```
.root {
    -fx-font-family: monospace;
    -fx-font-size: 11pt;
    -fx-background-color:  orange;
}

.label {
    -fx-text-fill: blue;
}

.text-area {
    -fx-background-color:  orange;
}
```

30/30 points for Program 4

As always, an awesome solution. Fun to run.

Tested with 11 zombies and a 7X7 grid compare and the program behaved with than expectations.

GUI excellent. Very cool.

```
/*********************************************************************************************
 *  This class contains stores the zombie positions and visited locations for the zombie application
 *
 *  Note: when thinking of a position position (i, j) - i represents the column and j represents the row
 *  similar to cartesian coordinates.
 *  e.g. (4,0) is the 5th element in the 1st row
 *
 *  Features:
 *  This class implements the creation, resetting and updating of the zombie location and visitation grids
 *
 *  It implements the zombie movement - for each step a zombie moves exactly one space in a random location
 *  and does not move off the grid
 *
 *  Based on user options, a grid may be displayed to the console displaying a combined grid of zombie
 *  positions and visited locations
 *
 *  CST 283 Programming Assignment 4
 *  @author Michael Clinesmith
 *********************************************************************************************/

import java.util.Random;

public class ZombieZone
{
    private int MAX_DIMENSION = 20, MIN_DIMENSION=3;
    private int widthDimension, heightDimension;
    private int[][] zombieGridCount = new int[MAX_DIMENSION][MAX_DIMENSION];
    private int[][] zombieGridTemp = new int[MAX_DIMENSION][MAX_DIMENSION];
    private boolean[][] zombieGridVisit = new boolean[MAX_DIMENSION][MAX_DIMENSION];
    private int steps=0;

    /**
     * No-argument constructor creates a default zombie zone of 10X10 dimension
     */
    public ZombieZone()
    {
        widthDimension=10;
        heightDimension=10;
        resetZombieGrid();
        resetZombieVisitGrid();
    }

    /**
     * Constructor that creates a zombie zone of size width X height
     * @param width int: the width of the created zone
     * @param height int: the height of the created zone
     */
    public ZombieZone(int width, int height)
    {
        widthDimension=width;
        heightDimension=height;
        // Keep array dimensions in bounds
        if (width<MIN_DIMENSION)
        {
            widthDimension=MIN_DIMENSION;
        }
        if (height<MIN_DIMENSION)
        {
            heightDimension=MIN_DIMENSION;
        }
```

```
    if (width>MAX_DIMENSION)
    {
        widthDimension=MAX_DIMENSION;
    }
    if (height>MAX_DIMENSION)
    {
        heightDimension=MAX_DIMENSION;
    }
    resetZombieGrid();
    resetZombieVisitGrid();
}

/**
 * Constructor that creates a zombie zone of size width X height with zombies in the center
 * @param width int: the width of the created zone
 * @param height int: the height of the created zone
 * @param zombieNumber int: The number of zombies to place in the center of the zone
 */
public ZombieZone(int width, int height, int zombieNumber)
{
    widthDimension=width;
    heightDimension=height;
    // Keep array dimensions in bounds
    if (width<MIN_DIMENSION)
    {
        widthDimension=MIN_DIMENSION;
    }
    if (height<MIN_DIMENSION)
    {
        heightDimension=MIN_DIMENSION;
    }
    if (width>MAX_DIMENSION)
    {
        widthDimension=MAX_DIMENSION;
    }
    if (height>MAX_DIMENSION)
    {
        heightDimension=MAX_DIMENSION;
    }
    resetZombieGrid(zombieNumber);
    resetZombieVisitGrid();
}

/**
 * Method to get the number of zombies currently located at position (x,y)
 * @param x int: x coordinate position
 * @param y int: y coordinate position
 * @return int: The number of zombies located at position (x,y)
 */
public int getZombiesAt(int x, int y)
{
    return zombieGridCount[x][y];
}

/**
 * Method to get if the position (x,y) has been visited by zombies
 * @param x int: x coordinate position
 * @param y int: y coordinate position
 * @return int: true if the position (x,y) has been visited by zombies, false otherwise
 */
```

```
public boolean getVisitedAt(int x, int y)
{
    return zombieGridVisit[x][y];
}

/**
 * Accessor method to get the width of the current grid
 * @return int: The width of the grid (number of columns)
 */
public int getWidth()
{
    return widthDimension;
}

/**
 * Accessor method to get the height of the current grid
 * @return int: The height of the grid (number of rows)
 */
public int getHeight()
{
    return heightDimension;
}

/**
 * Accessor method to get the number of steps taken in the grid
 * @return
 */
public int getSteps()
{
    return steps;
}

/**
 * Mutator method to set the width of the current grid
 * @param widthDimension int: The width of the grid (number of columns)
 */
public void setWidth( int widthDimension )
{
    this.widthDimension = widthDimension;

    if (widthDimension<MIN_DIMENSION)       // make sure widthDimension stays within bounds
    {
        this.widthDimension=MIN_DIMENSION;
    }
    if (widthDimension>MAX_DIMENSION)
    {
        this.widthDimension=MAX_DIMENSION;
    }
}

/**
 * Mutator method to set the height of the current grid
 * @param heightDimension int: The height of the grid (number of rows)
 */
public void setHeight( int heightDimension )
{
    this.heightDimension = heightDimension; // make sure heightDimension stays within bounds

    if (heightDimension<MIN_DIMENSION)
    {
```

```
        this.heightDimension=MIN_DIMENSION;
    }
    if (heightDimension>MAX_DIMENSION)
    {
        this.heightDimension=MAX_DIMENSION;
    }
}

/**
 * Mutator to set the number of steps taken
 * @param steps int: The number of steps taken
 */
public void setSteps( int steps )
{
    this.steps = steps;
}

/**
 * Method to set the number of zombies located at position (x,y)
 * @param x int: x coordinate position
 * @param y int: y coordinate position
 * @param zombieCount int: the number of zombies to set at position (x,y)
 */
public void setZombiesAt(int x, int y, int zombieCount)
{
    if (zombieCount>=0)          // only set value if zombieCount is at least 0
    {
        zombieGridCount[x][y] = zombieCount;
    }
}

/**
 * Method to set if a zombie has visited the position (x,y) or not
 * @param x int: x coordinate position
 * @param y int: y coordinate position
 * @param isVisited boolean: set if a zombie has visited a position or not
 */
public void setVisitedAt(int x, int y, boolean isVisited)
{
    zombieGridVisit[x][y]=isVisited;
}

/**
 *  Method that creates a new zombie grid based on currently set values for widthDimension and heightDimension
 *  Also resets the number of steps
 */
public void resetZombieGrid()
{
    for (int i=0; i<MAX_DIMENSION; i++)
    {
        for (int j=0; j<MAX_DIMENSION; j++ )
        {
            zombieGridCount[i][j]=0;
        }
    }
    steps=0;
}

/**
 * Method that creates a new zombie grid based on currently set values for widthDimension and heightDimension
```

```
 * and places zombies in the center of the grid
 * @param zombieNumber int: the number of zombies to place in the grid
 */
public void resetZombieGrid( int zombieNumber)
{
    resetZombieGrid();

    // put zombies in middle if the value is greater than 0
    if (zombieNumber>0)
    {
        zombieGridCount[widthDimension / 2][heightDimension / 2] = zombieNumber;
        zombieGridVisit[widthDimension / 2][heightDimension / 2] = true;
    }
}

/**
 * Method that creates a new zombie visiting field based on currently set values for the ZombieZone object
 */
public void resetZombieVisitGrid()
{
    for (int i=0; i<MAX_DIMENSION; i++)
    {
        for (int j=0; j<MAX_DIMENSION; j++ )
        {
            zombieGridVisit[i][j]=false;
        }
    }
    if(zombieGridCount[widthDimension/2][heightDimension/2]>0) // check if zombies in middle of grid
    {
        zombieGridVisit[widthDimension / 2][heightDimension / 2] = true;
    }
}

/**
 * Method to update the ZombieGridVisit array to account for moved zombies
 */
public void updateZombieVisitGrid()
{
    for (int i=0; i<widthDimension; i++)
    {
        for (int j=0; j<heightDimension; j++ )
        {
            if(zombieGridCount[i][j]>0)
            {
                zombieGridVisit[i][j] = true;
            }
        }
    }
}

/**
 * Metheod to add a zombie at the given position
 * @param x int: x coordinate position
 * @param y int: y coordinate position
 */
public void addZombieAt(int x, int y)
{
    zombieGridCount[x][y]++;
    zombieGridVisit[x][y]=true;
}
```

```
/**
 * Method to remove a zombie at the given position
 * The method does not allow the number of zombies to become less than 0
 * @param x int: the x coordinate position
 * @param y int: the y coordinate position
 */
public void removeZombieAt(int x, int y)
{
    zombieGridCount[x][y]--;
    if (zombieGridCount[x][y]<0)
    {
        zombieGridCount[x][y]=0;
    }
}

/**
 * Method that counts the number of zombies in the grid
 * @return int: The total number of zombies in the grid
 */
public int zombieTotal()
{
    int total=0;
    for (int i=0; i<widthDimension; i++)
    {
        for (int j=0; j<heightDimension; j++ )
        {
            total += zombieGridCount[i][j];
        }
    }
    return total;
}

/**
 * Method to reset the zombie grid with new dimensions
 * @param width int: The number of columns in the grid
 * @param height int: the number of rows in the grid
 * @param zom int: The number of zombies to put in the middle of the grid
 */
public void newZombieGrid(int width, int height, int zom)
{
    // Keep array dimensions in bounds
    if (width<MIN_DIMENSION)
    {
        widthDimension=MIN_DIMENSION;
    }
    if (height<MIN_DIMENSION)
    {
        heightDimension=MIN_DIMENSION;
    }
    if (width>MAX_DIMENSION)
    {
        widthDimension=MAX_DIMENSION;
    }
    if (height>MAX_DIMENSION)
    {
        heightDimension=MAX_DIMENSION;
    }
    resetZombieGrid(zom);
    resetZombieVisitGrid();
```

```
    steps=0;
}

/**
 * Method ot reset the temporary grid used to move zombies
 */
public void resetTempGrid()
{
    for (int i=0; i<MAX_DIMENSION; i++)
    {
        for (int j=0; j<MAX_DIMENSION; j++ )
        {
            zombieGridTemp[i][j]=0;
        }
    }
}

/**
 * Method to move all the zombies in the grid
 * The zombies will move exactly one space and may not move off the grid
 * It will increase the number of steps by one.
 * @param isToConsole boolean: true is the grid is to be displayed to the console, false if not.
 */
public void oneStep(boolean isToConsole)    √
{
    resetTempGrid();
    boolean isValidMove;
    int moveX=0, moveY=0;
    int randomValue;
    Random randomNumbers = new Random(  );
    steps++;

    // loops to move zombies to temp grid
    for (int i=0; i<widthDimension; i++)
    {
        for (int j=0; j<heightDimension; j++ )
        {
            for (int k=0; k<zombieGridCount[i][j]; k++) // loop once for each zombie
            {
                isValidMove=false;
                while (!isValidMove) // loop until zombie moves to a valid location
                {
                    moveX = i;
                    moveY = j;
                    randomValue = randomNumbers.nextInt( 4 );
                    switch (randomValue)
                    {
                        case 0:
                            moveX++;        √
                            break;
                        case 1:
                            moveY++;
                            break;
                        case 2:
                            moveX--;
                            break;
                        case 3:
                            moveY--;
                            break;
                    }
```

```
                    if (moveX >= 0 && moveX < widthDimension && moveY >= 0 && moveY < heightDimension) // check if valid move
                    {
                        isValidMove = true;
                    }
                }
                zombieGridTemp[moveX][moveY]++;
            }
        }
    }
    // loops to copy temp grid to real grid
    for (int i=0; i<widthDimension; i++)
    {
        for (int j=0; j<heightDimension; j++ )
        {
            zombieGridCount[i][j]=zombieGridTemp[i][j];
        }
    }
    // update visit grid
    updateZombieVisitGrid();

    // display to console if true
    if(isToConsole)
    {
        displayGrid();
    }

}

/**
 * Method to check is all locations in the grid have been visited by zombies
 * @return
 */
public boolean isAllVisited()
{
    boolean isAllVisited=true;
    for (int i=0; i<widthDimension && isAllVisited; i++)
    {
        for (int j=0; j<heightDimension && isAllVisited; j++ )
        {
            if (!zombieGridVisit[i][j]) // if a position has not been visited
            {
                isAllVisited=false;
            }
        }
    }
    return isAllVisited;
}

/**
 * Method to display a combination of the grids to the console
 * The grid has a border and prints the number of zombies in each location
 * If there is no zombie in a location, it will display an "X" for visited locations
 * or a blank for unvisited locations
 */
public void displayGrid()
{
    // first line
    System.out.print("|");
    for(int i=0; i<widthDimension; i++)
    {
```

```
            System.out.print("-");
        }
        System.out.print("|\n");
        for(int j=0; j<heightDimension; j++) // each iteration of j loop prints a row of the grid
        {
            System.out.print("|");
            for (int i=0; i<widthDimension; i++)
            {
                if (zombieGridCount[i][j]>9)        // if many zombies, display Z
                {
                    System.out.print( "Z" );
                }
                else if(zombieGridCount[i][j]>0)    // display the number of zombies
                {
                    System.out.print( zombieGridCount[i][j] );
                }
                else if(zombieGridVisit[i][j])      // if no zombies but visited, display "x"
                {
                    System.out.print("x");
                }
                else                                // otherwise display blank space
                {
                    System.out.print(" ");
                }
            }
            System.out.print("|\n");
        }
        // last line of grid
        System.out.print("|");
        for(int i=0; i<widthDimension; i++)
        {
            System.out.print("-");
        }
        System.out.print("|\n");
        // list step number then 2 new lines
        System.out.println( "Step: "+steps +"\n");
    }
}
```

```
/*********************************************************************************************
 *  This class contains the main driver and interface for viewing a zombie scenario
 *
 *  Note: when thinking of a position position (i, j) - i represents the column and j represents the row
 *  similar to cartesian coordinates.
 *  e.g. (4,0) is the 5th element in the 1st row
 *
 *  Features:
 *  The user is able to adjust the size of the grid, going from 3X3 up to 20X20, also being able to
 *  adjust the height and width independently of each other
 *
 *  The user can easily reset the board, clear the board of zombies
 *  Zombies can be placed and removed from the gridboard, also locations can be set to visited or
 *  not visited if desired
 *
 *  The simulation can be run in its entirity, or one step can be taken at a time.
 *
 *  The user can choose whether to steps should be displayed to the console or not.
 *
 *  The number of steps taken to run a simulation is displayed in the bottom of the interface
 *
 *  CST 283 Programming Assignment 4
 *  @author Michael Clinesmith
 *********************************************************************************************/

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import java.util.Optional;

public class ZombieInterface extends Application
{
    // important overall objects
    private ZombieZone zone;
    private BorderPane mainLayout;
    private final int MAX_ROWS=20, MAX_COLUMNS=20;
    private ButtonHandler aButtonHandler = new ButtonHandler();

    // title objects
    private Label title;
    private Label creator;
    private Button quitButton;
    private HBox titleHBox;

    // copyright label
    private String COPYRIGHT_WEB = "<div>Icons made by <a href=\"https://www.flaticon.com/authors/freepik\" title=\"Freepik\">Freepik</a> from <
    private String COPYRIGHT = "Icons made by Freepik from \"https://www.flaticon.com/\"";

    // zombie grid
    private GridPane gridPane;
    private Button[][] tileButton = new Button[MAX_ROWS][MAX_COLUMNS];
    private int tileRowsCount=10, tileColumnsCount=10, centerZombies=4;
```

```
private VBox zombieVBox;
private Image zombieHeadImage;
private ImageView[][] zombieHeadImageView =  new ImageView[MAX_ROWS][MAX_COLUMNS];

// dimension setting objects
private Slider widthSlider, heightSlider;
private Label widthLabel, heightLabel, initialZombieLabel;
private VBox widthVBox, heightVBox, initialZombieVBox;
private HBox widthHeightHBox;
private TextField initialZombie;

// buttons
private Button resetButton, clearButton;
private HBox buttonHBox;
private VBox optionsVBox;
private ComboBox<String> addDeleteZombies;

// simulation controls
private CheckBox toConsoleCheckBox;
private Button simulationButton, stepButton;
private VBox simulationVBox;
private Label stepCount, stepCountLabel;
private HBox simulationControlsHBox;
private TextArea copyright;
private VBox bottomVBox;

/**
 * Starting method of application - calls launch
 * @param args String[]: Not used
 */
public static void main( String[] args )
{
    // Launch the application.
    launch( args );
}

/**
 * Method that calls the initializeScene method and creates the scene
 * @param primaryStage Stage object used to create the stage
 */
@Override
public void start( Stage primaryStage )
{
    zone = new ZombieZone(tileColumnsCount, tileRowsCount, centerZombies);
    initializeScene();

    // Set up overall scene
    Scene scene = new Scene( mainLayout, 1100, 900 );
    scene.getStylesheets().add( "Zombie.css" );
    primaryStage.setScene( scene );
    primaryStage.setTitle( "Zombie Viewer" );
    primaryStage.show();
}

/**
 * Method that calls other methods to create the interface then puts the pieces into the mainLayout
 */
public void initializeScene()
{
    initializeImages();
```

```
    initializeTitle();
    initializeSimulationControls();
    initializeTileGrid();
    initializeButtons();
    initializeSliders();

    mainLayout = new BorderPane(  );
    mainLayout.setCenter( zombieVBox );
    mainLayout.setTop( optionsVBox );
    mainLayout.setBottom( bottomVBox );
}

/**
 * Method that initializes the images used in the interface, creating a two dimensional array of ImageViews
 */
public void initializeImages()
{
    zombieHeadImage = new Image("file:zombie.png");
    for (int i=0; i<MAX_ROWS; i++)
    {
        for (int j=0; j<MAX_COLUMNS; j++)
        {
            zombieHeadImageView[i][j] = new ImageView(zombieHeadImage );
            zombieHeadImageView[i][j].setFitWidth( 20 );
            zombieHeadImageView[i][j].setPreserveRatio( true );
        }
    }
}

/**
 * Method to initialize the objects part of the title/top bar on the interface
 */
public void initializeTitle()
{
    title = new Label( "Zombie Tracer" );
    title.setStyle( "-fx-font-size: 24pt; -fx-text-fill: black;" );

    creator = new Label( "created by Michael Clinesmith" );

    quitButton = new Button( "Quit" );
    quitButton.setOnAction( aButtonHandler );
    quitButton.setAlignment( Pos.CENTER );
    quitButton.setPadding( new Insets( 20 ) );

    titleHBox= new HBox( 50, title, creator, quitButton );
    titleHBox.setAlignment( Pos.CENTER );

}

/**
 * Method that creates the simulation objects at the bottom of the interface
 */
public void initializeSimulationControls()
{
    simulationButton = new Button( "Simulate Zombies" );
    simulationButton.setOnAction( aButtonHandler );
    simulationButton.setAlignment( Pos.CENTER );
    simulationButton.setPadding( new Insets( 10 ) );

    stepButton = new Button( "One Step" );
```

```
    stepButton.setOnAction( aButtonHandler );
    stepButton.setAlignment( Pos.CENTER );
    stepButton.setPadding( new Insets( 10 ) );

    simulationVBox = new VBox( 10, simulationButton, stepButton );
    simulationVBox.setAlignment( Pos.CENTER );

    toConsoleCheckBox = new CheckBox( "Display steps to console" );

    stepCountLabel = new Label("Step Count:");
    stepCount = new Label("0");

    copyright = new TextArea( COPYRIGHT );
    copyright.setEditable( false );
    copyright.setMaxSize( 600,50 );
    copyright.setWrapText( true );

    simulationControlsHBox = new HBox( 10, simulationVBox, toConsoleCheckBox,
            stepCountLabel, stepCount);
    simulationControlsHBox.setAlignment( Pos.CENTER );

    bottomVBox = new VBox( 10, simulationControlsHBox, copyright );
    bottomVBox.setPadding( new Insets( 10 ) );
    bottomVBox.setAlignment( Pos.CENTER );
}

/**
 * Method that creates all the buttons zombie grid then places them into a gridPane
 */
public void initializeTileGrid()
{
    gridPane = new GridPane();

    // creates all possible grid buttons
    for (int i=0; i<MAX_ROWS; i++)
    {

        for (int j=0; j<MAX_COLUMNS; j++)
        {
            tileButton[i][j]=new Button( );
            tileButton[i][j].setMinSize( 65,35 );
            tileButton[i][j].setMaxSize( 65,35 );
            tileButton[i][j].setOnAction( aButtonHandler );
            tileButton[i][j].setAlignment( Pos.CENTER );

            gridPane.add(tileButton[i][j], i, j);
        }
    }
    gridPane.setAlignment( Pos.CENTER );
    // set grid to default
    resetGrid();

    zombieVBox = new VBox( 20, gridPane);
    zombieVBox.setAlignment( Pos.CENTER );
    zombieVBox.setPadding( new Insets( 10 ) );
}

/**
 * Method that creates the buttons in the third row in the interface
 */
```

```
public void initializeButtons()
{
    resetButton = new Button( "Reset Grid" );
    resetButton.setOnAction( aButtonHandler );
    resetButton.setAlignment( Pos.CENTER );
    resetButton.setPadding( new Insets( 20 ) );

    clearButton = new Button( "Clear Board" );
    clearButton.setOnAction( aButtonHandler );
    clearButton.setAlignment( Pos.CENTER );
    clearButton.setPadding( new Insets( 20 ) );

    addDeleteZombies = new ComboBox<String>();
    addDeleteZombies.getItems().addAll( "Add Zombie", "Delete Zombie", "Set isVisited", "Set isNotVisited" );
    addDeleteZombies.setValue( "Adjust Zombies" );

    buttonHBox = new HBox( 20, resetButton, clearButton, addDeleteZombies );
    buttonHBox.setAlignment( Pos.CENTER );
    buttonHBox.setPadding( new Insets( 20 ) );

}

/**
 * Method that creates the objects in the second row of the interface, the sliders, zombie count
 */
public void initializeSliders()
{
    widthLabel = new Label("Columns");
    heightLabel = new Label("Rows");

    widthSlider = new Slider(3, 20, tileColumnsCount);
    widthSlider.setShowTickMarks( true );
    widthSlider.setShowTickLabels(true);
    widthSlider.setMajorTickUnit( 1.0 );
    widthSlider.setMinorTickCount( 0 );
    widthSlider.setSnapToTicks(true);
    widthSlider.setPrefWidth( 300 );
    widthVBox = new VBox (10, widthLabel, widthSlider);
    widthVBox.setAlignment( Pos.CENTER );

    heightSlider = new Slider(3, 20, tileRowsCount);
    heightSlider.setShowTickMarks(true);
    heightSlider.setShowTickLabels(true);
    heightSlider.setMajorTickUnit( 1.0 );
    heightSlider.setMinorTickCount( 0 );
    heightSlider.setSnapToTicks( true );
    heightSlider.setPrefWidth( 300 );
    heightVBox = new VBox (10, heightLabel, heightSlider);
    heightVBox.setAlignment( Pos.CENTER );

    initialZombieLabel= new Label("Initial Zombies");
    initialZombie = new TextField( "4" );
    initialZombieVBox = new VBox( 10, initialZombieLabel, initialZombie );
    initialZombieVBox.setAlignment( Pos.CENTER );

    widthHeightHBox = new HBox( 20, widthVBox, heightVBox, initialZombieVBox);
    widthHeightHBox.setAlignment( Pos.CENTER );
    optionsVBox= new VBox(20, titleHBox, widthHeightHBox, buttonHBox);
    optionsVBox.setAlignment( Pos.CENTER );
}
```

```
/**
 * Method that resets the zombie grid
 * It calls methods to set all zombie counts to 0, sets all areas to not visited,
 * places zombies in the center of the grid, then resets the grid in case the dimensions have changed
 */
public void resetGrid()
{
    zone.resetZombieGrid( centerZombies );
    zone.resetZombieVisitGrid();
    // clears grid and replaces grid buttons
    gridPane.getChildren().clear();
    updateGridLabels();

    for (int i=0; i<tileColumnsCount; i++)
    {
        for (int j=0; j<tileRowsCount; j++)
        {
            gridPane.add(tileButton[i][j], i, j);
        }
    }
    updateVisited();
    stepCount.setText( Integer.toString( zone.getSteps() ) );
}

/**
 *  Method to update the zombie grid based on zombie movement, marking locations visited by zombies
 *  and updating the step count
 */
public void updateGrid()
{
    updateGridLabels();
    updateVisited();
    stepCount.setText( Integer.toString( zone.getSteps() ) );
}

/**
 * Method to update the button labels based on number of zombies occupying
 */
public void updateGridLabels()
{
    for (int i=0; i<tileColumnsCount; i++)
    {
        for (int j=0; j<tileRowsCount; j++)
        {
            if (zone.getZombiesAt( i,j )==0)        // no zombies
            {
                tileButton[i][j].setGraphic( null );
                tileButton[i][j].setText( "0" );
            }
            else if (zone.getZombiesAt( i,j )>99)   // too many zombies, so just display number
            {
                tileButton[i][j].setGraphic( null );
                tileButton[i][j].setText( Integer.toString( zone.getZombiesAt( i,j ) ) );
            }
            else
            {
                tileButton[i][j].setGraphic( zombieHeadImageView[i][j] );
                tileButton[i][j].setText( Integer.toString( zone.getZombiesAt( i,j ) ) );
            }
```

```
        }
    }
}

/**
 * Method that sets the grid button color based on if a location has been visited or not
 */
public void updateVisited()
{
    for (int i=0; i<tileColumnsCount; i++)
    {
        for (int j=0; j<tileRowsCount; j++)
        {
            if(zone.getVisitedAt( i,j ))
            {
                tileButton[i][j].setStyle( "-fx-background-color:RED" );
            }
            else
            {
                tileButton[i][j].setStyle( "-fx-background-color:LIGHTGRAY" );
            }
        }
    }
}

/**
 * Method to simulate the zombie movement until all locations have been visited by zombies,
 * then it will stop and display the final zombie positions and step count to the interface
 */
public void simulateZombies()
{
    zone.oneStep(toConsoleCheckBox.isSelected());
    if (zone.zombieTotal()>0)   // do not simulate if there are no zombies
    {
        while(!zone.isAllVisited())
        {
            zone.oneStep(toConsoleCheckBox.isSelected());
        }
    }
    updateGrid();
    stepCount.setText( Integer.toString( zone.getSteps() ) );

}

/**
 * Class that handles ActionEvents for setting handling zombies
 */
class ButtonHandler implements EventHandler<ActionEvent>
{
    /**
     * Method that handles ActionEvents for the new game and quit buttons
     * @param event ActionEvent: Event caused by clicking the new game and quit buttons
     */
    @Override
    public void handle( ActionEvent event )
    {
        String message;

        if (event.getSource() == resetButton)      // user chooses to reset the zombie grid
        {
```

```
    message = "Do you want to reset the grid?";

    Alert alert = new Alert( Alert.AlertType.CONFIRMATION );
    alert.setTitle( "Reset Grid" );
    alert.setContentText( message );
    Optional<ButtonType> result = alert.showAndWait();
    if (result.get() == ButtonType.OK)
    {
        tileColumnsCount= (int)widthSlider.getValue();
        tileRowsCount = (int)heightSlider.getValue();
        zone.setWidth( tileColumnsCount );
        zone.setHeight( tileRowsCount );
        try
        {
            centerZombies = Integer.parseInt( initialZombie.getText() ); // get number of initial zombies
        }
        catch(NumberFormatException e) // catch if format problem and set to 0
        {
            initialZombie.setText("0");
            centerZombies=0;
        }
        resetGrid();
    }
}
else if (event.getSource() == clearButton)      // user chooses to clear of grid of zombies and visits
{
    message = "Do you want to clear the grid?";

    Alert alert = new Alert( Alert.AlertType.CONFIRMATION );
    alert.setTitle( "Clear Grid" );
    alert.setContentText( message );
    Optional<ButtonType> result = alert.showAndWait();
    if (result.get() == ButtonType.OK)
    {
        zone.resetZombieGrid( 0 );
        zone.resetZombieVisitGrid();
        updateGrid();
    }
}
else if (event.getSource() == quitButton)           // user chooses to quit
{
    message = "Do you want to quit the game?";

    Alert alert = new Alert( Alert.AlertType.CONFIRMATION );
    alert.setTitle( "Quit?" );
    alert.setContentText( message );
    Optional<ButtonType> result = alert.showAndWait();
    if (result.get() == ButtonType.OK)
    {
        System.exit( 0 );
    }
}
else if (event.getSource() == simulationButton)     // user chooses to simulate the zombie movement
{
    simulateZombies();
}
else if (event.getSource() == stepButton)           // user chooses to simulate one step of zombie movement
{
    zone.oneStep(toConsoleCheckBox.isSelected());
    updateGrid();
```

```
            }

            for (int i=0; i<tileColumnsCount; i++)              // check if any of the grid buttons was clicked
            {
                for (int j=0; j<tileRowsCount; j++)
                {
                    if (event.getSource()==tileButton[i][j])
                    {
                        if(addDeleteZombies.getValue().equals( "Add Zombie" ))
                        {
                            zone.addZombieAt( i, j );
                        }
                        else if(addDeleteZombies.getValue().equals( "Delete Zombie" ))
                        {
                            zone.removeZombieAt( i, j );
                        }
                        else if(addDeleteZombies.getValue().equals( "Set isVisited" ))
                        {
                            zone.setVisitedAt( i,j,true );
                        }
                        else if(addDeleteZombies.getValue().equals( "Set isNotVisited" ))
                        {
                            zone.setVisitedAt( i,j,false );
                        }
                        updateGrid();
                    }
                }
            }
        }
    }
}
```