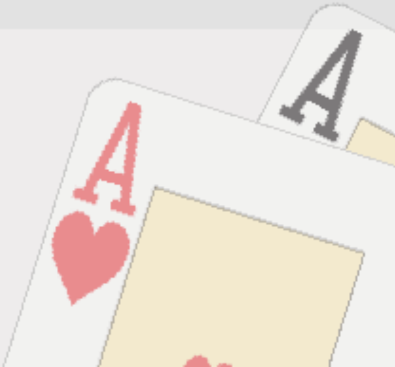
AI Games course

Certificate 1, session 3

“What does the data say?”

AIgaming.com



Strategy so far

- In the *target mode*, search through the space around the wounded piece of ship.
- In the *hunt mode*, aim randomly.

Strategy so far

- In the target *mode*, search through the space around the wounded piece of ship.
- In the *hunt mode*, aim randomly.

Let's improve on that!

© Sarah Andersen

Learning from data

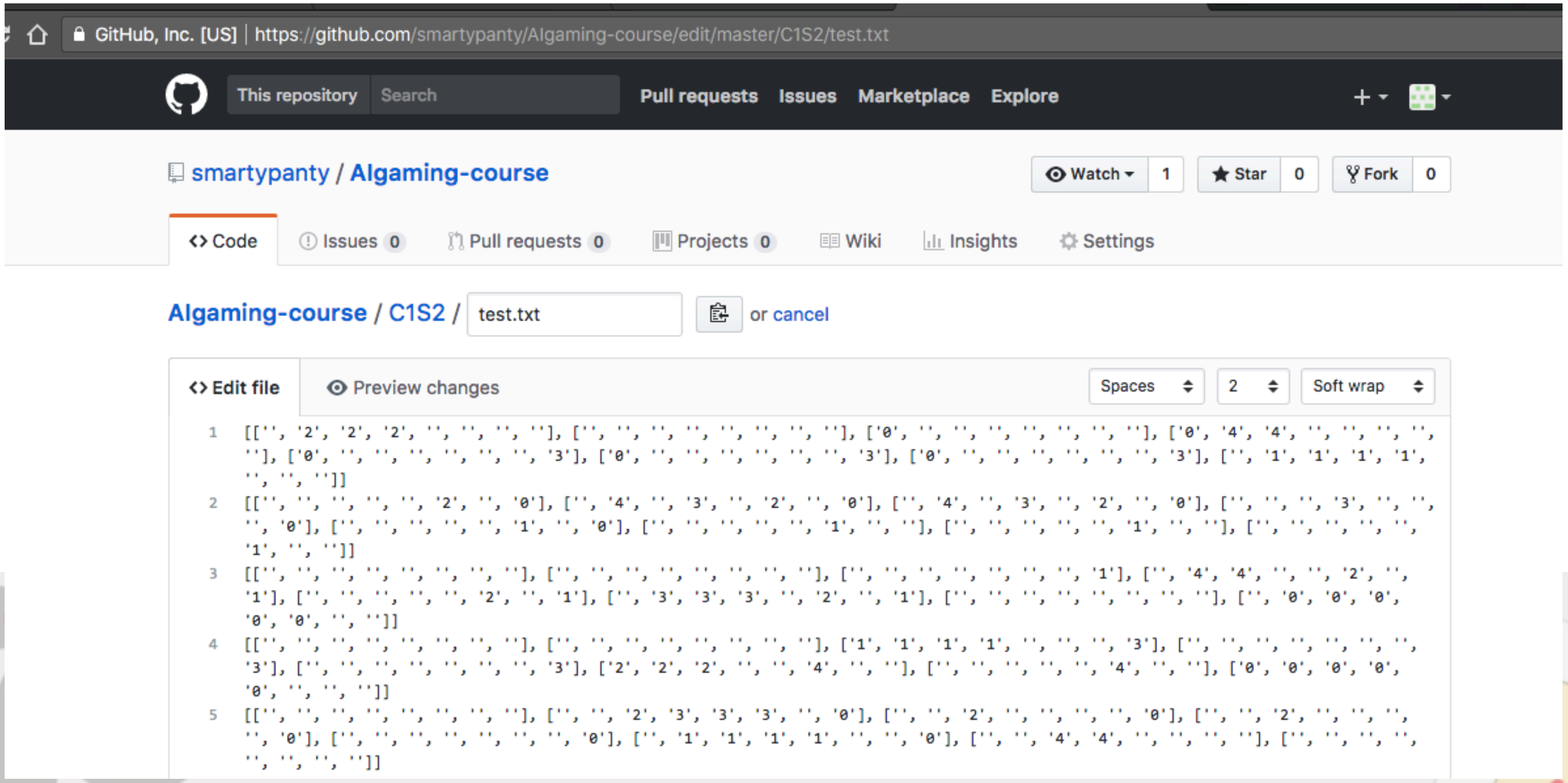
- **Idea**: instead of randomly searching for a ship, let’s *predict* where it *probably* is.

- there may be a pattern in how a user arranges ships, e.g.:
  - all ships stacked in one place, or
  - all ships in corners, etc.

- given what you already know about the opponent’s board, hit the cell that is *most likely* to contain a ship, *based on the data.*

User logs

- download *boards.txt*
- https://github.com/smartypanty/AIgaming-course/blob/master/C1S2/boards.txt
- 60000+ real board arrangements generated by the users (only 179 default arrangements)

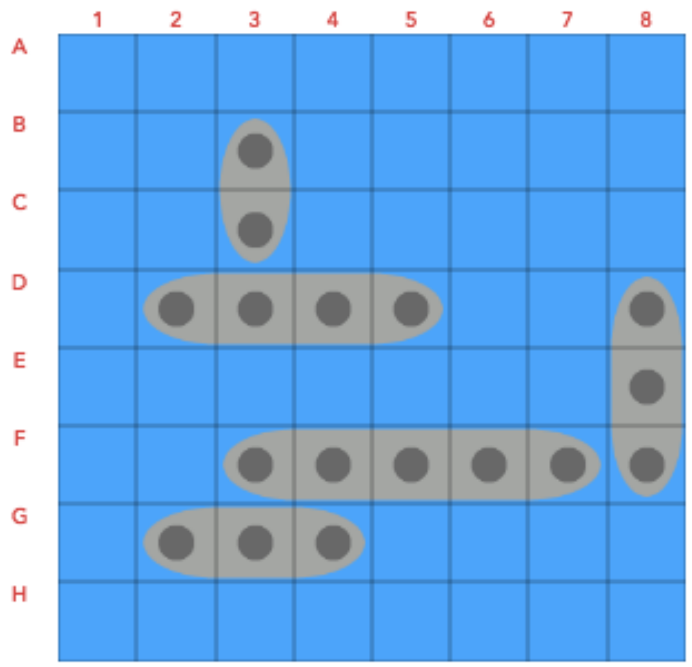


AIgaming-course / C1S2 / test.txt

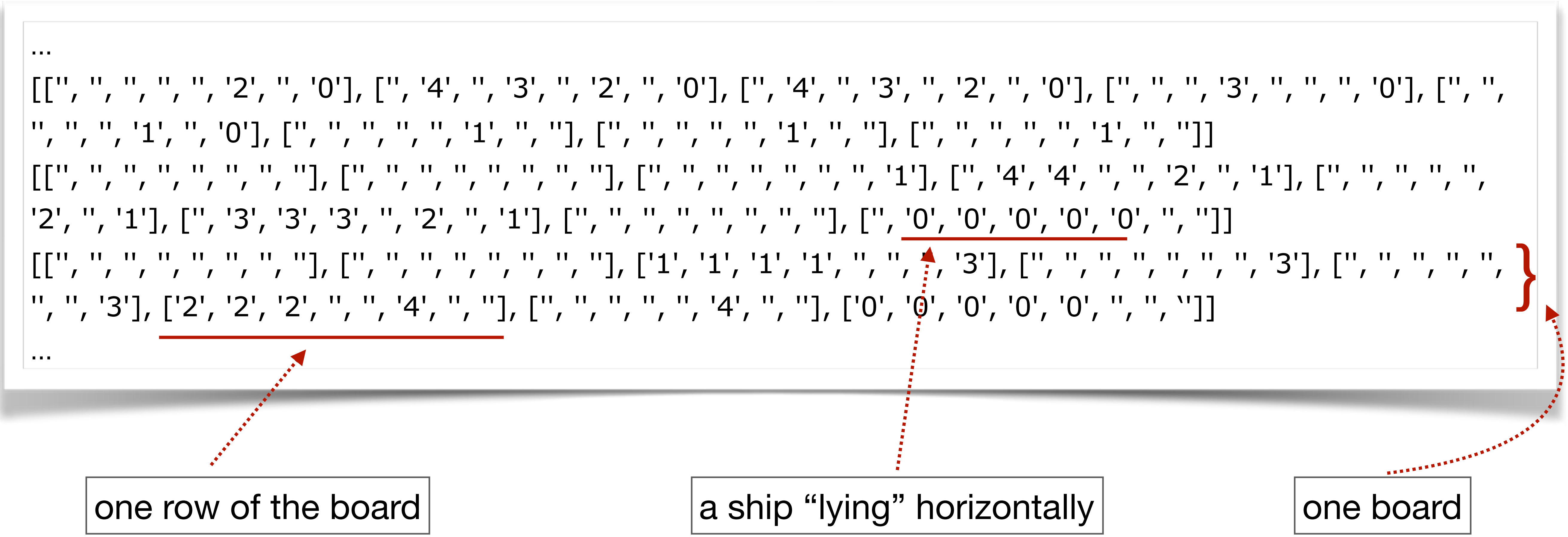
Spaces: 2, Soft wrap

```
[['', '2', '2', '2', '', '', '', ''], ['', '', '', '', '', '', '', ''], ['0', '', '', '', '', '', '', ''], ['0', '4', '4', '', '', '', '', ''], ['0', '', '', '', '', '', '', '3'], ['0', '', '', '', '', '', '', '3'], ['0', '', '', '', '', '', '', '3'], ['', '1', '1', '1', '1', '', '', '']]
[['', '', '', '', '', '2', '', '0'], ['', '4', '', '3', '', '2', '', '0'], ['', '4', '', '3', '', '2', '', '0'], ['', '', '', '3', '', '', '', '0'], ['', '', '', '', '', '1', '', '0'], ['', '', '', '', '', '1', '', ''], ['', '', '', '', '', '1', '', ''], ['', '', '', '', '', '1', '', '']]
[['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', '1'], ['', '4', '4', '', '', '2', '', '1'], ['', '', '', '', '', '2', '', '1'], ['', '3', '3', '3', '', '2', '', '1'], ['', '', '', '', '', '', '', ''], ['', '0', '0', '0', '0', '0', '', '']]
[['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', ''], ['1', '1', '1', '1', '', '', '', '3'], ['', '', '', '', '', '', '', '3'], ['', '', '', '', '', '', '', '3'], ['2', '2', '2', '', '', '4', '', ''], ['', '', '', '', '', '4', '', ''], ['0', '0', '0', '0', '0', '', '', '']]
[['', '', '', '', '', '', '', ''], ['', '', '2', '3', '3', '3', '', '0'], ['', '', '2', '', '', '', '', '0'], ['', '', '2', '', '', '', '', '0'], ['', '', '', '', '', '', '', '0'], ['', '1', '1', '1', '1', '', '', '0'], ['', '', '4', '4', '', '', '', ''], ['', '', '', '', '', '', '', '']]
```

User logs



- download *boards.txt*



```
...
[['', '', '', '', '', '2', '', '0'], ['', '4', '', '3', '', '2', '', '0'], ['', '4', '', '3', '', '2', '', '0'], ['', '', '', '3', '', '', '', '0'], ['', '',
'', '', '', '1', '', '0'], ['', '', '', '', '', '1', '', ''], ['', '', '', '', '', '1', '', ''], ['', '', '', '', '', '1', '', '']]
[['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', '1'], ['', '4', '4', '', '', '2', '', '1'], ['', '', '', '', '',
'2', '', '1'], ['', '3', '3', '3', '', '2', '', '1'], ['', '', '', '', '', '', '', ''], ['', '0', '0', '0', '0', '0', '', '']]
[['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', ''], ['1', '1', '1', '1', '', '', '', '3'], ['', '', '', '', '', '', '', '3'], ['', '', '', '', '',
'', '', '3'], ['2', '2', '2', '', '', '4', '', ''], ['', '', '', '', '', '4', '', ''], ['0', '0', '0', '0', '0', '', '', `']]
...
```

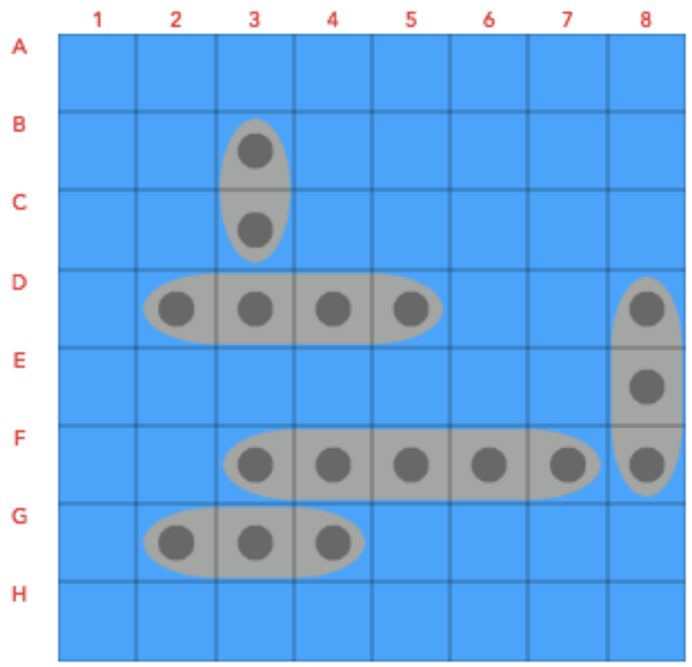
one row of the board

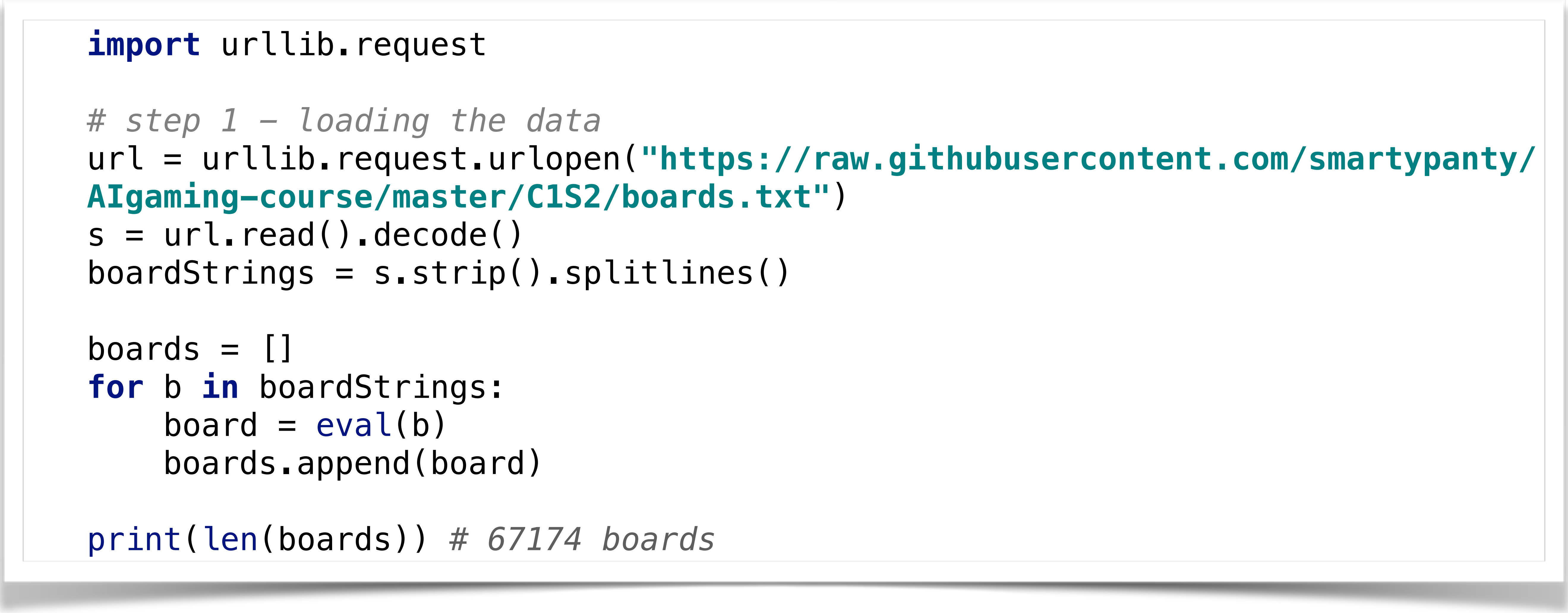
a ship “lying” horizontally

one board

User logs

- access *boards.txt*

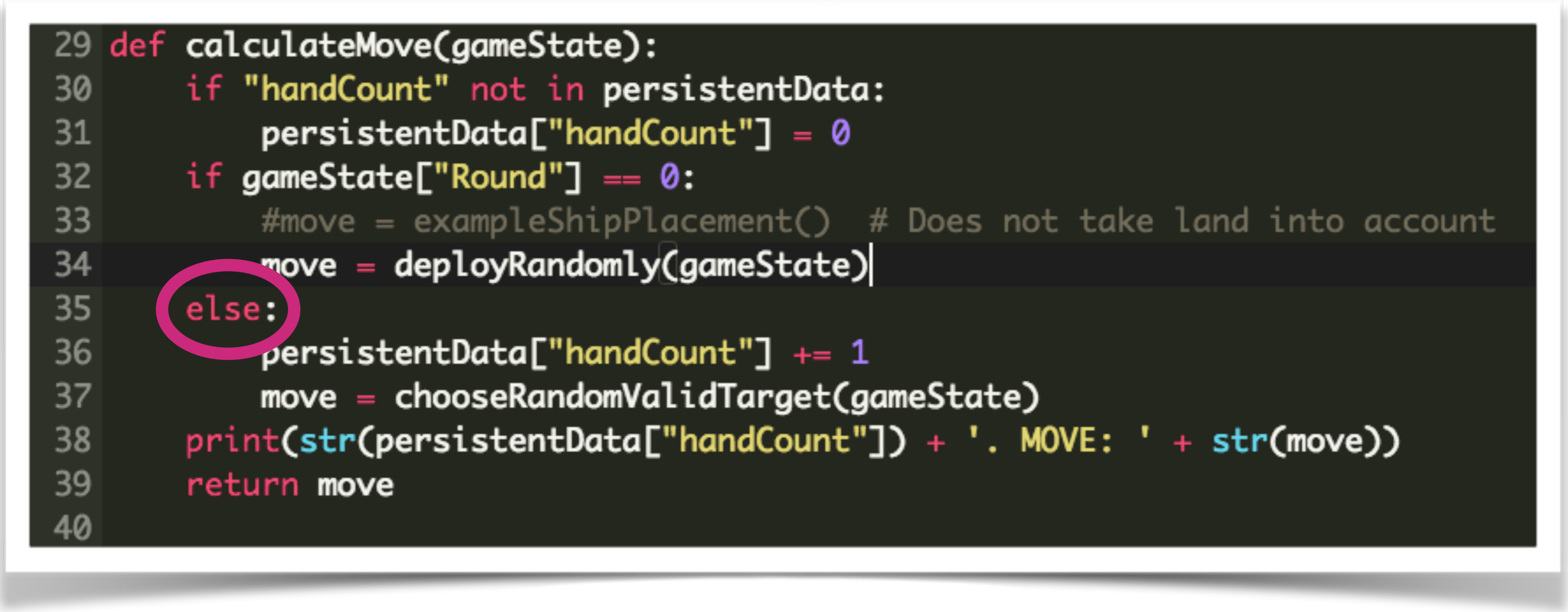




```
import urllib.request

# step 1 - loading the data
url = urllib.request.urlopen("https://raw.githubusercontent.com/smartypanty/AIgaming-course/master/C1S2/boards.txt")
s = url.read().decode()
boardStrings = s.strip().splitlines()

boards = []
for b in boardStrings:
    board = eval(b)
    boards.append(board)

print(len(boards)) # 67174 boards
```

Predicting ship placement

- **def calculateMove:**



```python
def calculateMove(gameState):
    if "handCount" not in persistentData:
        persistentData["handCount"] = 0
    if gameState["Round"] == 0:
        #move = exampleShipPlacement()  # Does not take land into account
        move = deployRandomly(gameState)
    else:
        persistentData["handCount"] += 1
        move = chooseRandomValidTarget(gameState)
    print(str(persistentData["handCount"]) + '. MOVE: ' + str(move))
    return move

```

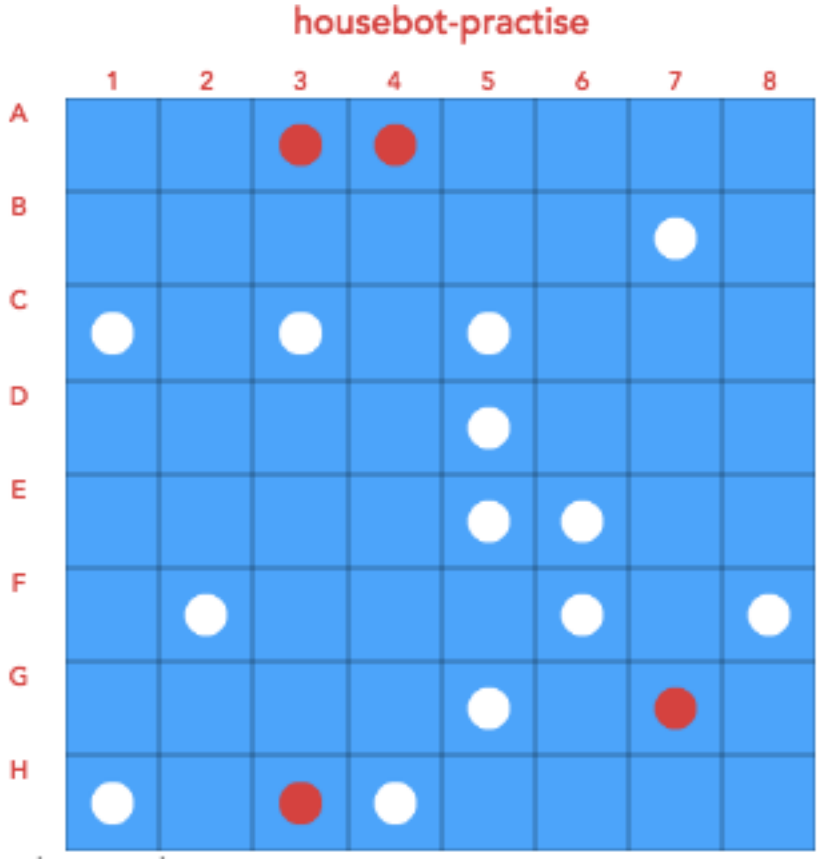
Let’s modify the *else*-statement to accommodate probabilities.

Predicting ship placement

1) From *gameState*, parse the current state of the opponent's board:

‘OppBoard': [['', '', 'H', 'H', '', '', '', ''], ['', '', '', '', '', '', 'M', ''], ['M', '', 'M', '', 'M', '', '', ''], ['', '', '', '', 'M', '', '', ''], ['', '', '', '', 'M', 'M', '', ''], ['', 'M', '', '', '', 'M', '', 'M'], ['', '', '', '', 'M', '', 'H', ''], ['M', '', 'H', 'M', '', '', '', '']]

```
oppBoard = gameState['OppBoard']
```



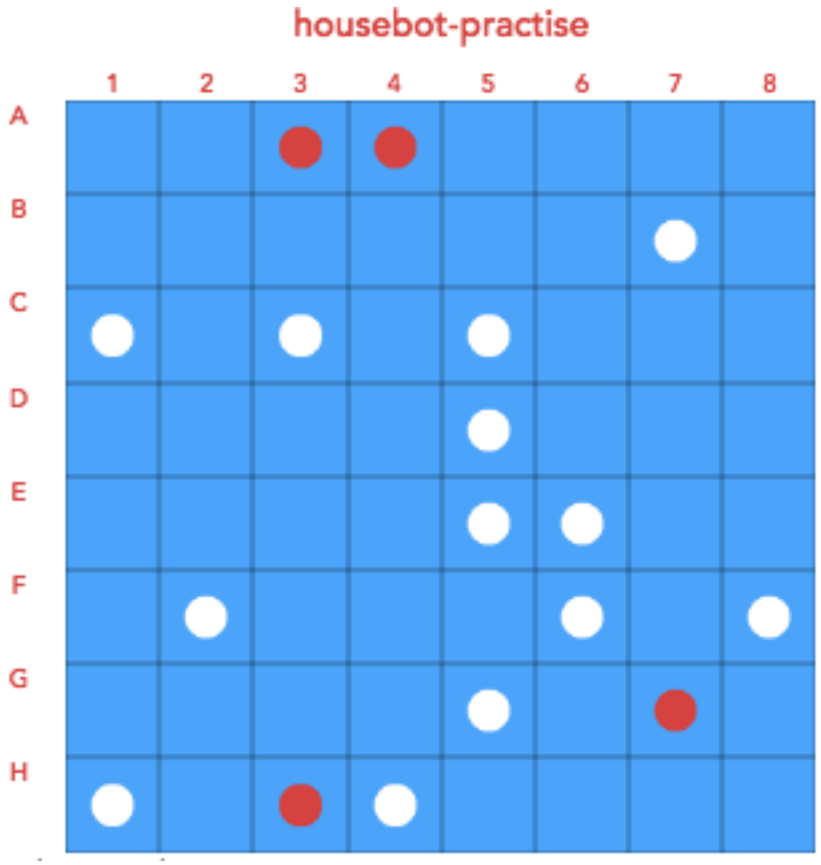
	1	2	3	4	5	6	7	8
A			●	●				
B							○	
C	○		○		○			
D					○			
E					○	○		
F		○				○		○
G					○		●	
H	○		●	○				

Predicting ship placement

2) in *boards.txt*, find those boards that coincide with *OppBoard* on the known cells:

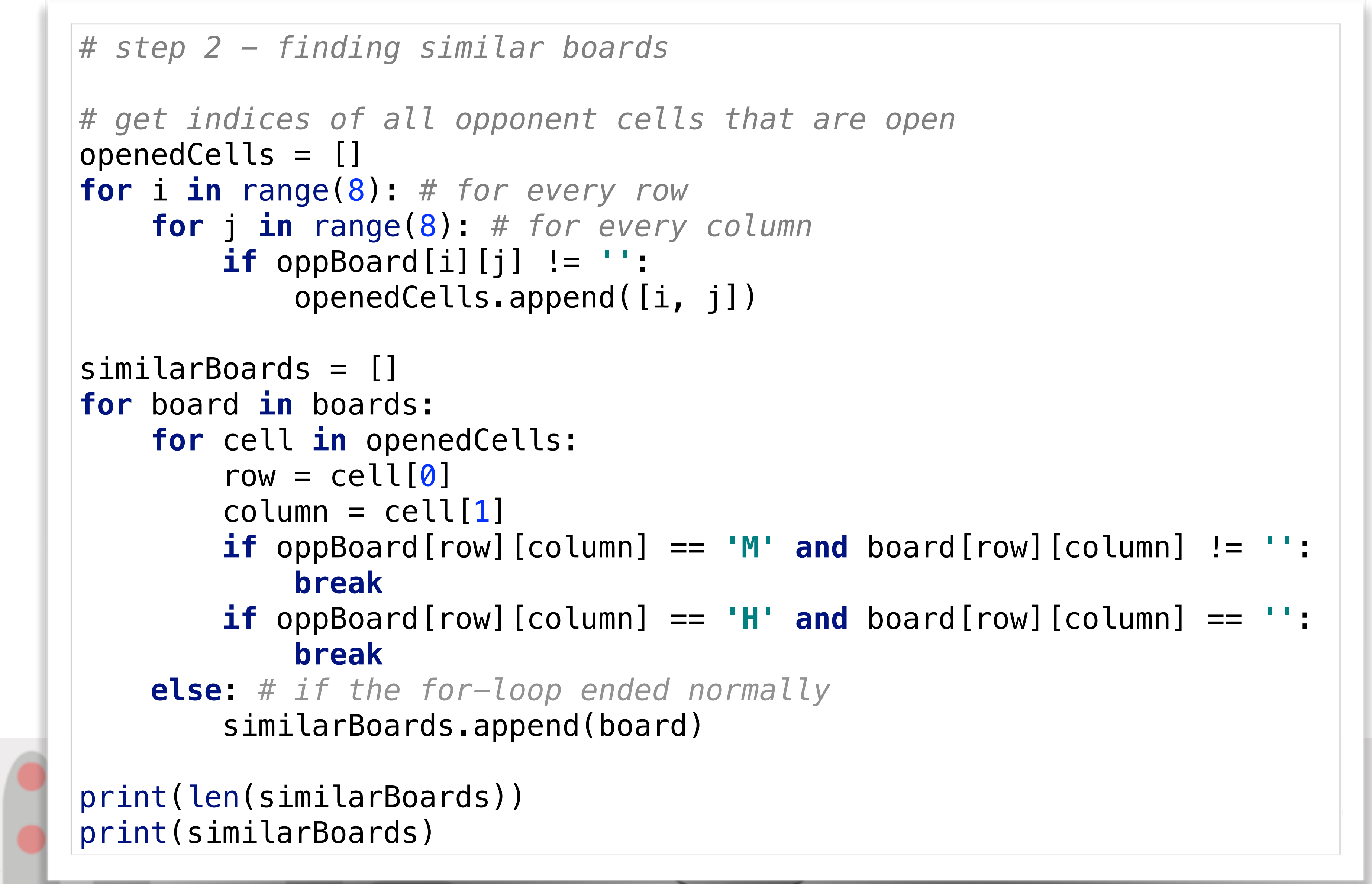
E.g., [['0', '0', '0', '0', '0', '', '', '3'], ['', '', '', '', '', '', '', '3'], ['', '', '', '', '', '', '', '3'], ['2', '2',   '2', '2', '', '', '', ''], ['', '', '', '', '', '', '1', ''], ['', '', '', '', '', '', '1', ''], ['', '', '', '', '', '', '1', ''], [‘', '4', '4', '', '', '', '1', '']]

We will call them *similar boards.*

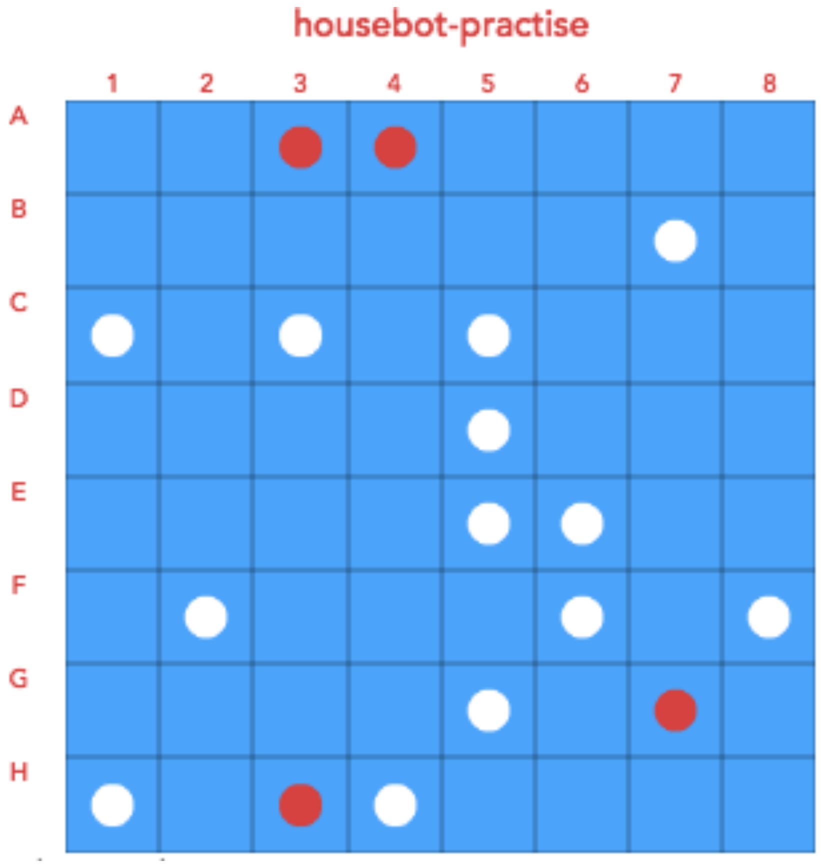


Predicting ship placement

2) in *boards.txt*, find those boards that coincide with *OppBoard* on the known cells:



```
# step 2 - finding similar boards

# get indices of all opponent cells that are open
openedCells = []
for i in range(8): # for every row
    for j in range(8): # for every column
        if oppBoard[i][j] != '':
            openedCells.append([i, j])

similarBoards = []
for board in boards:
    for cell in openedCells:
        row = cell[0]
        column = cell[1]
        if oppBoard[row][column] == 'M' and board[row][column] != '':
            break
        if oppBoard[row][column] == 'H' and board[row][column] == '':
            break
    else: # if the for-loop ended normally
        similarBoards.append(board)

print(len(similarBoards))
print(similarBoards)
```



housebot-practise

	1	2	3	4	5	6	7	8
A			●	●				
B							○	
C	○		○		○			
D					○			
E					○	○		
F		○				○		○
G					○		●	
H	○		●	○				

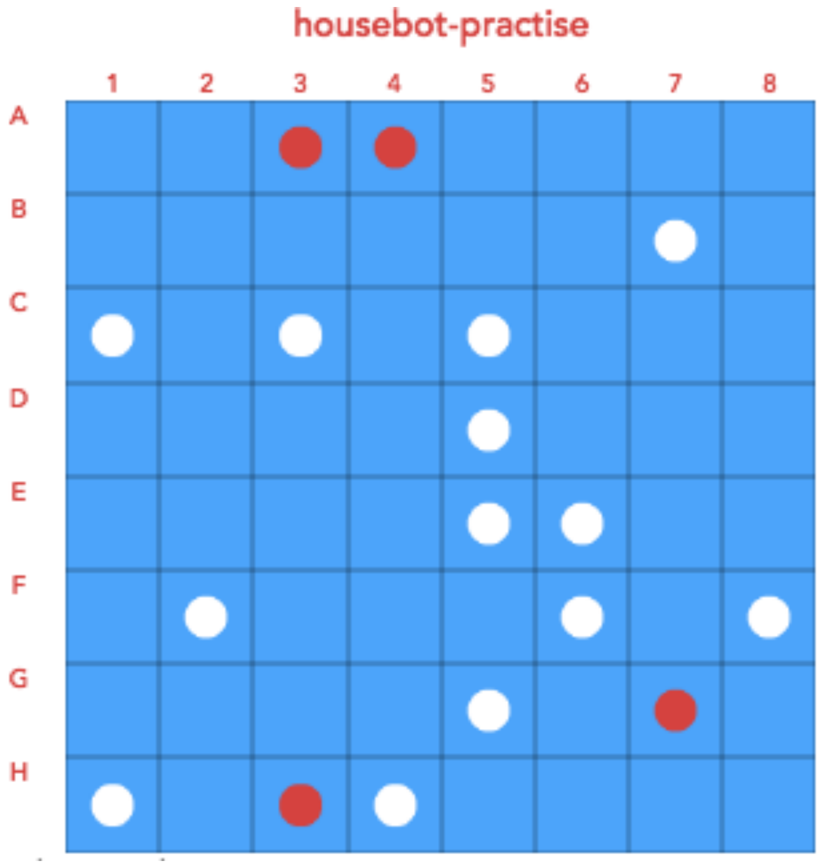
Predicting ship placement

3) find cells on similar boards that:

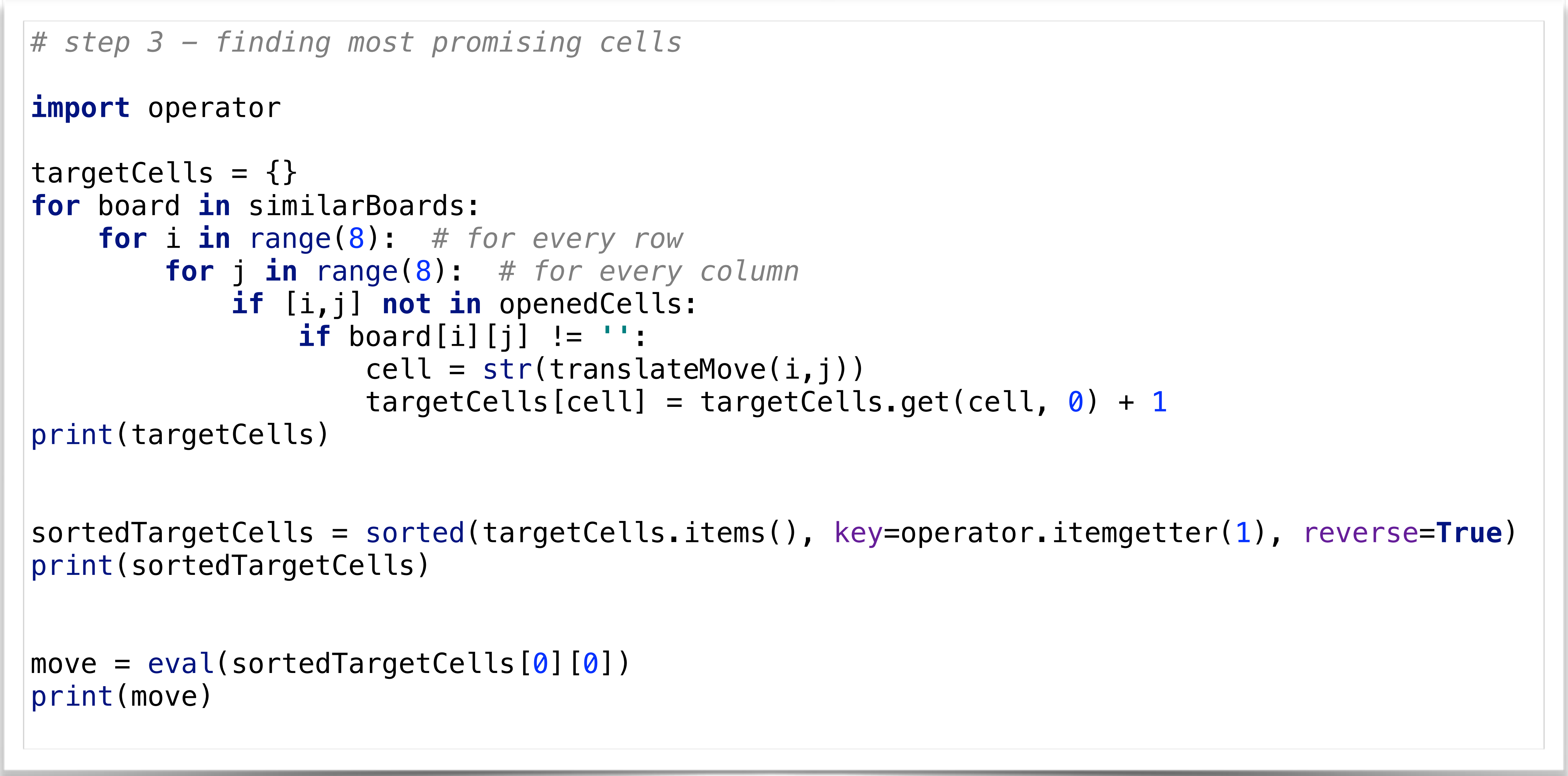
- are not yet opened on *OppBoard*, and

- contain ships in the maximum number of similar boards.

```
for every unopened cell on OppBoard:

     cell_score := # of similar boards that contain
     ship in that cell;

pick the cell with the highest score.
```



Predicting ship placement



```python
# step 3 - finding most promising cells

import operator

targetCells = {}
for board in similarBoards:
    for i in range(8):  # for every row
        for j in range(8):  # for every column
            if [i,j] not in openedCells:
                if board[i][j] != '':
                    cell = str(translateMove(i,j))
                    targetCells[cell] = targetCells.get(cell, 0) + 1
print(targetCells)


sortedTargetCells = sorted(targetCells.items(), key=operator.itemgetter(1), reverse=True)
print(sortedTargetCells)


move = eval(sortedTargetCells[0][0])
print(move)
```

Handling data sparsity

- 60,000 boards is not that much data to learn from!
- what if you cannot find similar boards?

Ex 1: round 3, thousands of similar boards that do not contain a ship in (row B, column 7)

‘OppBoard': [['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', 'M', ''], ['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', ''],
['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', ''],['', '', '', '', '', '', '', ''], ['', '', '', '', '', '', '', '']]

Ex 2: round 30+, possibly no similar boards with the given ship arrangement

E.g., [['H', 'H', 'H', 'H', 'M', '', '', 'H'], ['M', 'M', '', 'M', '', '', '', 'H'], ['M', 'M', '', '', 'M', '', '', ''],
['H', 'M', 'M', 'H', 'M', '', 'M', ''], ['M', 'M', 'M', '', 'M', 'M', 'H', ''], ['M', '', 'M', 'M', '', 'M', 'H', 'M'],
['M', 'M', 'M', 'M', '', '', 'H', ''], ['', 'H', 'M', '', '', 'M', 'H', '']]

Handling data sparsity

- in case there are no similar boards for a given `OppBoard`:
    - either revert to `deployRandomly(gamestate)`, or
    - make a *soft comparison* of boards, i.e., find boards that are most similar to `OppBoard`.

Has the performance improved?

- In the hunt mode, we moved from a random guess to an *educated guess* which is based on data.
- In the target mode, we moved from a random guess to search.

Have we improved?

Has the performance improved?

- In the hunt mode, we moved from a random guess to an *educated guess* which is based on data.
- In the target mode, we moved from a random guess to search.

Have we improved?

Try running your bot *n* times (10, 50, 100…), and then run the default random bot against the same opponent (e.g., housebot-practice) *n* times.

Has the win/lose ration changed? Tell us!