Assignment 1: Coding with Reflection

Name of puzzle: Shadows of the Knight - Episode 1

Language used: Python3

Joshua Moore

Electrical & Computer Engineering

Mississippi State University

Spend 15 minutes working on the problem.

I set an alarm for 15 minutes and worked on the problem. I considered any time that I spent reading the problem statement, deciding on my approach, and coding in this time as well.

Stop and reflect: in considering the problem, what approach did you take? Did you do any writing / drawing / sketching, or did you begin by coding?

I loosely read the problem statement given to me and chose the language I would try to implement my solution in which was python since dealing with 2 dimensional arrays is a bit easier in that language from the few languages that I know. I did not write out the problem statement or any of my directions on solving the problem nor did I begin to sketch out a figure to assist with the solving of the problem. After the 15 minutes was over, I was left with a partial solution that did not fully solve the problem nor did I know if I was on the right path towards the optimal solution.

Before proceeding to solve the problem, write / draw / sketch out your approach. Include a photo of any sketches in a Word document where you place these notes.

Understanding the problem fully

I paid more attention to the following underlined components. "*The goal of this puzzle is to guess the coordinate of a bomb (line and column of a 2-dimensional array). You will have to make a guess at each step of the puzzle and adjust it from given feedback. Of course, you have a limited number of guesses*.".

Previously I had missed further hints at a solution given in the learning opportunities section for the problem which states that Binary Search and Intervals could be used.

Rules

Before each jump, the heat-signature device will provide you with the direction of the bombs based on your current position:

- *U (Up)*
- *UR (Up-Right)*
- *R (Right)*
- *DR (Down-Right)*
- *D (Down)*
- *DL (Down-Left)*
- *L (Left)*
- *UL (Up-Left)*

*Your mission is to program the device so that **it indicates the location of the next window you should jump to** in order to reach the bombs' room **as soon as possible**.*

Based on these rules I know that I will have to store the direction in a variable and consider which coordinates to move to next. It is important that I do not waste any moves so that I can optimize the time it takes.

*Buildings are represented as a rectangular array of windows, the window in the **top left corner of the building is at index (0,0)***

This means that I figure out what the new coordinates are for each direction. But it is simpler than this because the direction given is a string of chars and i can simply find D, U , L , R and the rest are just combinations of this. Simply offset to exclude current position

U (Up): ymax = y0 - 1

L (Left): xmax = x0 - 1

R (Right): xmin = x0 + 1

D (Down): ymin = y0 + 1

Since we will just need find the middle of X and Y coordinate for any direction given, we can use the fact that we know the current location to do this. In the case that the direction is not a combination we must have default values for the greatest boundary possible.

xmin: The minimum x-coordinate (left boundary) of the search area. Any potential bomb location must have an x-coordinate greater than or equal to xmin.

xmax: The maximum x-coordinate (right boundary) of the search area. Any potential bomb location must have an x-coordinate less than or equal to xmax.

ymin: The minimum y-coordinate (bottom boundary) of the search area. Any potential bomb location must have a y-coordinate greater than or equal to ymin.

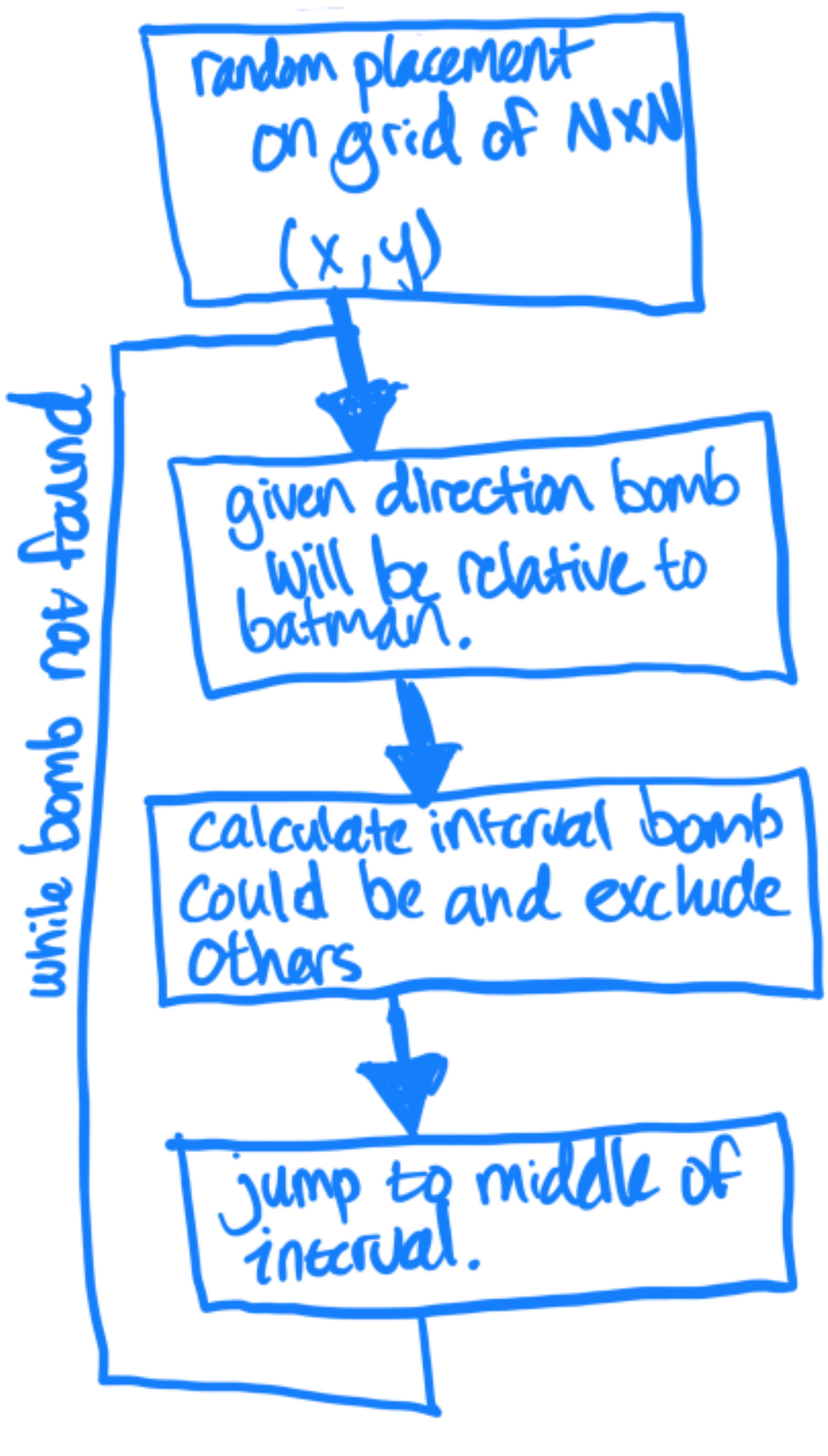
ymax: The maximum y-coordinate (top boundary) of the search area. Any potential bomb location must have a y-coordinate less than or equal to ymax.

I should keep track of which coordinates the bomb could still be in. I also want to make sure that each time I make a direction is given, that im excluding everything not in that area based on where batman is. So for instance if I start at 2,5 and the direction is UR every coordinate not greater than 2,5 should not be consider again. Once I exclude those intervals and store them in an updated list, I then need to use binary search algorithm to place batman in one of those coordinates in which the bomb could be in in the interval. Since I do not know where the bomb is we cannot check against the key, but we can still move batman to the middle of the searchable space (interval) each time like in binary search. If the bomb is found, the program will end. I can quickly get the bounds of the interval from the method provided by the interval library.

I may need to check that I will not advance to a coordinate outside of the range I mentioned previously.

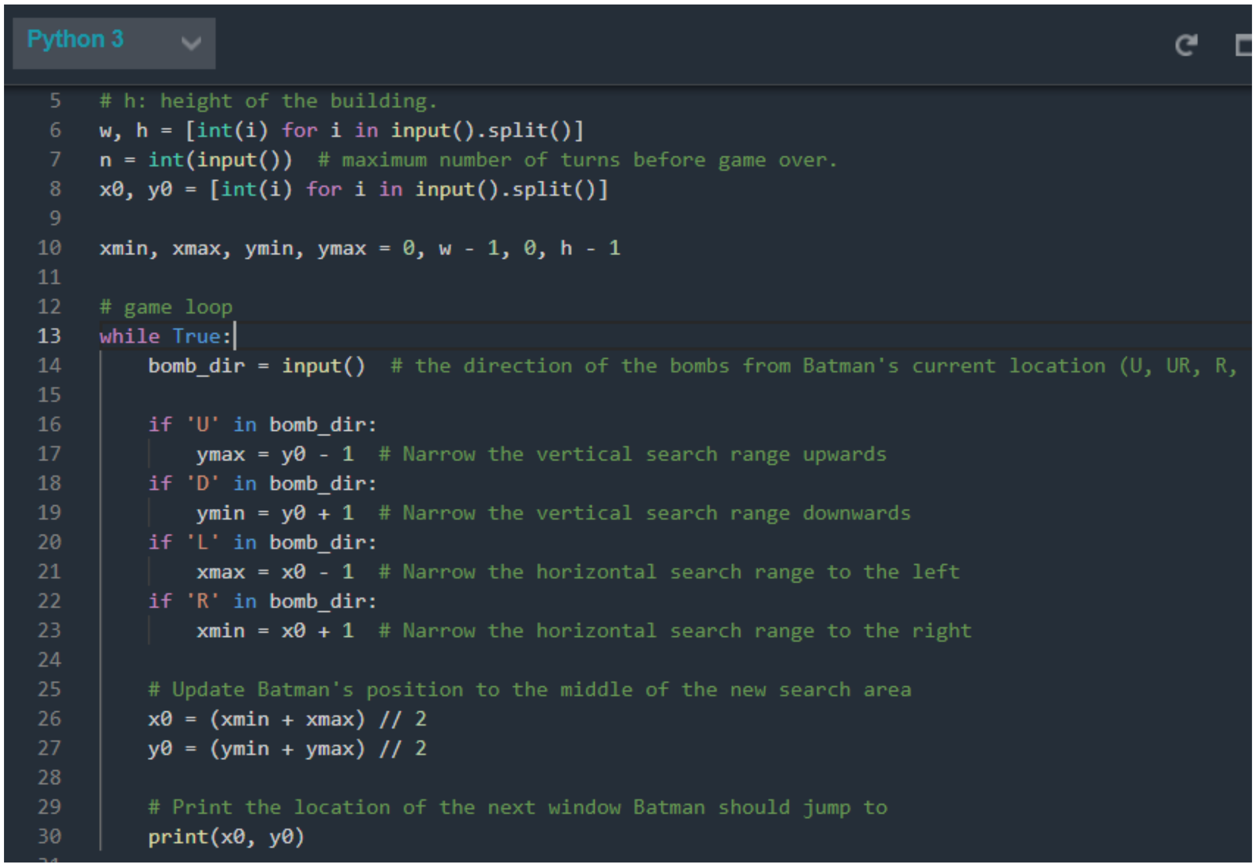
Creating a diagram to guide myself towards a solution

Now that I had fully written and understood the problem, I wanted to create a diagram to guide my development towards a final solution. Below is a high level diagram of events.


random placement on grid of NxN (x,y)
given direction bomb will be relative to batman.
calculate interval bomb could be and exclude others
jump to middle of interval.
while bomb not found

Solve the problem.

The worst case time complexity for searching the 2d array using binary search is O(m+n) where m is the number of rows and n is the number of columns respectively.



```python
# h: height of the building.
w, h = [int(i) for i in input().split()]
n = int(input())  # maximum number of turns before game over.
x0, y0 = [int(i) for i in input().split()]

xmin, xmax, ymin, ymax = 0, w - 1, 0, h - 1

# game loop
while True:
    bomb_dir = input()  # the direction of the bombs from Batman's current location (U, UR, R,

    if 'U' in bomb_dir:
        ymax = y0 - 1  # Narrow the vertical search range upwards
    if 'D' in bomb_dir:
        ymin = y0 + 1  # Narrow the vertical search range downwards
    if 'L' in bomb_dir:
        xmax = x0 - 1  # Narrow the horizontal search range to the left
    if 'R' in bomb_dir:
        xmin = x0 + 1  # Narrow the horizontal search range to the right

    # Update Batman's position to the middle of the new search area
    x0 = (xmin + xmax) // 2
    y0 = (ymin + ymax) // 2

    # Print the location of the next window Batman should jump to
    print(x0, y0)
```

Having an example 2d grid was helpful in trying different solutions for each direction and visualizing what interval should be produced.This was key to my understanding of the problem. I did not need to sort, because the grid was already in order.

```
#   [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0)],

#   [(0, 1), (1, 1), (2, 1), (3, 1), (4, 1)],

#   [(0, 2), (1, 2), (2, 2), (3, 2), (4, 2)],

#   [(0, 3), (1, 3), (2, 3), (3, 3), (4, 3)],

#   [(0, 4), (1, 4), (2, 4), (3, 4), (4, 4)],
```

Write 1-2 paragraphs describing how you solved this problem, before the reflection in step 3 and after. Did this affect your problem-solving abilities? Why or why not?

Initially, I began coding the solution by implementing a straightforward approach to narrow down the search space based on the direction of the bomb. Without fully understanding the problem's requirements, I relied on trial and error to adjust the boundaries of the search area. This led to inefficient calculations and an unclear methodology for updating Batman's position. The lack of a structured plan resulted in redundant operations and a more complex approach than necessary.

After drawing a diagram to visualize the search space and its boundaries, I gained a clearer understanding of how the directional feedback affected the search area. This visual representation allowed me to systematically adjust the boundaries and recalculate the center of the search area more effectively. By refining the approach based on this diagram, I implemented a more efficient binary search strategy that reduced unnecessary computations and improved the accuracy of finding the bomb.

This process highlighted the importance of visualizing problems and structuring solutions to enhance problem-solving efficiency. Not only did it take me less time to solve after i had drawn a high level diagram and an example grid (this was key) I was able to find an elegant solution that was not very complex to understand once implemented. For example, before i though i would have to store the search interval each time, but i did not do this i simply used that fact that it is a loop and max and mins would simply persist until the bomb was found anyways.