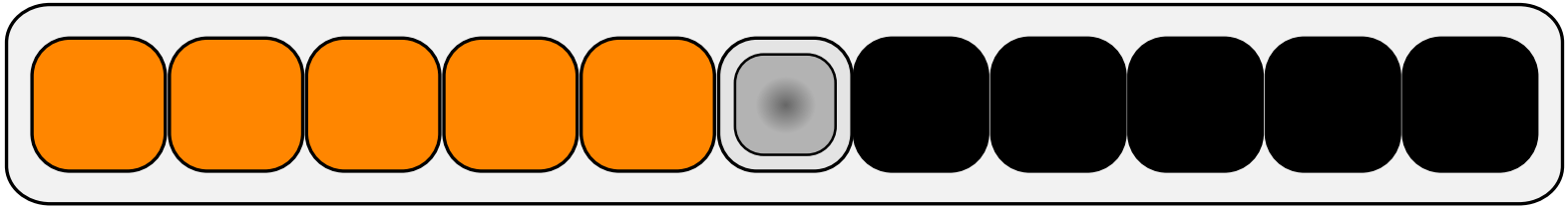
Java Assignment: Peg Board

Shaun Hegarty

Dublin City University

April 23, 2015

Problem Definition: The Peg Board



Consider a board with holes in which coloured pegs can be placed. For this problem we have:

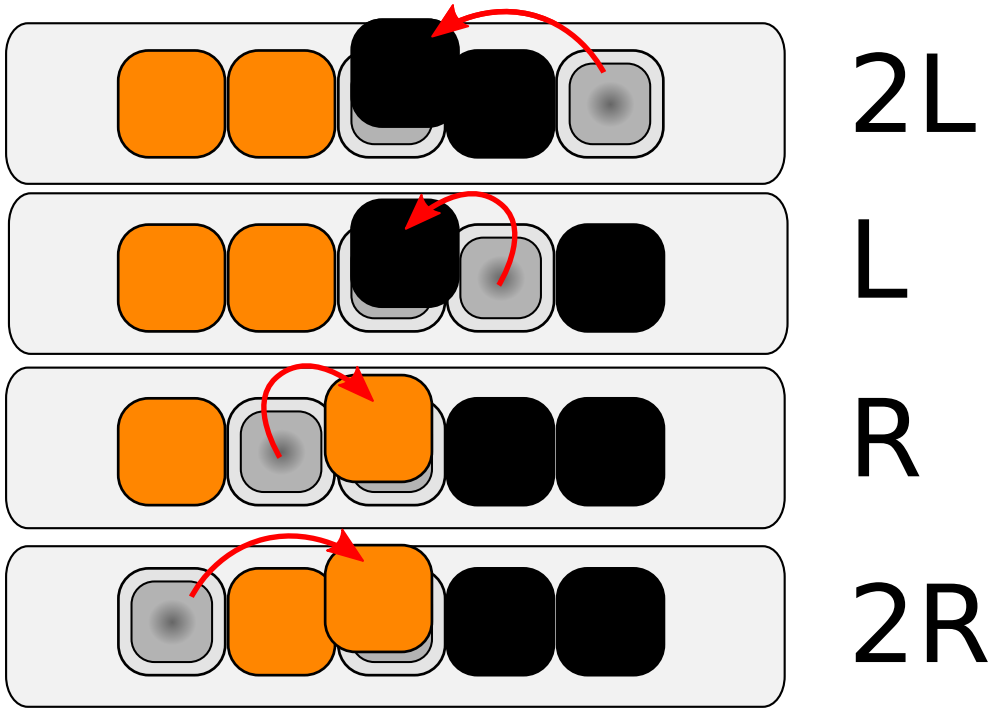
- pegs of two colours
- an equal number of each
- a board with enough spaces for all pegs
- one additional space which allows for movement

Rules:

- A peg adjacent to the space may move into it
- A peg may jump over a peg adjacent to the space into that space

Design: Breaking it down

The aim is to move all pegs of one colour to the other side of the board in the minimum number of steps while following the rules.



Given the rules we can see that there are only four moves possible at any one time.

Design: Approaching an Algorithm

The problem was solved manually numerous times to determine a pattern.

- Initial algorithm was effective but inefficient
- However, continued efforts revealed a best case algorithm

I determined that the minimum number of steps for a board of a given size could be calculated with following definition:

$$f(n) = \begin{cases} f(n-2) + n & \text{if } n \geq 3 \text{ and } n \text{ odd} \\ 0 & \text{if } n < 3 \end{cases}$$

where n is the number of spaces on the board.

Design: The Algorithm I - Conditions and Rules

Starting conditions:

- The board is set with the hole in the middle
- Pegs on either side will be made of one colour only.
- We have a predecided starting direction/colour of which we keep track.

When determining the next move to be made the algorithm uses the following rules:

- Pieces of the colour on the left, can only make moves to the right, and vice versa
- Thus the current direction of moves is tied to the colour
- Prioritize moves of size two, then one.
- If no move can be made in the current direction, change direction
- Direction is stored simply as +1 or -1.

Design: The Algorithm II - The Pseudocode

The solving loop:

```
while (!finished)
    nextMove()
```

nextMove() method:

if (current colour can jump over a different colour into hole (in an allowed direction))
    move(2 * direction)
else if (current colour can move into the hole & be placed adjacent to a different colour (in an allowed direction))
    move(1 * direction)
else if ((hole is at edge of board & direction is away from the edge of the board)
OR (pieces beyond the space are all in their final positions))
    move(1 * direction)
else
    change direction

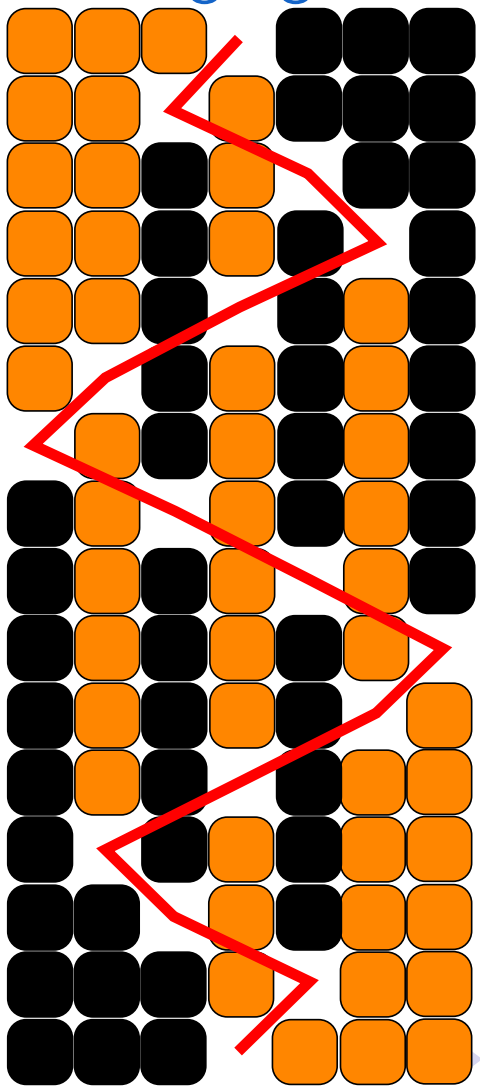
Design: The Algorithm III - A New Pattern Emerges

Another pattern

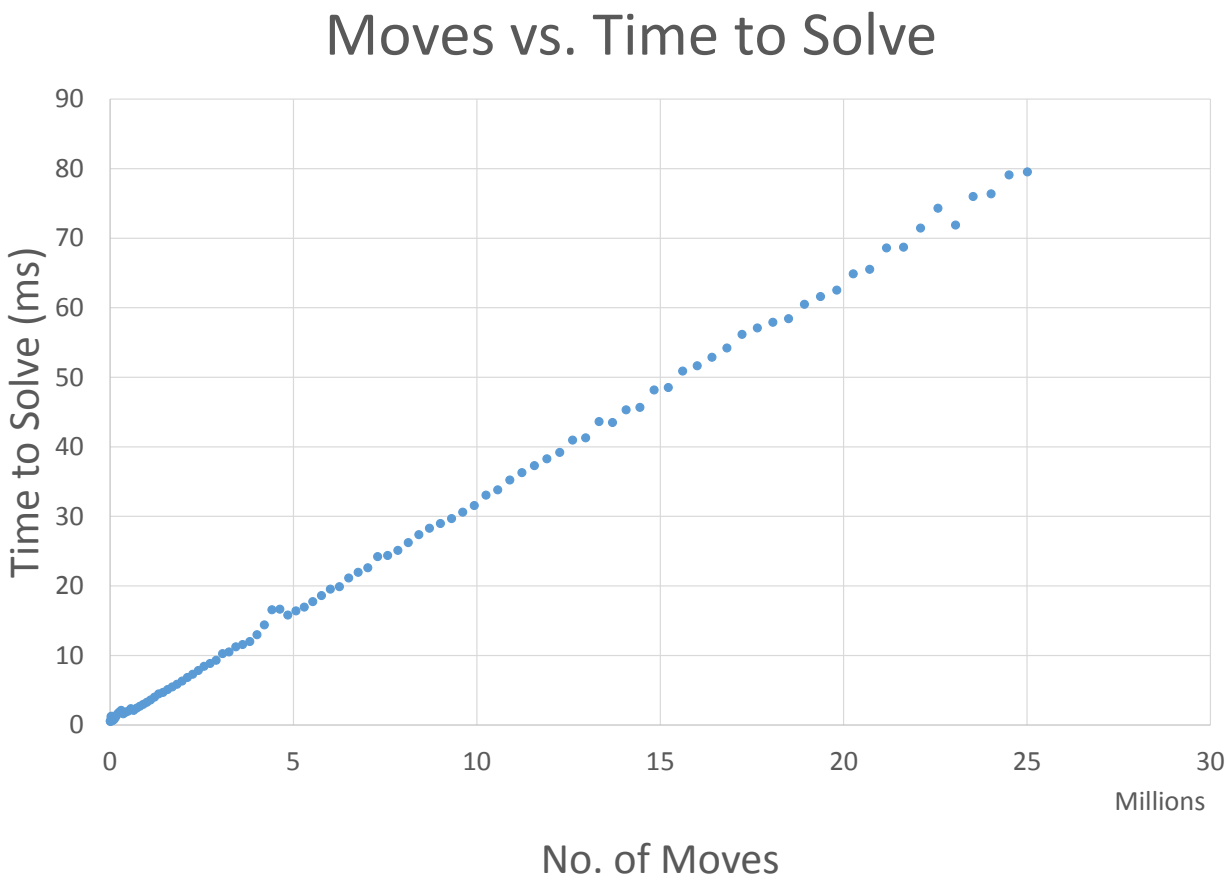
Having run my algorithm a number of times and printing the board with each step, this symmetric zigzag pattern becomes apparent.

My solution uses the hole as a reference frame and ignores everything outside of that frame. If instead I took the perspective of the board as a whole I would have taken this approach

The Zigzag



Some Analysis



The time to complete the solution scaled linearly with the number of moves for large board sizes.

Program Demo and that little bit more

To complement the solution I created a GUI in which the user can both play the game, or watch the solution presented step by step.

The user can click on any piece, if this piece can make a valid move into the hole, it will.

The user can also increase or decrease the size of the board, and reset the board at any time.

Limitations & Future Work

Limitations

- The algorithm does not solve the puzzle for an incomplete non-optimal solution.

  i.e. if the algorithm was tasked finishing a game in which someone had made a move which would not lead to a solution in the minimum number of steps, it would fail

Future Work

- Make implementations of alternate algorithms (the zigzag)
- Make the algorithm work for any starting point
- The problem can be expanded in numerous ways, such as to a 2D or 3D board which still only has one space for movement

Reflection: What did I learn?

- Need to step back and look at the bigger picture more often.
- Sometimes miss more obvious patterns by breaking things down too quickly.

- I used this assignment as a vehicle to learn about the Graphics2D library in Java.
- In the process I learned about key bindings, mouse listeners and even a little bit about threads and concurrency.

Reflection: What could I have learned?

What could I have learned?

- I only acquired the information I needed to make threads work in my program
- I know I didn't touch on vast areas of concurrency such as
  - multithreading
  - resource locking
  - synchronisation

  and more to tackle before I would consider myself to be approaching competency in that area.
- I toyed with animating the blocks moving to each new location. I feel it was feasible but (as far as I could tell) I would have had to change my implementation of the pegboard class to make it work effectively.

Conclusion

I successfully created an application which could solve the given pegboard problem.

Additionally I created a GUI which allowed the user to attempt to solve it themselves or watch the program solve it.

I learned a great deal of new content for the latter part as well as some areas in which I can improve myself.

Resources Used:

- ZetCode. *Hit testing, moving objects, 2013*[Online]. Available from: http://zetcode.com/gfx/java2d/hitmove/ [Last Accessed 23 April 2015]
- Oracle Java Documentation, *How to Use Key Bindings, 2015*[Online]. Available from: http://docs.oracle.com/javase/tutorial/uiswing/misc/keybinding.html [Last Accessed 23 April 2015]