Logo, company name

Description automatically generated

**UNITAR GRADUATE SCHOOL**

**Course: ITWM5113 Software Design and Development**

**Course Instructor: Simon Lau**

**Assignment Submission**

**(Group/Individual)**



Project Title: ANIMAL KINGDOM

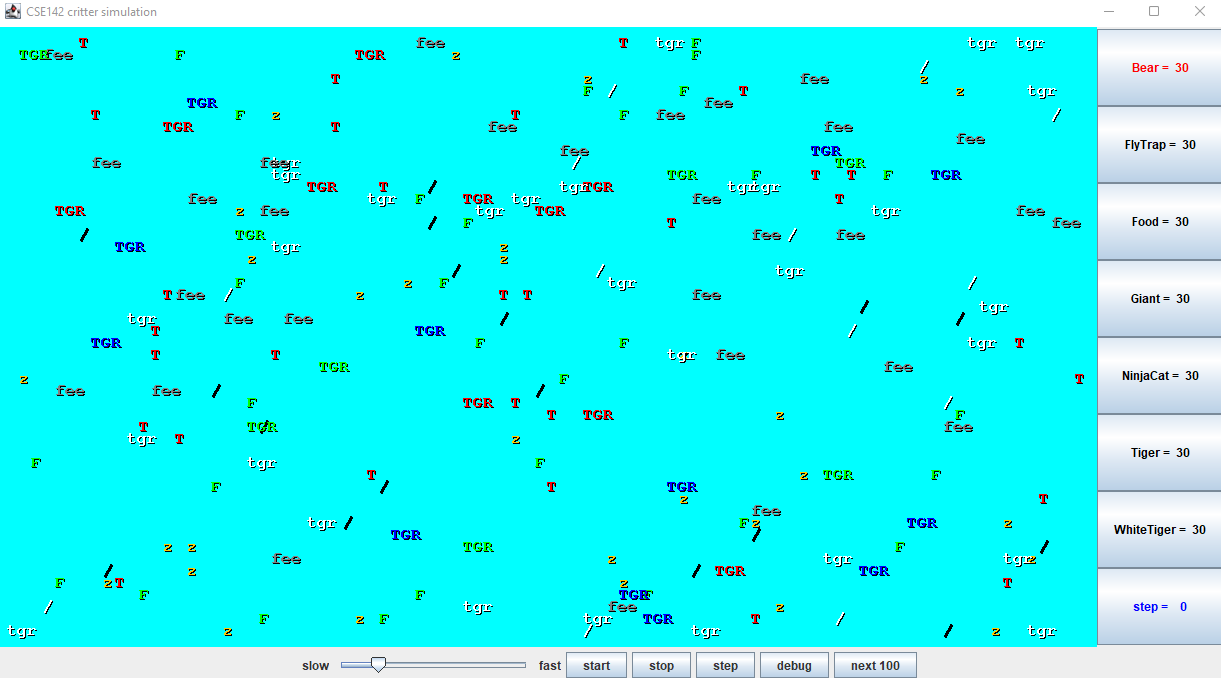
| **Name** | **Student ID** | **Section** |
| --- | --- | --- |
| DEVAN THANGA SELVARAJ | UNU2200575 | Coding + Report |

Table of Contents Pg

1. ABSTRACT……………………………………………………………………………………2
2. PROJECT INTRODUCTION………………………………………………………………….3
3. PROJECT DESCRIPTION…………………………………………………………………….4
4. PROJECT DESIGN……………………………………………………………………………6
5. PROGRAM WORKFLOW AND LOGIC……………………………………………………..7
6. DEVELOPMENT DETAILS…………………………………………………………….…8-15
7. RESULTS……………………………………………………………………………………...16

1. DISCUSSION …………………………………………………………………………………18
2. CONCLUSION………………………………………………………………………………...18
3. GITHUB INFORMATION……………………………………………………………………19

**FINAL PROJECT – ANIMAL KINGDOM**



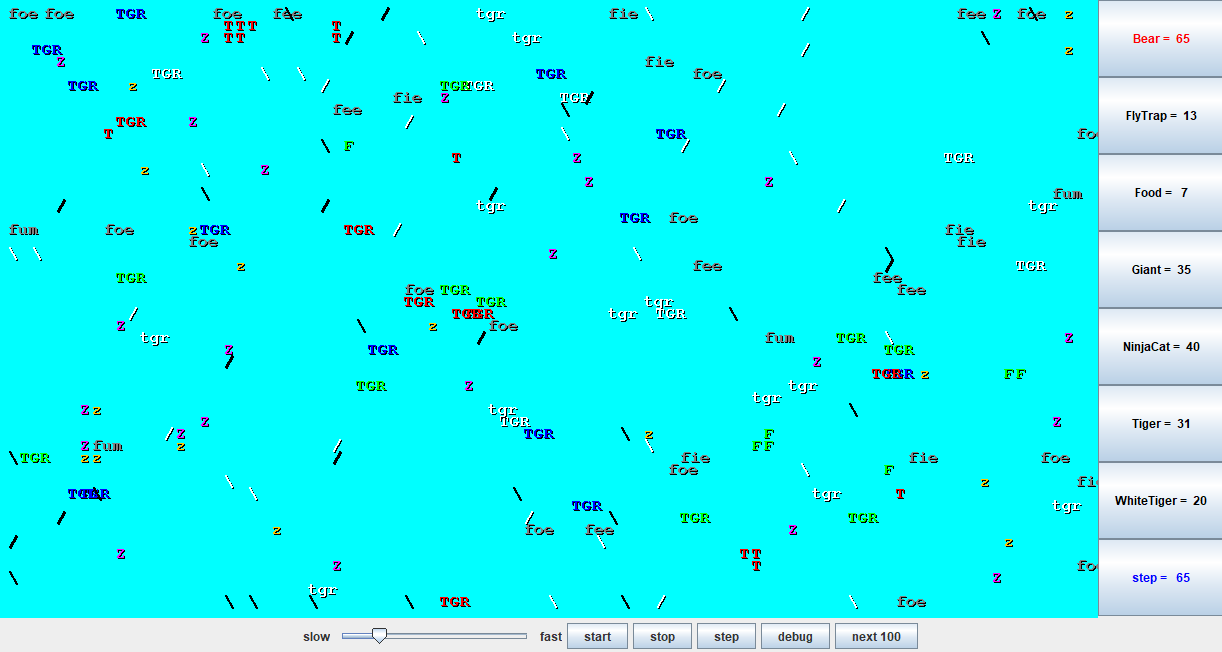
1. **ABSTRACT**

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior. OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained. These features include Abstraction, encapsulation, inheritance and polymorphism. Inheritance is one of the main concepts in Object Oriented Programming (OOP). Using inheritance, we can model a problem properly and we can reduce the number of lines we have to write. An object can be defined as a data field that has unique attributes and behaviour. OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained. This includes programs for manufacturing and design, as well as mobile applications. Writing this report and building this application in Java using Object Oriented Programming (OOP) can be quite a daunting experience which requires understanding of some large libraries, and advanced aspects of Java in OOP. In this project, requires writing a set of classes that define the behaviour of certain animals and will be given a program that runs a simulation of a world with many animals wandering around in it. Different kinds of animals will behave in different ways, and you are defining those differences. To produce the result, an immerse research and test an error has been in action.

1. **PROJECT INTRODUCTION**

This final project is given to opportunity to practice with set of classes that define character of certain animals. The output of the program project the many animals wandering around it.

Each animal behave differently for this project focuses on the use of multiple objects and classes and how they interact with each other and given a lot of code for this project which includes a simulation of many animals roaming about. Different kinds of animals will behave in different ways, and we’re defining those differences. For this assignment, have will be given a lot of supporting code that runs the simulation. While it is running, the simulation will look like the image below:



1. **PROJECT DESCRIPTION**

This project require us write Java code where Critter is the super class with default behavior defined and will be writing five classes, each representing a different type of Animal: Bear, Tiger, WhiteTiger, Giant and NinjaCat. All of the classes right should be sub classes of Critter. On each round of the simulation, each critter is asked for 3 pieces of information such as How should it act?, What color is it? And What string represents that critter?. These 3 pieces of information are provided by 3 methods present in each Critter class and be responsible for overriding these methods and programming their appropriate behavior.

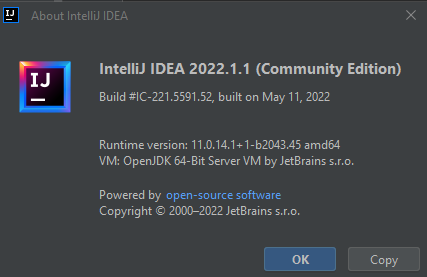
The code to these behaviors will be contained within each of the Critter classes and I need to override each of the methods within the classes and program the behavior. The simulator what color to draw the Critter. I need to declare a color by using the code color dot and enter color. For random, each choice should have an equal likelihood of getting chosen. So, I can use an object or math.random(). The toString method is the next one to tackle. This will return a string of text telling the simulator what to display when using your Critter. The getMove method is our final method to answer the behaviors.

This method can only return 4 things. action.up, action.Left, action.Right,and action.Infect. To determine which direction the method should return, use a few additional methods to examine the Critters surroundings and environments. Throughout all the Critters, there may come a time when one of the variables will need to change based on several moves of the Critter has made. It would be helpful in the long run to add a move counter within getMove so that the other methods may use it when necessary. As I complete each class, I have verified that everything is working correctly.

This is where testing comes in handy. To isolate the Critter, I was working on in the simulator comment out the frame.add lines in Critter main. This way I will be able to single out the Critter and watch step by step that my code is working. For the direction instructions, the simulator has a debug mode which will display the Critter as an arrow pointing in the direction that your Critter is facing.

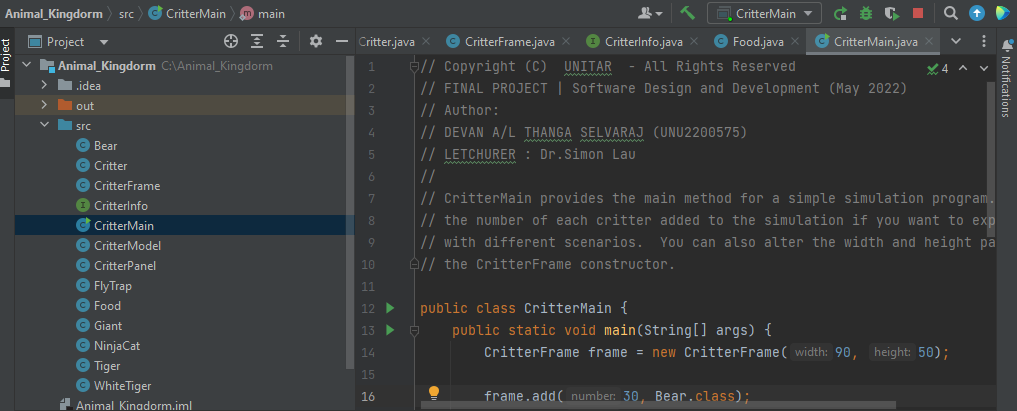
**4.0 PROJECT DESIGN**

The requirement is to develop an Animal Kingdom Game using JAVA Programming Language. As to achive the goals, I use the IntelliJ IDEA 2022.1.1 (Cummunity Edition)



The project uses classes to build the entire structure. It requires to compile using the JAVA 18 SDK and run the CritterMain.java as the main program to call the main form of interface.

The below image is the components of the 8 classes for the project of Animal Kingdom.



Each class has it color, move,behavior and string.

**5.0 PROGRAM WORKFLOW AND LOGIC**

There are five animal classes. I have selected one of the main constructors (getMove()) from the five animal classes to demonstrate the flowchart sequence as below:

* **Bear.java**
  + getMove()

No

No

return

Action.HOP;

Front==

Neighbor.EMPTY

Front==

Neighbor.OTHER

END

START

Yes

Yes

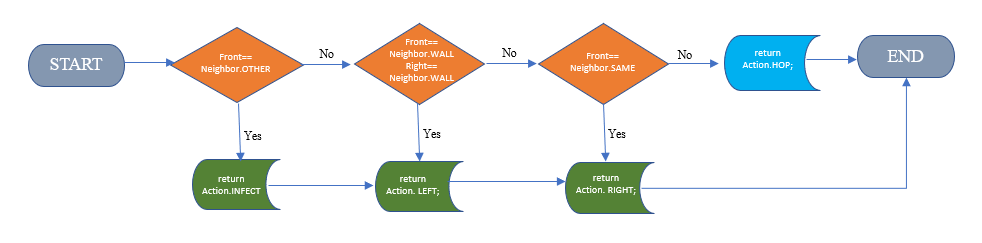
return

Action.HOP;

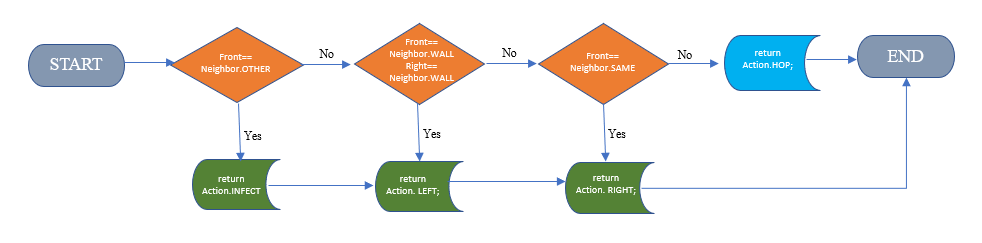
return

Action.INFECT

* **Tiger.java**
  + getMove()

****

* **WhiteTiger.java**
  + getMove()

****

* **Giant.java**
  + getMove()

START

END

Front==

Neighbor.OTHER

Front==

Neighbor.EMPTY

return

Action. RIGHT;

return

Action.INFECT

return

Action.HOP;

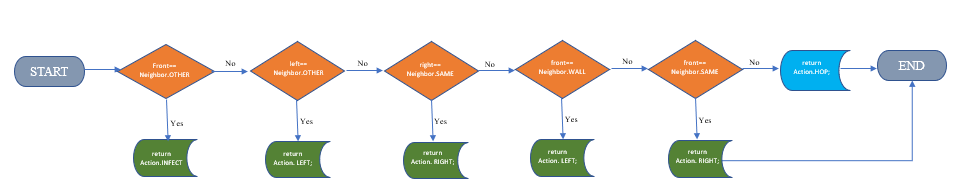
No

No

Yes

Yes

* **NinjaCat.java**
  + getMove()



**6.0. DEVELOPMENT DETAILS**

Total of 5 Classes Behavior given to build the source code of the project as below:

* **6.1 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. |

// \* Copyright (C) UNITAR - All Rights Reserved  
// \* FINAL PROJECT | Software Design and Development (May 2022)  
// \* Author:  
// \* DEVAN A/L THANGA SELVARAJ (UNU2200575)  
  
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() {  
 //Color.WHITE for a polar bear (when polar is true),  
 // Color.BLACK otherwise (when polar is false)  
 if (this.polar){  
 return Color.*WHITE*;  
 } else {  
 return Color.*BLACK*;  
 }  
 }  
  
 public String toString(){  
 //Should alternate on each different move between a slash character (/)  
 // and a backslash character () starting with a slash.  
 if (moves%2==0){  
 return "/";  
 } else {  
 return "\\";  
 }  
  
 }  
  
 public Action getMove(CritterInfo info){  
 //always infect if an enemy is in front, otherwise hop if possible, otherwise turn left.  
 moves++;  
 if(info.getFront()==Neighbor.*OTHER*){  
 return Action.*INFECT*;  
 } else if (info.getFront()==Neighbor.*EMPTY*){  
 return Action.*HOP*;  
 } else {  
 return super.getMove(info);  
 }  
 }  
  
}

* **6.2 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. |

// \* Copyright (C) UNITAR - All Rights Reserved  
// \* FINAL PROJECT | Software Design and Development (May 2022)  
// \* Author:  
// \* DEVAN A/L THANGA SELVARAJ (UNU2200575)  
// \*  
import java.awt.\*;  
import java.util.\*;  
  
  
public class Tiger extends Critter {  
 private int colorMoves;  
 Color tigerColor;  
 Random rand = new Random();  
  
 public Tiger(){  
 colorMoves=0;//1,2,3  
 getColor();  
 }  
  
 public Color 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.  
 if (colorMoves%3==0){ // set new color  
 int x=0;  
 while (x==0){  
 int i=rand.nextInt(3); //0.Red 1.Green 2.Black  
 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) {  
 //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.  
 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*;  
 }  
 }  
}

* **6.3 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. |

// \* Copyright (C) UNITAR - All Rights Reserved  
// \* FINAL PROJECT | Software Design and Development (May 2022)  
// \* Author:  
// \* DEVAN A/L THANGA SELVARAJ (UNU2200575)  
// \*  
import java.awt.\*;  
  
public class WhiteTiger extends Tiger {  
 boolean hasInfected;  
  
 public WhiteTiger(){  
 hasInfected=false;  
 }  
  
  
 public Color getColor() {  
 //Always Color.WHITE.  
 return Color.*WHITE*;  
 }  
  
  
 public String toString() {  
 //"tgr" if it hasn’t infected another Critter yet, “TGR” if it has infected.  
 if (hasInfected){  
 return super.toString();  
 } else {  
 return "tgr";  
 }  
 }  
  
  
 public Action getMove(CritterInfo info) {  
 //Same as a Tiger.  
 // Note: you’ll have to override this method to figure out if it has infected another Critter.  
 if (info.getFront()==Neighbor.*OTHER*){  
 hasInfected=true;  
 }  
 return super.getMove(info);  
  
 }  
}

* **6.4 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. |

// \* Copyright (C) UNITAR - All Rights Reserved  
// \* FINAL PROJECT | Software Design and Development (May 2022)  
// \* Author:  
// \* DEVAN A/L THANGA SELVARAJ (UNU2200575)  
// \*  
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() {  
 //"fee" for 6 moves, then "fie" for 6 moves, then "foe" for 6 moves, then "fum" for 6 moves, then repeat.  
 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) {  
 //always infect if an enemy is in front, otherwise hop if possible, otherwise turn right  
 //track moves  
 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++;  
 }  
 }  
}

* **6.5 NinjaCat**

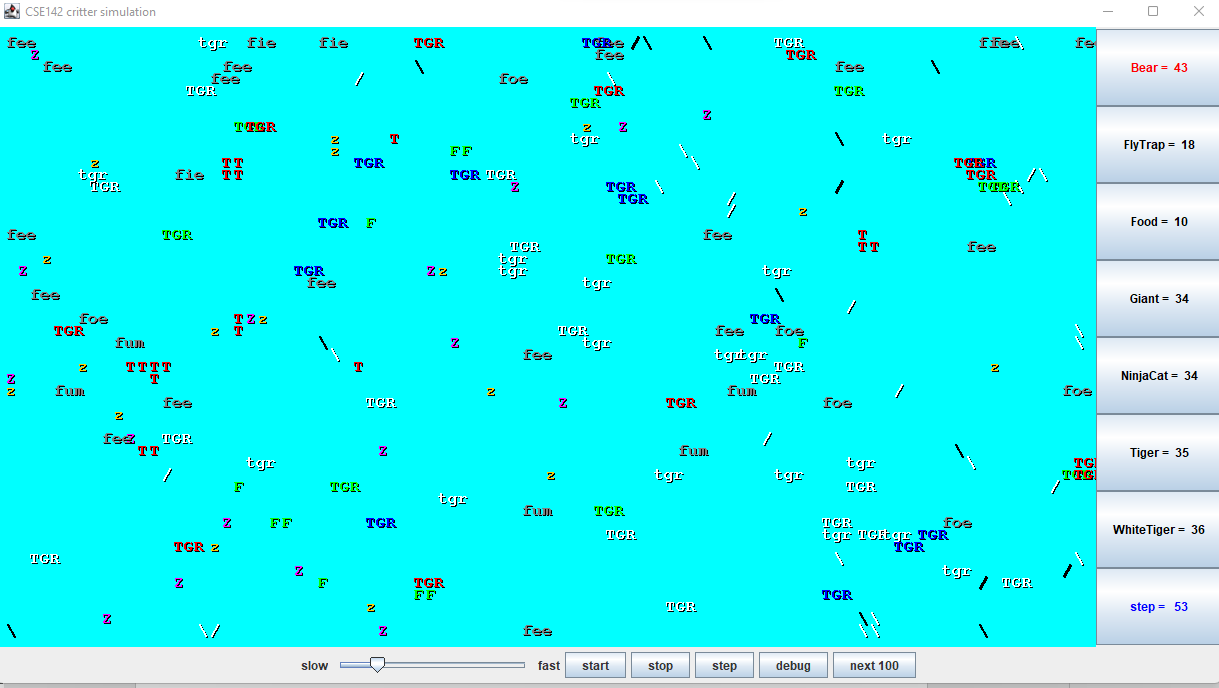
|  |  |
| --- | --- |
| Constructor | public NinjaCat() |
| getColor | Randomly picks one of three colors (Color.RED, Color.GREEN, Color.BLUE) and uses that color for three moves, then randomly picks one of those colors again for the next three moves, then randomly picks another one of those colors for the next three moves, and so on. |
| toString | “(=^.^=)” |
| 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 NinjaCat is in front, then turn right, otherwise hop. |

// \* Copyright (C) UNITAR - All Rights Reserved  
// \* FINAL PROJECT | Software Design and Development (May 2022)  
// \* Author:  
// \* DEVAN A/L THANGA SELVARAJ (UNU2200575)  
// \*  
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) {  
 //same as Tiger, but changes color when has infected  
 if (info.getFront()==Neighbor.*OTHER*){  
 hasInfected=true;  
 }  
 return super.getMove(info);  
  
 }  
}

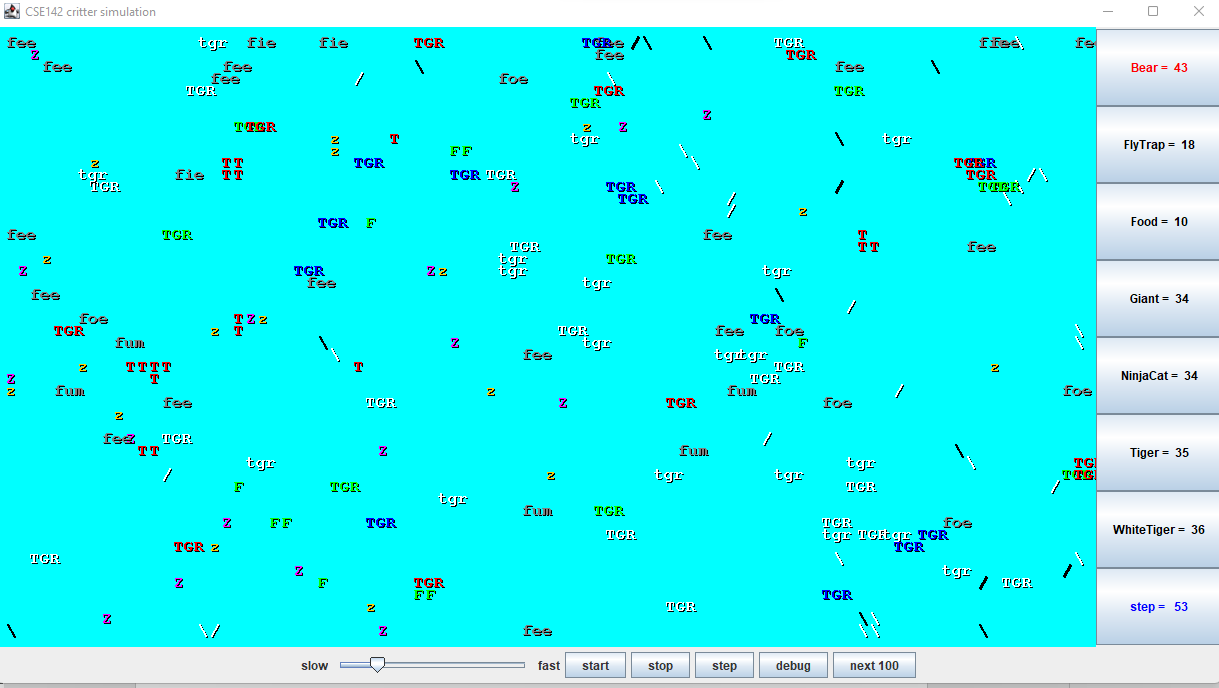
**7.0 RESULTS**

Building project and running in simulation mode for checking as below:-

All classes has been build in and is successfully running and the life status at the right hand dashboard. Please view the results. The spped function has been added to run the animal faster by toggle.



**7.1 LAYOUT**



**7.2 SCREEN FUNCTIONALITY**



|  |  |
| --- | --- |
| Screen | Layout Button Explanation |
| Speed | Toggle between slow/fast slider |
| Start | Begin the Game |
| Stop | Pause/Stop Completely |
| Debug | Stop game for checking/debugging |
| Next 100 | Display result of next 100 steps |

**7.3 Dashboard Right Side**

|  |  |
| --- | --- |
| Count of Life Remaning | Layout Button Explanation |
| 43 | Bear |
| 18 | FlyTrap |
| 10 | Food |
| 34 | Giant |
| 34 | NinjaCat |
| 35 | Tiger |
| 36 | White Tiger |

**8.0 DISCUSSION**

Tremendous challenges during the development of this projects. Program errors, coding mischaracters, typo errors, Java SDK issues. Preservarance and never give up attitude has procuded great result in completing this project.

Project produces good programming skills, good practices during writing codes, keep proper documentations, minimizing coded by having private and Private / Public Method or Classes.

During there final project it also given opputunity to promote analytical thinking in designing and developing a complete software, Furthermore it give you room for upgrading into new versions.

**9.0 CONCLUSION**

The exposure gained in this project to cultivate Object Oriented Programming (OOP) in using Java programming Languange. The expectations from the project is to grow in programming skils using OOP methods in Java Classes / Methods to develop Animal Kindom Game. Ew

I am glad to announce the journey has been completed and new route will begin soon in Software Design & Development. I have learned the core methodology in software desining and coding to bring my ideas into practical using java languang in very effective way.

This also opening new room for new ideas to take place in Application Solutioning in Inofrmation Technology. New venturing into promoting great ideas will always brings new dimentions to the world and community.

Last but not least, I would like to thank Dr.Simon Lau for wonderful lecturing and sharing of knowledge during the whole class. The guidance and help is commentable throughout the entire journey. Thanks UNITAR International University !