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

from .lib.elements import Base, Player, Trap, Bomb, Fruit, Lintern
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

        curses.init_pair(17, 225, 0)  # LINTERN
        curses.init_pair(18, 225, 243)  # LINTERN

        # 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))
        )

        self.screen_limits = (self.min_y, self.max_y, self.min_x, self.max_x)
        self.screen_size = (self.max_x - self.min_x) * (self.max_y - self.min_y)

        # 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(self.screen_size // 400)
            ]
        ]

        self.mines = []
        self.cannons = []
        self.linterns = []
        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(self.min_x, self.max_x + 1)
        }
        area_fog = set()
        clock = time.time()
        while True:

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

            area_light = set(surronding_area(self.player, 5, *self.screen_limits))

            if len(self.linterns) > 0:

                area_light = set(
                    chain(
                        area_light,
                        chain.from_iterable(
                            surronding_area(l, 5, *self.screen_limits)
                            for l in self.linterns
                        ),
                    )
                )

            if self.base.deployed:
                area_light = set(
                    chain(
                        area_light,
                        surronding_area(
                            self.base,
                            10,
                            self.min_y,
                            self.max_y,
                            self.min_x,
                            self.max_x,
                        ),
                    )
                )

            # Remove fog from light area.
            self.render_fog(area_light, method="remove")

            for item in chain(
                self.mountains,
                buildings,
                self.linterns,
                self.enemies,
                self.spawners,
                self.fruits,
                self.bombs_activated,
                self.bombs_topick,
                [self.base, self.player, self.trap],
            ):

                if (item.y, item.x) in area_light:
                    self.render(item)

            if area.difference(area_light) != area_fog:
                # render background if it has changed
                area_fog = area.difference(area_light)
                self.render_fog(area_fog)


            ## Process Buildings (Mine -> Dig, Cannon -> Shoot...)
            ## , unless they are destroyed by an enemy
            for building in buildings:
                if building.health <= 0:
                    buildings.remove(building)
                    self.erase(building)

                    if building.kind == "Mine":
                        self.mines.remove(building)

                    elif building.kind == "Cannon":
                        self.cannons.remove(building)

                else:
                    if building.kind == "Mine" and building.dig_success():
                        self.base.gold += building.dig_value

                    elif building.kind == "Cannon" and building.shot_success():
                        target = nearby_elements(
                            building,
                            self.enemies,
                            d=building.production_rate,
                            ret="choice",
                        )

                        if target is not None and target in self.enemies:
                            self.enemies.remove(target)
                            self.erase(target)
                            self.player.points += 1
                            building.kills += 1

            ## Spawn Enemies
            if random.randint(0, 1000) < 10:
                s = random.choice(self.spawners)
                self.enemies.append(s.spawn())

            ## Enemies movements
            if time.time() > clock + max(0.2, 1 - self.player.level / 12):
                for enemy in self.enemies:

                    ## scan targets
                    targets = [
                        {"target": target, "d": enemy.distance(target)}
                        for target in chain(
                            buildings,
                            [
                                self.base,
                                self.player,
                            ],
                        )
                    ]

                    if len(targets) > 0:
                        # choose the nearest target and moves towards it
                        # TODO: Set weight to target kinds
                        target = sorted(targets, key=lambda x: x["d"])[0]["target"
]

                        if target.x - enemy.x > 0:
                            delta_x = 1
                        elif target.x - enemy.x < 0:
                            delta_x = -1
```

```
                    if target.y - enemy.y > 0:
                        delta_y = 1
                    elif target.y - enemy.y < 0:
                        delta_y = -1

                else:
                    delta_y = random.randint(-1, 1)
                    delta_x = random.randint(-1, 1)

                if (enemy.y, enemy.x) in area_light:
                    self.erase(enemy)

                enemy.move(
                    max(1, min(self.max_y, enemy.y + delta_y)),
                    max(1, min(self.max_x, enemy.x + delta_x)),
                )

                # check collision with player
                if collision(self.player, enemy):
                    combat_result = random.randint(0, 99)
                    if combat_result < 80 and enemy in self.enemies:
                        play_sound("pos")
                        self.enemies.remove(enemy)
                        self.player.points += 1
                        self.player.health -= random.randint(0, 2)

                    else:
                        play_sound("scream_fight")
                        self.player.health -= random.randint(5, 10)

                # check collision with buildings
                for building in buildings:
                    if collision(enemy, building):
                        building.health -= 1

                # check collision with base
                if collision(self.base, enemy) and enemy in self.enemies:
                    self.enemies.remove(enemy)
                    self.player.points += 1
                    self.base.health -= random.randint(0, 2)

                if self.trap.deployed:
                    if distance(self.trap, enemy) <= 5 and enemy in self.enemi
es:

                        self.enemies.remove(enemy)
                        enemy.color = 9
                        self.render(enemy)

            clock = time.time()

        delta_x = delta_y = 0

        ## Check Bombs
        if len(self.bombs_activated) > 0:
            for bomb in self.bombs_activated:

                for (y, x) in bomb.area:
                    if (y > 0 and y < self.max_y) and (x > 0 and x < self.max_
x):

                        self.screen.addstr(y, x, "~", curses.color_pair(2))

                if bomb.is_kaboom:
                    play_sound("kaboom")

                    victims = nearby_elements(
                        bomb, chain(self.enemies, self.spawners), d=bomb.stren
gth
                    )

                    if victims is not None:
                        for victim in victims:
                            victim.health -= 550

                    if (self.player.y, self.player.x) in bomb.area:
                        play_sound("scream-bomb")
                        self.player.health -= 510

                    for (y, x) in bomb.area:
                        if (y > 0 and y < self.max_y) and (
                            x > 0 and x < self.max_x - 1
                        ):
                            self.clear(y, x)

                    self.bombs_activated.remove(bomb)
                    self.erase(bomb)
                    self.screen.refresh()

        for enemy in chain(self.enemies, self.spawners):
            if enemy.health < 0 and enemy in chain(self.enemies, self.spawners
):
                if enemy.kind == "Zombie":
                    self.enemies.remove(enemy)
                elif enemy.kind == "Spawner":
                    self.spawners.remove(enemy)

                self.erase(enemy)
                self.player.points += enemy.level

        ## Recover Trap
        if self.trap.deployed and distance(self.trap, self.player) == 0:
            self.trap.deployed = False

        ## Fruit Spawner
        if random.randint(0, 1000) < 2:
            self.fruits.append(
                Fruit(
                    random.randint(self.min_y, self.max_y),
                    random.randint(self.min_x, self.max_x),
                )
            )

        ## Bombs Spawner
        if random.randint(0, 1000) < 1:
            self.bombs_topick.append(
                Bomb(
                    random.randint(self.min_y, self.max_y),
                    random.randint(self.min_x, self.max_x),
                )
            )

        ## Fruit check for collision
        if len(self.fruits) > 0:
            for fruit in self.fruits:
                if collision(self.player, fruit):
                    play_sound("bonus")
                    self.player.health += 10
                    self.fruits.remove(fruit)

```

```
380:             if len(self.bombs_topick) > 0:
381:                 for bomb in self.bombs_topick:
382:                     if collision(self.player, bomb):
383:                         play_sound("bonus")
384:                         self.player.bombs += 1
385:                         self.bombs_topick.remove(bomb)
386:
387:             # Wait for a keystroke
388:
389:             #  key_bindings ={'q': exit,
390:             #  'h': move_left,
391:             #  'j': move_down,
392:             #  'k': move_up,
393:             #  'l': move_right,
394:             #
395:             #  'curses_KEY_LEFT': move_left,
396:             #  'curses_KEY_DOWN': move_down,
397:             #  'curses_KEY_UP': move_up,
398:             #  'curses_KEY_RIGHT': move_right,
399:             #
400:             #  'v': build_base,
401:             #  'm': build_mine,
402:             #  'c': build_cannon,
403:             #  'u': upgrade_building,
404:             #  's': sell_building,
405:             #  'b': deploy_bomb,
406:             #  ' ': deplot_trap,
407:             #  }
408:             key = self.screen.getch()
409:
410:             # Process the keystroke
411:             if key is not curses.ERR:
412:                 if key == ord("q"):
413:                     break
414:
415:                 if key == ord("p"):
416:                     self.pause()
417:
418:                 if key in [ord("h"), curses.KEY_LEFT]:
419:                     self.player.to_move = True
420:                     delta_x = -1
421:
422:                 if key in [ord("l"), curses.KEY_RIGHT]:
423:                     self.player.to_move = True
424:                     delta_x = 1
425:
426:                 if key in [ord("k"), curses.KEY_UP]:
427:                     self.player.to_move = True
428:                     delta_y = -1
429:
430:                 if key in [ord("j"), curses.KEY_DOWN]:
431:                     self.player.to_move = True
432:                     delta_y = 1
433:
434:                 if key == ord("b"):
435:                     # deploy bomb
436:                     if self.player.bombs > 0:
437:                         self.bombs_activated.append(Bomb(self.player.y, self.playe
r.x))
438:                         self.player.bombs -= 1
439:
440:                 if key == ord("m"):
441:                     # build mine, in the distance of 1 of a mine, but not ontop an
d
442:                     # not possible in an already built mine
443:                     if (
444:                         self.base.deployed
445:                         and nearby_elements(
446:                             self.player,
447:                             chain(
448:                                 buildings,
449:                                 self.mountains,
450:                                 [
451:                                     self.base,
452:                                 ],
453:                             ),
454:                         )
455:                         is None
456:                         and min(self.player.distance(mnt) for mnt in self.mountain
s)
457:                         == 1
458:                         and self.base.gold >= 50
459:                     ):
460:                         self.base.gold -= 50
461:                         self.mines.append(Mine(self.player.y, self.player.x))
462:
463:                 if key == ord("c"):
464:                     # build cannon
465:                     # not possible in an already built mine
466:                     if (
467:                         self.base.deployed
468:                         and nearby_elements(
469:                             self.player,
470:                             chain(
471:                                 buildings,
472:                                 self.mountains,
473:                                 [
474:                                     self.base,
475:                                 ],
476:                             ),
477:                         )
478:                         is None
479:                         and self.base.gold >= 50
480:                     ):
481:                         self.base.gold -= 50
482:                         self.cannons.append(Cannon(self.player.y, self.player.x))
483:
484:                 if key == ord("u"):
485:                     # upgrade building
486:                     building = nearby_elements(self.player, buildings, ret="one")
487:
488:                     if building is not None and building.level < 9:
489:                         cost = building.cost_to_upgrade()
490:                         if self.base.gold >= cost:
491:                             self.base.gold -= cost
492:                             building.upgrade()
493:
494:                 if key == ord("s"):
495:                     # sell building
496:                     building = nearby_elements(self.player, buildings, ret="one")
497:                     if building is not None:
498:                         self.base.gold += building.cost_to_recover()
499:                         buildings.remove(building)
500:                         if building.kind == "Mine":
501:                             self.mines.remove(building)
502:                         elif building.kind == "Cannon":
503:                             self.cannons.remove(building)
504:
```

```
505:                 if key == ord("v"):
506:                     # deploy base
507:                     if not self.base.deployed:
508:                         self.base.deployed = True
509:                         self.base.y = self.player.y
510:                         self.base.x = self.player.x
511:
512:                 if key == ord("g"):
513:                     # deploy lintern
514:                     self.linterns.append(Lintern(self.player.y, self.player.x))
515:
516:                 if key == ord(" "):
517:                     # deploy trap
518:                     if self.trap.deployed == False:
519:                         self.trap.deployed = True
520:                         self.trap.y = self.player.y + self.player.dir_y * 2
521:                         self.trap.x = self.player.x + self.player.dir_x * 2
522:
523:             if self.player.to_move:
524:                 self.erase(self.player)
525:                 self.player.move(
526:                     max(1, min(self.max_y, self.player.y + delta_y)),
527:                     max(1, min(self.max_x, self.player.x + delta_x)),
528:                 )
529:                 delta_x = delta_y = 0
530:                 self.player.to_move = False
531:
532:             ####
533:
534:             def exit():
535:                 return
536:
537:             #####
538:
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 panic(self):
563:         pass
564:
565:     def print_stats(self):
566:         # print stats
567:         place = nearby_elements(
568:             self.player,
569:             chain(
570:                 self.mines,
571:                 self.cannons,
572:                 self.mountains,
573:                 [
574:                     self.base,
575:                 ],
576:             ),
577:             ret="one",
578:         )
579:
580:         stats_line0 = f"Coord: ({self.player.y:3},{self.player.x:3})"
581:         if place is not None:
582:             if place.kind == "Mine":
583:                 stats_line0 += f"  Place: {place.kind}, lvl: {place.level}, produc
tion: {place.production_rate}, health: {place.health}, cost to (u)pgrade: {place.cost_to_
upgrade()}, (s)ell for {place.cost_to_recover()}"
584:                 stats_line0 += f"  Time: {place.time_pending}"
585:
586:             elif place.kind == "Cannon":
587:                 stats_line0 += f"  Place: {place.kind}, lvl: {place.level}, kills:
 {place.kills}, health: {place.health}, cost to (u)pgrade: {place.cost_to_upgrade()}"
588:                 stats_line0 += f"  Time: {place.time_pending}"
589:
590:             else:
591:                 stats_line0 += (
592:                     f"  Place: {place.kind}, lvl: {place.level}, health: {place.he
alth}"
593:                 )
594:
595:         stats_line1 = f"Level: {self.player.level:2}     "
596:         stats_line1 += f"Health: {self.player.health:3}     "
597:         stats_line1 += f"Points: {self.player.points:3}     "
598:         stats_line1 += f"Base Health: {self.base.health:3}     "
599:         stats_line1 += f"Gold: {self.base.gold:4}     "
600:         stats_line1 += f"Enemies: {len(self.enemies):3}     "
601:         stats_line1 += f"Bombs: {self.player.bombs:3}"
602:
603:         self.screen.addstr(self.max_y + 2, 23, 138 * " ")
604:         self.screen.addstr(self.max_y + 2, 5, stats_line0)
605:         self.screen.addstr(self.max_y + 3, 23, stats_line1)
606:
607:         self.player.level = self.player.points // 20 + 1
608:
609:     def pause(self, key_continue=None):
610:         while True:
611:             key = self.screen.getch()
612:             if key is not curses.ERR:
613:                 if key_continue is not None:
614:                     if key == ord(key_continue):
615:                         break
616:                 else:
617:                     break
618:
619:     def gameover(self):
620:         self.screen.addstr(
621:             self.max_y // 2, self.max_x // 2, "Â¡Â¡Â¡ GAME OVER !!!", curses.A_BLI
NK
622:         )
623:         self.pause("q")
624:         sys.exit()
625:
626:     def gamewon(self):
627:         self.screen.addstr(
```

```
            self.max_y // 2,
            self.max_x // 2,
            "Â¡Â¡Â¡ CONGRATULATIONS, YOU WON !!!",
            curses.A_BLINK,
        )
        self.screen.addstr(
            self.max_y // 2 + 1,
            self.max_x // 2,
            "This is very impresive.",
            curses.A_BLINK,
        )
        self.pause("q")
        sys.exit()

    def erase(self, element):
        """
        Erase element position
        """
        self.clear(element.y, element.x)

    def clear(self, y, x):
        self.screen.addch(y, x, " ", curses.color_pair(1))

    def render(self, element, *args, **kwargs):
        """
        render elements in screen
        """

        if not element.deployed or not element.visible:
            return

        c = element.color

        if "symbol" not in kwargs.keys():
            symbol = None

        else:
            symbol = kwargs["symbol"]

        if element.symbol is None and symbol is None:
            print("Element has not symbol defined, this it is not drawable")
            raise BaseException

        if symbol is None:  # and element.color is not None:
            self.screen.addch(
                element.y, element.x, element.symbol, curses.color_pair(c)
            )
        else:
            self.screen.addch(element.y, element.x, symbol, curses.color_pair(c))

    def render_fog(self, area, method="set"):
        if method == "set":
            for (y, x) in area:
                self.screen.addch(y, x, "-", curses.color_pair(2))

        elif method == "remove":
            for (y, x) in area:
                self.screen.addch(y, x, " ", curses.color_pair(1))


def distance(objA, objB):
    return objA.distance(objB)


def surronding_area(obj, distance, min_y, max_y, min_x, max_x, includes_self=True)
:
    area = [
        (max(min_y, min(max_y, (obj.y + dy))), max(min_x, min(max_x, (obj.x + dx))
))
        for dy in range(-distance, distance + 1)
        for dx in range(-distance, distance + 1)
        if int(math.sqrt((obj.y - (obj.y + dy)) ** 2 + (obj.x - (obj.x + dx)) ** 2
))
        <= distance
    ]

    if not includes_self:
        area = set(area).difference({(obj.y, obj.x)})

    return list(area)


def is_inside(obj, area):
    return {(obj.y, obj.x)} in area


def collision(objA, objB):
    return objA.distance(objB) == 0


def nearby_elements(objA, lst, d=0, ret="all"):
    """
    returns nearby elements from lst within d distance of objA
    """
    result = [objB for objB in lst if objB.distance(objA) <= d]

    if len(result) == 0:
        return None

    if ret == "all":
        return result

    elif ret == "one":
        return result[0]

    elif ret == "choice":
        return random.choice(result)


def play_sound(asset):
    f = Path(f"./assets/{asset}.mp3")
    if not f.is_file():
        f = Path(f"./assets/{asset}.wav")

    if f.is_file():
        threading.Thread(target=playsound, args=(f,), daemon=True).start()


def start():
    game = Game.create()
    curses.wrapper(game.init)


if __name__ == "__main__":
    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 = 0.1

    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
```

```
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:     deployed: bool = False
160:     visible: bool = True
161:     health: int = 100
162:     gold: int = 100
163:     symbol: int = 4194400  # curses.ACS_DIAMOND
164:     color: int = 7
165:
166:
167: @dataclass
168: class Lintern(Element):
169:     visible: bool = True
170:     symbol: str = "@"
171:     color: int = 17
172:
173:
174: @dataclass
175: class Trap(Element):
176:     deployed: bool = False
177:     symbol: str = "%"
178:
179:
180: @dataclass
181: class Player(Element):
182:     dir_y: int = 0
183:     dir_x: int = 0
184:     health: int = 100
185:     points: int = 0
186:     bombs: int = 2
187:     level: int = 1
188:     to_move: bool = False
189:     symbol: str = "*"
190:     color: int = 13
191:     visible: bool = True
192:
193:     def move(self, new_y, new_x):
194:         self.dir_y = new_y - self.y
195:         self.dir_x = new_x - self.x
196:
197:         self.y = new_y
198:         self.x = new_x
199:
200:
201: @dataclass
202: class Bomb(Element):
203:     symbol: str = "+"
204:     strength: int = 5
205:     timer: int = 2
206:     t0: float = field(default_factory=time.time)
207:
208:     @property
209:     def area(self) -> list:
210:         s = self.strength
211:         return [
212:             (self.y + dy, self.x + dx)
213:             for dy in range(-s, s + 1)
214:             for dx in range(-s, s + 1)
215:             if int(
216:                 math.sqrt((self.x - (self.x + dx)) ** 2 + (self.y - (self.y + dy))
 ** 2)
217:             )
218:             <= s
219:         ]
220:
221:     @property
222:     def is_kaboom(self):
223:         """
224:         check if timer is over and returns True to handle bomb self destruction, o
r False otherwise
225:         """
226:
227:         return time.time() - self.t0 > self.timer
```