Building Java Programs

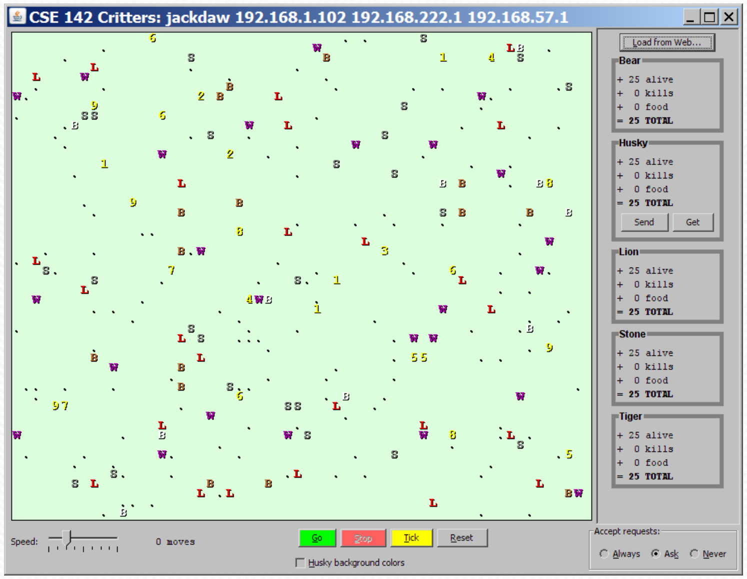
Homework 8: Critters

reading: Critters Assignment Spec

Critters

- A simulation world with animal objects with behavior:

`eat`	eating food
`fight`	animal fighting
`getColor`	color to display
`getMove`	movement
`toString`	letter to display


CSE 142 Critters: jackdaw 192.168.1.102 192.168.222.1 192.168.57.1

```
Bear
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

Husky
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

Lion
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

Stone
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

Tiger
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

0 moves
```

Accept requests: Always [✓ Ask] Never

- You must implement:
  - `Ant`
  - `Bird`
  - `Hippo`
  - `Vulture`
  - `Husky` (creative)

A `Critter` subclass

```
public class name extends Critter {
    ...
}
```

- `extends Critter` tells the simulator your class is a critter
  - an example of *inheritance*
- Write some/all 5 methods to give your animals behavior.

How the simulator works

- When you press "Go", the simulator enters a loop:
  - move each animal once (`getMove`), in random order
  - if the animal has moved onto an occupied square, `fight`!
  - if the animal has moved onto food, ask it if it wants to `eat`

- Key concept: The simulator is in control, NOT your animal.
  - Example: `getMove` can return only one move at a time. `getMove` can't use loops to return a sequence of moves.
    - It wouldn't be fair to let one animal make many moves in one turn!
  - Your animal must keep state (as fields) so that it can make a single move, and know what moves to make later.

Critter exercise: `Cougar`

- Write a critter class `Cougar` (the dumbest of all animals):

Method	**Behavior**
constructor	`public Cougar()`
`eat`	Always eats.
`fight`	Always pounces.
`getColor`	Blue if the `Cougar` has never fought; red if he has.
`getMove`	Walks west until he finds food; then walks east until he finds food; then goes west and repeats.
`toString`	`"C"`

Ideas for state

- You must not only have the right state, but update that state properly when relevant actions occur.

- Counting is helpful:
  - How many total moves has this animal made?
  - How many times has it eaten? Fought?

- Remembering recent actions in fields is helpful:
  - Which direction did the animal move last?
    - How many times has it moved that way?
  - Did the animal eat the last time it was asked?
  - How many steps has the animal taken since last eating?
  - How many fights has the animal been in since last eating?

Keeping state

- How can a critter move west until it finds food?

```
~~public Direction getMove() {~~
~~    while (animal has not eaten) {~~
~~        return Direction.EAST;~~
~~    }~~
~~    while (animal has not eaten a second time) {~~
~~        return Direction.EAST;~~
~~    }~~
~~}~~
```

```
private int moves;    // total moves made by this Critter

public Direction getMove() {
    moves++;
    if (moves % 4 == 1 || moves % 4 == 2) {
        return Direction.WEST;
    } else {
        return Direction.EAST;
    }
}
```

Cougar solution

```
import java.awt.*;  // for Color

public class Cougar extends Critter {
    private boolean west;
    private boolean fought;

    public Cougar() {
        west = true;
        fought = false;
    }

    public boolean eat() {
        west = !west;
        return true;
    }

    public Attack fight(String opponent) {
        fought = true;
        return Attack.POUNCE;
    }

    ...
```

Cougar solution

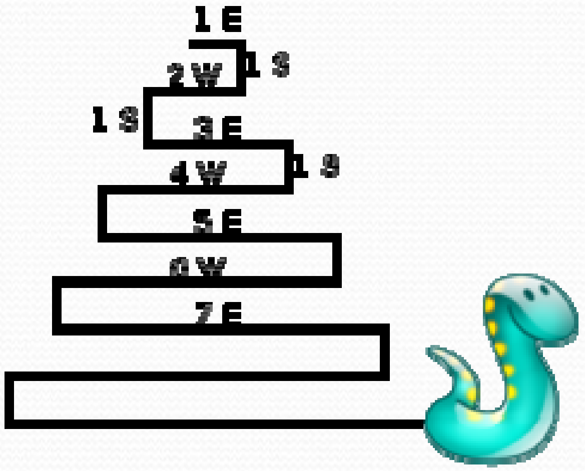
```
    ...

    public Color getColor() {
        if (fought) {
            return Color.RED;
        } else {
            return Color.BLUE;
        }
    }

    public Direction getMove() {
        if (west) {
            return Direction.WEST;
        } else {
            return Direction.EAST;
        }
    }

    public String toString() {
        return "C";
    }
}
```

Testing critters

- Focus on one specific critter of one specific type
  - Only spawn 1 of each animal, for debugging
- Make sure your fields update properly
  - Use `println` statements to see field values
- Look at the behavior one step at a time
  - Use "Tick" rather than "Go"

Critter exercise: `Snake`

Method	Behavior
constructor	`public Snake()`
`eat`	Never eats
`fight`	always forfeits
`getColor`	black
`getMove`	1 E, 1 S; **2** W, 1 S; **3** E, 1 S; **4** W, 1 S; **5** E, ...
`toString`	`"S"`



Determining necessary fields

- Information required to decide what move to make?
  - Direction to go in
  - Length of current cycle
  - Number of moves made in current cycle

- Remembering things you've done in the past:
  - an `int` counter?
  - a `boolean` flag?

Snake solution

```
import java.awt.*;     // for Color

public class Snake extends Critter {
    private int length;    // # steps in current horizontal cycle
    private int step;      // # of cycle's steps already taken

    public Snake() {
        length = 1;
        step = 0;
    }

    public Direction getMove() {
        step++;
        if (step > length) {    // cycle was just completed
            length++;
            step = 0;
            return Direction.SOUTH;
        } else if (length % 2 == 1) {
            return Direction.EAST;
        } else {
            return Direction.WEST;
        }
    }

    public String toString() {
        return "S";
    }
}
```