```python
#! /usr/bin/python

class Component(object):
    def __init__(self, type, parent=None):
        self.type = type
        self.parent = parent

    def update(self, dt):
        pass

class ComponentSpatial(Component):
    def __init__(self, grid, position, parent=None):
        super(ComponentSpatial, self).__init__('spatial', parent)
        self.grid = grid
        self.pid = None
        self._put(position)

    def _put(self, point):
        pid = self.grid.put(point, self.parent)
        if (pid == None):
            return False
        if self.pid:
            self.grid.remove(self.grid.id_to_point(self.pid))
        self.pid = pid
        return True

    def getPosition(self):
        return self.grid.id_to_point(self.pid)

    def isAtPosition(self, point):
        return self.grid.point_to_id(point) == self.pid

    def move(self, direction):
        pass

class GameObject(object):
    def __init__(self):
        self.components = {}

    def __getattr__(self, name):
        return self.components[name.lower()]

    def addComponent(self, component):
        self.components[component.type] = component


#~ class Direction(object):
    #~ NEUTRAL = { 'xmod': 0, 'ymod': 0 }
    #~ NORTH = { 'xmod': 0, 'ymod': -1 }
    #~ EAST = { 'xmod': 1, 'ymod': 0 }
    #~ SOUTH = { 'xmod': 0, 'ymod': 1 }
    #~ WEST = { 'xmod': -1, 'ymod': 0 }

class Direction(object):

    directions = {}

    def __init__(self, name, xmod, ymod, deg):
        self.name = name
        self.xmod = xmod
        self.ymod = ymod
        self.deg = deg
        self.__class__.directions[deg] = self

    def __str__(self):
        return "%s [(%d %d) %d]" % (self.name, self.xmod, self.ymod, self.deg)

    @classmethod
    def values(cls):
        return cls.directions.values()


```

```
Direction.NEUTRAL = Direction("neutral", 0, 0, 0)
Direction.NORTH = Direction("north", 0, -1, 90)
Direction.EAST = Direction("east", 1, 0, 0)
Direction.SOUTH = Direction("south", 0, 1, -90)
Direction.WEST = Direction("west", -1, 0, 180)


class Grid(object):
    def __init__(self, width=5, height=4, origin=(0,0), scale=20):
        self.width = width
        self.height = height
        self.origin = origin
        self.scale = scale
        self.cells = [None for i in range(width*height)]

    def _clamp_x(self, x):
        "expects grid-internal value, clamps to value in (0, width-1)"
        if x < 0:
            return 0
        if x >= self.width:
            return self.width - 1
        return x

    def _clamp_y(self, y):
        "expects grid-internal value, clamps to value in (0, height-1)"
        if y < 0:
            return 0
        if y >= self.height:
            return self.height- 1
        return y

    # grosser quatsch:
    def _clamp_id(self, id):
        x = self._clamp_x(id % self.width)
        y = self._clamp_y(id / self.width)
        return x + y*self.width


    # def _id_to_xy: pass
    # def _xy_to_id: pass


    def point_to_id(self, point):
        x = round((point[0] - self.origin[0]) / self.scale)
        y = round((point[1] - self.origin[1]) / self.scale)
        x = self._clamp_x(x)
        y = self._clamp_y(y)
        id = int(x + y*self.width)
        return id


    def id_to_point(self, id):
        x = (id % self.width)
        y = (id / self.width)
        x = self._clamp_x(x) * self.scale + self.origin[0]
        y = self._clamp_y(y) * self.scale + self.origin[1]
        return (x,y)


    def get(self, point):
        return self.cells[self.point_to_id(point)]

    def put(self, point, gameobject):
        id = self.point_to_id(point)
        if (self.cells[id] != None):
            return None
        self.cells[id] = gameobject
        return id

    def remove(self, point):
        id = self.point_to_id(point)
        old = self.cells[id]
```

```python
        self.cells[id] = None
        return old

    def locationIsOccupied(self, point):
        return self.get(point) != None

    def transform_id_by_direction(self, id, direction):
        x = (id % width) + direction.xmod
        y = (id / width) + direction.ymod
        x = self._clamp_x(x)
        y = self._clamp_y(y)
        newid = x + y*width
        return newid

    #TODO CHECK! -- integrated into converting methods
    def calculateSquaredDistance(self, pointa, pointb):
        diff = self.id_to_point( self.point_to_id(pointb) - self.point_to_id(pointa) )
        return diff[0]*diff[0] + diff[1]*diff[1]

    def calculateBestDirection(self, orgPoint, destPoint):
        best_distance = float('inf')
        best_direction = Direction.NEUTRAL
        for dir in Direction.values():
            newx = orgPoint[0]+dir.xmod*self.scale
            newy = orgPoint[1]+dir.ymod*self.scale
            print (newx, newy)
            d = self.calculateSquaredDistance( (newx, newy), destPoint)
            print str(dir) + "=" + str(d)
            if d < best_distance:
                best_distance = d
                best_direction = dir
        return best_direction

    def put_damage(self, point, damage):
        pass


```