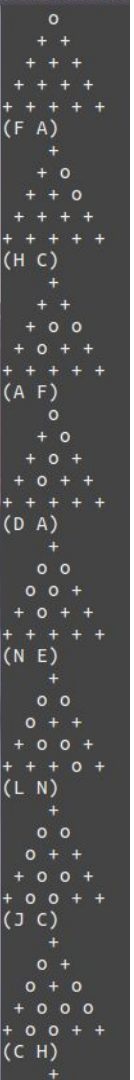
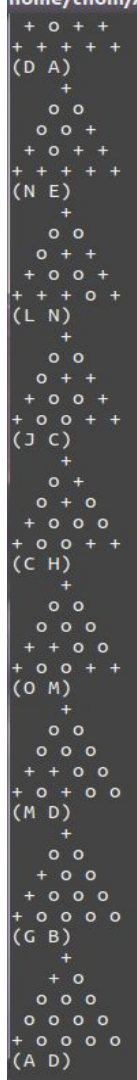


Peg Solitaire

A game you do not like, and I don't like either.



Original goal

The task is to create a machine to learn to play peg solitaire, a game that can be solved by mathematical functions, but also by reasoning and critical thinking. The game is simple, certain pegs can jump over others as long as there is a space on the other side. When a peg has been jumped over, that peg will be removed from the board. The goal of the game is to end with the fewest possible pegs on the board, after no moves be made any longer. The peg Solitaire solving machine will learn through playing the game the tricks to solving it and getting the best possible outcome. Hopefully the machine will also after completion be able to solve the game on each of the many different boards that peg solitaire can be played on.

Updates to the program:

It’s been a while.. What has been added?

1)Reverse-play functionality.

2)One-way Graph generation.

3)Addition of weights to edges.

4)Board manipulation ability.

5) Trials and Solution.

Rotations

```
[7]> (display board)
    +
   o +
  + o +
 + o o o
+ + o + o
NIL
[8]> (display (left-rotate-triangle board))
    +
   + +
  o o +
 + o o o
o o + + +
NIL
[9]>
```

Weights

```
Node : #S(HASH-TABLE TEST FASTHAS
Edges : (I B), Weight : 4
Edges : (M D), Weight : 4
Edges : (K D), Weight : 6

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (M K), Weight : 6

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (F M), Weight : 6

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (D F), Weight : 6

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (N L), Weight : 6

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (K M), Weight : 6

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (M D), Weight : 6

Node : #S(HASH-TABLE TEST FASTHAS

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (F M), Weight : 1

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (D F), Weight : 1

Node : #S(HASH-TABLE TEST FASTHAS
Edges : (K D), Weight : 1
```

Reverse

```
[2]> (reverse-auto-play)
    o
   o o
  o o o
 + o o o
o o o o o



0: Move G to I
1: Move G to B
Selected Move: 0
    o
   o o
  o o o
 o + + o
o o o o o



0: Move I to B
1: Move H to C
Selected Move: 1
    o
   o +
  o + o
 o o + o
o o o o o



0: Move I to G
1: Move E to L
2: Move C to J
Selected Move: 1
    o
   o +
  o o o
 o + + o
o + o o o
```

The problem with peg solitaire

It's really a puzzle, not a game. However, I wanted to model and solve it as if it was a game.

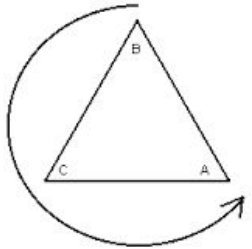
It’s a puzzle that does not like randomness. With random movement there comes a 50% chance of it becoming completely insolvable after first two moves.

Many solutions, but based off of a few main patterns the machine would need to search for.

Heuristics

1) Many games have symmetrical and/or rotational boards. Peg solitaire has both but I decided to limit it to just rotation, as that would allow the machine to apply already found solutions to boards rotated left and right.
2) A human player is able to look at the game they are playing and try to find a solution from the current board. This machine looks for solutions and plays backwards in hopes of finding connection with start and finish.

Correct Start -> Incorrect End
Correct End -> Incorrect Start



In my original plan I wanted to add some kind of foresight feature to the game, where it would look a few moves past its last one to see if a move was better or worse than another.

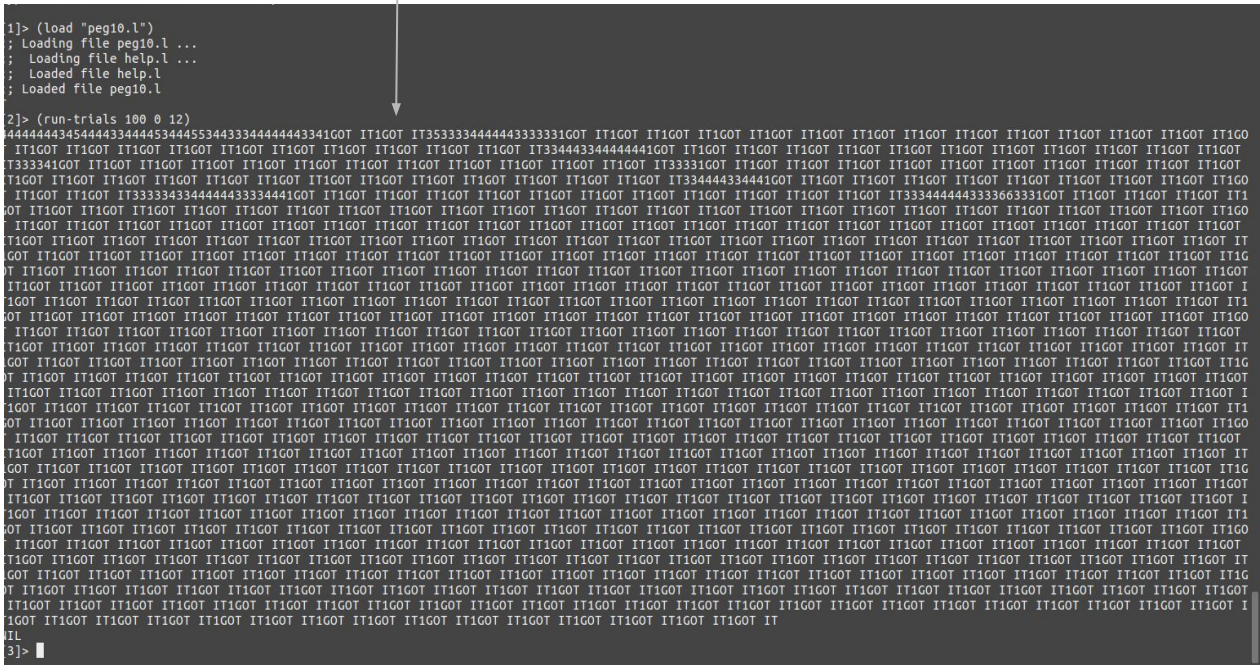
Results

Here the game was run forwards 100 times and backwards 100 times in order to find a solution to the puzzle with the start position at the top.

This was just a guarantee, the machine was able to find the solution in less than 30 plays of each.

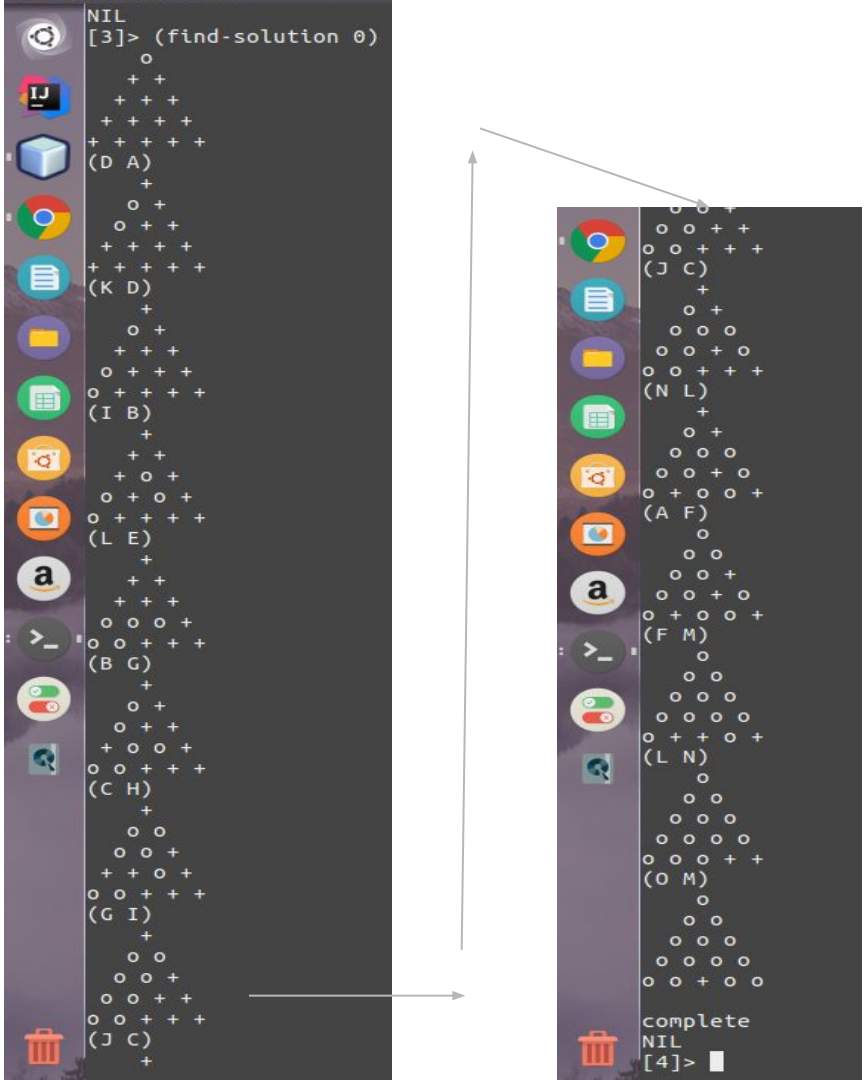
Time to solve averages about 10 seconds.

Solution found here during run.



Cont.

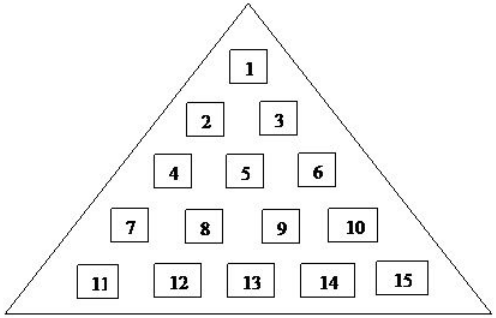
Although the machine came up with multiple ways of solving, the solution returned is...



If not one then two.

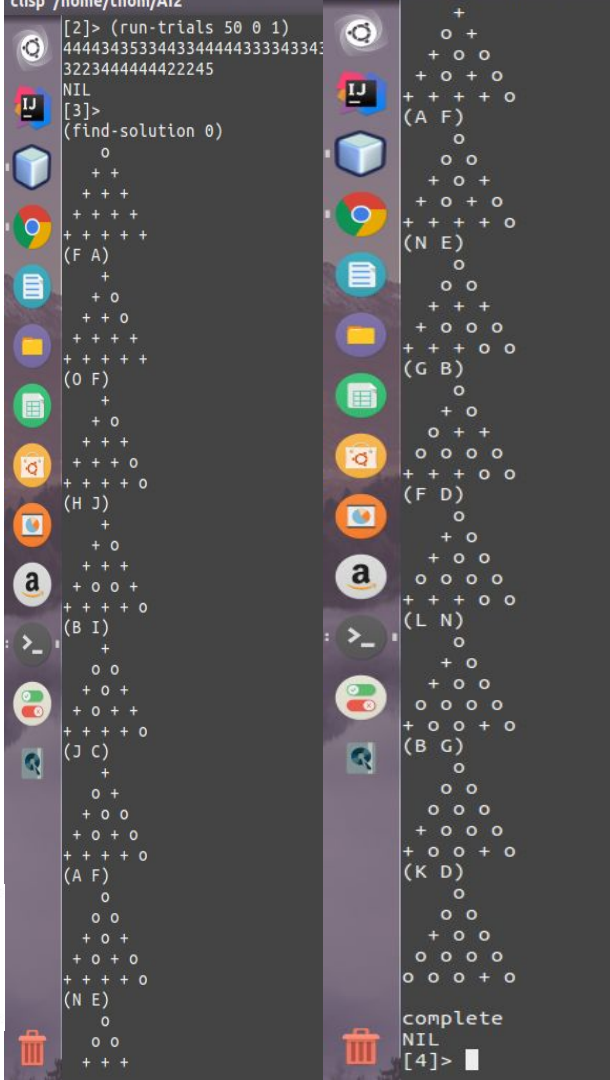
The machine is designed to find the solution from a to b, not from any start to any finish, as some are impossible.

Each start position has a solution. The machine takes in a requested start and end. If it is possible to find a path, the machine will, if it isn't, the machine will give the path to the best possible outcome (lowest final peg count).



Thanks to rotation, the start positions can be simplified to four that cover each one.

1	1, 11, 15
2	2, 3, 7, 12, 10, 14
4	4, 6, 13
5	5, 8, 9



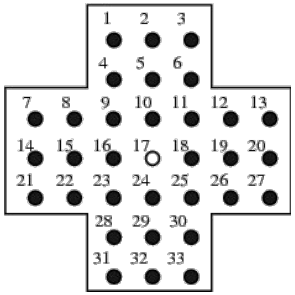
What else could be done with this?

I originally planned to apply this machine to the cross or ‘english’ style solitaire board, but I already really knew that it would work similar to the board I already am using.

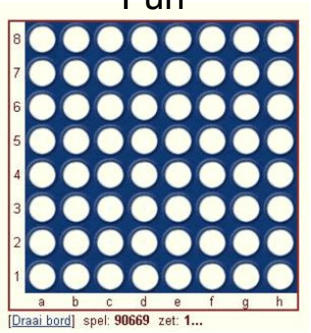
Since I modeled the solution to work as a game similar to the tictactoe exercise I wanted to apply this program to a game.

What I wanted to do was to convert it to a connect four learning machine using the functions I have already built, to see how it would hold up against competition instead of a challenge.

Boring



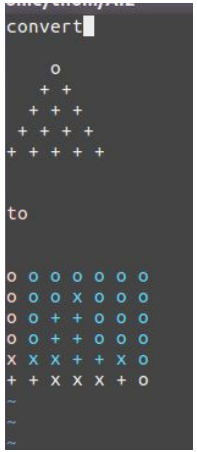
Fun



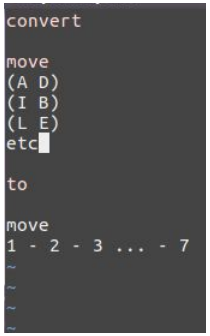
Converting to connect four

Of course it would have to be trained by semi-supervised learning, or else with randomized play the machines might make some unnatural habits. After some supervision, if two someone intelligent machines learned from each other, the peg solitaire learning machine might become a good connect four player.

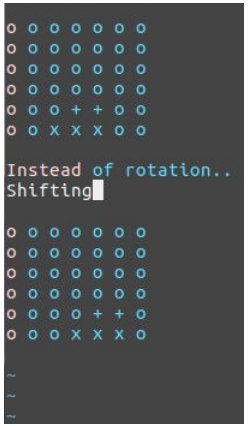
Board



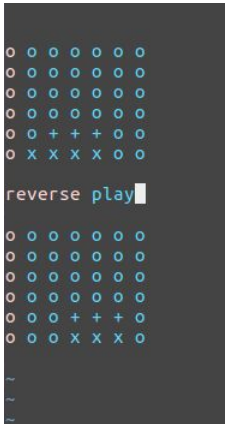
Moves



Manipulation



Reverse play



References / Questions

http://www.wikihow.com/Win-the-Peg-Game

http://www.joenord.com/puzzles/peggame/

https://softwareengineering.stackexchange.com/questions/264474/generate-algorithm-to-solve-peg-solitaire

http://blogs.sas.com/content/operations/2015/03/11/how-to-solve-puzzles-peg-solitaire-with-optimization/

http://www.cs.lamar.edu/faculty/osborne/4172/PegSolitaire/Iris%20Garcia/Peg%20Solitaire%20with%20Depth%20First%20Search.pptx

http://www.mathematische-basteleien.de/solitaire.htm