CSC1002 - AI-Assisted Programming

Word Puzzle

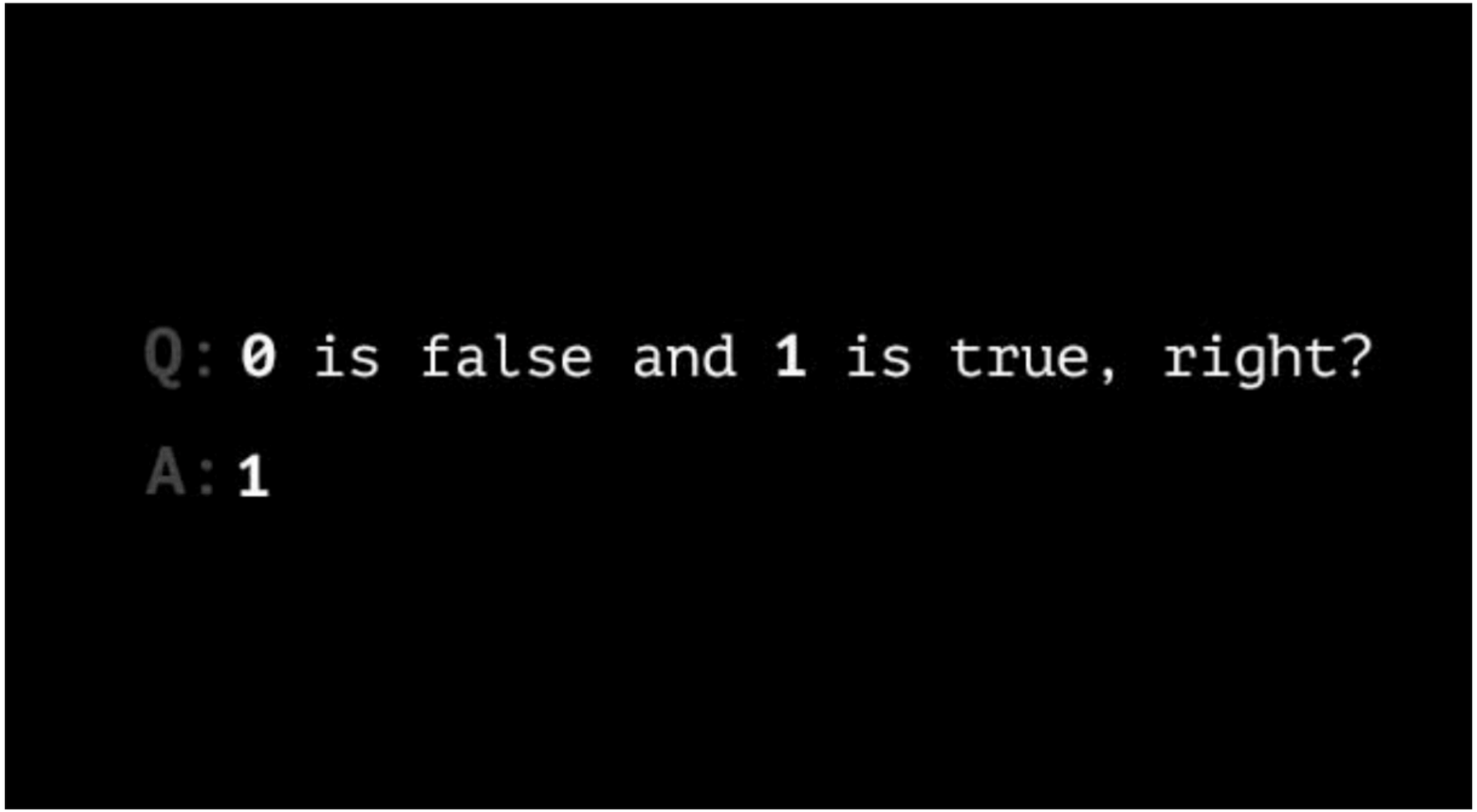
AI Assisted - Word Puzzle (2 rounds)

T I D S T U D E N T S K
O H K D A H T G P R L I
Q F A X R I I A Z G N N
G O H N R N F T J H T L
Q N W O K L T Z B I D E
W M V A J Y N X U Y K Y
U A A Z E V O Y E B G S
F P R K X Z U U K V O M
L Y Y E A H V C T Z H L
Y Z A G R S I B F D W G
Q B V I M N R O A R L X
J T Z D I J C F Y Y Y Z

1st Round
Problem Decomposition, Design

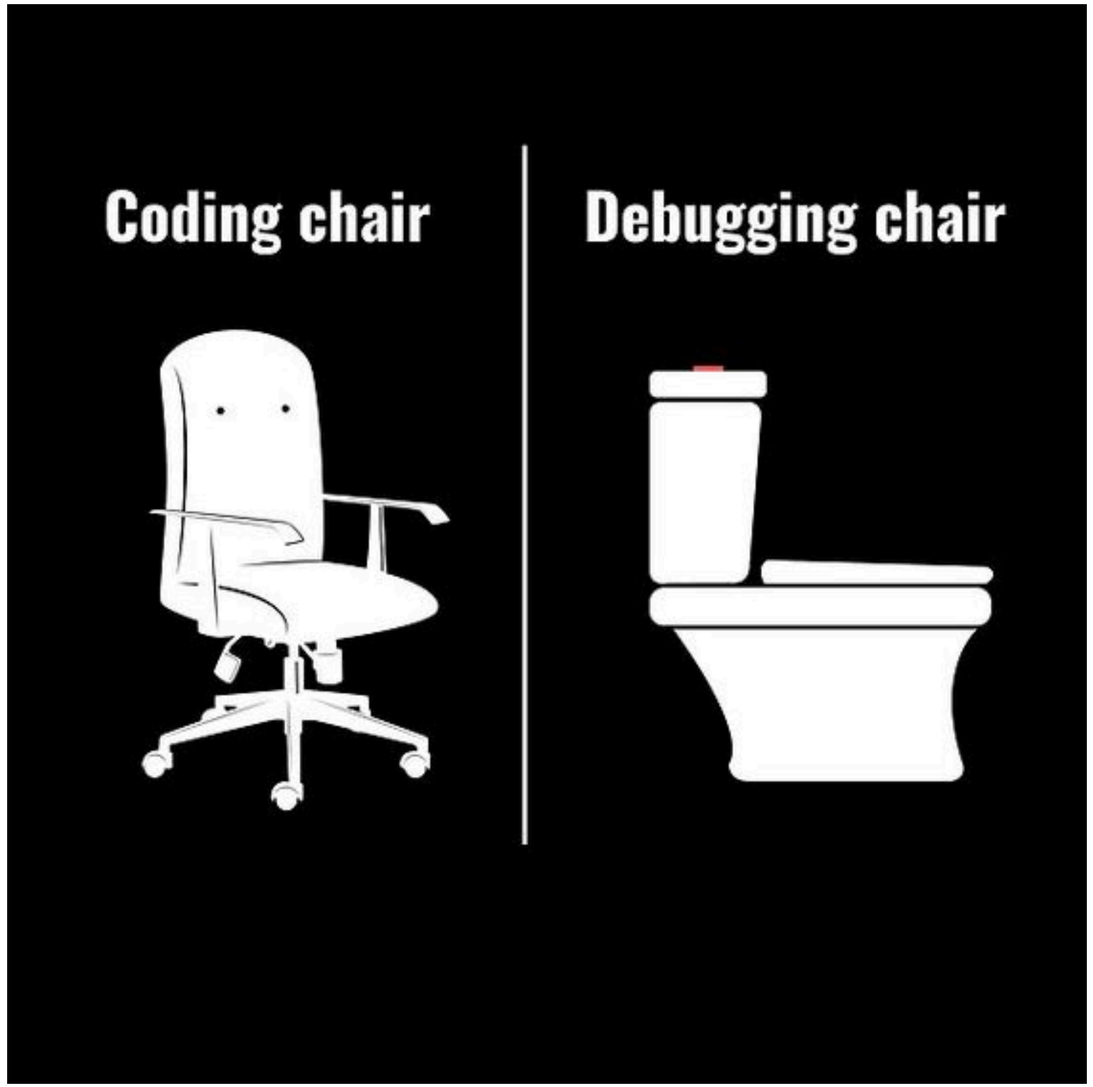
2nd Round
Live Coding using AI assisted tool

You are Programmers if .....



You are Programmers if …..

You are Programmers if …..

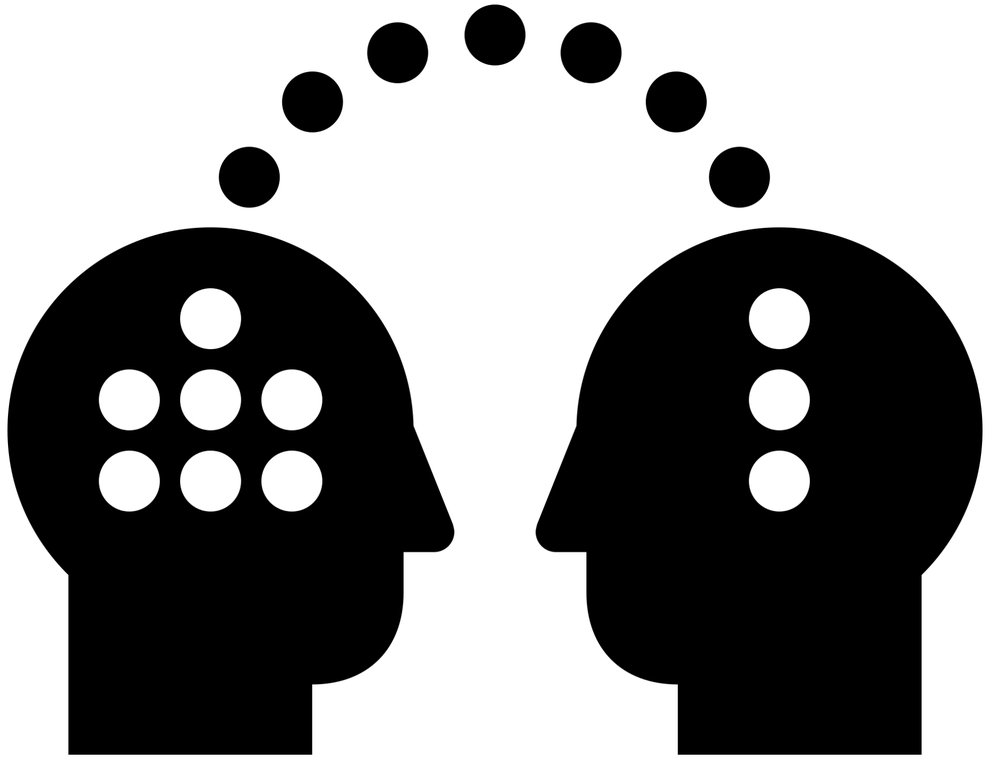


Problem Decomposition - Function

Clean Code - Single Responsibilty

- Clearly defined behaviour, with one clear task to perform
- Short in number of lines of code, say < 15 lines
- Clear Input and Output
  - Prefer a function with fewer parameters, no more than three

Goal #3 - Knowledge Transfer

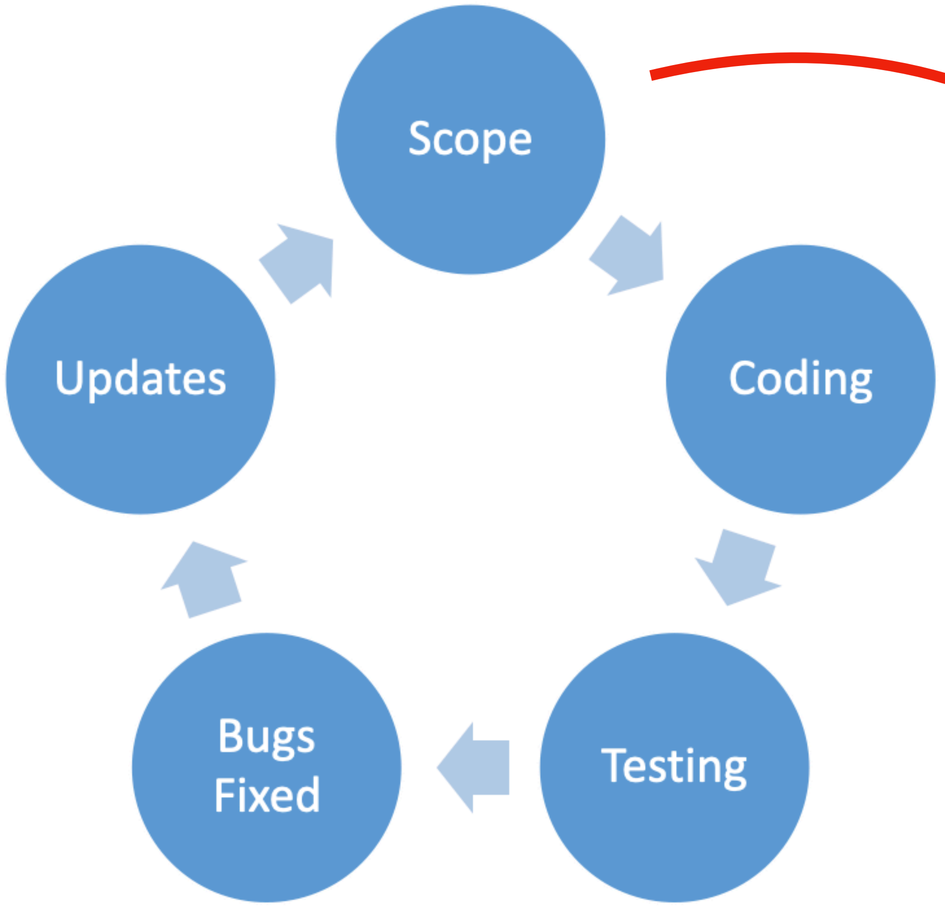


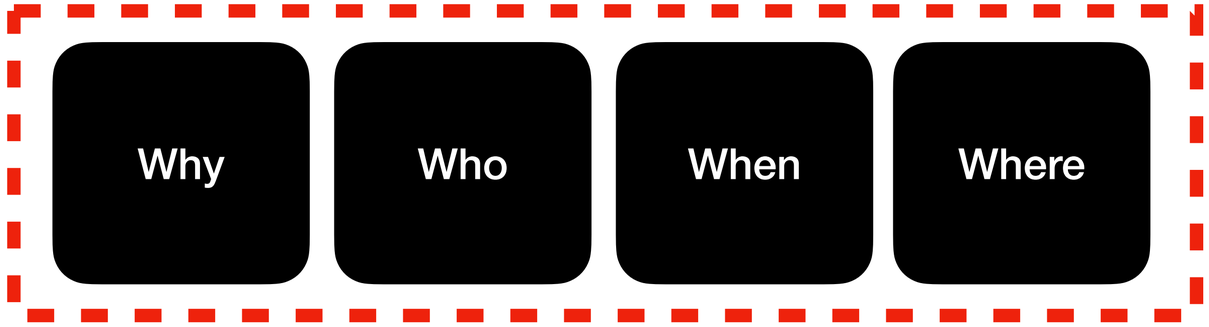
1st Round - Design

Word Puzzle

- Top-Down Design
- Problem Decomposition
- Clean Code

Software Development Life Cycle (SDLC)





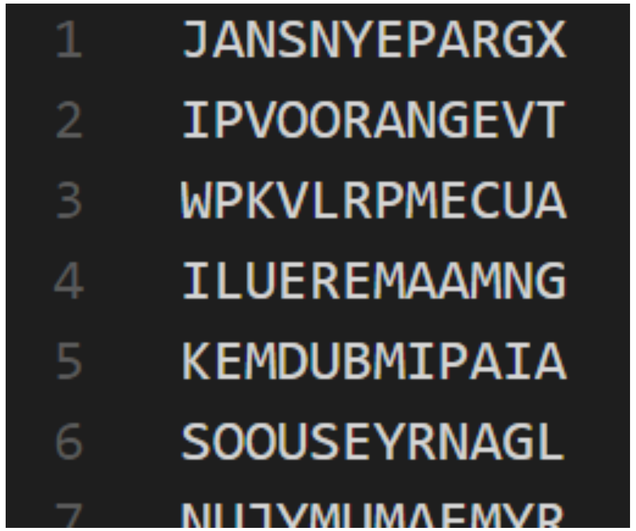
assumptions

- simple strategies (straight-forward solution)
- letters in both puzzle and word list are in upper-case
- assume all of the words in the word list are min. 4 letters long
- hidden word(s) in any directions (left to right, right to left, top to bottom, bottom to top, ..etc)
- simple text-based (ascii) puzzle, no Unicode or any international characters
- display puzzle in simple text-based format
- the puzzle is stored in a text file, line by line, puzzle is small, say 12 x 12, NOT 10000 x 10000!
- word list is stored in a text file with one word per line

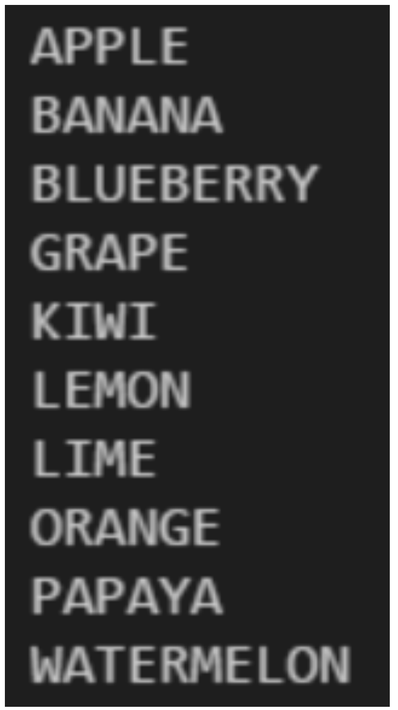
puzzle & word list files

puzzle

line by line

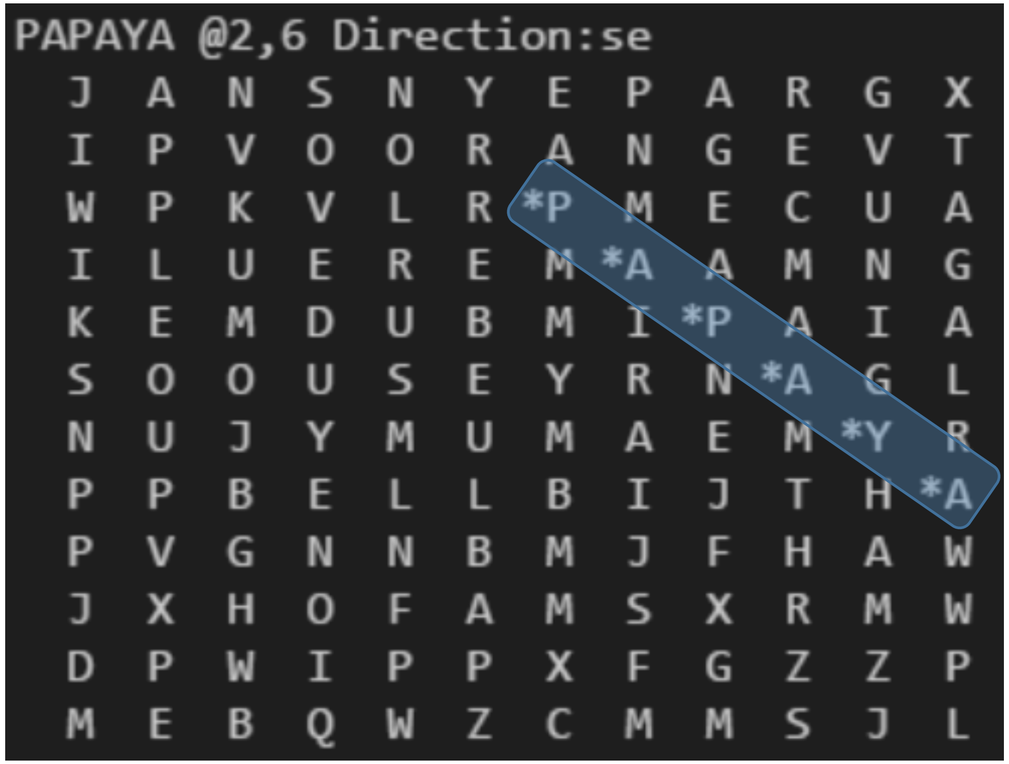


word list

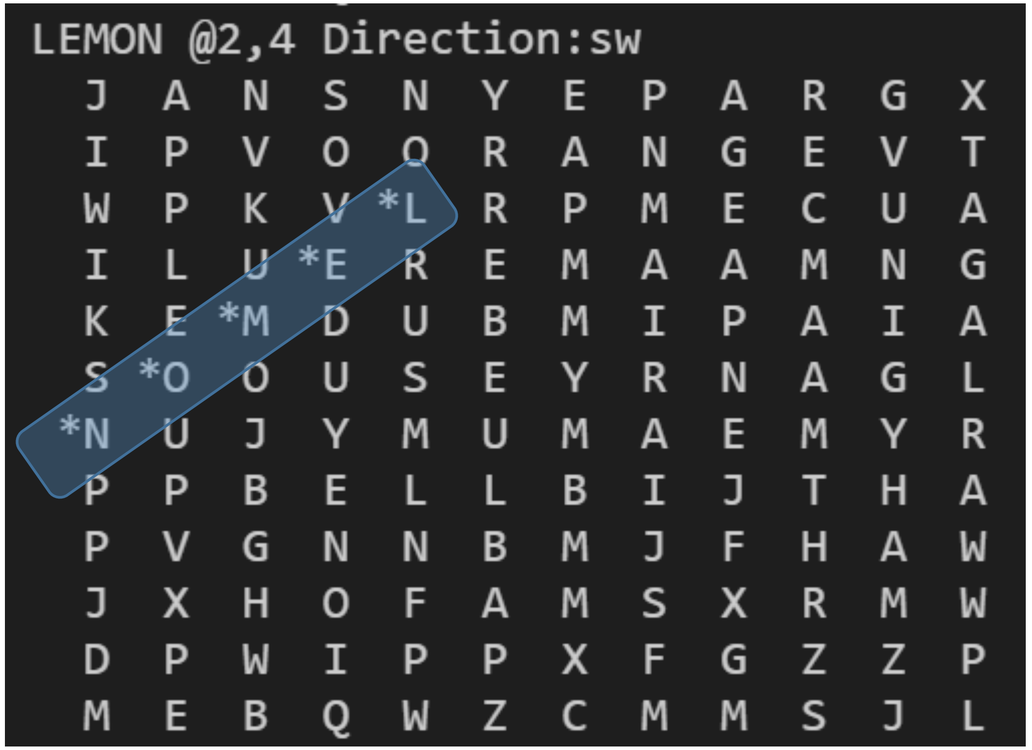


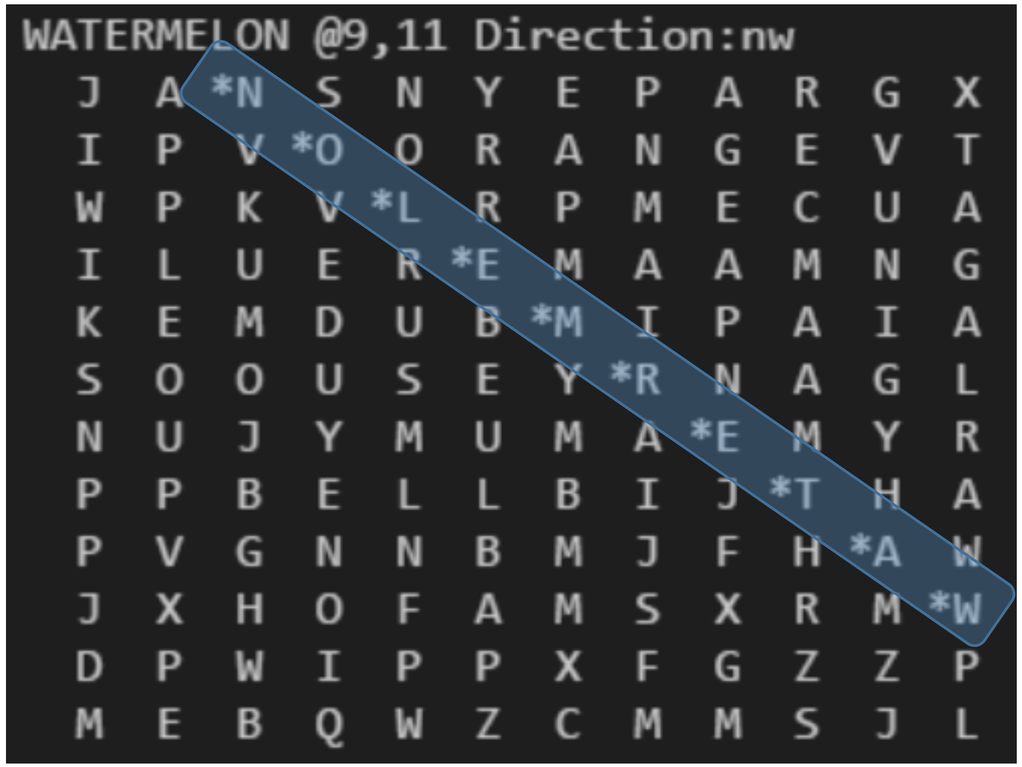
Words Found

```
J A N S N Y E P A R G X
I P V O O R A N G E V T
W P K V L R P M E C U A
I L U E R E M A A M N G
K E M D U B M I P A I A
S O O U S E Y R N A G L
N U J Y M U M A E M Y R
P P B E L L B I J T H A
P V G N N B M J F H A W
J X H O F A M S X R M W
D P W I P P X F G Z Z P
M E B Q W Z C M M S J L
```



Words Found


LEMON @2,4 Direction:sw
J A N S N Y E P A R G X
I P V O O R A N G E V T
W P K V *L R P M E C U A
I L U *E R E M A A M N G
K E *M D U B M I P A I A
S *O O U S E Y R N A G L
*N U J Y M U M A E M Y R
P P B E L L B I J T H A
P V G N N B M J F H A W
J X H O F A M S X R M W
D P W I P P X F G Z Z P
M E B Q W Z C M M S J L


WATERMELON @9,11 Direction:nw
J A *N S N Y E P A R G X
I P V *O O R A N G E V T
W P K V *L R P M E C U A
I L U E R *E M A A M N G
K E M D U B *M I P A I A
S O O U S E Y *R N A G L
N U J Y M U M A *E M Y R
P P B E L L B I J *T H A
P V G N N B M J F H *A W
J X H O F A M S X R M *W
D P W I P P X F G Z Z P
M E B Q W Z C M M S J L

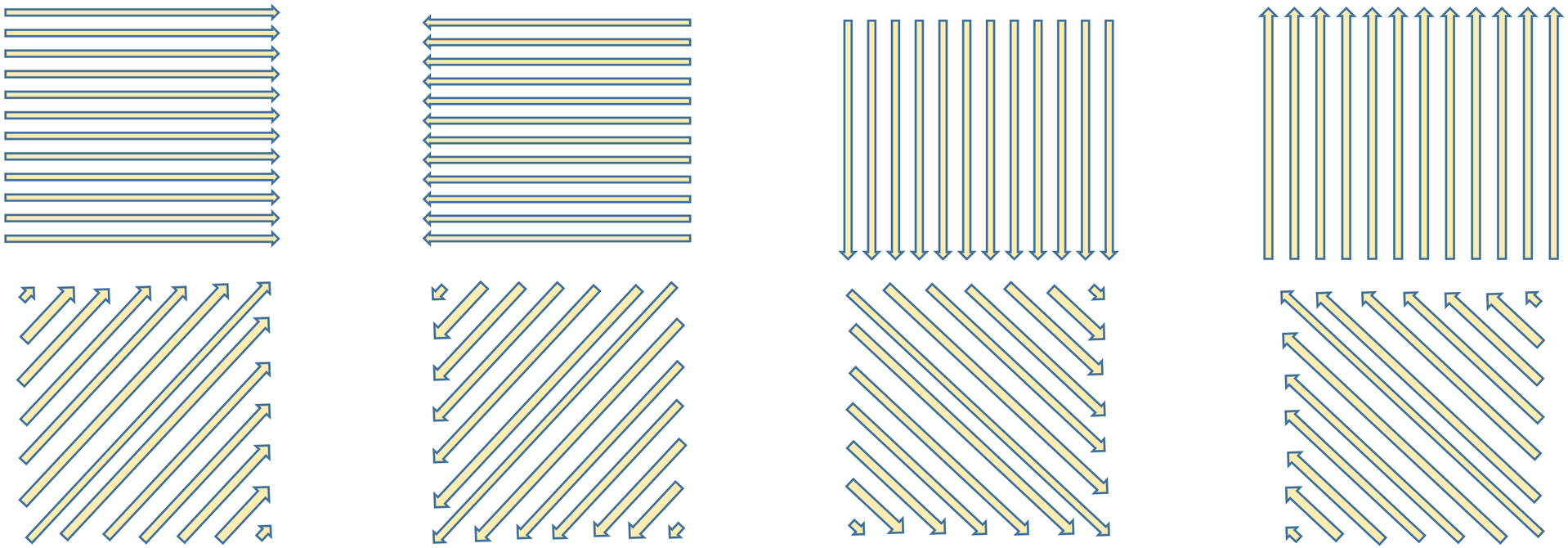
Design - Questions to Answer

- Game Setup
  - Load the puzzle from text file
  - Load the word list from text file
- Approach - Word Searching
  - Design, Problem Decomposition
  - Display the Words found
- Internal Representation of the word puzzle

Approach - Word Searching

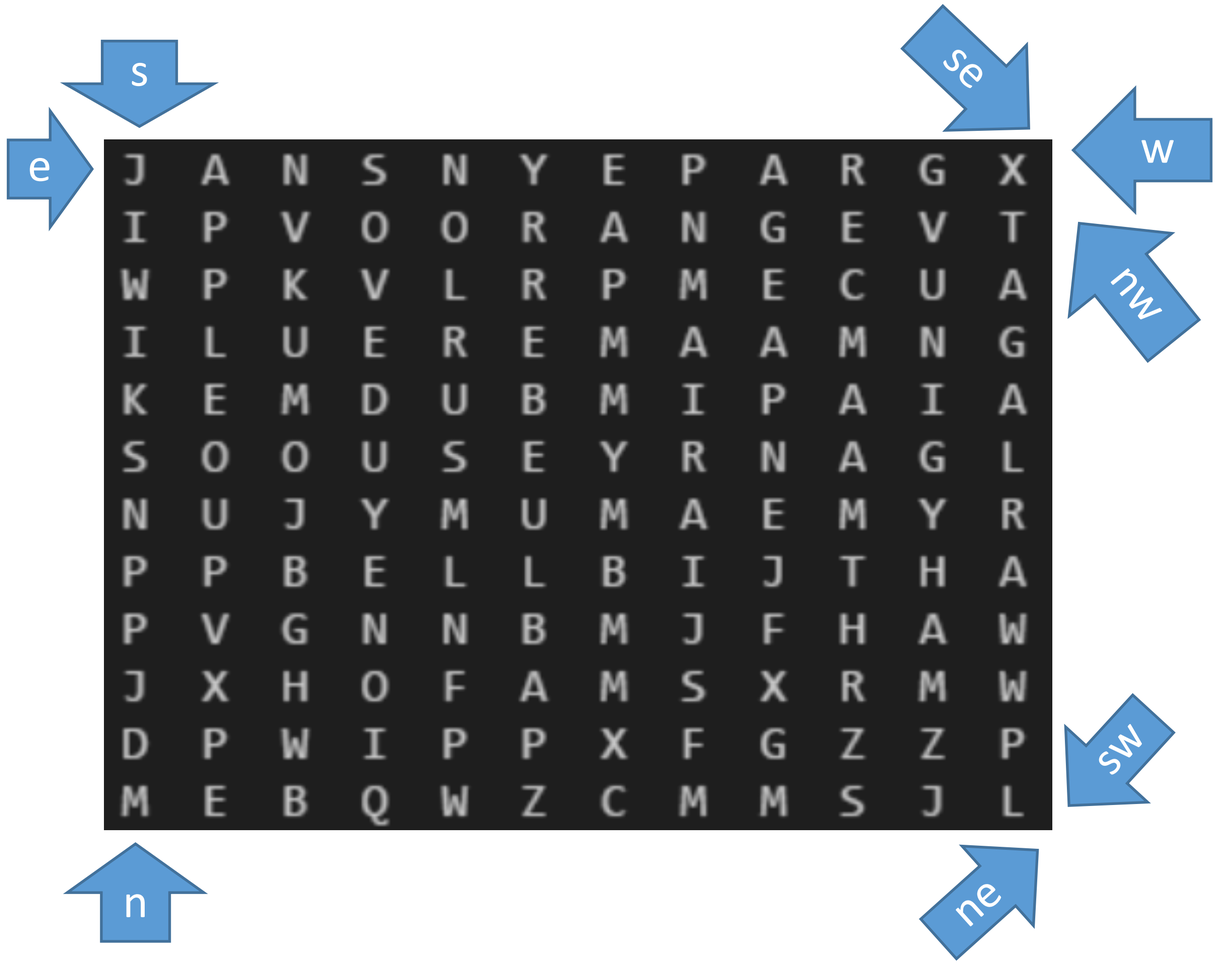
word-searching approach

- what about slicing up the puzzle as strings in each direction as follows:



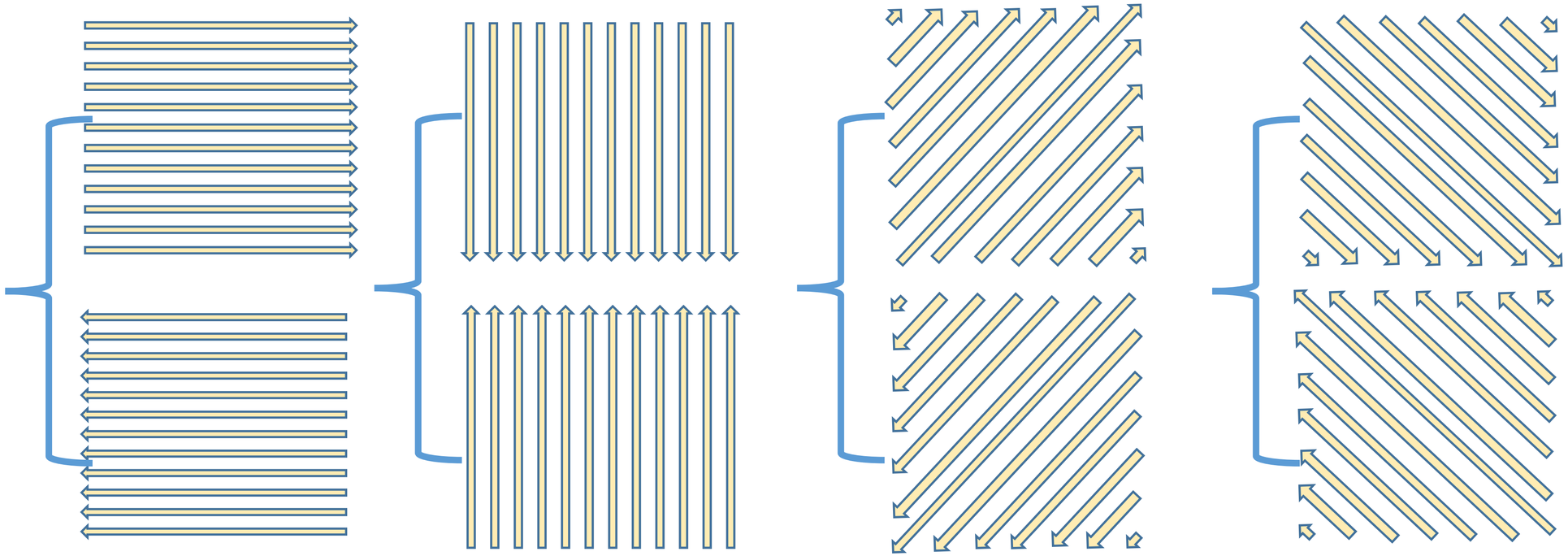
string sequence per direction

- ‘n’ – MDJPPNSKIWIJ, EPXVP….
- ‘s’ – JIWIKSNPPJDM, APPLE….
- ‘e’ – JANSNYEPARGX, IPVOO….
- ‘w’ – XGRAPEYNSNAJ, TVEGN….
- ‘ne’ – L, JP, SZW, MZMW…
- ‘sw’ – L, PJ, WZS, WMZM…
- ‘se’ – X, GT, RVA, AEUG…
- ‘nw’ – X, TG, AVR, GUEA…



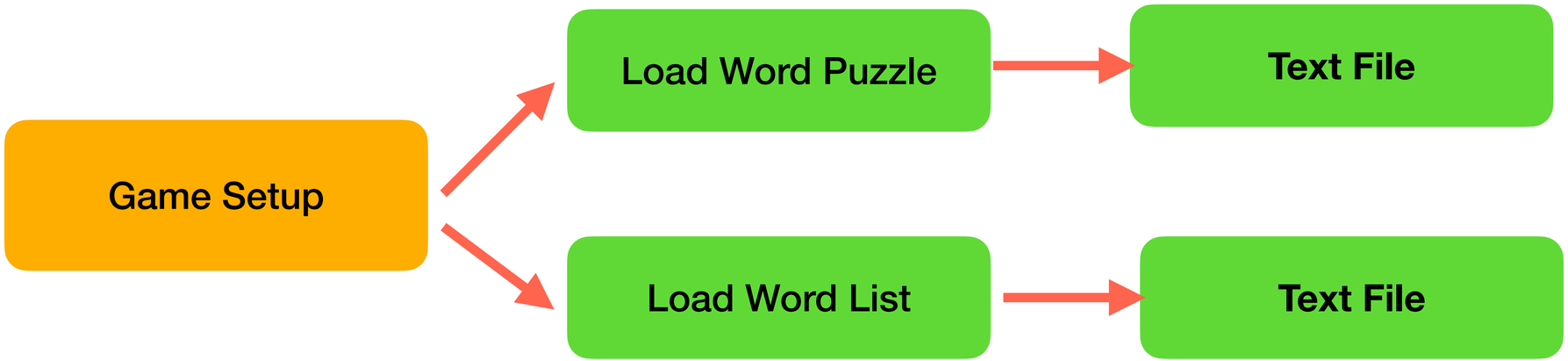
note on searching

- combine search in pair (e,w), (n,s), (ne,sw), (se,nw)



Top-Down Design - Game Setup

Problem Decomposition



Input Files - Puzzle and Word list

2024-Copilot > ws-puzzle.txt

```
 1   T W H O I U O O W M R B
 2   E M J B E O C O D Y K B
 3   I L U Q S D K Z H U M U
 4   A L Q L U X S W A E O K
 5   S R W I L N H F G V E I
 6   F T M R M G X U W B B N
 7   D P U F A V O R I T E L
 8   O L L D Q X J C V G V E
 9   H P E V E M T Z T W N Y
10   V Y N Z I N Y O C L K S
11   U G W W N Z T A O F U B
12   H N P T F B I S U L M S
13
```

2024-Copilot > ws-dict.txt

```
 1   APPLE
 2   BANANA
 3   BLUEBERRY
 4   GRAPE
 5   KIWI
 6   LEMON
 7   LIME
 8   ORANGE
 9   PAPAYA
10   WATERMELON
11   CHARACTER
12   KINLEY
13   PIANO
14   PUZZLE
15   POSSIBLE
16   VIOLIN
```

Internal Representation

data model - puzzle

- how to represent the puzzle?
  - a single string
    - “J A N S N Y E P A R G X I P V O O R A N G E …”
  - a single-dimensional list
    - [‘J’, ‘A’, ‘N’, ‘S’, ‘N’, ‘Y’, ‘E’, ‘P’, ‘A’, ‘R’ …]
  - a list of string

    [“J A N S N Y E P A R G X”,

    “I P V O O R A N G E V T”, …]
  - a nested list (**)
    - [[‘J’, ‘A’, ‘N’, ‘S’, ‘N’, ‘Y’, ‘E’, ‘P’, ‘A’, ‘R’ …],

      [‘I’, ‘P’, ‘V’, ‘O’, ‘O’ ……], ….]

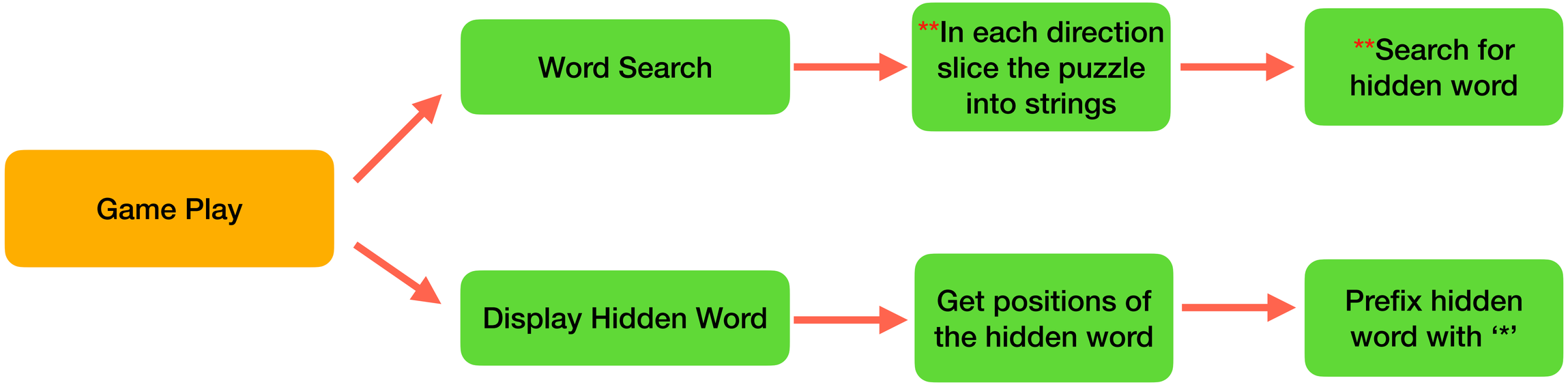
data types – puzzle & word list

```
g_puzzle = [
        ['J', 'A', 'N', 'S', 'N', 'Y', 'E', 'P', 'A', 'R', 'G', 'X'],
        ['I', 'P', 'V', 'O', 'O', 'R', 'A', 'N', 'G', 'E', 'V', 'T'],
        ['W', 'P', 'K', 'V', 'L', 'R', 'P', 'M', 'E', 'C', 'U', 'A'],
        ['I', 'L', 'U', 'E', 'R', 'E', 'M', 'A', 'A', 'M', 'N', 'G'],
        ['K', 'E', 'M', 'D', 'U', 'B', 'M', 'I', 'P', 'A', 'I', 'A'],
        ['S', 'O', 'O', 'U', 'S', 'E', 'Y', 'R', 'N', 'A', 'G', 'L'],
        ['N', 'U', 'J', 'Y', 'M', 'U', 'M', 'A', 'E', 'M', 'Y', 'R'],
        ['P', 'P', 'B', 'E', 'L', 'L', 'B', 'I', 'J', 'T', 'H', 'A'],
        ['P', 'V', 'G', 'N', 'N', 'B', 'M', 'J', 'F', 'H', 'A', 'W'],
        ['J', 'X', 'H', 'O', 'F', 'A', 'M', 'S', 'X', 'R', 'M', 'W'],
        ['D', 'P', 'W', 'I', 'P', 'P', 'X', 'F', 'G', 'Z', 'Z', 'P'],
        ['M', 'E', 'B', 'Q', 'W', 'Z', 'C', 'M', 'M', 'S', 'J', 'L']
    ]
```

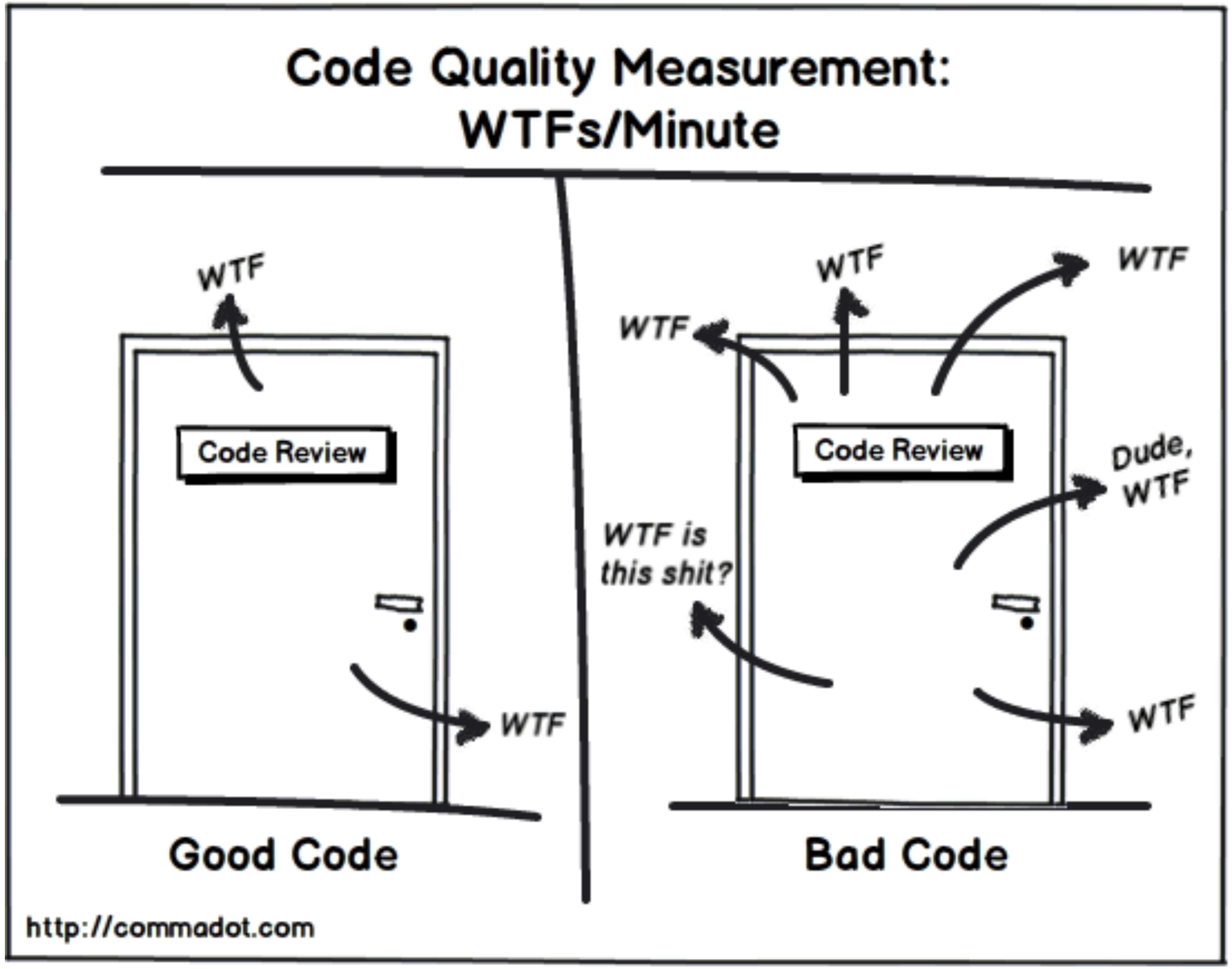
```
g_dictword = ['APPLE', 'BANANA', 'BLUEBERRY', 'GRAPE', 'KIWI', 'LEMON', 'LIME', 'ORANGE', 'PAPAYA', 'WATERMELON']
```

Top-Down Design - Game Play

Problem Decomposition



Goal #2 - Clean Code



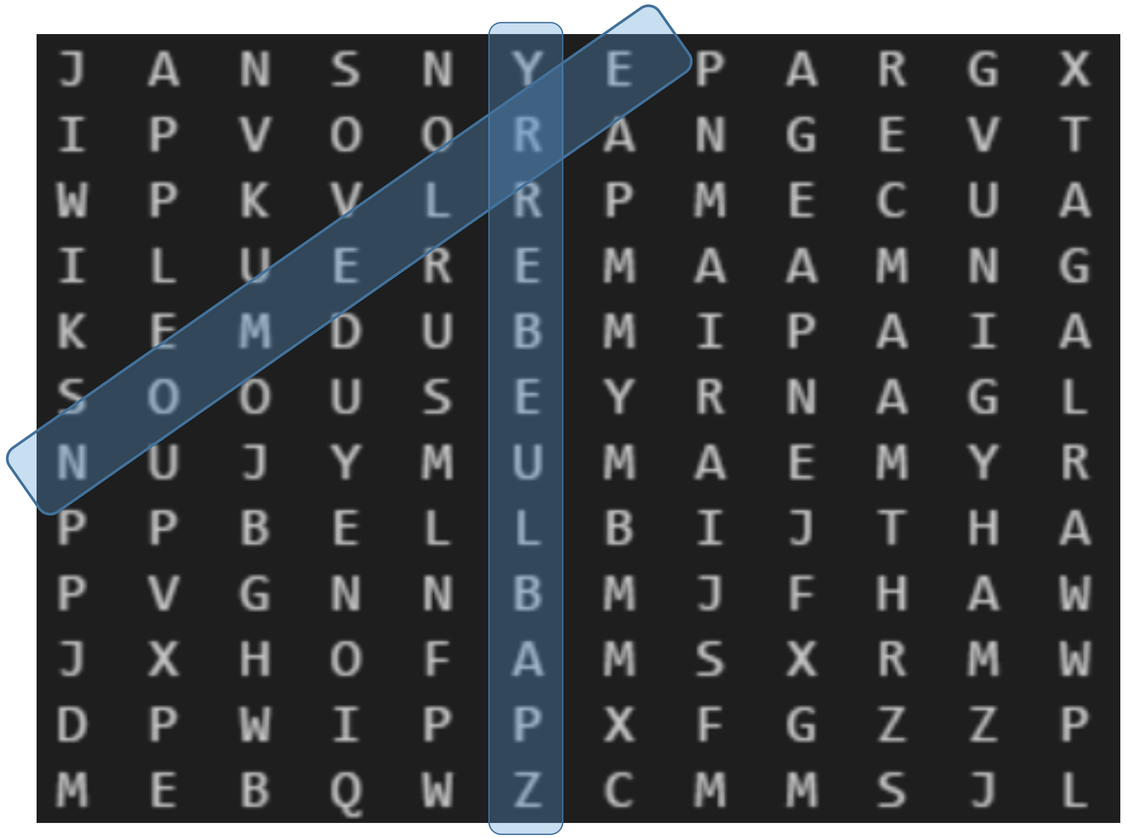
**In each direction slice the puzzle into strings

wait a minute !!!!!! what about slicing !!!!

- can we not use slicing ?????
- p[row] E
- p[row][::-1] W
- [x[col] for x in p] S
- [x[col] for x in p][::-1] N

What about NE, NW ???

Found Word ??



**Search for hidden word

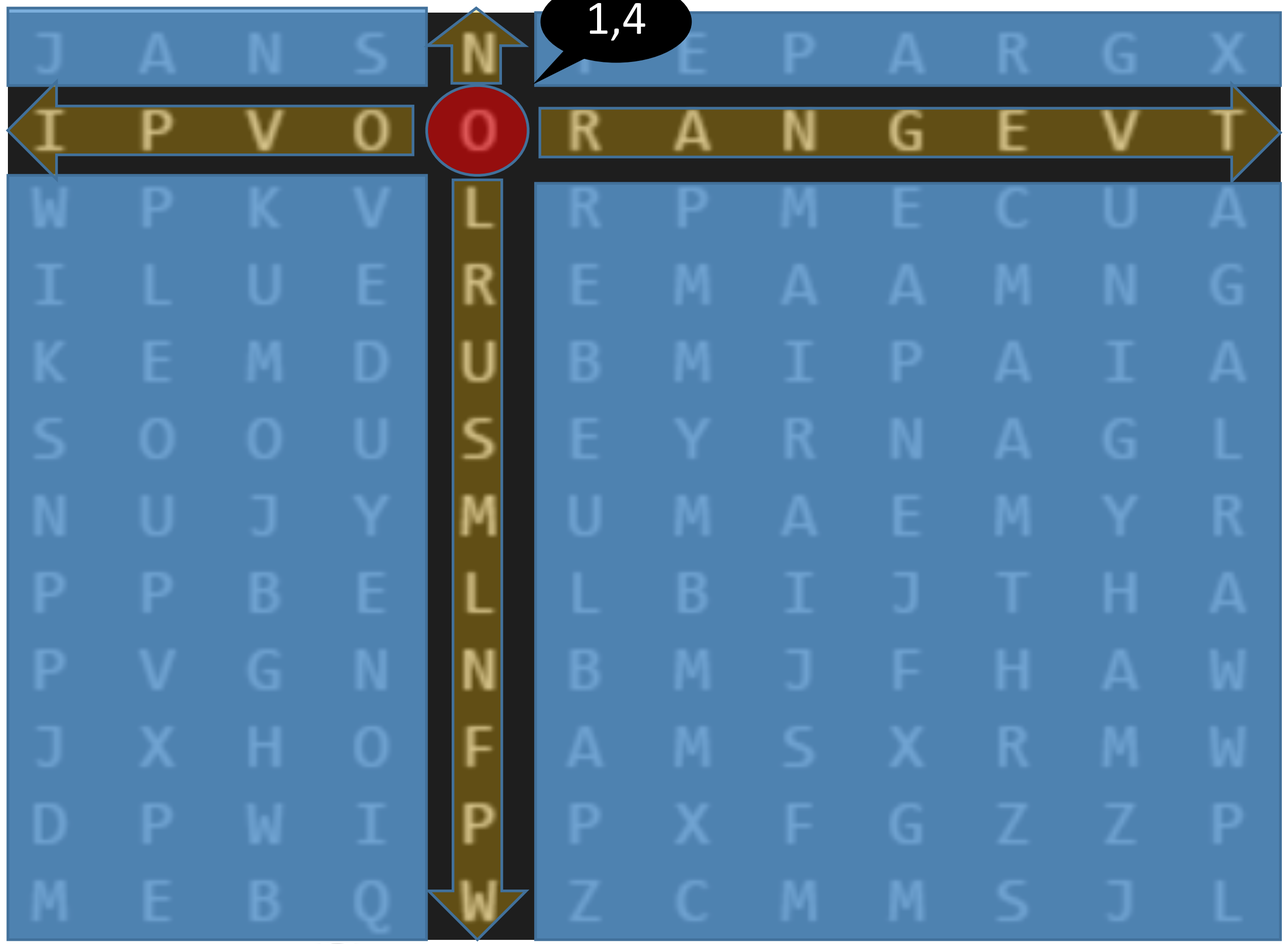
N

O N

W

O O V P I

1,4



E

O R A N G E V T

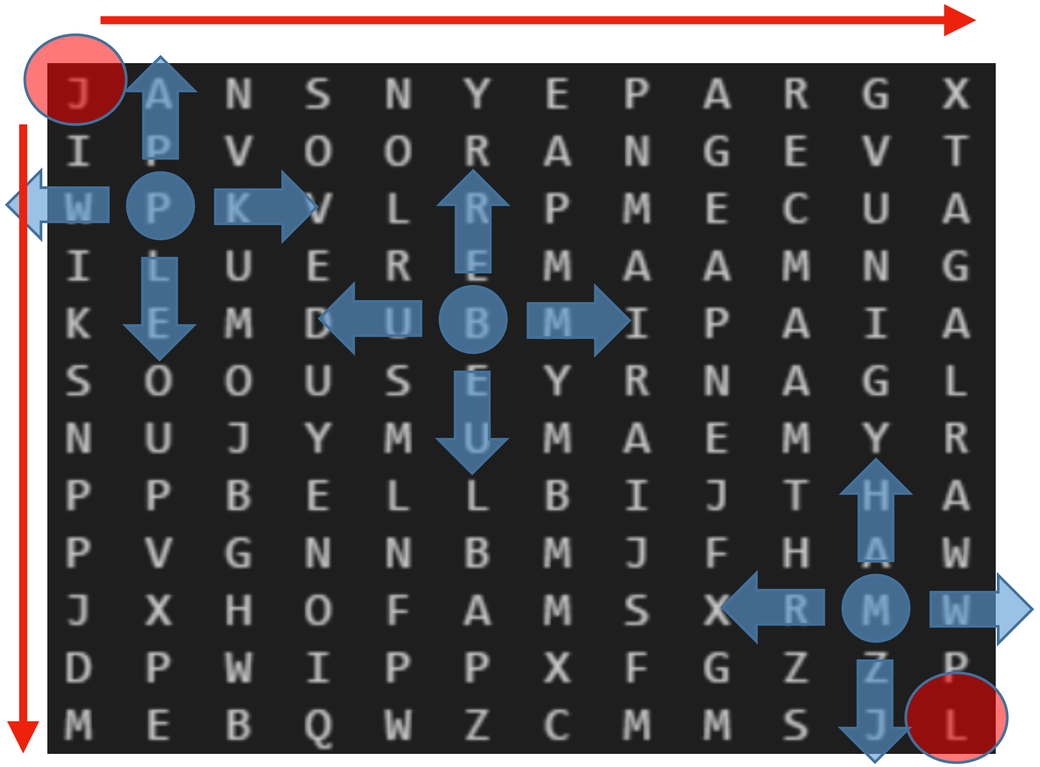
S

O L R U S M L N F P W

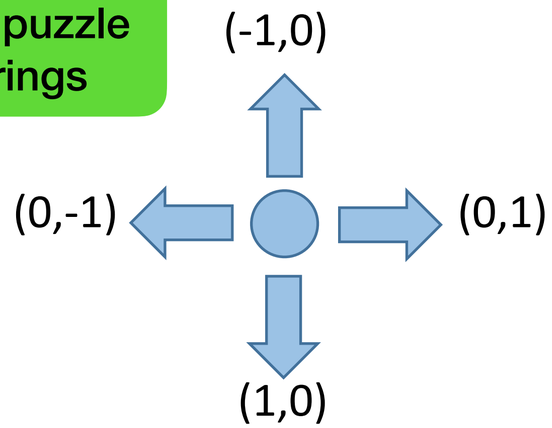
KISS approach - brute force search

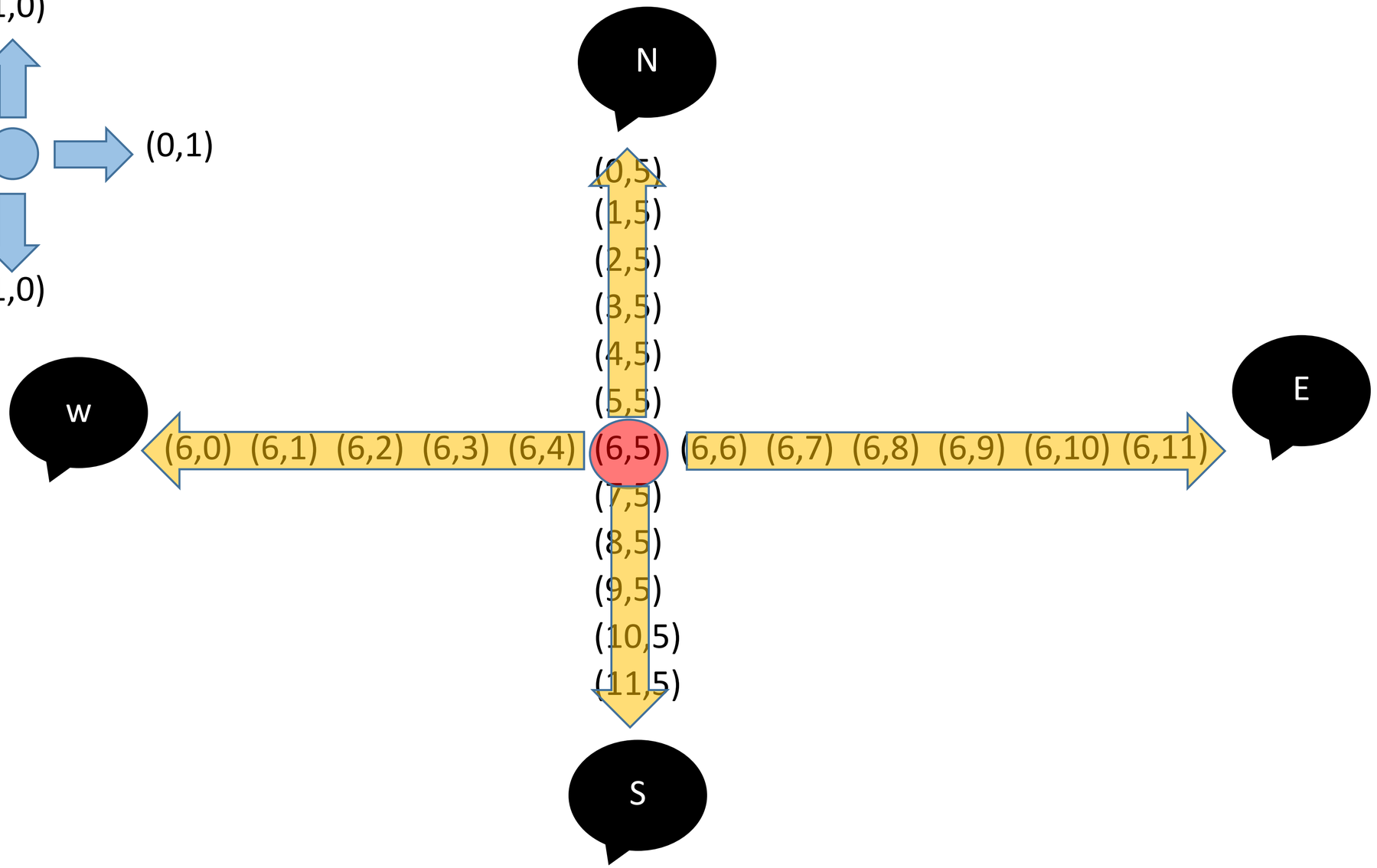
Word Search

- iterate each & every single location
  - start from upper left corner (0,0)
  - end at lower right corner (N-1,N-1)
- at location (row, col), for each direction
  - return all letters as a string in order of the given direction
  - check if any word is a prefix of the string (use words with same first letter as needed)
  - display hidden word at location (row,col) in given direction
    - prefix each letter with a character ‘*’



**In each direction slice the puzzle into strings





```
 3    def get_next_location(p_row, p_col, p_dir):
 4 >      """…
15          row, col = p_row, p_col
16
17          if p_dir == 'n':
18              row -= 1
19          elif p_dir == 's':
20              row += 1
21          elif p_dir == 'e':
22              col += 1
23          elif p_dir == 'w':
24              col -= -1
25
26          return row, col
```

get_next_location – data model

- define the direction and incremental steps
- use “dictionary” and “tuple” to store the step size in each direction
- steps = {

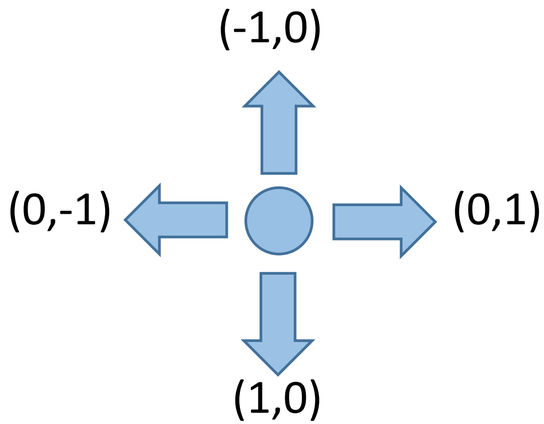
    ‘n’: (-1,0),

    ‘e’: (0,1),

    ‘s’: (1,0),

    ‘w’: (0,-1)

  }



```python
def get_next_location(p_row:int, p_col:int, p_dir:str) -> tuple[int,int]:
    """…
    row_step, col_step = g_step.get(p_dir, (0, 0))
    next_row, next_col = p_row + row_step, p_col + col_step
    n_row, n_col = len(g_puzzle), len(g_puzzle[0])
    if next_row in range(n_row) and next_col in range(n_col):
        return next_row, next_col
    return -1, -1
```

Design - Functions

- get_next_location(row, col, dir) -> row, col
- get_all_locations(row, col, dir) -> [(row,col)]
- get_all_letters(row, col, dir) -> str
- display_puzzle(puzzle) -> None
- show_hidden_word(row, col, dir, word_found) -> None
- solve_puzzle() -> None
- Parameters:
  - col, row are integers, representing the (x,y) position in the puzzle
  - dir is the cardinal directions (n, e, w, s, ne, …) as string, lower case
  - puzzle is the nested list representing the word puzzle
  - word_found is the hidden word discovered

Main Processing Logic + Search Engine

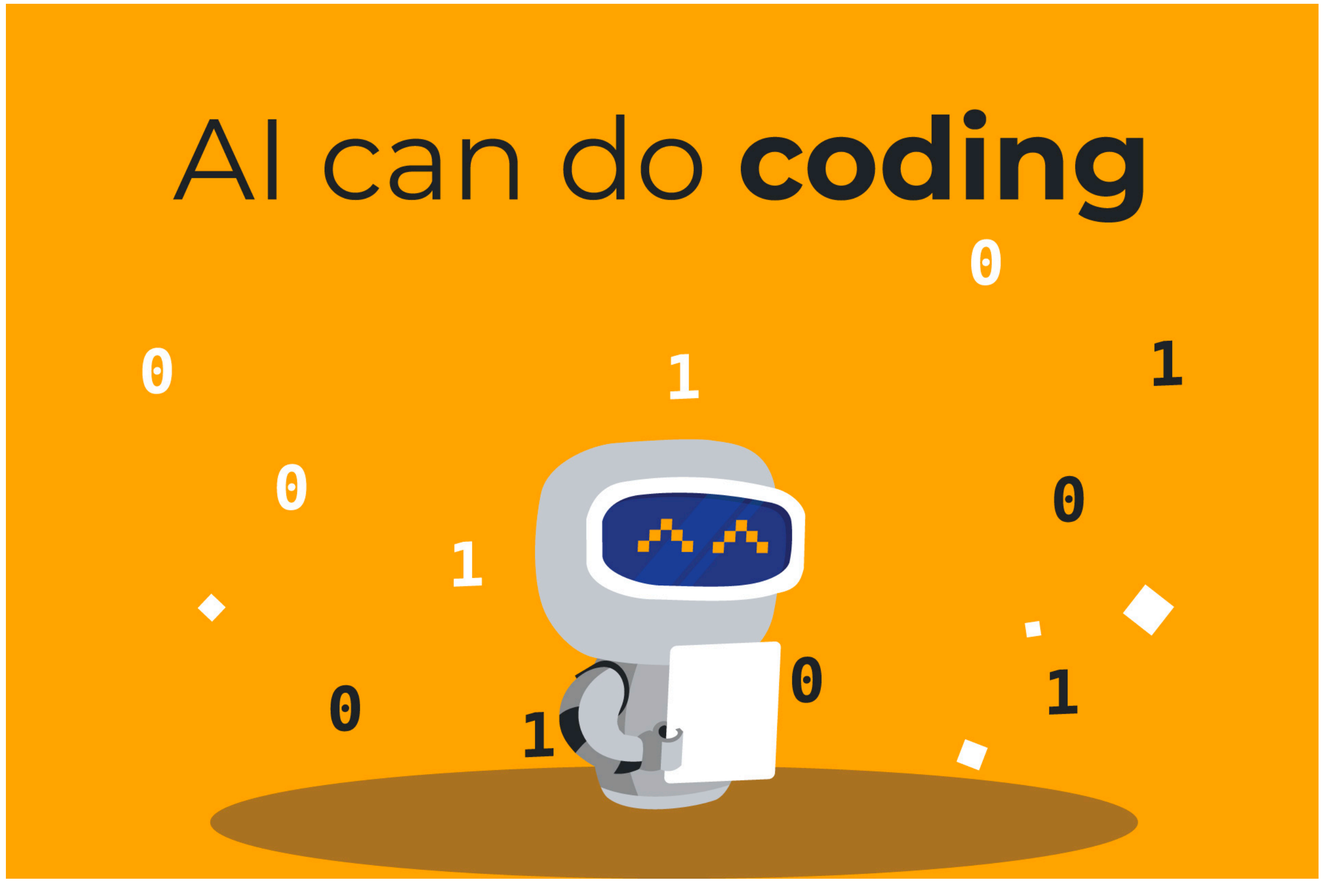
```
def search_word(p_row:int, p_col:int, p_dir:str, p_min=4) -> None:
    """…
    letters = get_all_letters(p_row, p_col, p_dir)
    if len(letters) < p_min:
        return

    for word in g_dictword:
        if letters.startswith(word):
            show_hidden_word(p_row, p_col, p_dir, word)
```

```
sz_across = len(g_puzzle[0])
sz_down = len(g_puzzle)
for row in range(sz_down):
    for col in range(sz_across):
        for dir in g_step.keys():
            search_word(row, col, dir)
```

Goal #1 - AI Assisted Coding

CHAT



Live Coding - Demo prior to AI

Implementation - AI

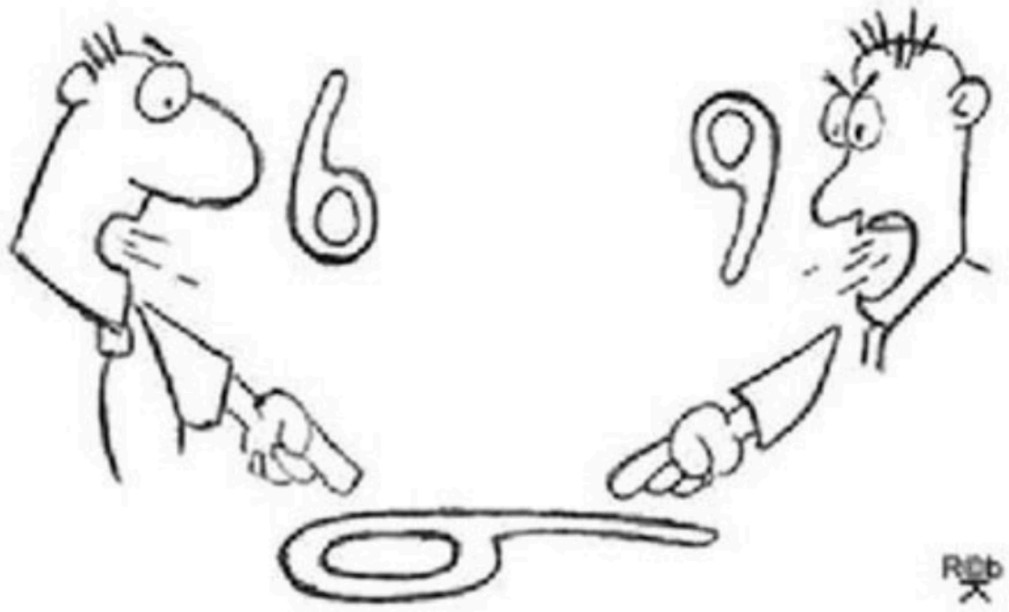
Scope Change

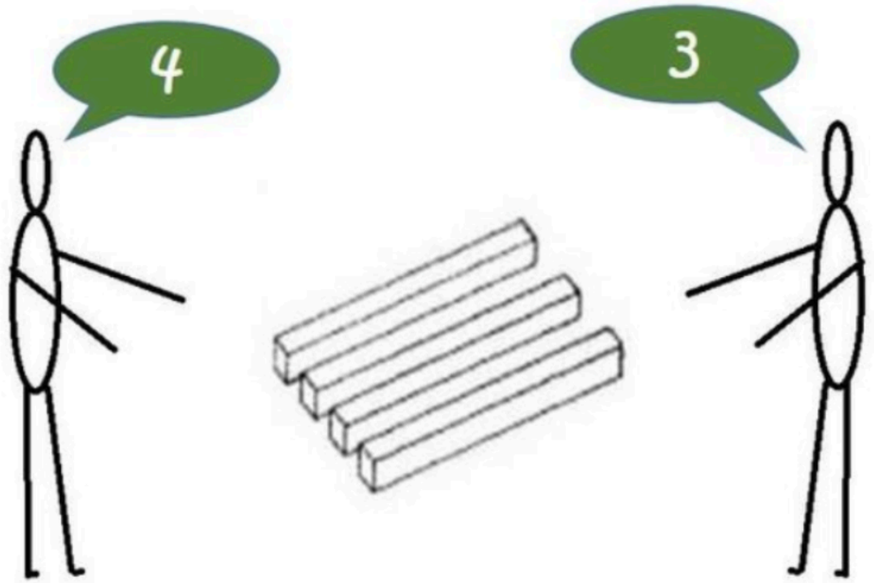
assumptions

Only one hidden word in any directions

- simple strategies (straight-forward solution)
- letters in both puzzle and word list are in upper-case
- assume all of the words in the word list are min. 4 letters long
- ~~hidden word(s)~~ in any directions (left to right, right to left, top to bottom, bottom to top, ..etc)
- simple text-based (ascii) puzzle, no Unicode or any international characters
- display puzzle in simple text-based format
- the puzzle is stored in a text file, line by line, puzzle is small, say 12 x 12, NOT 10000 x 10000!
- word list is stored in a text file with one word per line

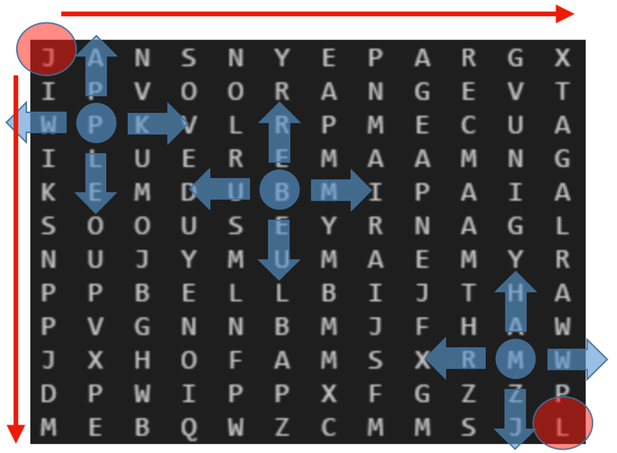
Design - Tradeoff

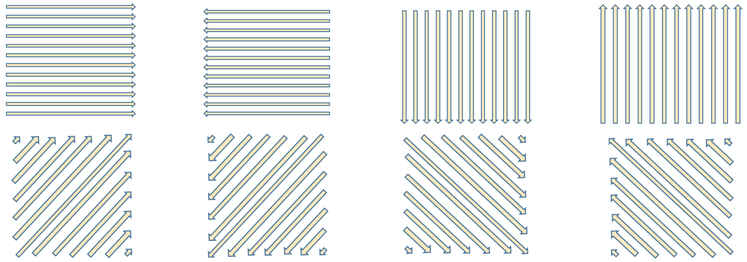


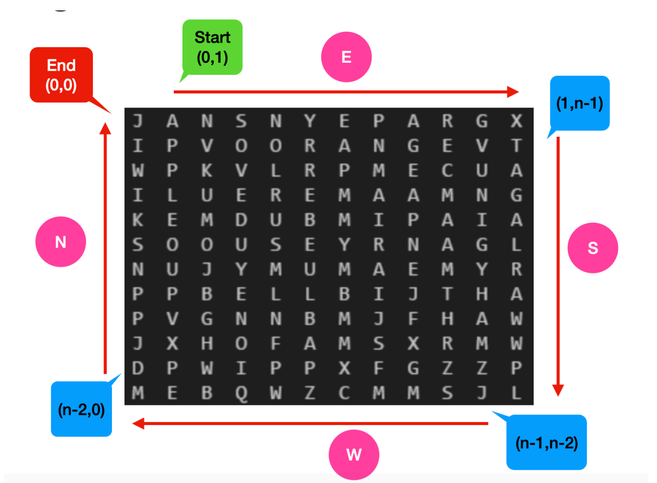


Computation Time

- KISS Approach:
  - N x N x 8 times of calling search_word()
- Slicing Plan A - Along 4 Edges with SINGLE direction
  - N x 8 times of calling search_word()
- Slicing Plan B - Along 4 Edges with ALL directions
  - N x 16 times of calling search_word()
- For N = 10:
  - KISS = 800, Plan A = ~80, Plan B = ~160
  - KISS = O(n^2), Plan A = 0(n), Plan B = O(n)



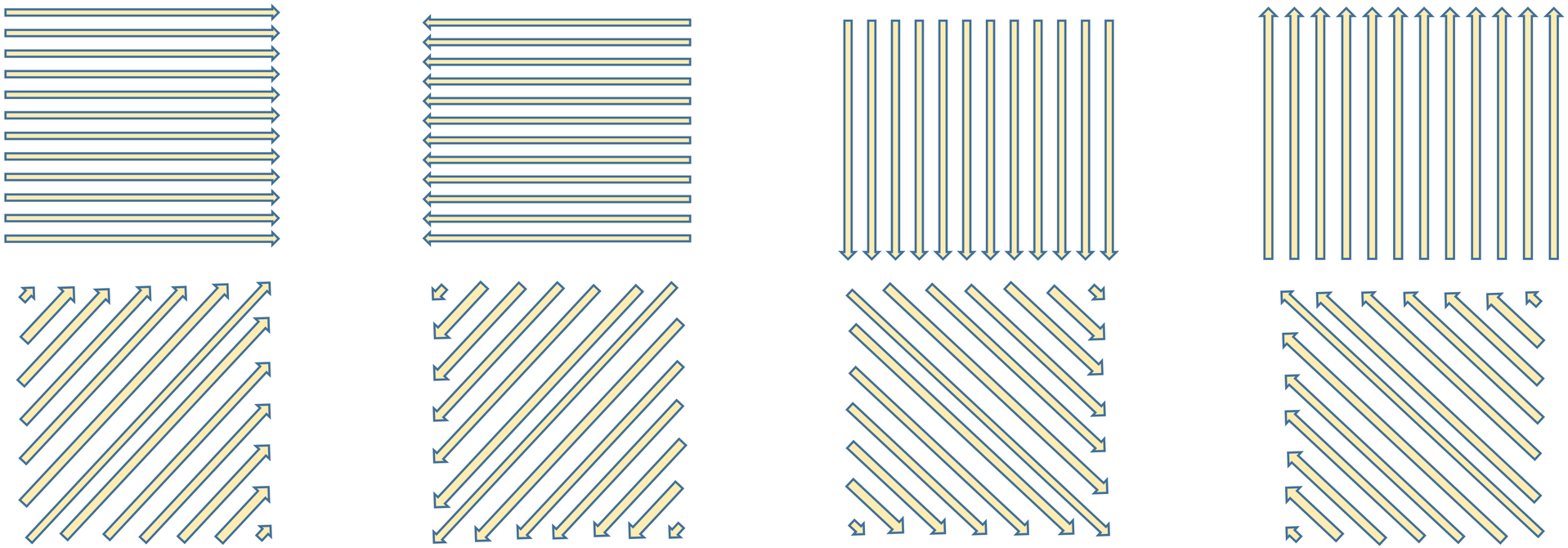




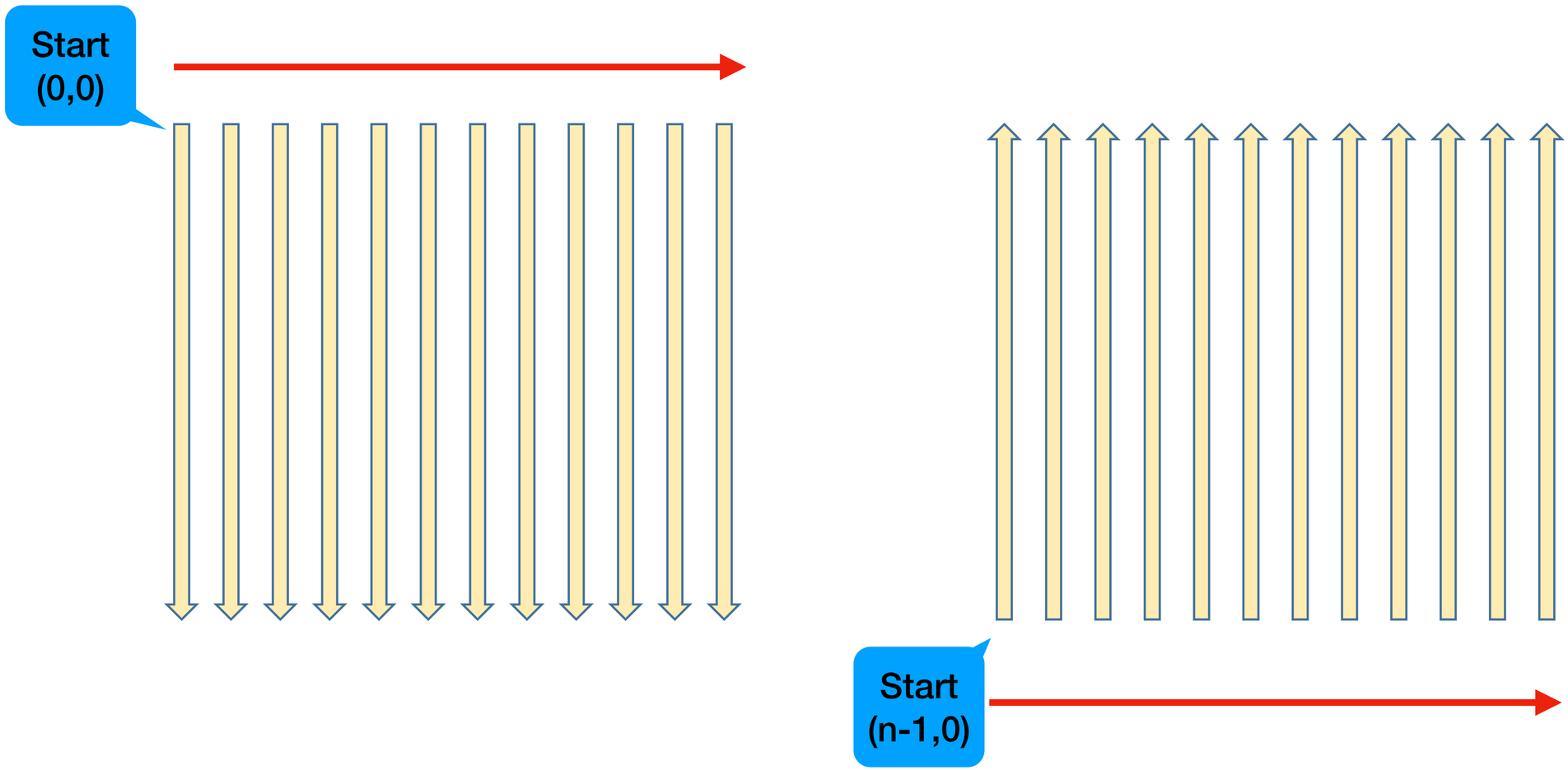
Proposal 1 - Slicing Plan A

Slicing Plan A

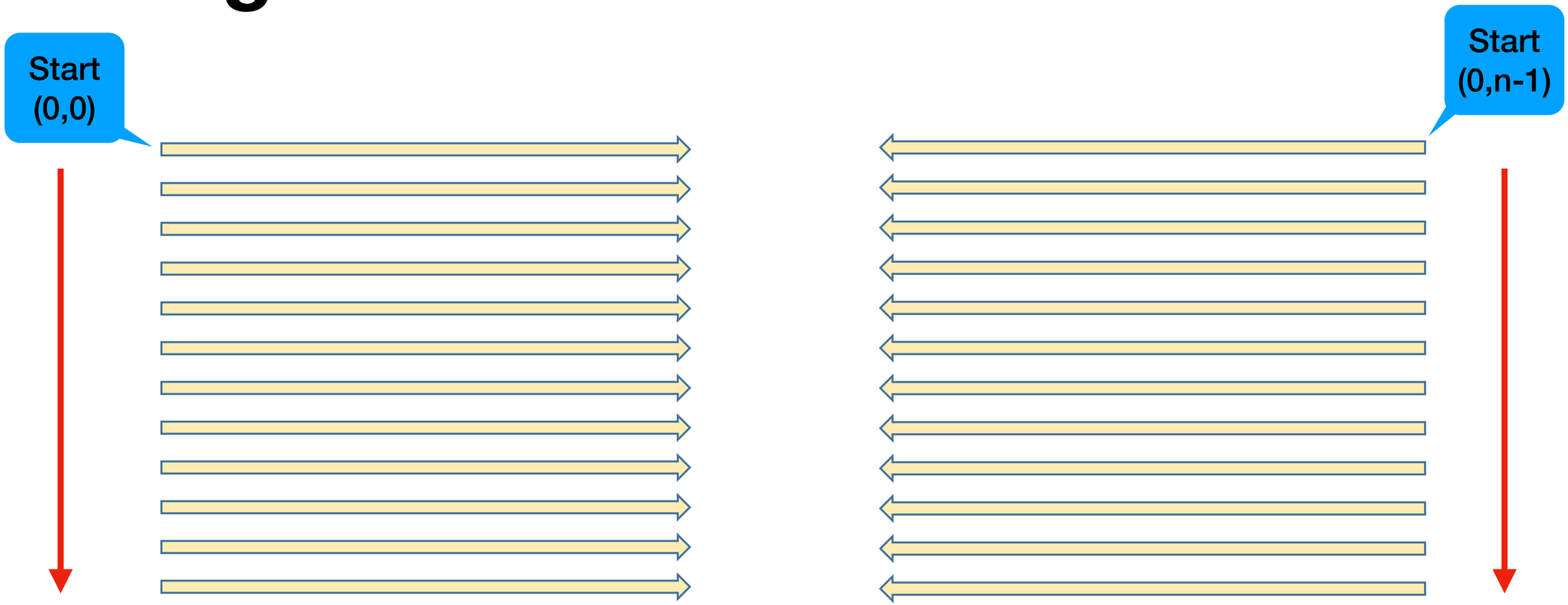
- what about slicing up the puzzle as strings in each direction as follows:



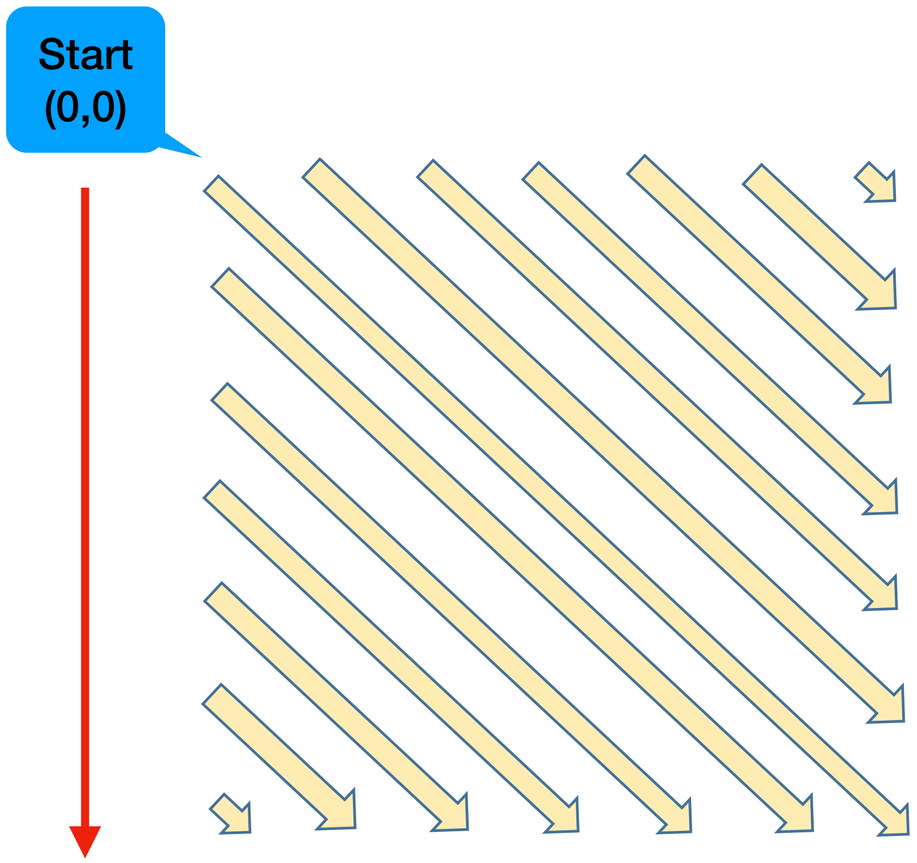
Slicing - South & North

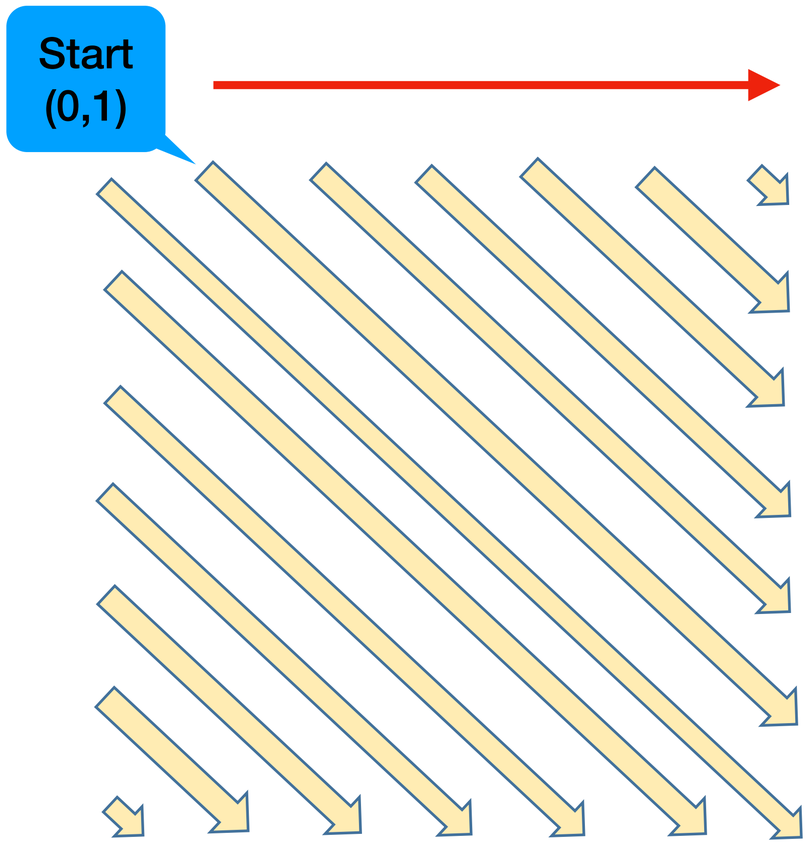


Slicing - East & West

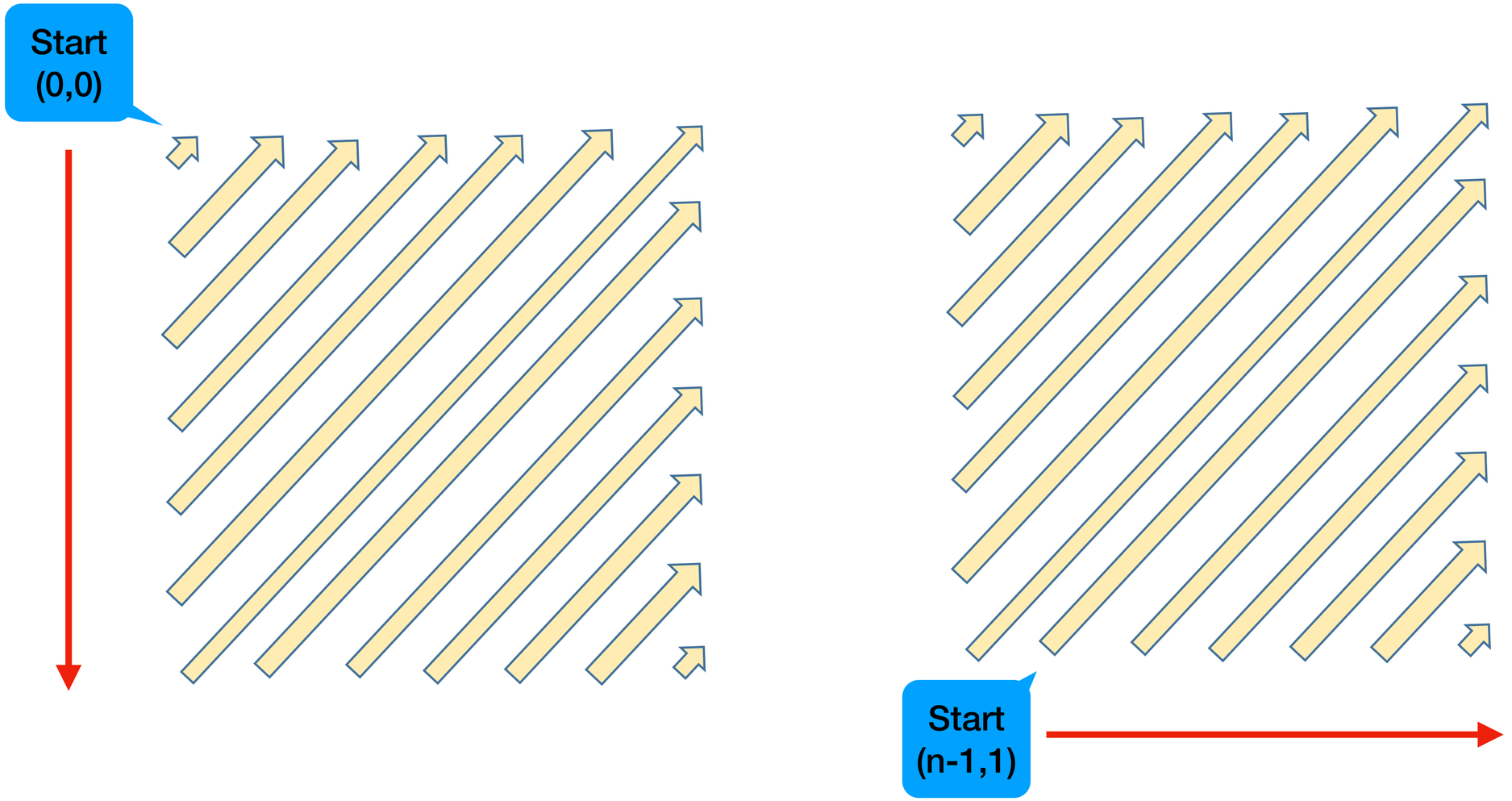


Slicing - South East

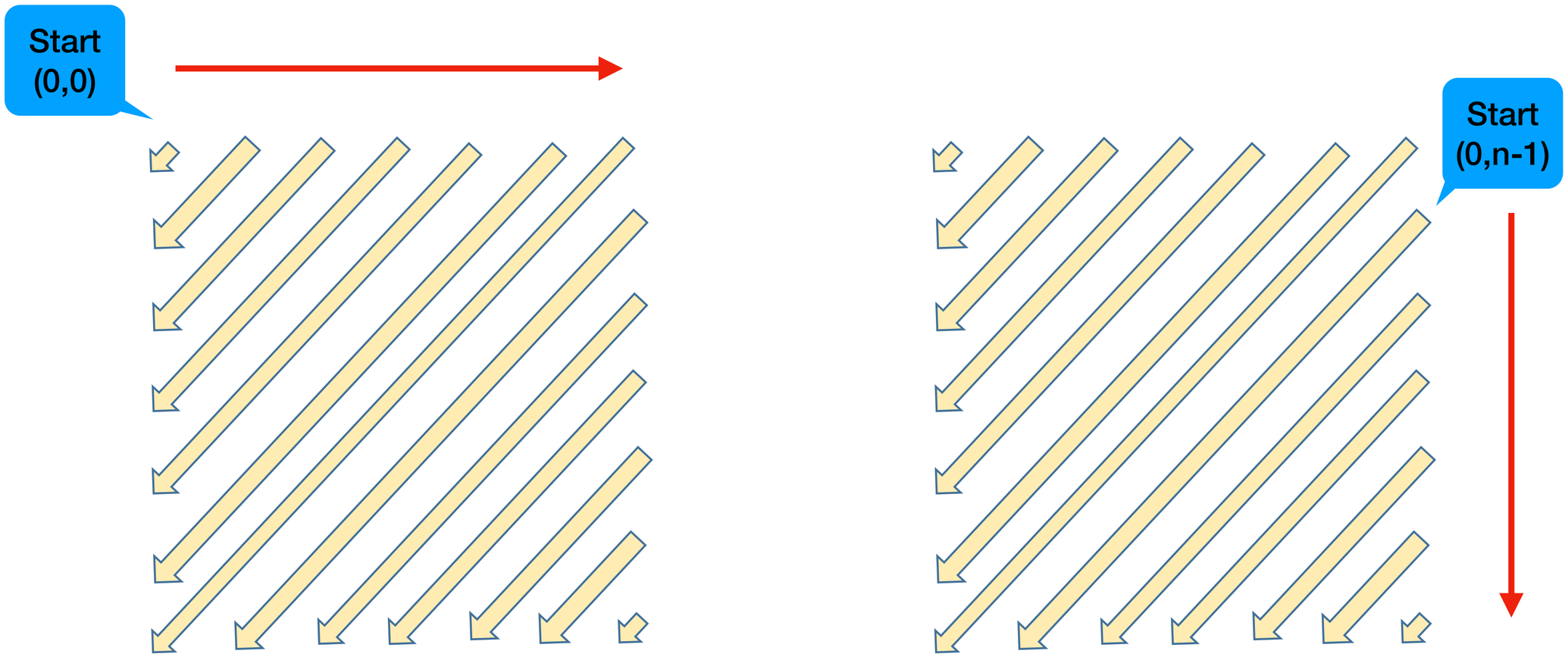




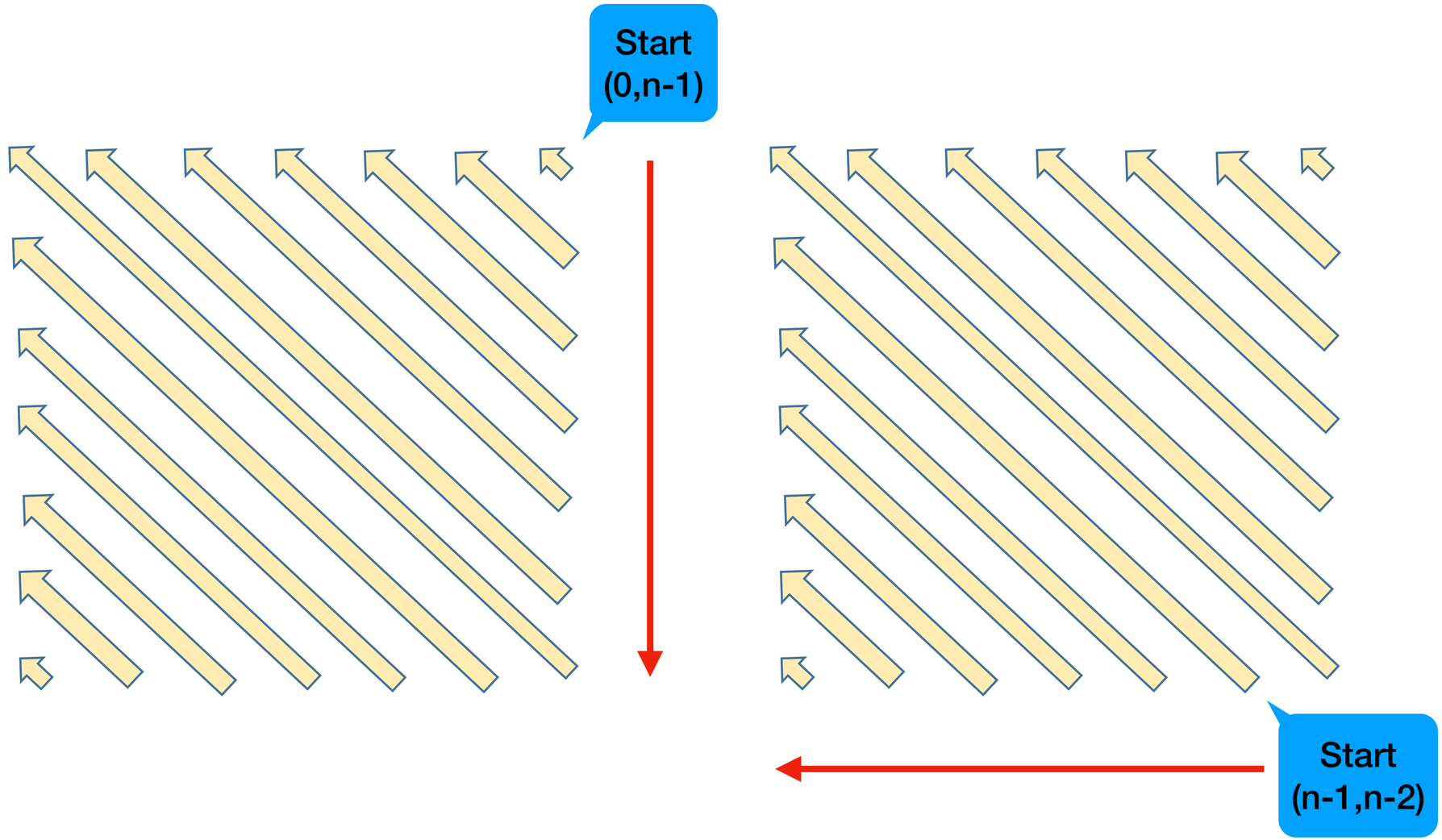
Slicing - North East



Slicing - South West



Slicing - North West



Implementation

```
search_dir = {
        's':'e',
        'e':'s',
        'se_00_south':'s',
        'se_01_east':'e',
        'ne_00_south':'s',
        'ne_n1_south':'e',

        'n':'e',
        'w':'s',
        'sw_00_east':'e',
        'sw_0n_south':'s',
        'nw_0n_north':'s',
        'nw_nn_west':'w'
      }
```

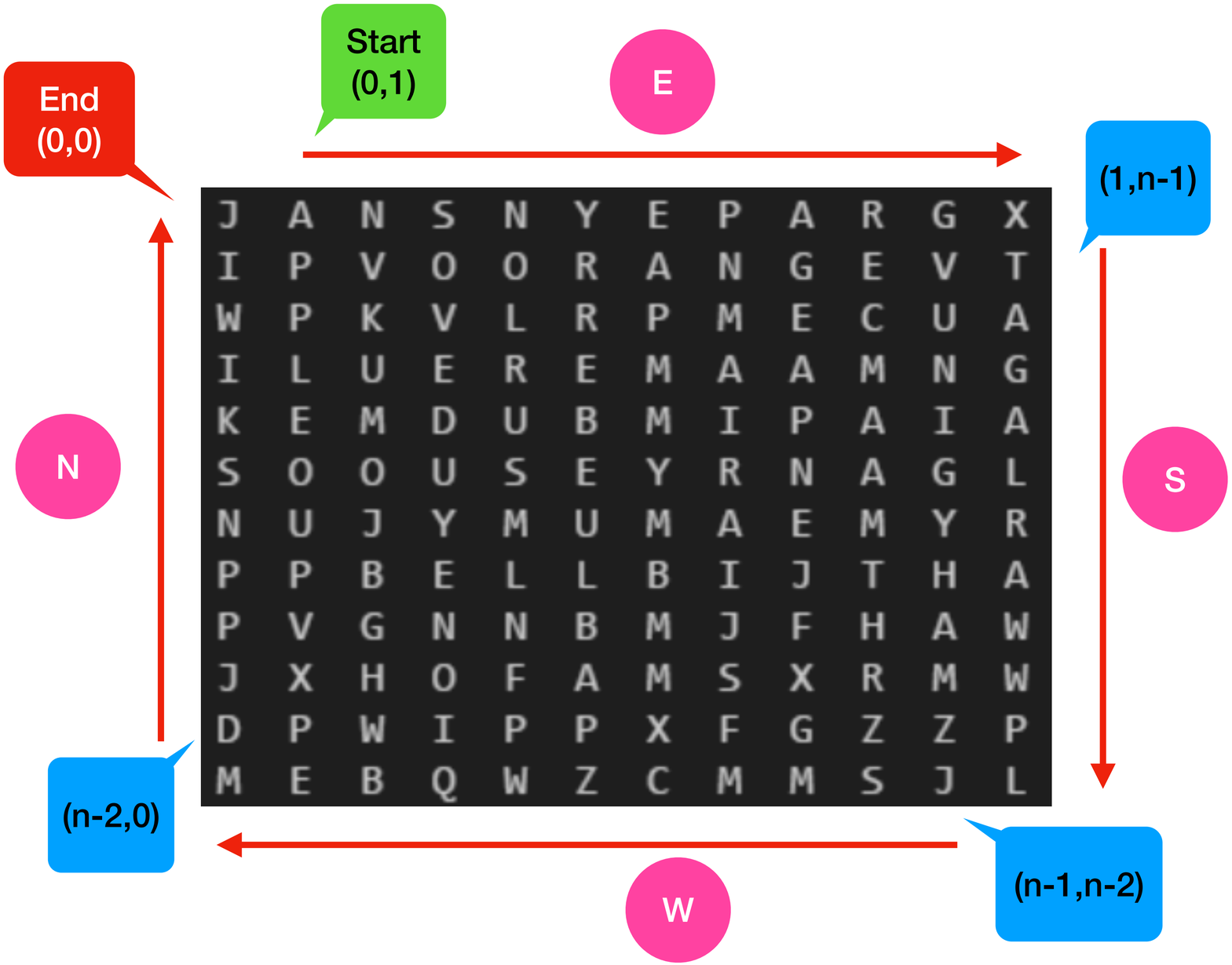
```
search_start = {
        'se_01_east': (0, 1),
        'ne_n1_south': (sz_down-1, 1),

        'n': (sz_down-1,0),
        'w': (0, sz_across-1),
        'sw_0n_south': (0, sz_across-1),
        'nw_0n_north': (0, sz_across-1),
        'nw_nn_west': (sz_down-1,  sz_across-2),
        }
```

```
for dir, next_dir in search_dir.items():
    row, col = search_start.get(dir, (0,0))
    while row != -1:
        search_word(row, col, dir[:2])
        row, col = get_next_location(row, col, next_dir)
```

Proposal 1 - Slicing B

Slicing - Plan B



Implementation

```
corners = {
        'e': (0,1),
        's': (1,sz_across-1),
        'w': (sz_down-1, sz_across-2),
        'n': (sz_down-2,0)
        }
```

```
for corner_dir in corners.keys():
    row, col = corners[corner_dir]
    while row != -1:
        for dir in g_step.keys():
            search_word(row, col, dir)
        row, col = get_next_location(row, col, corner_dir)
```

Implementation

```
corners = {
        'e': (0,1),
        's': (1,sz_across-1),
        'w': (sz_down-1, sz_across-2),
        'n': (sz_down-2,0)
        }
```

Will discover same hidden word multiple times

```
for corner_dir in corners.keys():
    row, col = corners[corner_dir]
    while row != -1:
        for dir in g_step.keys():
            search_word(row, col, dir)
        row, col = get_next_location(row, col, corner_dir)
```

word_search() - updated version

```
def search_word(p_row:int, p_col:int, p_dir:str, p_min=4) -> None:

    letters = get_all_letters(p_row, p_col, p_dir)

    if len(letters) < p_min:
        return False

    for word in g_dictword:
        idx = letters.find(word)
        if idx != -1:
            row_step, col_step = g_step[p_dir]
            row, col = p_row + row_step * idx, p_col + col_step * idx
            show_hidden_word(row, col, p_dir, word)
            g_dictword.remove(word)
```

The End - Thank You