Chess and Python revisited

- there exists a PyGame project ⟨http://www.pygame.org/project-ChessBoard-282-.html⟩ which draws a chess board and allows users to make FIDE legal moves (by clicking on the mouse)
  - it does not play against you
  - the graphics are pretty and it also has a neat feature that once you highlight a piece it proceeds to highlight all legal destination squares
- conversely here ⟨http://floppsie.comp.glam.ac.uk/download/m2/m2chess-0.3.tar.gz⟩ is a command line chess game which plays a very basic game of chess
  - however it has a horrible command line interface
  - its redeeming virtue is that it is easy to beat!

pexpect

- recall that the pexpect module can be used to allow Python to control command line programs
  - it should be possible to modify the `ChessBoard` package to use `pexpect` to control the `m2chess` program

Running m2chess from the command line

```
$ m2chess
Enter Stage Of Game :  opening

Is the present Board in initial position? yes

White - Computer or Human (c/h) ? h

Black - Computer or Human (c/h) ? c

The Board :
      a b c d e f g h
    +-----------------+
8   | r n b q k b n r |   8
7   | p p p p p p p p |   7
6   | . . . . . . . . |   6
5   | . . . . . . . . |   5
4   | . . . . . . . . |   4
3   | . . . . . . . . |   3
2   | P P P P P P P P |   2
1   | R N B Q K B N R |   1
    +-----------------+
      a b c d e f g h
```


Running m2chess from the command line

```
Please enter move: e2e4

My move is:  E7-E5     0
The Board :
      a b c d e f g h
    +-----------------+
8   | r n b q k b n r |   8
7   | p p p p . p p p |   7
6   | . . . . s . . . |   6
5   | . . . . p . . . |   5
4   | . . . . P . . . |   4
3   | . . . . . . . . |   3
2   | P P P P . P P P |   2
1   | R N B Q K B N R |   1
    +-----------------+
      a b c d e f g h
Please enter move:
```

Running m2chess from the command line

- recall from lecture 11 ⟨`http://floppsie.comp.glam.ac.uk/Southwales/gaius/games/11.html`⟩ that we can `import pexpect` and interact with a command line program in a similar way to that of keyboard interaction
- however we must program all activity
  - we must make our python program match output from the command line tool and provide sensible input for this tool
- so in the case above we need to give the appropriate initialisation parameters to the program as it starts up
  - and respond to `Please enter move:` prompts
  - and retrieve output from `My move is:` statements

Running m2chess from the command line

- finally any outputs need to be fed to the ChessBoard GUI and a new move need

Running m2chess from the command line

```
import pexpect, sys, string, os
from pexpect import TIMEOUT, EOF

class m2chess:
    def __init__ (self, debugging = False, level = 1,
        filename = "./chess", directory = "."):
        if os.path.isdir(directory):
            os.chdir(directory)
            print "cd ", directory, " and running ", file
        else:
            print "error as, directory: ", \
                  directory, " does not exist"
            sys.exit(0)

        self.child = pexpect.spawn (filename)
        self.child.delaybeforesend = 0
        self.level = level
        self.finished = False
        self.debugging = debugging
```


Running m2chess from the command line

```
        self.child.expect('Enter Stage Of Game')
        self.child.sendline('opening0)

        if self.debugging:
            print self.child.before
        self.child.expect('Is the present Board in initial
        self.child.sendline('yes')
        if self.debugging:
            print self.child.before
```

Running m2chess from the command line

```
        self.child.expect('Human')
        self.child.sendline('h')
        if self.debugging:
            print self.child.before
        self.child.expect('Human')
        self.child.sendline('c')
        if self.debugging:
            print self.child.before
```


Running m2chess from the command line

```
    def makeMove(self, move):
        if self.debugging:
            print "making move"
            print self.child.before
        self.child.expect('Please enter move')
        self.child.sendline(move)

    def getMove(self):
        if self.debugging:
            print "getting move"
            print self.child.before
        i = self.child.expect([pexpect.TIMEOUT, '(gdb)',
            'My move is:\s+(.*[A-H][1-8].*[A-H][1-8])'],
            timeout=1000)
        if i==0 or i==1:
            print "something has gone wrong..."
            self.child.interact()
            sys.exit(0)
        return self.child.match.groups()[0]
```


Running m2chess from the command line

```
    def doInteract(self):
        if self.finished:
            print "no m2chess interactive session availab
        else:
            try:
                self.child.interact()
            except os.error:
                self.finished = True
```


Running m2chess from the command line

```
def main():
    foo = m2chess(False)
    foo.makeMove('e2e4')
    print foo.getMove()
    foo.makeMove('d2d4')
    print foo.getMove()

if __name__ == '__main__': main()
```

Tutorial

- using wikipedia search for Chess openings and in particular the openings starting D2-D4 (white plays Queens pawn to row 4)
  - find the book moves which classically are used to combat this move
  - now download m2chess-0.3.tar.gz ⟨`http://floppsie.comp.glam.ac.uk/download/m2/m2chess-0.3.tar.gz`⟩
  - and extract and build the file contents by typing:

```
$ tar zxf m2chess-0.3.tar.gz
$ cd m2chess
$ make
```

- from a command line terminal

Tutorial

- now download a modified ModifiedChessBoard.tar.gz ⟨`http://floppsie.comp.glam.ac.uk/download/m2/ModifiedChessBoard.tar.gz`⟩
  - and extract and run it by:

```
$ tar zxf ModifiedChessBoard.tar.gz
$ cd ModifiedChessBoard
$ python PlayGame.py
```

- firstly see whether the chess program can defend against fools mate:
  - white plays: e2-e4, f1-c4, d1-h5 and possibly h5-f7 checkmate
  - assuming black does not defend correctly

Tutorial

- and extend the file `Book` with these recognised replies
- modify the weightings (held in file `in`) to make `m2chess` capture the center ground and also encourage the computer to castle

Description of the evaluation function weightings

- there are three stages of the game of chess
  - opening, middle game and end game
- each of which has the following weightings for the evaluation function:
- `Material Balance` which scores points for pawns, knights, bishops, rooks and queen
  - ratio of 1, 3, 3, 6 and 9
  - the value given as the `Material Balance` determines the value of a pawn

Description of the evaluation function weightings

- `Mobility Wgt` score points for number of moves the pieces are able to make
- `Pawn Doubled` counts the number of pawns on the same column (subtracts by one) and multiplies by this value
  - normally a negative value to encourage good pawn structure
- `Bishop Doubled` is added if a bishop is on the same diagonal as its queen
  - encourages good piece structure, both defensive and attacking

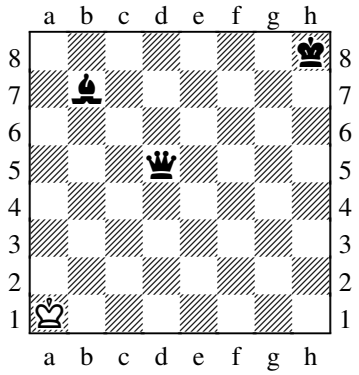
Description of the evaluation function weightings

- `Rook Doubled` are the two rooks on the same row or column?
  - same reason as bishop doubling
- `Fork Pts` value of a fork
- `Can Castle` can player castle
- `Has Castled` has player castled
- `Center Control` how near the center is the piece

Description of the evaluation function weightings

- `Near King` this weighting is multiplied by the number of squares away from the king
- `King Safety` how many squares away are the enemy pieces to our king
  - the total of this value (for each piece) is multiplied by this weighting
- `King Center`
  - how close is the king to the center?
  - if the king is in the center 16 squares this value is added to the evaluation function
- `Advance Pawn` a value of 1..8 is multiplied by this weighting depending upon how close a pawn is from the end of the board

Example of Bishop Doubled



Example of Bishop Doubled

- here the evaluation function adds the Bishop Doubled value to the score for black as the bishop and queen are on the same diagonal