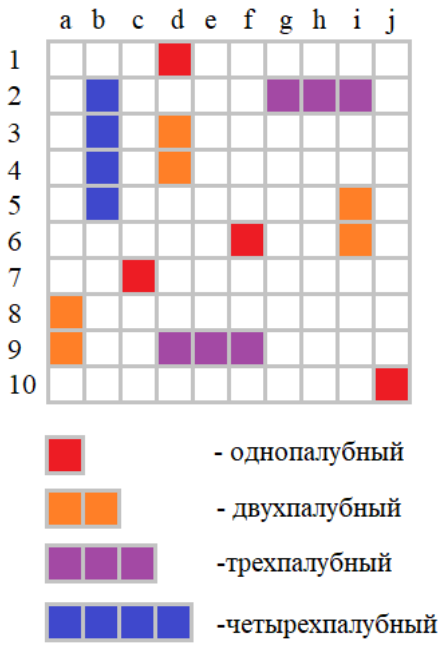
5.6 Посвящение в объектно-ориентированное программирование 1 из 1 шага пройден 15 из 15 баллов получено

Посвящение в ООП

Вы прошли серию испытаний и совершили множество подвигов, чтобы лицом к лицу столкнуться с настоящим вызовом, достойным лишь избранных! Для подтверждения своих знаний и навыков вам предлагается пройти этап посвящения в объектно-ориентированное программирование. И вот задание, которое выпало на вашу долю.

Руководство компании целыми днями не знает куда себя деть. Поэтому они решили дать задание своим программистам написать программу игры "Морской бой". Но эта игра будет немного отличаться от классической. Для тех, кто не знаком с этой древней, как мир, игрой, напомню ее краткое описание.

Каждый игрок у себя на бумаге рисует игровое поле 10 x 10 клеток и расставляет на нем десять кораблей: однопалубных - 4; двухпалубных - 3; трехпалубных - 2; четырехпалубный - 1.



Корабли расставляются случайным образом, но так, чтобы не выходили за пределы игрового поля и не соприкасались друг с другом (в том числе и по диагонали).

Затем, игроки по очереди называют клетки, куда производят выстрелы. И отмечают эти выстрелы на другом таком же поле в 10 x 10 клеток, которое представляет поле соперника. Соперник при этом должен честно отвечать: "промах", если ни один корабль не был задет и "попал", если произошло попадание. Выигрывает тот игрок, который первым поразит все корабли соперника.

Но это была игра из глубокого прошлого. Теперь же, в компьютерную эру, корабли на игровом поле могут перемещаться в направлении своей ориентации на одну клетку после каждого хода соперника, если в них не было ни одного попадания.

Итак, лично вам поручается сделать важный фрагмент этой игры - расстановку и управление кораблями в этой игре. А само задание звучит так.

Техническое задание

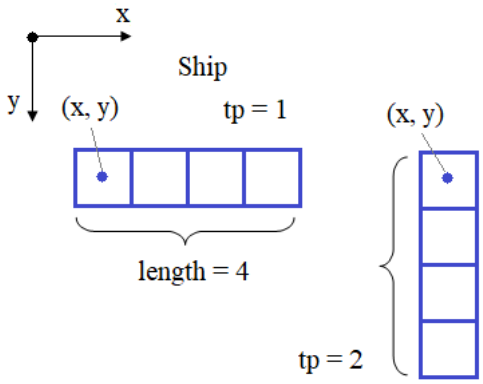
В программе необходимо объявить два класса:

Ship - для представления кораблей;
GamePole - для описания игрового поля.

Класс Ship

Класс Ship должен описывать корабли набором следующих параметров:

x, y - координаты начала расположения корабля (целые числа);
length - длина корабля (число палуб: целое значение: 1, 2, 3 или 4);
tp - ориентация корабля (1 - горизонтальная; 2 - вертикальная).



Объекты класса Ship должны создаваться командами:

```
ship = Ship(length)
ship = Ship(length, tp)
ship = Ship(length, tp, x, y)
```

По умолчанию (если не указывается) параметр tp = 1, а координаты x, y равны None.

В каждом объекте класса Ship должны формироваться следующие локальные атрибуты:

_x, _y - координаты корабля (целые значения в диапазоне [0; size), где size - размер игрового поля);
_length - длина корабля (число палуб);
_tp - ориентация корабля;
_is_move - возможно ли перемещение корабля (изначально равно True);
_cells - изначально список длиной length, состоящий из единиц (например, при length=3, _cells = [1, 1, 1]).

Список _cells будет сигнализировать о попадании соперником в какую-либо палубу корабля. Если стоит 1, то попадания не было, а если стоит значение 2, то произошло попадание в соответствующую палубу.

При попадании в корабль (хотя бы одну его палубу), флаг _is_move устанавливается в False и перемещение корабля по игровому полю прекращается.

В самом классе Ship должны быть реализованы следующие методы (конечно, возможны и другие, дополнительные):

set_start_coords(x, y) - установка начальных координат (запись значений в локальные атрибуты _x, _y);
get_start_coords() - получение начальных координат корабля в виде кортежа x, y;
move(go) - перемещение корабля в направлении его ориентации на go клеток (go = 1 - движение в одну сторону на клетку; go = -1 - движение в другую сторону на одну клетку); движение возможно только если флаг _is_move = True;
is_collide(ship) - проверка на столкновение с другим кораблем ship (столкновением считается, если другой корабль или пересекается с текущим или просто соприкасается, в том числе и по диагонали); метод возвращает True, если столкновение есть и False - в противном случае;
is_out_pole(size) - проверка на выход корабля за пределы игрового поля (size - размер игрового поля, обычно, size = 10); возвращается булево значение True, если корабль вышел из игрового поля и False - в противном случае;

С помощью магических методов __getitem__() и __setitem__() обеспечить доступ к коллекции _cells следующим образом:

```
value = ship[indx] # считывание значения из _cells по индексу indx (индекс отсчитывается от 0)
ship[indx] = value # запись нового значения в коллекцию _cells
```

Класс GamePole

Следующий класс GamePole должен обеспечивать работу с игровым полем. Объекты этого класса создаются командой:

```
pole = GamePole(size)
```

где size - размеры игрового поля (обычно, size = 10).

В каждом объекте этого класса должны формироваться локальные атрибуты:

_size - размер игрового поля (целое положительное число);
_ships - список из кораблей (объектов класса Ship); изначально пустой список.

В самом классе GamePole должны быть реализованы следующие методы (возможны и другие, дополнительные методы):

init() - начальная инициализация игрового поля; здесь создается список из кораблей (объектов класса Ship): однопалубных - 4; двухпалубных - 3; трехпалубных - 2; четырехпалубный - 1 (ориентация этих кораблей должна быть случайной).

Корабли формируются в коллекции _ships следующим образом: однопалубных - 4; двухпалубных - 3; трехпалубных - 2; четырехпалубный - 1. Ориентация этих кораблей должна быть случайной. Для этого можно воспользоваться функцией randint следующим образом:

```
[Ship(4, tp=randint(1, 2)), Ship(3, tp=randint(1, 2)), Ship(3, tp=randint(1, 2)), ...]
```

Начальные координаты x, y не расставленных кораблей равны None.

После этого, выполняется их расстановка на игровом поле со случайными координатами так, чтобы корабли не пересекались между собой.

get_ships() - возвращает коллекцию _ships;
move_ships() - перемещает каждый корабль из коллекции _ships на одну клетку (случайным образом вперед или назад) в направлении ориентации корабля; если перемещение в выбранную сторону невозможно (другой корабль или пределы игрового поля), то попытаться переместиться в противоположную сторону, иначе (если перемещения невозможны), оставаться на месте;
show() - отображение игрового поля в консоли (корабли должны отображаться значениями из коллекции _cells каждого корабля, вода - значением 0);

get_pole() - получение текущего игрового поля в виде двумерного (вложенного) кортежа размерами size x size элементов.

Пример отображения игрового поля:

```
0 0 1 0 1 1 1 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 1
0 0 0 0 1 0 1 0 0 1
0 0 0 0 0 0 1 0 0 0
1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0
```

Пример использования классов (эти строчки в программе не писать):

```
SIZE_GAME_POLE = 10

pole = GamePole(SIZE_GAME_POLE)
pole.init()
pole.show()

pole.move_ships()
print()
pole.show()
```

В программе требуется только объявить классы Ship и GamePole с соответствующим функционалом. На экран выводить ничего не нужно.

P.S. Для самых преданных поклонников программирования и ООП. Завершите эту программу, добавив еще один класс **SeaBattle** для управления игровым процессом в целом. Игра должна осуществляться между человеком и компьютером. Выстрелы со стороны компьютера можно реализовать случайным образом в свободные клетки. Сыграйте в эту игру и выиграйте у компьютера.

Чтобы решить это задание откройте https://stepik.org/lesson/727588/step/1