Weekly task log

**Week 1 – Stack and Queue**

**Goal:**

A Stack class with the following functions:

\_\_init\_\_()

is\_empty()

push(item)

pop() (raises an IndexError if empty)

A Queue class with the following functions:

\_\_init\_\_()

is\_empty()

enqueue(item)

dequeue() (raises an IndexError if empty)

**Problems I encountered:**

Accidentally nested is\_empty function inside \_\_init\_\_() function of the stack class causing an attribute error when calling it.

Missed an indentation issue atg first but solved it after too long of missing it and downloaded indent-Rainbow plugin in VSC to help me catch indentation issues easier.

**Week 2 – Linked Queues and Circular Queues**

**Goal:**

Node class for building linked structures

LinkedQueue class with:

\_\_init\_\_(), enqueue(), dequeue(), and is\_empty() functions

A CircularQueue class with:

Having a fixed-size array implementation

Enqueue() and dequeue() with indexing wraparound using modulo operator

Is\_full() and is\_empty() status checks

**Problems I encountered:**

I keep nesting method’s wrong and nested is\_empty inside \_\_init\_\_ by accident causing method access errors.

Made multiple typo errors when trying to index list self.queue(self.rear) by using brackets () instead of square brackets [] accidentally calling functions when I was trying to access list elements.

Got caught up by another typo when I tried to multiply None \* capacity rather than doing [None] \* capacity which subsequently caused a TypeError.

Accidentally left in old left over test code from earlier in dequeue(“OverFlow test”) which led to me calling it with an argument when it didn’t need it.

**Week 3 – Priority Queue, Sets & Graphs and Module Debugging**

**Goal:**

PriorityQueue class with:

\_\_init\_\_() to sort initial data

add(item) to insert new tuples and re-sorting

get() to remove the item with highest priority

peek() to look at highest-priority item without removing it

Rebuild main.py test script to check functionality with multiple test cases

Implemented BFS (Breadth-First Search) for traversing a graph represented as a dictionary

**Problems I encountered:**

I initially edited the correct code logic in a renamed file to differentiate my work from the starter pack but like a dunce, I left the original PriorityQueue in the same folder, causing Python to import the wrong one leading to nonsensical logic on output.

Returned full tuples like (“Northern”, 4) when testing only required the item name leading to the code outputting “False” across the board.

Had to debug the Python import behaviour and change the default module provided.

**Week 4 - Dijkstra’s Algorithm & Graph Pathfinding**

**Goal:**

Walk() function that:

Initialised distance and previous node trackers

Used a min-heap priority queue via heapq

Traverses the graph while updating shortest known distances

Reconstructes and returns the shortest path from start to end

Edge weights stored in dictionary

Testing suite in main.py that validate path correctness for two different graphs

**Problems I encountered:**

This project was a doozy and it ended with me crashing out. I accidentally redefined the edges input within \_\_init\_\_, which broke graph structure.

Nested walk() inside a “for” loop within \_\_init\_\_, which made it inaccessible to the object.

Multiple indentation errors led to syntax errors (Python indentation was eating me alive)

Structurally broke functions and control flow without even realising because of improper indentation levels

I personally crashed out in real-time after hours of debugging and getting nowhere before finally locking in and realising what’s going wrong.

This was genuinely the hardest task yet.

**Week 5 – Vectors, Matrices and Transformations**

**Goal:**

Created and manipulated 2D vectors using NumPy

Performed scalar multiplication and vector addition

Calculated dock products and magnitude (lengths) of vectors

Created transformation matrices to rotate and scale vectors

Explored how these mathematical tools map directly to real-world applications in --

Game development (e.g movement, rotation, hitboxes)

AI (e.g field of view cones, directionality)

3D software like 3ds Max and Unreal Engine

Problems I encountered:

Had issues in VSC with it using the wrong interpreter that lacked NumPy.

Had a brainfart moment and tried to run a .Py directly inside the terminal after installing NumPy and that had me tripped up for an embarrassing 10mins or so.

Wrapping my head around the maths wasn’t second nature easy because I was never really very amazing at maths.

**Week 7 – Matrix and Linear Transformation**

**Goal:**

Completing the code on the 2D ship game provided.

Completed the rotate() function in utils.py using the 2D rotation matrix.

Updated ship’s update() method to calculate it’s direction vectors from ship to mouse.

Normalise and scale it based on distance, rotate the ship to face mouse cursor using atan2 and moving the ship at a variable speed.

Also modified asterioid logic so each asterioid gets a random direction and speed on spawn, and continues to move. Added screen wrap for asterioids that float off-screen to bring them back.

**Problems I encountered:**

The Ship was initially facing away from the mouse, I fixed by rotating 180 degrees using + math.pi

Got cooked by Python nesting again

Later on, I added a restart button to the game that would trigger after the failure state triggered and that was kinda difficult. Implementing this seemingly simple function had me resetting global objects (ship, asteroids) inside the event-handling logic. The program would crash during runtime because of type mismatches between int64 and float64 NumPy arrays after resetting. I also had major issues with Python’s random() function due to coflictiong imports at the top of the code. Also had to learn and use dtype=float across the position arrays once I debugged the Python imports.