Internship Report

By Yuuv Jauhari

The task given was to create a 5x5 chess board ( can be upscaled), with two knights starting on opposite corners of the board. Whenever a knight moves from its place, the previous place gets blocked and neither knight can move to the aforementioned square.

**Abstract**

A chess board with two knights on opposite sides of the board was created which block the previous positions of the knight till the the last knight has no available positions to move to.

We created a program using Java to simulate this exact scenario over multiple outcomes.

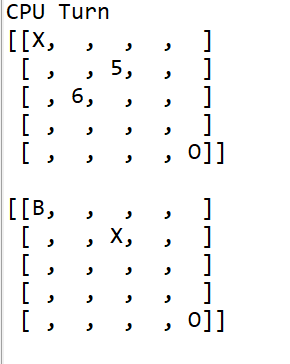
**Background**

A mobile program was created by students of Cluster Innovation Centre, Delhi University that had a 4x4 chess board with two knights at opposite ends and previous place being blocked, similar to our implementation, with both knights being manually controlled by the user.

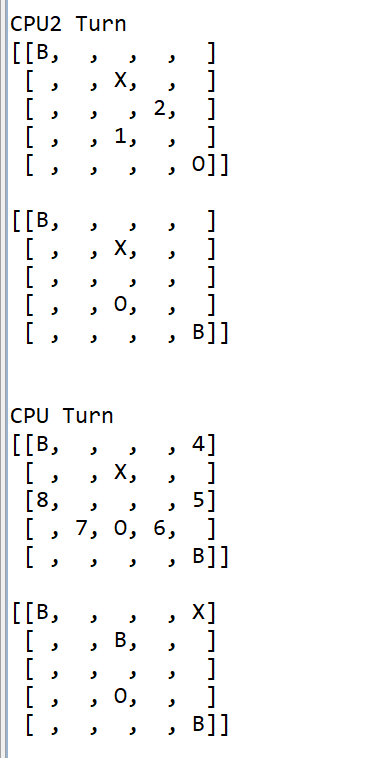
However, with our implementation, we expand on this by having an expandable chess board and AI to play the game to simulate the game quickly and get different results.

**Original Problem**

Originally designed on a 5x5 board, a demonstration of this would be



The ‘X’ and ‘O’ are the two knight pieces on the opposite sides of the board, the numbers display the potential places to move the chess piece to. The places display the ‘B’ to show that the place has been blocked for both the playing pieces.



As we can see, the places previously occupied by the knights have been blocked. By this pattern we keep blocking spaces until the knights have no place to move, which ends the game

The goal was to create a program and predict the possible outcome of moves for the two knights in a expanding moveset tree fashion. It should be noted that the initial moveset follows symmetry as both knights only get two places to move on the first turn.

**Code**

The pseudocode for such is as follows:

Set up a 5x5 character grid using square brackets, setting pieces in opposite corners, being 0.0 and 5.5

While (gamestop = false){

stepcounter += 1;

For alternate steps

If ( stepcounter%2 == 1){

- Setting up moves

- Since we are taking the side left move, from relative position to the knight, the coordinates for the moves would be

(X-2, Y-1)

viable1 = **true**;

move1X = XmoveX-2;

move1Y = XmoveY-1;

-To stop the overflow of the move coordinates outside the 5x5 board

If it overflows, the viability of the move is set to false

**if** (move1X < 0) {

viable1 = **false**;

}

**if** (move1Y < 0) {

viable1 = **false**;

}

**if** (move1X > 4) {

viable1 = **false**;

}

**if** (move1Y > 4) {

viable1 = **false**;

}

-Similarly we can do the same for other moves

Note that Y adds to go down and subtracts to go up as we are using a char grid

Topleft Move (2) : (X-1,Y-2)

TopRight Move (3) : (X+1,Y-2)

SideRight Move (4) : (X+2,Y-1)

SideRight Bottom Move (5) : (X+2, Y+1)

BottomRight Move (6) : (X+1, Y+2)

BottomLeft Move (7) : (X-1, Y+2)

SideLeft Bottom Move (8) : (X-2, Y+1)

-Now after setting coordinates, we check the viability of each move to be true and then if the place has already been blocked or not.

If the move is both viable and the place has not been blocked, then we set true for the move and repeat the code for all other moves

-After which we print the grid

**if**(viable1 == **true** && grid[move1Y][move1X] != 'B'){

grid[move1Y][move1X] = '1';

show1 = **true**;

moved1 = **true**;

}

-After that, we clear every displayed move by, repeating 8 times

**if**(show1 == **true**) {

grid[cmove1Y][cmove1X] = ' ';

}

-Now we put the character (X in this case), into the move which is the lowest to highest (i.e 1, if not 1, then 2 and so on…), as suit, we repeat this 8 times, the second piece of code is to stop the game if it encounters ‘O’, i.e the other player

**if**(moved1 == **true** && grid[move1Y][move1X] != 'O') {

grid[move1Y][move1X] = 'X';

moved = **true**;

XmoveX = cmove1X;

XmoveY = cmove1Y;

}

**if** (cmoved1 == **true** && grid[cmove1Y][cmove1X]== 'O') {

grid[cmove1Y][cmove1X] = 'X';

gamestop = **true**;

}

* This concludes the movement of the first piece
* Using an else statement to set up an alternating system of moves for ‘X’ and ‘O’ as X moves on odd instances of stepcounter, hence ‘O’ will move on even instances.
* For code of ‘O’, we simply repeat the code for X with new variables, and replacing and switching certain instances as necessary.

**Conclusion**

The Java implementation of this game and subsequent automation of both players helps us to easily cycle through possible outcomes of the match of two players to find different possibilities of moves.

Further Implementions

- Machine learning can be used to analyse patterns of winning and find optimal strategies

-By simple code commenting, make it a player vs CPU game to play