Building Java Programs

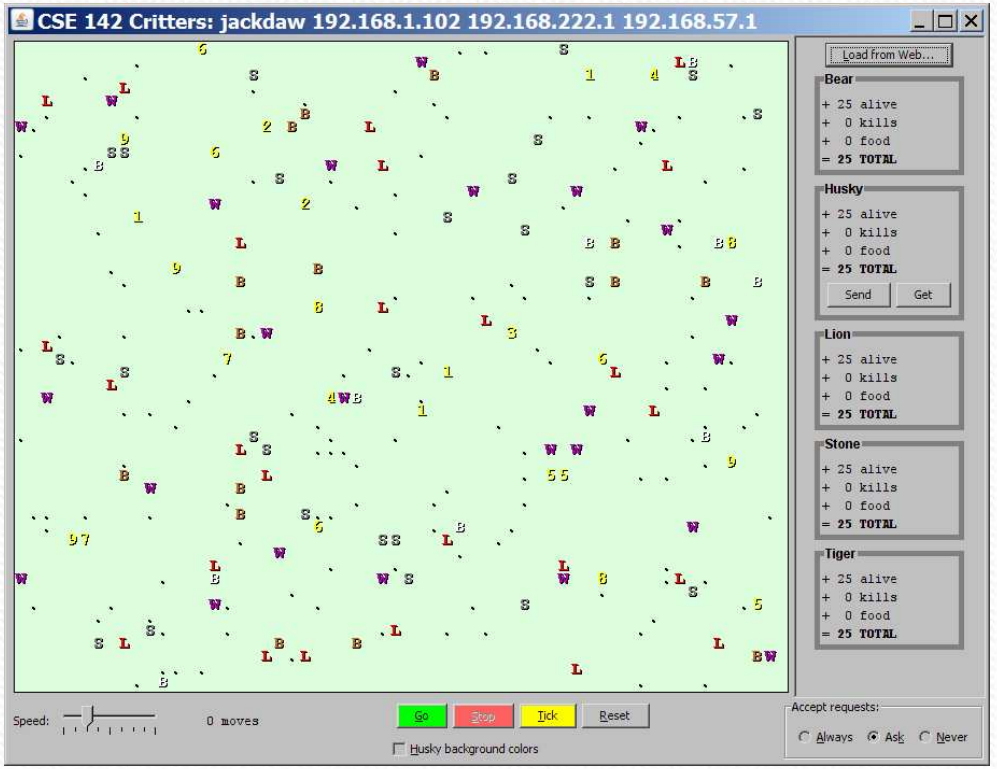
Chapter 9
Lecture 9-x: Critters

reading: HW9 Spec

Critters

- A simulation world with animal objects with behavior:
  - `getMove` movement
  - `eat` eating food
  - `fight` animal fighting
  - `toString` letter to display
  - `getColor` color to display

- You must implement:
  - `Bear`
  - `Lion`
  - `Tiger`
  - `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.

  - Your animal must keep state (as fields) so that it can make a single move, and know what moves to make later.

Critter exercise

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

Method	Behavior
`eat`	Always eats.
`fight`	Always pounces.
`getColor`	Blue if the `Cougar` has never fought; red if he has.
`getMove`	The drunk `Cougar` staggers left 2, right 2, repeats.
`toString`	Always returns `"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 left 2, right 2, and repeat?

```
public Direction getMove() {
    for (int i = 1; i <= 2; i++) {
        return Direction.WEST;
    }
    for (int i = 1; i <= 2; i++) {
        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

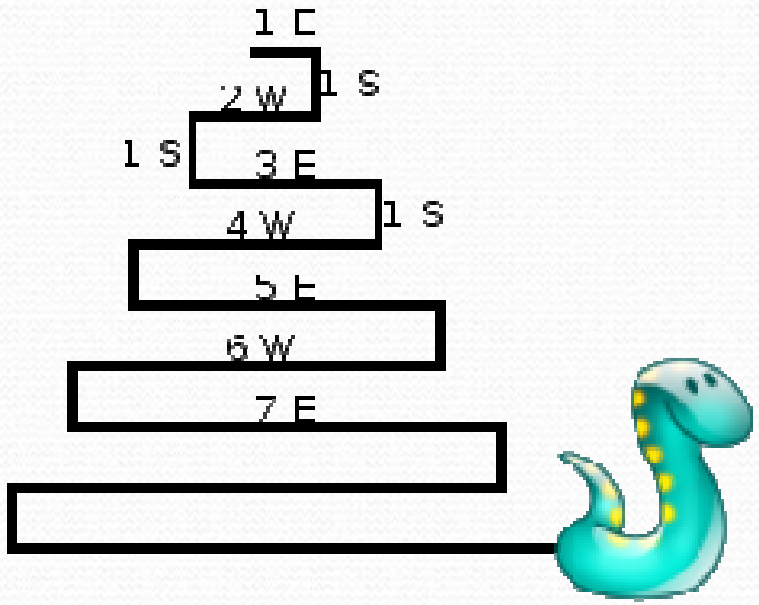
```
public class Cougar extends Critter {
    private int moves;
    private boolean fought;

    public Cougar() {
        moves = 0;
        fought = false;
    }
    public boolean eat() {
        return true;
    }
    public Attack fight() {
        fought = true;
        return Attack.POUNCE;
    }
    public Color getColor() {
        if (fought) {
            return Color.RED;
        } else {
            return Color.BLUE;
        }
    }
    public Direction getMove() {
        moves++;
        if (moves % 4 == 1 || moves % 4 == 2) {
            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 Critter type
- 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"

A complex Critter: Snake

Method	Behavior
`eat`	Never eats
`fight`	Randomly choose to roar or pounce
`getColor`	*(red=20, green=50, blue=128)*
`getMove`	1 E, 1 S; **2** W, 1 S; **3** E, 1 S; **4** W, 1 S; **5** E, ...
`toString`	Always returns "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
- Information required to decide how to fight?
  - A `Random` object

Snake solution

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

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

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

    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;
        }
    }
    ...
```

Snake solution 2

```
    ...

    public Attack fight(String opponent) {
        int attack = rand.nextInt(2);
        if (attack == 0) {
            return Attack.POUNCE;
        } else {
            return Attack.ROAR;
        }
    }

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

    public Color getColor() {
        return new Color(20, 50, 128);
    }

    // We don't need to write an eat method;
    // We can just keep the default eat behavior (returning false)
}
```