Peg Solitaire using Artificial Intelligence

Subham Swastik Samal, Zijian Ding

July 10, 2023

1 Introduction and Related Work

Peg solitaire, a classic single-player board game dating back to the 17th century, is played on a board with holes filled with pegs, where the objective is to remove all but one peg by jumping them over each other. Valid moves involve jumping a peg (horizontally or vertically) over an adjacent peg into a hole two positions away and then removing the jumped peg. Previous solutions of the game that have been proposed involve search algorithms like Bidirectional BFIDA[1] and integer programming[2]. Despite its simplicity, peg solitaire presents a complex combinatorial problem, making it an ideal candidate for exploring the capabilities of artificial intelligence and learning algorithms such as Deep Q-Networks (DQNs).

DQNs are powerful extensions of the traditional Q-learning algorithms and use deep neural networks to approximate the Q function. Q functions is a fundamental concept in reinforcement learning which represents the expected future reward for taking a specific action in a given state. Among other implementations of DQNs, recently they have achieved remarkable success in solving various board games, including Atari[4], Go[5].

2 Problem Statement

This study aims to achieve the following objectives:

- Implement Deep-Q Networks to learn optimal strategies for various peg solitaire board configurations.
- Following a turn-based interaction model, develop a methodology for assisting human players for the game. In this approach, the bot and the human player take turns, with the bot making the first move and the human following. The bot's goal is to make the optimal move while considering the possible actions the human player might take in their subsequent turn. This requires the AI system to predict and account for human decision-making, incorporating human players' uncertainties.

By addressing these two research goals, the study aims to develop an AI-based peg solitaire solver that demonstrates strong performance in solving complex board configurations and provides an enhanced user experience by effectively collaborating with human players.

3 Methods

3.1 DQN Formulation

Any state in our study is represented by a n x n array, in which each element is 0 or 1, where 0 represents no peg present in the position, while 1 represents presence of peg in the position. An action is defined by a tuple of size 2 representing the start and end positions of a moving peg, for example: Action ((4,2),(2,2)) represents peg from position (4,2) to (2,2) over (3,2). The Action space consists of all possible actions (e.g.: all 32 possible moves for a 4x4 board), but it should be noted that only a few actions are valid at each state.

To implement a DQN, a deep neural network architecture was designed, which consists of three fully connected layers, each containing 128 neurons. The network's input consists of the state representation, which for our case is the n x n array, converted to a n^2 x 1 vector, while the output yields the estimated

Q-values for all potential actions. During training, a DQN necessitates defining a loss function that quantifies the discrepancy between the predicted and target Q-values. The commonly employed mean squared error (MSE) loss used in our case. The Adam optimization algorithm is utilized to adjust the network's parameters iteratively to minimize the loss. A crucial aspect of DQNs is experience replay, which addresses challenges related to correlated samples and non-stationary target values. This technique employs a replay buffer to store past experiences as tuples (s, a, r, s'), where 's' represents the state, 'a' the action, 'r' the reward, and 's'' the following state. During training, random 8192 experiences (batch) are sampled from the buffer for training the framework. A pseudo-code summary of this implementation is shown in Algorithm 1.

Algorithm 1

while running **do**
  $a \leftarrow \arg\max Q(s, a)$
  Add (s^t, a^t, r^t, s^{t+1}) to memory
  if len(memory) $>$ batch size **then**
    Sample batch of (s, a, r, s')
    $Q_{\text{target}} \leftarrow r + \gamma * Q'(s')$
    $Q_{\text{expected}} \leftarrow Q(s)$
    $L(\theta) \leftarrow ||Q_{\text{target}} - Q_{\text{expected}}||$
    $Q' \leftarrow$ weights closer to Q

3.2 Human Modeling

The Boltzmann Human model is a complex mathematical framework designed to capture and describe human decision-making processes in the context of reinforcement learning and artificial intelligence. It is based on the premise that human choices are influenced by the estimated value of different actions and the degree of uncertainty or exploration associated with those actions. By generalizing the randomness of human decisions, the Boltzmann human model provides a more realistic representation of how people make choices in various situations. We define our model of human action, Eqn.1, based on the Boltzmann human model.

$$P(a_H|s_1, \theta) = \frac{e^{\beta * Q_\theta(s_1, a_H)}}{\sum_{a'_H \in A} e^{\beta * Q_\theta(s_1, a'_H)}} \tag{1}$$

Where $P(a_H|s_1, \theta)$ represents the probability of selecting action a_H depends on s_1 and θ, $Q_\theta(s, a_H)$ represents the estimated action value, and β is the parameter regulating the rationality of the human model. The denominator term $\sum_{a'_H \in A} e^{\beta * Q_\theta(s_1, a'_H)}$ is used as a normalization factor to ensure that the sum of the probabilities of all possible actions is 1. The parameter β plays a crucial role in the balance. When β is large, the human model is more rational, resulting in the highest-value behaviors being selected more frequently, and when β is small, the probability of all behaviors being selected more evenly.

4 Experiments And Results

4.1 4*4 Peg Board

For a 4x4 board, we use the reward function as given in Eqn.(2) and train a Deep Q-Network (DQN) to learn the best strategy for solving the peg solitaire problem. We set the learning rate to 5e-2 (0.05) for the DQN and use a discount factor (gamma) of 0.99 for the Q-value updates. The starting configuration of the board is shown in Fig.1(L), with the solution obtained after 1000 iterations is given in Fig.1(R).

$$Reward = \begin{cases} 100 & \text{if Minimum possible Pegs (2)} \\ -2^{\text{no of pegs on board}} & \text{otherwise} \end{cases} \tag{2}$$

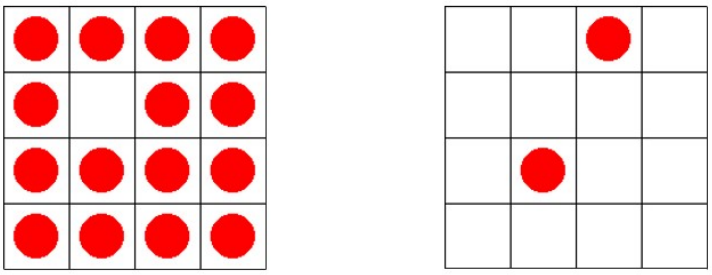

Figure 1: The initial state of a 4*4 Peg Board (L) and the solution obtained with DQN (R)

The game was run for 1000 iterations, from which for the first 400 iterations, the agent takes random steps for data collection. Post that, the algorithm explores 10% of the time. Fig.2 shows how many pegs are left after each training, and after about 800 episodes, the DQN converges to a state where only two pegs remain, the minimum possible solution for the 4x4 board. In addition, a moving average is displayed on the right side of the figure with a window size of 100 steps, further showing the trend clearly and concisely.

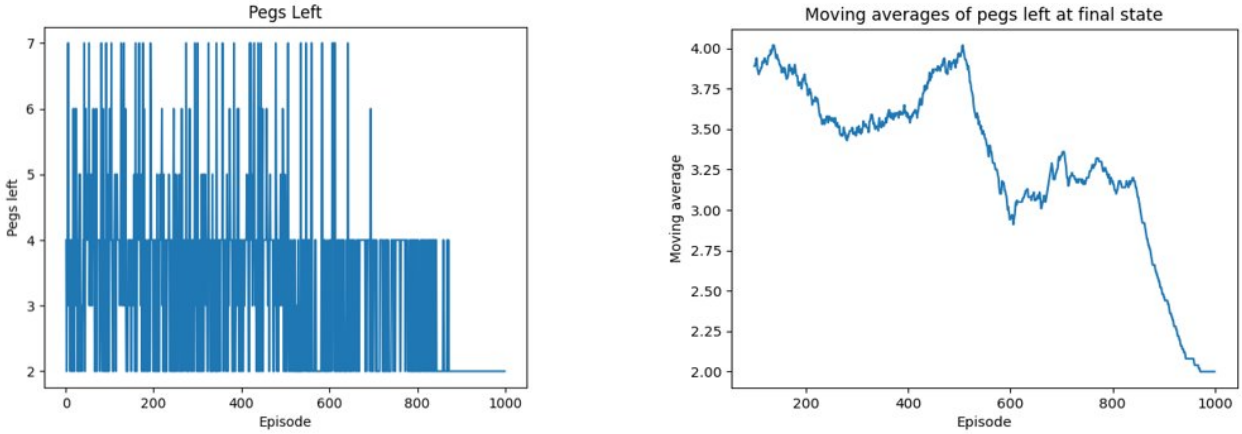

Figure 2: The overall trend of the remaining pegs.

4.2 5*5 Peg Board

For higher-dimensional boards, the complexity of board configurations presents significant challenges. As the number of pegs increases, the possible board configurations expand exponentially, making it difficult for an agent to explore and learn from every potential state. Additionally, the agent must grapple with long-term dependencies, as the reward for a specific move is not immediately apparent. The agent may need to execute a long sequence of moves to achieve a desirable state, which complicates the learning process for an optimal policy. Consequently, the agent must carefully consider the long-term implications of its actions in order to navigate this intricate environment effectively.

In light of these challenges, we propose two strategies to assist the DQN in finding a solution: improving the reward function and incorporating additional data.

- Improving the Reward Function: We introduce an enhanced reward function that assigns extra rewards to states with valid actions. This is given by Eqn.(4).

$$Reward = \begin{cases} 10^8 & \text{if number of pegs} = 1 \\ 2^{16-\text{no of pegs on board}} & \text{If no. of pegs } >1 \text{ and no possible moves} \\ 2 * 2^{16-\text{no of pegs on board}} & \text{If no. of pegs } > 1 \text{ and at least 1 possible moves} \end{cases} \tag{3}$$

  This modification incentivizes the agent to explore states with more valid action possibilities, leading to reaching final state with less pegs remaining

- Adding Additional Data: While data collection in the initial runs, since the steps are taken randomly starting from the initial position, only a few iterations lead to states with a small number of pegs (e.g., to 2 or 3 or 4 pegs left), and the data set for states with fewer pegs and the steps to reach those sets is quite small. Therefore, before starting the training, we manually augment the training dataset with more examples of states with fewer pegs.

Applying the above modifications, the results obtained were compared, as shown in Table 1. It was observed that the modified reward function leads to a substantially higher occurrence of scenarios with only four pegs remaining compared to the original reward function. Furthermore, when the DQN is trained with both the modified reward function and the manually added data, the instances of achieving two or three pegs left increases to 38%, while it was hardly ever attained with the other configurations.

Pegs Left	Random	DQN With Initial Reward	DQN With Modified Reward	DQN With Modified Reward and Added Data
>=10	6.60 %	0.00 %	0.00 %	0.00 %
9	4.22 %	0.00 %	0.00 %	0.00 %
8	7.44 %	3.80 %	0.00 %	9.01 %
7	13.91 %	22.01 %	0.20 %	0.00 %
6	27.33 %	20.08 %	55.30 %	0.87 %
5	25.71 %	46.43 %	5.87 %	20.15 %
4	11.91 %	7.61 %	38.63 %	31.89 %
3	2.62 %	0.07 %	0.00 %	24.48 %
2	0.24 %	0.00 %	0.00 %	13.61 %

Table 1: Pegs left for 1000 iterations with different situations.

4.3 Classical 7*7 Peg Board

The traditional 7x7 board, as shown in Fig.3, differs from the smaller 5∗5 and 4∗4 boards in size and its distinct cross-shaped configuration. Compared to other board designs, most pegs on this particular board have only two or three adjacent pegs instead of four, which increases the difficulty of eliminating pegs. This is especially true for the farthest pegs, as while solving the board, if one of the furthest pegs gets left behind, it becomes extremely difficult to eliminate it afterwards. This implies that clearing the peripheral pegs can facilitate the game's completion. In other words, it is crucial to encourage peg movement towards the center of the board, where they are more likely to interact with other pegs, thereby increasing the chances of successful elimination.

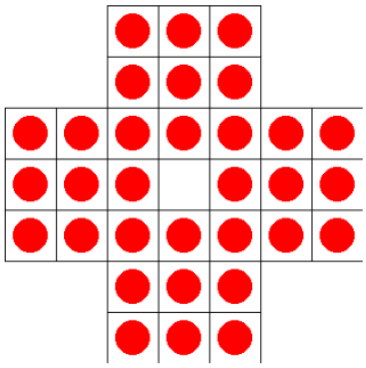

Figure 3: The initial state of a 7*7 Peg Board

Consequently, we have incorporated a spatial component into the existing reward function for this unique

characteristic of the 7x7 board. As a result, the updated reward equation is as follows:

$$Reward = \begin{cases} 10^8 & \text{if number of pegs} = 1 \\ 2^{n-16} + (n * \sum_{i=1}^{n} d_i) & \text{If no. of pegs } >1 \text{ and no possible moves} \\ 2 * 2^{n-16} + (n * \sum_{i=1}^{n} d_i) & \text{If no. of pegs } > 1 \text{ and at least 1 possible moves} \end{cases} \quad (4)$$

where n represents the number of holes on the board, d denotes the distance between a hole and the center of the board, and $\sum_{i=1}^{n} d_i$ signifies the cumulative distance of all holes from the center of the board. Table 2 showcases a comparison of the DQN's performance when implementing various reward functions. As can be observed, when we incorporate the spatial component into the reward function, the probability of reaching a state with fewer than 5 pegs left increases to about 48%. This level of performance is unattainable without the integration of the spatial element.

Pegs Left	DQN (Original)	DQN (Modified)	DQN (Modified + Spatial)	DQN(Modified + Spatial + Manual Data)
>=10	0.03 %	0.00 %	0.00 %	0.00%
9	17.95 %	3.55 %	0.03 %	0.03%
8	42.48 %	8.97 %	1.13 %	0.53%
7	30.95 %	26.77 %	3.83 %	0.38%
6	7.83 %	7.11 %	16.78 %	24.85%
5	0.35 %	51.56 %	29.93 %	21.68%
4	0.40 %	2.04 %	30.53 %	34.20%
3	0.00 %	0.00 %	17.80 %	18.35%
2	0.00 %	0.00 %	0.00 %	0.00%
1	0.00 %	0.00 %	0.00 %	0.00%

Table 2: Pegs left for 4000 iterations with different situations.

4.4 Assisting Human Player

In this context, the model and human player collaborate to complete the game as a single team, with the gameplay alternating between the robot and the human player. The robotic movements consider the human player's subsequent choices, ensuring a smooth transition and optimal strategy. Fig.4 offers an illustration of this cooperative approach.

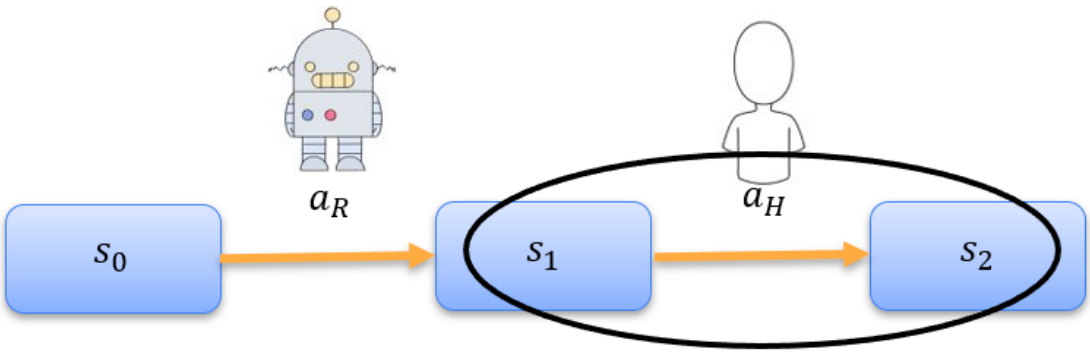

Figure 4: Robot Assisted Play

In Fig.4, s_0 represents the current board state, s_1 denotes the board state after the robot takes its action, and s_2 signifies the board state following the human player's subsequent action. To optimize its performance, the robot must consider both the rewards associated with s_1 and those of s_2 each time it makes a move. Thus, the formula for the robot's decision-making process can be defined as follows:

$$\begin{aligned} a_H &= M(s_1) \\ a_R^* &= \text{argmax}_{a_R} R_R\left(s_0, a_R, M\left(s_1\right)\right) \end{aligned} \quad (5)$$

Here $M(s_1)$ is the action taken by human basis the Human model defined in section 3.2. It is assumed that the robot has the model of human. Since the robot is assisting the human, the robot's reward function can be defined as follows:

$$R_R = R_R\left(s_0, a_R\right) + R_H\left(s_1, a_H\right) \tag{6}$$

The key concept here is that maximizing the reward in state s_1 is not necessarily the optimal solution for the robot. Instead, it must ensure it gets the highest score in state s_2. Thus, the robot's behavior may not produce the highest reward at s_1 but a significantly higher reward at s_2. And when using Boltzmann-based human models in the human behavior selection process, human player behavior is not always guaranteed to be optimal. As a result, the robot may encounter different rewards in the same state and action. Therefore, the robot needs to consider this to minimize the occurrence of valid options that human has the possibility to choose, which also could cause the game to end prematurely.

Table 3 displays the performance of this model in conjunction with different beta values, representing the human model rationality. It can be observed that when the human player makes completely random choices, the model's assistance generally results in approximately 8 or 9 remaining pegs. As the human model rationality increases, the overall number of pegs left on the board decreases. On the other hand, Table 4 shows that the robot is able to assist better when the human is more rational, as there is hardly any difference in results when the human is highly irrational.

Pegs Left	$\beta = 0$	$\beta = 0.5$	$\beta = 0.8$
>=10	34.10%	3.40%	1.40%
9	32.10%	8.70%	3.20%
8	12.30%	13.40%	6.80%
7	10.90%	11.20%	8.10%
6	6.40%	35.10%	10.20%
5	2.10%	16.50%	66.30%
4	1.10%	11.00%	3.10%
3	0.90%	0.40%	0.80%
2	0.00%	0.20%	0.00%
1	0.00%	0.00%	0.00%

Table 3: Pegs left for 1000 iterations with different human rationality

Pegs Left	$\beta = 0$ (Only H)	$\beta = 0$ (R+H)	$\beta = 0.8$ (Only H)	$\beta = 0.8$ (R+H)
>=10	38.1%	34.1%	1.5%	1.4%
9	27.2%	32.1%	3.4%	3.2%
8	16.1%	12.3%	11.9%	6.8%
7	11.1%	10.9%	18.7%	8.1%
6	3.0%	6.4%	35.4%	10.2%
5	1.9%	2.1%	27.7%	66.3%
4	1.7%	1.1%	1.1%	3.1%
3	0.9%	0.0%	0.3%	0.8%
2	0.0%	0.0%	0.0%	0.0%
1	0.0%	0.0%	0.0%	0.0%

Table 4: Pegs left for 1000 iterations with different human rationality

5 Conclusion

This project attempts to solve the Peg Solitaire problem using Deep-Q Networks and assist a human player playing the game. It was observed that by improving the reward function and data augmentation, better results were achieved. However, since the reward function was changed as we moved to higher dimensional boards, the solution is not generalized. A generalized reward function to solve boards of different configurations

could be designed. Further, instead of linear layers in the hidden layers of the neural network, convolutional layers could be tried, which were used to solve the 2048 game in [3].

Further results suggested that the robot was able to assist humans with higher rationality better to achieve better results. However, when the human is less rational, the performance is not great. Future research could focus on addressing this challenge to further enhance the model's ability to adapt to various levels of human rationality, ultimately leading to more effective AI-human collaborations in complex tasks.

References

[1] Joseph Barker and Richard Korf. Solving peg solitaire with bidirectional bfida. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 420–426, 2012.

[2] Masashi Kiyomi and Tomomi Matsui. Integer programming based algorithms for peg solitaire problems. In *Computers and Games: Second International Conference, CG 2000 Hamamatsu, Japan, October 26–28, 2000 Revised Papers 2*, pages 229–240. Springer, 2001.

[3] Naoki Kondo and Kiminori Matsuzaki. Playing game 2048 with deep convolutional neural networks trained by supervised learning. *Journal of Information Processing*, 27:340–347, 2019.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.