```
#!/usr/bin/env python
# -*- coding: utf-8 -*

from .lib.elements import Base, Player, Trap, Bomb, Fruit
from .lib.elements import Mountain, Mine, Cannon
from .lib.elements import Spawner, Enemy

from dataclasses import dataclass, field
from playsound import playsound
from itertools import chain
from pathlib import Path

import threading
import random
import curses
import time
import math
import sys
import os

FPS = 50

@dataclass
class Game:
    screen = None

    @classmethod
    def create(cls):
        game = cls()
        return game

    def init(self, screen):

        self.screen = screen

        # Curses Settings
        curses.curs_set(False) # Do not display blinking cursor
        curses.noecho()
        curses.cbreak()
        curses.start_color()

        # Curses Color Pairs
        curses.init_color(curses.COLOR_BLACK,0,100,100)
        curses.init_pair(1, 250, 0) # Default Color
        curses.init_pair(2, 137, 236)

        curses.init_pair(3, curses.COLOR_MAGENTA, 0) # FRUIT
        curses.init_pair(4, curses.COLOR_MAGENTA, 243) # FRUIT

        curses.init_pair(5, curses.COLOR_YELLOW, 0) # ENEMIES
        curses.init_pair(6, curses.COLOR_YELLOW, 243) # ENEMIES

        curses.init_pair(7, curses.COLOR_GREEN, 0) # BASE
        curses.init_pair(8, curses.COLOR_GREEN, 243) # BASE

        curses.init_pair(9, curses.COLOR_BLUE, 0) # ENEMY TRAPPED
        curses.init_pair(10, curses.COLOR_BLUE, 243) # ENEMY TRAPPED

        curses.init_pair(11, curses.COLOR_RED, 0) # MOUNTAIN
        curses.init_pair(12, curses.COLOR_RED, 243) # MOUNTAIN

        curses.init_pair(13, 25, 231)  # PLAYER
        curses.init_pair(14, 25, 247)  # PLAYER

        curses.init_pair(15, 199, 0) # ENEMY TRAPPED
        curses.init_pair(16, 199, 243) # ENEMY TRAPPED

        # Screen Settings
        self.screen.keypad(True)
        self.screen.nodelay(True)
        self.screen.border(0)

        self.min_y, self.min_x = (2, 2)
        self.max_y, self.max_x = tuple(i-j for i,j in zip(self.screen.getmaxyx(),
                                                          (5,2) ))

        # Draw Window Borders
        self.screen.addch(self.max_y+1, 0, curses.ACS_SSSB)
        self.screen.addch(self.max_y+1, self.max_x+1, curses.ACS_SBSS)

        for x in range(1,self.max_x+1):
            self.screen.addch(self.max_y+1, x, curses.ACS_HLINE)

        # Game Elements
        self.player = Player(20, 20)
        self.trap = Trap(20, 20)
        self.base = Base( self.max_y//2, self.max_x//2 , deployed = False)

        self.mountains = [
            Mountain( y, x ) for y, x in [
                (
                    random.randint(self.min_y, self.max_y),
                    random.randint(self.min_x, self.max_x)
                ) for i in range(10)
            ]
        ]

        self.spawners= [
            Spawner( y, x ) for y, x in [
                (
                    random.randint(self.min_y, self.max_y),
                    random.randint(self.min_x, self.max_x)
                ) for i in range(40)
            ]
        ]

        self.mines = []
        self.cannons = []
        self.enemies = []
        self.fruits = []
        self.bombs_topick = []
        self.bombs_activated = []

        self.loop()


    def loop(self):

        area = { (y, x) for y in range(self.min_y, self.max_y+1) for x in range(se
lf.min_x, self.max_x+1)}
        clock = time.time()
        while True:

            buildings = list( chain(self.mines, self.cannons) )

            area_light = set(surronding_area(self.player, 5, self.min_y, self.max_
y, self.min_x, self.max_x))
            if self.base.deployed:
```

```
127:                 area_light = set(chain(area_light,
128:                                 surronding_area(self.base, 10, self.min_y, self.max
_y, self.min_x, self.max_x)))
129:
130:             area_dark = area.difference( area_light )
131:
132:             self.render_fog ( area_light, method = "remove" )
133:
134:
135:             for item in chain(self.mountains,
136:                               buildings,
137:                               self.enemies,
138:                               self.spawners,
139:                               self.fruits,
140:                               self.bombs_activated,
141:                               self.bombs_topick,
142:                               [self.base, self.player, self.trap],):
143:
144:                 if (item.y, item.x) in area_light:
145:                     self.render(item)
146:
147:             self.render_fog ( area_dark )
148:
149:             ## Process Buildings (Mine -> Dig, Cannon -> Shoot...)
150:             ## , unless they are destroyed by an enemy
151:             for building in buildings:
152:                 if building.health <= 0:
153:                     buildings.remove(building)
154:                     self.erase(building)
155:
156:                     if building.kind == "Mine":
157:                         self.mines.remove(building)
158:
159:                     elif building.kind == "Cannon":
160:                         self.cannons.remove(building)
161:
162:                 else:
163:                     if building.kind == "Mine" and building.dig_success():
164:                         self.base.gold += building.dig_value
165:
166:                     elif building.kind == "Cannon" and building.shot_success():
167:                         target = nearby_elements(building,
168:                                                  self.enemies,
169:                                                  d = building.production_rate,
170:                                                  ret='choice')
171:
172:                         if target is not None \
173:                                 and target in self.enemies:
174:                             self.enemies.remove(target)
175:                             self.erase(target)
176:                             self.player.points += 1
177:                             building.kills += 1
178:
179:             ## Spawn Enemies
180:             if random.randint(0, 1000) < 10:
181:                 s = random.choice(self.spawners)
182:                 self.enemies.append( s.spawn() )
183:
184:             ## Enemies movements
185:             if time.time() > clock + max(0.2, 1 - self.player.level / 12 ):
186:                 for enemy in self.enemies:
187:
188:                     ## scan targets
189:                     targets = [{'target': target, 'd': enemy.distance(target)} for
target in chain(buildings, [self.base, self.player,]) ]
190:
191:                     if len(targets) > 0:
192:                         # choose the nearest target and moves towards it
193:                         # TODO: Set weight to target kinds
194:                         target = sorted(targets, key=lambda x: x['d'])[0]['target'
]
195:
196:                         if target.x - enemy.x > 0:
197:                             delta_x = 1
198:                         elif target.x - enemy.x < 0:
199:                             delta_x = -1
200:
201:                         if target.y - enemy.y > 0:
202:                             delta_y = 1
203:                         elif target.y - enemy.y < 0:
204:                             delta_y = -1
205:
206:                     else:
207:                         delta_y = random.randint( -1, 1 )
208:                         delta_x = random.randint( -1, 1 )
209:
210:                     if (enemy.y, enemy.x) in area_light:
211:                         self.erase(enemy)
212:
213:                     enemy.move(max(1, min(self.max_y, enemy.y + delta_y)),
214:                                max(1, min(self.max_x, enemy.x + delta_x)))
215:
216:                     # check collision with player
217:                     if collision(self.player, enemy):
218:                         combat_result = random.randint(0,99)
219:                         if combat_result < 80 and enemy in self.enemies:
220:                             play_sound("pos")
221:                             self.enemies.remove(enemy)
222:                             self.player.points += 1
223:                             self.player.health -= random.randint(0, 2)
224:
225:                         else:
226:                             play_sound("scream_fight")
227:                             self.player.health -= random.randint(5, 10)
228:
229:                     # check collision with buildings
230:                     for building in buildings:
231:                         if collision(enemy, building):
232:                             building.health -= 1
233:
234:                     # check collision with base
235:                     if collision(self.base, enemy) and enemy in self.enemies:
236:                         self.enemies.remove(enemy)
237:                         self.player.points += 1
238:                         self.base.health -= random.randint(0, 2)
239:
240:                     if self.trap.deployed:
241:                         if distance(self.trap, enemy) <= 5\
242:                                 and enemy in self.enemies:
243:                             self.enemies.remove(enemy)
244:                             enemy.color = 9
245:                             self.render(enemy)
246:
247:                 clock = time.time()
248:
249:
250:             delta_x = delta_y = 0
251:
```

```
252:             ## Check Bombs
253:             if len(self.bombs_activated) > 0:
254:                 for bomb in self.bombs_activated:
255:
256:                     for (y, x) in bomb.area:
257:                         if (y > 0 and y < self.max_y) \
258:                                 and ( x > 0 and x < self.max_x):
259:                             self.screen.addstr(y, x, '~', curses.color_pair(2))
260:
261:                     if bomb.is_kaboom:
262:                         play_sound("kaboom")
263:
264:                         victims = nearby_elements(bomb, chain(self.enemies, self.s
pawners),
265:                                                   d = bomb.strength)
266:
267:                         if victims is not None:
268:                             for victim in victims:
269:                                 victim.health -= 5
270:
271:                         if (self.player.y,self.player.x) in bomb.area:
272:                             play_sound("scream-bomb")
273:                             self.player.health -= 50
274:
275:
276:                         for (y, x) in bomb.area:
277:                             if (y > 0 and y < self.max_y)\
278:                                     and (x > 0 and x < self.max_x - 1):
279:                                 self.clear(y, x)
280:
281:                         self.bombs_activated.remove(bomb)
282:                         self.erase(bomb)
283:                         self.screen.refresh()
284:
285:             for enemy in chain(self.enemies, self.spawners):
286:                 if enemy.health < 0 and enemy in chain(self.enemies, self.spawners
):
287:                     if enemy.kind == 'Zombie':
288:                         self.enemies.remove(enemy)
289:                     elif enemy.kind == 'Spawner':
290:                         self.spawners.remove(enemy)
291:
292:                     self.erase(enemy)
293:                     self.player.points += enemy.level
294:
295:
296:             ## Recover Trap
297:             if self.trap.deployed and distance(self.trap, self.player) == 0:
298:                 self.trap.deployed = False
299:
300:             ## Fruit Spawner
301:             if random.randint(0, 1000) < 2:
302:                 self.fruits.append(Fruit(random.randint(self.min_y, self.max_y),
303:                                          random.randint(self.min_x, self.max_x)))
304:
305:             ## Bombs Spawner
306:             if random.randint(0, 1000) < 1:
307:                 self.bombs_topick.append(Bomb(random.randint(self.min_y, self.max_
y),
308:                                               random.randint(self.min_x, self.max_x)))
309:
310:             ## Fruit check for collision
311:             if len(self.fruits) > 0:
312:                 for fruit in self.fruits:
313:                     if collision(self.player, fruit):
314:                         play_sound("bonus")
315:                         self.player.health += 10
316:                         self.fruits.remove(fruit)
317:
318:             if len(self.bombs_topick) > 0:
319:                 for bomb in self.bombs_topick:
320:                     if collision(self.player, bomb):
321:                         play_sound("bonus")
322:                         self.player.bombs += 1
323:                         self.bombs_topick.remove(bomb)
324:
325:
326:             # Wait for a keystroke
327:             key = self.screen.getch()
328:
329:             # Process the keystroke
330:             if key is not curses.ERR:
331:                 if key == ord('q'):
332:                     break
333:
334:                 if key == ord('p'):
335:                     self.pause()
336:
337:                 if key in [ord('h'), curses.KEY_LEFT]:
338:                     self.player.to_move = True
339:                     delta_x = -1
340:
341:                 if key in [ord('l'), curses.KEY_RIGHT]:
342:                     self.player.to_move = True
343:                     delta_x = 1
344:
345:                 if key in [ord('k'), curses.KEY_UP]:
346:                     self.player.to_move = True
347:                     delta_y = -1
348:
349:                 if key in [ord('j'), curses.KEY_DOWN]:
350:                     self.player.to_move = True
351:                     delta_y = 1
352:
353:                 if key == ord('b'):
354:                     # deploy bomb
355:                     if self.player.bombs > 0:
356:                         self.bombs_activated.append(Bomb(self.player.y,self.player
.x))
357:                         self.player.bombs -= 1
358:
359:                 if key == ord('m'):
360:                     # build mine, in the distance of 1 of a mine, but not ontop an
d
361:                     # not possible in an already built mine
362:                     if self.base.deployed \
363:                             and nearby_elements(self.player, chain(buildings, self
.mountains, [self.base,]))  is None  \
364:                             and min(self.player.distance(mnt) for mnt in self.moun
tains) == 1 \
365:                             and self.base.gold >= 50:
366:                         self.base.gold -= 50
367:                         self.mines.append(Mine(self.player.y,self.player.x))
368:
369:                 if key == ord('c'):
370:                     # build cannon
371:                     # not possible in an already built mine
372:                     if self.base.deployed \
```

```
                            and nearby_elements(self.player, chain(buildings, self
.mountains, [self.base,])) is None \
                            and self.base.gold >= 50:
                        self.base.gold -= 50
                        self.cannons.append(Cannon(self.player.y,self.player.x))

                if key == ord('u'):
                    # upgrade building
                    building = nearby_elements(self.player, buildings, ret = 'one'
)

                    if building is not None \
                            and building.level < 9:
                        cost = building.cost_to_upgrade()
                        if self.base.gold >= cost:
                            self.base.gold -= cost
                            building.upgrade()

                if key == ord('s'):
                    # sell building
                    building = nearby_elements(self.player, buildings, ret = 'one'
)
                    if building is not None:
                        self.base.gold += building.cost_to_recover()
                        buildings.remove(building)
                        if building.kind == 'Mine':
                            self.mines.remove(building)
                        elif building.kind == 'Cannon':
                            self.cannons.remove(building)

                if key == ord('v'):
                    # deploy base
                    if not self.base.deployed:
                        self.base.deployed = True
                        self.base.y = self.player.y
                        self.base.x = self.player.x

                if key == ord(' '):
                    # deploy trap
                    if self.trap.deployed == False:
                        self.trap.deployed = True
                        self.trap.y = self.player.y +self.player.dir_y * 2
                        self.trap.x = self.player.x +self.player.dir_x * 2

            if self.player.to_move:
                self.erase(self.player)
                self.player.move(max(1, min(self.max_y,self.player.y + delta_y)),
                            max(1, min(self.max_x, self.player.x + delta_x)))
                delta_x = delta_y = 0
                self.player.to_move = False

            self.print_stats()

            # Gameover Condition
            if self.player.health <= 0 \
                    or self.base.health <= 0 \
                    or (self.base.gold < 50 and len(self.mines) == 0):
                self.gameover()

            # Gamewon Condition
            if self.player.level > 10 \
                    and self.trap.deployed \
                    and distance(self.base, self.trap) <= 3 \
                    and distance(self.base, self.player) <= 3 \
                    and len(self.enemies) < 2:
                self.gamewon()

            self.screen.refresh()
            curses.napms( 1000 // FPS )

    def panic(self):
        pass

    def print_stats(self):
            # print stats
            place = nearby_elements(self.player, chain(self.mines, self.cannons, s
elf.mountains, [self.base, ]), ret='one')

            stats_line0 =  f"Coord: ({self.player.y:3},{self.player.x:3})"
            if place is not None:
                if place.kind == 'Mine':
                    stats_line0 += f"  Place: {place.kind}, lvl: {place.level}, pr
oduction: {place.production_rate}, health: {place.health}, cost to (u)pgrade: {place.cost
_to_upgrade()}, (s)ell for {place.cost_to_recover()}"
                    stats_line0 += f"  Time: {place.time_pending}"

                elif place.kind == 'Cannon':
                    stats_line0 += f"  Place: {place.kind}, lvl: {place.level}, ki
lls: {place.kills}, health: {place.health}, cost to (u)pgrade: {place.cost_to_upgrade()}"
                    stats_line0 += f"  Time: {place.time_pending}"

                else:
                    stats_line0 += f"  Place: {place.kind}, lvl: {place.level}, he
alth: {place.health}"

            stats_line1 =  f"Level: {self.player.level:2}     "
            stats_line1 += f"Health: {self.player.health:3}     "
            stats_line1 += f"Points: {self.player.points:3}     "
            stats_line1 += f"Base Health: {self.base.health:3}     "
            stats_line1 += f"Gold: {self.base.gold:4}     "
            stats_line1 += f"Enemies: {len(self.enemies):3}     "
            stats_line1 += f"Bombs: {self.player.bombs:3}"


            self.screen.addstr(self.max_y +2, 23, 138*" ")
            self.screen.addstr(self.max_y +2, 5, stats_line0)
            self.screen.addstr(self.max_y +3, 23, stats_line1)

            self.player.level =self.player.points // 20 + 1


    def pause(self, key_continue = None):
        while True:
            key = self.screen.getch()
            if key is not curses.ERR:
                if key_continue is not None:
                    if key == ord(key_continue):
                        break
                else:
                    break

    def gameover(self):
        self.screen.addstr(self.max_y//2, self.max_x//2, "Â¡Â¡Â¡ GAME OVER !!!", c
urses.A_BLINK)
        self.pause('q')
        sys.exit()

    def gamewon(self):
```

```
492:         self.screen.addstr(self.max_y//2, self.max_x//2, "Â¡Â¡Â¡ CONGRATULATIONS,
YOU WON !!!", curses.A_BLINK)
493:         self.screen.addstr(self.max_y//2+1, self.max_x//2, "This is very impresive
.", curses.A_BLINK)
494:         self.pause('q')
495:         sys.exit()
496:
497:     def erase(self, element):
498:         """
499:         Erase element position
500:         """
501:         self.clear(element.y, element.x)
502:
503:     def clear(self, y, x):
504:         self.screen.addch(y, x, ' ', curses.color_pair(1))
505:
506:     def render(self, element, *args, **kwargs):
507:         """
508:         render elements in screen
509:         """
510:
511:         if not element.deployed or not element.visible:
512:             return
513:
514:         c = element.color
515:
516:         if 'symbol' not in kwargs.keys():
517:             symbol = None
518:
519:         else:
520:             symbol = kwargs['symbol']
521:
522:         if element.symbol is None and symbol is None:
523:             print("Element has not symbol defined, this it is not drawable")
524:             raise BaseException
525:
526:         if symbol is None: # and element.color is not None:
527:             self.screen.addch(element.y,
528:                               element.x,
529:                               element.symbol,
530:                               curses.color_pair(c) )
531:         else:
532:             self.screen.addch(element.y,
533:                               element.x,
534:                               symbol,
535:                               curses.color_pair(c))
536:
537:     def render_fog(self, area, method = 'set' ):
538:         if method == 'set':
539:             for (y, x) in area:
540:                 self.screen.addch(y, x, '-', curses.color_pair(2))
541:
542:         elif method == 'remove':
543:             for (y, x) in area:
544:                 self.screen.addch(y, x, ' ', curses.color_pair(1))
545:
546: def distance(objA, objB):
547:     return objA.distance(objB)
548:
549: def surronding_area(obj, distance,  min_y, max_y, min_x, max_x, includes_self = Tr
ue):
550:         area = [( max(min_y, min(max_y, (obj.y + dy))),
551:                   max(min_x, min(max_x, (obj.x + dx))) ) \
552:                 for dy in range(-distance, distance + 1) \
553:                 for dx in range(-distance, distance + 1) \
554:                 if int(
555:                     math.sqrt(
556:                         (obj.y-(obj.y + dy))**2 + (obj.x-(obj.x+dx))**2
557:                     )
558:                 ) <= distance ]
559:
560:         if not includes_self:
561:             area = set(area).difference({(obj.y, obj.x)})
562:
563:         return list(area)
564:
565: def is_inside(obj, area):
566:     return {(obj.y, obj.x)} in area
567:
568: def collision(objA, objB):
569:     return objA.distance(objB) == 0
570:
571: def nearby_elements(objA, lst, d = 0, ret='all'):
572:     """
573:     returns nearby elements from lst within d distance of objA
574:     """
575:     result = [objB for objB in lst if objB.distance(objA)<=d]
576:
577:     if len(result) == 0:
578:         return None
579:
580:     if ret == "all":
581:         return result
582:
583:     elif ret == "one":
584:         return result[0]
585:
586:     elif ret == "choice":
587:         return random.choice(result)
588:
589: def play_sound(asset):
590:     f = Path(f"./assets/{asset}.mp3")
591:     if not f.is_file():
592:         f = Path(f"./assets/{asset}.wav")
593:
594:     if f.is_file():
595:         threading.Thread(target=playsound, args=(f,), daemon=True).start()
596:
597: def start():
598:     game = Game.create()
599:     curses.wrapper(game.init)
600:
601: if __name__ == "__main__":
602:     start()
```

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from dataclasses import dataclass, field
import time
import math

@dataclass
class Element:
    """
    Game Element Base Class
    """
    y: int
    x: int
    kind: str = ""
    color: int = 1 # default color
    symbol: None = None
    fmt: str = None
    deployed: bool = True
    visible: bool = True
    level: int = 10
    health: int = 1000

    def distance(self, other):
        """
        return the euclidean distance between 2 elements
        """
        return int(math.sqrt((self.x - other.x)**2 + (self.y-other.y)**2))

@dataclass
class Mountain(Element):
    symbol: str = "^"
    kind: str = "Mountain"
    resource: str = "Gold"
    color: int = 11


@dataclass
class Building(Element):
    base_cost: int = 50
    production_rate: int = 5
    production_factor: int = 1.5
    timer: int = 5
    clock: float = field(default_factory = time.time)
    visible: bool = True

    def cost_to_upgrade(self):
        return self.base_cost + self.base_cost * (2 ** (self.level - 1))

    def cost_to_recover(self):
        return sum (
            int(self.base_cost + self.base_cost * (2 ** (lvl - 2))) // 2 \
            for lvl in range(1, self.level + 1)
        )

    def _process(self):
        if time.time() - self.clock > self.timer:
            self.clock = time.time()
            return True
        else:
            return False

    @property
    def time_pending(self):
        return f"{self.timer - (time.time() - self.clock):0.2}"

    def upgrade(self):
        self.level += 1
        self.health = 5 * self.level
        self.production_rate = int(self.production_rate * self.production_factor)
        self._update_symbol()

    def _update_symbol(self):
        """
        define in each instance of building if different
        """
        pass

@dataclass
class Mine(Building):
    kind: str = "Mine"
    symbol: str = "1"
    resource: str = "Gold"
    level: int = 1
    health: int = 5
    timer: int = 2

    def dig_success(self):
        return self._process()

    @property
    def dig_value(self):
        return self.production_rate * self.level

    def _update_symbol(self):
        self.symbol = str(self.level)
        if self.level > 1:
            self.color = 15

@dataclass
class Cannon(Building):
    kind: str = "Cannon"
    symbol: str = "I"
    fmt: str = None
    level: int = 2
    kills: int = 0
    health: int = 6
    production_factor: int = 1.2 # factor for upgrade
    production_rate: int = 2 # distance
    timer: int = 4   # speed

    def shot_success(self):
        return self._process()

    def _update_symbol(self):
        symbols = 'I V X D I V X D C'.split(' ')
        self.symbol = symbols[self.level-1]
        if self.level > 4:
            self.color = 15

@dataclass
class Enemy(Element):
    symbol: int = 4194430 # curses.ACS_BULLET
    kind: str = "Zombie"
    color: int = 5
    health: int = 2
    level: int = 1

    def move(self, new_y, new_x):
```

```
129:         self.y = new_y
130:         self.x = new_x
131:
132: @dataclass
133: class Spawner(Element):
134:     symbol: str = "#"
135:     kind: str = "Spawner"
136:     health: int = 10
137:     level: int = 10
138:     color: int = 5
139:
140:     def spawn(self):
141:         return Enemy(self.y, self.x)
142:
143: @dataclass
144: class Fruit(Element):
145:     symbol: int = 4194409 # curses.ACS_LANTERN
146:     color: int = 3
147:
148: @dataclass
149: class Base(Element):
150:     deployed: bool = False
151:     visible: bool = True
152:     health: int = 100
153:     gold: int = 100
154:     symbol: int = 4194400 # curses.ACS_DIAMOND
155:     color: int = 7
156:
157: @dataclass
158: class Trap(Element):
159:     deployed: bool = False
160:     symbol: str = "%"
161:
162: @dataclass
163: class Player(Element):
164:     dir_y: int = 0
165:     dir_x: int = 0
166:     health: int = 100
167:     points: int = 0
168:     bombs: int = 2
169:     level: int = 1
170:     to_move: bool = False
171:     symbol: str = '*'
172:     color: int = 13
173:     visible: bool = True
174:
175:     def move(self, new_y, new_x):
176:         self.dir_y = new_y - self.y
177:         self.dir_x = new_x - self.x
178:
179:         self.y = new_y
180:         self.x = new_x
181:
182: @dataclass
183: class Bomb(Element):
184:     symbol: str = '+'
185:     strength: int = 5
186:     timer: int = 2
187:     t0: float = field(default_factory = time.time)
188:
189:     @property
190:     def area(self) -> list:
191:         s = self.strength
192:         return [(self.y + dy, self.x + dx) \
193:                 for dy in range(-s, s + 1) \
194:                 for dx in range(-s, s + 1) \
195:                 if int(
196:                     math.sqrt(
197:                         (self.x-(self.x+dx))**2 + (self.y-(self.y+dy))**2
198:                     )
199:                 ) <= s ]
200:
201:     @property
202:     def is_kaboom(self):
203:         """
204:         check if timer is over and returns True to handle bomb self destruction, o
r False otherwise
205:         """
206:
207:         return time.time() - self.t0 > self.timer
```