Задание.

Создайте набор классов, описывающих поле для игры в "Морской бой" с кораблями. Поле должно иметь квадратную форму, на нем можно размещать Корабли произвольного типа по горизонтали или вертикали. Корабль должен размещаться на Поле, у Корабля есть такие параметры, как Расположение, Длину, Здоровье, Пределы здоровья (минимум и максимум). Корабль можно разместить на поле, удалить с поля, переместить вперед или назад, повернуть на угол относительно любой из его клеток на угол 90, 180, 270 градусов. Корабли можно повреждать, уменьшая Здоровье (при повреждении ниже минимального уровня Корабль удаляется с поля), чинить (не выше максимального уровня Здоровья).

Решение.

```
################################################################################
###
# add b
################################################################################
###
class Array
  def add b
    (zip b).map { |el| el[0] + el[1] }
  end
end
################################################################################
###

################################################################################
###
# Класс Field
################################################################################
###
class Field
  FieldSize = 10

  def initialize
    @field = Array.new( FieldSize ) { Array.new( FieldSize ) }
  end

  # Размер поля
  def self.size
    FieldSize
  end
```

```
  # Получение по флагу горизонтальности hor
  # направления смещения клеток корабля от первой клетки
  def self.hor_vector h
    [ h ? 1 : 0, h ? 0 : 1 ]
  end

  # Размещение Корабля
  def set!( n, x, y, hor, ship )
    vect = Field.hor_vector hor
    n.times {
      @field[ x ][ y ] = ship
      x += vect[ 0 ]
      y += vect[ 1 ]
    }
  end

  # Строковое представление
  def to_s
    space_char = " "
    line0 = "+"
    FieldSize.times { line0 += '-' }
    line0 += "+\n"
    res = line0
    FieldSize.times { |x|
      line1 = "|"
      FieldSize.times { |y|
        if @field[ x ][ y ].nil? then line1 += space_char else line1 +=
@field[ x ][ y ].to_s end
      }
      line1 += "|\n"
      res += line1
    }
    res + line0
  end

  # Вывод
  def print_field
    puts to_s
  end

  # Помещается ли клетка (x, y) на игровое поле?
  # "Статический" метод класса Field
  def self.is_cell_valid( x, y )
    x.between?( 0, FieldSize-1 ) && y.between?( 0, FieldSize-1 )
  end

  # В заданной клетке нет ничего (кроме, возможно, корабля shift;
  # или клетка вне поля)
  def nothing_but_ship( x, y, ship )
    ( not Field.is_cell_valid( x, y ) ) ||
    @field[ x ][ y ].nil? ||
    ( @field[ x ][ y ] == ship )
  end
```

```
  # В окрестностях (смежных клетках) заданной клетки
  # нет ничего в том же смысле, что и в предыдущей функции
  def nothing_but_ship_around( x, y, ship )
    all_good = true
    [ -1, 0, 1 ].each { |dx|
      [ -1, 0, 1 ].each { |dy|
        all_good &= nothing_but_ship( x+dx, y+dy, ship )
      }
    }
    all_good
  end

  def free_space?( n, x, y, hor, ship )
    all_is_good = true
    vect = Field.hor_vector hor
    n.times {
      all_is_good &= Field.is_cell_valid( x, y ) && nothing_but_ship_around(
x, y, ship )
      x += vect[ 0 ]
      y += vect[ 1 ]
    }
    all_is_good
  end

end
# конец описания класса Field
##############################################################################
###


##############################################################################
###
# Класс Ship
##############################################################################
###
class Ship
  attr_reader :len
  attr_reader :coord
  def initialize( field, len )
    @myfield = field
    @len = len
    @maxhealth = 100 * @len
    @minhealth = 30 * @len
    @health = @maxhealth
  end

  # Строкове представление
  def to_s
    "X"
  end

  # Удаление поля
```

```
  def clear
    @myfield.set!( @len, @coord[ 0 ], @coord[ 1 ], @hor, nil )
  end

  # Размещение на поле
  def set!( x, y, hor )
    hor_vector = Field.hor_vector hor
    if @myfield.free_space?( @len, x, y, hor, self )
      if not @coord.nil?
        clear
      end
      @myfield.set!( @len, x, y, hor, self )
      @hor = hor
      @coord = [ x, y, x + hor_vector[0] * (@len-1), y + hor_vector[1] *
(@len-1) ]
      true
    else
      false
    end
  end

  # Уничтожение Корабля
  def kill
    if not @coord.nil?
      clear
      @coord = nil
    end
  end

  # Нанесение повреждения
  def explode
    @health -= 70
    if @health <= @minhealth
      kill
      @len
    else
      nil
    end
  end

  # Починка
  def cure
    @health = [ @health + 30, @maxhealth ].min
  end

  # Текущее Здоровье
  def health
    ( 100.0 * @health.to_f / @maxhealth ).round( 2 )
  end

  # Перемещение вперед или назад
  def move( forward )
    vect = Field.hor_vector @hor
```

```
    vect = forward ? vect : [ -vect[ 0 ], -vect[ 1 ] ]
    if @myfield.free_space?( @len, @coord[ 0 ] + vect[ 0 ], @coord[ 1 ] +
vect[ 1 ],
                             @hor, self )
      set!( @coord[ 0 ] + vect[ 0 ], @coord[ 1 ] + vect[ 1 ], @hor )
    else
      false
    end
  end

  # Вычисление косинуса и синуса угла, соответствующего числу k (1..3)
  def get_cos_sin k
    cos_k = ( k == 1 ) ? 0 : ( ( k == 2 ) ? -1 : 0 )
    sin_k = ( k == 1 ) ? 1 : ( ( k == 2 ) ? 0 : -1 )
    [ cos_k, sin_k ]
  end

  # Флаг hor после поворота, соответствующего числу k (1..3)
  def get_hor( hor, k )
    new_hor = ( k == 2 ) ? hor : ! hor
  end

  # Поворот клетки (x,y) вокруг клетки (x_c, y_c) на угол с косинусом
  # и синусом из аргумента cos_sin (это пара [cos_k, sin_k])
  # Возврат пары [ new_x, new_y ]
  def rotate_cell( x_c, y_c, x, y, cos_sin )
    new_x = x_c + (x-x_c) * cos_sin[ 0 ] - (y-y_c) * cos_sin[ 1 ]
    new_y = y_c + (x-x_c) * cos_sin[ 1 ] + (y-y_c) * cos_sin[ 0 ]
    return [ new_x, new_y ]
  end

  def rotate( n, k )
    if ( ! n.between?( 1, @len ) || ! k.between?( 1, 3 ) ) then return false
end
    vect = Field.hor_vector @hor
    x_c = @coord[ 0 ] + vect[ 0 ] * ( n-1 )
    y_c = @coord[ 1 ] + vect[ 1 ] * ( n-1 )
    cos_sin = get_cos_sin k
    xs = []; ys = []
    @len.times { |index|
      new_x_y = rotate_cell( x_c, y_c,
                             @coord[ 0 ] + vect[ 0 ] * index,
                             @coord[ 1 ] + vect[ 1 ] * index,
                             cos_sin )
      xs.push( new_x_y[ 0 ] )
      ys.push( new_x_y[ 1 ] )
    }
    new_coord = [ xs.min, ys.min ]
    new_hor = get_hor( @hor, k )
    if @myfield.free_space?( @len, new_coord[ 0 ], new_coord[ 1 ], new_hor,
self )
      set!( new_coord[ 0 ], new_coord[ 1 ], new_hor )
    else
```

```
      false
    end
  end

end
# конец описания класса Ship
#############################################################################
###
```