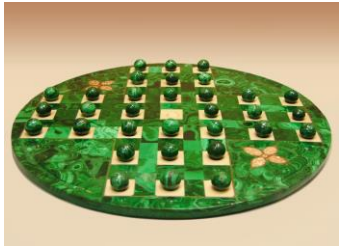
Peg Solitaire

Subham Swastik Samal, Zijian Ding

Peg Solitaire



Peg Solitaire, also known as Solo Noble or Brainvita, is a board game for one player involving movement of pegs on a board with holes. The game is known simply as Solitaire in the United Kingdom where the card games are called Patience. It is also referred to as Brainvita (especially in India).

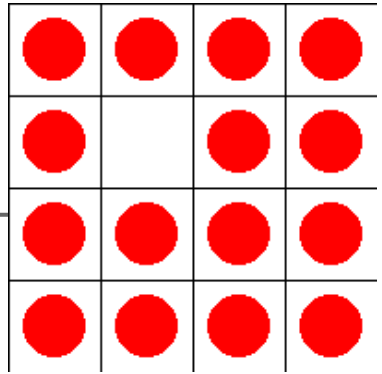
The objective is to empty the entire board except for one solitary peg left. Valid moves involve jumping a peg (horizontally or vertically) over an adjacent peg into a hole two positions away and then removing the jumped peg. Diagonal moves are not allowed.

https://en.wikipedia.org/wiki/Peg_solitaire#/media/File:Solitaire_01.jpg
https://en.wikipedia.org/wiki/Peg_solitaire#/media/File:Spielzug_von_Soli
t%C3%A4r.gif

Project Aims

- Implement a Deep-Q Network to solve different board configurations.
- Implement a methodology to assist a human playing the game. This would work as the following: The bot and the human take turns; the bot moves first followed by the human. The bot's aim is to make the optimal move considering the human's possible moves.

Environment



State: An array of 4 x 4 represents the state.

- 0: No peg present in the position
- 1: Peg is present in the position

Action: A tuple of size 2 representing the start and end positions

- Action ((4,2),(2,2)) represents peg from position (4,2) to (2,2) over (3,2)
- Action space consists of all possible actions (e.g.: 40 for a 4x4 board), but only a few valid at each state

Reward: Reward is defined as:

- 100 if the minimum number of pegs is achieved
- Else, Reward = $-2^{number\ of\ pegs\ on\ board}$

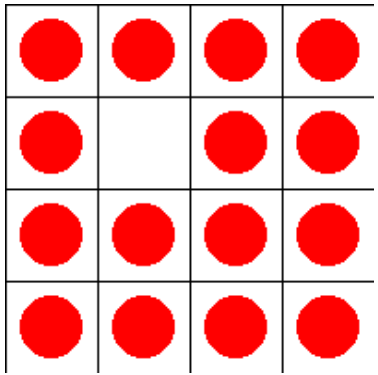
Deep Q-Learning

Discrete Actions: Peg solitaire's discrete actions suit a DQN framework
Objective: To learn an optimal policy that maximizes the expected discounted sum of rewards

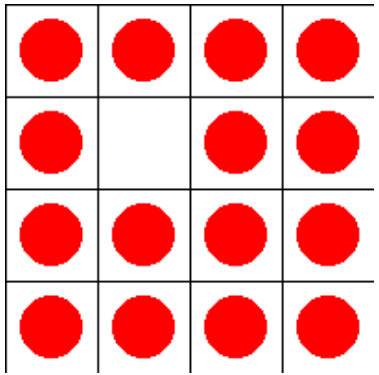
While running

- $a \leftarrow$ argmax $Q(s, a)$
- Add s^t, a^t, r^t, s^{t+1} to memory
- If len(memory) > batch_size
  - $Sample\ batch\ of\ s, a, r, s'$
  - $Qtarget \leftarrow r + \gamma \cdot Q'(s')$
  - $Qexpected \leftarrow Q(s)$
  - $\mathcal{L}(\theta) \leftarrow ||Q_{target} - Q_{expected}||$
  - Q' $\leftarrow$ weights closer to Q

4 x 4 Peg Board

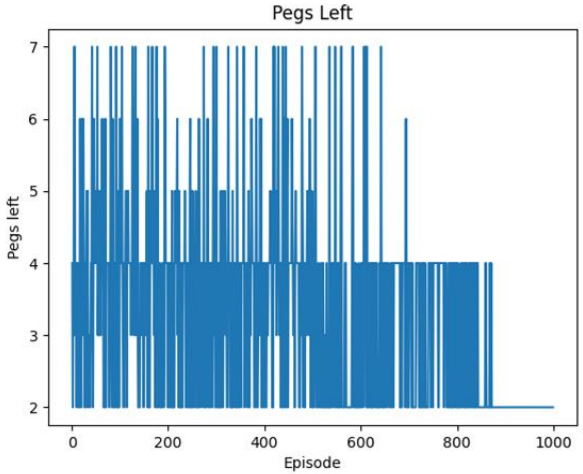


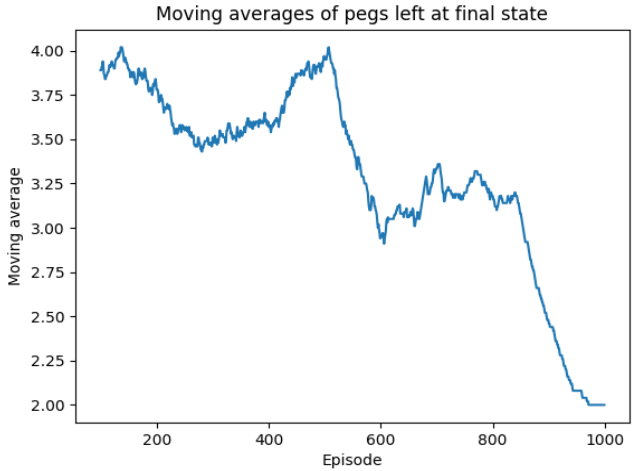
Starting position with 15 pegs



Solution with DQN

Solving with DQN





Higher Dimensional Boards

- ***Board configurations***: The number of possible board configurations increases exponentially with the number of pegs, making it challenging to explore and learn from all possible states.

- ***Long-term dependencies:*** The reward for a particular move is not immediately evident, and the agent may have to make a series of moves to reach a desirable state. This can make it difficult for the agent to learn an optimal policy, as it needs to consider the long-term consequences of its actions.

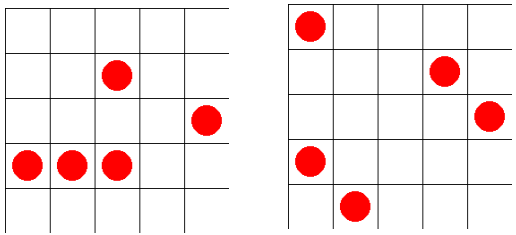
Improving the Reward Function (5 X 5)

Added extra Reward to states which have valid actions:

10^8 if the minimum number of pegs is achieved

Else:

- If state has valid actions: 2 x $2^{(16-number\ of\ pegs\ on\ board)}$
- ***Else***: $2^{(16-number\ of\ pegs\ on\ board)}$

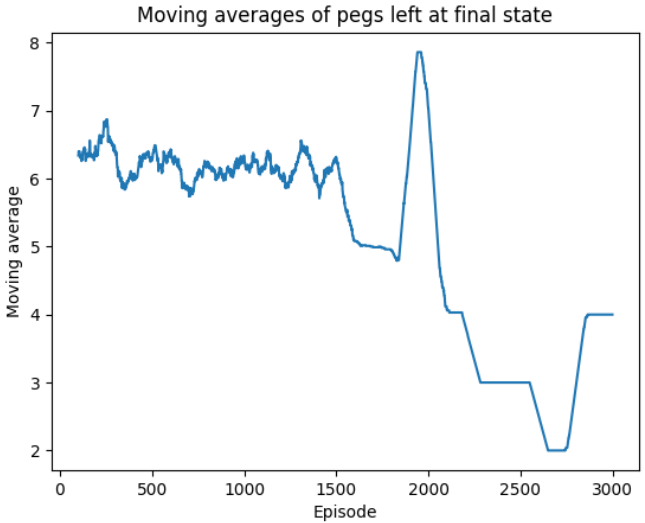


$R_1 \quad = \quad 2R_2$

Adding Additional Data (5 X 5)

- Randomly starting from initial state doesn't lead to many steps with very less pegs (2,3 etc.)
- Thus, we add more data for states with less pegs manually to get more data to train

Results (5 x 5)



DQN With Modified Reward and Added Data

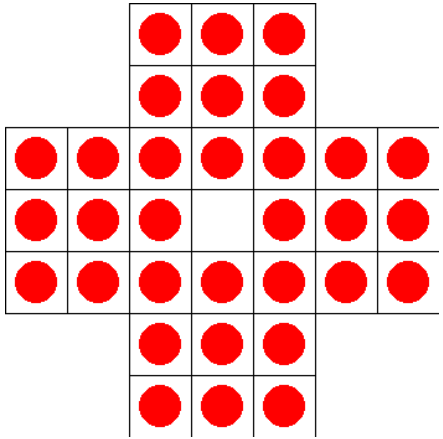
1000 iterations each:

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%

Classical Board

New reward term =

$$\text{Modified Reward} + n * \sum_{i=1}^{n}(d_i)$$



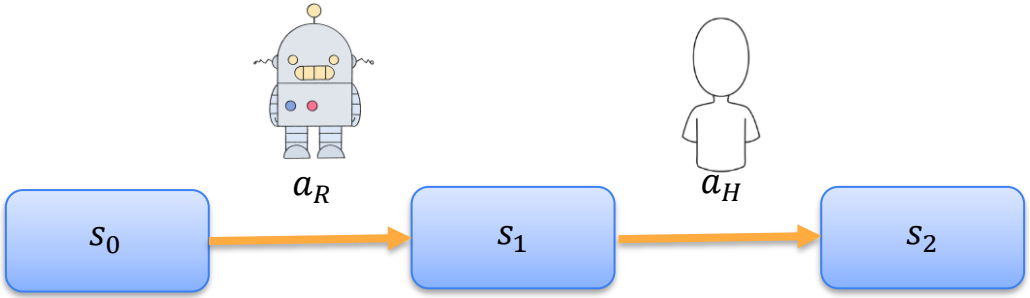
- n: the number of empty holes in the board
- d: the distance of the hole from the board's center

Classical Board

4000 iterations each:

Pegs Left	Random	DQN With Initial Reward	DQN With Modified Reward	DQN With Modified Reward and Added Data
>=10	24.98%	0.03%	0.00%	0.00%
9	18.33%	17.95%	0.03%	0.03%
8	28.03%	42.48%	1.13%	0.53%
7	17.10%	30.95%	3.83%	0.38%
6	8.08%	7.83%	16.78%	24.85%
5	3.05%	0.35%	29.93%	21.68%
4	0.45%	0.40%	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%

Assisting Human Player

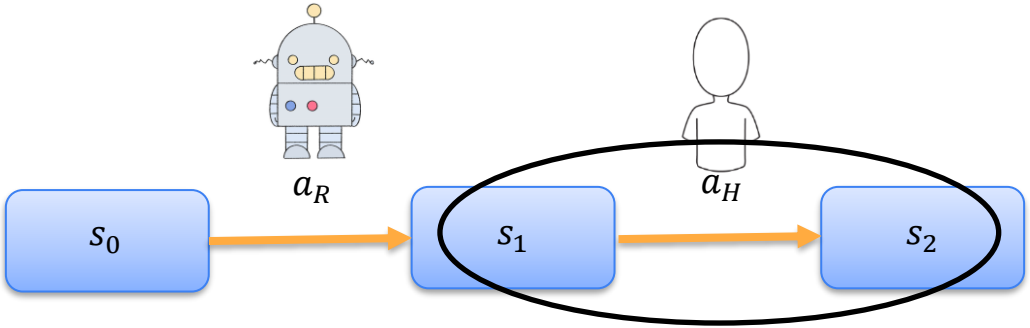


Human action follows Boltzmann Model

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

Q is taken to be same as human's reward function, which is normalized to 0 to 1 range

Assisting Human Player



Robot takes action considering the human model:

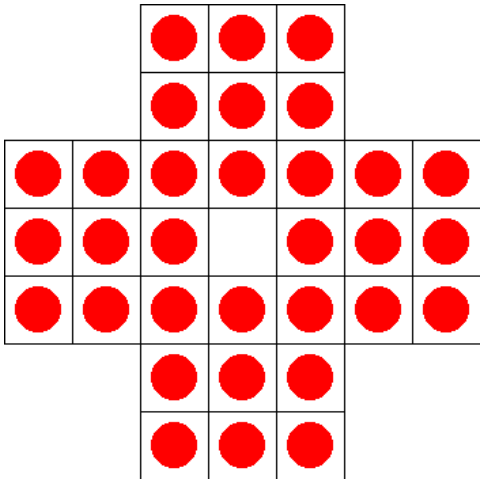
$$a_H = M(s_1)$$

$$a_R^* = argmax_{a_R} R_R(s_0, a_R, M(s_1))$$

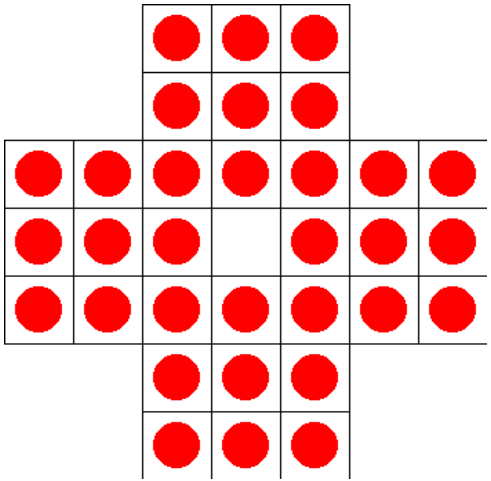
Robot's reward is given as:

$$R_R = R_R(s_0, a_R) + R_H(s_1, a_H)$$

Assisting human player



Only human model with β equal 0.7



Human with robot

Results with different β

1000 iterations each

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%