REPORT FOR FINAL PROJECT: ANIMAL KINGDOM

STUDENT NAME:
ABDUL HAKIM BIN ABDUL RASHID

MATRIC NUMBER:
MC210413691

COURSE:
ITVM5013 – SOFTWARE DESIGN AND DEVELOPMENT

LECTURER:
DR. FERAS ZEN ALDEN

MASTER IN INFORMATION TECHNOLOGY
FACULTY OF BUSINESS AND TECHNOLOGY

2021

TABLE OF CONTENTS

1. ABSTRACT

The report presents the documentation regarding to the final project for course ITVM5310 Software Design and Development, which is required for semester of April 2021. The project title is Animal Kingdom that need to write a set of classes for define the behavior of certain animals. This final project would graded bases on the few criteria such as delivery and report presentation, coding design and standards, development and runtime and efficiency with 40% marks in overall.

2. INTRODUCTION

Animal kingdom, when look around, it will observe different animal structures and forms. As over a million species of animals have been described till now that need for classification becomes more important. The classification also helps in assigning a systematic position to newly described species.

Classification of animals helps us to understand their characteristics, as well as their differences with other organisms. Different kinds of animals will behave in different ways and can defining those differences.

For this final project, it will run the JAVA programming code simulation by using a combination of abstract classes, and interface and sorting for define the behavior of certain animals.

3. PROJECT DESCRIPTION

This project will expose the way to defining classes that need to write code a set of classes that define the behavior of certain animals.

Program given are to runs a simulation of a world with many animals wandering around in it. The specific task for this project that need to write code for five classes, each representing a different type of Animal which is, Bear, Tiger, White Tiger, Giant, and Ninja Cat.

Every object in this word is a “critter”, where Critter is the super class with default behavior defined. All the classes should be sub classes of Critter.

On each round of the simulation, each critter is asked for 3 pieces of information:

i. How should it act?
ii. What color is it?
iii. What string represents that critter?

These 3 pieces of information are provided by 3 methods present in each Critter class that it be responsible for overriding the method and programming their appropriate behavior.

4. PROJECT DESIGN

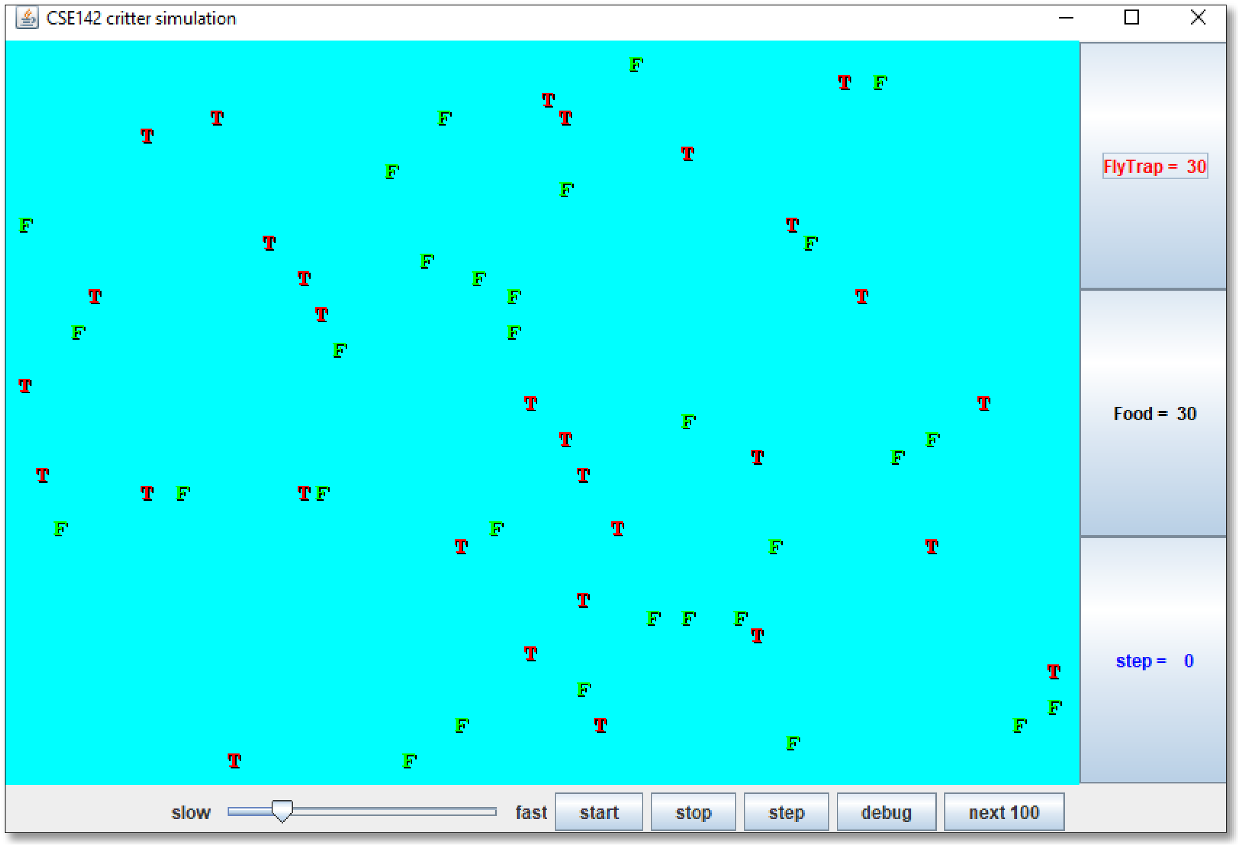
For this project, some JAVA programming language was provided to run the simulation such as,

i. Critter.java
ii. CritterInfo.java
iii. CritterPanel.java
iv. CritterModel.java
v. CritterMain.java
vi. CritterFrame.java
vii. Food.java
viii. FlyTrap.java

The task of the project to design the set of 5 classes that represent the behavior of each animal below,

i. Bear
ii. Tiger
iii. White Tiger
iv. Giant
v. NinjaCat

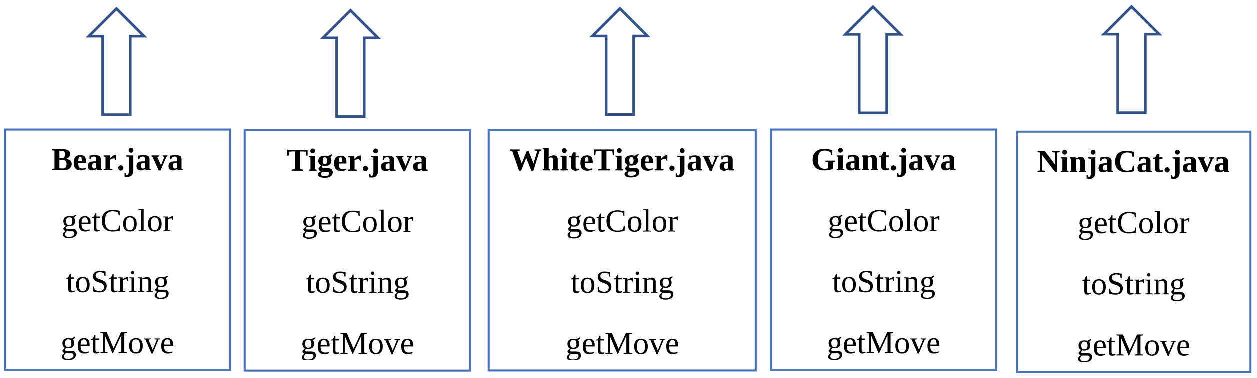
And then, that set of the 5 classes need to connect with simulation program by re-write the code in CritterMain.java and need to try run back the simulation.



The Simulation Program

Included: FlyTrap.Java & Food.Java

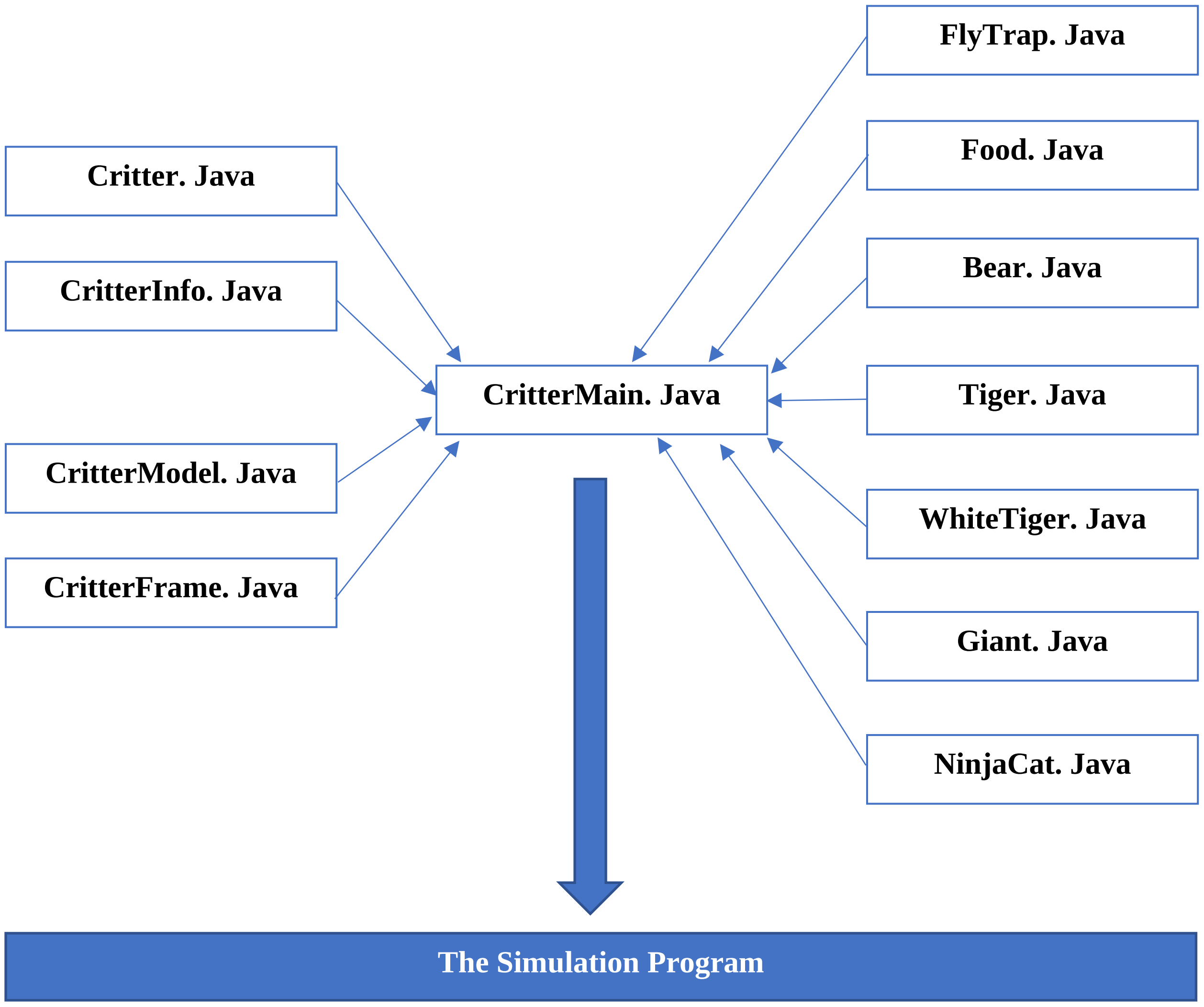
Combine new 5 classes together with **CritterMain.java** and run the simulation



Each class represent the behavior of each animal

5. PROJECT WORKFLOW AND LOGICS

The diagram below shows the project workflow and logics,



Five classes behavior that added in the simulation program:

1. public class **Bear** extends Critter

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. public class **Tiger** extends Critter

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. public class **WhiteTiger** extends **Tiger**

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. public class **Giant** extends Critter

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. public class **NinjaCat** extends Critter

Constructor	**public NinjaCat()**
getColor	Color.YELLOW
toString	"Meow”
getMove	always infect if an enemy is in front, otherwise if a wall or same is in front or to the right, then turn left, and otherwise hop.

The method that are used:

1. getColor()

As the simplest method, to writing the getColor() method first. For getColor should return whatever color the simulator to use when drawing that each critter. Colors are represented like "Color.WHITE". For the random colors, each possible choice must be equally likely. May use either a Random object or the Math.random() method. If color changes based on moves, it will want some way to count how many moves a critter has made. This state should only be updated in the getMove method, and simply referenced in others (like getColor or toString).

2. toString()

For the toString method, should return whatever text that want the simulator to use when displaying your critter (normally a single character). Remember, that if String changes based on moves might need to keep track of the critter’s number of moves. For this assignment, do not worry about integer overflow (Won’t run the simulation that long).

3. getMove()

The getMove method can return only 4 different things:

Constant	Description
Action.HOP	Move forward one square in its current direction
Action.LEFT	Turn left (rotate 90 degrees counter-clockwise)
Action.RIGHT	Turn right (rotate 90 degrees clockwise)
Action.INFECT	Infect the critter in front of you

This is why the return type is “Action”. It will need to use information about the area around the Critter to determine which Action to return. For example, Bear, Tiger, WhiteTiger and NinjaCat should all return Action.INFECT if an enemy is in front of them that can tell what is around the critter based on the CritterInfo object parameter:

CritterInfo Method	Description	Possible Return Values
public Neighbor getFront()	returns neighbor in front of you	Neighbor.WALL, Neighbor.EMPTY, Neighbor.SAME, Neighbor.OTHER
public Neighbor getBack()	returns neighbor in back of you	Neighbor.WALL, Neighbor.EMPTY, Neighbor.SAME, Neighbor.OTHER
public Neighbor getLeft()	returns neighbor to your left	Neighbor.WALL, Neighbor.EMPTY, Neighbor.SAME, Neighbor.OTHER
public Neighbor getRight()	returns neighbor to your right	Neighbor.WALL, Neighbor.EMPTY, Neighbor.SAME, Neighbor.OTHER
public Direction getDirection()	returns direction you are facing	Direction.NORTH, Direction.SOUTH, Direction.EAST, Direction.WEST
public boolean frontThreat()	whether there is an enemy facing you in front of you	TRUE or FALSE
public boolean backThreat()	whether there is an enemy facing you in back of you	TRUE or FALSE
public boolean leftThreat()	whether there is an enemy facing you to your left	TRUE or FALSE
public boolean rightThreat()	whether there is an enemy facing you to your rig	TRUE or FALSE

6. DEVELOPMENT DETAILS

Below are the details that contains of JAVA Programming code that has been wrote to develop the simulation program:

i. Critter.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT
//Student Name: ABDUL HAKIM BIN ABDUL RASHID
//Matric Number: MC210413691
//Course: MASTER OF INFORMATION TECHNOLOGY
//Final Project: Animal Kingdom
// This is the superclass of all of the Critter classes.  Your class should
// extend this class.  The class provides several kinds of constants:
//
//    type Neighbor  : WALL, EMPTY, SAME, OTHER
//    type Action    : HOP, LEFT, RIGHT, INFECT
//    type Direction : NORTH, SOUTH, EAST, WEST
//
// Override the following methods to change the behavior of your Critter:
//
//     public Action getMove(CritterInfo info)
//     public Color getColor()
//     public String toString()
//
```

```
// The CritterInfo object passed to the getMove method has the following
// available methods:
//
//     public Neighbor getFront();           neighbor in front of you
//     public Neighbor getBack();            neighbor in back of you
//     public Neighbor getLeft();            neighbor to your left
//     public Neighbor getRight();           neighbor to your right
//     public Direction getDirection();      direction you are facing
//     public boolean frontThreat();         threatening critter in front?
//     public boolean backThreat();          threatening critter in back?
//     public boolean leftThreat();          threatening critter to the left?
//     public boolean rightThreat();         threatening critter to the right?

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
    };


    // This method should be overriden (default action is turning left)
    public Action getMove(CritterInfo info) {
        return Action.LEFT;
    }


    // This method should be overriden (default color is black)
    public Color getColor() {
        return Color.BLACK;
    }
    // This method should be overriden (default display is "?")
    public String toString() {
        return "?";
    }
    // This prevents critters from trying to redefine the definition of
    // object equality, which is important for the simulator to work properly.
    public final boolean equals(Object other) {
        return this == other;
    }
}
```

ii. CritterInfo.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT

//Student Name: ABDUL HAKIM BIN ABDUL RASHID

//Matric Number: MC210413691

//Course: MASTER OF INFORMATION TECHNOLOGY

//Final Project: Animal Kingdom


// The CritterInfo interface defines a set of methods for querying the

// state of a critter simulation.  You should not alter this file.  See

// the documentation in the Critter class for a more detailed explanation.


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();

}
```

iii. CritterPanel.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT
//Student Name: ABDUL HAKIM BIN ABDUL RASHID
//Matric Number: MC210413691
//Course: MASTER OF INFORMATION TECHNOLOGY
//Final Project: Animal Kingdom

// Class CritterPanel displays a grid of critters

import javax.swing.*;
import java.awt.Point;
import java.awt.*;
import java.awt.event.*;
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) {
        // this prevents someone from trying to create their own copy of
        // the GUI components
        if (created)
            throw new RuntimeException("Only one world allowed");
        created = true;

        myModel = model;
        // construct font and compute char width once in constructor
        // for efficiency
        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);
        }
    }
}
```

iv. CritterModel.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT
//Student Name: ABDUL HAKIM BIN ABDUL RASHID
//Matric Number: MC210413691
//Course: MASTER OF INFORMATION TECHNOLOGY
//Final Project: Animal Kingdom

// Class CritterModel keeps track of the state of the critter simulation.

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

public class CritterModel {
    // the following constant indicates how often infect should fail for
    // critters who didn't hop on their previous move (0.0 means no advantage,
    // 1.0 means 100% advantage)
    public static final double HOP_ADVANTAGE = 0.2; // 20% advantage

    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) {
        // this prevents someone from trying to create their own copy of
        // the GUI components
        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);
}
@SuppressWarnings("unchecked")
private Critter makeCritter(Class critter) throws Exception {
    Constructor c = critter.getConstructors()[0];
    if (critter.toString().equals("class Bear")) {
        // flip a coin
        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) {
    // Override specified toString if debug flag is true
    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);
}

// returns the result of rotating the given direction clockwise
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 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;
}
@SuppressWarnings("unchecked")
public void update() {
    simulationCount++;
    Object[] list = info.keySet().toArray();
    Collections.shuffle(Arrays.asList(list));
```

```
// This keeps track of critters that are locked and cannot be
// infected this turn. The happens when:
// * a Critter is infected
// * a Critter hops
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) {
        // happens when creature was infected earlier in this round
        continue;
    }
    // clear any prior setting for having hopped in the past
    boolean hadHopped = data.justHopped;
    data.justHopped = false;
    Point p = data.p;
    Point p2 = pointAt(p, data.direction);

                // try to perform the critter's action
    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); //successful hop locks a critter from
                         // being infected for the rest of the
                         // turn
            data.justHopped = true;  // remember a successful hop
        }
    } 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];
            // remember the old critter's private data
            PrivateData oldData = info.get(other);
            // then remove that old critter
            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);
            // and add a new one to the grid
            try {
                grid[p2.x][p2.y] = makeCritter(next.getClass());
                // This critter has been infected and is now locked
                // for the rest of this turn
                locked.add(grid[p2.x][p2.y]);
            } catch (Exception e) {
                throw new RuntimeException("" + e);
            }
            // and add to the map
            info.put(grid[p2.x][p2.y], oldData);
            // but it's new, so it didn't just hop
```

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

// calling this method causes each critter to update the stored color and
// text for toString; should be called each time update is performed and
// once before the simulation begins
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;
        }
    }

    // an object used to query a critter's state (neighbors, 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];
        }
    }
}
```

v. CritterMain.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT
//Student Name: ABDUL HAKIM BIN ABDUL RASHID
//Matric Number: MC210413691
//Course: MASTER OF INFORMATION TECHNOLOGY
//Final Project: Animal Kingdom

// CSE 142 Homework 9 (Critters)
// Authors: Stuart Reges and Marty Stepp
//          modified by Kyle Thayer
//
// CritterMain provides the main method for a simple simulation program.  Alter
// the number of each critter added to the simulation if you want to experiment
// with different scenarios.  You can also alter the width and height passed to
// the CritterFrame constructor.

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

        // uncomment each of these lines as you complete these classes
        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();
    }
}
```

vi. CritterFrame.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT

//Student Name: ABDUL HAKIM BIN ABDUL RASHID

//Matric Number: MC210413691

//Course: MASTER OF INFORMATION TECHNOLOGY

//Final Project: Animal Kingdom


// Class CritterFrame provides the user interface for a simple simulation

// program.

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) {
    // this prevents someone from trying to create their own copy of
    // the GUI components
    if (created)
        throw new RuntimeException("Only one world allowed");
    created = true;

    // create frame and model
    setTitle("CSE142 critter simulation");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    myModel = new CritterModel(width, height);

    // set up critter picture panel
    myPicture = new CritterPanel(myModel);
    add(myPicture, BorderLayout.CENTER);

    addTimer();
    constructSouth();

    // initially it has not started
    started = false;
}
```

```
// construct the controls and label for the southern panel
private void constructSouth() {
    // add timer controls to the south
    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));
        }
    });
    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);

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

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

    add(p, BorderLayout.SOUTH);
}

// starts the simulation...assumes all critters have already been added
public void start() {
    // don't let anyone start a second time and remember if we have started
    if (started) {
        return;
    }
    // if they didn't add any critters, then nothing to do
    if (myModel.getCounts().isEmpty()) {
        System.out.println("nothing to simulate--no critters");
        return;
    }
    started = true;
    addClassCounts();
    myModel.updateColorString();
    pack();
    setVisible(true);
}

// add right-hand column showing how many of each critter are alive
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]);
    }

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


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


private void setCounts() {
    Set<Map.Entry<String, Integer>> entries = myModel.getCounts();
    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);
    }

    // add a certain number of critters of a particular class to the simulation
    public void add(int number, Class<? extends Critter> c) {
        // don't let anyone add critters after simulation starts
        if (started) {
            return;
        }
        // temporarily turning on started flag prevents critter constructors
        // from calling add
        started = true;
        myModel.add(number, c);
        started = false;
    }

    // post: creates a timer that calls the model's update
    //       method and repaints the display
    private void addTimer() {
        ActionListener updater = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                doOneStep();
            }
        };
        myTimer = new javax.swing.Timer(0, updater);
        myTimer.setCoalesce(true);
    }

    // one step of the simulation
    private void doOneStep() {
        myModel.update();
        setCounts();
        myPicture.repaint();
    }

    // advance the simulation until step % n is 0
    private void multistep(int n) {
        myTimer.stop();
        do {
            myModel.update();
        } while (myModel.getSimulationCount() % n != 0);
        setCounts();
        myPicture.repaint();
    }
}
```

vii. Food.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT
//Student Name: ABDUL HAKIM BIN ABDUL RASHID
//Matric Number: MC210413691
//Course: MASTER OF INFORMATION TECHNOLOGY
//Final Project: Animal Kingdom
// This defines a simple class of critters that sit around waiting to be
// taken over by other critters.
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";
    }
}
```

viii. FlyTrap.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT
//Student Name: ABDUL HAKIM BIN ABDUL RASHID
//Matric Number: MC210413691
//Course: MASTER OF INFORMATION TECHNOLOGY
//Final Project: Animal Kingdom
// This defines a simple class of critters that infect whenever they can and
// otherwise just spin around, looking for critters to infect.  This simple
// strategy turns out to be surpisingly successful.
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";
   }
}
```

ix. Bear.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT
//Student Name: ABDUL HAKIM BIN ABDUL RASHID
//Matric Number: MC210413691
//Course: MASTER OF INFORMATION TECHNOLOGY
//Final Project: Animal Kingdom

import java.awt.*;

public class Bear extends Critter{
    boolean polar;
    int count;

    public Bear(boolean polar) {
        super();
        this.polar = polar;
        count = 0;
    }


    public Color getColor() {
        if (polar==false) {return Color.BLACK;}
        return Color.WHITE;
    }



    public String toString() {
        if (count%2==0) {return "/";}
        return "\\";
    }

    public Action getMove(CritterInfo info) {
        count++;
        if (info.getFront() == Neighbor.OTHER) {
            return Action.INFECT;
        }
        else if(info.getFront() == Neighbor.EMPTY) { return Action.HOP;}

        return Action.LEFT;
    }
}
```

x. Tiger.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT

//Student Name: ABDUL HAKIM BIN ABDUL RASHID

//Matric Number: MC210413691

//Course: MASTER OF INFORMATION TECHNOLOGY

//Final Project: Animal Kingdom


import java.awt.Color;

import java.util.Random;


public class Tiger extends Critter{

   int count;

   int i;




   public Tiger() {

      super();

      count = 0;

   }




   public Color getColor() {

      Color[] colorTiger = {Color.RED , Color.GREEN , Color.BLUE};
```

```
        if (count%3==0) {i = (int) (Math.random()*3); }

        return colorTiger[i];

    }



    public String toString() {

        return "TGR";

    }



    public Action getMove(CritterInfo info) {

        count++;

        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; }

        return Action.HOP;

    }


}
```

xi. WhiteTiger.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT

//Student Name: ABDUL HAKIM BIN ABDUL RASHID

//Matric Number: MC210413691

//Course: MASTER OF INFORMATION TECHNOLOGY

//Final Project: Animal Kingdom


import java.awt.Color;


public class WhiteTiger extends Tiger{

    boolean hasInfectedAnother;


    public Color getColor() {

        return Color.WHITE;

    }




    public WhiteTiger() {

        super();

        hasInfectedAnother = false;

    }
    public String toString() {
        if (hasInfectedAnother) { return "TGR"; }
        return "tgr";



    }




     public Action getMove(CritterInfo info) {
        count++;
        if(info.getFront() == Neighbor.OTHER) {
            hasInfectedAnother = true;
            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; }
        return Action.HOP;
    }


}
```

xii. Giant.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT

//Student Name: ABDUL HAKIM BIN ABDUL RASHID

//Matric Number: MC210413691

//Course: MASTER OF INFORMATION TECHNOLOGY

//Final Project: Animal Kingdom


import java.awt.Color;


public class Giant extends Critter{
   int count;


   public Giant() {
      super();
      count = 0;
   }


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


   public String toString() {
      int Remaining = count%24;
```

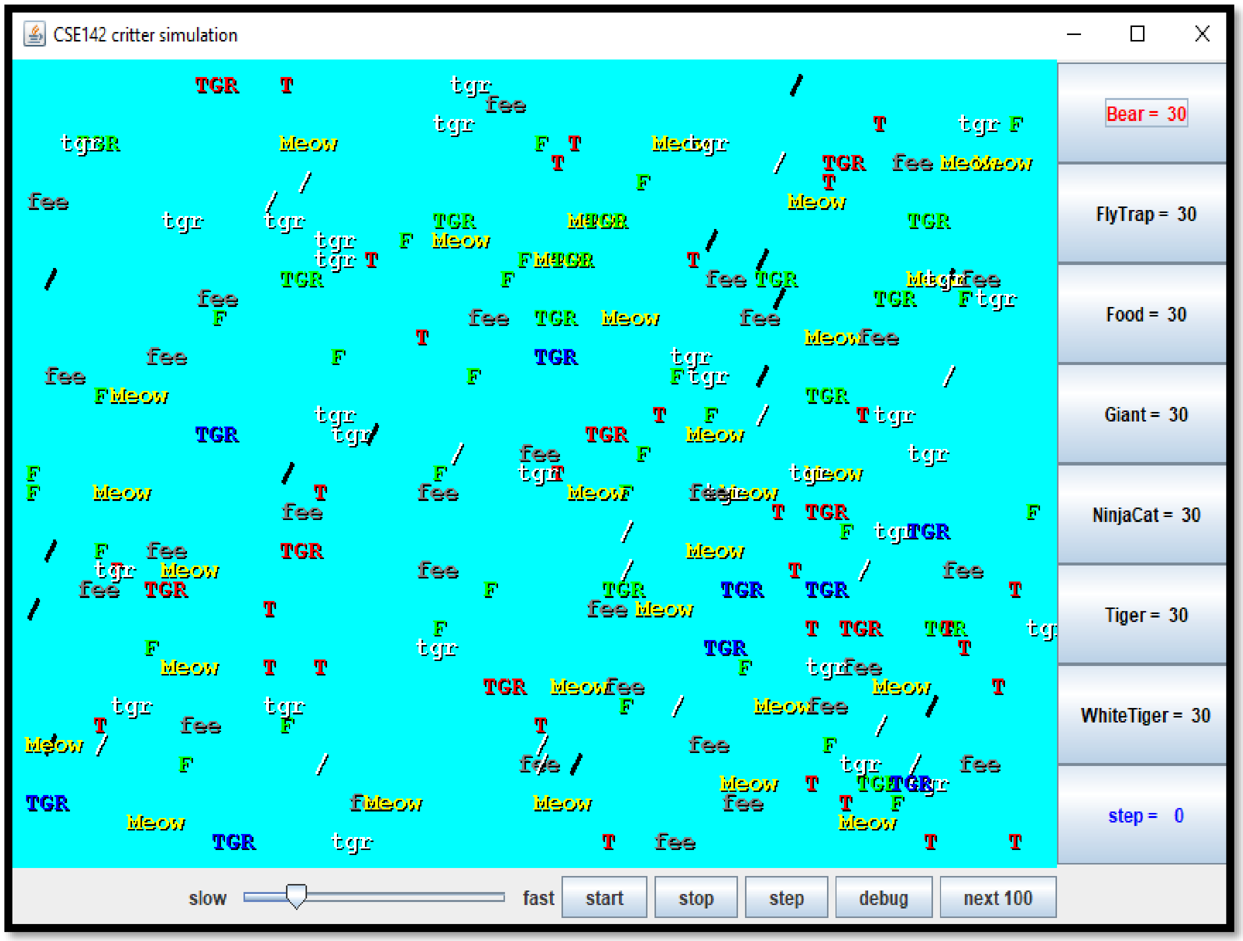
```
        if(Remaining<6) { return "fee"; }

        else if(Remaining >= 6 && Remaining < 12) { return "fie"; }

        else if(Remaining >= 12 && Remaining < 18) { return "fum"; }

        else if(Remaining >= 18 && Remaining < 24) { return "foe"; }

        else return "fee";

    }


    public Action getMove(CritterInfo info) {

        count++;

        if (info.getFront() == Neighbor.OTHER) { return Action.INFECT; }

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

        return Action.RIGHT;

    }

}
```

xiii. NinjaCat.java

```
//Subject: ITVM5013 SOFTWARE DESIGN AND DEVELOPMENT

//Student Name: ABDUL HAKIM BIN ABDUL RASHID

//Matric Number: MC210413691

//Course: MASTER OF INFORMATION TECHNOLOGY

//Final Project: Animal Kingdom


import java.awt.Color;

import java.util.Random;



public class NinjaCat extends Critter{


    int count;

    Action[] RL = {Action.LEFT, Action.RIGHT};

    Random rand;


    public NinjaCat() {

        super();

        count = 0;

        rand = new Random();

    }
    public Color getColor() {

        return Color.YELLOW;

    }



    public String toString() {

        return "Meow";

    }



      public Action getMove(CritterInfo info) {

        if(info.getFront() == Neighbor.OTHER) { return Action.INFECT; }

        else if(info.getFront() == Neighbor.WALL || info.getFront() == Neighbor.SAME)
{return RL[rand.nextInt(RL.length)]; }

        return Action.HOP;

    }

}
```

7. RESULT

As a result, after all the 5 classes of Animals behavior was combined, the Simulation Program will display like below screenshot:



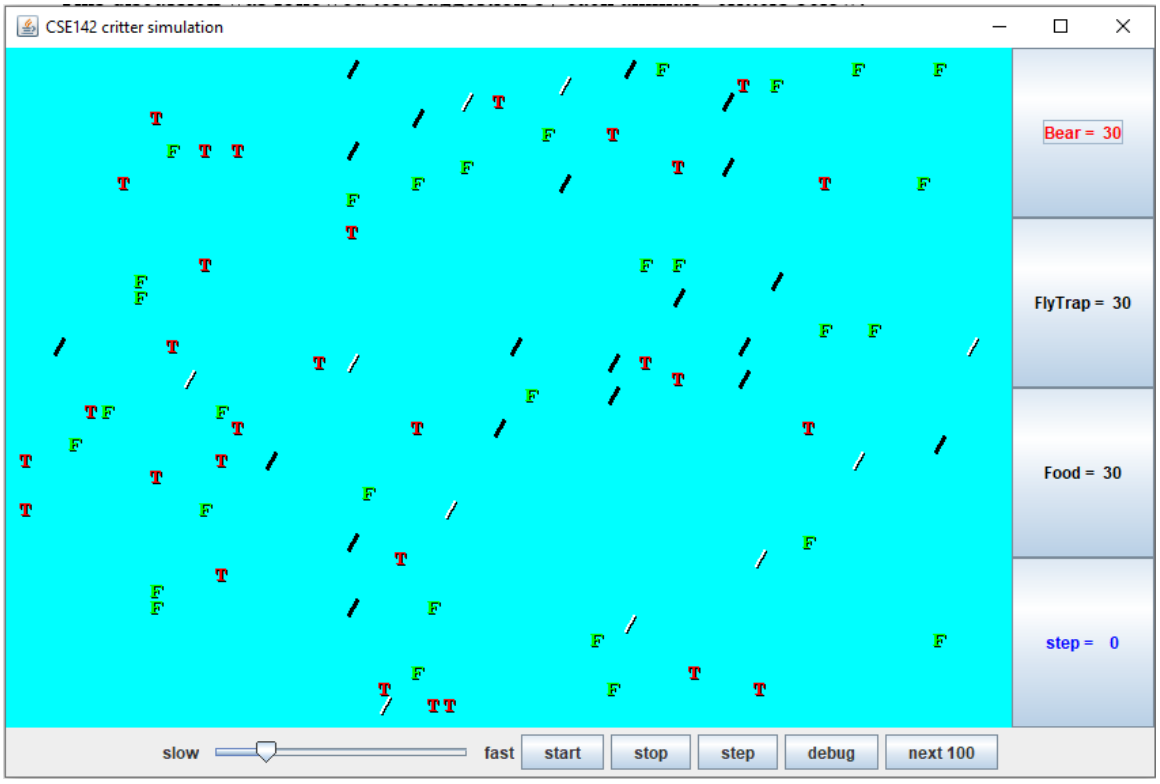
The simulation Program was display all that related to Bear.java, Tiger.java, WhiteTiger.java, Giant.java and NinjaCat.java, and when click the button start, all animals will run the each method getMove() and CritterInfo Method according to possible return values.

8. DISCUSSION

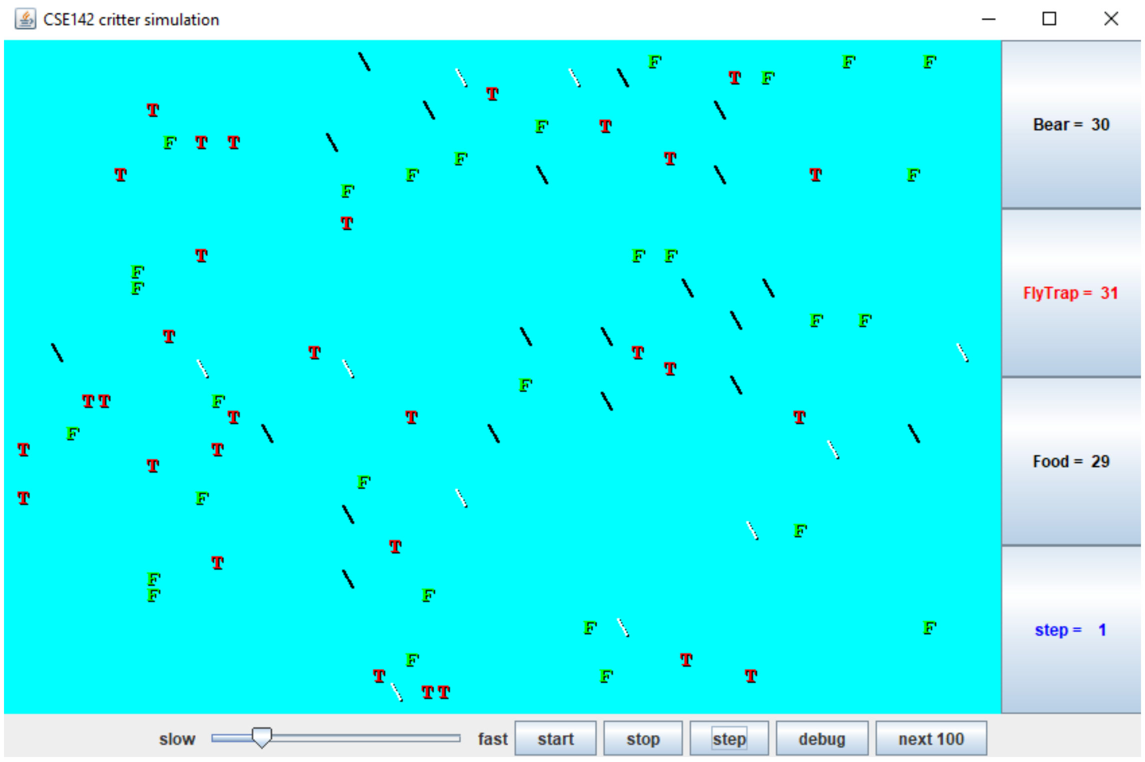
This discussion was followed test suggestion by each animals' critters below:

i. **Bear**: Try running the simulator with just 30 bears in the world. It should see about half of them being white and about half being black. Initially they should all be displayed with slash characters. When click "step", they should all switch to backslash characters. When click "step" again they should go back to slash characters and so on. When click "start", it should observe the bears heading towards walls and then hugging the walls in a counterclockwise direction. They will sometimes bump into each other and go off in other directions, but their tendency should be to follow along the walls.

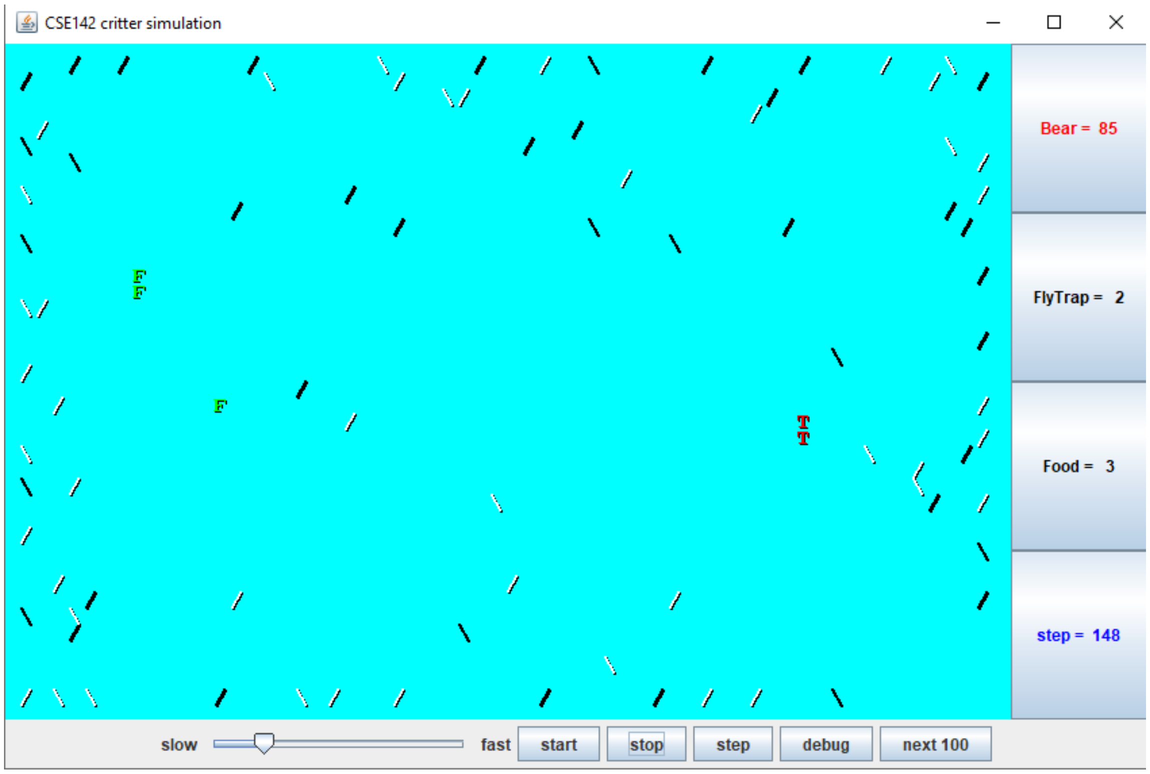
When click button "step" first time:



When click button “step” 2 times:



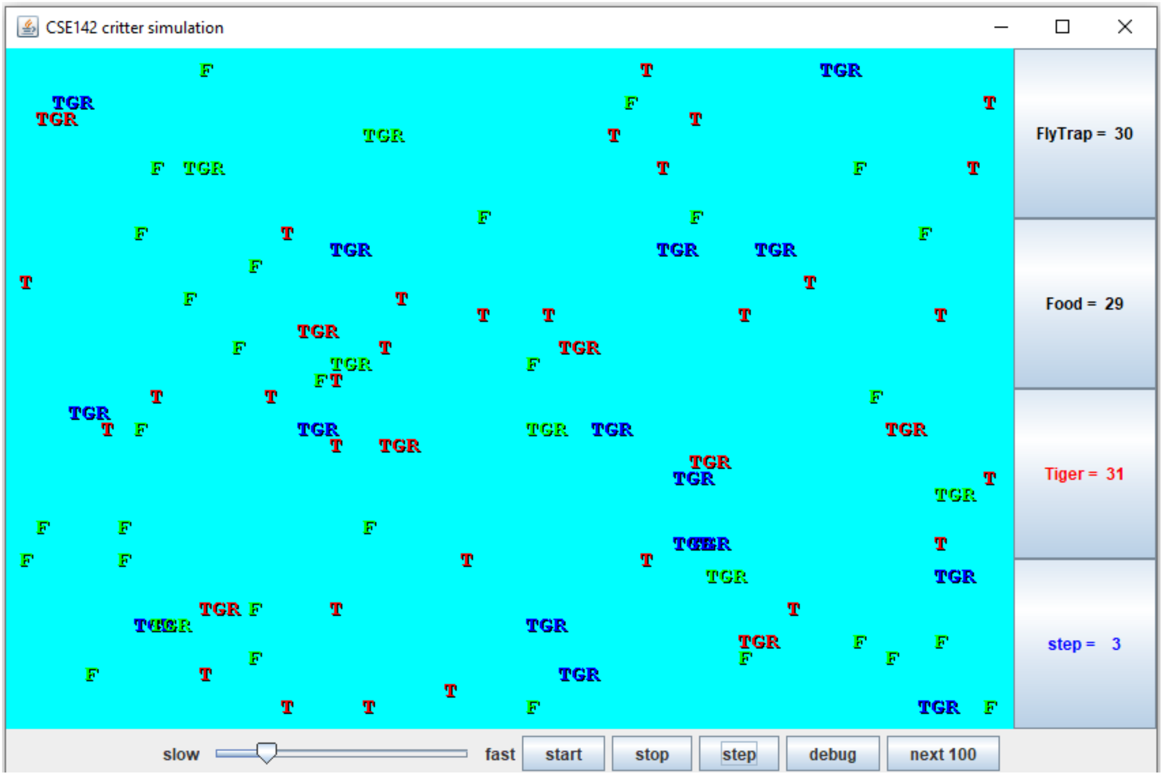
When click button “start”:



ABDUL HAKIM BIN ABDUL RASHID (MC210413691)

ii. **Tiger:** Try running the simulator with just 30 Tigers in the world. It should see about one third of them being red and one third being green and one third being blue. Use the "step" button to make sure that the colors alternate properly. They should keep these initial colors for three moves. That means that they should stay this color while the simulator is indicating that it is step 0, step 1, and step 2. They should switch colors when the simulator indicates that you are up to step 3 and should stay with these new colors for steps 4 and 5. Then you should see a new color scheme for steps 6, 7, and 8 and so on. When you click "start" you should see them bouncing off of walls. When they bump into a wall, they should turn around and head back in the direction they came. They will sometimes bump into each other as well. They shouldn't end up clustering together anywhere.

When click button "step" 3 times:



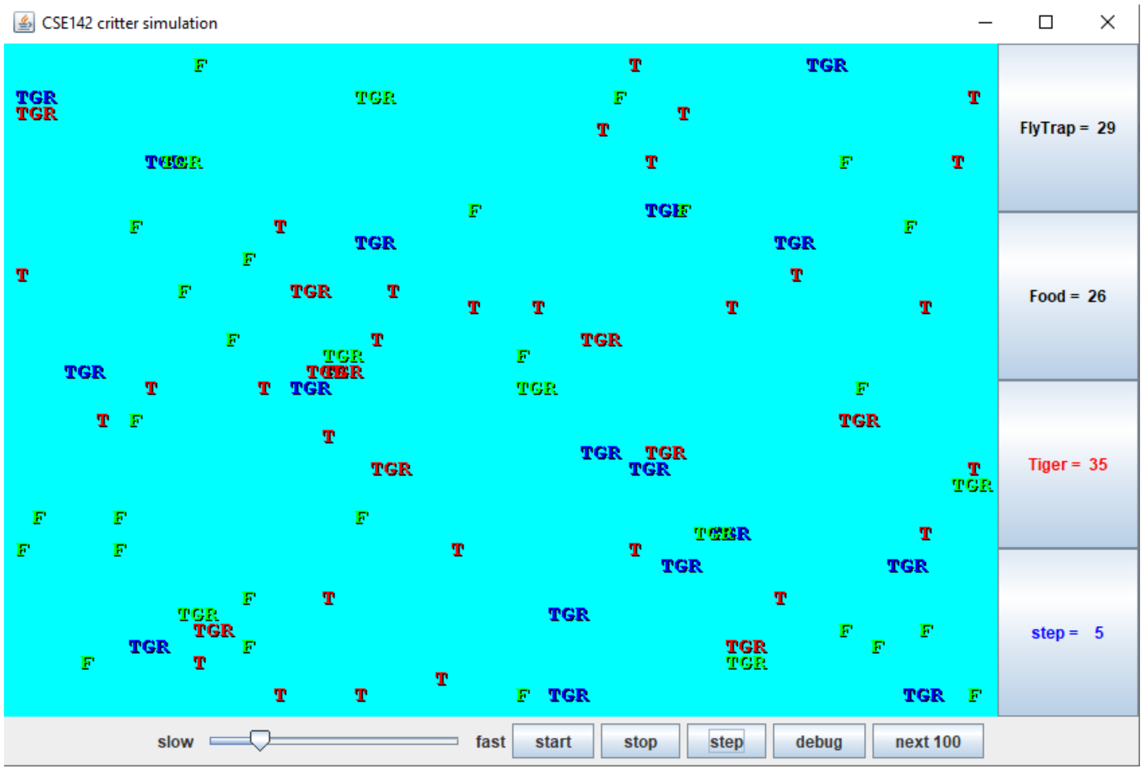
FlyTrap = 30

Food = 29

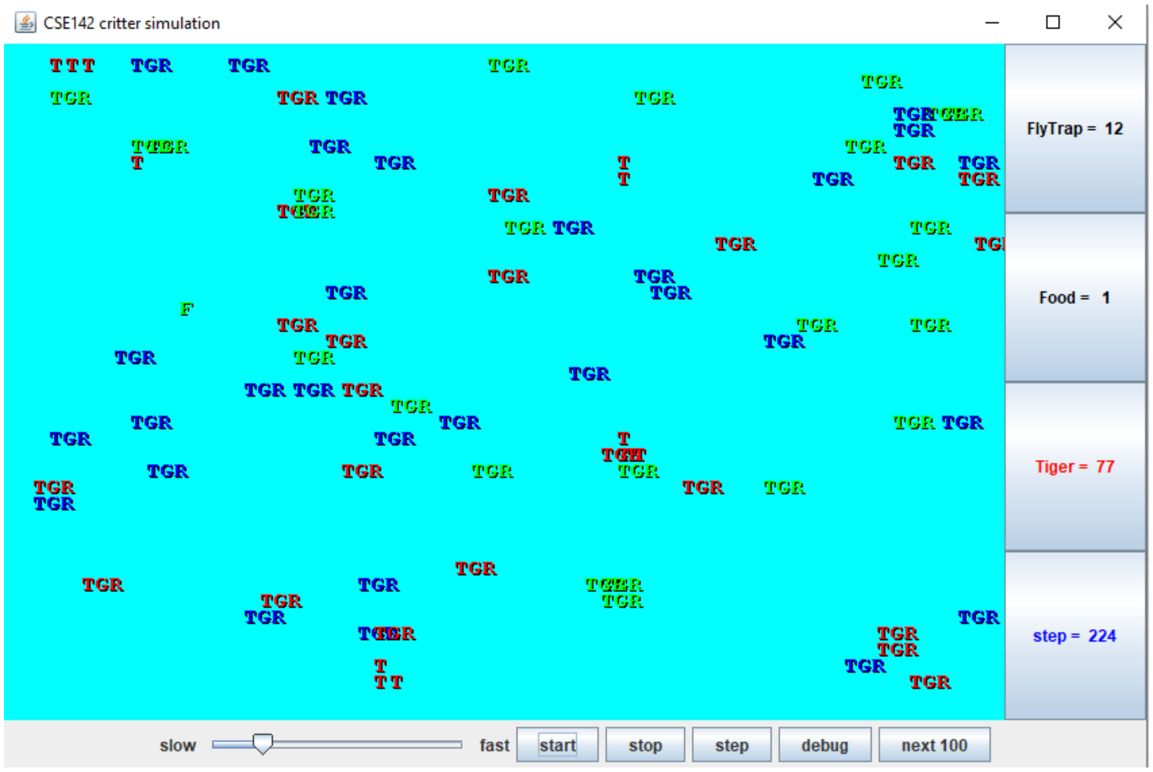
Tiger = 31

step = 3

When click button “step” 5 times:



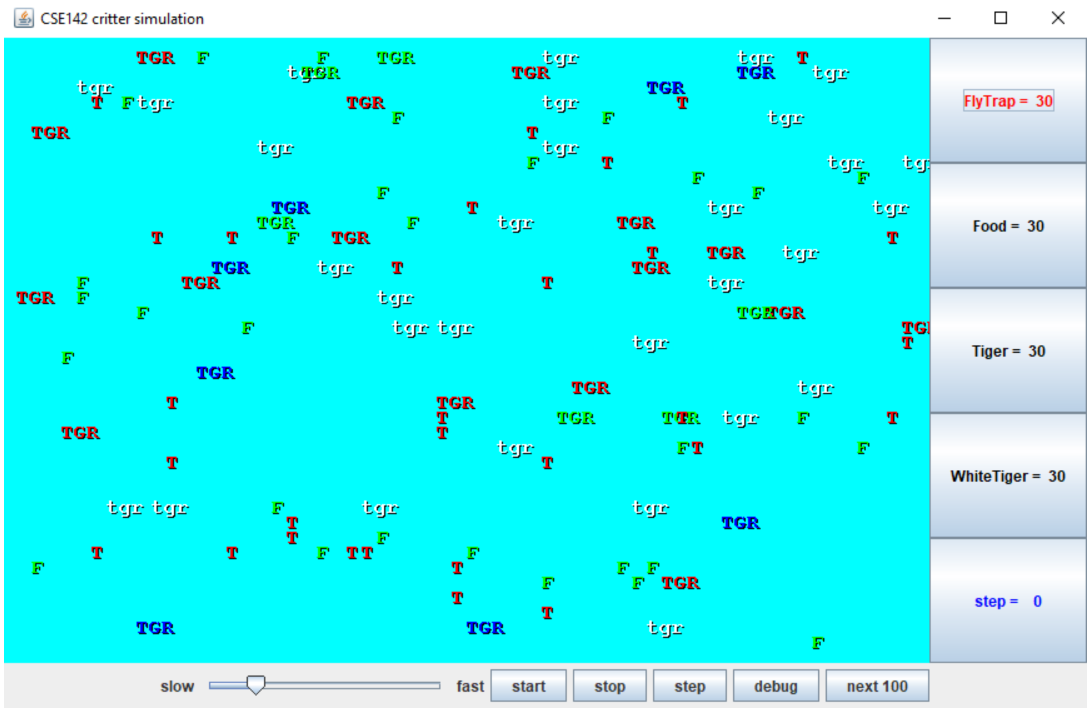
When click button “start”:



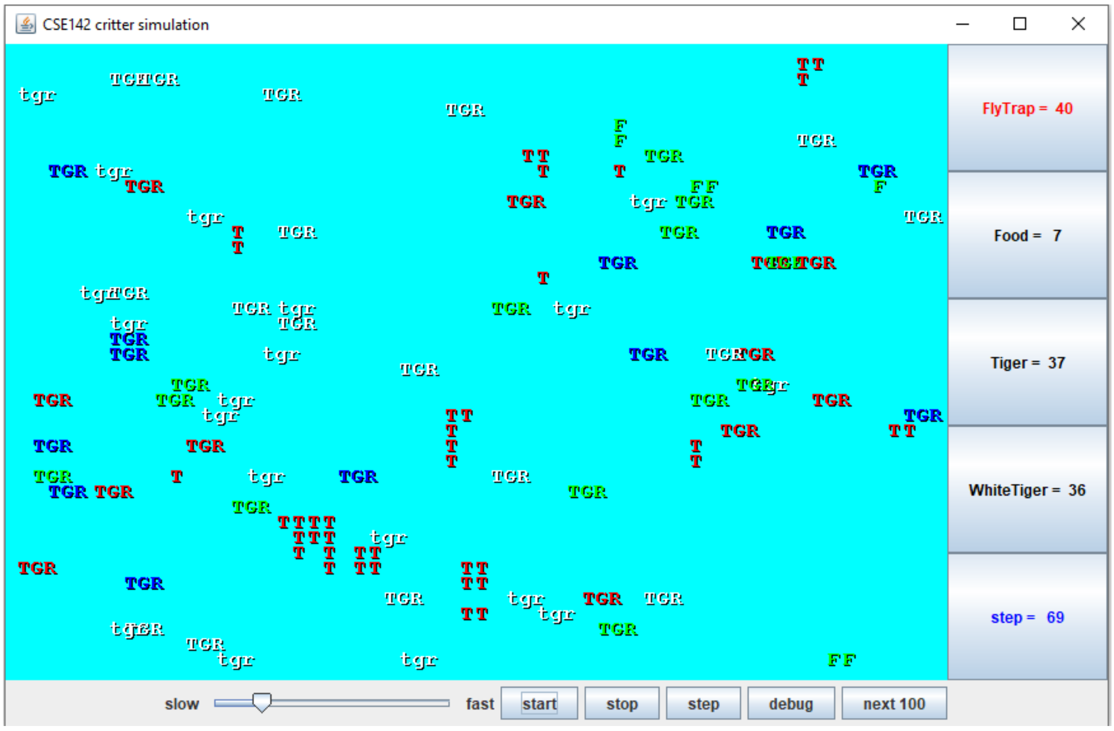
ABDUL HAKIM BIN ABDUL RASHID (MC210413691)

WhiteTiger: This should behave just like a Tiger except that they will be White. They will also be lower-case until they infect another Critter, then they "grow up".

Before click button "start":

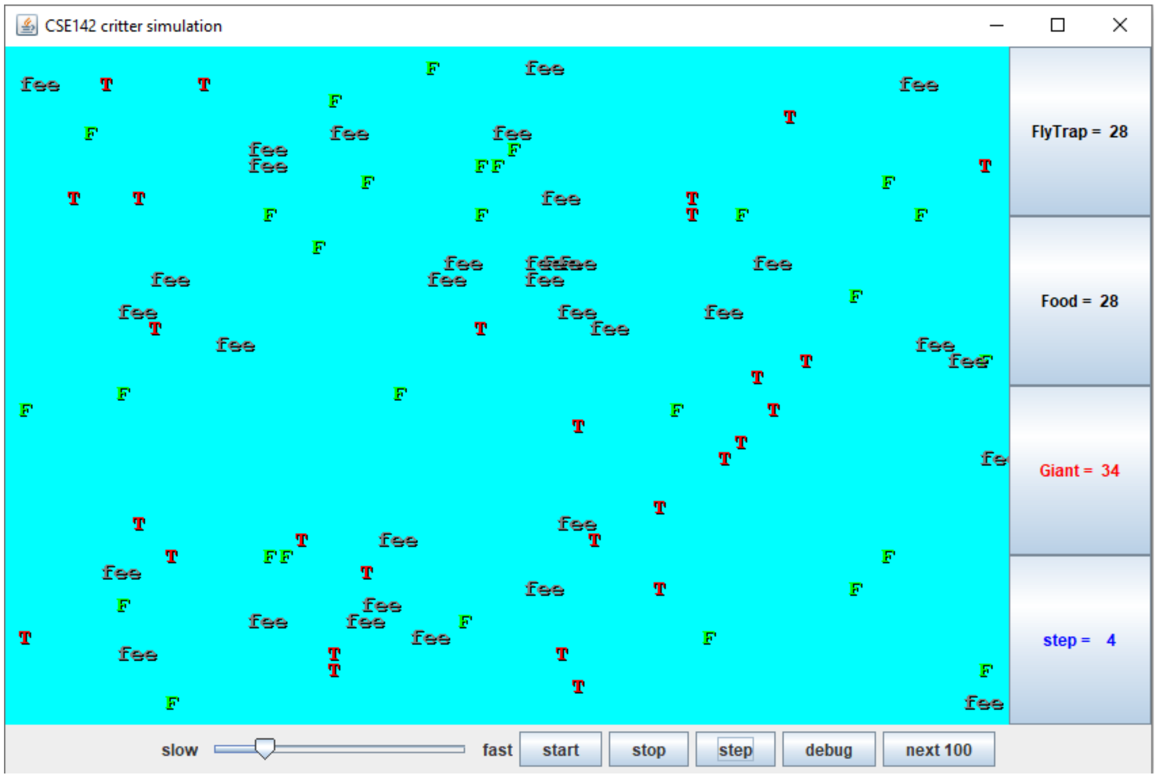


When click button "start":

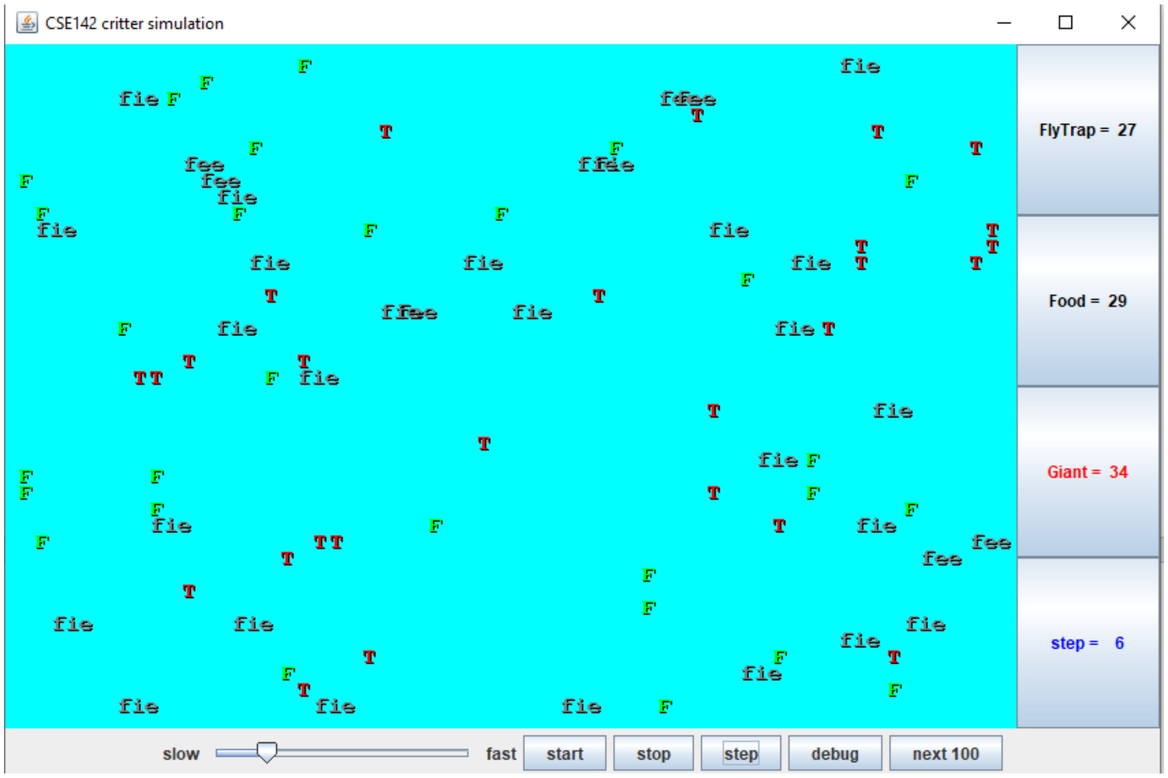


iii. **Giant:** Try running the simulator with just 30 giants in the world. They should all be displayed as "fee". This should be true for steps 0, 1, 2, 3, 4, and 5. When you get to step 6, they should all switch to displaying "fie" and should stay that way for steps 6, 7, 8, 9, 10, and 11. Then they should be "foe" for steps 12, 13, 14, 15, 16, and 17. And they should be "fum" for steps 18, 19, 20, 21, 22, and 23. Then they should go back to "fee" for 6 more steps, and so on. When you click "start", you should observe the same kind of wall-hugging behavior that bears have, but this time in a clockwise direction.

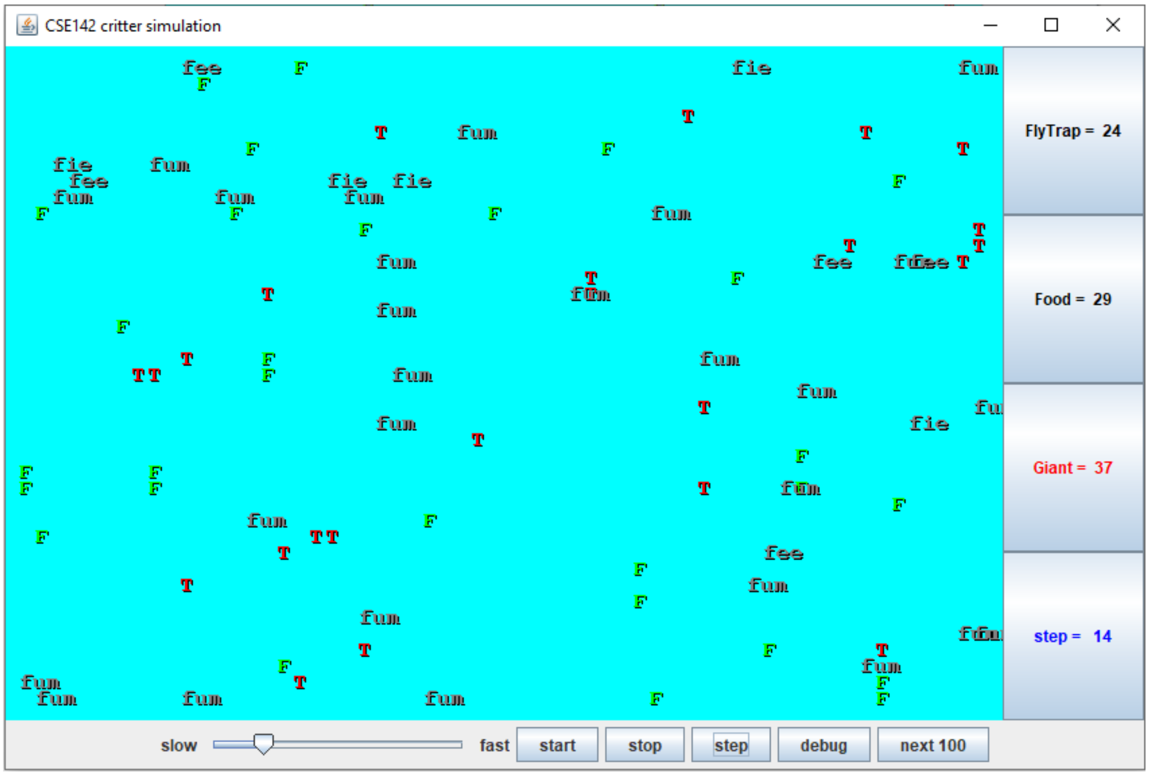
When click button “step” 5 times:



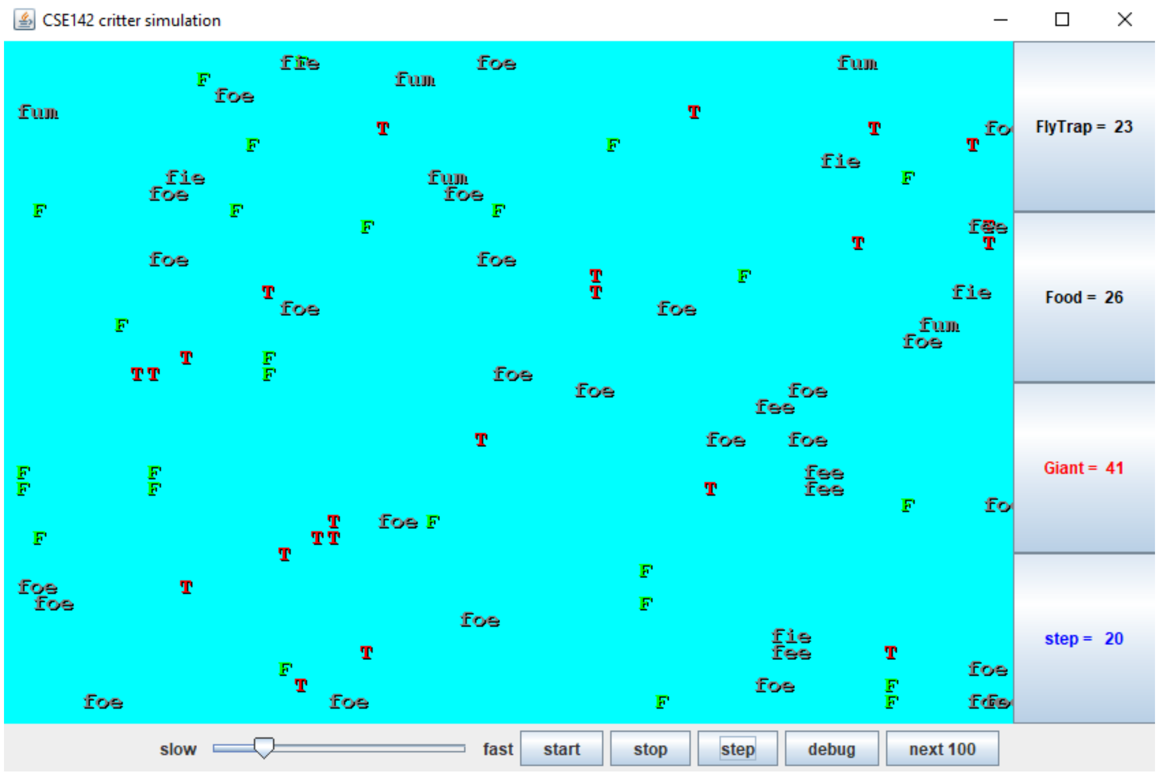
When click button "step" 6 times:



When click button "step" 14 times:

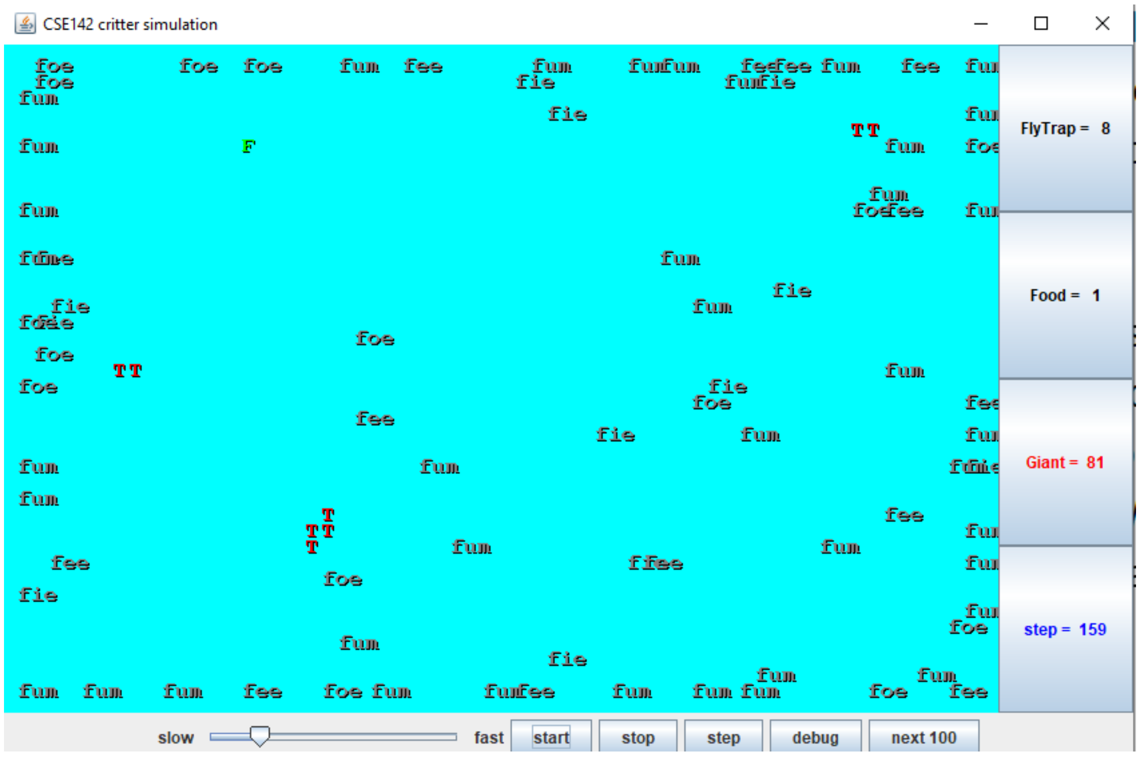


When click button “step” 20 times:



FlyTrap = 23
Food = 26
Giant = 41
step = 20

When click button “start”:



FlyTrap = 8
Food = 1
Giant = 81
step = 159

9. CONCLUSION

The conclusion for this project, the Animal Kingdom simulator program are good example for develop system with certain requirements that related with each other and will get the reflection if the requirements was encountered such as behavior of animals in this Animal Kingdom Simulator.

Student was done this project also can has experienced to write JAVA code programming by using several methods, classes and combine the JAVA file with “extend” code for connect JAVA File with another JAVA File also can test to see the effect from the code they wrote.

This kind of project can expose for do the enhancement for more advanced technology soon. More experience will create many opportunities for develop the better systems.