ĐẠI HỌC QUỐC GIA VIỆT NAM THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

NHẬP MÔN TRÍ TUỆ NHÂN TẠO (CO3061)

ASSIGNMENT BONUS

Giảng viên hướng dẫn: Vương Bá Thịnh
Sinh viên thực hiện: Nguyễn Đắc Hoàng Phú - 2010514
Đoàn Trần Cao Trí - 2010733
Du Thành Đạt - 2010206

Thành phố Hồ Chí Minh, 05/2022

Contents

1 Cây quyết định

Nhận xét thấy, mỗi bước chơi sẽ là sự di chuyển của người chơi theo hướng nhất định và chọn ô block thích hợp. Cuối cùng thì kết quả chỉ có 2 trường hợp đó là người 1 thắng hoặc người 2 thắng. Từ suy nghĩ đó, nhóm hình thành ý tưởng xây dựng cây quyết định.

1.1 Lý thuyết cây quyết định

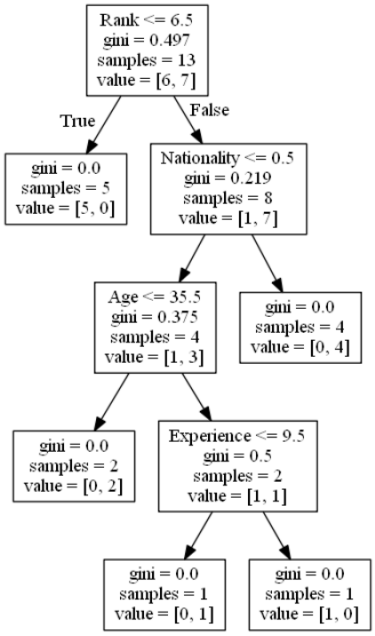

Figure 1: *Mô phỏng cây quyết định*

Cây quyết định là cây xây dựng nhằm mục đích đưa ra quyết định cuối cùng khi phải dự đoán kết quả từ một danh sách chứa các thuộc tính.

Thông tin trên cây quyết định bao gồm:

- Các nút trong chứa các điều kiện để được phép đi tiếp đến các nút con của nó.
- Nút lá chứa giá trị quyết định cuối cùng của cây.

Qua từng bước duyệt từng thuộc tính trong danh sách trên, ta sẽ chọn hướng đi theo nhánh cây thỏa điều kiện và lặp lại quá trình đến khi gặp node lá. Từ đó đưa ra giá trị quyết định tại node lá này.

1.2 Ý tưởng xây dựng cây quyết định đối với bài toán Isolation

Cụ thể hóa với trò chơi như sau. Tại vị trí hiện tại của người chơi, nếu di chuyển sẽ được mã hóa thành giá trị như bảng sau.

0	3	6
1	x	7
2	5	8

Và mỗi ô block cũng được mã hóa. Với ô ở hàng r, cột c thì sẽ được mã hóa thành số r * 6 + c

Từ đó, nhóm xây dựng được tập dữ liệu cây quyết định sau:

Có 4 loại vị trí trên từng dữ liệu chụ thể như sau:

- Vị trí chia hết cho 4 là hướng đi của người chơi 1.
- Vị trí chia 4 dư 1 là vị trí ô block đã được mã hóa.

```
7 28 0 10 5 27 6 16 5 26 1 22 1 15 1 9 5 14 1 25 1 13 5 20 0 19 8 7 5 24 7 12 -1
7 28 0 10 5 19 6 16 2 26 3 9 5 22 1 15 7 7 5 14 5 25 0 20 8 8 2 27 6 13 5 34 5 31 5 29 -1
8 28 0 8 2 20 6 9 7 16 2 10 5 19 1 26 0 32 0 21 0 13 3 14 1 7 5 1 3 25 5 6 -1
8 28 1 16 2 27 0 20 2 26 1 14 0 19 3 7 3 13 8 1 5 32 5 18 3 25 1 12 3 31 3 6 -1
8 28 1 3 2 27 1 13 5 26 0 15 1 25 8 14 2 32 1 19 8 24 1
8 28 0 16 5 19 6 10 2 26 0 15 0 9 3 20 3 13 0 14 0 8 5 1 5 2 2 7 3 18 3 1
8 28 0 2 5 21 1 10 2 26 1 15 1 25 3 14 2 13 7 31 5 20 2 1
8 28 0 10 5 25 0 22 2 14 1 15 7 13 5 16 5 19 8 20 2 26 1 27 1 24 1
0 28 8 10 3 25 8 22 6 14 7 15 1 13 3 16 3 19 0 20 6 26 7 27 7 24 -1
8 28 0 8 5 27 0 10 2 14 8 15 1 21 3 26 1 25 1 13 3 20 2 7 8 31 3 24 0 19 1
8 28 0 8 5 25 0 10 2 14 8 15 7 19 7 16 7 20 8 27 6 29 7 22 5 34 1
5 28 0 9 8 33 1 10 2 26 3 15 2 14 1 20 8 13 8 27 1 19 0 25 0 12 1
8 28 1 15 2 27 0 9 2 20 7 14 1 21 0 13 3 26 2 7 5 19 7 24 3 32 5 18 3 25 1 1 5 31 1
5 28 1 14 2 27 0 8 8 26 1 15 1 25 3 20 2 13 3 19 8 7 5 24 7 12 1
5 28 0 3 7 20 1 16 2 26 3 9 1 14 1 8 5 13 8 24 0 19 8 7 3 27 1 1 3 25 7 6 -1
2 1 27 7 6 6 28 7 9 7 29 7 10 7 34 1
5 0 19 8 20 7 25 7 21 8 27 1 28 7 26 7 29 1 31 1
8 1 27 0 21 0 20 1 14 0 19 3 7 3 13 8 1 5 25 8 18 3 25 7 12 3 28 7 6 -1
```

Figure 1: Dữ liệu thu thập được

- Vị trí chia 4 dư 2 là hướng đi của người chơi 2.
- Vị trí chia 4 dư 3 là vị trí ô block đã được mã hóa.

1.3 Quy trình thu thập dữ liệu

Các quy trình thu thập dữ liệu đều sử dụng phương pháp chung là so sánh trạng thái trước đó với trạng thái hiện tại để xác định sự thay đổi. Nghĩa là từ 2 trạng thái sẽ biết được người chơi nào di chuyển như thế nào và chặn BLOCK tại đâu. Tại mỗi lượt chơi như vậy, sẽ có 1 danh sách toàn cục lưu dữ trạng thái mã hóa này thành cặp (hướng di chuyển, ô block).

Các nguồn thu thập trạng thái

- Người với người HumanVsHuman.py
- Người với Minimax MinmaxVsHuman.py
- Cho máy tính tự train với chính nó MinimaxVsMinimax.py

Khi trò chơi kết thúc, danh sách toàn cục trên sẽ thêm vào dữ liệu cuối cùng có giá trị là 1 nếu người chơi 1 thắng và ngược lại, có giá trị là -1 nếu người chơi 2 thắng.

Dữ liệu hoàn chính được thêm vào dòng mới của file **experience.txt**

1.4 Tải và xây dựng model

Khi chương trình bắt đầu chạy, Model sẽ tiến hành build cây quyết định (nếu chưa có). Dữ liệu được đọc vào từ file **experience.txt**. Mỗi điểm dữ liệu gồm 2 phần:

- Danh sách từ đầu đến cuối (trừ dữ liệu cuối danh sách) sẽ là thuộc tính (bước đi) để xây dựng cây quyết định.
- Dữ liệu cuối cùng sẽ là kết quả thực tế nếu đi theo từng bước như trên.

Xây dựng DecisionTree với phương pháp Regressor để xem xét trong các hướng đi có thể dành chiến thắng thì dành chiến thắng nào là cao nhất.

Thư viện sử dụng **sklearn**

Do tham số danh sách thuộc tính (X) để xây dựng cây quyết định phải có độ dài bằng nhau nên với các trường hợp trò chơi kết thúc với số lượt đi khác nhau, nhóm sẽ bổ sung dữ liệu -1 vào cuối danh sách thuộc tính (X) và ấn định độ dài cố định là 100.

Sau khi chuẩn bị đủ tham số danh sách thuộc tính (X) và tham số kết quả (Y), nhóm tiến hành xây dựng cây quyết định:

self.model = tree.DecisionTreeRegressor()

self.model = self.model.fit(self.X, self.Y)

1.5 Chơi và kiểm thử

Dựa trên ý tưởng và cách xây dựng trên, nhóm đã tiến hành và xây dựng thành công và chơi để kiểm thử. Kết quả model chơi không được tốt. Các nước đi khá vô nghĩa và không làm khó được đối thủ.

Từ đó, nhóm phân tích nguyên do là số nước đi lượt thực hiện block ô trên bàn cờ là quá lớn, số lượng trạng thái ở các tầng này trong cây quyết định khá lớn và không rõ ràng, dẫn đến model có nhiều dự đoán sai và quyết định nước đi không hiệu quả.

Từ đó, nhóm tiến hành nghiên cứu và thực hiện với giải thuật Q_learning với mong muốn đổi mới suy nghĩ và model hiệu quả hơn.

2 Q_learning

2.1 Cơ sở lý thuyết

2.1.1 Khái niệm Q-learning

Học tăng cường - Reinforcement Learning là một trong ba kiểu học máy chính (cùng với Học giám sát - Supervised Learing và Học không giám sát - Unsupervise Learning). Q-learing là một thuật toán học tăng cường không mô hình (model-free). Q viết tắt cho Quality, biểu thị cho độ tốt của việc thực hiện một hành động (action) tới việc tăng phần thưởng (reward) trong dài hạn. Từ đó, thuật toán có thể tạo ra action-value function hay một Q-table được khởi tạo với một giá trị bất kỳ.

2.1.2 Siêu thông số (Q-learning hyper parameters)

Có hai thông số cần được chú ý trong Q-learning là discount factor ϵ và số bước tối đa mỗi episode (maximum number of steps of an episode).

Discount factor ϵ là một số thực trong khoảng (0;1) xác định sự phụ thuộc của trạng thái hiện tại (current state) với trạng thái tương lai (future state). Giá trị ϵ càng lớn, trạng thái hiện tại (current state) càng phụ thuộc trạng thái tương lai (future state). Thông thường, ta dùng $\epsilon = 0.95$.

Trong một số trường hợp, trò chơi rơi vào vòng lặp khi người chơi lặp lại các hành động giống nhau. Để ngăn chặn vòng lặp này, chúng ta giới hạn của người chơi còn 36 bước.

2.1.3 Replay Memory

Trong quá trình lặp qua các episode, chúng ta cần lưu lại training data để huấn luyện (train) model. Để có đủ lượng data phục vụ huấn luyện (training) cũng như ngăn chặn hiện tượng forgetting của Neural Network, chúng ta giới hạn lower bound là ... và upper bound là .... Nếu data mới đạt ngưỡng giới hạn của upper bound, data cũ sẽ bị thay thế bởi data mới.

2.1.4 Deep Q-learning

Mục tiêu của Q-learning là làm tối ưu Q-table, một bảng 2 chiều với các trục lần lượt là trạng thái (state) và hành động (action). Từ Q-table và các trạng thái (state) cho sẵn, chúng ta có thể tối ưu hành động (action) bằng cách tìm ra Q-value lớn nhất từ hàng của trạng thái (state).

Thế nhưng, khi số trạng thái (state) và hành động (action) tăng lên đáng kể, việc sử dụng Q-table không còn khả thi. Lúc này, Deep Q-learning xuất hiện và thay thế Q-table bằng Deep Neural Network. Nhìn chung, Deep Q-learning thừa hưởng toàn bộ từ Q-learning, chỉ khác ở Deep Neural Network. Chỉ cần đưa state vào như input là Deep Neural Network lập tức đưa ra output là các giá trị action-value.

2.2 Ý tưởng chung

Cho hai Minimax đánh với nhau, một Minimax với mức chơi giỏi và một Minimax với mức chơi trung bình (tức không gian tìm kiếm hẹp hơn) hoặc đánh với randomAgent, đánh khoảng 1000 ván, rồi cho aiAgent học các đặc trưng từ nước đi qua các ván đã đánh, để agent biết khi gặp nước đi của đối phương, nó sẽ rút trích các đặc trưng đã học để dự đoán nước đi tiếp theo đó.

2.3 Mục tiêu chính

Tạo ra một aiAgent để có thể giải quyết đi bùng nổ về không gian tìm kiếm như trong giải thuật minimax, bên cạnh đó tìm cách tối ưu giúp cho agent có thể suy nghĩ lập kế hoạch trước để có đạt được mục đích thắng ván cờ.

2.4 Dữ liệu đầu vào

Ta build model với dữ liệu là các states và actions để tạo ra các states đó và reward được evaluate từ states, các states, actions, rewards này có được do quá trình đánh nhau giữa hai minimax được thể hiện ở đoạn code bên dưới.

```
#state, and action create from here
def play(self,max_steps,player):
    done = False
    reward = 0
    random_factor = np.random.choice([-1,1], 1)[0]
    state_lst, reward_lst, action_lst = [], [], []

    for z in range(max_steps * 2):
        # print("================================")
        # print(z)
        # print("================================")
        env_1 = MinimaxII(self.depth,self.path)
        env_2 = MinimaxII(self.depth2,self.path)

        state_lst += [self.env.getBoard(self.board.copy())]
        if player == 1:
            #fix erorr here
            reward_lst += [reward]
        else:
            reward_lst += [-reward]

        self.board = self.env.board

        ret = random_factor==player
        if ret:
            action = env_1.minimax_act(self.board,player)
        else:
            action = env_2.minimax_act(self.board,player)

        reward,done = self.check_win()

        action_lst += [action]
        player = -player
        if done:
            break
        # print(self.board)
    return state_lst,reward_lst, action_lst,done

################################################
state_lst,reward_lst,action_lst,done = env.play(max_steps,player)
# print(action_lst)
agent.observe_on_training(state_lst, reward_lst, action_lst, done)
################################################
```

2.5 Quá trình huấn luyện dữ liệu - build model

Sẽ có hai mạng neural chính được sử dụng, đầu tiên là mạng neural cho quá trình training, và mạng neural cho việc học được những đặc trưng tốt nhất từ mạng training đó gọi là target network. Sau khi có được dữ liệu đầu vào, ta bắt đầu cho dữ liệu đó đi qua một mạng neural traing nhằm có thể tạo ra một hàm mà hàm đó có thể ánh xạ từ stats qua reward thông qua action học được. Ta sẽ train từ các states,action và rewards trong 1 ván và lấy một số mẫu để giúp cho agent học như vậy 36 lần, và lặp lại điều này trong 999 ván tiếp theo. file train model ghi là train.py

```
hist = [agent.train_network(64 ,64,1,verbose=0, cer_mode=True) for _ in range(epoch_per_eps)]
```

```
if hist is not None:
    loss_records += hist
    reward_records.append(reward_lst[-1])

if ep%target_update == 0:
    agent.update_target_network()
```

2.6 Kết quả xây dựng model

Kết quả MSE sau 1000 ván bao gồm các nước đi và các nước lặp kế hoạch:

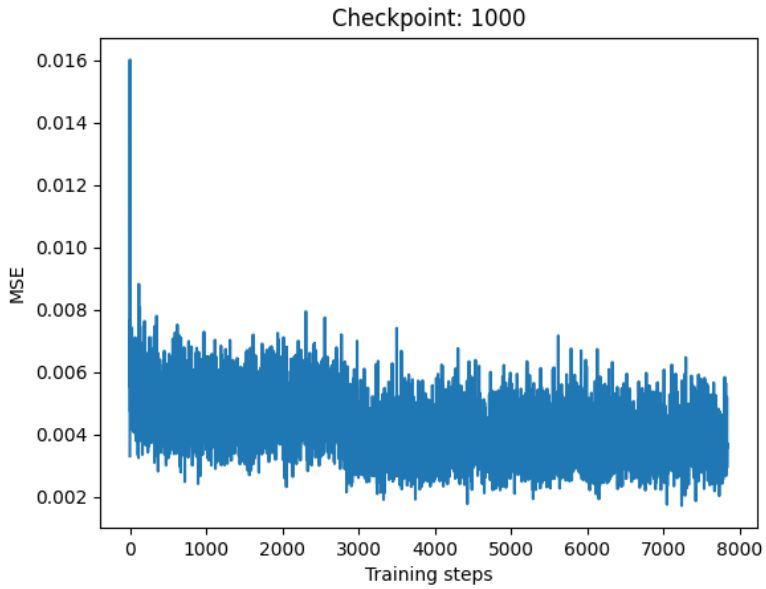

Figure 3: *kết quả xu hướng giảm dần, chứng tỏ, hàm ánh xạ học được đang tốt hơn, cho kết quả predict gần kết quả thực tế hơn*

2.7 Chơi và đánh giá

Nhóm có viết một file để aiAgent đánh với randomAgent khoảng 1000 ván và kết quả đạt được khá tốt. file đánh là tournamentAI.py

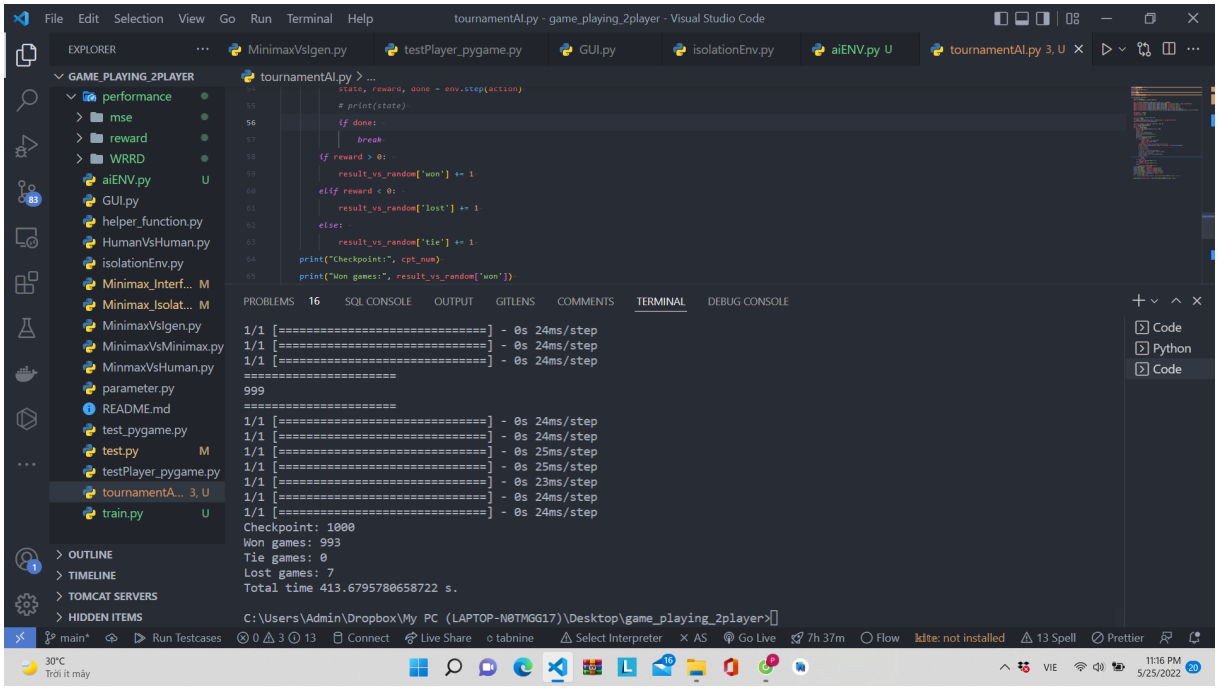

Figure 4: *aiAgent đạt tỉ lệ chiến thắng khoảng 99%*

2.8 Hướng đi kế tiếp

Model chỉ mới train khoảng một 1000 ván đã rất thông minh, nhóm dự tính sẽ train nhiều ván hơn và thử cho aiAgent đánh thử với aiMinimax cũng như aiAgent khác. Bên cạnh đó, thay vì cho hai minimax đấu nhau, nhóm dự tính có thể để aiAgent đấu nhau và xem thử kết quả mô hình có cải thiện hơn hay không.