

CS 101: Introduction to Computer Science

Project Seven - Evil Genius Army

Student Name Mohammad El-Abid

Student ID

Professor Yasser Elleithy

Date 12/7/11

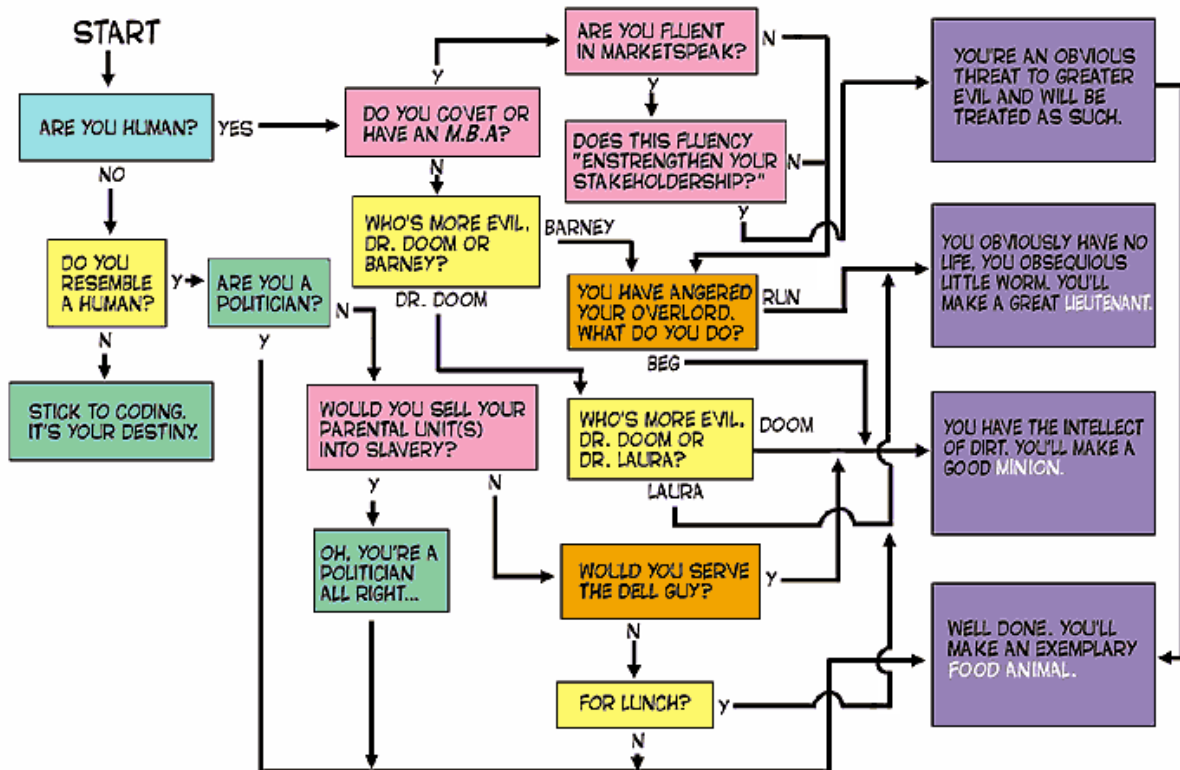
Table of Contents

<u>Abstract</u>	page 3
<u>Introduction</u>	page 3
<u>Screenshots</u>	page 4
<u>Code</u>	page 5-21
<u>Conclusion</u>	page 22
<u>Works Used</u>	page 22

Abstract

This application implements the flow chart suggested by UserFriendly.org for “[serving] an evil genius”. I used a node list for this since they allowed me to link questions to the next and to conclude at their “answer”.

THE 'HOW CAN I SERVE AN EVIL GENIUS' FLOWCHART



USERFRIENDLY DOT ORG

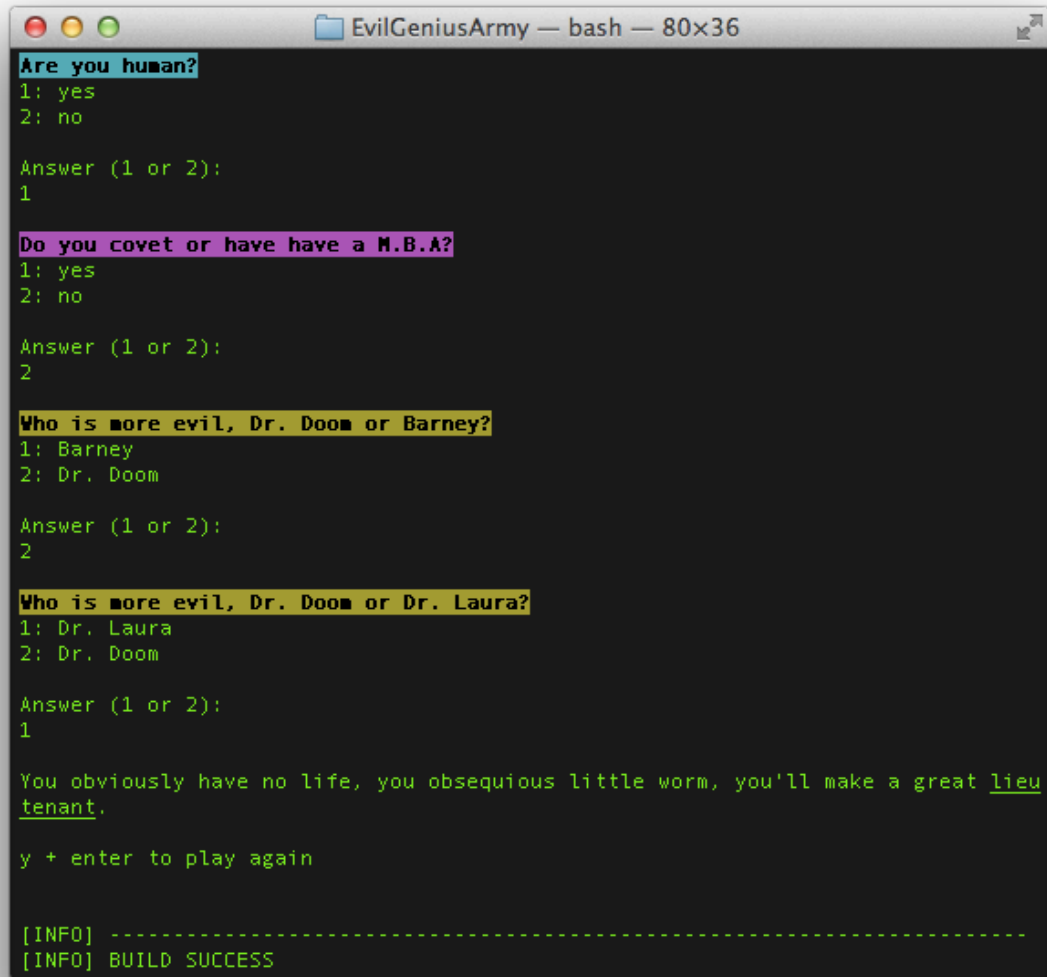
COPYRIGHT (C) 2002 ILLIAD

Introduction

I began working on this application by creating the QuestionNode and ConclusionNode classes. In order to link a QuestionNode to the ConclusionNode, I created a superclass and put any common methods or data, Node. Once the nodes were properly working and able to react to user input, I began working creating and linking all of the nodes.

Screenshots

Application output to stdout:



```
EvilGeniusArmy — bash — 80x36
Are you human?
1: yes
2: no

Answer (1 or 2):
1

Do you covet or have have a M.B.A?
1: yes
2: no

Answer (1 or 2):
2

Who is more evil, Dr. Doom or Barney?
1: Barney
2: Dr. Doom

Answer (1 or 2):
2

Who is more evil, Dr. Doom or Dr. Laura?
1: Dr. Laura
2: Dr. Doom

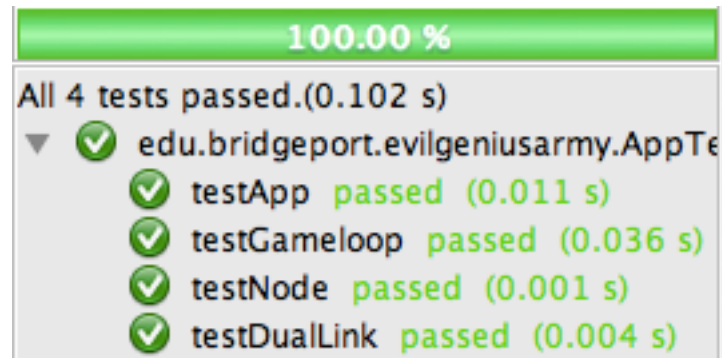
Answer (1 or 2):
1

You obviously have no life, you obsequious little worm, you'll make a great lieu
tenant.

y + enter to play again

[INFO] -----
[INFO] BUILD SUCCESS
```

Test Suite:



Application Code

File: App.java

```
package edu.bridgeport.evilgeniusarmy;

import java.util.Scanner;

/**
 * Contains the driver and any helpers the driver needs.
 */
public class App
{
    /**
     * No chance of the exception happening since it's
     initialized with a Scanner object.
     * @param args
     * @throws Exception The game doesn't have a scanner
     */
    public static void main( String[] args ) throws Exception
    {
        Scanner in = new Scanner(System.in);
        Game g = new Game(in);
        do {
            g.gameLoop();
            System.out.println("y + enter to play again");
        } while(playAgain(in));
    }

    /**
     * Used to easily determine when to exit or repeat.
     * @param in Scanner to read from
     */
}
```

```

        */
        private static boolean playAgain(Scanner in){
            String next = in.nextLine().trim();
            System.out.println();
            return(next.toLowerCase().equals("y"));
        }
    }
}

```

File: ConclusionNode.java

```

package edu.bridgeport.evilgeniusarmy;

/**
 * A node subclass for answers, used with "instanceof"
 */
public class ConclusionNode extends Node {

    /**
     * Default constructor
     */
    public ConclusionNode() {    }

    /**
     * Constructor that sets a text
     * @param newText The text
     */
    public ConclusionNode(String newText){
        setText(newText);
    }

    /**
     * Constructor that has a text and next node
     * @param newText The text
     * @param newNode The next node
     */
    public ConclusionNode(String newText, Node newNode){
        setText(newText);
        setNextNode(newNode);
    }

    @Override
    public boolean setNextNode(Node newNode) {
        return super.setNextNode(newNode);
    }

    @Override
    public boolean setText(String newText) {
        return super.setText(newText);
    }
}

```

```
}
```

File: Game.java

```
package edu.bridgeport.evilgeniusarmy;
import java.util.Scanner;

/**
 * What handles the flow of the nodes
 */
public class Game {
    /**
     * Scanner from constructor
     */
    private Scanner input;
    /**
     * Where the "player" is in the game
     */
    private Node currentNode;
    /**
     * When creating a game, where do we start from?
     */
    private Node firstNode;
    /**
     * Where should we output our data? Default to System.out
     */
    private java.io.PrintStream output = System.out;

    /**
     * Create a game that needs a Scanner
     */
    public Game(){
        seedData();
    }

    /**
     * Create a game that reads input from a Scanner
     * @param newInput
     */
    public Game(Scanner newInput){
        input = newInput;
        seedData();
    }

    /**
     * Set a new input source
     */
    public boolean setScanner(Scanner newScanner){
        input = newScanner;
    }
}
```

```

        return true;
    }

    /**
     * Set the output if not System.out
     */
    public boolean setOutput(java.io.PrintStream newOut){
        output = newOut;
        return true;
    }

    /**
     * The main loop of the game. Loops until currentNode is
null.
     * When currentNode is null, it means we've reached the end.
     * @throws Exception Scanner not attached
     */
    public void gameLoop() throws Exception {
        if(input == null){
            throw new Exception("Please attach a scanner to this
game object before use");
        }

        currentNode = firstNode;

        while(currentNode != null){
            while(currentNode instanceof QuestionNode){
                askQuestion();
            }

            while(currentNode instanceof ConclusionNode){
                displayAnswer();
            }
        }
    }

    private void outputln(String string){
        output.println(string);
    }

    private void output(String string){
        output.print(string);
    }

    /**
     * Ask question, get answer, move to next node.
     */
    private void askQuestion(){
        QuestionNode node = (QuestionNode) currentNode;

```



```

        outputln(node.buildQuestion());
        outputln("");

        outputln("Answer (1 or 2): ");
        while(!input.hasNextInt() && input.hasNext("[0-2]\\s*
$")){
            outputln("Please answer 1 or 2");
            output("Answer (1 or 2): ");
            input.nextLine();
        }
        // save answer
        int answer = input.nextInt();
        // move pointer to end of input
        input.nextLine();
        // Add a clear line
        outputln("");

        if(answer == 1){
            currentNode = node.getPositiveNode();
        } else {
            currentNode = node.getNegativeNode();
        }
    }

    /**
     * Outputs the an ConclusionNode and then moves to the next
     node (may be null).
     */
    private void displayAnswer(){
        outputln(currentNode.getText());
        outputln(""); // clear line
        // Some answers redirect to other answers as well
        currentNode = currentNode.getNextNode();
    }

    /**
     * Seeds game nodes and links the flow. Adds color in
     terminals that support it.
     */
    private void seedData(){
        // Sneaking in black foreground here
        String blackfg = "\033[30m";
        String underline = "\033[4m";
        String green = "\033[1;42m" + blackfg;
        String cyan = "\033[1;46m" + blackfg;
        String magneta = "\033[1;45m" + blackfg;
        String yellow = "\033[1;43m" + blackfg;
        String reset = "\033[0m";
    }

```

```

// col 1
QuestionNode humanNode = new QuestionNode(cyan + "Are you
human?" + reset, "yes", "no");
firstNode = humanNode;

QuestionNode resNode = new QuestionNode(yellow + "Do you
resemble a human?" + reset, "yes", "no");
humanNode.setNegativeNode(resNode);

ConclusionNode coder = new ConclusionNode(green + "Stick
to coding. It's your destiny." + reset);
resNode.setNegativeNode(coder);

// col 2
QuestionNode polit = new QuestionNode(green + "Are you a
politician?" + reset, "yes", "no");
resNode.setPositiveNode(polit);

// col 3
QuestionNode parents = new QuestionNode(magneta + "Would
you sell your parental unit(s) into slavery?" + reset, "yes",
"no");
polit.setNegativeNode(parents);
ConclusionNode liar = new ConclusionNode(green + "Oh,
you're a politician alright..." + reset);
parents.setPositiveNode(liar);

// col 3.5
QuestionNode mbaNode = new QuestionNode(magneta + "Do you
covet or have have a M.B.A?" + reset, "yes", "no");
humanNode.setPositiveNode(mbaNode);

QuestionNode evil1 = new QuestionNode(yellow + "Who is
more evil, Dr. Doom or Barney?" + reset, "Barney", "Dr. Doom");
mbaNode.setNegativeNode(evil1);

// col 4
QuestionNode market = new QuestionNode(magneta + "Are you
fluent in market speak?" + reset, "yes", "no");
mbaNode.setPositiveNode(market);

QuestionNode enstrength = new QuestionNode(magneta +
"Does this fluency \"enstrengthen your stakeholdership\"" +
reset, "yes", "no");
market.setPositiveNode(enstrength);

```

```

        QuestionNode angered = new QuestionNode(yellow + "You
have angered your overlord, what do you do?" + reset, "run",
"beg");
        evil1.setPositiveNode(angered);
        enstrength.setNegativeNode(angered);
        market.setNegativeNode(angered);

        QuestionNode evil2 = new QuestionNode(yellow + "Who is
more evil, Dr. Doom or Dr. Laura?" + reset, "Dr. Laura", "Dr.
Doom");
        evil1.setNegativeNode(evil2);

        QuestionNode dell = new QuestionNode(yellow + "Would you
serve the Dell guy?" + reset, "yes", "no");
        parents.setNegativeNode(dell);

        QuestionNode lunch = new QuestionNode(yellow + "For
lunch?" + reset, "yes", "no");
        dell.setNegativeNode(lunch);

        // col 5
        ConclusionNode genius = new ConclusionNode(underline +
"You're an obvious threat" + reset + " to greater evil and will
be treated as such.");
        enstrength.setPositiveNode(genius);

        ConclusionNode lieutenant = new ConclusionNode("You
obviously have no life, you obsequious little worm, you'll make a
great " + underline + "lieutenant" + reset + ".");
        angered.setPositiveNode(lieutenant);
        evil2.setPositiveNode(lieutenant);
        lunch.setPositiveNode(lieutenant);

        ConclusionNode minion = new ConclusionNode("You have the
intellect of dirt, you'll make a good " + underline + "minion" +
reset + ".");
        evil2.setNegativeNode(minion);
        dell.setPositiveNode(minion);
        angered.setNegativeNode(minion);

        ConclusionNode food = new ConclusionNode("Well done.
You'll make an exemplary " + underline + "food animal" + reset +
".");
        polit.setPositiveNode(food);
        lier.setNextNode(food);
        lunch.setNegativeNode(food);
        genius.setNextNode(food);
    }
}

```

File: Node.java

```
package edu.bridgeport.evilgeniusarmy;

/**
 * A String value node class
 */
public class Node {
    /**
     * Text/value of the node.
     */
    private String text;
    /**
     * The next node in the list.
     */
    private Node nextNode;

    public Node() { }

    public Node(String newText){
        setText(newText);
    }

    public Node(String newText, Node newNode){
        setText(newText);
        setNextNode(newNode);
    }

    public boolean setText(String newText){
        text = newText;
        return true;
    }

    public boolean setNextNode(Node newNode){
        nextNode = newNode;
        return true;
    }

    public boolean hasNextNode(){
        return nextNode != null;
    }

    public String getText(){
        return text;
    }

    public Node getNextNode(){
        return nextNode;
    }
}
```

```

        @Override
        public String toString(){
            return getText();
        }
    }
}

```

File: QuestionNode.java

```

package edu.bridgeport.evilgeniusarmy;

/**
 * A node for storing a question that has two available answers,
 * each answer can have its own "next node".
 */
public class QuestionNode extends Node {
    /**
     * What is the positive answer? ex: yes
     */
    private String positiveAnswer;
    /**
     * And the negative answer?
     */
    private String negativeAnswer;
    /**
     * Negative answer node
     */
    private Node negativeNode;
    /**
     * System breakline, commonly \n on *nix or \r on Windows
     */
    private static final String breakline =
System.getProperty("line.separator");

    public QuestionNode(){    }

    public QuestionNode(String newQuestion, String newPositive,
String newNegative){
        setQuestion(newQuestion);
        setPositiveAnswer(newPositive);
        setNegativeAnswer(newNegative);
    }

    public QuestionNode(String newQuestion, String newPositive,
String newNegative, Node positive, Node negative){
        setQuestion(newQuestion);
        setPositiveAnswer(newPositive);
        setNegativeAnswer(newNegative);
    }
}

```

```

        setPositiveNode(positive);
        setNegativeNode(negative);
    }

    /**
     * Display the question
     * @return formatted String
     */
    public String buildQuestion(){
        String construction =
            getQuestion() + breakline +
            "1: " + positiveAnswer + breakline +
            "2: " + negativeAnswer;
        return construction;
    }

    @Override
    public boolean hasNextNode(){
        return (getPositiveNode() != null) || (negativeNode !=
null);
    }

    public boolean hasNextNodes(){
        return (getPositiveNode() == null) || (negativeNode ==
null);
    }

    public String getQuestion(){
        return getText();
    }

    public boolean setQuestion(String newQuestion){
        return setText(newQuestion);
    }

    public boolean setPositiveAnswer(String newPositive){
        positiveAnswer = newPositive;
        return true;
    }

    public boolean setNegativeAnswer(String newNegative){
        negativeAnswer = newNegative;
        return true;
    }

    public Node getPositiveNode(){
        return getNextNode();
    }

```

```

    public boolean setPositiveNode(Node positive){
        setNextNode(positive);
        return true;
    }

    public Node getNegativeNode(){
        return negativeNode;
    }

    public boolean setNegativeNode(Node negative){
        negativeNode = negative;
        return true;
    }
}

```

Test Suite:

AppTest.java:

```

package edu.bridgeport.evilgeniusarmy;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PrintStream;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Ensure all application functions are working as expected
 */
public class AppTest extends TestCase {
    /**
     * Create the test case
     * @param testName name of the test case
     */
    public AppTest( String testName )
    {
        super( testName );
    }

    /**
     * @return the suite of tests being tested
     */
    public static Test suite()
    {
        TestSuite suite = new TestSuite(AppTest.class);
        suite.addTest(new TestSuite(GameTest.class));
    }
}

```

```

        suite.addTest(new TestSuite(NodeTest.class));
        suite.addTest(new TestSuite(QuestionNodeTest.class));
        return suite;
    }

    /**
     * Test that it reads the input and loops correctly
     */
    public void testApp()
    {
        ByteArrayOutputStream outputData = new
        ByteArrayOutputStream();
        PrintStream output = new PrintStream(outputData);
        PrintStream orgOut = System.out;
        System.setOut(output);

        String input = TestData.ani[0] + "y" + TestData.breakline
+ TestData.leu[0] + TestData.breakline;
        InputStream orgIn = System.in;
        System.setIn(new ByteArrayInputStream(input.getBytes()));

        String[] args = {""};

        try {
            App.main(args);

            assertTrue(outputData.toString().indexOf(TestData.animal) != -1);
            assertTrue(outputData.toString().indexOf(TestData.leut) != -1);
        } catch (Exception ex) {
            orgOut.println(ex);
        }

        System.setIn(orgIn);
        System.setOut(orgOut);
    }
}

```

GameTest.java:

```

package edu.bridgeport.evilgeniusarmy;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
import java.util.Scanner;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

```



```

public class GameTest extends TestCase {

    public GameTest(String testName) {
        super(testName);
    }

    /**
     * @return the suite of tests being tested
     */
    public static Test suite()
    {
        return new TestSuite( GameTest.class );
    }

    /**
     * Test that it flows as expected
     */
    public void testGameLoop()
    {
        Game game = new Game();
        ByteArrayOutputStream outputData = new
ByteArrayOutputStream();
        PrintStream output = new PrintStream(outputData);
        game.setOutput(output);

        // Stick to coding
        for(int i = 0; TestData.stickToCoding.length > i; i++){
            Scanner in = new Scanner(TestData.stickToCoding[i]);
            game.setScanner(in);
            try {
                game.gameLoop();

assertFalse(outputData.toString().indexOf(TestData.stick) == -1);
            } catch (Exception ex) {
                System.out.println(ex);
            }
        }

        // Threat
        for(int i = 0; TestData.threat.length > i; i++){
            Scanner in = new Scanner(TestData.threat[i]);
            game.setScanner(in);
            try {
                game.gameLoop();

assertFalse(outputData.toString().indexOf(TestData.genius) ==
-1);
            }
        }
    }
}

```

```

assertFalse(outputData.toString().indexOf(TestData.animal) ==
-1);
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }

    // Leutanit
    for (int i = 0; TestData.leu.length > i; i++){
        Scanner in = new Scanner(TestData.leu[i]);
        game.setScanner(in);
        try {
            game.gameLoop();
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }

    // Minon
    for (int i = 0; TestData.min.length > i; i++){
        Scanner in = new Scanner(TestData.min[i]);
        game.setScanner(in);
        try {
            game.gameLoop();
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }

    // Animal
    for (int i = 0; TestData.ani.length > i; i++){
        Scanner in = new Scanner(TestData.ani[i]);
        game.setScanner(in);
        try {
            game.gameLoop();
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }

    assertFalse(outputData.toString().indexOf(TestData.animal) ==
-1);
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}

```

```
}
```

NodeTest.java:

```
package edu.bridgeport.evilgeniusarmy;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class NodeTest extends TestCase {

    public NodeTest(String testName) {
        super(testName);
    }

    public static Test suite()
    {
        return new TestSuite( NodeTest.class );
    }

    public void testNode(){
        Node n1 = new Node("Hello World");
        Node n2 = new Node("Goodbye World");

        assertFalse(n1.hasNextNode());
        assertNull(n1.getNextNode());
        n1.setNextNode(n2);
        assertTrue(n1.hasNextNode());
        assertEquals(n1.getNextNode(), n2);

        assertEquals(n1.getText(), "Hello World");
    }
}
```

QuestionNodeTest.java:

```
package edu.bridgeport.evilgeniusarmy;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class QuestionNodeTest extends TestCase {

    public QuestionNodeTest(String testName) {
        super(testName);
    }

    public static Test suite()
```

```

    {
        return new TestSuite( QuestionNodeTest.class );
    }

    public void testDualLink(){
        QuestionNode qn1 = new QuestionNode("Is this a
question?", "yes", "no");
        QuestionNode qn2 = new QuestionNode("Are you a
question?", "yes", "no");
        ConclusionNode a1 = new ConclusionNode("You're pretty
smart!");
        ConclusionNode a2 = new ConclusionNode("You need some
help.");

        qn1.setPositiveNode(a1);
        qn1.setNegativeNode(qn2);
        qn2.setPositiveNode(a2);
        qn2.setNegativeNode(a1);

        assertEquals(qn1.getPositiveNode(), a1);
        assertEquals(qn1.getNegativeNode(), qn2);
    }
}

```

TestData.java:

```

package edu.bridgeport.evilgeniusarmy;

/**
 * Static strings for test data
 */
public class TestData {
    public static final String breakline =
System.getProperty("line.separator");

    public static String stick = "Stick to coding. It's your
destiny.";
    public static String genius = "You're an obvious threat";
    public static String leut = "You obviously have no life,
you obsequious little worm, you'll make a great";
    public static String minio = "You have the intellect of
dirt, you'll make a good";
    public static String animal = "Well done. You'll make an
exemplary";

    public static String[] stickToCoding = {
        "2" + breakline + "2" + breakline
    };
}

```

```

        public static String[] threat = {
            "1" + breakline + "1" + breakline + "1" + breakline + "1"
+ breakline
        };

        public static String[] leu = {
            "1" + breakline + "1" + breakline + "2" + breakline + "1"
+ breakline,
            "1" + breakline + "1" + breakline + "1" + breakline + "2"
+ breakline + "1" + breakline,
            "1" + breakline + "2" + breakline + "1" + breakline + "1"
+ breakline ,
            "1" + breakline + "2" + breakline + "2" + breakline + "1"
+ breakline,
            "2" + breakline + "1" + breakline + "2" + breakline + "2"
+ breakline + "2" + breakline + "1" + breakline
        };

        public static String[] min = {
            "1" + breakline + "2" + breakline + "1" + breakline + "2"
+ breakline,
            "1" + breakline + "2" + breakline + "2" + breakline + "2"
+ breakline,
            "1" + breakline + "1" + breakline + "2" + breakline + "2"
+ breakline,
            "1" + breakline + "1" + breakline + "1" + breakline + "2"
+ breakline + "2" + breakline,
            "2" + breakline + "1" + breakline + "2" + breakline + "2"
+ breakline + "1" + breakline
        };

        public static String[] ani = {
            "2" + breakline + "1" + breakline + "2" + breakline + "1"
+ breakline,
            "2" + breakline + "1" + breakline + "2" + breakline + "2"
+ breakline + "2" + breakline + "2" + breakline,
            "2" + breakline + "1" + breakline + "1" + breakline
        };
    }
}

```

Conclusion

The application was fun as it allowed me to implement nodes and try giving them dual links and keeping the user input out of the `QuestionNode`. It also was a challenge to write the tests. While I wanted to test it in a more pragmatic way, it would then defeat the purpose of the test, which set out to ensure the certain outputs on series of inputs. I also saw some classmates that used if and else statements to build the program and saw that their code was messy and harder to read. Using the nodes I was able to abstract the flowchart nodes into the `Game#seedData` and have the application handle the flow.

Works Used

No works used