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

from .lib.elements import Element, Base, Satelite, Player, Trap, Bomb, Fruit, Lint
ern
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 = (1, 1)
        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.satelites = []
        self.mines = []
        self.cannons = []
        self.linterns = []
        self.enemies = []
        self.fruits = []
        self.bombs_topick = []
        self.bombs_activated = []

        self.area = set(
            (y, x)
            for y in range(self.min_y, self.max_y + 1)
            for x in range(self.min_x, self.max_x + 1)
        )
        self.area_fog = set()

        self.loop()

    def loop(self):

        clock = time.time()
        while True:

            # 1. Process Buildings (Mine -> Dig, Cannon -> Shoot...)
            #    ,unless they are destroyed by an enemy

            self.buildings = list(chain(self.mines, self.cannons, self.satelites))
            for building in self.buildings:
                if building.health <= 0:
                    self.buildings.remove(building)
                    self.clear(building)

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

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

                    elif building.kind == "Satelite":
                        # When a satelite is destroyed, all dependent buildings collapses next turn.
                        self.satelites.remove(building)
                        for building_dep in nearby_elements(
                            building, chain(self.mines, self.cannons), 10
                        ):
                            building_dep.health = 0

                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.clear(target)
                            self.player.points += 1
                            building.kills += 1

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

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

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

                    # b. Choose the nearest target and moves towards it
                    # TODO: Set weight to target kinds
                    if len(targets) > 0:
                        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

                    # if no targets, move randomly
                    else:
                        delta_y = random.randint(-1, 1)
                        delta_x = random.randint(-1, 1)

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

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

                    # c. check collisions with player, buildings, base
                    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)

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

                    for b in chain(
                        self.satelites,
                        [
                            self.base,
                        ],
                    ):
                        if collision(b, enemy) and enemy in self.enemies:
                            self.enemies.remove(enemy)
                            self.player.points += 1
                            b.health -= random.randint(0, 2)

                    if self.trap.deployed:
                        if distance(self.trap, enemy) <= 5 and enemy in self.enemies:
                            self.enemies.remove(enemy)
                            enemy.color = 9
                            self.render(enemy)

                clock = time.time()

            delta_x = delta_y = 0

            # 4. Monitor Activated 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(4))

                    if bomb.is_kaboom:
                        play_sound("kaboom")

                        victims = nearby_elements(
                            bomb,
                            chain(
                                self.enemies,
                                self.spawners,
                                [
                                    self.player,
                                ],
                            ),
                            d=bomb.strength,
                        )

                        if victims is not None:
                            for victim in victims:
                                if victim.kind == "Player":
                                    play_sound("scream-bomb")
                                    self.player.health -= 50

                                else:
                                    victim.health -= 5

                        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.clear(bomb)

            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.clear(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)

            if len(self.bombs_topick) > 0:
                for bomb in self.bombs_topick:
                    if collision(self.player, bomb):
                        play_sound("bonus")
                        self.player.bombs += 1
                        self.bombs_topick.remove(bomb)

            # Wait for a keystroke

            #  key_bindings ={'q': sys.exit,
            #  'h': move_left,
            #  'j': move_down,
            #  'k': move_up,
            #  'l': move_right,
            #
            #  'curses_KEY_LEFT': move_left,
            #  'curses_KEY_DOWN': move_down,
            #  'curses_KEY_UP': move_up,
            #  'curses_KEY_RIGHT': move_right,
            #
            #  'v': build_base,
            #  'm': build_mine,
            #  'c': build_cannon,
            #  'u': upgrade_building,
            #  's': sell_building,
            #  'b': deploy_bomb,
            #  ' ': deplot_trap,
            #  }
            key = self.screen.getch()

            # Process the keystroke
            if key is not curses.ERR:
                if key == ord("q"):
                    break

                if key == ord("p"):
                    self.pause()

                if key in [ord("h"), curses.KEY_LEFT]:
                    self.player.to_move = True
                    delta_x = -1

                if key in [ord("l"), curses.KEY_RIGHT]:
                    self.player.to_move = True
                    delta_x = 1

                if key in [ord("k"), curses.KEY_UP]:
                    self.player.to_move = True
                    delta_y = -1

                if key in [ord("j"), curses.KEY_DOWN]:
                    self.player.to_move = True
                    delta_y = 1

                if key == ord("b"):
                    # deploy bomb
                    if self.player.bombs > 0:
                        self.bombs_activated.append(Bomb(self.player.y, self.player.x))

                        self.player.bombs -= 1

                if key == ord("m"):
                    # build mine, in the distance of 1 of a mine, but not ontop and
                    # not possible in an already built mine
                    if (
                        self.base.deployed
                        and nearby_elements(
                            self.player,
                            chain(
                                self.buildings,
                                self.mountains,
                                [
                                    self.base,
                                ],
                            ),
                        )
                        is None
                        and nearby_elements(
                            self.player,
                            chain(
                                self.satelites,
                                [
                                    self.base,
                                ],
                            ),
                            d=10,
                        )
                        is not None
                        and min(self.player.distance(mnt) for mnt in self.mountains)
                        == 1
                        and self.base.gold >= 50
                    ):
                        self.base.gold -= 50
                        self.mines.append(Mine(self.player.y, self.player.x))

                if key == ord("c"):
                    # build cannon
                    # not possible in an already built mine
                    if (
                        self.base.deployed
                        and nearby_elements(
                            self.player,
                            chain(
                                self.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, self.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, self.buildings, ret="one")

                    if building is not None:
                        self.base.gold += building.cost_to_recover()
                        self.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

                    # deploy satelite
                    else:
                        if (
                            nearby_elements(
                                self.player,
                                chain(
                                    self.satelites,
```

```python
                                    [
                                        self.base,
                                    ],
                                ),
                                d=20,
                            )
                            is None
                            and self.base.gold >= 500
                        ):
                            self.base.gold -= 500
                            self.satelites.append(
                                Satelite(self.player.y, self.player.x)
                            )

                if key == ord("g"):
                    # deploy lintern
                    self.linterns.append(Lintern(self.player.y, 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.clear(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.render_all()
            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 print_stats(self):
        # print stats
        place = nearby_elements(
            self.player,
            chain(
                self.mines,
                self.cannons,
                self.mountains,
                self.satelites,
                [
                    self.base,
                ],
            ),
            ret="one",
        )

        stats_line0 = f"Coord: ({self.player.y:3},{self.player.x:3})"
        if place is not None:
            stats_line0 += (
                f"  Place: {place.kind}, lvl: {place.level}, health: {place.health
}"
            )

            if place.kind == "Mine":
                stats_line0 += f", production: {place.production_rate}, 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", kills: {place.kills}, cost to (u)pgrade: {place
.cost_to_upgrade()}"
                stats_line0 += f"  Time: {place.time_pending}"

        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):
        self.centered_msg("PAUSE", None, curses.A_BOLD | curses.A_UNDERLINE)

    def centered_msg(self, text, key_continue=None, *args):

        if isinstance(text, str):
            text = [
                text,
            ]

        if key_continue == None:
            key_str = "any"

        else:
            key_str = f"'{key_continue}'"

        text.append(f"Press {key_str} key to continue")

        cols = max(len(t) for t in text) + 2
        rows = len(text)

        win = curses.newwin(
            rows + 2, cols + 2, (self.max_y - rows) // 2, (self.max_x - cols) // 2
        )
        win.border(0)

        for row, t in enumerate(text):
            win.addstr(row + 1, (cols - len(t)) // 2 + 1, t, *args)

        win.refresh()

        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

        curses.endwin()
        self.render_all(reset_fog=True)

    def gameover(self):
        self.centered_msg(
            "Â¡Â¡Â¡ GAME OVER !!!",
            "q",
            curses.A_STANDOUT,
        )
        sys.exit()

    def gamewon(self):
        self.centered_msg(
            ["Â¡Â¡Â¡ CONGRATULATIONS, YOU WON !!!", "This is very impresive"], "q"
        )
        sys.exit()

    def clear(self, *args):
        """
        clears one pixel from screen
        calling with an Element instance (Player, Enemy...), or directly by coordinate

        """
        if isinstance(args[0], Element):
            y, x = args[0].y, args[0].x

        else:
            y, x = args[0:2]

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

    def render_all(self, reset_fog=False):
        """
        render all visible elements and updates fog area
        """
        ## Update Area Light
        self.area_light = set(surronding_area(self.player, 5, *self.screen_limits))


        if self.base.deployed:
            self.area_light = set(
                chain(
                    self.area_light,
                    surronding_area(self.base, 10, *self.screen_limits),
                )
            )

        if len(self.linterns) > 0:

            self.area_light = set(
                chain(
                    self.area_light,
                    chain.from_iterable(
                        surronding_area(l, 5, *self.screen_limits)
                        for l in self.linterns
                    ),
                )
            )
        if len(self.satelites) > 0:

            self.area_light = set(
                chain(
                    self.area_light,
                    chain.from_iterable(
                        surronding_area(s, 10, *self.screen_limits)
                        for s in self.satelites
                    ),
                )
            )

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

        for item in chain(
            self.mountains,
            self.buildings,
            self.satelites,
            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 self.area_light:
                self.render(item)

        if self.area.difference(self.area_light) != self.area_fog or reset_fog:
            # render fog bg if it has changed or forced to reset
            self.area_fog = self.area.difference(self.area_light)
            self.render_fog(self.area_fog)

    def render(self, element, *args, **kwargs):
        """
        render single element
        """

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

        c = element.color

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

        else:
            symbol = kwargs["symbol_overwrite"]

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

        if symbol_overwrite is 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: Element,
    distance: int,
    min_y: int,
    max_y: int,
    min_x: int,
    max_x: int,
    includes_self: bool = True,
) -> list:
    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: Element, area) -> bool:
    return {(obj.y, obj.x)} in area


def collision(objA: Element, objB: Element) -> bool:
    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
    kind: str = "Zombie"
    color: int = 5
    health: int = 2
    level: int = 1

    def move(self, new_y, new_x):
        self.y = new_y
        self.x = new_x


@dataclass
class Spawner(Element):
    symbol: str = "#"
    kind: str = "Spawner"
    health: int = 10
    level: int = 10
    color: int = 5

    def spawn(self):
        return Enemy(self.y, self.x)


@dataclass
class Fruit(Element):
    symbol: int = 4194409  # curses.ACS_LANTERN
    color: int = 3


@dataclass
class Base(Element):
    kind: str = "Base"
    deployed: bool = False
    visible: bool = True
    health: int = 100
    gold: int = 100
    symbol: int = 4194400  # curses.ACS_DIAMOND
    color: int = 7


@dataclass
class Satelite(Element):
    kind: str = "Satelite"
    visible: bool = True
    health: int = 10
    symbol: int = 4194400  # curses.ACS_DIAMOND
    color: int = 17


@dataclass
class Lintern(Element):
    visible: bool = True
    symbol: str = "@"
    color: int = 17


@dataclass
class Trap(Element):
    deployed: bool = False
    symbol: str = "%"


@dataclass
class Player(Element):
    kind: str = "Player"
    dir_y: int = 0
    dir_x: int = 0
    health: int = 100
    points: int = 0
    bombs: int = 2
    level: int = 1
    to_move: bool = False
    symbol: str = "*"
    color: int = 13
    visible: bool = True

    def move(self, new_y, new_x):
        self.dir_y = new_y - self.y
        self.dir_x = new_x - self.x

        self.y = new_y
        self.x = new_x


@dataclass
class Bomb(Element):
    symbol: str = "+"
    strength: int = 5
    timer: int = 2
    t0: float = field(default_factory=time.time)

    @property
    def area(self) -> list:
        s = self.strength
        return [
            (self.y + dy, self.x + dx)
            for dy in range(-s, s + 1)
            for dx in range(-s, s + 1)
            if int(
                math.sqrt((self.x - (self.x + dx)) ** 2 + (self.y - (self.y + dy))
 ** 2)
            )
            <= s
        ]

    @property
    def is_kaboom(self):
        """
        check if timer is over and returns True to handle bomb self destruction, o
r False otherwise
        """

        return time.time() - self.t0 > self.timer
```