ITWM5113 – SOFTWARE DESIGN AND DEVELOPMENT

Project – Animal Kingdom

Lecturer Name: DR. SIMON LAU BOUNG YEW

Prepared By:

STUDENT NAME	**MATRIC NO.**
Patricia A/P Sitinathan	**MC210914598**

Abstract

In this environment, everything is a "critter," where Critter is the super class with predefined default behaviour. Five classes—Bear, Tiger, WhiteTiger, Giant, and NinjaCat—each corresponding to a different kind of animal, will be assigned to it. Every class that we listed should be a subclass of Critter. During testing, we will see that CritterMain has lines of code that look like this: / frame.add (30, Tiger.class); as we finish writing the various classes we have been requested to develop, we should uncomment these lines of code. The simulation will thereafter feature creatures of that kind. We can observe animals moving across the world because the simulator gives us excellent visual feedback about where they are. However, it provides limited information regarding the direction that animals are looking. This is simpler to see using the "debug" button on the simulator. We critters will appear as arrow characters that point in the direction they are facing when we ask for debug mode. As the simulation progresses, the simulator also displays the "step" number (initially displaying a 0).

Program workflow and logics

We assume that all classes should be subclasses of Critter. Each critter is questioned about three different pieces of information during each simulation round:

- How should it to behave?
- What colour is that?
- What string best describes that animal?

Three methods found in each Critter class offer these three bits of data. We will be in charge of overriding these procedures and establishing the proper behaviour for them.

The Behaviour of Our Five Classes:

1. Bear Constructor public Bear(boolean polar) getColor Color.WHITE for a polar bear (when polar is true), Color.BLACK otherwise (when polar is false) toString Should alternate on each different move between a slash character (/) and a backslash character () starting with a slash. getMove always infect if an enemy is in front, otherwise hop if possible, otherwise turn left.

2. Tiger Constructor public Tiger() getColor Randomly picks one of three colors (Color.RED, Color.GREEN, Color.BLUE) and uses that color for three moves, then randomly picks one of those colors again for the next three moves, then randomly picks another one of those colors for the next three moves, and so on. toString "TGR" getMove always infect if an enemy is in front, otherwise if a wall is in front or to the right, then turn left, otherwise if a fellow Tiger is in front, then turn right, otherwise hop.

3. WhiteTiger Constructor public WhiteTiger() getColor Always Color.WHITE. toString "tgr" if it hasn’t infected another Critter yet, “TGR” if it has infected. getMove Same as a Tiger. Note: you’ll have to override this method to figure out if it has infected another Critter.

4. Giant Constructor public Giant() getColor Color.GRAY toString "fee" for 6 moves, then "fie" for 6 moves, then "foe" for 6 moves, then "fum" for 6 moves, then repeat. getMove always infect if an enemy is in front, otherwise hop if possible, otherwise turn right.

5. NinjaCat Constructor public NinjaCat() getColor You decide toString You decide getMove You decide.

Results

Bear.java

```java
package me.patricia.forest;

import java.awt.*;

public class Bear extends Critter {
    private boolean polar;
    private int moves;

    public Bear(boolean polar) {
        this.polar = polar;
        getColor();
    }

    public Color getColor() {

        if (this.polar) {
            return Color.WHITE;
        } else {
            return Color.BLACK;
        }
    }

    public String toString() {

        if (moves % 2 == 0) {
            return "/";
        } else {
            return "\\";
        }

    }

    public Action getMove(CritterInfo info) {

        moves++;
        if (info.getFront() == Neighbor.OTHER) {
            return Action.INFECT;
        } else if (info.getFront() == Neighbor.EMPTY) {
            return Action.HOP;
        } else {
            return super.getMove(info);
        }
    }
}
```

Critter.java

```java
package me.patricia.forest;

import java.awt.*;

public class Critter {
    public static enum Neighbor {
        WALL, EMPTY, SAME, OTHER
    };

    public static enum Action {
        HOP, LEFT, RIGHT, INFECT
    };

    public static enum Direction {
        NORTH, SOUTH, EAST, WEST
    };

    public Action getMove(CritterInfo info) {
        return Action.LEFT;
    }

    public Color getColor() {
        return Color.BLACK;
    }

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

    public final boolean equals(Object other) {
        return this == other;
    }
}
```

CritterFrame.java

```
package me.patricia.forest;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.util.*;

public class CritterFrame extends JFrame {
    private CritterModel myModel;
    private CritterPanel myPicture;
    private javax.swing.Timer myTimer;
    private JButton[] counts;
    private JButton countButton;
    private boolean started;
    private static boolean created;

    public CritterFrame(int width, int height) {

        if (created)
            throw new RuntimeException("Only one world allowed");
        created = true;

        setTitle("CSE142 critter simulation");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        myModel = new CritterModel(width, height);

        myPicture = new CritterPanel(myModel);
        add(myPicture, BorderLayout.CENTER);

        addTimer();

        constructSouth();

        started = false;
    }

    private void constructSouth() {

        JPanel p = new JPanel();

        final JSlider slider = new JSlider();
        slider.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent e) {
                double ratio = 1000.0 / (1 + Math.pow(slider.getValue(),
0.3));
                myTimer.setDelay((int) (ratio - 180));
```

```java
            }
        });
        slider.setValue(20);
        p.add(new JLabel("slow"));
        p.add(slider);
        p.add(new JLabel("fast"));

        JButton b1 = new JButton("start");
        b1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                myTimer.start();
            }
        });
        p.add(b1);
        JButton b2 = new JButton("stop");
        b2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                myTimer.stop();
            }
        });
        p.add(b2);
        JButton b3 = new JButton("step");
        b3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                doOneStep();
            }
        });
        p.add(b3);

        JButton b4 = new JButton("debug");
        b4.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                myModel.toggleDebug();
                myPicture.repaint();
            }
        });
        p.add(b4);

        JButton b5 = new JButton("next 100");
        b5.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                multistep(100);
            }
        });
        p.add(b5);

        add(p, BorderLayout.SOUTH);
    }
    public void start() {

        if (started) {
            return;
        }

        if (myModel.getCounts().isEmpty()) {
            System.out.println("nothing to simulate--no critters");
            return;
        }
        started = true;
        addClassCounts();
        myModel.updateColorString();
        pack();
        setVisible(true);
    }

    private void addClassCounts() {
        Set<Map.Entry<String, Integer>> entries = myModel.getCounts();
        JPanel p = new JPanel(new GridLayout(entries.size() + 1, 1));
counts = new JButton[entries.size()];
        for (int i = 0; i < counts.length; i++) {
            counts[i] = new JButton();
            p.add(counts[i]);
        }

        countButton = new JButton();
        countButton.setForeground(Color.BLUE);
        p.add(countButton);

        add(p, BorderLayout.EAST);
        setCounts();
    }

    private void setCounts() {
        int i = 0;
        int max = 0;
        int maxI = 0;
        for (Map.Entry<String, Integer> entry : myModel.getCounts()) {
            String s = String.format("%s =%4d", entry.getKey(),
                    (int) entry.getValue());
            counts[i].setText(s);
            counts[i].setForeground(Color.BLACK);
            if (entry.getValue() > max) {
                max = entry.getValue();
                maxI = i;
            }
            i++;
        }
        counts[maxI].setForeground(Color.RED);
        String s = String.format("step =%5d", myModel.getSimulationCount());
        countButton.setText(s);
    }

    public void add(int number, Class<? extends Critter> c) {

        if (started) {
            return;
        }

        started = true;
        myModel.add(number, c);
        started = false;
    }

    private void addTimer() {
        ActionListener updater = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                doOneStep();
            }
        };
        myTimer = new javax.swing.Timer(0, updater);
        myTimer.setCoalesce(true);
    }

    private void doOneStep() {
        myModel.update();
        setCounts();
        myPicture.repaint();
    }

    private void multistep(int n) {
        myTimer.stop();
        do {
            myModel.update();
        } while (myModel.getSimulationCount() % n != 0);
        setCounts();
        myPicture.repaint();
    }
}
```

CritterInfo.java

```java
package me.patricia.forest;

public interface CritterInfo {
    public Critter.Neighbor getFront();

    public Critter.Neighbor getBack();

    public Critter.Neighbor getLeft();

    public Critter.Neighbor getRight();

    public Critter.Direction getDirection();

    public boolean frontThreat();

    public boolean backThreat();

    public boolean leftThreat();

    public boolean rightThreat();
}
```

CritterMain.Java

```java
package me.patricia.forest;

public class CritterMain {
    public static void main(String[] args) {
        CritterFrame frame = new CritterFrame(60, 40);

        // frame.add(30, Bear.class);
        frame.add(30, Tiger.class);
        frame.add(30, WhiteTiger.class);
        frame.add(30, Giant.class);
        frame.add(30, NinjaCat.class);
        frame.add(30, FlyTrap.class);
        frame.add(30, Food.class);

        frame.start();
    }
}
```

CritterModel.Java

```java
package me.patricia.forest;

import java.util.*;
import java.awt.Point;
import java.awt.Color;
import java.lang.reflect.*;

public class CritterModel {

    public static final double HOP_ADVANTAGE = 0.2;

    private int height;
    private int width;
    private Critter[][] grid;
    private Map<Critter, PrivateData> info;
    private SortedMap<String, Integer> critterCount;
    private boolean debugView;
    private int simulationCount;
    private static boolean created;

    public CritterModel(int width, int height) {

        if (created)
            throw new RuntimeException("Only one world allowed");
        created = true;

        this.width = width;
        this.height = height;
        grid = new Critter[width][height];
        info = new HashMap<Critter, PrivateData>();
        critterCount = new TreeMap<String, Integer>();
        this.debugView = false;
    }

    public Iterator<Critter> iterator() {
        return info.keySet().iterator();
    }

    public Point getPoint(Critter c) {
        return info.get(c).p;
    }

    public Color getColor(Critter c) {
        return info.get(c).color;
    }

    public String getString(Critter c) {
        return info.get(c).string;
    }

    public void add(int number, Class<? extends Critter> critter) {
        Random r = new Random();
        Critter.Direction[] directions = Critter.Direction.values();
        if (info.size() + number > width * height)
            throw new RuntimeException("adding too many critters");
        for (int i = 0; i < number; i++) {
            Critter next;
            try {
                next = makeCritter(critter);
            } catch (IllegalArgumentException e) {
                System.out.println("ERROR: " + critter + " does not have" +
                        " the appropriate constructor.");
                System.exit(1);
                return;
            } catch (Exception e) {
                System.out.println("ERROR: " + critter + " threw an " +
                        " exception in its constructor.");
                System.exit(1);
                return;
            }
            int x, y;
            do {
                x = r.nextInt(width);
                y = r.nextInt(height);
            } while (grid[x][y] != null);
            grid[x][y] = next;

            Critter.Direction d = directions[r.nextInt(directions.length)];
            info.put(next, new PrivateData(new Point(x, y), d));
        }
        String name = critter.getName();
        if (!critterCount.containsKey(name))
            critterCount.put(name, number);
        else
            critterCount.put(name, critterCount.get(name) + number);
    }

    private Critter makeCritter(Class critter) throws Exception {
        Constructor c = critter.getConstructors()[0];
        if (critter.toString().equals("class Bear")) {

            boolean b = Math.random() < 0.5;
            return (Critter) c.newInstance(new Object[] { b });
        } else {
            return (Critter) c.newInstance();
        }
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public String getAppearance(Critter c) {

        if (!debugView)
            return info.get(c).string;
        else {
            PrivateData data = info.get(c);
            if (data.direction == Critter.Direction.NORTH)
                return "^";
            else if (data.direction == Critter.Direction.SOUTH)
                return "v";
            else if (data.direction == Critter.Direction.EAST)
                return ">";
            else
                return "<";
        }
    }

    public void toggleDebug() {
        this.debugView = !this.debugView;
    }

    private boolean inBounds(int x, int y) {
        return (x >= 0 && x < width && y >= 0 && y < height);
    }

    private boolean inBounds(Point p) {
        return inBounds(p.x, p.y);
    }

    private Critter.Direction rotate(Critter.Direction d) {
        if (d == Critter.Direction.NORTH)
            return Critter.Direction.EAST;
        else if (d == Critter.Direction.SOUTH)
            return Critter.Direction.WEST;
        else if (d == Critter.Direction.EAST)
            return Critter.Direction.SOUTH;
        else
```

```java
            return Critter.Direction.NORTH;
    }

    private Point pointAt(Point p, Critter.Direction d) {
        if (d == Critter.Direction.NORTH)
            return new Point(p.x, p.y - 1);
        else if (d == Critter.Direction.SOUTH)
            return new Point(p.x, p.y + 1);
        else if (d == Critter.Direction.EAST)
            return new Point(p.x + 1, p.y);
        else
            return new Point(p.x - 1, p.y);
    }

    private Info getInfo(PrivateData data, Class original) {
        Critter.Neighbor[] neighbors = new Critter.Neighbor[4];
        Critter.Direction d = data.direction;
        boolean[] neighborThreats = new boolean[4];
        for (int i = 0; i < 4; i++) {
            neighbors[i] = getStatus(pointAt(data.p, d), original);
            if (neighbors[i] == Critter.Neighbor.OTHER) {
                Point p = pointAt(data.p, d);
                PrivateData oldData = info.get(grid[p.x][p.y]);
                neighborThreats[i] = d == rotate(rotate(oldData.direction));
            }
            d = rotate(d);
        }
        return new Info(neighbors, data.direction, neighborThreats);
    }

    private Critter.Neighbor getStatus(Point p, Class original) {
        if (!inBounds(p))
            return Critter.Neighbor.WALL;
        else if (grid[p.x][p.y] == null)
            return Critter.Neighbor.EMPTY;
        else if (grid[p.x][p.y].getClass() == original)
            return Critter.Neighbor.SAME;
        else
            return Critter.Neighbor.OTHER;
    }

    public void update() {
        simulationCount++;
        Object[] list = info.keySet().toArray();
        Collections.shuffle(Arrays.asList(list));

        Set<Critter> locked = new HashSet<Critter>();
        for (int i = 0; i < list.length; i++) {
            Critter next = (Critter) list[i];
            PrivateData data = info.get(next);
            if (data == null) {

                continue;
            }

            boolean hadHopped = data.justHopped;
            data.justHopped = false;
            Point p = data.p;
            Point p2 = pointAt(p, data.direction);

            Critter.Action move = next.getMove(getInfo(data,
next.getClass()));
            if (move == Critter.Action.LEFT)
                data.direction = rotate(rotate(rotate(data.direction)));
            else if (move == Critter.Action.RIGHT)
                data.direction = rotate(data.direction);
            else if (move == Critter.Action.HOP) {
                if (inBounds(p2) && grid[p2.x][p2.y] == null) {
                    grid[p2.x][p2.y] = grid[p.x][p.y];
                    grid[p.x][p.y] = null;
                    data.p = p2;
                    locked.add(next);

                    data.justHopped = true;
                }
            } else if (move == Critter.Action.INFECT) {
                if (inBounds(p2) && grid[p2.x][p2.y] != null
                        && grid[p2.x][p2.y].getClass() != next.getClass()
                        && !locked.contains(grid[p2.x][p2.y])
                        && (hadHopped || Math.random() >= HOP_ADVANTAGE)) {
                    Critter other = grid[p2.x][p2.y];

                    PrivateData oldData = info.get(other);

                    String c1 = other.getClass().getName();
                    critterCount.put(c1, critterCount.get(c1) - 1);
                    String c2 = next.getClass().getName();
                    critterCount.put(c2, critterCount.get(c2) + 1);
                    info.remove(other);

                    try {
                        grid[p2.x][p2.y] = makeCritter(next.getClass());

                        locked.add(grid[p2.x][p2.y]);
                    } catch (Exception e) {
                        throw new RuntimeException("" + e);
                    }

                    info.put(grid[p2.x][p2.y], oldData);

                    oldData.justHopped = false;
                }
            }
        }
        updateColorString();
    }

    public void updateColorString() {
        for (Critter next : info.keySet()) {
            info.get(next).color = next.getColor();
            info.get(next).string = next.toString();
        }
    }

    public Set<Map.Entry<String, Integer>> getCounts() {
        return Collections.unmodifiableSet(critterCount.entrySet());
    }

    public int getSimulationCount() {
        return simulationCount;
    }

    private class PrivateData {
        public Point p;
        public Critter.Direction direction;
        public Color color;
        public String string;
        public boolean justHopped;

        public PrivateData(Point p, Critter.Direction d) {
            this.p = p;
            this.direction = d;
        }

        public String toString() {
            return p + " " + direction;
        }
    }

    private static class Info implements CritterInfo {
        private Critter.Neighbor[] neighbors;
        private Critter.Direction direction;
        private boolean[] neighborThreats;
        public Info(Critter.Neighbor[] neighbors, Critter.Direction d,
                boolean[] neighborThreats) {
            this.neighbors = neighbors;
            this.direction = d;
            this.neighborThreats = neighborThreats;
        }

        public Critter.Neighbor getFront() {
            return neighbors[0];
        }

        public Critter.Neighbor getBack() {
            return neighbors[2];
        }

        public Critter.Neighbor getLeft() {
            return neighbors[3];
        }

        public Critter.Neighbor getRight() {
            return neighbors[1];
        }

        public Critter.Direction getDirection() {
            return direction;
        }

        public boolean frontThreat() {
            return neighborThreats[0];
        }

        public boolean backThreat() {
            return neighborThreats[2];
        }

        public boolean leftThreat() {
            return neighborThreats[3];
        }

        public boolean rightThreat() {
            return neighborThreats[1];
        }
    }
}
```

CritterPanel.java

```java
package me.patricia.forest;

import javax.swing.*;
import java.awt.Point;
import java.awt.*;
import java.util.*;

public class CritterPanel extends JPanel {
    private CritterModel myModel;
    private Font myFont;
    private static boolean created;

    public static final int FONT_SIZE = 12;

    public CritterPanel(CritterModel model) {

        if (created)
            throw new RuntimeException("Only one world allowed");
        created = true;

        myModel = model;

        myFont = new Font("Monospaced", Font.BOLD, FONT_SIZE + 4);
        setBackground(Color.CYAN);
        setPreferredSize(new Dimension(FONT_SIZE * model.getWidth() + 20,
                FONT_SIZE * model.getHeight() + 20));
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setFont(myFont);
        Iterator<Critter> i = myModel.iterator();
        while (i.hasNext()) {
            Critter next = i.next();
            Point p = myModel.getPoint(next);
            String appearance = myModel.getAppearance(next);
            g.setColor(Color.BLACK);
            g.drawString("" + appearance, p.x * FONT_SIZE + 11,
                    p.y * FONT_SIZE + 21);
            g.setColor(myModel.getColor(next));
            g.drawString("" + appearance, p.x * FONT_SIZE + 10,
                    p.y * FONT_SIZE + 20);
        }
    }
}
```

FlyTrap.java

```java
package me.patricia.forest;

import java.awt.*;

public class FlyTrap extends Critter {
    public Action getMove(CritterInfo info) {
        if (info.getFront() == Neighbor.OTHER) {
            return Action.INFECT;
        } else {
            return Action.LEFT;
        }
    }

    public Color getColor() {
        return Color.RED;
    }

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

Food.java

```java
package me.patricia.forest;

import java.awt.*;

public class Food extends Critter {
    public Action getMove(CritterInfo info) {
        return Action.INFECT;
    }

    public Color getColor() {
        return Color.GREEN;
    }

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

Giant.java

```java
package me.patricia.forest;

import java.awt.*;

public class Giant extends Critter {
    private int moves;

    public Giant() {
        moves = 1;
        getColor();
    }

    public Color getColor() {
        return Color.GRAY;
    }

    public String toString() {

        if (moves <= 6) {
            return "fee";
        } else if (moves <= 12) {
            return "fie";
        } else if (moves <= 18) {
            return "foe";
        } else {
            return "fum";
        }
    }

    public Action getMove(CritterInfo info) {

        if (info.getFront() == Neighbor.OTHER) {
            countMoves();
            return Action.INFECT;
        } else if (info.getFront() == Neighbor.EMPTY) {
            countMoves();
            return Action.HOP;
        } else {
            countMoves();
            return Action.RIGHT;
        }
    }

    public void countMoves() {
        if (moves == 24) {
            moves = 1;
        } else {
            moves++;
        }
    }
}
```

NinjaCat.java

```
package me.patricia.forest;

import java.awt.*;

public class NinjaCat extends Tiger {

    public boolean hasInfected;

    public NinjaCat() {
        hasInfected = false;
    }

    public Color getColor() {
        if (hasInfected) {
            return Color.MAGENTA;
        } else {
            return Color.orange;
        }

    }

    public String toString() {
        if (hasInfected) {
            return "Z";
        } else {
            return "z";
        }

    }

    public Action getMove(CritterInfo info) {

        if (info.getFront() == Neighbor.OTHER) {
            hasInfected = true;
        }
        return super.getMove(info);

    }
}
```

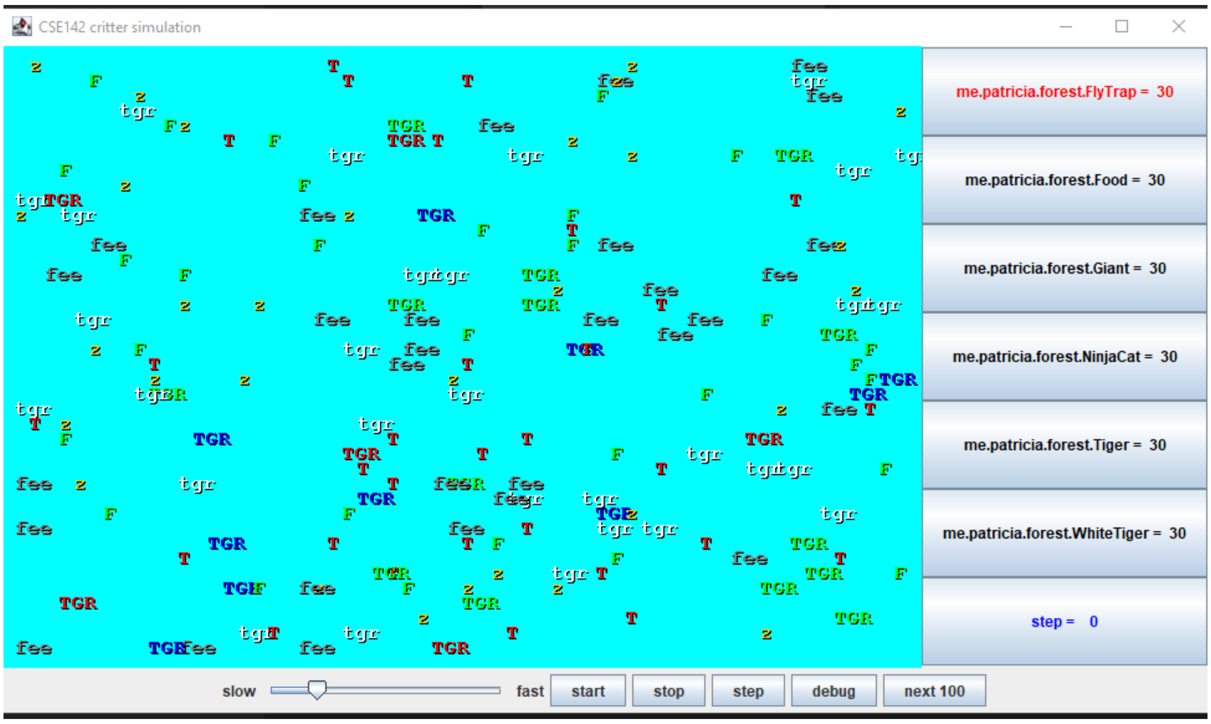
Tiger.Java

```java
package me.patricia.forest;

import java.awt.*;
import java.util.*;

public class Tiger extends Critter {
    private int colorMoves;
    Color tigerColor;
    Random rand = new Random();

    public Tiger() {
        colorMoves = 0;
        getColor();
    }

    public Color getColor() {
        if (colorMoves % 3 == 0) {
            int x = 0;
            while (x == 0) {
                int i = rand.nextInt(3);
                if (i == 0 && this.tigerColor != Color.RED) {
                    this.tigerColor = Color.RED;
                    x++;
                }
                if (i == 1 && tigerColor != Color.GREEN) {
                    this.tigerColor = Color.GREEN;
                    x++;
                }
                if (i == 2 && tigerColor != Color.BLUE) {
                    this.tigerColor = Color.BLUE;
                    x++;
                }
            }

        }
        return tigerColor;
    }

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

    public Action getMove(CritterInfo info) {
        colorMoves++;
        if (info.getFront() == Neighbor.OTHER) {
            return Action.INFECT;
        } else if (info.getFront() == Neighbor.WALL || info.getRight() ==
Neighbor.WALL) {
            return Action.LEFT;
        } else if (info.getFront() == Neighbor.SAME) {
            return Action.RIGHT;
        } else {
            return Action.HOP;
        }
    }
}
```

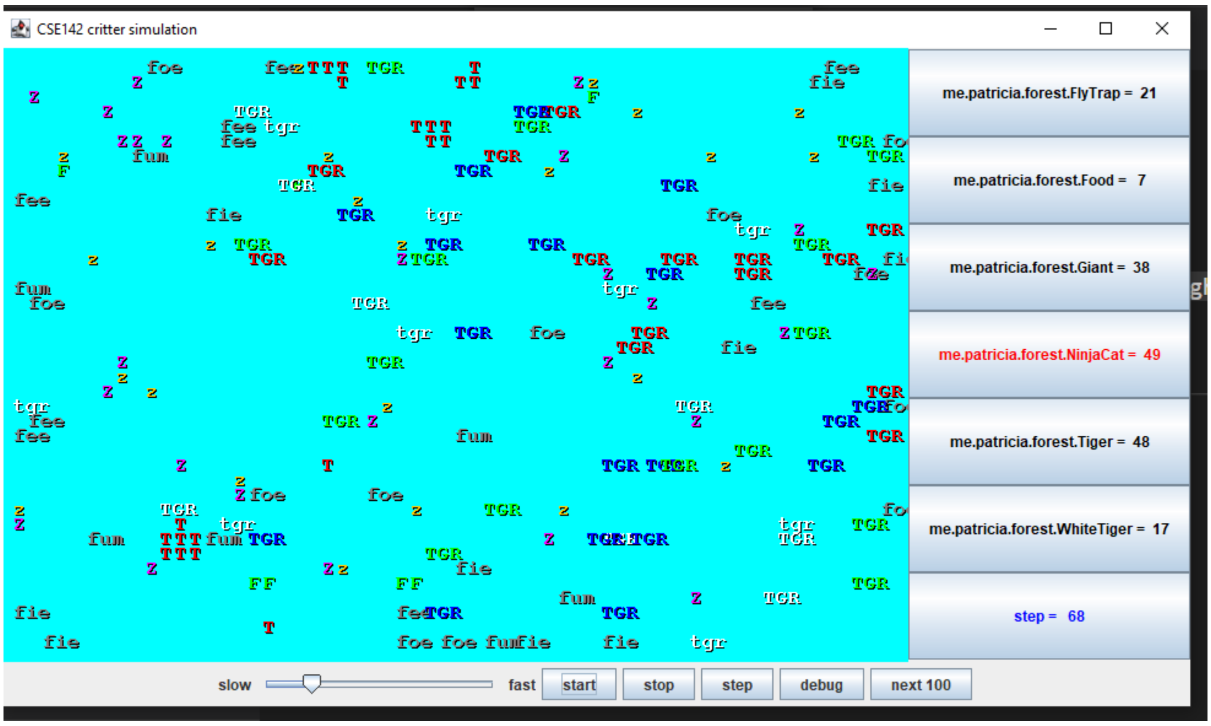
WhiteTiger.java

```
package me.patricia.forest;

import java.awt.*;

public class WhiteTiger extends Tiger {
    boolean hasInfected;

    public WhiteTiger() {
        hasInfected = false;
    }

    public Color getColor() {
        return Color.WHITE;
    }

    public String toString() {
        if (hasInfected) {
            return super.toString();
        } else {
            return "tgr";
        }
    }

    public Action getMove(CritterInfo info) {
        if (info.getFront() == Neighbor.OTHER) {
            hasInfected = true;
        }
        return super.getMove(info);

    }
}
```

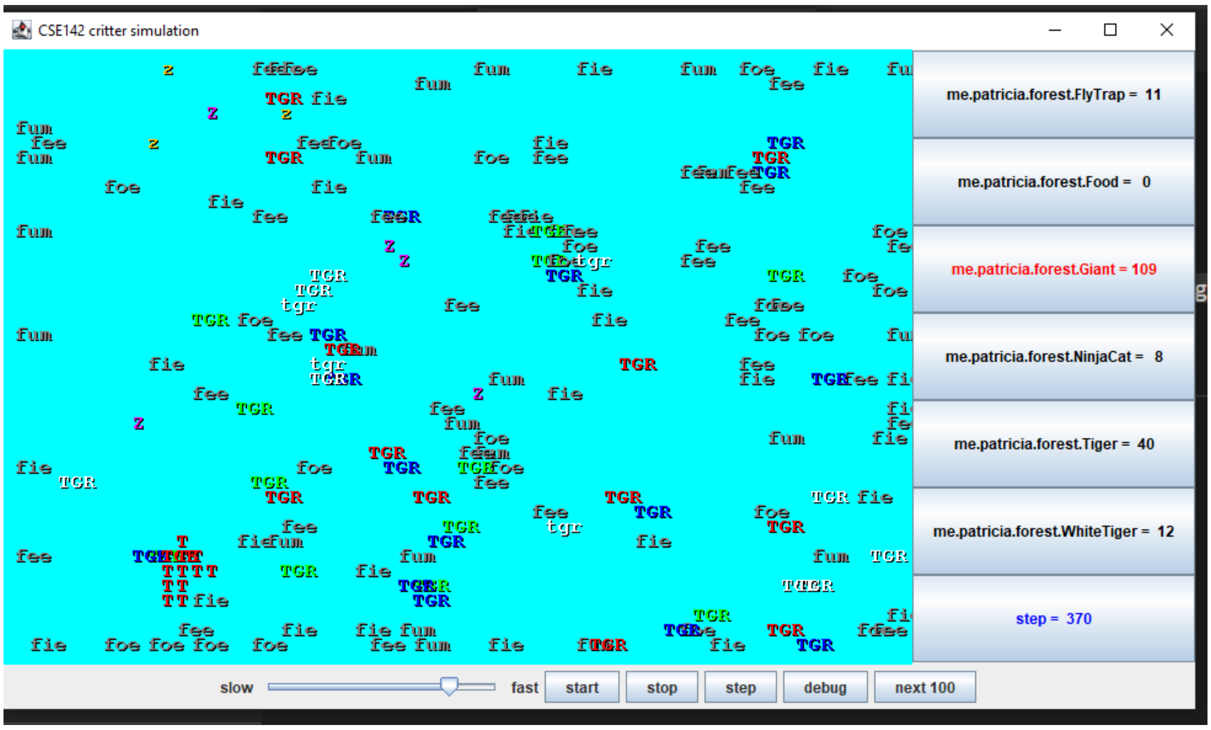
Outcome


CSE142 critter simulation

me.patricia.forest.FlyTrap = 30
me.patricia.forest.Food = 30
me.patricia.forest.Giant = 30
me.patricia.forest.NinjaCat = 30
me.patricia.forest.Tiger = 30
me.patricia.forest.WhiteTiger = 30
step = 0


CSE142 critter simulation

me.patricia.forest.FlyTrap = 21
me.patricia.forest.Food = 7
me.patricia.forest.Giant = 38
me.patricia.forest.NinjaCat = 49
me.patricia.forest.Tiger = 48
me.patricia.forest.WhiteTiger = 17
step = 68



me.patricia.forest.FlyTrap = 11

me.patricia.forest.Food = 0

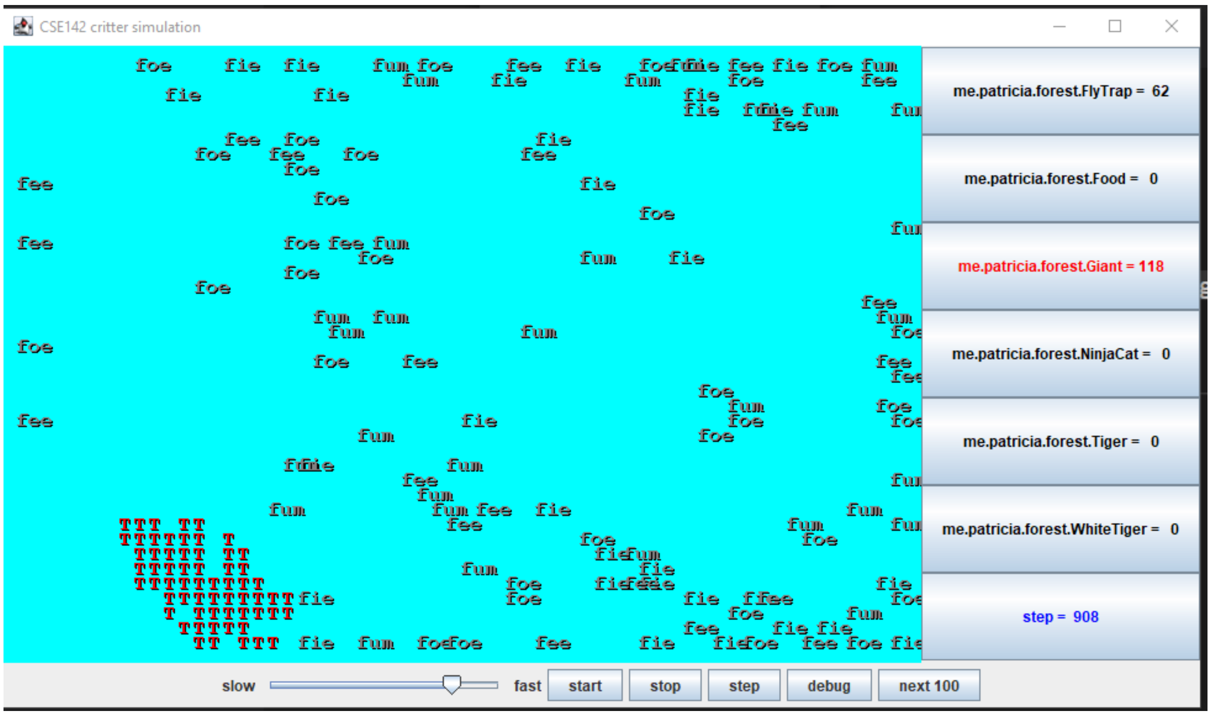
me.patricia.forest.Giant = 109

me.patricia.forest.NinjaCat = 8

me.patricia.forest.Tiger = 40

me.patricia.forest.WhiteTiger = 12

step = 370



me.patricia.forest.FlyTrap = 62

me.patricia.forest.Food = 0

me.patricia.forest.Giant = 118

me.patricia.forest.NinjaCat = 0

me.patricia.forest.Tiger = 0

me.patricia.forest.WhiteTiger = 0

step = 908

Discussion

Here are some ideas for how we can evaluate our creatures:

- Bear: Try running the simulator only with 30 bears in the entire world. Approximately half of them should be white and half should be black. They should all be presented with slash characters at first. All these need to change to backslash characters after you hit "step." They should return to slash characters and continue when you click "step" once more. As soon as you press "start," bears should start moving in a clockwise way toward walls and embracing them. Even though they occasionally collide and diverge, their natural tendency should be to follow the walls.

- Tiger: Attempt to run the simulator with only 30 Tigers present worldwide. Approximately one third of them should be red, one third green, and one third blue. To ensure that the colours alternate appropriately, press the "step" button. These first colours should be held for three moves. That implies they must maintain this hue while the simulator displays step 0, step 1, and step 2 indications. When the simulator indicates that you have completed step 3, they should change colours, and they should continue to do so for phases 4 and 5. The colour scheme for stages 6, 7, 8, and so on should then change. They would need to start hitting barriers after you click "start." They should turn around and go back the way they came when they run into a wall. They occasionally run into one another as well. They shouldn't congregate anywhere at all.

- WhiteTiger: The only difference between this one and a Tiger should be that they are White. Until they infect another Critter, they will likewise be lowercase; after that, they "grow up."

- Giant: Try running the sim with only 30 giants on the planet. All of these should be marked as "fee." This should be accurate for stages 0 through 5. They should all change to displaying "fie" when you reach step 6, and they should continue to do so for steps 6, 7, 8, 9, 10, and 11. Then, for steps 12, 13, 14, 15, 16, and 17, they need to be "foe."
  And for steps 18, 19, 20, 21, 22, and 23, they should all be "fum." Then they must return to "fee" for a further six steps, and so on. You should see the bear-like wall-hugging behaviour when you click "start," only this time it will be going clockwise.