

```
1:#!/usr/bin/env python
2: #-*- coding: utf-8 -*-
3:
4:from .lib.elements import Element, Base, Satellite, Player, Trap, Bomb, Fruit, Lint
ern
5:from .lib.elements import Mountain, Mine, Cannon
6:from .lib.elements import Spawner, Enemy
7:
8:from dataclasses import dataclass, field
9:from playsound import playsound
10:from itertools import chain
11:from pathlib import Path
12:
13:import threading
14:import random
15:import curses
16:import time
17:import math
18:import sys
19:import os
20:
21:FPS = 50
22:
23:
24:@dataclass
25:class Game:
26:    screen = None
27:
28:    @classmethod
29:    def create(cls):
30:        game = cls()
31:        return game
32:
33:    def init(self, screen):
34:
35:        self.screen = screen
36:
37:        # Curses Settings
38:        curses.curs_set(False) # Do not display blinking cursor
39:        curses.noecho()
40:        curses.cbreak()
41:        curses.start_color()
42:
43:        # Curses Color Pairs
44:        curses.init_color(curses.COLOR_BLACK, 0, 100, 100)
45:        curses.init_pair(1, 250, 0) # Default Color
46:        curses.init_pair(2, 137, 236)
47:
48:        curses.init_pair(3, curses.COLOR_MAGENTA, 0) # FRUIT
49:        curses.init_pair(4, curses.COLOR_MAGENTA, 243) # FRUIT
50:
51:        curses.init_pair(5, curses.COLOR_YELLOW, 0) # ENEMIES
52:        curses.init_pair(6, curses.COLOR_YELLOW, 243) # ENEMIES
53:
54:        curses.init_pair(7, curses.COLOR_GREEN, 0) # BASE
55:        curses.init_pair(8, curses.COLOR_GREEN, 243) # BASE
56:
57:        curses.init_pair(9, curses.COLOR_BLUE, 0) # ENEMY TRAPPED
58:        curses.init_pair(10, curses.COLOR_BLUE, 243) # ENEMY TRAPPED
59:
60:        curses.init_pair(11, curses.COLOR_RED, 0) # MOUNTAIN
61:        curses.init_pair(12, curses.COLOR_RED, 243) # MOUNTAIN
62:
63:        curses.init_pair(13, 25, 231) # PLAYER
64:
65:
66:        curses.init_pair(15, 199, 0) # ENEMY TRAPPED
67:        curses.init_pair(16, 199, 243) # ENEMY TRAPPED
68:
69:        curses.init_pair(17, 225, 0) # LINTERN
70:        curses.init_pair(18, 225, 243) # LINTERN
71:
72:        # Screen Settings
73:        self.screen.keypad(True)
74:        self.screen.nodelay(True)
75:        self.screen.border(0)
76:
77:        self.min_y, self.min_x = (1, 1)
78:        self.max_y, self.max_x = tuple(
79:            i - j for i, j in zip(self.screen.getmaxyx(), (5, 2))
80:        )
81:
82:        self.screen_limits = (self.min_y, self.max_y, self.min_x, self.max_x)
83:        self.screen_size = (self.max_x - self.min_x) * (self.max_y - self.min_y)
84:
85:        # Draw Window Borders
86:        self.screen.addch(self.max_y + 1, 0, curses.ACS_SSSB)
87:        self.screen.addch(self.max_y + 1, self.max_x + 1, curses.ACS_SBSS)
88:
89:        for x in range(1, self.max_x + 1):
90:            self.screen.addch(self.max_y + 1, x, curses.ACS_HLINE)
91:
92:        # Game Elements
93:        self.player = Player(20, 20)
94:        self.trap = Trap(20, 20)
95:        self.base = Base(self.max_y // 2, self.max_x // 2, deployed=False)
96:
97:        self.mountains = [
98:            Mountain(y, x)
99:            for y, x in [
100:                (
101:                    random.randint(self.min_y, self.max_y),
102:                    random.randint(self.min_x, self.max_x),
103:                )
104:                for i in range(10)
105:            ]
106:        ]
107:
108:        self.spawners = [
109:            Spawner(y, x)
110:            for y, x in [
111:                (
112:                    random.randint(self.min_y, self.max_y),
113:                    random.randint(self.min_x, self.max_x),
114:                )
115:                for i in range(self.screen_size // 400)
116:            ]
117:        ]
118:
119:        self.satellites = []
120:        self.mines = []
121:        self.cannons = []
122:        self.linterns = []
123:        self.enemies = []
124:        self.fruits = []
125:        self.bombs_topick = []
126:        self.bombs_activated = []
127:
```

```
128:         self.area = set(
129:             (y, x)
130:             for y in range(self.min_y, self.max_y + 1)
131:             for x in range(self.min_x, self.max_x + 1)
132:         )
133:         self.area_fog = set()
134:
135:         self.loop()
136:
137:     def loop(self):
138:
139:         clock = time.time()
140:         while True:
141:
142:             # 1. Process Buildings (Mine -> Dig, Cannon -> Shoot...)
143:             # , unless they are destroyed by an enemy
144:
145:             self.buildings = list(chain(self.mines, self.cannons, self.satellites))
146:             for building in self.buildings:
147:                 if building.health <= 0:
148:                     self.buildings.remove(building)
149:                     self.clear(building)
150:
151:                 if building.kind == "Mine":
152:                     self.mines.remove(building)
153:
154:                 elif building.kind == "Cannon":
155:                     self.cannons.remove(building)
156:
157:                 elif building.kind == "Satelite":
158:                     # When a satellite is destroyed, all dependent buildings co
lapses next turn.
159:                     self.satellites.remove(building)
160:                     for building_dep in nearby_elements(
161:                         building, chain(self.mines, self.cannons), 10
162:                     ):
163:                         building_dep.health = 0
164:
165:                 else:
166:                     if building.kind == "Mine" and building.dig_success():
167:                         self.base.gold += building.dig_value
168:
169:                     elif building.kind == "Cannon" and building.shot_success():
170:                         target = nearby_elements(
171:                             building,
172:                             self.enemies,
173:                             d=building.production_rate,
174:                             ret="choice",
175:                         )
176:
177:                         if target is not None and target in self.enemies:
178:                             self.enemies.remove(target)
179:                             self.clear(target)
180:                             self.player.points += 1
181:                             building.kills += 1
182:
183:             # 2. Spawn Enemies
184:             if random.randint(0, 1000) < 10:
185:                 s = random.choice(self.spawnners)
186:                 self.enemies.append(s.spawn())
187:
188:             # 3. Enemies Actions
189:             if time.time() > clock + max(0.2, 1 - self.player.level / 12):
190:                 for enemy in self.enemies:
191:
192:                     # a. scan targets
193:                     targets = [
194:                         {"target": target, "d": enemy.distance(target)}
195:                         for target in chain(
196:                             self.buildings,
197:                             self.satellites,
198:                             [
199:                                 self.base,
200:                                 self.player,
201:                             ],
202:                         )
203:                     ]
204:
205:                     # b. Choose the nearest target and moves towards it
206:                     # TODO: Set weight to target kinds
207:                     if len(targets) > 0:
208:                         target = sorted(targets, key=lambda x: x["d"])[0]["target"]
209:
210:                         if target.x - enemy.x > 0:
211:                             delta_x = 1
212:                         elif target.x - enemy.x < 0:
213:                             delta_x = -1
214:
215:                         if target.y - enemy.y > 0:
216:                             delta_y = 1
217:                         elif target.y - enemy.y < 0:
218:                             delta_y = -1
219:
220:                     # if no targets, move randomly
221:                     else:
222:                         delta_y = random.randint(-1, 1)
223:                         delta_x = random.randint(-1, 1)
224:
225:                     if (enemy.y, enemy.x) in self.area_light:
226:                         self.clear(enemy)
227:
228:                     enemy.move(
229:                         max(1, min(self.max_y, enemy.y + delta_y)),
230:                         max(1, min(self.max_x, enemy.x + delta_x)),
231:                     )
232:
233:                     # c. check collisions with player, buildings, base
234:                     if collision(self.player, enemy):
235:                         combat_result = random.randint(0, 99)
236:                         if combat_result < 80 and enemy in self.enemies:
237:                             play_sound("pos")
238:                             self.enemies.remove(enemy)
239:                             self.player.points += 1
240:                             self.player.health -= random.randint(0, 2)
241:
242:                         else:
243:                             play_sound("scream_fight")
244:                             self.player.health -= random.randint(5, 10)
245:
246:                     for building in self.buildings:
247:                         if collision(enemy, building):
248:                             building.health -= 1
249:
250:                     for b in chain(
251:                         self.satellites,
252:                         [
253:                             self.base,
```

ctower/main.py

```
254:         l,
255:     ):
256:         if collision(b, enemy) and enemy in self.enemies:
257:             self.enemies.remove(enemy)
258:             self.player.points += 1
259:             b.health -= random.randint(0, 2)
260:
261:         if self.trap.deployed:
262:             if distance(self.trap, enemy) <= 5 and enemy in self.enemi
es:
263:                 self.enemies.remove(enemy)
264:                 enemy.color = 9
265:                 self.render(enemy)
266:
267:         clock = time.time()
268:
269:         delta_x = delta_y = 0
270:
271:         # 4. Monitor Activated Bombs
272:         if len(self.bombs_activated) > 0:
273:             for bomb in self.bombs_activated:
274:
275:                 for (y, x) in bomb.area:
276:                     if (y > 0 and y < self.max_y) and (x > 0 and x < self.max_
x):
277:                         self.screen.addstr(y, x, "~", curses.color_pair(4))
278:
279:             if bomb.is_kaboom:
280:                 play_sound("kaboom")
281:
282:             victims = nearby_elements(
283:                 bomb,
284:                 chain(
285:                     self.enemies,
286:                     self.spawnners,
287:                     [
288:                         self.player,
289:                     ],
290:                 ),
291:                 d=bomb.strength,
292:             )
293:
294:             if victims is not None:
295:                 for victim in victims:
296:                     if victim.kind == "Player":
297:                         play_sound("scream-bomb")
298:                         self.player.health -= 50
299:
300:                     else:
301:                         victim.health -= 5
302:
303:             for (y, x) in bomb.area:
304:                 if (y > 0 and y < self.max_y) and (
305:                     x > 0 and x < self.max_x - 1
306:                 ):
307:                     self.clear(y, x)
308:
309:             self.bombs_activated.remove(bomb)
310:             self.clear(bomb)
311:
312:         for enemy in chain(self.enemies, self.spawnners):
313:             if enemy.health < 0 and enemy in chain(self.enemies, self.spawnners
):
314:                 if enemy.kind == "Zombie":
315:                     self.enemies.remove(enemy)
316:                 elif enemy.kind == "Spawner":
317:                     self.spawnners.remove(enemy)
318:
319:                 self.clear(enemy)
320:                 self.player.points += enemy.level
321:
322:         ## Recover Trap
323:         if self.trap.deployed and distance(self.trap, self.player) == 0:
324:             self.trap.deployed = False
325:
326:         ## Fruit Spawner
327:         if random.randint(0, 1000) < 2:
328:             self.fruits.append(
329:                 Fruit(
330:                     random.randint(self.min_y, self.max_y),
331:                     random.randint(self.min_x, self.max_x),
332:                 )
333:             )
334:
335:         ## Bombs Spawner
336:         if random.randint(0, 1000) < 1:
337:             self.bombs_topick.append(
338:                 Bomb(
339:                     random.randint(self.min_y, self.max_y),
340:                     random.randint(self.min_x, self.max_x),
341:                 )
342:             )
343:
344:         ## Fruit check for collision
345:         if len(self.fruits) > 0:
346:             for fruit in self.fruits:
347:                 if collision(self.player, fruit):
348:                     play_sound("bonus")
349:                     self.player.health += 10
350:                     self.fruits.remove(fruit)
351:
352:         if len(self.bombs_topick) > 0:
353:             for bomb in self.bombs_topick:
354:                 if collision(self.player, bomb):
355:                     play_sound("bonus")
356:                     self.player.bombs += 1
357:                     self.bombs_topick.remove(bomb)
358:
359:         # Wait for a keystroke
360:
361:         # key_bindings = {'q': sys.exit,
362:         # 'h': move_left,
363:         # 'j': move_down,
364:         # 'k': move_up,
365:         # 'l': move_right,
366:         #
367:         # 'curses_KEY_LEFT': move_left,
368:         # 'curses_KEY_DOWN': move_down,
369:         # 'curses_KEY_UP': move_up,
370:         # 'curses_KEY_RIGHT': move_right,
371:         #
372:         # 'v': build_base,
373:         # 'm': build_mine,
374:         # 'c': build_cannon,
375:         # 'u': upgrade_building,
376:         # 's': sell_building,
377:         # 'b': deploy_bomb,
378:         # ' ': deplot_trap,
```

```

379:         # }
380:         key = self.screen.getch()
381:
382:         # Process the keystroke
383:         if key is not curses.ERR:
384:             if key == ord("q"):
385:                 break
386:
387:             if key == ord("p"):
388:                 self.pause()
389:
390:             if key in [ord("h"), curses.KEY_LEFT]:
391:                 self.player.to_move = True
392:                 delta_x = -1
393:
394:             if key in [ord("l"), curses.KEY_RIGHT]:
395:                 self.player.to_move = True
396:                 delta_x = 1
397:
398:             if key in [ord("k"), curses.KEY_UP]:
399:                 self.player.to_move = True
400:                 delta_y = -1
401:
402:             if key in [ord("j"), curses.KEY_DOWN]:
403:                 self.player.to_move = True
404:                 delta_y = 1
405:
406:             if key == ord("b"):
407:                 # deploy bomb
408:                 if self.player.bombs > 0:
409:                     self.bombs_activated.append(Bomb(self.player.y, self.player.x))
410:                     self.player.bombs -= 1
411:
412:             if key == ord("m"):
413:                 # build mine, in the distance of 1 of a mine, but not ontop an
414:                 # not possible in an already built mine
415:                 if (
416:                     self.base.deployed
417:                     and nearby_elements(
418:                         self.player,
419:                         chain(
420:                             self.buildings,
421:                             self.mountains,
422:                             [
423:                                 self.base,
424:                             ],
425:                         ),
426:                     )
427:                     is None
428:                     and nearby_elements(
429:                         self.player,
430:                         chain(
431:                             self.satellites,
432:                             [
433:                                 self.base,
434:                             ],
435:                         ),
436:                     d=10,
437:                 )
438:                 is not None
439:                 and min(self.player.distance(mnt) for mnt in self.mountain
s)
440:                     == 1
441:                     and self.base.gold >= 50
442:                 ):
443:                     self.base.gold -= 50
444:                     self.mines.append(Mine(self.player.y, self.player.x))
445:
446:             if key == ord("c"):
447:                 # build cannon
448:                 # not possible in an already built mine
449:                 if (
450:                     self.base.deployed
451:                     and nearby_elements(
452:                         self.player,
453:                         chain(
454:                             self.buildings,
455:                             self.mountains,
456:                             [
457:                                 self.base,
458:                             ],
459:                         ),
460:                     )
461:                     is None
462:                     and self.base.gold >= 50
463:                 ):
464:                     self.base.gold -= 50
465:                     self.cannons.append(Cannon(self.player.y, self.player.x))
466:
467:             if key == ord("u"):
468:                 # upgrade building
469:                 building = nearby_elements(self.player, self.buildings, ret="o
ne")
470:
471:                 if building is not None and building.level < 9:
472:                     cost = building.cost_to_upgrade()
473:                     if self.base.gold >= cost:
474:                         self.base.gold -= cost
475:                         building.upgrade()
476:
477:             if key == ord("s"):
478:                 # sell building
479:                 building = nearby_elements(self.player, self.buildings, ret="o
ne")
480:
481:                 if building is not None:
482:                     self.base.gold += building.cost_to_recover()
483:                     self.buildings.remove(building)
484:                     if building.kind == "Mine":
485:                         self.mines.remove(building)
486:                     elif building.kind == "Cannon":
487:                         self.cannons.remove(building)
488:
489:             if key == ord("v"):
490:                 # deploy base
491:                 if not self.base.deployed:
492:                     self.base.deployed = True
493:                     self.base.y = self.player.y
494:                     self.base.x = self.player.x
495:
496:                 # deploy satellite
497:                 else:
498:                     if (
499:                         nearby_elements(
500:                             self.player,
501:                             chain(
502:                                 self.satellites,

```

```

502:         [
503:             self.base,
504:         ],
505:     ),
506:     d=20,
507: )
508:     is None
509:     and self.base.gold >= 500
510: ):
511:     self.base.gold -= 500
512:     self.satellites.append(
513:         Satellite(self.player.y, self.player.x)
514:     )
515:
516:     if key == ord("g"):
517:         # deploy lintern
518:         self.linterns.append(Lintern(self.player.y, self.player.x))
519:
520:     if key == ord(" "):
521:         # deploy trap
522:         if self.trap.deployed == False:
523:             self.trap.deployed = True
524:             self.trap.y = self.player.y + self.player.dir_y * 2
525:             self.trap.x = self.player.x + self.player.dir_x * 2
526:
527:     if self.player.to_move:
528:         self.clear(self.player)
529:         self.player.move(
530:             max(1, min(self.max_y, self.player.y + delta_y)),
531:             max(1, min(self.max_x, self.player.x + delta_x)),
532:         )
533:         delta_x = delta_y = 0
534:         self.player.to_move = False
535:
536:     ####
537:
538:     self.render_all()
539:     self.print_stats()
540:
541:     # Gameover Condition
542:     if (
543:         self.player.health <= 0
544:         or self.base.health <= 0
545:         or (self.base.gold < 50 and len(self.mines) == 0)
546:     ):
547:         self.gameover()
548:
549:     # Gamewon Condition
550:     if (
551:         self.player.level > 10
552:         and self.trap.deployed
553:         and distance(self.base, self.trap) <= 3
554:         and distance(self.base, self.player) <= 3
555:         and len(self.enemies) < 2
556:     ):
557:         self.gamewon()
558:
559:     self.screen.refresh()
560:     curses.napms(1000 // FPS)
561:
562:     def print_stats(self):
563:         # print stats
564:         place = nearby_elements(
565:             self.player,

```

```

566:             chain(
567:                 self.mines,
568:                 self.cannons,
569:                 self.mountains,
570:                 self.satellites,
571:                 [
572:                     self.base,
573:                 ],
574:             ),
575:             ret="one",
576:         )
577:
578:     stats_line0 = f"Coord: ({self.player.y:3},{self.player.x:3})"
579:     if place is not None:
580:         stats_line0 += (
581:             f"   Place: {place.kind}, lvl: {place.level}, health: {place.health}"
582:         )
583:
584:         if place.kind == "Mine":
585:             stats_line0 += f", production: {place.production_rate}, cost to (u"
586:             pgrade: {place.cost_to_upgrade()}), (s)ell for {place.cost_to_recover()}"
587:             stats_line0 += f"   Time: {place.time_pending}"
588:
589:         elif place.kind == "Cannon":
590:             stats_line0 += f", kills: {place.kills}, cost to (u)pgrade: {place"
591:             .cost_to_upgrade()}"
592:             stats_line0 += f"   Time: {place.time_pending}"
593:
594:         stats_line1 = f"Level: {self.player.level:2}      "
595:         stats_line1 += f"Health: {self.player.health:3}      "
596:         stats_line1 += f"Points: {self.player.points:3}      "
597:         stats_line1 += f"Base Health: {self.base.health:3}      "
598:         stats_line1 += f"Gold: {self.base.gold:4}      "
599:         stats_line1 += f"Enemies: {len(self.enemies):3}      "
600:         stats_line1 += f"Bombs: {self.player.bombs:3}"
601:
602:         self.screen.addstr(self.max_y + 2, 23, 138 * " ")
603:         self.screen.addstr(self.max_y + 2, 5, stats_line0)
604:         self.screen.addstr(self.max_y + 3, 23, stats_line1)
605:
606:         self.player.level = self.player.points // 20 + 1
607:
608:     def pause(self):
609:         self.centered_msg("PAUSE", None, curses.A_BOLD | curses.A_UNDERLINE)
610:
611:     def centered_msg(self, text, key_continue=None, *args):
612:
613:         if isinstance(text, str):
614:             text = [
615:                 text,
616:             ]
617:
618:         if key_continue == None:
619:             key_str = "any"
620:
621:         else:
622:             key_str = f"'{key_continue}'"
623:
624:         text.append(f"Press {key_str} key to continue")
625:
626:         cols = max(len(t) for t in text) + 2
627:         rows = len(text)

```

```

627:         win = curses.newwin(
628:             rows + 2, cols + 2, (self.max_y - rows) // 2, (self.max_x - cols) // 2
629:         )
630:         win.border(0)
631:
632:         for row, t in enumerate(text):
633:             win.addstr(row + 1, (cols - len(t)) // 2 + 1, t, *args)
634:
635:         win.refresh()
636:
637:         while True:
638:             key = self.screen.getch()
639:             if key is not curses.ERR:
640:                 if key_continue is not None:
641:                     if key == ord(key_continue):
642:                         break
643:                 else:
644:                     break
645:
646:         curses.endwin()
647:         self.render_all(reset_fog=True)
648:
649:     def gameover(self):
650:         self.centered_msg(
651:             "⚠️⚠️⚠️ GAME OVER !!!",
652:             "q",
653:             curses.A_STANDOUT,
654:         )
655:         sys.exit()
656:
657:     def gamewon(self):
658:         self.centered_msg(
659:             ["⚠️⚠️⚠️ CONGRATULATIONS, YOU WON !!!", "This is very impressive"], "q"
660:         )
661:         sys.exit()
662:
663:     def clear(self, *args):
664:         """
665:         clears one pixel from screen
666:         calling with an Element instance (Player, Enemy...), or directly by coordi
nate
667:         """
668:         if isinstance(args[0], Element):
669:             y, x = args[0].y, args[0].x
670:
671:         else:
672:             y, x = args[0:2]
673:
674:         self.screen.addch(y, x, " ", curses.color_pair(1))
675:
676:     def render_all(self, reset_fog=False):
677:         """
678:         render all visible elements and updates fog area
679:         """
680:         ## Update Area Light
681:         self.area_light = set(surrounding_area(self.player, 5, *self.screen_limits)
)
682:
683:         if self.base.deployed:
684:             self.area_light = set(
685:                 chain(
686:                     self.area_light,
687:                     surrounding_area(self.base, 10, *self.screen_limits),
688:
689:                 )
690:             )
691:             if len(self.linterns) > 0:
692:                 self.area_light = set(
693:                     chain(
694:                         self.area_light,
695:                         chain.from_iterable(
696:                             surrounding_area(l, 5, *self.screen_limits)
697:                             for l in self.linterns
698:                         ),
699:                     ),
700:                 )
701:             )
702:             if len(self.satellites) > 0:
703:                 self.area_light = set(
704:                     chain(
705:                         self.area_light,
706:                         chain.from_iterable(
707:                             surrounding_area(s, 10, *self.screen_limits)
708:                             for s in self.satellites
709:                         ),
710:                     ),
711:                 )
712:             )
713:
714:             # Remove fog from light area.
715:             self.render_fog(self.area_light, method="remove")
716:
717:             for item in chain(
718:                 self.mountains,
719:                 self.buildings,
720:                 self.satellites,
721:                 self.linterns,
722:                 self.enemies,
723:                 self.spawners,
724:                 self.fruits,
725:                 self.bombs_activated,
726:                 self.bombs_topick,
727:                 [self.base, self.player, self.trap],
728:             ):
729:
730:                 if (item.y, item.x) in self.area_light:
731:                     self.render(item)
732:
733:             if self.area.difference(self.area_light) != self.area_fog or reset_fog:
734:                 # render fog bg if it has changed or forced to reset
735:                 self.area_fog = self.area.difference(self.area_light)
736:                 self.render_fog(self.area_fog)
737:
738:     def render(self, element, *args, **kwargs):
739:         """
740:         render single element
741:         """
742:
743:         if not element.deployed or not element.visible:
744:             return
745:
746:         c = element.color
747:
748:         if "symbol_overwrite" not in kwargs.keys():
749:             symbol_overwrite = None
750:
751:         else:
752:             symbol = kwargs["symbol_overwrite"]

```

```
753:
754:     if element.symbol is None and symbol_overwrite is None:
755:         print("Element has not symbol defined, this it is not drawable")
756:         raise BaseException
757:
758:     if symbol_overwrite is None:
759:         self.screen.addch(
760:             element.y, element.x, element.symbol, curses.color_pair(c)
761:         )
762:     else:
763:         self.screen.addch(element.y, element.x, symbol, curses.color_pair(c))
764:
765: def render_fog(self, area, method="set"):
766:     if method == "set":
767:         for (y, x) in area:
768:             self.screen.addch(y, x, "-", curses.color_pair(2))
769:
770:     elif method == "remove":
771:         for (y, x) in area:
772:             self.screen.addch(y, x, " ", curses.color_pair(1))
773:
774:
775: def distance(objA, objB):
776:     return objA.distance(objB)
777:
778:
779: def surrounding_area(
780:     obj: Element,
781:     distance: int,
782:     min_y: int,
783:     max_y: int,
784:     min_x: int,
785:     max_x: int,
786:     includes_self: bool = True,
787: ) -> list:
788:     area = [
789:         (max(min_y, min(max_y, (obj.y + dy))), max(min_x, min(max_x, (obj.x + dx)))
790:     )
791:     for dy in range(-distance, distance + 1)
792:     for dx in range(-distance, distance + 1)
793:     if int(math.sqrt((obj.y - (obj.y + dy)) ** 2 + (obj.x - (obj.x + dx)) ** 2
794: ))
795:     <= distance
796: ]
797:
798: if not includes_self:
799:     area = set(area).difference({(obj.y, obj.x)})
800:
801: return list(area)
802:
803: def is_inside(obj: Element, area) -> bool:
804:     return {(obj.y, obj.x)} in area
805:
806:
807: def collision(objA: Element, objB: Element) -> bool:
808:     return objA.distance(objB) == 0
809:
810:
811: def nearby_elements(objA, lst, d=0, ret="all"):
812:     """
813:     returns nearby elements from lst within d distance of objA
814:     """
815:     result = [objB for objB in lst if objB.distance(objA) <= d]
816:
817: if len(result) == 0:
818:     return None
819:
820: if ret == "all":
821:     return result
822:
823: elif ret == "one":
824:     return result[0]
825:
826: elif ret == "choice":
827:     return random.choice(result)
828:
829: def play_sound(asset):
830:     f = Path(f"./assets/{asset}.mp3")
831:     if not f.is_file():
832:         f = Path(f"./assets/{asset}.wav")
833:
834:     if f.is_file():
835:         threading.Thread(target=playsound, args=(f,), daemon=True).start()
836:
837:
838: def start():
839:     game = Game.create()
840:     curses.wrapper(game.init)
841:
842:
843: if __name__ == "__main__":
844:     start()
```

```
1: #!/usr/bin/env python
2: # -*- coding: utf-8 -*-
3: from dataclasses import dataclass, field
4: import time
5: import math
6:
7:
8: @dataclass
9: class Element:
10:     """
11:     Game Element Base Class
12:     """
13:
14:     y: int
15:     x: int
16:     kind: str = ""
17:     color: int = 1 # default color
18:     symbol: None = None
19:     fmt: str = None
20:     deployed: bool = True
21:     visible: bool = True
22:     level: int = 10
23:     health: int = 1000
24:
25:     def distance(self, other):
26:         """
27:         return the euclidean distance between 2 elements
28:         """
29:         return int(math.sqrt((self.x - other.x) ** 2 + (self.y - other.y) ** 2))
30:
31:
32: @dataclass
33: class Mountain(Element):
34:     symbol: str = "^"
35:     kind: str = "Mountain"
36:     resource: str = "Gold"
37:     color: int = 11
38:
39:
40: @dataclass
41: class Building(Element):
42:     base_cost: int = 50
43:     production_rate: int = 5
44:     production_factor: int = 1.5
45:     timer: int = 5
46:     clock: float = field(default_factory=time.time)
47:     visible: bool = True
48:
49:     def cost_to_upgrade(self):
50:         return self.base_cost + self.base_cost * (2 ** (self.level - 1))
51:
52:     def cost_to_recover(self):
53:         return sum(
54:             int(self.base_cost + self.base_cost * (2 ** (lvl - 2))) // 2
55:             for lvl in range(1, self.level + 1)
56:         )
57:
58:     def _process(self):
59:         if time.time() - self.clock > self.timer:
60:             self.clock = time.time()
61:             return True
62:         else:
63:             return False
64:
```

```
65:     @property
66:     def time_pending(self):
67:         return f"{self.timer - (time.time() - self.clock):0.2}"
68:
69:     def upgrade(self):
70:         self.level += 1
71:         self.health = 5 * self.level
72:         self.production_rate = int(self.production_rate * self.production_factor)
73:         self._update_symbol()
74:
75:     def _update_symbol(self):
76:         """
77:         define in each instance of building if different
78:         """
79:         pass
80:
81:
82: @dataclass
83: class Mine(Building):
84:     kind: str = "Mine"
85:     symbol: str = "1"
86:     resource: str = "Gold"
87:     level: int = 1
88:     health: int = 5
89:     timer: int = 0.1
90:
91:     def dig_success(self):
92:         return self._process()
93:
94:     @property
95:     def dig_value(self):
96:         return self.production_rate * self.level
97:
98:     def _update_symbol(self):
99:         self.symbol = str(self.level)
100:         if self.level > 1:
101:             self.color = 15
102:
103:
104: @dataclass
105: class Cannon(Building):
106:     kind: str = "Cannon"
107:     symbol: str = "I"
108:     fmt: str = None
109:     level: int = 2
110:     kills: int = 0
111:     health: int = 6
112:     production_factor: int = 1.2 # factor for upgrade
113:     production_rate: int = 2 # distance
114:     timer: int = 4 # speed
115:
116:     def shot_success(self):
117:         return self._process()
118:
119:     def _update_symbol(self):
120:         symbols = "I V X D I V X D C".split(" ")
121:         self.symbol = symbols[self.level - 1]
122:         if self.level > 4:
123:             self.color = 15
124:
125:
126: @dataclass
127: class Enemy(Element):
128:     symbol: int = 4194430 # curses.ACS_BULLET
```



```
129:     kind: str = "Zombie"
130:     color: int = 5
131:     health: int = 2
132:     level: int = 1
133:
134:     def move(self, new_y, new_x):
135:         self.y = new_y
136:         self.x = new_x
137:
138:
139: @dataclass
140: class Spawner(Element):
141:     symbol: str = "#"
142:     kind: str = "Spawner"
143:     health: int = 10
144:     level: int = 10
145:     color: int = 5
146:
147:     def spawn(self):
148:         return Enemy(self.y, self.x)
149:
150:
151: @dataclass
152: class Fruit(Element):
153:     symbol: int = 4194409 # curses.ACS_LANTERN
154:     color: int = 3
155:
156:
157: @dataclass
158: class Base(Element):
159:     kind: str = "Base"
160:     deployed: bool = False
161:     visible: bool = True
162:     health: int = 100
163:     gold: int = 100
164:     symbol: int = 4194400 # curses.ACS_DIAMOND
165:     color: int = 7
166:
167:
168: @dataclass
169: class Satalite(Element):
170:     kind: str = "Satalite"
171:     visible: bool = True
172:     health: int = 10
173:     symbol: int = 4194400 # curses.ACS_DIAMOND
174:     color: int = 17
175:
176:
177: @dataclass
178: class Lintern(Element):
179:     visible: bool = True
180:     symbol: str = "@"
181:     color: int = 17
182:
183:
184: @dataclass
185: class Trap(Element):
186:     deployed: bool = False
187:     symbol: str = "%"
188:
189:
190: @dataclass
191: class Player(Element):
192:     kind: str = "Player"
```

```
193:     dir_y: int = 0
194:     dir_x: int = 0
195:     health: int = 100
196:     points: int = 0
197:     bombs: int = 2
198:     level: int = 1
199:     to_move: bool = False
200:     symbol: str = "*"
201:     color: int = 13
202:     visible: bool = True
203:
204:     def move(self, new_y, new_x):
205:         self.dir_y = new_y - self.y
206:         self.dir_x = new_x - self.x
207:
208:         self.y = new_y
209:         self.x = new_x
210:
211:
212: @dataclass
213: class Bomb(Element):
214:     symbol: str = "+"
215:     strength: int = 5
216:     timer: int = 2
217:     t0: float = field(default_factory=time.time)
218:
219:     @property
220:     def area(self) -> list:
221:         s = self.strength
222:         return [
223:             (self.y + dy, self.x + dx)
224:             for dy in range(-s, s + 1)
225:             for dx in range(-s, s + 1)
226:             if int(
227:                 math.sqrt((self.x - (self.x + dx)) ** 2 + (self.y - (self.y + dy))
228:                 ** 2)
229:                 )
230:                 <= s
231:         ]
232:
233:     @property
234:     def is_kaboom(self):
235:         """
236:         check if timer is over and returns True to handle bomb self destruction, o
237:         r False otherwise
238:         """
239:         return time.time() - self.t0 > self.timer
```