README FILE

To run the program :

run the : *main.py* file with the path to file you want to try.
full command (it works this way on my machine) :

```
python main.py PATH_TO_FILE
```

Elaboration about the algorithm which I used to solve the problem:

I started with the case of even-edge square board.
I divided the game board to four parts :

A	B
D	C

When I read the boxes and hunters coordinates I counted the number of boxes and hunters in each part, then I concluded the number of free seats in each part.
Then, I use the follow equations :
Fix for each part A,B,C,D

$H_A - number\ of\ hunters\ in\ A$

$B_A - number\ of\ boxes\ in\ A$

$F_A - number\ of\ free\ seats\ in\ A\ \left(\left(\frac{N}{2}\right)^2 - H_A - B_A\right)$

$X_A - the\ number\ of\ hunters\ can\ be\ added\ to\ part\ A\ (limited\ by\ F_A)$

Now, according to the rules I get :

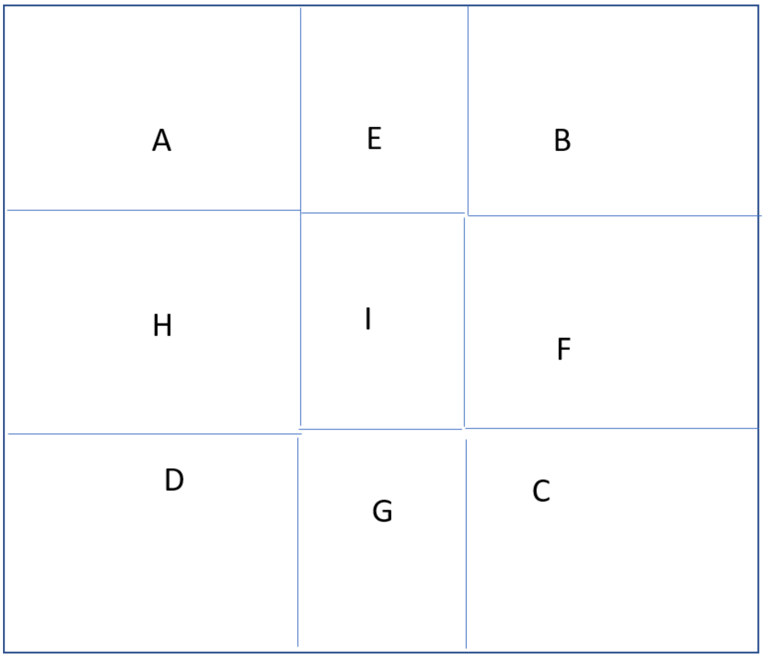
$X_A + H_A + X_B + H_B = X_C + H_C + X_D + H_D$

$X_A + H_A + X_D + H_D = X_B + X_C + H_B + H_D$

$0 < X_{part} < F_{part}\ (part \in \{A, B, C, D\})$

Then, I tried to find maximal X values such that : $X_A + X_B + X_C + X_D \to max$

For the case of odd-edge game boards :
I created different partition to the game board -



The idea is that the only difference between the even and odd game board is the row or column in the middle.

So I set the row/column as a different part of the board ($size\ of\ \frac{N-1}{2} * 1$)

And then I have to balance the parts A,B,C,D like before and I have to balance E against G and H against F.

The part I In the center is just the center point of the board (size 1*1).