Random Walk Assignment
Common Programming Errors

Some Basics

- Follow the input specification
  - Use the given I/O specifications
- Submit .rb files only
  - Folders, Rails files, IDE project files etc. are not .rb files
- Provide a comment on submission
  - If it works, tell me.
- Now we look at some common programming errors...

Don't print from inside a class

```
class Die
   ....
   def print_stats
      print "{ "
      end
      @sides_stats.each do |key,value
         print "#{key}=>#{value} "
      end
      puts "}"
   end
   ...
end # end of Die class
```

Prefer a `to_s` method that returns a string

Why is this bad?

Avoid Duplication

```
if @dice.die_face == :East
   @location.x += 1
   @path_taken.push(self.location_of_kangaroo)
end
if @dice.die_face == :West
   @location.x -= 1
   @path_taken.push(self.location_of_kangaroo)
end
if @dice.die_face == :North
   @location.y += 1
   @path_taken.push(self.location_of_kangaroo)
end
if @dice.die_face == :South
   @location.y -= 1
   @path_taken.push(self.location_of_kangaroo)
end
```

How to improve this code?

Write clear methods; use clear method names

A method in the Point class:

```
def point a, b                  #"a" and "b" are arguments
  @p = Point.new @x, @y
  @x += a                       #Adds argument "a" to "@x"
  @y += b                       #Adds argument "b" to "@y"
  @q = Point.new @x, @y
  puts "#{@p} -> #{@q}"         #Prints out both points
  if beyond_dimensions?         #Calls method below
    puts "^invalid move^"       #Prints out
    @p
  else
    @q
  end
end
```

- method too complicated
- bad name
- uninformative comments

How to improve this?

Make lower level classes do the work

A method in the Kangaroo class:

```
def hop!
    direction = @die.throw
    if direction == :EAST
       if @point.x==@gridsize
           hop!
       else
           @point.east
           @die.stats[:EAST]+=1
           @die.stats[:TOTAL]+=1
       end
    elsif direction == :WEST
       if @point.x==0
           hop!
       else
           @point.west
           @die.stats[:WEST]+=1
           @die.stats[:TOTAL]+=1
       end
```

Similar code for NORTH and SOUTH omitted...

- too much detail!
- switch statement better
- avoid recursion

See sample solution on next slide

Similar method from sample solution

```
# Hop to new location inside the grid
def hop!
  @location.move(@die.throw)
  while @grid.lies_outside?(@location) do
    @location.undo
    @location.move(@die.throw)
  end
  @num_of_hops += 1
  @locations_visited.push @location.clone
end
```

The Die and Grid classes do the work. The algorithm in the method is clearer.

Another example

Part of the main script:

```
# Print die stats
total_throws = 0
rw.stats.each do |direction, num_of_throws|
  total_throws += num_of_throws
end
puts "Total hops the kangaroo took: #{rw.num_of_hops}"
puts "Die stats are as follows:"
[:NORTH, :SOUTH, :EAST, :WEST].each do |direction|
  percentage = ...
  puts "#{direction}: #{percentage}%"
end
```

First 4 lines better moved to a more suitable class.

Minimise Commenting (yes, they lied to you)

```
while (!final_location?)
direction = @die.throw
   x = @skippy.location.x
   y = @skippy.location.y
   ...
end
```

Nice, clear code...

```
# while loop to run until the final_location?
# method returns true meaning the kangaroo has
# reached its destination
while (!final_location?)
# get a random direction by calling the throw method
# of the die class
direction = @die.throw
# store the current x and y points of the kangaroo's
# location in local variables
x = @skippy.location.x
y = @skippy.location.y
```

same code ruined by comments

Avoid long methods

```
# Move the kangaroo in direction given by dir
 def hop! die,dim
 dir = die.throw
 @dim = dim
 @xval = @pos.x1
 @yval = @pos.y1

 # Checks to see if kangaroo object is at a boundary,
 # given by the dim argument.
 # If true it checks to see what boundary, movement is restricted.
 # If false it is allowed move in any direction
 if at_boundary? == true
   if @yval == 0 && @xval==0
     case dir
     when ":North" then @pos.x1 += 1
     when ":East" then @pos.y1 += 1
     end
   elsif @yval ==0
     @hops += 1
     case dir
     when ":North" then @pos.x1 += 1
     when ":East" then @pos.y1 += 1
     when ":South" then @pos.x1 -= 1
     end
   elsif @xval == 0
     @hops += 1
     case dir
     when ":North" then @pos.x1 += 1
     when ":East" then @pos.y1 += 1
     when ":West" then @pos.y1 -=1
     end
   elsif @yval == (@dim-1)
     @hops += 1
     case dir
     when ":North" then @pos.x1 += 1
     when ":South" then @pos.x1 -= 1
     when ":West" then @pos.y1 -=1
     end
   elsif @xval == (@dim-1)
     @hops += 1
     case dir
     when ":East" then @pos.y1 += 1
     when ":South" then @pos.x1 -= 1
     when ":West" then @pos.y1 -=1
     end
   end
 else
   @hops += 1
   case dir
   when ":North" then @pos.x1 += 1
   when ":East" then @pos.y1 += 1
   when ":South" then @pos.x1 -= 1
   when ":West" then @pos.y1 -=1
   end
 end
end
```

- Rewrite long methods
- Consider splitting

Poor layout shows you don't care

```
class Die

  # attr_accessor :die

  $die = { :NORTH=>0, :SOUTH=>0,:EAST=>0,:WEST=>0 }

  def throws

    @num =  rand(4)

#    print " rolled a: #{@num} == "

    ret_num = @num

       @num = case
           when @num == 0

#       puts "\\n @num is: "; puts $@num
#       print :NORTH
            $die[:NORTH] +=1

           when @num == 1
#             print :SOUTH
              $die[:SOUTH] += 1

           when @num == 2
#             print :EAST
             $die[:EAST] += 1
       when @num == 3
#             print :WEST
              $die[:WEST] += 1
       end

    return ret_num

  end

end
```

Code that looks well shows you care about:
- the code
- your colleagues

Don't define instance variables you don't need

```
class RandomWalk
  def initialize dimension
    @grid = Grid.new dimension
    @skippy = Kangaroo.new @grid
  end

  def start
    while !@skippy.at_home?
      @skippy.hop!
    end
  end

  def stats
    @skippy.die_stats
  end

  def num_of_hops
    @skippy.num_of_hops
  end

  def locations_visited
    @skippy.locations_visited
  end
end
```

@grid should be a normal variable

Design of Sample Solution

Discussed in lecture.