Я изучил двумерные массивы. Они никак не могут мне помочь в корреляции с функциями.

Возможно, можно было перестроить доску таким образом, чтобы работать по клеткам двумерного массива. Но так клетки будут лишены жизни. А мне удобнее работать с информацией в клетках. Да, это, скорее всего, замедлит работу, но как никак сделает код более простым.

Моя текущая проблема (test 10) заключается в том, что я понятия не имею, как реализовывать наследование классов. То есть, да, я могу заняться тупой «копипастой» и переписать так для любой клетки, но хорошо бы заняться наследованием. С другой стороны, мы не ещё не закончили работу над функцией клетки. То есть в клетку мы ещё будем отправлять запросы по координатам и фигурам, так что пока что, возможно, имеет смысл не торопиться.

Итак, первая цель – сделать 64 функции для каждой клетки. Занятие тупой «копипастой», но придётся.

В переменную changes можно добавить координаты, чтобы таким образом проверять та ли клетка, или нет.

Итак, приступим. 13:57

Пока что я не буду добавлять в массив картинки фигур, потому что это можно сделать потом.

Потом все клетки из функций можно будет переделать в классы, чтобы образовывать наследственные связи.

**ИДЕИ**

1. После успешного написания кода цвет и фигуры можно обозначать цифрами. Стоит над этим подумать.
2. Не обязательно указывать координаты клетки в её массиве. Достаточно имени.
3. Не нужно координату обозначать за ‘1’, можно просто 1.
4. Не нужны вообще координаты в массиве, поскольку в функции указано её имя, а значит и координаты. Но их лень пока что считывать, так что пусть будут.

chr(1+ord(‘A’))

‘B’

Таким образом мы увеличиваем координату по буквенному значению.

Но теперь есть другая проблема. Как извлекать информацию из нужного нам массива.

Как вызвать нужную нам функцию? Получается очень большая проблема, из-за которой наша программа может оказаться бесполезной. Ведь мы не знаем, как вызвать нужную нам функцию.

Ух-ты! В test 15 получилось реализовать концепцию связанных друг с другом клеток.

class Cell(object):

def \_\_init\_\_(self):

self

def A1(coord, changes):

CL = ['A', 1, 'WHITE', 'ROOK']

if (coord == ['A', 1]):

if (changes != []):

del CL[2:]

CL.append(changes[0])

CL.append(changes[1])

Cell\_info = [CL[2], CL[3]]

return CL

else:

return Cell.A2(coord, changes)

def A2(coord, changes):

CL = ['A', 2, 'WHITE', 'ROOK']

if (coord == ['A', 2]):

if (changes != []):

del CL[2:]

CL.append(changes[0])

CL.append(changes[1])

Cell\_info = [CL[2], CL[3]]

return CL

else:

return Cell.A3(coord, changes)

def A3(coord, changes):

CL = ['A', 3, '', '']

if (coord == ['A', 3]):

if (changes != []):

del CL[2:]

CL.append(changes[0])

CL.append(changes[1])

Cell\_info = [CL[2], CL[3]]

return CL

else:

return Cell.A4(coord, changes)

changes = ['BLACK', 'BISHOP']

coord = ['A', 3]

print (Cell.A1(coord, changes))

Итак, вся эта программа заняла 718 строк. Это не очень хорошо. Хорошо бы, конечно, сделать клетки отдельными классами и через общий класс наследовать клетку. Но это будет долго. В перспективе, возможно, я это реализую. Зависит от того, насколько хорошей получиться программа. Потому что если она получится очень хорошей, я буду переписывать её на С++ при поддержке Андрея. Если нет, то будет обидно и я буду дорабатывать её.

В любом случае, главный параметр для меня пока что – не время, а точность. То есть даже если программа будет работать со скоростью 18000 позиций в секунду, мне главное, чтобы при достижении нужной глубины, она выдавала схожий с оценкой Стокфиша результат.

Кстати, в PyCharm отлаженный алгоритм работает быстрее.

Что интересно, если сразу внести второй виток изменений после Н8, программа скажет, что что-то не так. Значит надо использовать какую-то функцию stop, которая будет возвращать перебор к А1.

Сейчас я решил немного отступить от шахматной программы и на базе Коллатца поизучать потоки и multiprocessing.

Но это как-то сложно, так что пока что я пойду спать. Завтра продолжу работу над движком.