2, 3, 4, 5, 6 để đến các ô bên cạnh, cụ thể sẽ mô tả như hình dưới đây:



b. Cấu trúc PredatorPreyHexWorldMG:

Do đặc điểm của hex map chứa các hex cell rời rạc nên ta dùng structure DiscreteMDP để mô tả

Biến	Kiểu dữ liệu	Mô tả
hexes	Vector{Tuple{Int,Int}}	1 mảng chứa tọa độ của 12 hex cell (3.3.1) .
hexWorldDiscreteMDP	DiscreteMDP { + T::Array{Float64,3} + R::Array{Float64,2} + γ ::Float64 }	Sử dụng các thông số sau để hỗ trợ tính toán mảng T, R + Xác suất đi theo hướng dự định là: 0.7 + Chiết khấu γ là 0.9

c. Cấu trúc MG (Markov Game):

Biến	Kiểu dữ liệu	Mô tả
γ	Float64	Chiết khấu. Chọn $\gamma = 0.9$
\mathcal{I}	Vector{Int}	Tập các agents predator: 1 và prey: 2
\mathcal{S}	Vector{ Tuple{Int,Int}}	Tập trạng thái của 2 agents Mỗi phần tử là một tuple thể hiện vị trí 2 ô trong hex map của 2 agents. Ví dụ ô (3, 5) xác định vị trí của predator là 3, còn ô của prey là 5 trên bản đồ hex.
\mathcal{A}	Vector{ Vector{Int}}	Tập các actions của agents Gồm 2 mảng, mỗi mảng là tập 6 phần tử [1, 2, 3, 4, 5, 6], tương ứng với 6 hướng đi của agent)

$T(s, a, s')$	Method	Hàm tính toán khả năng chuyển đổi trạng thái từ s sang s' với action a
$R(s, a)$	Method	Hàm tính điểm thưởng ứng của 2 agents tại trạng thái s (trả về mảng 2 reward)

3.3. Phương pháp giải quyết

a. Phương pháp Fictitious Play:

- Nhóm đã trình bày Fictitious Play ở bài toán Rock-paper-scissors, tuy nhiên đối với dạng toán Markov Game, ta cần phải xem xét trạng thái s hiện tại của agents và khả năng chuyển đổi từ s sang s' (state tiếp theo) thông qua action a , hơn nữa, lúc này mỗi agent i sẽ nhận được reward theo hàm $R^i(s, a)$ của riêng nó. Do bài toán phức tạp hơn và phụ thuộc vào state s nên cách cài đặt cũng sẽ tổng quát hơn.
- Mô phỏng với số iteration là 10, tại mỗi iteration, phương pháp duy trì một maximum-likelihood model thông qua policy của các agents, dùng maximum likelihood estimates theo đổi trạng thái và action được dùng của mỗi agent. Từ ước lượng đó, tính toán đưa ra chiến lược tốt nhất có thể (best response).
- Maximum likelihood model sẽ lưu lại số lần agent j thực hiện action a^j tại trạng thái s , lưu trong Dictionary $N(j, s, a^j)$. Chẳng hạn:

$$(2, (5,10), 6) \Rightarrow 10$$

nghĩa là agent 2 (prey) sử dụng action 6 tại trạng thái s (5,10) là 10 lần

- Lúc này, ta tính best response dựa trên việc giả sử rằng agent j theo một stochastic policy phụ thuộc trạng thái như sau:

$$\pi^j(a^j|s) \propto N(j, s, a^j) \quad (4.3.2)$$

b. Fictitious Play cho bài toán có cấu trúc như sau:

Biến	Kiểu dữ liệu	Mô tả
\mathcal{P}	MG	Được đề cập ở phần 3.2.c
i	Int	Agent index (1 hoặc 2)

Q _i	Dict	<p>Hàm ước tính giá trị state-action cho agent i, có dạng Dict((s, a) ⇒ value)</p> <p>Tính bằng reward tại state s + state value estimate cho state s' tiếp theo khi chọn thực hiện action a</p> <p>Khởi tạo: "key" ⇒ reward của agent i tại "key"</p>
N _i	Dict	<p>Hàm đếm state-action có dạng Dict((j, a, a_j) ⇒ count) lưu lại số lần agent j thực hiện action a^j tại trạng thái s</p> <p>Khởi tạo: mọi keys đều có value bằng 1</p>

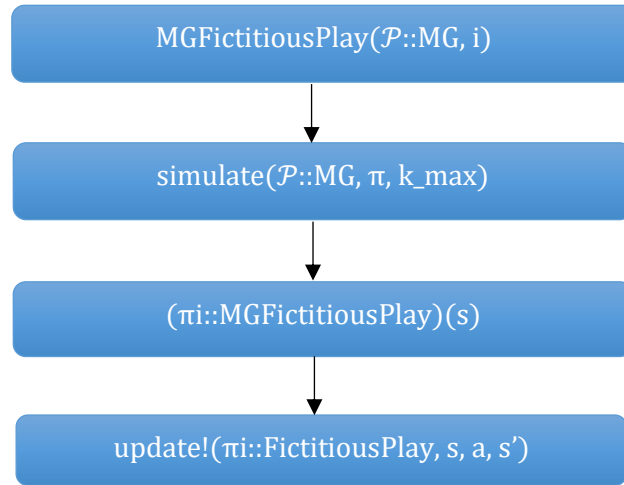
c. Lý do lựa chọn phương pháp:

- Dễ theo dõi, mô phỏng được quá trình chơi từng ván, khả năng ra quyết định (policy), trạng thái và sự đuổi bắt của 2 agents như thế nào qua từng lần lặp. Từ đó, có thể visualize và đánh giá được độ hiệu quả của phương pháp.
- Fictitious Play là phương pháp dựa trên học tập (learning-based approach), vừa mô phỏng chơi game rồi học và thích ứng với cách chơi của đối thủ, rất phù hợp với bài toán này. Khi đó, predator học được chiến thuật lẩn trốn của prey, từ đó tìm policy phù hợp để đuổi và ngược lại.

3.4. Code

- Trong source code, nhóm đã có comment giải thích về cách chạy, phương pháp cũng như mục tiêu, kết quả của những hàm, dòng lệnh. Trong báo cáo này, nhóm sẽ mô tả chung về luồng đi chính, cũng như một số hàm quan trọng.

a. Luồng đi chính:



- **MGFictitiousPlay($\mathcal{P}::\text{MG}$, i)**: khởi tạo MGFictitiousPlay cho agent thứ i tương ứng trong Markov Game \mathcal{P} . Quy ước ban đầu, mỗi state-action counts N_i được khởi tạo với count là 1 và state-action value estimate Q_i sẽ gán bằng reward agent i nhận được tại trạng thái s $R(s, a)[i]$
- Function **simulate($\mathcal{P}::\text{MG}$, π , k_max, b)**: mô phỏng việc chơi Markov Game \mathcal{P} trong k_max iterations với policy π chứa 2 MGFictitiousPlay tương ứng với 2 agents là predator và prey.
 - Lúc khởi đầu, random state (vị trí hex cell) của 2 agents, sau đó lặp k_max lần.
 - Ở mỗi lần lặp, gọi đến **($\pi::\text{MGFictitiousPlay}$)(s)** để tính toán trả về được **SimpleGamePolicy** với action được lựa chọn có Q-value lớn nhất, nghĩa là đi theo action này thì khả năng nhận được reward cao nhất theo góc nhìn của mỗi agent. Sau đó, gọi **($\pi::\text{SimpleGamePolicy}$)()** return random action dựa trên xác suất của action ai của 2 agents.
 - Tiếp theo random state tiếp theo s' bằng cách gọi hàm **randstep($\mathcal{P}::\text{MG}$, s, a)** để random dựa trên xác suất transition từ s sang s', trả về s' và reward nhận được tại state s.
 - Tiếp tục gọi đến **update!($\pi::\text{MGFictitiousPlay}$, s, a, s')** để cập nhật lại N_i và Q_i cho policy π_i của agent i
 - Cuối cùng, gán s' cho s để cập nhật state hiện tại và tiếp tục vòng lặp kế.

b. Các hàm hỗ trợ:

- Function ($\pi_i::\text{MGFictitiousPlay}$)(s): tìm ra best response a_i từ trạng thái s dựa trên policy π_i của agent i , trả về SimpleGamePolicy với xác suất của action a_i là 1

Dùng 2 hàm hỗ trợ là $U(s, \pi)$ và $Q(s, \pi)$ để tính toán $Q(a_i)$: state value estimate agent i đạt được khi chọn thực hiện action a_i tại state s

$$U(s, \pi) = \sum(\pi_i.Q_i[s, a] * \text{probability}(\mathcal{P}, s, \pi, a) \text{ for } a \text{ in } \text{joint}(\mathcal{A}))$$

$$Q(s, \pi) = \text{reward}(\mathcal{P}, s, \pi, i) + \gamma * \sum(\text{transition}(\mathcal{P}, s, \pi, s') * U(s', \pi) \text{ for } s' \text{ in } \mathcal{S})$$

Trong đó:

- + $\text{probability}(\mathcal{P}, s, \pi, a)$: khả năng 2 agents sẽ chọn action a tại state s
- + $\text{transition}(\mathcal{P}::\text{MG}, s, \pi, s')$: tổng khả năng 2 agents sẽ chuyển đổi từ s sang s'

Sau đó xét trên toàn bộ action space của agent i , dùng **argmax** để chọn ra action a_i có $Q(a_i)$ tốt nhất làm policy.

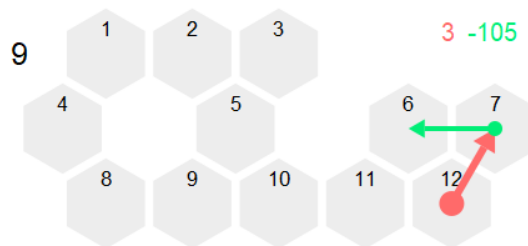
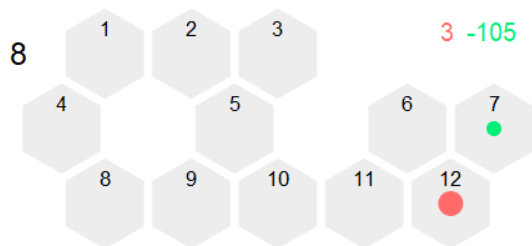
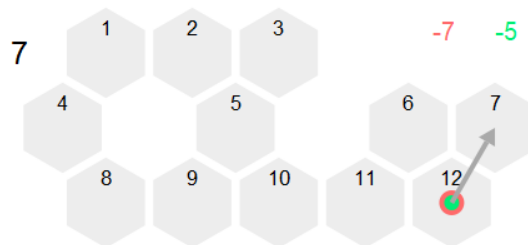
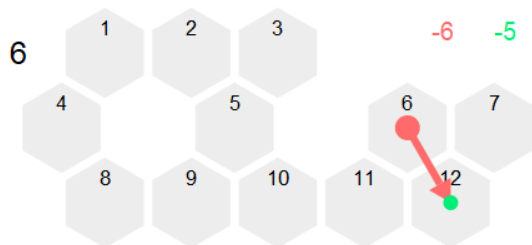
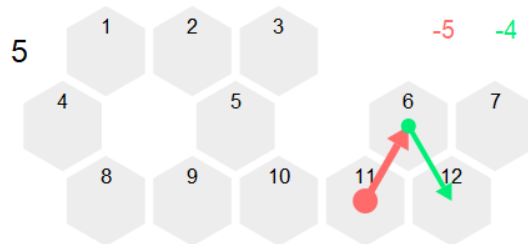
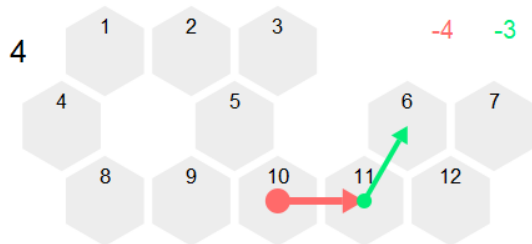
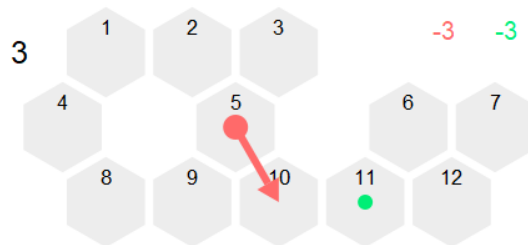
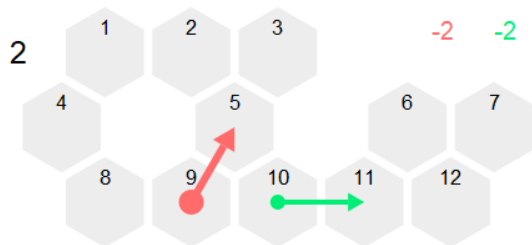
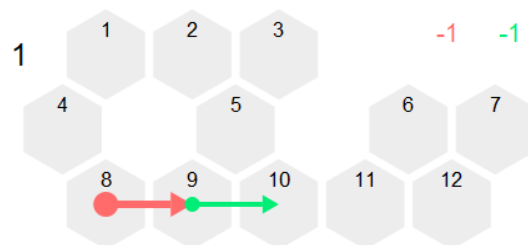
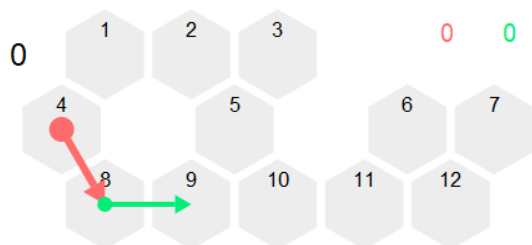
3.5. Phân tích

a. Kết quả đạt được

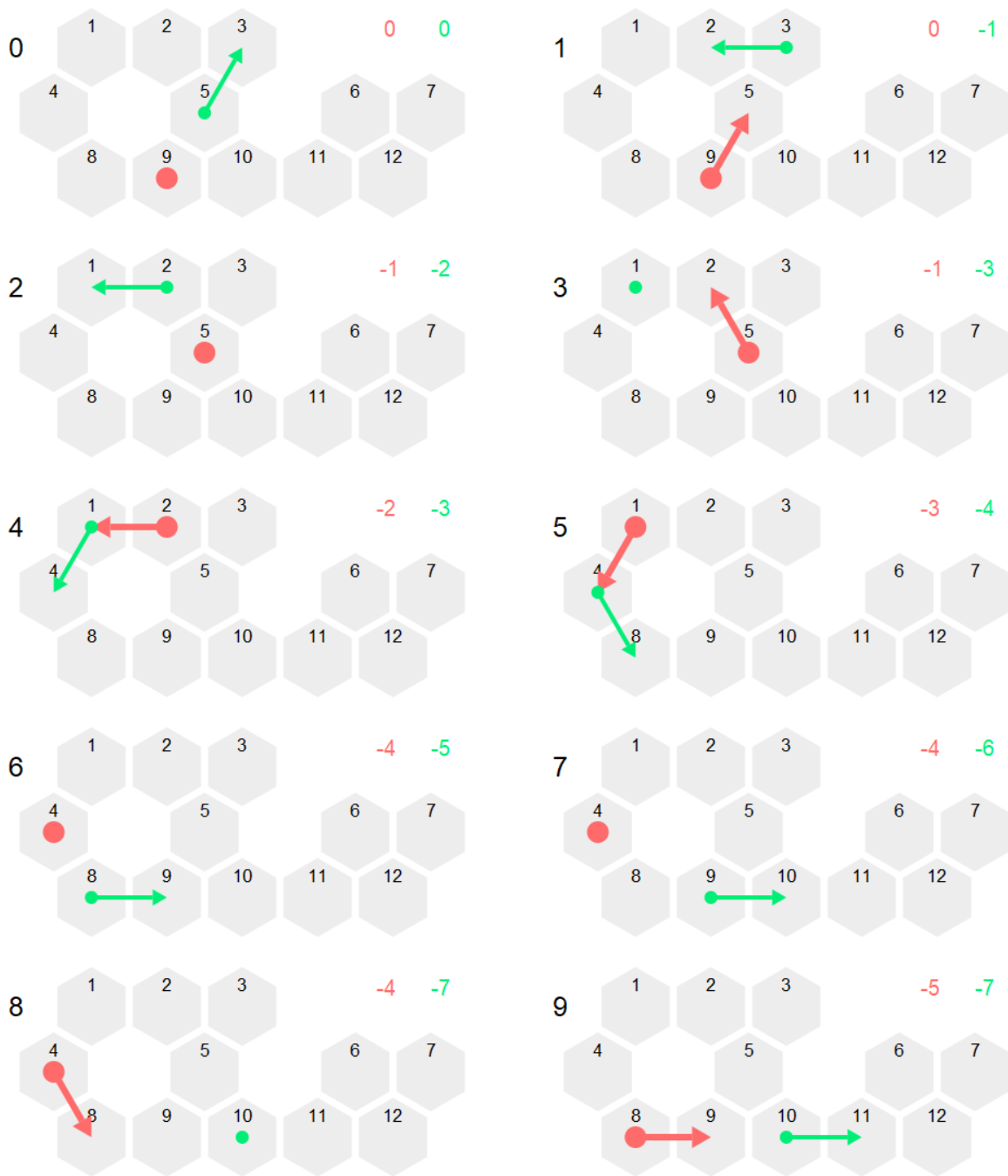
a.i. Giới hạn $k_{\text{max}} \leq 10$

- Do giới hạn về kích thước **Drawing**, nhóm chỉ có thể vẽ được tối đa 10 iterations.
- Kết quả thực nghiệm sẽ được visualize với quy ước như sau:
 - Hình tròn: vị trí của agent
 - Mũi tên: hướng agent sẽ di chuyển
 - Predator: màu **đỏ**
 - Prey: màu **xanh**
 - Hex cell: màu **xám**
 - State: màu **đen**, nằm trong mỗi hex cell
 - Iteration: góc trên bên trái mỗi hex map
 - Reward: góc trên bên phải mỗi hex map
 - Tại iteration 0: cả 2 agents sẽ được random 2 vị trí không trùng nhau và cùng nhận reward 0 – 0
 - Khi prey bị bắt, nó sẽ xuất hiện random tại một hex cell bất kì, thể hiện bằng mũi tên màu **xám**

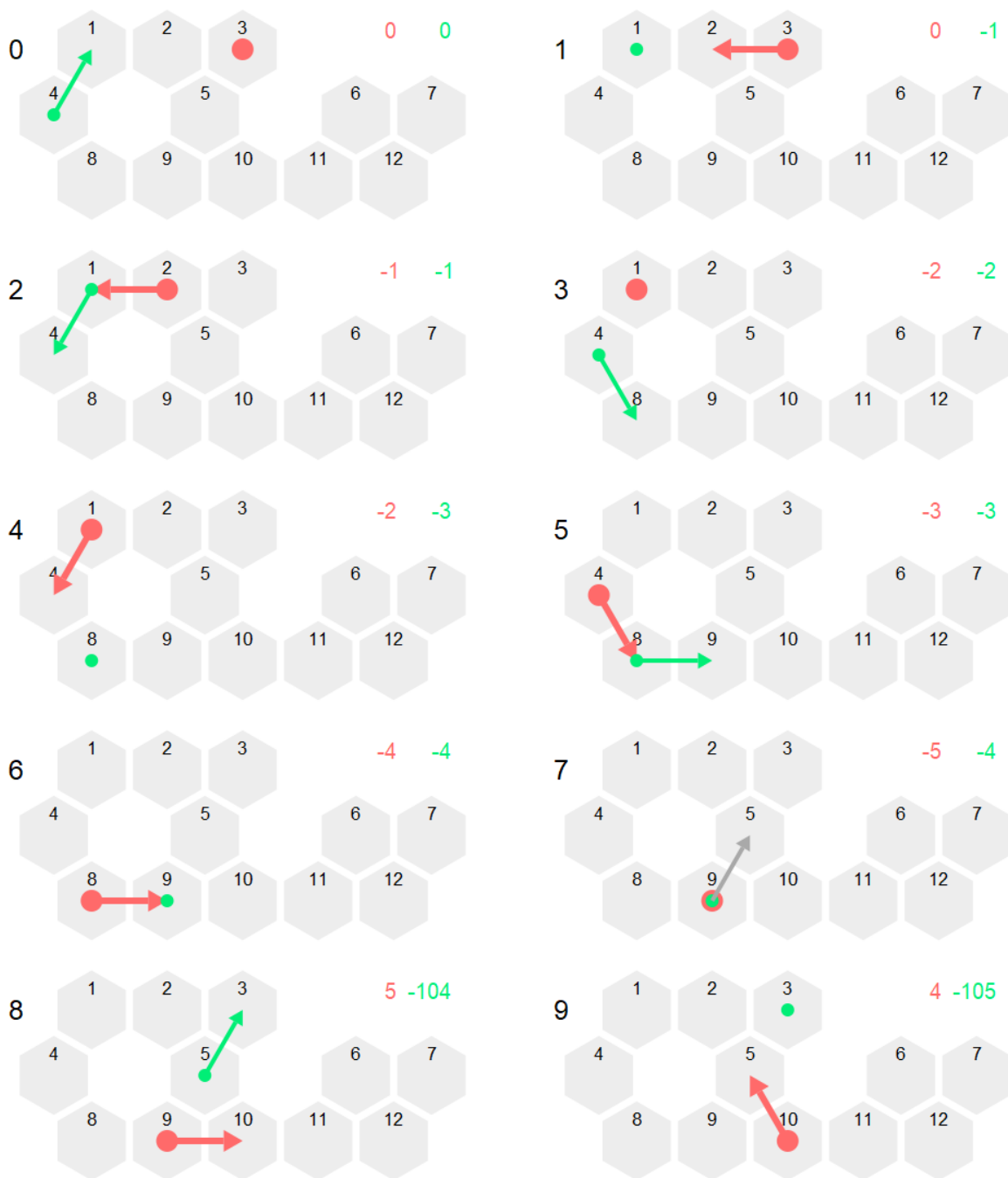
- Kết quả 1: $k_{\max} = 10$, prey bị bắt :



- Kết quả 2: $k_{\max} = 10$, prey không bị bắt :



- Kết quả 3: $k_{\max} = 10$, prey bị bắt :

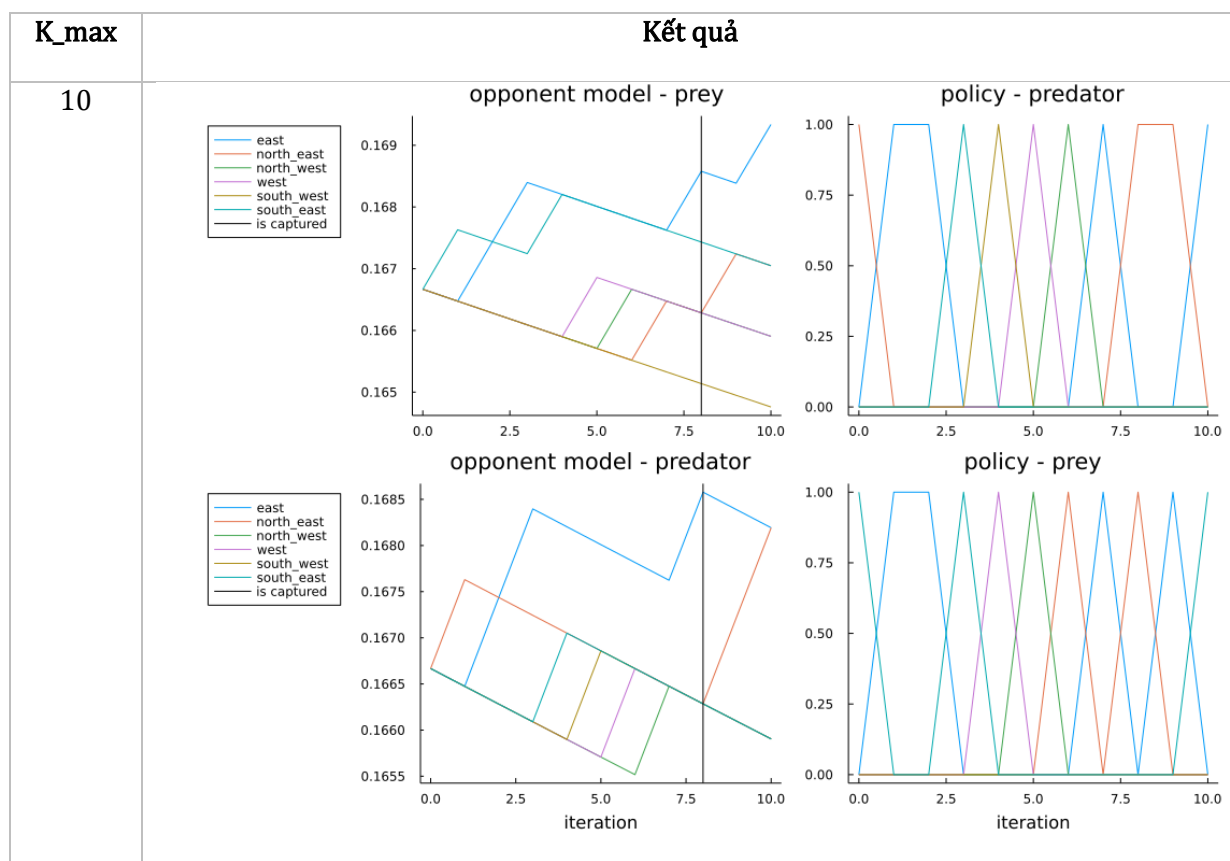


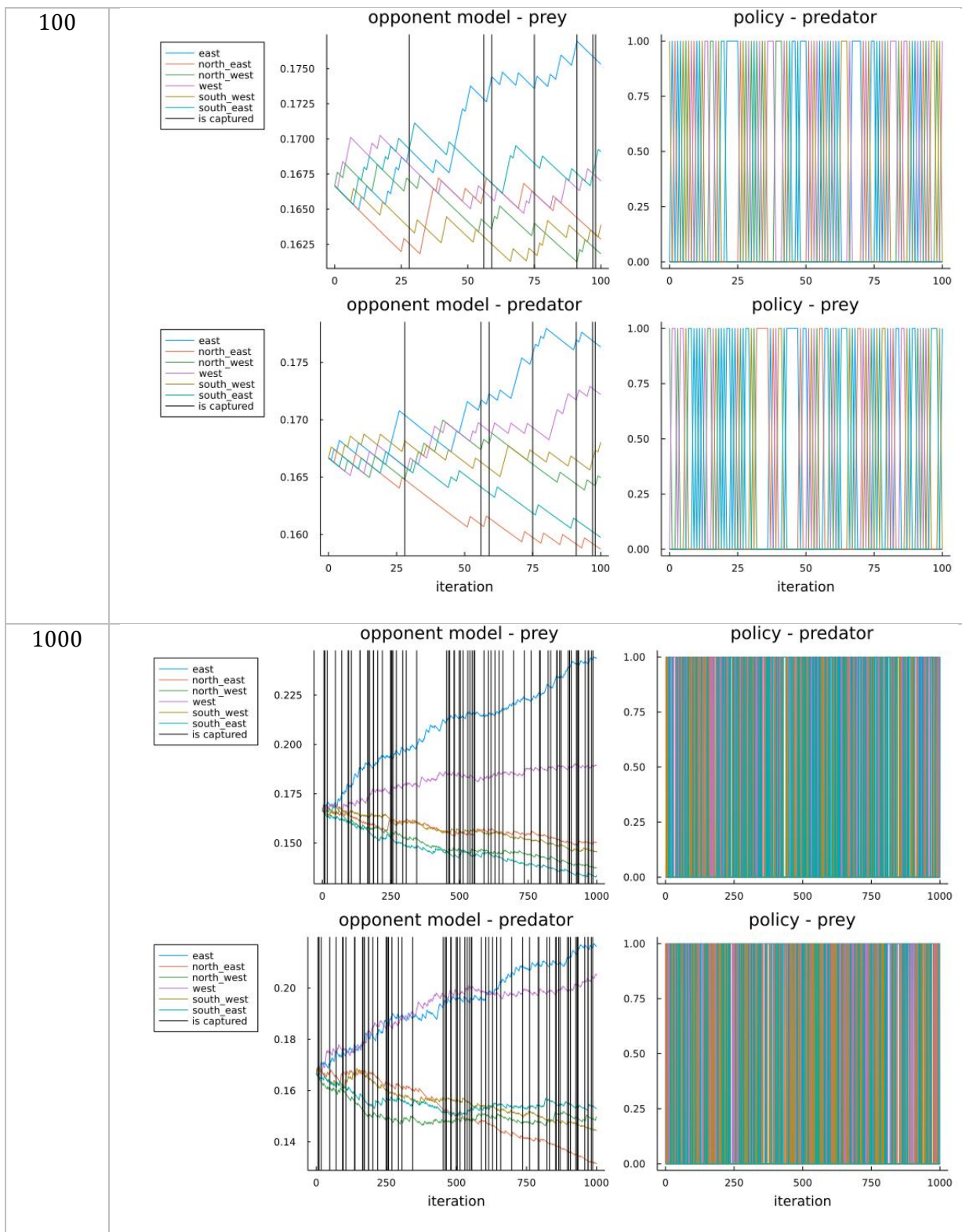
a.ii. Giới hạn $k_{\max} \leq 1000$

Khi $k_{\max} \geq 10000$, thời gian chạy rất lâu. Do đó nhóm chỉ visualize khi $k \leq 1000$

Opponent model được xây dựng từ việc đếm số lần agent thực hiện action ai thuộc [1:6]. Từ đó tìm ra xác suất từng action

Policy là action tốt nhất dưới góc nhìn của chính agent đó trong mỗi iteration





b. Phân tích kết quả

- Từ kết quả visualize ở mục a.i, ta thấy rằng prey đã biết cách chạy trốn khỏi predator, predator cũng biết cách đuổi theo prey. Cuộc truy đuổi diễn ra khá gay cấn
- Tuy nhiên, ở kết quả 3 mục a.i, tại iteration 6, rõ ràng prey có thể tiếp tục chạy trốn khỏi predator nhưng prey lại chọn đứng yên, và nó đã bị bắt ở iteration 7. Qua quá trình chạy thực nghiệm nhiều lần, nhóm cho rằng trường hợp tương tự như vậy khá ít. Bằng chứng là ở mục a.ii, ta thấy rằng số lần prey bị bắt là không nhiều
- Cách hành xử của các agent tại một state s ở các thời điểm khác nhau là không giống nhau, điều này khiến đối thủ không dễ dàng đoán được ý đồ của mình
- Qua kết quả visualize ở mục a.ii, prey và predator có xu hướng di chuyển về phía đông nhiều hơn, prey học được rằng nếu nó di chuyển đến góc phía đông và đợi, nó có thể trốn thoát khỏi predator ngay lập tức khi predator nhảy đến vị trí của nó, từ đó thoát được và chạy đến phía khác của bản đồ. Predator cũng biết được chính xác hướng trốn của prey, nên nó đuổi theo về phía này dẫn đến 2 model có xu hướng di chuyển như hình.
- Xác suất của mỗi action của cả 2 model dao động quanh $\frac{1}{6}$ (0.166) ở $k_{\max} = 10$ và $k_{\max} = 100$. Khi số iteration lớn (1000), prey có kinh nghiệm và xu hướng chạy về phía đông nhiều hơn, xác suất mở rộng lên đến 0.225.
- Qua các lần chạy thực nghiệm và quan sát, nhóm thấy rằng prey thường bị bắt ở khu vực phía đông. Nguyên nhân của việc này được đề cập ở phần 4.2
- Tần suất prey bị bắt qua các iteration là không đều, nguyên nhân của điều này xuất phát từ việc random vị trí xuất hiện cho prey sau khi bị bắt vì hex map này không có cấu trúc đồng đều và đối xứng.

4. Đánh giá kết quả

4.1. Điểm tốt:

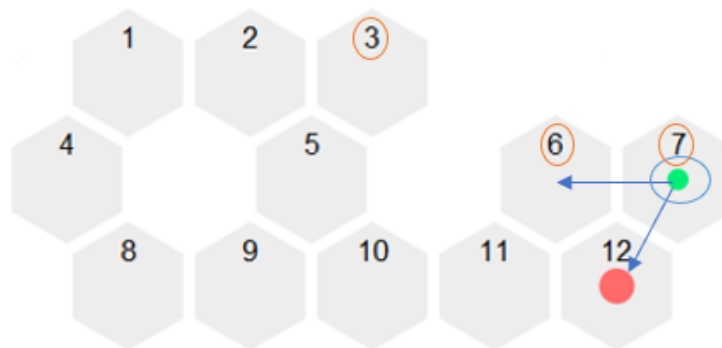
- Mô hình cho kết quả khá khả quan, predator đã biết cách săn prey, prey cũng cố gắng học được chiến lược phù hợp trốn thoát khỏi predator. Cả 2 qua quá trình mô phỏng dài đã học được chính sách và đọc vị trí của nhau, từ đó cuộc đuổi bắt trở nên khá hấp dẫn.
- Chiến lược của đối thủ là không dễ đoán cho cả prey và predator ở những tình huống nguy hiểm (đề cập ở mục 4.2), điều này rất quan trọng và thuật toán đã đáp ứng được điều kiện này

- Từ chính sách rất dài và rối (đôi khi tràn cả console), code kết hợp khá nhiều khái niệm như Hex World, DiscreteMDP, Simple Game, FictitiousPlay, Markov Game.... Nhưng nhóm đã cùng nhau phân tích từng dòng và hàm code để hiểu rõ phương pháp, từ đó nghiên cứu visualize kết quả.
- Chọn được phương pháp phù hợp để áp dụng cho bài toán là MG Fictitious Play, mô phỏng quá trình thực tế đuổi bắt của cả 2 agents, cập nhật policy dựa trên state transition và action của đối thủ, duy trì một maximum-likelihood model dựa trên policy đối thủ. Phương pháp giúp dễ dàng theo dõi state, lựa chọn action nào, state kế tiếp... mô phỏng được toàn bộ quá trình chơi, từ đó có thể visualize dễ dàng.
- Vị trí xuất hiện sau mỗi lần bị bắt là random tạo nên sự công bằng và hấp dẫn cho trò chơi.
- Visualization sáng sủa, dễ nhìn, dễ hiểu và rõ ràng. Có cả visualize cụ thể một quá trình đuổi bắt thành công trong hex map và biểu đồ phân tích mối quan hệ opponent model – policy học được tương ứng, đồng thời show được số lượng những lần bị bắt và bị bắt tại iteration nào.

4.2. Hạn chế:

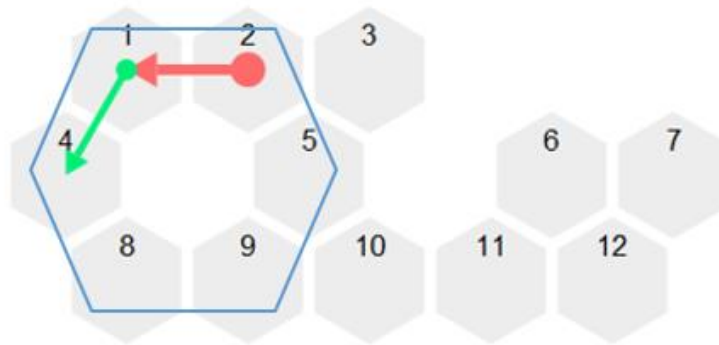
Qua quan sát, nhóm có vài nhận xét về hex map như sau:

- Dưới góc nhìn Prey:
 - Prey là agent chủ động trong hex map, quyết định hướng đi của trò chơi
 - Sẽ có 3 vị trí mà prey không nên di chuyển tới nếu predator đang ở gần, vì nó có thể tạo ra tình huống nguy hiểm
 - + Vị trí số 3 (predator ở vị trí 2/5)
 - + Vị trí số 6 (predator ở vị trí 12)
 - + Vị trí số 7 (predator ở 6/12)
 - Tình huống nguy hiểm: dù prey di chuyển theo hướng nào thì đều có khả năng bị predator bắt được.



- Giả sử prey đang ở vị trí 7, predator ở vị trí 12:
 - + nếu prey di chuyển sang 6 \Rightarrow predator di chuyển sang 6 \Rightarrow bị bắt

- + nếu prey di chuyển sang 12 \Rightarrow predator đứng yên \Rightarrow bị bắt
- + nếu prey đứng yên \Rightarrow predator di chuyển sang 7 \Rightarrow bị bắt
- Tập trung di chuyển ở phía tây như hình sau, chỉ cần di chuyển thành vòng tròn, ngược hướng với vị trí predator đang đứng thì sẽ luôn tìm được hướng đi không bị bắt



- Dưới góc nhìn Predator:
 - Cố gắng dồn prey vào góc phía đông (gồm các ô 6, 7, 11, 12) hoặc ô số 3, 10 (lúc này predator sẽ đứng ở phía bên trái của prey) để tăng khả năng bắt được prey

Từ đó, nhóm kì vọng các agent có thể có được chiến lược khôn ngoan hơn để tối đa hóa reward của mình chứ không chỉ hành xử phụ thuộc vào tình huống cụ thể

5. Tài liệu tham khảo

<https://algorithmsbook.com/files/dm.pdf>

<https://algorithmsbooks.github.io/DecisionMakingProblems.jl/>

<https://medium.com/m/globalidentity?redirectUrl=https%3A%2F%2Ftowardsdatascience.com%2Fintroduction-to-fictitious-play-12a8bc4ed1bb>

[Tutorial · Luxor \(juliagraphics.github.io\)](#)

[Overview · Plots \(juliaplots.org\)](#)

V. Multi-Caregiver Crying Baby

1. Phát biểu bài toán

- Có 1 em bé và 2 người trông trẻ. Em bé sẽ có 2 trạng thái là đói (hungry) hoặc no (sated). Người trông trẻ có các hành động là cho ăn (feed), hát (sing) và ngó lơ (ignore).
- Người trông trẻ không thể biết chính xác trạng thái của em bé mà chỉ có thể đoán phần nào thông qua việc quan sát các hành động của em bé là khóc (crying) và im lặng (quiet).
- Tại mỗi thời điểm, người trông trẻ sẽ phải quyết định thực hiện hành động nào để đạt được số điểm tích lũy là lớn nhất. Nếu em bé bị đói, cả 2 người cùng bị trừ 10 điểm. Nếu người 1 cho em bé ăn, anh ta sẽ bị trừ 2.5, nếu người 2 cho em bé ăn, anh ta sẽ bị trừ 5. Nếu người 1 hát cho em bé nghe, anh ta bị trừ 0.5, nếu người 2 hát cho em bé nghe, anh ta bị trừ 0.25.
- Nếu em bé đang đói và chúng ta không cho em bé ăn, chắc chắn thời điểm tiếp theo em bé sẽ vẫn đói. Nếu chúng ta cho em bé ăn thì ở thời điểm tiếp theo em bé chắc chắn sẽ no bất kể trạng thái hiện tại của em bé là gì. Nếu em bé đang đói thì có 80% khả năng em bé sẽ khóc. Nếu em bé đang no thì có 10% khả năng em bé sẽ khóc. Hát là 1 cách để ta quan sát em bé tốt hơn. Khi chúng ta đang hát, nếu em bé đói thì có tới 90% khả năng em bé sẽ khóc, và nếu em bé đang no thì không bao giờ em bé khóc.

2. Thách thức

- Các kiến thức về lớp bài toán POMG là khó so với nhóm
- Phải tìm hiểu thêm các kiến thức về POMDP và bài toán Crying baby
- Format lại kết quả trả về cho dễ đọc
- Có ít tài liệu tham khảo

3. Thực nghiệm

3.1. Mô hình hóa tính toán

- S: Không gian trạng thái của em bé
- A: Không gian của các hành động người trông trẻ
- O: Sự quan sát người trông trẻ đối với em bé

Bảng giá trị tương ứng:

Biến số	Giá trị
S	{HUNGRY, SATED}
A	{FEED, SING, IGNORE}
O	{CRY, QUIET}

- $T(s' | a, s)$: xác suất ban đầu em bé ở trạng thái s , người trông trẻ thực hiện hành động chung a , em bé chuyển sang trạng thái s'
- $O(o | a, s)$: xác suất người trông trẻ thực hiện hành động chung a , em bé chuyển sang trạng thái s , và người trông trẻ quan sát được o

Ngoài ra, ta cài đặt thêm một số biến chung sau đây, gọi:

- a_not_feed là 1 hành động bất kỳ không phải là feed, nghĩa là có thể là ignore hoặc là sing
- a_not_sing là 1 hành động bất kỳ không phải là sing, nghĩa là có thể là ignore hoặc là feed
- a^* là hành động bất kỳ, $a^* \in \{feed, sing, ignore\}$
- s^* là 1 trạng thái bất kỳ, $s^* \in \{hungry, sated\}$

Ta có bảng mô tả thông số sau đây:

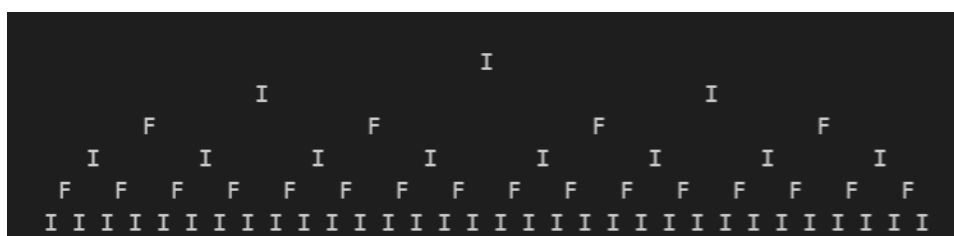
$T(hungry (a_not_feed, a_not_feed), hungry)$	1
$T(hungry (a_not_feed, a_not_feed), sated)$	0.8
$T(sated (feed, a^*), s^*)$	1
$T(sated (a^*, feed), s^*)$	1
$O(crying (a_not_sing, a_not_sing), hungry)$	0.8
$O(crying (a_not_sing, a_not_sing), sated)$	1
$O(crying (sing, a^*), hungry)$	0.9
$O(crying (a^*, sing), hungry)$	0.9
$O(crying (sing, a^*), sated)$	0
$O(crying (a^*, sing), sated)$	0

Gọi $R(s, a)$ là số điểm dành cho cả 2 người chơi tại thời điểm này, em bé ở trạng thái s và người chơi thực hiện hành động chung là a . Từ đó, ta có bảng giá trị điểm tương ứng cho các trường hợp sau đây:

Trường hợp	Số điểm	
	Người 1	Người 2
$s = hungry$	-10	-10
$a[1] = feed$	-2.5	0
$a[2] = feed$	0	-5
$a[1] = sing$	-0.5	0
$a[2] = sing$	0	-0.25

Nhóm dùng tree-based conditional plan để biểu diễn chiến thuật của mỗi người trông trẻ. Đó là 1 cây với gốc là hành động sẽ thực hiện ($F=FEED$, $S=SING$, $I=IGNORE$), nhánh con bên trái là chiến thuật sẽ thực hiện tiếp theo khi đứa trẻ không khóc, nhánh con bên phải là chiến thuật sẽ thực hiện tiếp theo nếu đứa trẻ khóc.

Ví dụ: tree-based conditional plan có độ sâu là 5.



3.2. Phương pháp giải quyết

a. Thực nghiệm 1: Tiếp cận bằng cân bằng Nash d-step

Việc tính cân bằng Nash cho bài toán POMG là tốn kém và khó nên ta chuyển qua tính cân bằng Nash cho d-step POMG. Thuật toán này liệt kê tất cả joint conditional plan có độ sâu là d và xây dựng 1 SimpleGame từ joint conditional plan này. SimpleGame này sẽ hoạt động như sau: 2 người trông trẻ

là 2 agent, joint action space là joint conditional plan (mỗi người chơi sẽ chọn luôn 1 conditional plan, sự lựa chọn của 2 người chơi sẽ hợp thành 1 joint conditional plan là (lựa_chọn_1, lựa_chọn_2), nghĩa là không còn tính liên tiếp của lớp bài toán Markov Game nữa mà chỉ lựa 1 joint action rồi kết thúc (tính chất của bài toán SimpleGame), joint reward là lợi ích (utilities) mà 2 người trông trẻ đạt được nếu thực hiện theo joint conditional plan. Cân bằng Nash cho SimpleGame này cũng là cân bằng Nash cho bài toán POMG tương ứng. Sau khi tạo ra SimpleGame từ POMG, em sẽ tính cân bằng Nash theo thuật toán của SimpleGame.

b. Thực nghiệm 2: Tiếp cận bằng POMGDynamicProgramming

Cách này cũng y hệt cách trên ngoại trừ việc ta không liệt kê tất cả joint conditional plan có độ sâu là d , mà ta sẽ bỏ đi tất cả những conditional plan bị thống trị (dominated) bởi các conditional plan khác. Đầu tiên chúng ta có 1-step plan, chúng ta bỏ đi tất cả nhánh bị thống trị rồi mở rộng nó thành 2-step plan. Cứ như vậy cho đến khi ta có d -step plan.

Một conditional plan π_i thuộc về agent i bị thống trị nếu tồn tại 1 conditional plan π_i' sao cho π_i' luôn cho kết quả tốt hơn hoặc bằng π_i .

3.3. Code

Trong source code, nhóm đã có comment giải thích về cách chạy, phương pháp cũng như mục tiêu, kết quả của những hàm, dòng lệnh. Trong báo cáo này, nhóm sẽ mô tả chung về luồng đi chính, cũng như một số hàm quan trọng.

a. Các hàm quan trọng

- function **transition(pomg::BabyPOMG, s, a, s')**: tính xác suất em bé đang ở trạng thái s , sau khi 2 caregiver thực hiện hành động chung a em bé chuyển sang trạng thái s' . Theo mô hình tính toán thì hàm này tính $T(s'|a, s)$
- function **joint_observation(pomg::BabyPOMG, a, s', o)**: tính xác suất sau khi thực hiện hành động chung a , em bé chuyển sang trạng thái s' và 2 caregiver quan sát được quan sát chung là o . Theo mô hình tính toán thì hàm này tính $O(o|a, s')$
- function **joint_reward(pomg::BabyPOMG, s, a)**: tính lợi ích chung của 2 caregiver biết em bé ở trạng thái s và họ thực hiện hành động chung là a
- function **create_conditional_plans(P, d)**: liệt kê tất cả d -step conditional plan của đối tượng $P::POMG$

- function **expand_conditional_plans**(P, Π): liệt kê toàn bộ k-step conditional plan từ Π là tập hợp (k-1)-step conditional plan của đối tượng P::POMG
- function **solve**(M::POMGNashEquilibrium, P::POMG): đưa ra cân bằng Nash cho bài toán của đối tượng P::POMG theo cách tiếp cận d-step conditional plan. Cụ thể hàm này sẽ gọi **create_conditional_plans** để tạo ra toàn bộ d-step conditional plan, sau đó sẽ tạo đối tượng G::SimpleGame với A là toàn bộ d-step conditional plan. Cuối cùng hàm này sẽ gọi hàm **solve**(M::NashEquilibrium, G::SimpleGame) để tìm cân bằng Nash cho G::SimpleGame. Kết quả trả về cũng là kết quả của hàm **solve**(M::NashEquilibrium, G::SimpleGame)
- function **is_dominated**(P::POMG, Π , i, π_i): kiểm tra xem liệu conditional plan π_i mà người chơi I thực hiện có bị thống trị bởi 1 conditional plan bất kỳ nào thuộc $\Pi[i]$ hay không. $\Pi[i]$ là tập hợp các conditional plan mà người trông trẻ I có thể thực hiện
- function **prune_dominated**(Π , P::POMG): xóa đi toàn bộ conditional plan bị thống trị trong Π
- function **solve**(M::POMGDynamicProgramming, P::POMG): đưa ra cân bằng Nash của đối tượng P::POMG theo cách tiếp cận POMG Dynamic programming. Cụ thể hàm này lặp d lần, với mỗi vòng lặp sẽ gọi hàm **expand_conditional_plan** để mở rộng các conditional_plan, và sau đó sẽ gọi hàm **prune_dominated** để xóa đi toàn bộ các policy nào bị thống trị. Cuối cùng sẽ tạo G::SimpleGame từ với A là toàn bộ d-step conditional plan sau d vòng lặp. Và gọi hàm **solve**(M::NashEquilibrium, G::SimpleGame) để tìm cân bằng Nash cho G::SimpleGame. Kết quả trả về cũng là kết quả của hàm **solve**(M::NashEquilibrium, G::SimpleGame)

b. Luồng đi chính

b.i. Thực nghiệm 1

- Tạo multicaregiver_cryingbaby là đối tượng lớp MultiCaregiverCryingBaby
- Tạo pomg là đối tượng lớp POMG từ đối tượng multicaregiver_cryingbaby
- Tạo pomgNashEquilibrium là đối tượng lớp POMGNash
- Gọi hàm solve(pomgNash, pomg) để trả về cân bằng Nash cho bài toán Multi-Caregiver Crying Baby
- Dùng hàm printAns(ans) để in kết quả dưới dạng cây cho dễ đọc
- Gọi hàm test(C1, C2). Hàm này sẽ khởi tạo 1000 C3 thách thức C1. Hàm này sẽ in số lần $utility(C1, C2) \geq utility(C3, C2)$

b.ii. Thực nghiệm 2

- Tạo multicaregiver_cryingbaby là đối tượng lớp MultiCaregiverCryingBaby

- Tạo pomg là đối tượng lớp POMG từ đối tượng multicaregiver_cryingbaby
- Tạo pomgDP là đối tượng lớp POMGDynamicProgramming
- Gọi hàm solve(pomgDP, pomg) để trả về cân bằng Nash cho bài toán multi-caregiver crying baby
- Dùng hàm printAns(ans) để in kết quả dưới dạng cây cho dễ đọc
- Gọi hàm test(C1, C2). Hàm này sẽ khởi tạo 1000 C3 thách thức C1. Hàm này sẽ in số lần $utility(C1, C2) \geq utility(C3, C2)$

c. Kiểm tra kết quả trả về

Nhóm đã viết hàm để kiểm tra xem liệu kết quả trả về đã thực sự tối ưu chưa.

c.i. Phương pháp test:

- Trước hết, ta cần những conditional plan sau

C1	Conditional plan của caregiver 1
C2	Conditional plan của caregiver 1
C3	Conditional plan thách thức C1

- Gọi $utility(C1, C2)$ là lợi ích kỳ vọng của C1 nếu 2 caregiver thực hiện joint conditional plan (C1, C2). Nếu $utility(C1, C2) \geq utility(C3, C2)$, thì C1 thắng
- Nhóm sẽ chạy ngẫu nhiên 1000 trường hợp C3 khác nhau để kiểm tra

c.ii. Hàm test kết quả trả về

- Hàm $utility(C1, C2)$ trả về lợi ích kỳ vọng của người trông trẻ 1 nếu cả 2 người trông trẻ thực hiện theo conditional plan chung (C1, C2).
- Ta xây dựng 1 probabilityTree với mỗi node sẽ có cấu trúc như sau: (p_hungry, p_cry) tương ứng với mỗi node ở tree conditional plan sẽ có xác suất trạng thái hiện tại của đứa bé là đói là p_hungry và xác suất sau đó đứa bé sẽ khóc là p_cry. Dựa trên đó, ta có công thức như sau:

- Node.p_hungry

Trường hợp	Node.p_hungry
Hành động trước đó không phải là feed ở cả 2 conditional plan	$\text{parentNode.p_hungry} + (1 - \text{parentNode.p_hungry}) * 0.8$
Ngược lại	0

- Node.p_cry:

Trường hợp	Node.p_cry
Hành động hiện tại không phải là sing ở cả 2 conditional plan	$\text{Node.p_hungry} * 0.8 + (1 - \text{Node.p_hungry}) * 0.1$
Ngược lại	$\text{Node.p_hungry} * 0.9$

- Ta xây dựng utilityTree với mỗi node là u tương ứng với lợi ích kỳ vọng của cây con gốc là node đó

Trước hết, ta cần xây dựng những dữ kiện sau:

- Gọi a là hành động chung của cả 2 người trông trẻ tại node hiện tại
- Gọi NodeP là node tương ứng với node ở utilityTree trên probabilityTree
- Giả sử hàm $R(s, a)$ trả về $\langle u_1, u_2 \rangle$ với u_1 là lợi ích của người trông trẻ 1 nếu em bé ở trạng thái s và 2 người thực hiện hành động chung là a
- Gọi CryNode(Node) là node con tương ứng với hành động hiện tại của đứa bé sẽ là cry
- Gọi QuietNode(Node) là node con tương ứng với hành động hiện tại của đứa bé sẽ là quiet

Từ đó, ta xây dựng được công thức tính Node.u. Kết quả trả về tương ứng sẽ là root.u với root là node ở utilityTree tương ứng với node gốc của conditional plan. Công thức như sau:

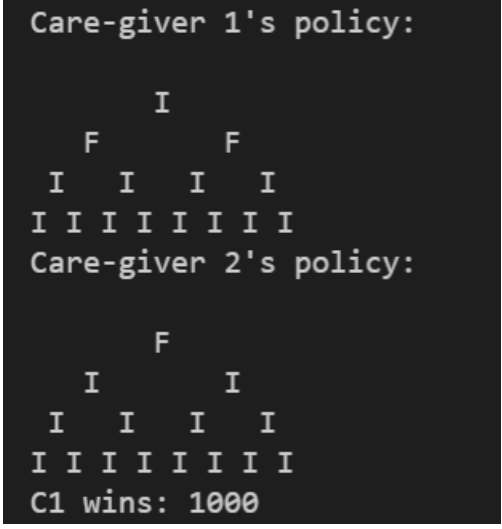
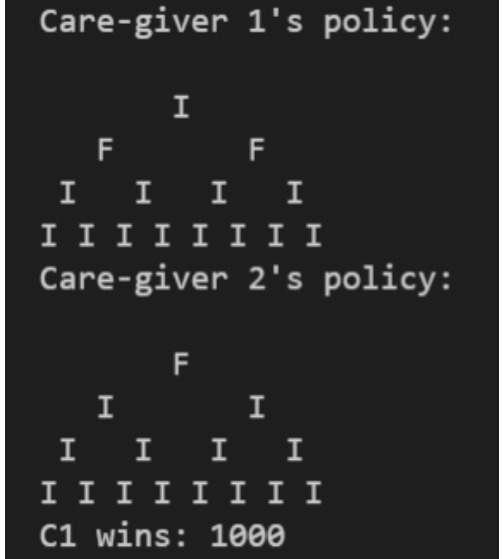
Node.u

$$= \text{NodeP.p_hungry} * R(\text{hungry}, a)[1] + (1 - \text{NodeP.p_hungry}) * R(\text{sated}, a)[1] \\ + \text{CryNode}(\text{Node}).u * \text{NodeP.p_cry} + \text{QuietNode}(\text{Node}).u * (1 - \text{NodeP.p_cry})$$

3.4. Phân tích

a. Thực nghiệm 1

a.i. Kết quả thực nghiệm

b	Kết quả	Đánh giá
[0.2, 0.8]		C1 thắng tuyệt đối
[0.5, 0.5]		C1 thắng tuyệt đối

a.ii. Ý nghĩa kết quả

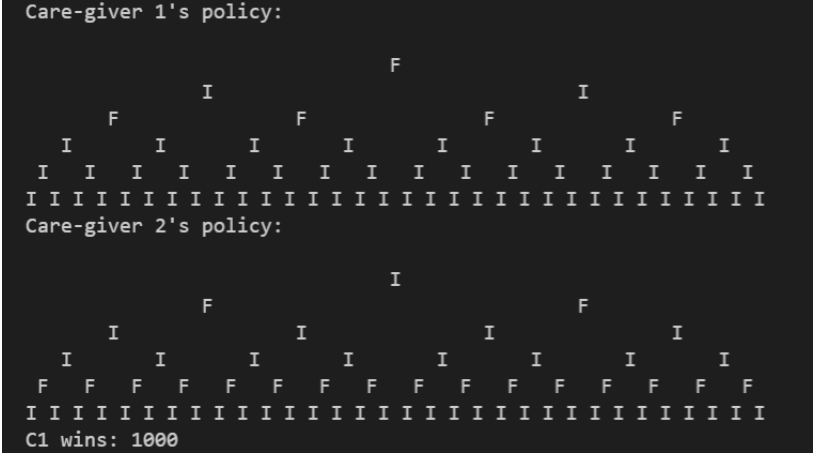
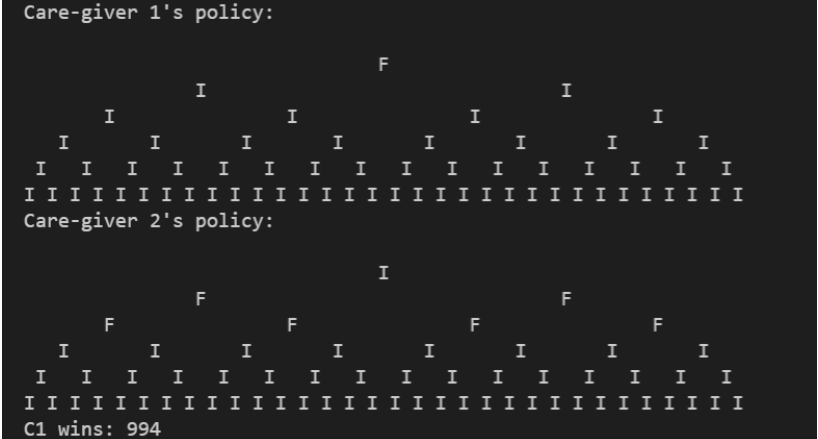
- Tại mỗi node là hành động sẽ thực hiện tại node đó với F: Feed, S: Sing, I: Ignore.

- Caregiver 1's policy và caregiver 2's policy là 2 conditional plan tạo thành cân bằng Nash. Ban đầu Ta sẽ thực hiện hành động tại nút gốc. Nếu sau đó quan sát được đứa trẻ đang khóc, ta sẽ đi xuống nhánh con bên phải, nếu đứa trẻ nín ta sẽ đi xuống nhánh con bên trái.
- Ta thấy, ở mỗi tầng, hành động mà người trông trẻ sẽ làm là giống nhau bất kể việc quan sát thấy đứa trẻ ra sao, và không có người trông trẻ nào thực hiện hành động SING cả
- Dòng cuối cùng là số lần C1 thắng (ta khởi tạo ngẫu nhiên 1000 conditional plan C3 thách thức C1). Ở đây ta thấy số lần C1 thắng là tuyệt đối 1000 lần, cho thấy độ tối ưu của thuật toán này là rất cao

b. Thực nghiệm 2

b.i. Kết quả thực nghiệm

Thực nghiệm 2 chạy nhanh hơn thực nghiệm 1 nên em đặt $d=5$

b	Kết quả	Đánh giá
[0.2, 0.8]		C1 có kết quả thắng tuyệt đối
[0.5, 0.5]		Tỉ lệ thắng của C1 là cực cao

b.ii. Ý nghĩa kết quả

- Tương tự như trên, ở cách tiếp cận này cũng không có người trông trẻ nào thực hiện SING cả, và tại mỗi tầng người trông trẻ sẽ thực hiện hành động giống hệt nhau bất kể quan sát được gì
- Ở kết quả C1 wins khi $b = [0.5, 0.5]$ không phải tuyệt đối 1000 lần có thể do vector phân phối xác suất ban đầu.

4. Đánh giá kết quả

4.1. Điểm mạnh

- Có thể thấy kết quả trả về nhìn thì vô lý (không người trông trẻ nào thực hiện hành động SING, ở mỗi tầng hành động đều giống nhau) nhưng hiệu quả lại cao (kết quả kỳ vọng cao hơn nhiều conditional plan khác)
- Từ kết quả trả về là 1 vector của conditional plan, nhóm đã mô hình hóa về dưới dạng cây khá dễ đọc
- Cách tiếp cận POMG Dynamic Programming tính toán nhanh hơn nhiều so với POMG d-step conditional plan do vừa xây dựng joint conditional plan space vừa bỏ đi các conditional plan không cần thiết, làm nhỏ đi đáng kể không gian tính toán

4.2. Hạn chế

- Ở đây ta chỉ tìm được d-step conditional plan mà không tìm được conditional plan nào có thể áp dụng mãi mãi (trường hợp có nhiều hơn d thời điểm)
- Tham số d tương đối nhỏ (với cách tiếp cận POMG Nash equilibrium, nếu $d > 3$, mất rất lâu để thuật toán cho ra kết quả)

5. Tài liệu tham khảo

- <https://github.com/algorithmsbooks/DecisionMakingProblems.jl>
- <https://algorithmsbooks.github.io/DecisionMakingProblems.jl/>

E. Tóm tắt kết quả

I. Điểm mạnh

- Tiếp cận 5 bài toán theo yêu cầu đề bài, phân tích chi tiết ở nhiều khía cạnh khác nhau cho một bài toán
- Đồ án này giúp nhóm hiểu được một số bài toán Game Theory khá hay, gần gũi và có thể áp dụng vào cuộc sống để có những quyết định khôn ngoan hơn
- Nhóm đã tìm hiểu nhiều phương pháp để có thể chọn ra phương pháp giải quyết tốt nhất, trực quan nhất cho từng bài toán trong khả năng của nhóm
- Với mỗi bài toán, nhóm đều cố gắng tìm cách visualize kết quả, phân tích và đánh giá kết quả đạt được
- Qua đồ án, tất cả các thành viên đã học thêm được ngôn ngữ Julia, nắm bắt được syntax, tìm hiểu thư viện và áp dụng được thư viện
- Phần source code của nhóm có comment đầy đủ, cố gắng giải thích dễ hiểu, tốt nhất trong khả năng

II. Điểm yếu

- Nhóm chưa tự đề xuất được phương pháp của riêng mình, đa phần là cài đặt tham khảo từ sách
- Các thuật toán trong 5 bài toán nhóm cài đặt đều có những hạn chế nhất định, chưa thật sự hoàn hảo
- Nhóm chưa nêu được nhiều thực nghiệm cho từng bài toán