AOC 2023\Day 03\Solve_2023_03.py

```
253 class SchematicPartB(list):
254     """the grid"""
255 
256     # all printable non-letters, non-numbers and removed '.' (46)
257     specials_chars = ["*"]
258 
259     cardinal = {
260         "up": (-1, 0),
261         "upright": (-1, +1),
262         "right": (0, +1),
263         "downright": (1, +1),
264         "down": (1, 0),
265         "downleft": (1, -1),
266         "left": (0, -1),
267         "upleft": (-1, -1),
268     }
269 
270     def __init__(self, data_list):
271         """where shit gets started"""
272 
273         # PART B ##################################################
274         # * loc key, val = [adjacent ships]
275         self.stars_found = {}
276         # PART B ##################################################
277 
278         for row in data_list:
279             this_row = row.strip("\n")
280             self.max_wide = len(this_row)
281             self.append(this_row)  # .split()
282         self.max_high = len(self)
283 
284         self.legal_locations = set()
285         for y in range(self.max_high):
286             for x in range(self.max_wide):
287                 self.legal_locations.add((y, x))
288 
289         self.coloured = {
290             "red": set(),
291             "green": set(),
292             "blue": set(),
293         }  # 000:[(x,y),(x,y),(x,y)]
294         _ = self.get_symbols()
295         _ = self.wet_targets()
296         _ = self.battleship()
297 
298     def at_loc(self, row, col):
299         """at_loc(y, x) -> "3"
300         return value at grid reference given"""
301         return self[row][col]
302 
303     def get_symbols(self):
304         """returns locs for all special symbols
305         found in the grid. Special characters
```

```
306         are printable, non-digit, non-letter and
307         not a period "." chr(46)
308         """
309         symbols = []
310         for y, row in enumerate(self):
311             for x, _ in enumerate(row):
312                 target_loc = (y, x)
313                 symbol_found = self.at_loc(*target_loc)
314                 if symbol_found in self.specials_chars:
315                     symbols.append(target_loc)
316                     self.stars_found[target_loc] = {
317                         "splash": [],
318                         "wet_targets": [],
319                         "ships": [],
320                     }
321 
322         self.coloured["red"] = set(symbols)
323         return symbols
324 
325     def adj_locs(self, row, col):
326         """adj_locs(y, x) -> [(y-1,x),(y+1,x),...]
327         return list of the 8 tupled grid references
328         around the grid reference given."""
329 
330         surround = []
331         for r, c in self.cardinal.values():
332             new_loc = (row + r, col + c)
333             if new_loc in self.legal_locations:
334                 surround.append(new_loc)
335         return surround
336 
337     def splash_zone(self):
338         """splash_zone(self) -> [(y,x),(y,x),...]
339         returns locations within range of
340         the Points of Impact (PoI's).
341         The ENTIRE AREA EFFECTED!
342         """
343         area_of_effect = []  # AoE
344         for s, zone in self.stars_found.items():
345             area_of_effect.extend(self.adj_locs(*s))
346             zone["splash"].extend(area_of_effect)
347 
348         return area_of_effect
349 
350     def wet_targets(self):
351         """wet_targets(self) -> [(y,x),(y,x),...]
352         Returns locs from within the AoE that are
353         digits.
354         """
355         wt = {}  # targets within AoE
356 
357         # do this for each * individually this time
358         for zone in self.stars_found.values():
359             for loc in zone["splash"]:
360                 found = self.at_loc(*loc)
361                 if found.isdigit():
```

```
                    wt[loc] = found
                    zone["wet_targets"].append(loc)
            self.coloured["blue"] = set(wt.keys())
        return wt

    def battleship(self):
        """self.battleship() -> [int('467'), ...]
        goes through all the wet_targets locs and
        tries to lefty/righty find the entire
        integer found there.
        Much like the game Battleship.
        """
        # For Part B: if the PoI was a * then also dump the found
        # ships in the self.stars_found dict for that *.
        # so, since we didn't pass the PoI to battleship(),
        # we need to backwards locate the PoI to be sure.

        for star_loc, body in self.stars_found.items():
            sorted_hits = set(sorted(body["wet_targets"]))
            checked_locs = []
            ships_found = []

            for wt in sorted_hits:
                if wt not in checked_locs:
                    # flag this loc as already checked
                    checked_locs.append(wt)

                    # start with the character we hit
                    this_ship = self.at_loc(*wt)

                    # go left (x-n)
                    row, col = wt

                    # for visualizer
                    self.coloured["green"].add(wt)

                    PoI = col
                    # reverse range indexes suck donkey
                    # for x in range(col - 1, -1, -1):
                    while True:
                        PoI -= 1

                        if (row, PoI) not in self.legal_locations:
                            break  # we've hit the start

                        search_loc = (row, PoI)
                        content_left = self.at_loc(*search_loc)

                        not_water = content_left != "."
                        is_digit = content_left.isdigit()
                        not_checked = search_loc not in checked_locs

                        if all([not_water, is_digit, not_checked]):
                            self.coloured["green"].add(search_loc)
                            checked_locs.append(search_loc)
                            this_ship = content_left + this_ship
```

```
418                             continue
419
420                         break  # stop looking, we don't need anymore.
421
422                     # go right (x+1)
423                     PoI = col
424                     # for x in range(col + 1, self.max_wide):
425                     while True:
426                         PoI += 1
427
428                         if (row, PoI) not in self.legal_locations:
429                             break  # we've hit the end
430
431                         search_loc = (row, PoI)
432                         content_right = self.at_loc(*search_loc)
433
434                         not_checked = search_loc not in checked_locs
435                         is_digit = content_right.isdigit()
436                         not_water = content_right != "."
437
438                         if all([not_water, is_digit, not_checked]):
439                             self.coloured["green"].add(search_loc)
440                             checked_locs.append(search_loc)
441                             this_ship += content_right
442                             continue
443
444                         break  # stop looking, we don't need anymore.
445
446                     ships_found.append(int(this_ship))
447
448                     # PART B ################################################
449
450                     self.stars_found[star_loc]["ships"].append(int(this_ship))
451                     # PART B ################################################
452
453             for loc in checked_locs:
454                 self.stars_found[star_loc]["wet_targets"].pop(loc)
455         return ships_found
456
457     @property
458     def final_answer_a(self):
459         """Part A Answer"""
460         return sum(self.battleship())
461
462     @property
463     def final_answer_b(self):
464         """Part B Answer"""
465         legal_finds = []
466         for ships in self.stars_found.values():
467             # find ONLY all * chars with 2 adjacent numbers
468             if len(ships) == 2:
469                 # multiply each pair together
470                 legal_finds.append(math.prod(ships))
471
472         # add those products together
473         return sum(legal_finds)
```

```
474
475     def show(self):
476         """Prints the grid to screen"""
477         os.system("cls")
478         # print("Reds:", self.coloured["red"])
479         # print("Greens:", self.coloured["green"])
480         # print("Blues:", self.coloured["blue"])
481         # _ = input("Press <ENTER> to continue:".center(80))
482
483         for y, row in enumerate(self):
484             row_content = str(y).rjust(5, "0") + " - "
485             for x, _ in enumerate(row):
486                 # if self.at_loc(y, x):
487
488                 if (y, x) in self.coloured["red"]:  # get_symbols():
489                     row_content += colored(self.at_loc(y, x), "red")
490
491                 elif (y, x) in self.coloured["green"]:  # .battleship():
492                     row_content += colored(self.at_loc(y, x), "green")
493
494                 elif (y, x) in self.coloured["blue"]:  # .wet_targets():
495                     row_content += colored(self.at_loc(y, x), "blue")
496
497                 else:
498                     row_content += self.at_loc(y, x)
499
500             print(row_content)
501         print("\n\n")  # end of page
502         # _ = input("Press <ENTER> to continue:".center(80))
```