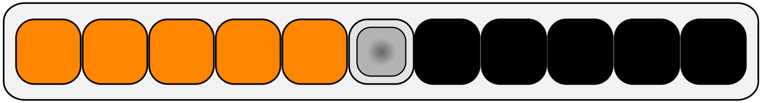
Java Assignment: Peg Board

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

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

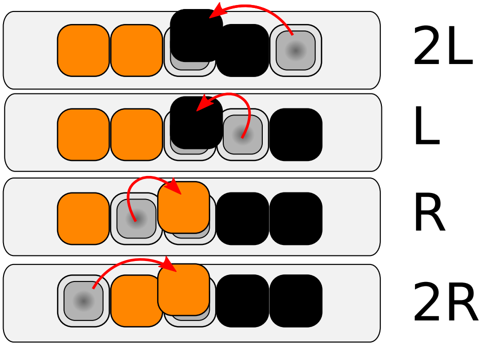
Problem Definition: What makes it interesting?

For me:

- Pure problem solving exercise
- Making a simple game
- Some ideas could be extended to 2D and various checkerboard games like draughts

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

A formula for the minimum number of steps:

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

where n is the number of spaces on the board.

Design: The Algorithm I - Conditions and Rules

Starting conditions:

- Pegs on either side of the space will be made of one colour only.
- We have a predecided starting direction/colour of which we keep track.

Rules when determining the next move:

- Each colour can only move in one direction
- Prioritize moves of size two, then one.
- If no moves change direction

Design: The Algorithm II - The Pseudocode

The solving loop:

```
while (not 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
```

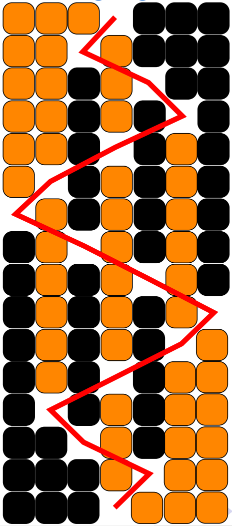
Design: The Algorithm III - A New Pattern Emerges

Another pattern

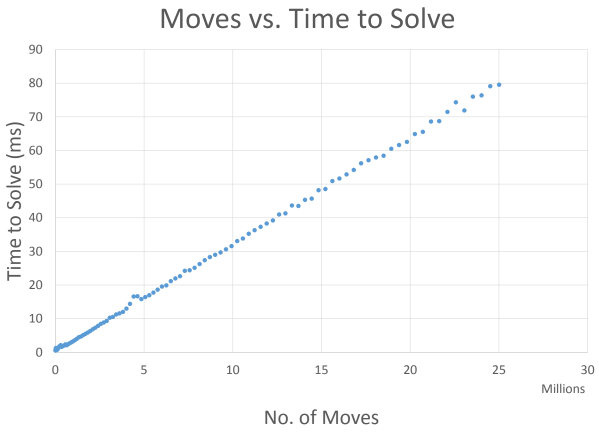
When running the program this symmetric zigzag pattern becomes apparent.

I framed my logic around the space. Looking at the board as a whole may have led to 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 given an incomplete non-optimal solution.

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 patterns by breaking things down too quickly.

- Graphics2D library in Java.
- In the process I learned about key bindings, mouse listeners, threads and concurrency.
- Made better of use of Github for version control.

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
- 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]

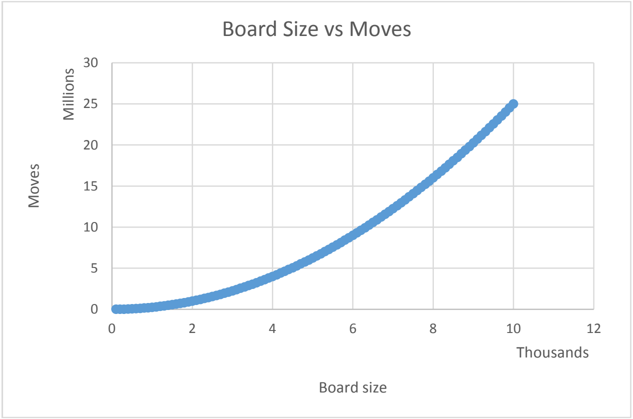
Minimum moves

Non recursive definition of the minimum moves function:

$$(floor(\frac{n}{2})+1)^2-1$$

where n is the number of spaces on the board, is odd and ≥ 3.

Board Size vs. Moves


Board Size vs Moves
Moves
Millions
Thousands
Board size