Marlene Azevedo

CS-499-Q4508 Computer Science Capstone



Milestone 3: Enhancement Two

Algorithms & Data Structures

**A. Briefly describe the artifact. What is it? When was it created?**

For this artifact, we had to integrate some type of data structures into the code. Since the code is mainly built using Binary Trees, then I had to implement a *Heap Table* and a *Search Algorithm* which seemed to be the best data structures to add. I mainly focused on the Board Coordinates (Line 43). There is an array there listed with the letters from A to H and text point coordinates. Since there is an array in there already, I was able to integrate the Heap table and Search algorithm. It took a while but it worked well with no errors.

**B. Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in algorithms and data structure? How was the artifact improved?**

I selected the Heap Table to prioritize and help rearrange the letters better in the queue. I mixed up the letters on purpose to see if the system can recognize the heap and push the last two letters in the end of the array. Additionally, the ‘*heapq*’ module did not print the last 2 letters in the array. The push() function permitted these letters to be included in part of the board coordinates.   
  
 I selected the Search Algorithm to help identify the grid letters (Line 43) already input in the code. This allows to search all data and confirm if a particular character was included in the array. When the character ‘A’ is searched for, the result should come out as a ‘True’ value. However, when the algorithm searches for the letter ‘H’ then it should return as a ‘False’ value because that was never inlcuded to the original grid letter array. This boolean output confirmed that the search algorithm worked well in the code.

**C. Did you meet the course objectives you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?**

The original plan I had did not fully meet the enhancement requirements. I followed the feedback given in the graded ePortfolio essay. I first focused on creating a *Hash Table* and implemented it somewhere in the code but that did not seem to work well. After receiving additional feedback from the instructor, I noticed that the Heap Table and Search Algorithm worked best. It took a few hours of coding until I got it to run with no errors.

**D. Reflect on the process of enhancing and/or modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?**

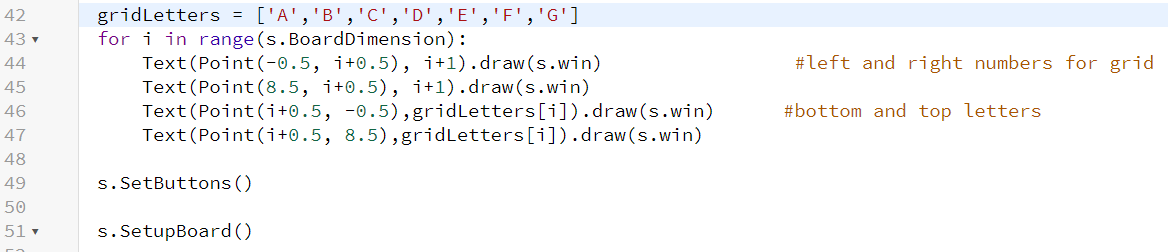
I have taken a data structures class before and learned about building binary trees and algorithms. Except, I never learned how to put it together in a code. This was a bit of challenge for me since I did not know how to incorporate that into the code. After reviewing different websites and reading up on it, I learned how it fits together with the code. It makes sense to have some type of data structure in order to be able to locate some type of data faster than having a regular array. I believe I was able to implement the Heap and Search Algorithm well in the beginning of the checkers code. The code would have worked okay without this extra data structure part except I feel like it makes searching for those arrays better and quicker.

**MODIFICATIONS:**

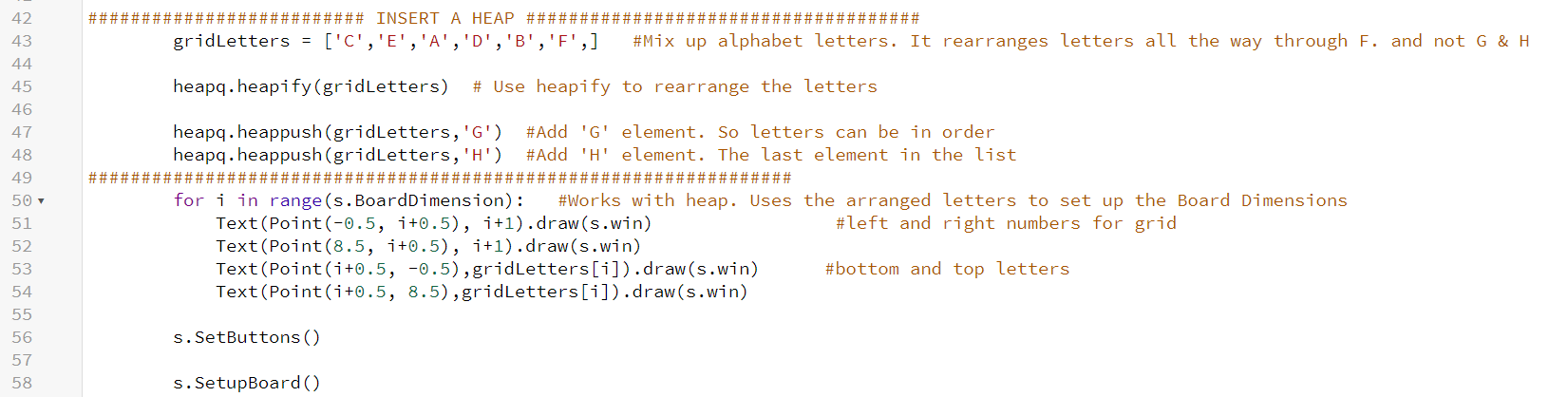
**HASH TABLE**

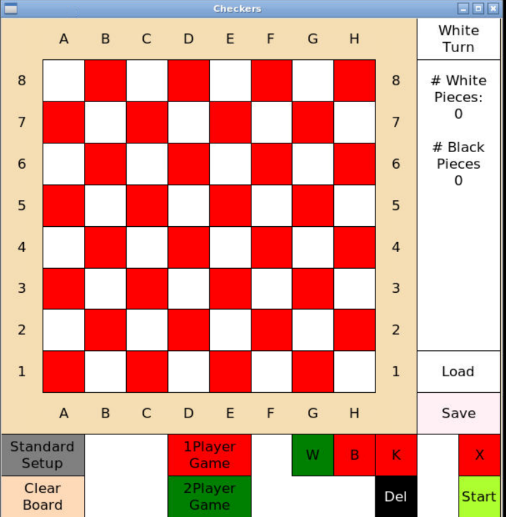
Before:

The letters are arranged in alphabetical order. This is necessary to set up the board coordinates.



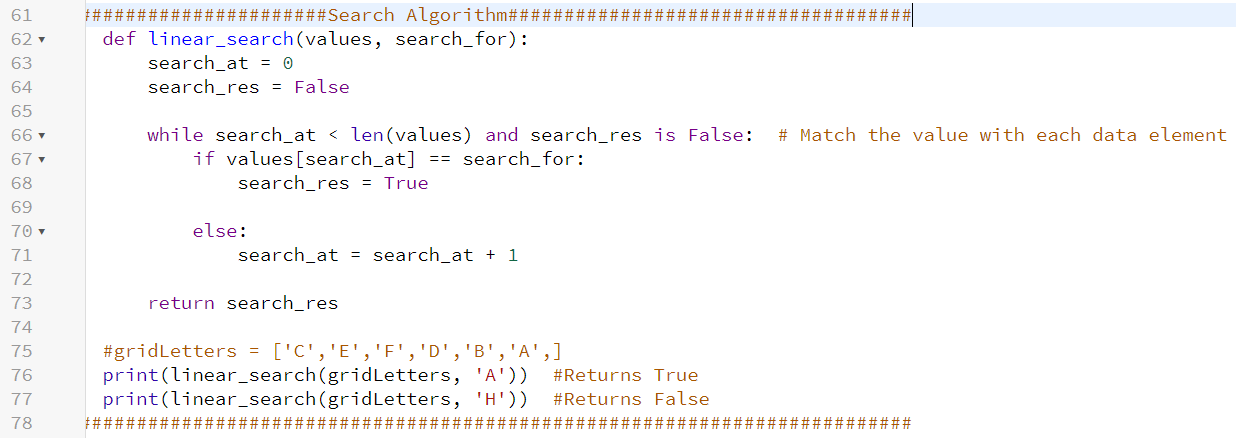
After:

When inserting the hash tables, if the grid Letters where not in orders, this helps put them in alphabetical order.

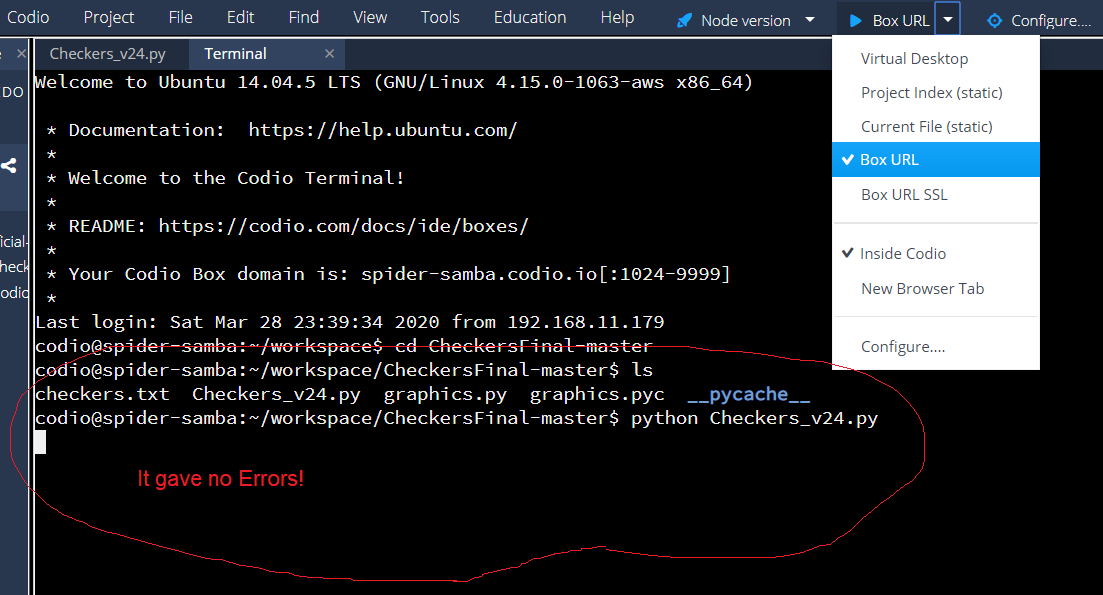


 The letters are *arranged* in order as shown here.

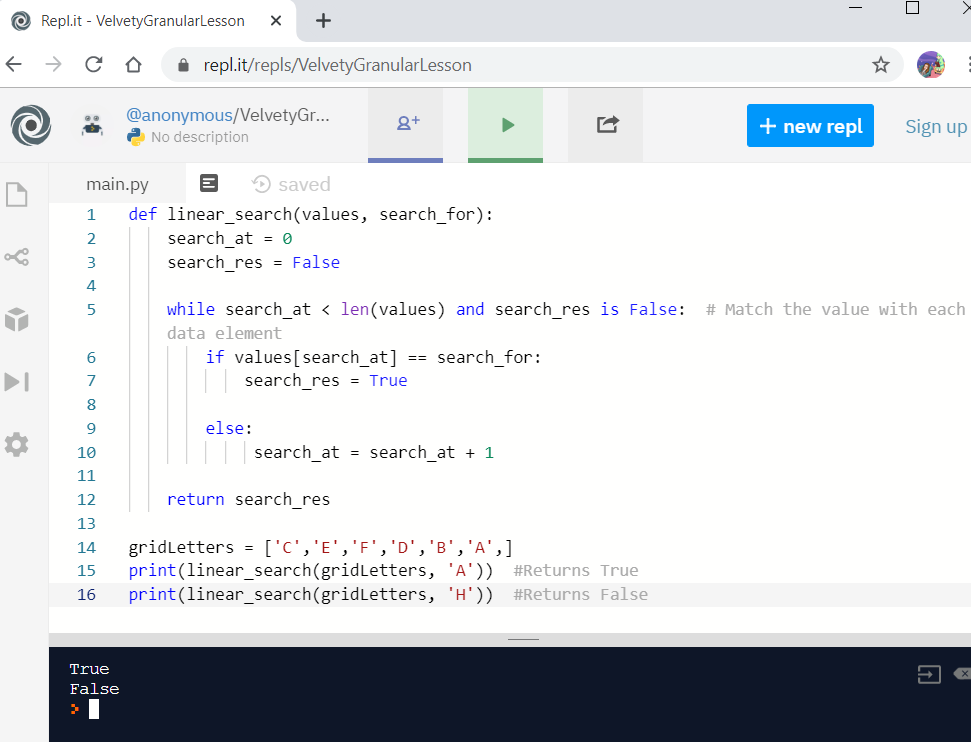
**SEARCH ALGORITHM**

For this part, I inserted a Search Algorithm to work with the gridLetters coordinates (Line 43).

When I ran the code in the Terminal, it gave me no errors.



I even tested the code in [repl.it](https://repl.it/repls/StaleRemorsefulRecursion) to make sure it ran well. It gave me the expected ‘True’ and ‘False’ output.



|  |
| --- |
| 1. def linear\_search(values, search\_for):  2. search\_at = 0  3. search\_res = False  4.  5. while search\_at < len(values) and search\_res is False:  6. if values[search\_at] == search\_for:  7. search\_res = True  8.  9. else:  10. search\_at = search\_at + 1  11.  12. return search\_res  13.  14. #gridLetters = ['C','E','F','D','B','A',]  15. print(linear\_search(gridLetters, 'A')) #Returns True  16. print(linear\_search(gridLetters, 'H')) #Returns False |

The Code: