Zhengmao Ouyang

ICS4U0 Final Project

BadIceCreamFour Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - BadIceCreamFour (main) class.
 */

package UFinal;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.InputStreamReader;

import javax.swing.JFrame;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;
import javax.swing.Timer;

import UFinal.lvl.LevelLoader;

public class BadIceCreamFour extends JPanel {

	//Create objects and variables
	public static BadIceCreamFour bic4;
	public static LevelLoader lvlLoader;

	private JFrame mainFrame;
	private JLayeredPane gamePane;
	private Timer updateTimer;
	private Refresher refresher;
	private ActionListener updateListener;
	private KeyAction keyAction;


	/**
```

```
         * Constructor method that begins the initial game loading and frame display.
         * pre: none.
         * post: the game begins.
         */
        public BadIceCreamFour() {

                //Initiates loading and assigns the proper textfile to the LevelLoader
                lvlLoader = new LevelLoader(new
InputStreamReader(getClass().getResourceAsStream("/UFinal/lvl/levels.txt")));
                lvlLoader.initPassword();

                //Gets the JLayeredPane to put on the JFrame
                gamePane = lvlLoader.getPane();

                //Creates new Refresher object to keep track of time
                refresher = new Refresher();

                //Creates new key handler
                keyAction = new KeyAction();

                //Creates and sets up the JFrame
                mainFrame = new JFrame("Bad Ice Cream 4");
                mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                mainFrame.setContentPane(gamePane);
                mainFrame.setFocusable(true);
                mainFrame.addKeyListener(keyAction);
                mainFrame.setSize(new Dimension(Constants.FRAME_WIDTH, Constants.FRAME_HEIGHT));
                mainFrame.setResizable(false);
                mainFrame.setVisible(true);

                //Creates the ActionListener that will cause the game graphics to update
                setUpdateListener();
                //Set ActionListener to a timer
                updateTimer = new Timer(1, updateListener);
                updateTimer.start();
        }

        /**
         * Creates the ActionListener that will be used to update the game
```

```
	 * pre: none
	 * post: ActionListener created and fully defined
	 */
	public void setUpdateListener() {

		updateListener = new ActionListener() {

			@Override
			public void actionPerformed(ActionEvent e) {

				//Refeshes game tactics if a level has been loaded
				if(Constants.levelLoaded) {

					//Called on the KeyAction class so it can handle multiple key presses at once
					keyAction.useKeys();

					//Refreshes all objects in the Tile ArrayList, including the player
					for(int i = 0; i < Constants.Y_TILEROWS; i++) {
						for(int j = 0; j < Constants.X_TILECOLS; j++) {
							lvlLoader.getTileList().get(i).get(j).refresh();
						}
					}

					//Refreshes all objects in the Fruit ArrayList that are part of the current batch of
fruit

					for(int j = 0; j < Constants.numFruit; j++) {
						lvlLoader.getFruitArray().get(Constants.fruitLvl).get(j).refresh();
					}

					//Refreshes all objects in the Monster ArrayList
					for(int k = 0; k < lvlLoader.getMonsterList().size(); k++) {
						lvlLoader.getMonsterList().get(k).refresh();
					}

					//Refreshes on screen timer
					lvlLoader.getTimer().refresh();
				}
			}
		};
	}

	/**
	 * Causes the GUI components to be set up via the constructor.
	 * pre: none
	 * post: constructor is run, and the GUI components are set up
	 */
	private static void runGUI() {
		bic4 = new BadIceCreamFour();
	}

	/**
	 * The main method. The program starts here.
	 * pre: none
	 * post: the program begins
	 */
	public static void main(String[] args) {

		javax.swing.SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				runGUI();
			}
		});
	}
}
```

Refresher Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Refresher class.
 */

package UFinal;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.Timer;

public class Refresher implements Runnable {

	//Creates objects and variables
	BadIceCreamFour bic4Obj = BadIceCreamFour.bic4;

	private Thread t;
	private ActionListener timerListener;
	private Timer timer;

	/**
	 * Constructor method - creates a new Thread for Refresher
	 * pre: none
	 * post: Timer and ActionListener instantiated. Start() called.
	 */
	public Refresher() {
		setTimerListener();
		timer = new Timer(1, timerListener);
		start();
	}

	/**
	 * Instantiates thread and runs it.
 * pre: none
 * post: thread is made and run
 */
public void start() {

      t = new Thread(this);
      t.run();
}

/**
 * Starts timer and updates the refreshCount variable to keep track of time in the game
 * pre: none
 * post: timer started
 */
public void run() {
      timer.start();
}

/**
 * Sets the ActionListener that the timer is linked to
 * pre: none
 * post: action listener is set, with refreshCounter incrementing by 1 per cycle
 */
public void setTimerListener() {

      timerListener = new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

                  Constants.refreshCount++;

                  //Reset refreshCount before value of the count becomes too large for an integer
                  if(Constants.refreshCount > 1000000000) {
                        Constants.refreshCount = 0;
                  }
            }
      };
}}
```

SecTimer Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - SecTimer class.
 */

package UFinal;

import java.awt.Color;
import java.awt.Font;

import javax.swing.JLabel;

import UFinal.lvl.LevelLoader;

public class SecTimer extends JLabel {

	//Create objects and variables
	private BadIceCreamFour bic4Obj = BadIceCreamFour.bic4;
	private LevelLoader lvlObj = bic4Obj.lvlLoader;

	private int init, secsLeft;

	/**
	 * Constructor for SecTimer, which sets its time limit, and position
	 * pre: secLimit, x, and y are positive
	 * post: SecTimer object created with its respective properties
	 */
	public SecTimer(int secLimit, int x, int y) {

		//Sets time limit, font, and position
		secsLeft = secLimit;
		init = Constants.refreshCount;
		setFont(new Font("Comic Sans MS", Font.BOLD, 25));
```

```
        setForeground(Color.BLACK);
        setText("Time left: " + secsLeft + "s");

        setBounds((int)x, (int)y, getPreferredSize().width, getPreferredSize().height);
    }

    /**
     * Refreshes the timer and updates the time shown
     * pre: none
     * post: timer is updated
     */
    public void refresh() {

        //If a level is loaded and has been not lost or won AND the time elapsed since the last timer change
        //is more than one second, change the time left
        if(!Constants.gameLost && !Constants.gameWon && Math.abs(Constants.refreshCount - init) >= 1000) {

            secsLeft = Math.max(0, secsLeft - 1);
            setText("Time left: " + secsLeft + "s");
            init = Constants.refreshCount;

            //If there is no time left, then cause a loss endgame
            if(secsLeft == 0) {

                //sets all the players to dead
                for(int i = 0; i < lvlObj.getPlayerList().size(); i++) {
                    lvlObj.getPlayer(i).isDead();
                }

                lvlObj.gameLost();
            }
        }
    }
}
```

KeyAction Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - KeyAction class.
 */

package UFinal;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;

import UFinal.lvl.LevelLoader;
import UFinal.tiles.TransparentTile;

public class KeyAction implements KeyListener {

	//Create objects and variables
	BadIceCreamFour bic4Obj = BadIceCreamFour.bic4;
	LevelLoader lvlObj = bic4Obj.LvlLoader;

	ArrayList<Integer> keyArray = new ArrayList<Integer>();
	ArrayList<Integer> counterArray = new ArrayList<Integer>();

	int keyNum, init = 0;

	/**
	 * Adds keys newly pressed to an ArrayList and/or triggers the creation or destruction of ice.
	 * pre: none
	 * post: key pressed is saved to an ArrayList and/or ice is created or destroyed.
	 */
	@Override
	public void keyPressed(KeyEvent e) {
```

```
            //If the key is newly pressed (avoids issues with keyPressed being repeatedly triggered when a key is
held down)
            if(!keyArray.contains(e.getKeyCode())) {

                  //Triggers the manipulation of ice if Space is hit.
                  if(e.getKeyCode() == KeyEvent.VK_SPACE) {
                        lvlObj.getPlayer(0).iceInteraction();
                  }

                  //Adds the key to an ArrayList, as well as when the key was pressed to a different ArrayList
                  keyArray.add(e.getKeyCode());
                  counterArray.add(Constants.refreshCount);
            }
      }

      /**
       * Triggered when a key corresponding to a typable character is pressed.
       * pre: none
       * post: code within the method is run
       */
      @Override
      public void keyTyped(KeyEvent e) {

      }

      /**
       * Removes a key from the Key ArrayList when it is released, as well as its timestamp in the counterArray.
       * pre: none
       * post: key is removed from its ArrayList, as well as its timestamp from the counterArray
       */
      @Override
      public void keyReleased(KeyEvent e) {

            counterArray.remove(keyArray.indexOf(e.getKeyCode()));
            keyArray.remove(keyArray.indexOf(e.getKeyCode()));
      }

      /**
       * Cycles through an Array containg keys currently pressed, and incites actions based on the keys pressed.
```

```
         * pre: none
         * post: player's character is updated
         */
        public void useKeys() {

                //if keyArray is not empty
                if(keyArray.size() > -1) {

                        //Cycle through the key array
                        for(int i = 0; i < keyArray.size(); i++) {

                                keyNum = keyArray.get(i);
                                init = counterArray.get(i);

                                //For arrow key buttons, if the player's movement will not be obstructed by ice, the player 
is trying to
                                //move off the edge of the level, the player is not currently traversing between tiles, and 
the key has been
                                //held for over 50 ms, then the player may traverse in the direction the arrow key 
indicates
                                if(keyNum == KeyEvent.VK_RIGHT && lvlObj.getPlayer(0).getRightX() < 
Constants.FRAME_WIDTH_ADJUSTED) {

                                        lvlObj.getPlayer(0).faceDirection('r');

                                        if(lvlObj.getPlayer(0).getTileX() + 1 < Constants.X_TILECOLS &&
                                                        lvlObj.getPlayer(0).deltaX() == 0 && lvlObj.getPlayer(0).deltaY() == 0 
&& Math.abs(Constants.refreshCount - init) >= 50
                                                        && 
(lvlObj.getTileList().get(lvlObj.getPlayer(0).getTileY()).get(lvlObj.getPlayer(0).getTileX() + 1) instanceof 
TransparentTile)) {

                                                lvlObj.getPlayer(0).moveTilesHor(true);
                                        }

                                } else if (keyNum == KeyEvent.VK_LEFT && lvlObj.getPlayer(0).getX() > 0) {

                                        lvlObj.getPlayer(0).faceDirection('l');
                                if(lvlObj.getPlayer(0).getTileX() - 1 >= 0 &&
                                        lvlObj.getPlayer(0).deltaX() == 0 && lvlObj.getPlayer(0).deltaY() == 0
&& Math.abs(Constants.refreshCount - init) >= 50
                                        &&
(lvlObj.getTileList().get(lvlObj.getPlayer(0).getTileY()).get(lvlObj.getPlayer(0).getTileX() - 1) instanceof
TransparentTile)) {

                                    lvlObj.getPlayer(0).moveTilesHor(false);
                                }

                            } else if (keyNum == KeyEvent.VK_DOWN && lvlObj.getPlayer(0).getBottomY() <
Constants.FRAME_HEIGHT_ADJUSTED) {

                                lvlObj.getPlayer(0).faceDirection('d');

                                if(lvlObj.getPlayer(0).getTileY() + 1 < Constants.Y_TILEROWS &&
                                        lvlObj.getPlayer(0).deltaX() == 0 && lvlObj.getPlayer(0).deltaY() == 0
&& Math.abs(Constants.refreshCount - init) >= 50
                                        && (lvlObj.getTileList().get(lvlObj.getPlayer(0).getTileY() +
1).get(lvlObj.getPlayer(0).getTileX()) instanceof TransparentTile)) {

                                    lvlObj.getPlayer(0).moveTilesVer(false);
                                }

                            } else if (keyNum == KeyEvent.VK_UP && lvlObj.getPlayer(0).getY() > 0) {

                                lvlObj.getPlayer(0).faceDirection('u');

                                if(lvlObj.getPlayer(0).getTileY() - 1 >= 0 &&
                                        lvlObj.getPlayer(0).deltaX() == 0 && lvlObj.getPlayer(0).deltaY() == 0
&& Math.abs(Constants.refreshCount - init) >= 50
                                        && (lvlObj.getTileList().get(lvlObj.getPlayer(0).getTileY() -
1).get(lvlObj.getPlayer(0).getTileX()) instanceof TransparentTile)) {

                                    lvlObj.getPlayer(0).moveTilesVer(true);
                                }
                            }
                        }
                }       }}
```

Constants Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Constants class.
 */

package UFinal;

public class Constants {

	//Variables used to set coordinates of / position graphical objects
	public static final int X_LEFTINSET = 11;
	public static final int X_RIGHTINSET = 11;
	public static final int Y_TOPINSET = 45;
	public static final int Y_BOTTOMINSET = 11;

	public static double X_OFFSET = 0;
	public static double Y_OFFSET = 0;

	public static final int CHAR_WIDTH = 36;
	public static final int CHAR_HEIGHT = 55;

	public static final int FRAME_WIDTH = 1250;
	public static final int FRAME_HEIGHT = 750;

	public static final int FRAME_WIDTH_ADJUSTED = FRAME_WIDTH - X_LEFTINSET - X_RIGHTINSET;
	public static final int FRAME_HEIGHT_ADJUSTED = FRAME_HEIGHT - Y_TOPINSET - Y_BOTTOMINSET;

	public static int X_TILECOLS = 0;
	public static int Y_TILEROWS = 0;
	public static int tileSize = 0;

	//Used to time objects in the game - as this game is iteratively based
	public static int refreshCount = 0;
```

```
    //Used to keep track of fruits in the level
    public static int numFruit = 0;
    public static int fruitLvl = 0;

    //Used to keep track of players in the level
    public static int numPlayers = 1;
    public static int numDead = 0;

    //Used to keep track of game state
    public static boolean gameWon = false;
    public static boolean gameLost = false;
    public static boolean levelLoaded = false;

    //Used to toggle sound
    public static boolean musicEnabled = true;
    public static boolean sfxEnabled = true;

}
```

Tile Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Tile class.
 */

package UFinal.tiles;

import java.awt.Image;
import java.awt.image.BufferedImage;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JLabel;

import UFinal.Constants;

public class Tile extends JLabel {

	//create variables and objects
	protected int tileX, tileY;
	protected double x, y, rightX, bottomY, centerX, centerY, incr, goalX, goalY;
	protected Update updater;
	protected boolean isX, isMvmtDone;

	/**
	 * Constructor for Tile that resizes it to a custom width and height.
	 * pre: all integer parameters are nonnegative
	 * post: Tile object created with its respective properties
	 */
	public Tile(String url, int xTile, int yTile, int width, int height, boolean isFruit) {

		setIcon(resizeImg(url, width, height)); //creates the graphic
		tileX = xTile; //sets variables
		tileY = yTile;

		setCoordinates(isFruit); //set Coordinates
	}

	/**
	 * Constructor for Tile that resizes it to a custom width and height for Tiles that are not Fruit.
	 * pre: all integer parameters are nonnegative
	 * post: Tile object created with its respective properties
	 */
	public Tile(String url, int xTile, int yTile, int width, int height) {

		this(url, xTile, yTile, width, height, false);
	}

	/**
	 * Constructor for Tile that resizes it to the tile width and height for the level
	 * pre: all integer parameters are nonnegative positive
	 * post: Tile object created with its respective properties
	 */
	public Tile(String url, int xTile, int yTile, boolean isFruit) {

		try {

			BufferedImage bufferedImg = ImageIO.read(getClass().getResource(url));

			if(isFruit) { //do not resize image if the object is a fruit and set it as the new icon
				setIcon(new ImageIcon(bufferedImg));
			} else { //resize image and set it as the new icon
				setIcon(resizeImg(bufferedImg, Constants.tileSize,

	(int)((double)bufferedImg.getHeight()/((double)bufferedImg.getWidth()/(double)Constants.tileSize))));
			}

		} catch (IOException e) {
			e.printStackTrace();
		}
		tileX = xTile;
		tileY = yTile;

		setCoordinates(isFruit);
	}

	/**
	 * Constructor for Tiles that are not fruit that resizes it to the tile width and height for the level
	 * pre: all integer parameters are nonnegative positive
	 * post: Tile object created with its respective properties
	 */
	public Tile(String url, int xTile, int yTile) {

		this(url, xTile, yTile, false);
	}

	/**
	 * Sets the location of a fruit based on tile coordinates
	 * pre: none
	 * post: fruit pixel coordinates set
	 */
	public void setCoordinates(boolean isFruit) {

		if(isFruit) { //sets the center coordinates of the images
			setCenterX((int)(tileX*Constants.tileSize + Constants.tileSize/2.0 + Constants.X_OFFSET));
			setCenterY((int)(tileY*Constants.tileSize + Constants.tileSize/2.0 + Constants.Y_OFFSET));
		} else {
			setCenterX((int)(tileX*Constants.tileSize + Constants.tileSize/2.0 + Constants.X_OFFSET));
			setCenterY((int)((tileY + 1)*Constants.tileSize + Constants.Y_OFFSET));
		}

		//Set bounds, refresh variables, and goal coordinates
		setBounds((int)x, (int)y, getWidth(), getHeight());
		setRefresh(true, 0);
		goalX = getCenterX();
		goalY = getCenterY();
	}

	/**
 * Resizes an image to a manually set width and height using a URL
 * pre: width and height are positive
 * post: image is resized and returned
 */
public ImageIcon resizeImg(String url, int width, int height) {

      BufferedImage bufferedImg; //new buffered image
      try {
            bufferedImg = ImageIO.read(getClass().getResource(url));
      } catch (IOException e) {
            e.printStackTrace();
            return null;
      }

      return resizeImg(bufferedImg, width, height); //call resizeImg
}

/**
 * Resizes an image to a manually set width and height using a buffered image
 * pre: width and height are positive
 * post: image is resized and returned
 */
public ImageIcon resizeImg(BufferedImage buffImg, int width, int height) {

      //Use getScaledInstance to resize the image
      Image image = buffImg.getScaledInstance(width, height, Image.SCALE_DEFAULT);
      return new ImageIcon(image);
}

/**
 * Sets leftmost x coordinate
 * pre: none
 * post: leftmost x is set
 */
public void setX(double xCoord) {
      setLocation((int)(x = xCoord), (int)y);
      setRightX();
}
```

```
/**
 * Sets leftmost y coordinate
 * pre: none
 * post: leftmost y is set
 */
public void setY(double yCoord) {
	setLocation((int)x, (int)(y = yCoord));
	setBottomY();
}

/**
 * Sets rightmost x coordinate
 * pre: none
 * post: rightmost x is set
 */
protected void setRightX() {
	rightX = x + getWidth();
}

/**
 * Sets bottom y coordinate
 * pre: none
 * post: bottom y is set
 */
protected void setBottomY() {
	bottomY = y + getHeight();
}

/**
 * Sets center x coordinate
 * pre: none
 * post: center x is set
 */
public void setCenterX(double xCoord) {

	centerX = xCoord;
	setX((int) (centerX - getWidth()/2.0));
}
```

```
/**
 * Sets center y coordinate
 * pre: none
 * post: center y is set
 */
public void setCenterY(double yCoord) {

      centerY = yCoord;
      setY((int) (centerY - getHeight()));
}

/**
 * Sets updater method according to the Update interface
 * pre: none
 * post: updater is set
 */
public void setUpdater(Update upd8) {
      updater = upd8;
}

/**
 * Sets whether the character moves horizontally or vertically and how its location will increment
 * pre: none
 * post: isX and incr are set
 */
public void setRefresh(boolean x, double num) {
      isX = x;
      incr = num;
}

/**
 * Sets goal x coordinate
 * pre: none
 * post: goal x is set
 */
public void setGoalX(double xGoal) {
      goalX = xGoal + (int)Constants.X_OFFSET;
}
```

```
/**
 * Sets goal y coordinate
 * pre: none
 * post: goal y is set
 */
public void setGoalY(double yGoal) {
	goalY = yGoal + (int)Constants.Y_OFFSET;
}

/**
 * Returns leftmost x-coordinate
 * pre: none
 * post: leftmost x-coordinate is returned
 */
public int getX() {
	return (int)x;
}

/**
 * Returns uppermost y-coordinate
 * pre: none
 * post: uppermost y-coordinate is returned
 */
public int getY() {
	return (int)y;
}

/**
 * Returns rightmost x-coordinate
 * pre: none
 * post: rightmost x-coordinate is returned
 */
public double getRightX() {
	return rightX;
}

/**
 * Returns bottom-most y-coordinate
 * pre: none
 * post: bottom-most y-coordinate is returned
 */
public double getBottomY() {
	return bottomY;
}

/**
 * Returns tile width.
 * pre: none
 * post: tile width returned
 */
public int getWidth() {
	return getPreferredSize().width;
}

/**
 * Returns tile height.
 * pre: none
 * post: tile height returned
 */
public int getHeight() {
	return getPreferredSize().height;
}

/**
 * return center x coordinate.
 * pre: none
 * post: center x returned
 */
public double getCenterX() {
	return centerX;
}

/**
 * return center y coordinate.
 * pre: none
 * post: center y returned
 */
public double getCenterY() {
		return centerY;
	}

	/**
	 * return tileX coordinate.
	 * pre: none
	 * post: tileX coordinate returned
	 */
	public int getTileX() {
		return tileX;
	}

	/**
	 * return tileY coordinate.
	 * pre: none
	 * post: tileY coordinate returned
	 */
	public int getTileY() {
		return tileY;
	}

	/**
	 * return goalX coordinate.
	 * pre: none
	 * post: goalX coordinate returned
	 */
	public double getGoalX() {
		return goalX;
	}

	/**
	 * return goalY coordinate.
	 * pre: none
	 * post: goalY coordinate returned
	 */
	public double getGoalY() {
		return goalY;
	}
	/**
	 * return the difference between the current x and goal x.
	 * pre: none
	 * post: difference in x is returned
	 */
	public double deltaX() {
		return (int)(Math.abs(goalX - getCenterX()));
	}

	/**
	 * return the difference between the current y and goal y.
	 * pre: none
	 * post: difference in y is returned
	 */
	public double deltaY() {
		return (int)(Math.abs(goalY - getCenterY()));
	}

	/**
	 * The Update interface, which contains the refresh() method that is repeatedly called for all Tile objects to
keep them updated
	 */
	public interface Update {
		public void refresh(boolean isX, double incr);
	}

	/**
	 * Refreshes the location of Tile objects, drawing them in a different place.
	 * pre: none
	 * post: movement occurs if the Character is not obstructed by ice and is not trying to go off the level.
	 */
	public void refresh() {
		updater.refresh(isX, incr);
	}
}
```

IceTile Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - IceTile class.
 */

package UFinal.tiles;

public class IceTile extends Tile {

	/**
	 * Constructor for IceTile that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: IceTile object created with its respective properties
	 */
	public IceTile(int xTile, int yTile, int width, int height) {
		//Passes parameters onto Tile
		super("/UFinal/img/Ice_Block.png", xTile, yTile, width, height);
	}

	/**
	 * Constructor for IceTile that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: IceTile object created with its respective properties
	 */
	public IceTile(int xTile, int yTile) {
		//Passes parameters onto Tile
		super("/UFinal/img/Ice_Block.png", xTile, yTile);
	}

	//Blank refresh method
	public void refresh() {};
}
```

TransparentTile Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - TransparentTile class.
 */

package UFinal.tiles;

public class TransparentTile extends Tile {

	/**
	 * Constructor for TransparentTile that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: TransparentTile object created with its respective properties
	 */
	public TransparentTile(int xTile, int yTile, int width, int height) {
		super("/UFinal/img/transparentTile.png", xTile, yTile, 1, 1);
		setOpaque(false);
	}

	/**
	 * Constructor for TransparentTile that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: TransparentTile object created with its respective properties
	 */
	public TransparentTile(int xTile, int yTile) {
		super("/UFinal/img/transparentTile.png", xTile, yTile, 1, 1);
		setOpaque(false);
	}

	//blank refresh method
	public void refresh() {};
}
```

Characters Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Characters class.
 */

package UFinal.characters;

import java.io.IOException;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;

import UFinal.BadIceCreamFour;
import UFinal.Constants;
import UFinal.lvl.LevelLoader;
import UFinal.tiles.Tile;
import UFinal.tiles.TransparentTile;

public class Characters extends Tile {

	//Create objects and variables
	BadIceCreamFour bic4Obj = BadIceCreamFour.bic4;
	LevelLoader lvlObj = bic4Obj.LvlLoader;

	private char lastDir;
	private boolean isAlive;
	private Clip iceSound;

	/**
	 * Constructor for Characters that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Character object created with its respective properties
	 */
	public Characters(String url, int xCoord, int yCoord, int width, int height) {

		//Passes parameters on to Tile
		super(url, xCoord, yCoord, width, height);

		lastDir = 'd';
		isAlive = true;
		makeUpdater();

		//Creates audio IceSound object
		try {

			AudioInputStream iceSfx = 
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/iceSound2.wav"));
			iceSound = AudioSystem.getClip();
			iceSound.open(iceSfx);

		} catch (UnsupportedAudioFileException e1) {
			e1.printStackTrace();
		} catch (IOException e1) {
			e1.printStackTrace();
		} catch (LineUnavailableException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Constructor for Characters that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Character object created with its respective properties
	 */
	public Characters(String url, int xCoord, int yCoord) {

		//Passes parameter on to Tile
		super(url, xCoord, yCoord);

		lastDir = 'd';
		isAlive = true;
```

```
            makeUpdater();

            //Creates audio IceSound object
            try {

                  AudioInputStream iceSfx =
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/iceSound2.wav"));
                  iceSound = AudioSystem.getClip();
                  iceSound.open(iceSfx);

            } catch (UnsupportedAudioFileException e1) {
                  e1.printStackTrace();
            } catch (IOException e1) {
                  e1.printStackTrace();
            } catch (LineUnavailableException e) {
                  e.printStackTrace();
            }
      }

      /**
       * Mutates the marker that indicates which direction the character is facing
       * pre: dir is 'u', 'd', 'l', 'r'
       * post: lastDir is set
       */
      public void faceDirection(char dir) {
            lastDir = dir;
      }

      /**
       * Causes the player to create/destroy ice blocks
       * pre: none
       * post: ice blocks created or destroyed
       */
      public void iceInteraction() {

            //If the character is alive
            if(isAlive) {

                  //Play the IceSound SFX if they are enabled
```

```
                        if(Constants.sfxEnabled) {
                                iceSound.setFramePosition(0);
                                iceSound.loop(0);
                        }

                        //Create a new Icing thread
                        new Icing(tileX, tileY, lastDir);
                }
        }

        /**
         * Causes a character to move between tiles horizontally. True indicates motion toward the right of the screen
         * pre: chararcter's movement will not be obstructed by an adjacent ice block and character is not going off
the edge of the level
         * post: character moves left or right one tile
         */
        public void moveTilesHor(boolean isRight) {

                //If the character is alive and set to go right
                if(isRight && isAlive) {

                        //Set the goal x coordinate
                        setGoalX((tileX + 1)*Constants.tileSize + Constants.tileSize/2);

                        //Change the Tile ArrayList to reflect the changes
                        lvlObj.getTileList().get(tileY).set(tileX + 1, this);
                        lvlObj.getTileList().get(tileY).set(tileX, (new TransparentTile(tileX, tileY)));
                        tileX++;

                        //Make the character move 1 pixel to the right with every refresh
                        setRefresh(true, 1);

                } else if (isAlive) { //If the character is alive

                        //Set the goal x coordinate
                        setGoalX((tileX - 1)*Constants.tileSize + Constants.tileSize/2);

                        //Change the Tile ArrayList to reflect the changes
                        lvlObj.getTileList().get(tileY).set(tileX - 1, this);
                lvlObj.getTileList().get(tileY).set(tileX, (new TransparentTile(tileX, tileY)));
                tileX--;

                //Make the character move 1 pixel to the left with every refresh
                setRefresh(true, -1);
            }
        }

        /**
         * Causes a character to move between tiles vertically. True indicates motion toward the top of the screen
         * pre: chararcter's movement will not be obstructed by an adjacent ice block and character is not going off
the edge of the level
         * post: character moves up or down one tile
         */
        public void moveTilesVer(boolean isUp) {

            //If the character is alive and set to go left
            if(isUp && isAlive) {

                //Set the goal y coordinate
                setGoalY(tileY*Constants.tileSize);
                setRefresh(false, -1); //Make the character move one pixel up with every refresh

                //Reflect the changes in the Tile ArrayList and set the character to a new layer
                lvlObj.getTileList().get(tileY - 1).set(tileX, this);
                lvlObj.getPane().setLayer(this, new Integer((tileY - 1)*10));

                lvlObj.getTileList().get(tileY).set(tileX, (new TransparentTile(tileX, tileY)));
                tileY--;

            } else if (isAlive) { //If the character is alive

                //Set the goal y coordinate
                setGoalY((tileY + 2)*Constants.tileSize);
                setRefresh(false, 1); //Make the character move one pixel down with every refresh

                //Reflect the changes in the Tile ArrayList and set the character to a new layer
                lvlObj.getTileList().get(tileY + 1).set(tileX, this);
                lvlObj.getPane().setLayer(this, new Integer((tileY + 1)*10));
```

```
				lvlObj.getTileList().get(tileY).set(tileX, (new TransparentTile(tileX, tileY)));
				tileY++;
			}
		}

		/**
		 * Sets the refresh() method to make the character move by a set increment until it reaches
		 * its goal X or goal Y coordinate
		 * pre: none
		 * post: refresh() method is set and the character increments if made to do so
		 */
		public void makeUpdater() {

			Update updater = new Update() {

				@Override
				public void refresh(boolean isX, double incr) {

					//Keeps character moving until they are at their goalX and goalY coordinates
					if(deltaX() == 0 && deltaY() == 0) {
						setRefresh(true, 0);
					}

					if(isX && deltaX() > 0) {
						setCenterX(getCenterX() + incr);
					} else if (deltaY() > 0){
						setCenterY(getCenterY() + incr);
					}
				}
			};

			setUpdater(updater);
		}

		/**
		 * Sets the character to being dead.
		 * pre: none
		 * post: the isAlive flag is set to false, and the Character is removed from the Tile ArrayList
```

```
 */
public void isDead() {
      isAlive = false;
      lvlObj.getTileList().get(tileY).set(tileX, new TransparentTile(tileX, tileY));
}

/**
 * Returns tileX.
 * pre: none
 * post: tileX returned
 */
public void setTileX(int xTile) {
      tileX = xTile;
}

/**
 * Returns tileY
 * pre: none
 * post: tileY returned
 */
public void setTileY(int yTile) {
      tileY = yTile;
}

/**
 * Returns lastDir
 * pre: none
 * post: lastDir returned
 */
public char getLastDir() {
      return lastDir;
}

/**
 * Returns isAlive
 * pre: none
 * post: isAlive returned
 */
public boolean getIsAlive() {
		return isAlive;
	}
}
```

Icing Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Icing class.
 */

package UFinal.characters;

import UFinal.BadIceCreamFour;
import UFinal.Constants;
import UFinal.lvl.LevelLoader;
import UFinal.tiles.IceTile;
import UFinal.tiles.TransparentTile;

public class Icing implements Runnable {

	//Creates objects and variables
	BadIceCreamFour bic4 = BadIceCreamFour.bic4;
	LevelLoader lvlObj = bic4.LvlLoader;

	private int tileX, tileY;
	private char lastDir;
	private Thread t;
	private boolean isMonsterBlocking;

	/**
	 * Constructor for Icing - starts creating or destroying according to where the character is
	 * and what direction they're facing
	 * pre: xTile and yTile are nonnegative, and dirLast is 'u', 'd', 'l', 'r'
	 * post: Icing object created and Threading process begins
	 */
	public Icing (int xTile, int yTile, char dirLast) {

		//Set variables
		tileX = xTile;
		tileY = yTile;
```

```
            lastDir = dirLast;
            isMonsterBlocking = false;

            start();
      }

      /**
       * Instantiates thread and runs it.
       * pre: none
       * post: thread is made and run
       */
      public void start() {

            t = new Thread(this);
            t.start();
      }

      /**
       * Starts creating or destroying ice
       * pre: none
       * post: ice is destroyed or created
       */
      public void run() {

            //Create temporary variables
            int temp = 1;
            IceTile temp2;
            TransparentTile temp3;
            int temp4 = Constants.refreshCount;

            //For each direction, ice is only created or destroyed if the player is not facing off the edge of the
level and
            //ice is created or destroyed until the edge of the level is released. IceTile objects or
TransparentTile objects
            //are iteratively created every 40ms and creation may be interrupted by a monster in the way.
            if(lastDir == 'd') {

                  //Only create ice there are no obstructions and creation is within the level boundaries
```

```
                    if(tileY + temp < Constants.Y_TILEROWS && !(lvlObj.getTileList().get(tileY + temp).get(tileX)
instanceof IceTile)) {

                        while(tileY + temp < Constants.Y_TILEROWS
                                && (lvlObj.getTileList().get(tileY + temp).get(tileX) instanceof
TransparentTile)) {

                            //Checks for obstructions by monsters
                            for(int i = 0; i < lvlObj.getMonsterList().size(); i++) {
                                if(lvlObj.getMonsterList().get(i).getTileX() == tileX
                                    && lvlObj.getMonsterList().get(i).getTileY() == tileY + temp) {
                                    isMonsterBlocking = true;
                                    break;
                                }
                            }

                            if(isMonsterBlocking) {
                                break; //break if a monster is causing obstruction
                            }

                            System.out.println(temp4); //Forcing printing smooths out the threading (not sure
why)

                            if(temp == 1 || Math.abs(Constants.refreshCount - temp4) >= 40) { //Creates ice once
every 40 ms
                                //Reflect changes in Tile ArrayList
                                if(lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof Characters) {
                                    tileY++;
                                }

                                temp2 = new IceTile(tileX, tileY + temp);
                                lvlObj.getTileList().get(tileY + temp).set(tileX, temp2);
                                lvlObj.getPane().add(temp2, new Integer((tileY + temp)*10));
                                temp++;
                                temp4 = Constants.refreshCount;
                            }
                        }

                    } else {
                        //Destroy ice until the continuous row of ice being destroyed ends
                        while(tileY + temp < Constants.Y_TILEROWS
                                && lvlObj.getTileList().get(tileY + temp).get(tileX) instanceof IceTile) {

                            System.out.println(temp4); //Forcing printing smooths threading (not sure why)

                            if(Math.abs(Constants.refreshCount - temp4) >= 40) { //destroys the next block after
40ms

                                //Reflect changes in the Tile ArrayList
                                temp3 = new TransparentTile(tileX, tileY + temp);

                                lvlObj.getPane().remove(lvlObj.getTileList().get(tileY + temp).get(tileX));
                                lvlObj.getTileList().get(tileY + temp).set(tileX, temp3);
                                lvlObj.getPane().revalidate();
                                lvlObj.getPane().repaint();

                                temp++;
                                temp4 = Constants.refreshCount;

                            }
                        }
                    }

                } else if (lastDir == 'u') { //Read comments for case 'd' or the general comment above that

                    if(!(lvlObj.getTileList().get(tileY - temp).get(tileX) instanceof IceTile)) {

                        while(tileY - temp >= 0 &&
                                (lvlObj.getTileList().get(tileY - temp).get(tileX) instanceof
TransparentTile)) {

                            for(int i = 0; i < lvlObj.getMonsterList().size(); i++) {
                                if(lvlObj.getMonsterList().get(i).getTileX() == tileX
                                    && lvlObj.getMonsterList().get(i).getTileY() == tileY - temp) {
                                    isMonsterBlocking = true;
                                    break;
                                }
                            }

                            if(isMonsterBlocking) {
                                break;
                            }

                            System.out.println(temp4);

                            if(temp == 1 || Math.abs(Constants.refreshCount - temp4) >= 40) {

                                if(lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof Characters) {
                                    tileY--;
                                }

                                temp2 = new IceTile(tileX, tileY - temp);
                                lvlObj.getTileList().get(tileY - temp).set(tileX, temp2);
                                lvlObj.getPane().add(temp2, new Integer((tileY - temp)*10));
                                temp++;
                                temp4 = Constants.refreshCount;
                            }
                        }

                    } else { //Read comments for case 'd' or the general comment above that

                        while(tileY - temp >= 0 && lvlObj.getTileList().get(tileY - temp).get(tileX) instanceof
IceTile) {

                            System.out.println(temp4);

                            if(Math.abs(Constants.refreshCount - temp4) >= 40) {

                                temp3 = new TransparentTile(tileX, tileY - temp);

                                lvlObj.getPane().remove(lvlObj.getTileList().get(tileY - temp).get(tileX));
                                lvlObj.getTileList().get(tileY - temp).set(tileX, temp3);
                                lvlObj.getPane().revalidate();
                                lvlObj.getPane().repaint();
                                temp++;
                                temp4 = Constants.refreshCount;
```

```
                              }
                        }
                  }

            } else if (lastDir == 'r') { //Read comments for case 'd' or the general comment above that

                  if(!(lvlObj.getTileList().get(tileY).get(tileX + temp) instanceof IceTile)) {

                        while(tileX + temp < Constants.X_TILECOLS &&
                                    (lvlObj.getTileList().get(tileY).get(tileX + temp) instanceof
TransparentTile)) {

                              for(int i = 0; i < lvlObj.getMonsterList().size(); i++) {
                                    if(lvlObj.getMonsterList().get(i).getTileX() == tileX + temp
                                          && lvlObj.getMonsterList().get(i).getTileY() == tileY) {
                                          isMonsterBlocking = true;
                                          break;
                                    }
                              }

                              if(isMonsterBlocking) {
                                    break;
                              }

                              System.out.println(temp4);

                              if(temp == 1 || Math.abs(Constants.refreshCount - temp4) >= 40) {

                                    if(lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof Characters) {
                                          tileX++;
                                    }

                                    temp2 = new IceTile(tileX + temp, tileY);
                                    lvlObj.getTileList().get(tileY).set(tileX + temp, temp2);
                                    lvlObj.getPane().add(temp2, new Integer(tileY*10));
                                    temp++;
                                    temp4 = Constants.refreshCount;
                              }
```

```
                    }

                } else { //Read comments for case 'd' or the general comment above that

                    while(tileX + temp < Constants.X_TILECOLS &&
                            lvlObj.getTileList().get(tileY).get(tileX + temp) instanceof IceTile) {

                        System.out.println(temp4);

                        if(Math.abs(Constants.refreshCount - temp4) >= 40) {

                            temp3 = new TransparentTile(tileX + temp, tileY);

                            lvlObj.getPane().remove(lvlObj.getTileList().get(tileY).get(tileX + temp));
                            lvlObj.getTileList().get(tileY).set(tileX + temp, temp3);
                            lvlObj.getPane().revalidate();
                            lvlObj.getPane().repaint();
                            temp++;
                            temp4 = Constants.refreshCount;

                        }
                    }
                }

            } else if (lastDir == 'l') { //Read comments for case 'd' or the general comment above that

                if(!(lvlObj.getTileList().get(tileY).get(tileX - temp) instanceof IceTile)) {

                    while(tileX - temp >= 0 &&
                            (lvlObj.getTileList().get(tileY).get(tileX - temp) instanceof
TransparentTile)) {

                        for(int i = 0; i < lvlObj.getMonsterList().size(); i++) {
                            if(lvlObj.getMonsterList().get(i).getTileX() == tileX - temp
                                && lvlObj.getMonsterList().get(i).getTileY() == tileY) {
                                isMonsterBlocking = true;
                                break;
                            }
                        }
                        if(isMonsterBlocking) {
                                break;
                        }

                        System.out.println(temp4);

                        if(temp == 1 || Math.abs(Constants.refreshCount - temp4) >= 40) {

                                if(lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof Characters) {
                                        tileX--;
                                }

                                temp2 = new IceTile(tileX - temp, tileY);
                                lvlObj.getTileList().get(tileY).set(tileX - temp, temp2);
                                lvlObj.getPane().add(temp2, new Integer(tileY*10));
                                temp++;
                                temp4 = Constants.refreshCount;
                        }
                }

        } else { //Read comments for case 'd' or the general comment above that

                while(tileX - temp >= 0 &&
                                lvlObj.getTileList().get(tileY).get(tileX - temp) instanceof IceTile) {

                        System.out.println(temp4);

                        if(Math.abs(Constants.refreshCount - temp4) >= 40) {

                                temp3 = new TransparentTile(tileX - temp, tileY);

                                lvlObj.getPane().remove(lvlObj.getTileList().get(tileY).get(tileX - temp));
                                lvlObj.getPane().add(temp3, new Integer(tileY*10));
                                lvlObj.getTileList().get(tileY).set(tileX - temp, temp3);
                                lvlObj.getPane().revalidate();
                                lvlObj.getPane().repaint();
                                temp++;
                                temp4 = Constants.refreshCount;
```

```
                                                  }
                                              }
                                          }
                                      }
                                  }
                              }
```

Fruit Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Fruit class.
 */

package UFinal.fruit;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;

import UFinal.BadIceCreamFour;
import UFinal.Constants;
import UFinal.characters.Characters;
import UFinal.lvl.LevelLoader;
import UFinal.tiles.IceTile;
import UFinal.tiles.Tile;
import UFinal.tiles.TransparentTile;

public class Fruit extends Tile {

	//Create objects and variables
	protected BadIceCreamFour bic4Obj = BadIceCreamFour.bic4;
	protected LevelLoader lvlObj = bic4Obj.lvlLoader;

	private boolean isIced, isPicked;
	private String imgUrl;
	private Clip fruitCollected;
	protected Fruit thisFruit;

	/**
	 * Constructor for Fruit that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Fruit object created with its respective properties
	 */
	public Fruit(String url, int xTile, int yTile, int width, int height, boolean isFruit) {

		//Passes parameters to Tile
		super(url, xTile, yTile, width, height, isFruit);

		//set variables
		imgUrl = url;
		isIced = false;
		isPicked = false;
		thisFruit = this;

		try { //Create audio objects

			AudioInputStream snapSfx = 
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/snap.wav"));
			fruitCollected = AudioSystem.getClip();
			fruitCollected.open(snapSfx);

		} catch (UnsupportedAudioFileException e1) {
			e1.printStackTrace();
		} catch (IOException e1) {
			e1.printStackTrace();
		} catch (LineUnavailableException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Constructor for Fruits that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Fruit object created with its respective properties
	 */
	public Fruit(String url, int xTile, int yTile, boolean isFruit) {
		
		//Passes parameters on to Tile
		super(url, xTile, yTile, isFruit);
		
		//set variables
		imgUrl = url;
		isIced = false;
		isPicked = false;
		thisFruit = this;
		
		try { //Creates new Audio objects
			
			AudioInputStream snapSfx = 
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/snap.wav"));
			fruitCollected = AudioSystem.getClip();
			fruitCollected.open(snapSfx);
			
		} catch (UnsupportedAudioFileException e1) {
			e1.printStackTrace();
		} catch (IOException e1) {
			e1.printStackTrace();
		} catch (LineUnavailableException e) {
			e.printStackTrace();
		}
		
		checkIsIced();
	}
	
	/**
	 * Checks if a fruit has been newly covered or uncovered by or from ice
	 * pre: none
	 * post: proper methods called to change object properties
	 */
	public void checkIsIced() {
		
		//If the state of the fruit has just been changed (in terms of ice)
		if(isIced != lvlObj.getTileList().get(tileY).get(tileX) instanceof IceTile) {
			isIced = !isIced; //set boolean flag
			
			if(isIced) { //call the respective methods
				coveredIce();
			} else {
				uncoverIce();
			}
		}
	}

	/**
	 * Changes fruit graphics to reflect it being covered in ice
	 * pre: none
	 * post: fruit image made more transparent
	 */
	public void coveredIce() {
		
		try {
			
			//create a new buffered image
			BufferedImage bfImg = ImageIO.read(getClass().getResource(imgUrl));
			Color tempColor, temp2Color;
			
			//Fill the buffered image up with transparency
			for(int i = 0; i < bfImg.getHeight(); i++) {
				for(int j = 0; j< bfImg.getWidth(); j++) {
					
					tempColor = new Color(bfImg.getRGB(j, i), true);
					temp2Color = new Color(tempColor.getRed(), tempColor.getGreen(), tempColor.getBlue(), 
100);
					
					bfImg.setRGB(j, i, temp2Color.getRGB());
				}
			}
			
			//Set the objects icon
			setIcon(resizeImg(bfImg, getWidth(), getHeight()));
			
		} catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }

    /**
     * Changes fruit graphics to reflect it being uncovered in ice
     * pre: none
     * post: fruit image made completely opaque
     */
    public void uncoverIce() {
        setIcon(resizeImg(imgUrl, getWidth(), getHeight()));
    }

    /**
     * Detects if a player occupies the same tile as the fruit
     * pre: none
     * post: fruit collected by player if player occupies the same tile
     */
    public void detectWhenPicked() {

        //If a player occupies the same tile
        if(lvlObj.getTileList().get(tileY).get(tileX) instanceof Characters) {

            isPicked = true; //set boolean flag to true

            setRefresh(true, 0); //stop movement

            //Play special SFX if they are enabled
            if(Constants.sfxEnabled) {
                fruitCollected.setFramePosition(0);
                fruitCollected.loop(0);
            }

            //Remove the fruit from the screen and the Fruit ArrayList
            lvlObj.getPane().remove(this);
            lvlObj.getPane().revalidate();
            lvlObj.getPane().repaint();

            lvlObj.getFruitArray().get(Constants.fruitLvl).remove(thisFruit);
				Constants.numFruit--; //change number of fruits left in batch

				if(Constants.numFruit <= 0) { //draw the next batch if the last fruit of this back is collected
					Constants.fruitLvl++;
					lvlObj.drawFruit();
				}
			}
		}

		/**
		 * Causes a fruit to move between tiles horizontally. True indicates motion toward the right of the screen
		 * pre: fruit's movement will not be obstructed by an adjacent ice block and character is not going off edge of 
the level
		 * post: fruit moves left or right one tile
		 */
		public void moveTilesHor(boolean isRight, double incR) {

			//If the fruit is set to go right
			if(isRight) {

				//Set goal x coordinate, set pixel increment, and change object's position properties
				setGoalX((tileX + 1)*Constants.tileSize + Constants.tileSize/2);
				setRefresh(true, incR);
				tileX++;

			} else { //left

				//Set goal x coordinate, set pixel increment, and change object's position properties
				setGoalX((tileX - 1)*Constants.tileSize + Constants.tileSize/2);
				setRefresh(true, incR);
				tileX--;
			}
		}

		public void moveTilesVer(boolean isUp, double incR) {

			//If fruit is set to go up
			if(isUp) {
```

```
            //Set goal y coordinate, set pixel increment, change object's layer
            //and change object's position properties
            setGoalY((tileY - 1)*Constants.tileSize + Constants.tileSize/2);
            setRefresh(false, incR);
            lvlObj.getPane().setLayer(thisFruit, new Integer((tileY - 1)*10));
            tileY--;

        } else {

            //Set goal y coordinate, set pixel increment, change object's layer
            //and change object's position properties
            setGoalY((tileY + 1)*Constants.tileSize + Constants.tileSize/2);
            setRefresh(false, incR);
            lvlObj.getPane().setLayer(thisFruit, new Integer((tileY + 1)*10));
            tileY++;
        }
    }

    /**
     * Causes the removal of an Ice Tile that is covering the fruit.
     * pre: none
     * post: ice is removed and fruit is freed
     */
    public void destroyIce(int xTile, int yTile) {

        //Replaces the IceTile with a TransparentTile in the Tile ArrayList
        //and removes the ice block from the screen
        TransparentTile transTile = new TransparentTile(xTile, yTile);

        lvlObj.getPane().remove(lvlObj.getTileList().get(yTile).get(xTile));
        lvlObj.getPane().add(transTile, new Integer((yTile)*10));
        lvlObj.getTileList().get(yTile).set(xTile, transTile);
        lvlObj.getPane().revalidate();
        lvlObj.getPane().repaint();

    }

    /**
```

```
         * Returns isIced.
         * pre: none
         * post: isIced returned
         */
        public boolean getIsIced() {
                return isIced;
        }

        /**
         * Returns isPicked.
         * pre: none
         * post: isPicked returned
         */
        public boolean getIsPicked() {
                return isPicked;
        }

        /**
         * Returns imgURL.
         * pre: none
         * post: imgURL returned
         */
        public String getImgURL() {
                return imgUrl;
        }
}
```

Avocado Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Avocado class.
 */

package UFinal.fruit;

import UFinal.Constants;
import UFinal.characters.Characters;

public class Avocado extends Fruit {

	//Create variables
	private int init;

	/**
	 * Constructor for Avocado that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Avocado object created with its respective properties
	 */
	public Avocado(int xTile, int yTile, int width, int height, boolean isFruit) {
		//Parameters passed onto parents
		super("/UFinal/img/avocado.png", xTile, yTile, width, height, isFruit);
		//set variables
		init = Constants.refreshCount;
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Constructor for Avocado that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Avocado object created with its respective properties
	 */
	public Avocado(int xTile, int yTile, boolean isFruit) {
        //Parameters passed onto parents
        super("/UFinal/img/avocado.png", xTile, yTile, Constants.tileSize/2, Constants.tileSize, isFruit);
        //set variables
        init = Constants.refreshCount;
        checkIsIced();
        setRefresh(true, 0);
        makeUpdater();
    }

    /**
     * Sets the refresh() method to make the Avocado move to a random new position every second
     * pre: none
     * post: refresh() method is set and the Avocado moves to a random new position every second
     */
    public void makeUpdater() {

        Update updater = new Update() {

            //set variables
            int newX;
            int newY;
            boolean validNew;
            Fruit fruit;
            Characters character;

            @Override
            public void refresh(boolean isX, double incr) {

                //Every second
                if(Math.abs(Constants.refreshCount - init) >= 1000) {

                    do {

                        validNew = true; //keeps generating random x and y tiles until a combination
is found
                        //that does not already hold an avocado

                        newX = (int)(Math.random()*Constants.X_TILECOLS);
                        newY = (int)(Math.random()*Constants.Y_TILEROWS);
                        for(int i = 0; i < lvlObj.getFruitArray().get(Constants.fruitLvl).size(); i++)
{

                            fruit = lvlObj.getFruitArray().get(Constants.fruitLvl).get(i);
                            if(fruit.getTileX() == newX && fruit.getTileY() == newY) {
                                validNew = false;
                                break;
                            }
                        }

                        //also keeps generating until a combination is found that does not contain a
player
                        for(int j = 0; j < lvlObj.getPlayerList().size(); j++) {

                            character = lvlObj.getPlayerList().get(j);
                            if(!validNew || character.getTileX() == newX && character.getTileY() ==
newY) {

                                validNew = false;
                                break;
                            }
                        }

                        init = Constants.refreshCount;

                    } while(!validNew);

                    if(validNew) { //set the new X and Y coordinates of the Avocado

                        setCenterX(newX*Constants.tileSize + Constants.tileSize/2 +
Constants.X_OFFSET);

                        setCenterY(newY*Constants.tileSize + Constants.tileSize/2 +
Constants.Y_OFFSET);

                        tileX = newX;
                        tileY = newY;
                        lvlObj.getPane().setLayer(thisFruit, new Integer(tileY*10 + 1));
                        validNew = false;
                    }
                }
                //check if theres ice and check if the fruit has been picked
                checkIsIced();
                detectWhenPicked();
            }
        };

        setUpdater(updater);
    }
}
```

Chili Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Chili class.
 */

package UFinal.fruit;

import UFinal.Constants;
import UFinal.monsters.Monster;
import UFinal.tiles.IceTile;
import UFinal.tiles.TransparentTile;

public class Chili extends Fruit {

	//Create variables
	private char lastDir;
	private boolean alreadyUpdated, alreadyIced;
	private int init;

	/**
	 * Constructor for Chili that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Chili object created with its respective properties
	 */
	public Chili(int xTile, int yTile, int width, int height, boolean isFruit) {
		//Parameters passed to parent
		super("/UFinal/img/chili.png", xTile, yTile, width, height, isFruit);
		//set variables
		setInitDir();
		alreadyUpdated = false;
		alreadyIced = false;
		setRefresh(true, 0);
		makeUpdater();
	}
	/**
	 * Constructor for Chili that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Chili object created with its respective properties
	 */
	public Chili(int xTile, int yTile, boolean isFruit) {
		//Parameters passed to parent
		super("/UFinal/img/chili.png", xTile, yTile, (int) (Constants.tileSize / 2), (int) (Constants.tileSize / 
1.2), 
				isFruit);
		//set variables
		setInitDir();
		alreadyUpdated = false;
		alreadyIced = false;
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Randomly sets the initial direction for the fruit to head
	 * pre: none
	 * post: initial direction set
	 */
	public void setInitDir() {

		int rand = (int) (Math.random() * 4);

		if (rand == 0) {
			lastDir = 'd';
		} else if (rand == 1) {
			lastDir = 'u';
		} else if (rand == 2) {
			lastDir = 'r';
		} else if (rand == 3) {
			lastDir = 'l';
		}
	}

	/**
 * Sets the refresh() method to make the Chili move randomly and melt adjacent ice blocks.
 * If the Chili is trapped within an ice block, it melts that ice block after some amount of time.
 * pre: none
 * post: refresh() method is set and Chili performs its respective actions
 */
public void makeUpdater() {

      Update updater = new Update() {

            //Set variables
            int rand;

            @Override
            public void refresh(boolean isX, double incr) {

                  if (!getIsPicked()) { //randomly sets direction if not picked

                        rand = (int) (Math.random() * 6);

                        if (rand == 0) {
                              lastDir = 'd';
                        } else if (rand == 1) {
                              lastDir = 'u';
                        } else if (rand == 2) {
                              lastDir = 'r';
                        } else if (rand == 3) {
                              lastDir = 'l';
                        }

                        if (deltaX() == 0 && deltaY() == 0) { //If the chili is not traversing between tiles,
                        //check for ice
                              checkIsIced();

                              if (getIsIced()) {
                                    //If the chili is newly frozen, record a timestamp
                                    if (!alreadyIced) {
                                          alreadyIced = true;
                                          init = Constants.refreshCount;
                                          setRefresh(true, 0);
```

```
					}
					//If chili has been trapped for more than 4 secs, destroy the ice
trapping it
					if (Math.abs(Constants.refreshCount - init) >= 4000) {
						destroyIce(tileX, tileY);
						alreadyIced = false;
					}

				} else { //Move about and destroy adjacent ice blocks

					if (tileY + 1 < Constants.Y_TILEROWS
							&& lvlObj.getTileList().get(tileY + 1).get(tileX)
instanceof IceTile) {

						destroyIce(tileX, tileY + 1);
					}

					if (tileY - 1 >= 0 && lvlObj.getTileList().get(tileY - 1).get(tileX)
instanceof IceTile) {

						destroyIce(tileX, tileY - 1);
					}

					if (tileX - 1 >= 0 && lvlObj.getTileList().get(tileY).get(tileX - 1)
instanceof IceTile) {

						destroyIce(tileX - 1, tileY);
					}

					if (tileX + 1 < Constants.X_TILECOLS
							&& lvlObj.getTileList().get(tileY).get(tileX + 1)
instanceof IceTile) {

						destroyIce(tileX + 1, tileY);
					}

					//Allows chili to move if the chili will not move off the edge of the
level,
					//and if chili movement will not be obstructed by ice.
```

```
                                              if (lastDir == 'd' && tileY + 1 < Constants.Y_TILEROWS
                                                          && (lvlObj.getTileList().get(tileY + 1).get(tileX)
instanceof TransparentTile
                                                                        || lvlObj.getTileList().get(tileY +
1).get(tileX) instanceof Monster)) {

                                                    moveTilesVer(false, 0.5);

                                              } else if (lastDir == 'u' && tileY - 1 > 0
                                                          && (lvlObj.getTileList().get(tileY - 1).get(tileX)
instanceof TransparentTile
                                                                        || lvlObj.getTileList().get(tileY -
1).get(tileX) instanceof Monster)) {

                                                    moveTilesVer(true, -0.5);

                                              } else if (lastDir == 'l' && tileX - 1 > 0
                                                          && (lvlObj.getTileList().get(tileY).get(tileX - 1)
instanceof TransparentTile
                                                                        || lvlObj.getTileList().get(tileY).get(tileX
- 1) instanceof Monster)) {

                                                    moveTilesHor(false, -0.5);

                                              } else if (lastDir == 'r' && tileX + 1 < Constants.X_TILECOLS
                                                          && (lvlObj.getTileList().get(tileY).get(tileX + 1)
instanceof TransparentTile
                                                                  || lvlObj.getTileList().get(tileY).get(tileX + 1)
instanceof Monster)) {

                                                    moveTilesHor(true, 0.5);
                                              }
                                        }
                                  }

                                  //Keeps chili moving until it has finished traversing its tile
                                  if (!getIsIced() && isX && deltaX() > 0) {
                                        setCenterX(getCenterX() + incr);
                                  } else if (!getIsIced() && deltaY() > 0) {
```

```
                        setCenterY(getCenterY() + incr);
                    } else if (deltaX() == 0 && deltaY() == 0) {
                        setRefresh(true, 0);
                    }

                    detectWhenPicked();
                }
            }
        };

        setUpdater(updater);
    }
}
```

Grapes Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Grapes class.
 */

package UFinal.fruit;

import UFinal.Constants;

public class Grapes extends Fruit {

	/**
	 * Constructor for Grapes that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Grapes object created with its respective properties
	 */
	public Grapes(int xTile, int yTile, int width, int height, boolean isFruit) {
		//Passes parameters onto parent
		super("/UFinal/img/grape.png", xTile, yTile, width, height, isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Constructor for Grapes that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Grapes object created with its respective properties
	 */
	public Grapes(int xTile, int yTile, boolean isFruit) {
		//Passes parameters onto parent
		super("/UFinal/img/grape.png", xTile, yTile, (int)(Constants.tileSize/1.2), 
(int)(Constants.tileSize/1.2), isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}
	/**
	 * Sets the refresh() method for Grapes
	 * pre: none
	 * post: refresh() method is set
	 */
	public void makeUpdater() {

		Update updater = new Update() {

			@Override
			public void refresh(boolean isX, double incr) {

				//check for ice and being picked
				checkIsIced();
				detectWhenPicked();
			}
		};

		setUpdater(updater);
	}
}
```

Kiwi Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Kiwi class.
 */

package UFinal.fruit;

import UFinal.Constants;
import UFinal.monsters.Monster;
import UFinal.tiles.TransparentTile;

public class Kiwi extends Fruit {

	//Create variables
	private char lastDir;
	private boolean alreadyUpdated;

	/**
	 * Constructor for Kiwi that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Kiwi object created with its respective properties
	 */
	public Kiwi(int xTile, int yTile, int width, int height, boolean isFruit) {

		//passes parameters to parent
		super("/UFinal/img/kiwi.png", xTile, yTile, width, height, isFruit);
		//set variables
		setInitDir();
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Constructor for Kiwi that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Kiwi object created with its respective properties
	 */
	public Kiwi(int xTile, int yTile, boolean isFruit) {

		//passes parameters to parent
		super("/UFinal/img/kiwi.png", xTile, yTile, (int) (Constants.tileSize / 1.2), (int) (Constants.tileSize
/ 1.8),
				isFruit);
		//set variables
		setInitDir();
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Randomly sets the initial direction for the fruit to head
	 * pre: none
	 * post: initial direction set
	 */
	public void setInitDir() {

		int rand = (int) (Math.random() * 4);

		if (rand == 0) {
			lastDir = 'd';
		} else if (rand == 1) {
			lastDir = 'u';
		} else if (rand == 2) {
			lastDir = 'r';
		} else if (rand == 3) {
			lastDir = 'l';
		}
	}

	/**
	 * Sets the refresh() method to make the Kiwi move randomly
	 * pre: none
	 * post: refresh() method is set and Kiwi performs its respective actions
	 */
	public void makeUpdater() {

		Update updater = new Update() {

			//Set variable
			int rand;

			@Override
			public void refresh(boolean isX, double incr) {

				if (!getIsPicked()) { //Set random direction

					rand = (int) (Math.random() * 6);

					if (rand == 0) {
						lastDir = 'd';
					} else if (rand == 1) {
						lastDir = 'u';
					} else if (rand == 2) {
						lastDir = 'r';
					} else if (rand == 3) {
						lastDir = 'l';
					}

					checkIsIced();

					if (deltaX() == 0 && deltaY() == 0 && !getIsIced()) { //If the kiwi is not traversing 
between tiles

						//If the randomly generated movement is legal (the fruit will not move off of 
the level

						//boundaries, and movement will not be restricted by ice etc), then cause the 
kiwi to move

						if (lastDir == 'd' && tileY + 1 < Constants.Y_TILEROWS
								&& (lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof 
TransparentTile

										|| lvlObj.getTileList().get(tileY + 1).get(tileX) 
instanceof Monster)) {
```

```
                                              moveTilesVer(false, 1);

                                       } else if (lastDir == 'u' && tileY - 1 > 0
                                                    && (lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof
TransparentTile

                                                                  || lvlObj.getTileList().get(tileY - 1).get(tileX)
instanceof Monster)) {

                                              moveTilesVer(true, -1);

                                       } else if (lastDir == 'l' && tileX - 1 > 0
                                                    && (lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof
TransparentTile

                                                                  || lvlObj.getTileList().get(tileY).get(tileX - 1)
instanceof Monster)) {

                                              moveTilesHor(false, -1);

                                       } else if (lastDir == 'r' && tileX + 1 < Constants.X_TILECOLS
                                                    && (lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof
TransparentTile

                                                                  || lvlObj.getTileList().get(tileY).get(tileX + 1)
instanceof Monster)) {

                                              moveTilesHor(true, 1);
                                       }
                                 }

                                 //Keeps kiwi moving until it has finished traversing its tile
                                 if (isX && deltaX() > 0) {
                                       setCenterX(getCenterX() + incr);
                                 } else if (deltaY() > 0) {
                                       setCenterY(getCenterY() + incr);
                                 } else if (deltaX() == 0 && deltaY() == 0) {
                                       setRefresh(true, 0);
                                 }

                                 detectWhenPicked();
                          }
```

```
            }
        };

        setUpdater(updater);
    }
}
```

Lemon Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Lemon class.
 */

package UFinal.fruit;

import UFinal.Constants;

public class Lemon extends Fruit {

	/**
	 * Constructor for Lemon that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Lemon object created with its respective properties
	 */
	public Lemon(int xTile, int yTile, int width, int height, boolean isFruit) {
		//passes parameters onto parent
		super("/UFinal/img/lemon.png", xTile, yTile, width, height, isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Constructor for Lemon that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Lemon object created with its respective properties
	 */
	public Lemon(int xTile, int yTile, boolean isFruit) {
		//passes parameters onto parent
		super("/UFinal/img/lemon.png", xTile, yTile, Constants.tileSize, (int)(Constants.tileSize/1.3),
isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Sets the refresh() method for Lemon
	 * pre: none
	 * post: refresh() method is set
	 */
	public void makeUpdater() {

		Update updater = new Update() {

			@Override
			public void refresh(boolean isX, double incr) {
				//check for ice and being picked
				checkIsIced();
				detectWhenPicked();
			}
		};

		setUpdater(updater);
	}
}
```

Orange Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Orange class.
 */

package UFinal.fruit;

import UFinal.Constants;

public class Orange extends Fruit {

	/**
	 * Constructor for Orange that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Orange object created with its respective properties
	 */
	public Orange(int xTile, int yTile, int width, int height, boolean isFruit) {
		//passes parameters onto parent
		super("/UFinal/img/orange.png", xTile, yTile, width, height, isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Constructor for Orange that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Orange object created with its respective properties
	 */
	public Orange(int xTile, int yTile, boolean isFruit) {
		//passes parameterso onto parent
		super("/UFinal/img/orange.png", xTile, yTile, Constants.tileSize, Constants.tileSize, isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}
	/**
	 * Sets the refresh() method for Orange
	 * pre: none
	 * post: refresh() method is set
	 */
	public void makeUpdater() {

		Update updater = new Update() {

			@Override
			public void refresh(boolean isX, double incr) {
				//check for ice and getting picked
				checkIsIced();
				detectWhenPicked();
			}
		};

		setUpdater(updater);
	}
}
```

Peach Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Peach class.
 */

package UFinal.fruit;

import UFinal.Constants;

public class Peach extends Fruit {

	/**
	 * Constructor for Peach that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Peach object created with its respective properties
	 */
	public Peach(int xTile, int yTile, int width, int height, boolean isFruit) {
		//passes parameters onto parent
		super("/UFinal/img/peach.png", xTile, yTile, width, height, isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Constructor for Peach that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Peach object created with its respective properties
	 */
	public Peach(int xTile, int yTile, boolean isFruit) {
		//passes parameters onto parent
		super("/UFinal/img/peach.png", xTile, yTile, Constants.tileSize, (int)(Constants.tileSize), isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}
	/**
	 * Sets the refresh() method for Peach
	 * pre: none
	 * post: refresh() method is set
	 */
	public void makeUpdater() {

		Update updater = new Update() {

			@Override
			public void refresh(boolean isX, double incr) {
				//checks for ice and getting picked
				checkIsIced();
				detectWhenPicked();
			}
		};

		setUpdater(updater);
	}

}
```

Strawberry Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Strawberry class.
 */

package UFinal.fruit;

import UFinal.Constants;
import UFinal.monsters.Monster;
import UFinal.tiles.TransparentTile;

public class Strawberry extends Fruit {

	//Creating variables
	private char lastDir;
	private boolean alreadyUpdated;

	/**
	 * Constructor for Strawberry that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Strawberry object created with its respective properties
	 */
	public Strawberry(int xTile, int yTile, int width, int height, boolean isFruit) {
		//passes parameters onto parent
		super("/UFinal/img/Strawberry.png", xTile, yTile, width, height, isFruit);
		setInitDir();
		alreadyUpdated = false;
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Constructor for Strawberry that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Strawberry object created with its respective properties
	 */
	public Strawberry(int xTile, int yTile, boolean isFruit) {
		//passes parameters onto parent
		super("/UFinal/img/Strawberry.png", xTile, yTile, (int)(Constants.tileSize/1.2),
(int)(Constants.tileSize/1.2), isFruit);
		setInitDir();
		alreadyUpdated = false;
		thisFruit = this;
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Randomly sets the initial direction for the fruit to head
	 * pre: none
	 * post: initial direction set
	 */
	public void setInitDir() {

		int rand = (int)(Math.random()*4);

		if(rand == 0) {
			lastDir = 'd';
		} else if (rand == 1) {
			lastDir = 'u';
		} else if (rand == 2) {
			lastDir = 'r';
		} else if (rand == 3) {
			lastDir = 'l';
		}
	}

	/**
	 * Sets the refresh() method to make the Strawberry move randomly
	 * pre: none
	 * post: refresh() method is set and Strawberry performs its respective actions
	 */
	public void makeUpdater() {
```

```
            Update updater = new Update() {

                //Set variable
                int rand;

                @Override
                public void refresh(boolean isX, double incr) {

                    if(!getIsPicked()) {

                        rand = (int)(Math.random()*6); //randomly generate a direction

                        if(rand == 0) {
                            lastDir = 'd';
                        } else if (rand == 1) {
                            lastDir = 'u';
                        } else if (rand == 2) {
                            lastDir = 'r';
                        } else if (rand == 3) {
                            lastDir = 'l';
                        }

                        checkIsIced();

                        if(deltaX() == 0 && deltaY() == 0 && !getIsIced()) { //If the fruit is not traversing
    between tiles

                            //If the randomly generated movement is legal (the fruit will not move off of
    the level

                            //boundaries, and movement will not be restricted by ice etc), then cause the
    strawberry to move

                            if(lastDir == 'd' && tileY + 1 < Constants.Y_TILEROWS
                                && (lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof
    TransparentTile

                                    || lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof
    Monster)) {

                                moveTilesVer(false, 0.4);
                            } else if (lastDir == 'u' && tileY - 1 > 0
                                && (lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof
    TransparentTile
                                    || lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof
    Monster)) {

                                moveTilesVer(true, -0.4);

                            } else if (lastDir == 'l' && tileX - 1 > 0
                                && (lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof
    TransparentTile
                                    || lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof
    Monster)) {

                                moveTilesHor(false, -0.4);

                            } else if (lastDir == 'r' && tileX + 1 < Constants.X_TILECOLS
                                && (lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof
    TransparentTile
                                    || lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof
    Monster)) {

                                moveTilesHor(true, 0.4);
                            }
                        }

                        //Keep the fruit moving until it finishes traversing its tile
                        if(isX && deltaX() > 0) {
                            setCenterX(getCenterX() + incr);
                        } else if (deltaY() > 0){
                            setCenterY(getCenterY() + incr);
                        }

                        if (deltaX() == 0 && deltaY() == 0) {
                            setRefresh(true, 0);
                        }

                        detectWhenPicked();
```

```
                    }
                }
            };

            setUpdater(updater);
        }
    }
```

Watermelon Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Watermelon class.
 */

package UFinal.fruit;

import UFinal.Constants;

public class Watermelon extends Fruit {

	/**
	 * Constructor for Watermelon that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Watermelon object created with its respective properties
	 */
	public Watermelon(int xTile, int yTile, int width, int height, boolean isFruit) {
		//passes parameters to parent
		super("/UFinal/img/watermelon.png", xTile, yTile, width, height, isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Constructor for Watermelon that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Watermelon object created with its respective properties
	 */
	public Watermelon(int xTile, int yTile, boolean isFruit) {
		//passes parameters to parent
		super("/UFinal/img/watermelon.png", xTile, yTile, Constants.tileSize, (int)(Constants.tileSize/2.5),
isFruit);
		setRefresh(true, 0);
		makeUpdater();
	}

	/**
	 * Sets the refresh() method for Watermelon
	 * pre: none
	 * post: refresh() method is set
	 */
	public void makeUpdater() {

		Update updater = new Update() {

			@Override
			public void refresh(boolean isX, double incr) {
				//check for ice and getting picked
				checkIsIced();
				detectWhenPicked();
			}
		};

		setUpdater(updater);
	}
}
```

Monster Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Monster class.
 */

package UFinal.monsters;

import UFinal.BadIceCreamFour;
import UFinal.Constants;
import UFinal.characters.Characters;
import UFinal.lvl.LevelLoader;
import UFinal.tiles.Tile;
import UFinal.tiles.TransparentTile;

public class Monster extends Tile {

	//Create objects and variables
	BadIceCreamFour bic4Obj = BadIceCreamFour.bic4;
	LevelLoader lvlObj = bic4Obj.LvlLoader;
	Monster thisMonster;
	Characters tempChar;

	/**
	 * Constructor for Monster that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Monster object created with its respective properties
	 */
	public Monster(String url, int xTile, int yTile, int width, int height) {

		//passes parameters onto Tile
		super(url, xTile, yTile, width, height);
		thisMonster = this;
	}
	/**
	 * Constructor for Monsters that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Monster object created with its respective properties
	 */
	public Monster(String url, int xTile, int yTile) {

		//passes parameters onto Tile
		super(url, xTile, yTile);
		thisMonster = this;
	}

	/**
	 * Causes a monster to move between tiles horizontally. True indicates motion toward the right of the screen
	 * pre: monster's movement will not be obstructed by an adjacent ice block and it is not going off the edge of
the level
	 * post: monster moves left or right one tile
	 */
	public void moveTilesHor(boolean isRight, double incR) {

		if(isRight) { //if monster is set to go right

			//Set goal x coordinate, set pixel increment, and change object's position properties
			setGoalX((tileX + 1)*Constants.tileSize + Constants.tileSize/2);
			setRefresh(true, incR);
			tileX++;

		} else { //left

			//Set goal x coordinate, set pixel increment, and change object's position properties
			setGoalX((tileX - 1)*Constants.tileSize + Constants.tileSize/2);
			setRefresh(true, incR);
			tileX--;
		}
	}

	public void moveTilesVer(boolean isUp, double incR) {

		if(isUp) { //if monster is set to go up
```

```
            //Set goal y coordinate, set pixel increment, change object's layer
            //and change object's position properties
            setGoalY((tileY)*Constants.tileSize);
            setRefresh(false, incR);
            lvlObj.getPane().setLayer(thisMonster, new Integer((tileY - 1)*10 + 2));
            tileY--;

        } else { //down

            //Set goal y coordinate, set pixel increment, change object's layer
            //and change object's position properties
            setGoalY((tileY + 2)*Constants.tileSize);
            setRefresh(false, incR);
            lvlObj.getPane().setLayer(thisMonster, new Integer((tileY + 1)*10 + 2));
            tileY++;
        }
    }

    /**
     * Checks if the player is occupying the same tile as the monster.
     * pre: none
     * post: player is killed if they are occupying the same pile
     */
    public void checkForKill() {

        //if the current tile is being occupied by a player
        if(lvlObj.getTileList().get(tileY).get(tileX) instanceof Characters) {

            ((Characters) lvlObj.getTileList().get(tileY).get(tileX)).isDead(); //Set character to dead

            Constants.numDead++; //increase the recorded number of dead

            if(Constants.numDead >= Constants.numPlayers) {
                lvlObj.gameLost(); //trigger a loss when all players are dead
            }
        }
    }
```

```java
        /**
         * Destroys an ice block at the tile specified
         * pre: xTile and yTile are nonnegative
         * post: player is killed if they are occupying the same pile
         */
        public void destroyIce(int xTile, int yTile) {

                //Replace player with transparent tile in the Tile ArrayList and remove the player from the window
                TransparentTile transTile = new TransparentTile(xTile, yTile);

                lvlObj.getPane().remove(lvlObj.getTileList().get(yTile).get(xTile));
                lvlObj.getPane().add(transTile, new Integer((yTile)*10));
                lvlObj.getTileList().get(yTile).set(xTile, transTile);
                lvlObj.getPane().revalidate();
                lvlObj.getPane().repaint();

        }
}
```

BlueCow Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - BlueCow class.
 */

package UFinal.monsters;

import java.awt.image.BufferedImage;
import java.io.IOException;

import javax.imageio.ImageIO;

import UFinal.Constants;
import UFinal.characters.Characters;
import UFinal.tiles.TransparentTile;
import UFinal.tiles.IceTile;

public class BlueCow extends Monster {

	//Create objects and variables
	private char lastDir;
	private double minDist;
	private int rand, init, width, height;
	private boolean isRandom, canChase;

	private Characters playerToChase;

	/**
	 * Constructor for BlueCow that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: BlueCow object created with its respective properties
	 */
	public BlueCow(int xTile, int yTile, int width, int height) {
		//parameters passed to parent
		super("/UFinal/img/bcow.png", xTile, yTile, width, height);
		init = Constants.refreshCount;
		width = getWidth();
		height = getHeight();
		isRandom = true;
		canChase = true;
		minDist = Math.sqrt(Math.pow(lvlObj.getPlayerList().get(0).getTileX() - tileX, 2)
					+ Math.pow(lvlObj.getPlayerList().get(0).getTileY() - tileY, 2));
		setInitDir();
		setUpdater();
	}

	/**
	 * Constructor for BlueCow that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: BlueCow object created with its respective properties
	 */
	public BlueCow(int xTile, int yTile) {
		//parameters passed to parent
		super("/UFinal/img/bcow.png", xTile, yTile);
		init = Constants.refreshCount;
		width = getWidth();
		height = getHeight();
		isRandom = true;
		canChase = true;
		minDist = Math.sqrt(Math.pow(lvlObj.getPlayerList().get(0).getTileX() - tileX, 2)
					+ Math.pow(lvlObj.getPlayerList().get(0).getTileY() - tileY, 2));
		setInitDir();
		setUpdater();
	}

	/**
	 * Changes character icon
	 * pre: w and h are positive
	 * post: character icon changed according to the inputting url
	 */
	public void changeIcon(String url, int w, int h) {

		try { //Create new buffered image using the URL and set the enemy icon to it
			BufferedImage bfImg = ImageIO.read(getClass().getResource(url));
			setIcon(resizeImg(bfImg, w, h));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Randomly sets the initial direction for the fruit to head
	 * pre: none
	 * post: initial direction set
	 */
	public void setInitDir() {

		rand = (int) (Math.random() * 8);

		if (rand == 0) {
			lastDir = 'd';
		} else if (rand == 1) {
			lastDir = 'u';
		} else if (rand == 2) {
			lastDir = 'r';
		} else if (rand == 3) {
			lastDir = 'l';
		}
	}

	/**
	 * Sets the refresh() method to make the BlueCow alternate between moving randomly and following the player
	 * every 5 secs
	 * pre: none
	 * post: refresh() method is set and BlueCow performs its respective actions
	 */
	public void setUpdater() {

		Update updater = new Update() {

			@Override
			public void refresh(boolean isX, double incr) {
```

```java
//Toggle between random movement and active chasing underground every 5 secs
if(Math.abs(Constants.refreshCount) - init >= 5000) {

    //Change icon accordingly to each change
    if(isRandom) {
        //switches to undergound active chasing
        changeIcon("/UFinal/img/circle.png", 50 , 50);
        setCoordinates(false);
        isRandom = false;
        lvlObj.getPane().setLayer(thisMonster, new Integer(tileY*10 + 2));
        init = Constants.refreshCount;

    } else {
        //switches to above ground random movement
        if(deltaX() == 0 && deltaY() == 0) { //Changes back when the monster is
finished traversing tiles

            if(lvlObj.getTileList().get(tileY).get(tileX) instanceof IceTile) {
                destroyIce(tileX, tileY); //Destroys ice if it is sitting atop
where the monster wants to come out
            }

            changeIcon("/UFinal/img/bcow.png", width, height);
            setCoordinates(false);
            lvlObj.getPane().setLayer(thisMonster, new Integer(tileY*10 + 2));
            isRandom = true;
            init = Constants.refreshCount;
        }
    }
}

if(isRandom) { //If moving about randomly

    checkForKill();

    rand = (int) (Math.random() * 8);

    if (rand == 0) { //Set random direction
        lastDir = 'd';
    } else if (rand == 1) {
```

```
                                                lastDir = 'u';
                                        } else if (rand == 2) {
                                                lastDir = 'r';
                                        } else if (rand == 3) {
                                                lastDir = 'l';
                                        }

                                        if (deltaX() == 0 && deltaY() == 0) { //If the monster is not traversing between
tiles

                                                //If the randomly generated movement is legal (the monster will not move off
of the level
                                                //boundaries, and movement will not be restricted by ice etc), then cause the
monster to move

                                                if (lastDir == 'd' && tileY + 1 < Constants.Y_TILEROWS
                                                        && (lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof
TransparentTile
                                                                || lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof
Characters)) {

                                                        moveTilesVer(false, 0.65);

                                                } else if (lastDir == 'u' && tileY - 1 >= 0
                                                        && (lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof
TransparentTile
                                                                || lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof
Characters)) {

                                                        moveTilesVer(true, -0.65);

                                                } else if (lastDir == 'l' && tileX - 1 > 0 && tileX - 1 >= 0
                                                        && (lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof
TransparentTile
                                                                || lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof
Characters)) {

                                                        moveTilesHor(false, -0.65);
```

```
                                } else if (lastDir == 'r' && tileX + 1 < Constants.X_TILECOLS
                                    && (lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof
TransparentTile
                                        ||lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof
Characters)) {

                                    moveTilesHor(true, 0.65);
                                }
                            }

                    } else { //If the monster is actively chasing

                            if(Constants.numPlayers > 1) { //Determing the closest player if there are multiple

                                for(int i = 0; i < lvlObj.getPlayerList().size(); i++) {

                                    if(lvlObj.getPlayer(i).getIsAlive() &&
Math.sqrt(Math.pow(lvlObj.getPlayer(i).getTileX() - tileX, 2)
                                        + Math.pow(lvlObj.getPlayer(i).getTileY() - tileY, 2)) < minDist)
{
                                        //Use pythagorean theorem to figure out the minimum distance
                                        minDist = Math.sqrt(Math.pow(lvlObj.getPlayer(i).getTileX() -
tileX, 2)
                                                + Math.pow(lvlObj.getPlayer(i).getTileY() - tileY,
2));

                                        playerToChase = lvlObj.getPlayer(i);
                                    }

                                    if(i == lvlObj.getPlayerList().size()
&& !lvlObj.getPlayer(i).getIsAlive()) {
                                        setRefresh(true, 0); //Does not move if all players are dead
                                    }
                                }

                            } else if (lvlObj.getPlayer(0).getIsAlive()) {
                                playerToChase = lvlObj.getPlayerList().get(0); //Chase the first player if
theres only 1
                            } else {
```

```
                                        canChase = false;
                                        setRefresh(true, 0); //Stop moving if the only player has died
                                }

                                if(deltaX() == 0 && deltaY() == 0 && canChase) {

                                        //Follows the player until the monster (underground) and the player occupy the
same tile
                                        if(Math.abs(playerToChase.getTileX() - tileX) >=
Math.abs(playerToChase.getTileY() - tileY)) {

                                                if(playerToChase.getTileX() - tileX > 0) {
                                                        moveTilesHor(true, 0.75);
                                                } else if (playerToChase.getTileX() - tileX == 0){
                                                        setRefresh(true, 0);
                                                } else {
                                                        moveTilesHor(false, -0.75);
                                                }

                                        } else {

                                                if(playerToChase.getTileY() - tileY > 0) {
                                                        moveTilesVer(false, 0.75);
                                                } else {
                                                        moveTilesVer(true, -0.75);
                                                }
                                        }
                                }
                        }

                        //Keeps the monster moving until it has finished traversing its tile
                        if (isX && deltaX() > 0) {
                                setCenterX(getCenterX() + incr);
                        } else if (deltaY() > 0) {
                                setCenterY(getCenterY() + incr);
                        } else if (deltaX() == 0 && deltaY() == 0) {
                                setRefresh(true, 0);
                        }
                }
                };

                setUpdater(updater);
        }

}
```

Cow Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Cow class.
 */

package UFinal.monsters;

import UFinal.Constants;
import UFinal.characters.Characters;
import UFinal.tiles.TransparentTile;

public class Cow extends Monster {

	//Create variables
	private char lastDir;
	private int rand;

	/**
	 * Constructor for Cow that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Cow object created with its respective properties
	 */
	public Cow(int xTile, int yTile, int width, int height) {
		//passes parameters to parent
		super("/UFinal/img/cow.png", xTile, yTile, width, height);
		setInitDir();
		setUpdater();
	}

	/**
	 * Constructor for Cow that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Cow object created with its respective properties
	 */
	public Cow(int xTile, int yTile) {
        //passes parameters to parent
        super("/UFinal/img/cow.png", xTile, yTile);
        setInitDir();
        setUpdater();
    }

    /**
     * Randomly sets the initial direction for the fruit to head
     * pre: none
     * post: initial direction set
     */
    public void setInitDir() {

        rand = (int) (Math.random() * 8);

        if (rand == 0) {
            lastDir = 'd';
        } else if (rand == 1) {
            lastDir = 'u';
        } else if (rand == 2) {
            lastDir = 'r';
        } else if (rand == 3) {
            lastDir = 'l';
        }
    }

    /**
     * Sets the refresh() method to make the Cow move randomly
     * pre: none
     * post: refresh() method is set and Cow performs its respective actions
     */
    public void setUpdater() {

        Update updater = new Update() {

            @Override
            public void refresh(boolean isX, double incr) {

                checkForKill();
```

```
                        rand = (int) (Math.random() * 8); //Set random direction

                        if (rand == 0) {
                                lastDir = 'd';
                        } else if (rand == 1) {
                                lastDir = 'u';
                        } else if (rand == 2) {
                                lastDir = 'r';
                        } else if (rand == 3) {
                                lastDir = 'l';
                        }

                        if (deltaX() == 0 && deltaY() == 0) { //If the monster is not traversing between tiles

                                //If the randomly generated movement is legal (the monster will not move off of the
level
                                //boundaries, and movement will not be restricted by ice etc), then cause the monster
to move

                                if (lastDir == 'd' && tileY + 1 < Constants.Y_TILEROWS
                                        && (lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof TransparentTile
                                                || lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof
Characters)) {

                                        moveTilesVer(false, 0.5);

                                } else if (lastDir == 'u' && tileY - 1 >= 0
                                        && (lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof TransparentTile
                                                || lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof
Characters)) {

                                        moveTilesVer(true, -0.5);

                                } else if (lastDir == 'l' && tileX - 1 >= 0
                                        && (lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof TransparentTile
                                                || lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof
Characters)) {
```

```
                                        moveTilesHor(false, -0.5);

                                   } else if (lastDir == 'r' && tileX + 1 < Constants.X_TILECOLS
                                        && (lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof TransparentTile
                                             ||lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof Characters))
{

                                        moveTilesHor(true, 0.5);
                                   }
                              }

                              //Keeps the monster moving until it has finished traversing its tile
                              if (isX && deltaX() > 0) {
                                   setCenterX(getCenterX() + incr);
                              } else if (deltaY() > 0) {
                                   setCenterY(getCenterY() + incr);
                              } else if (deltaX() == 0 && deltaY() == 0) {
                                   setRefresh(true, 0);
                              }
                         }
                    };

                    setUpdater(updater);
               }
}
```

OrangeSquid Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - OrangeSquid class.
 */

package UFinal.monsters;

import UFinal.Constants;
import UFinal.characters.Characters;
import UFinal.tiles.IceTile;
import UFinal.tiles.TransparentTile;

public class OrangeSquid extends Monster {

	//Make variables
	private char lastDir;
	private int rand;

	/**
	 * Constructor for OrangeSquid that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: OrangeSquid object created with its respective properties
	 */
	public OrangeSquid(int xTile, int yTile, int width, int height) {
		//pass parameters to parent
		super("/UFinal/img/osquid.png", xTile, yTile, width, height);
		setInitDir();
		setUpdater();
	}

	/**
	 * Constructor for OrangeSquid that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: OrangeSquid object created with its respective properties
 */
public OrangeSquid(int xTile, int yTile) {
      //pass parameters to parent
      super("/UFinal/img/osquid.png", xTile, yTile);
      setInitDir();
      setUpdater();
}

/**
 * Randomly sets the initial direction for the fruit to head
 * pre: none
 * post: initial direction set
 */
public void setInitDir() {

      rand = (int) (Math.random() * 8);

      if (rand == 0) {
            lastDir = 'd';
      } else if (rand == 1) {
            lastDir = 'u';
      } else if (rand == 2) {
            lastDir = 'r';
      } else if (rand == 3) {
            lastDir = 'l';
      }
}

/**
 * Sets the refresh() method to make the OrangeSquid move randomly and jump over ice blocks
 * pre: none
 * post: refresh() method is set and OrangeSquid performs its respective actions
 */
public void setUpdater() {

      Update updater = new Update() {

            @Override
            public void refresh(boolean isX, double incr) {
```

```
                          checkForKill();
                          rand = (int) (Math.random() * 8); //set Random direction

                          if (rand == 0) {
                                 lastDir = 'd';
                          } else if (rand == 1) {
                                 lastDir = 'u';
                          } else if (rand == 2) {
                                 lastDir = 'r';
                          } else if (rand == 3) {
                                 lastDir = 'l';
                          }

                          if (deltaX() == 0 && deltaY() == 0) { //If the monster is not traversing between tiles

                                 //If the randomly generated movement is legal (the monster will not move off of the
level
                                 //boundaries, and movement will not be restricted by ice etc), then cause the monster
to move.
                                 //If there is obstruction by ice, the monster my remove one adjacent block that is
restricting movement.

                                 if (lastDir == 'd' && tileY + 1 < Constants.Y_TILEROWS) {

                                        if(lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof TransparentTile
                                               || lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof Characters)
{

                                               moveTilesVer(false, 0.55);
                                        } else if (lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof IceTile)
{

                                               destroyIce(tileX, tileY + 1);
                                        }

                                 } else if (lastDir == 'u' && tileY - 1 >= 0) {

                                        if(lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof TransparentTile
                                                        || lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof Characters)
        {
                                                        moveTilesVer(true, -0.55);
                                                } else if (lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof IceTile)
        {
                                                        destroyIce(tileX, tileY - 1);
                                                }

                                        } else if (lastDir == 'l' && tileX - 1 > 0) {

                                                if(lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof TransparentTile
                                                        || lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof Characters)
        {
                                                        moveTilesHor(false, -0.55);
                                                } else if (lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof IceTile)
        {
                                                        destroyIce(tileX - 1, tileY);
                                                }

                                        } else if (lastDir == 'r' && tileX + 1 < Constants.X_TILECOLS) {

                                                if(lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof TransparentTile
                                                        || lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof Characters)
        {
                                                        moveTilesHor(true, 0.55);
                                                } else if (lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof IceTile)
        {
                                                        destroyIce(tileX + 1, tileY);
                                                }
                                        }
                                }

                                //Keep monster moving until it has finished traversing its tile
```

```
                    if (isX && deltaX() > 0) {
                        setCenterX(getCenterX() + incr);
                    } else if (deltaY() > 0) {
                        setCenterY(getCenterY() + incr);
                    } else if (deltaX() == 0 && deltaY() == 0) {
                        setRefresh(true, 0);
                    }
                }
            };

            setUpdater(updater);
        }
    }
```

Egg Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Egg class.
 */

package UFinal.monsters;

import UFinal.Constants;
import UFinal.characters.Characters;
import UFinal.tiles.IceTile;
import UFinal.tiles.TransparentTile;

public class Egg extends Monster {

	//Create variables
	private char lastDir;
	private int rand, jumpCounter;
	private boolean isFinishedJumping;

	/**
	 * Constructor for Egg that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Egg object created with its respective properties
	 */
	public Egg(int xTile, int yTile, int width, int height) {
		//passes parameters to parents
		super("/UFinal/img/egg.png", xTile, yTile, width, height);
		isFinishedJumping = true;
		jumpCounter = 0;
		setInitDir();
		setUpdater();
	}

	/**
	 * Constructor for Egg that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Egg object created with its respective properties
	 */
	public Egg(int xTile, int yTile) {
		//passes parameters to parents
		super("/UFinal/img/egg.png", xTile, yTile);
		isFinishedJumping = true;
		jumpCounter = 0;
		setInitDir();
		setUpdater();
	}

	/**
	 * Randomly sets the initial direction for the fruit to head
	 * pre: none
	 * post: initial direction set
	 */
	public void setInitDir() {

		rand = (int) (Math.random() * 8);

		if (rand == 0) {
			lastDir = 'd';
		} else if (rand == 1) {
			lastDir = 'u';
		} else if (rand == 2) {
			lastDir = 'r';
		} else if (rand == 3) {
			lastDir = 'l';
		}
	}

	/**
	 * Sets the refresh() method to make the Egg move randomly and jump over ice blocks
	 * pre: none
	 * post: refresh() method is set and Egg performs its respective actions
	 */
	public void setUpdater() {
        Update updater = new Update() {

                @Override
                public void refresh(boolean isX, double incr) {

                        if(isFinishedJumping) { //Sets random direction if not currently jumping

                                checkForKill();

                                rand = (int) (Math.random() * 6);

                                if (rand == 0) {
                                        lastDir = 'd';
                                } else if (rand == 1) {
                                        lastDir = 'u';
                                } else if (rand == 2) {
                                        lastDir = 'r';
                                } else if (rand == 3) {
                                        lastDir = 'l';
                                }
                        }

                        if (deltaX() == 0 && deltaY() == 0) { //If the monster is not traversing between tiles

                                if(jumpCounter == -1) { //set flags to a no-jump status when jumping is finished
                                        isFinishedJumping = true;
                                        jumpCounter++;
                                }

                                //If the randomly generated movement is legal (the monster will not move off of the
level
                                //boundaries, and movement will not be restricted by ice etc), then cause the monster
to move

                                if (lastDir == 'd' && tileY + 1 < Constants.Y_TILEROWS) {

                                        if(lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof TransparentTile
                                                || lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof Characters)
{
```

```
                                                moveTilesVer(false, 0.75);
                                        } else if (tileY + 2 < Constants.Y_TILEROWS
                                                        && lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof
IceTile
                                                        && (lvlObj.getTileList().get(tileY + 2).get(tileX) instanceof
TransparentTile
                                                                || lvlObj.getTileList().get(tileY + 2).get(tileX)
instanceof Characters)) {

                                                //Jump across an ice tile if the column downwards is only one ice tile
long

                                                if(jumpCounter == 0) {
                                                        setGoalY(tileY*Constants.tileSize + Constants.tileSize/2);
                                                        setRefresh(false, -0.3);
                                                        lvlObj.getPane().setLayer(thisMonster, new Integer((tileY +
1)*10));

                                                        jumpCounter++;
                                                        isFinishedJumping = false;
                                                } else if (jumpCounter == 1) {
                                                        setGoalY((tileY + 2)*Constants.tileSize + Constants.tileSize/2);
                                                        setRefresh(false, 0.6);
                                                        lvlObj.getPane().setLayer(thisMonster, new Integer((tileY +
2)*10));

                                                        jumpCounter++;
                                                } else if (jumpCounter == 2) {
                                                        setGoalY((tileY + 3)*Constants.tileSize);
                                                        setRefresh(false, 0.3);
                                                        tileY += 2;

                                                        jumpCounter = -1;
                                                }
                                        }

                                } else if (lastDir == 'u' && tileY - 1 >= 0) {

                                        if(lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof TransparentTile
                                                || lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof Characters)
{
```

```
                                              moveTilesVer(true, -0.75);
                                       } else if (tileY - 2 >= 0
                                                     && lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof
IceTile
                                                     && (lvlObj.getTileList().get(tileY - 2).get(tileX) instanceof
TransparentTile
                                                            || lvlObj.getTileList().get(tileY - 2).get(tileX)
instanceof Characters)) {

                                              //Jump across an ice tile if the column upwards is only one ice tile
long

                                              if(jumpCounter == 0) {
                                                     setGoalY(tileY*Constants.tileSize + Constants.tileSize/2);
                                                     setRefresh(false, -0.3);
                                                     jumpCounter++;
                                                     isFinishedJumping = false;
                                              } else if (jumpCounter == 1) {
                                                     setGoalY((tileY - 2)*Constants.tileSize + Constants.tileSize/2);
                                                     setRefresh(false, -0.6);
                                                     jumpCounter++;
                                              } else if (jumpCounter == 2) {
                                                     setGoalY((tileY - 1)*Constants.tileSize);
                                                     setRefresh(false, 0.3);
                                                     tileY -= 2;
                                                     lvlObj.getPane().setLayer(thisMonster, new Integer(tileY*10));

                                                     jumpCounter = -1;
                                              }
                                       }

                                } else if (lastDir == 'l' && tileX - 1 >= 0) {

                                       if(lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof TransparentTile
                                              || lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof Characters)
{

                                              moveTilesHor(false, -0.75);
                                       } else if (tileX - 2 >= 0
                                                   && lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof
IceTile
                                                   && (lvlObj.getTileList().get(tileY).get(tileX - 2) instanceof
TransparentTile
                                                          || lvlObj.getTileList().get(tileY).get(tileX - 2)
instanceof Characters)) {

                                             //Jump across an ice tile if the row leftwards is only one ice tile long

                                             if(jumpCounter == 0) {
                                                   setGoalY(tileY*Constants.tileSize);
                                                   setRefresh(false, -0.3);
                                                   jumpCounter++;
                                                   isFinishedJumping = false;
                                             } else if (jumpCounter == 1) {
                                                   setGoalX((tileX - 2)*Constants.tileSize + Constants.tileSize/2);
                                                   setRefresh(true, -0.6);
                                                   jumpCounter++;
                                             } else if (jumpCounter == 2) {
                                                   setGoalY((tileY + 1)*Constants.tileSize);
                                                   setRefresh(false, 0.3);
                                                   tileX -= 2;
                                                   jumpCounter = -1;
                                             }
                                       }

                                 } else if (lastDir == 'r' && tileX + 1 < Constants.X_TILECOLS) {

                                       if(lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof TransparentTile
                                             || lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof Characters)
{
                                             moveTilesHor(true, 0.75);
                                       } else if (tileX + 2 < Constants.X_TILECOLS
                                                   && lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof
IceTile
                                                   && (lvlObj.getTileList().get(tileY).get(tileX + 2) instanceof
TransparentTile
                                                          || lvlObj.getTileList().get(tileY).get(tileX + 2)
instanceof Characters)) {
```

```
                            //Jump across an ice tile if the row rightwards is only one ice tile
long

                            if(jumpCounter == 0) {
                                setGoalY(tileY*Constants.tileSize);
                                setRefresh(false, -0.3);
                                jumpCounter++;
                                isFinishedJumping = false;
                            } else if (jumpCounter == 1) {
                                setGoalX((tileX + 2)*Constants.tileSize + Constants.tileSize/2);
                                setRefresh(true, 0.6);
                                jumpCounter++;
                            } else if (jumpCounter == 2) {
                                setGoalY((tileY + 1)*Constants.tileSize);
                                setRefresh(false, 0.3);
                                tileX += 2;

                                jumpCounter = -1;
                            }
                        }
                    }
                }

                //Keeps monster moving until it's finished traversing its tile
                if (isX && deltaX() > 0) {
                    setCenterX(getCenterX() + incr);
                } else if (deltaY() > 0) {
                    setCenterY(getCenterY() + incr);
                } else if (deltaX() == 0 && deltaY() == 0) {
                    setRefresh(true, 0);
                }
            }
        };

        setUpdater(updater);
    }
}
```

Troll Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - Troll class.
 */

package UFinal.monsters;

import UFinal.Constants;
import UFinal.characters.Characters;
import UFinal.tiles.IceTile;

public class Troll extends Monster {

	//Create variable
	private char lastDir;

	/**
	 * Constructor for Troll that resizes it to a custom width and height.
	 * pre: all integer parameters are positive
	 * post: Troll object created with its respective properties
	 */
	public Troll(int xTile, int yTile, int width, int height) {
		super("/UFinal/img/troll.png", xTile, yTile, width, height);
		setInitDir();
		setUpdater();
	}

	/**
	 * Constructor for Troll that resizes it to the tile width and height for the level
	 * pre: all integer parameters are positive
	 * post: Troll object created with its respective properties
	 */
	public Troll(int xTile, int yTile) {
		super("/UFinal/img/troll.png", xTile, yTile);
		setInitDir();
```

```
        setUpdater();
    }

    /**
     * Randomly sets the initial direction for the fruit to head
     * pre: none
     * post: initial direction set
     */
    public void setInitDir() {

        int rand = (int) (Math.random() * 4);

        if (rand == 0) {
            lastDir = 'd';
        } else if (rand == 1) {
            lastDir = 'u';
        } else if (rand == 2) {
            lastDir = 'r';
        } else if (rand == 3) {
            lastDir = 'l';
        }
    }

    /**
     * Sets the refresh() method to make the Troll to move about in a controlled fashion
     * pre: none
     * post: refresh() method is set and Troll performs its respective actions
     */
    public void setUpdater() {

        Update updater = new Update() {

            //Make variables
            boolean iceOnLeft, iceOnRight, iceAbove, iceBelow;

            @Override
            public void refresh(boolean isX, double incr) {

                checkForKill();
```

```
                                        iceOnLeft = false; //Checks on how many sides there is ice
                                        iceOnRight = false;
                                        iceAbove = false;
                                        iceBelow = false;

                                        if(tileX == 0
                                                || lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof IceTile) {
                                                iceOnLeft = true;
                                        }

                                        if (tileX == Constants.X_TILECOLS - 1
                                                || lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof IceTile) {
                                                iceOnRight = true;
                                        }

                                        if (tileY == 0
                                                || lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof IceTile) {
                                                iceAbove = true;
                                        }

                                        if (tileY == Constants.Y_TILEROWS - 1
                                                || lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof IceTile) {
                                                iceBelow = true;
                                        }

                                        if(deltaX() == 0 && deltaY() == 0) { //If the monster is not traversing between tiles

                                                //Continue walking in the same direction until there is obstruction by ice of the
level boundaries.
                                                //Then, attempt an ordered list of directions based on how many sides are obstructed.
                                                //For example, if the troll is going down and can no longer go down, it will try
going right, then left, then up

                                                if(lastDir == 'd') {

                                                        if(iceBelow) {

                                                                if(iceOnLeft && iceOnRight && iceAbove) {
```

```
                setRefresh(true, 0);
            } else if (iceOnLeft && iceOnRight) {
                lastDir = 'u';
                moveTilesVer(true, -0.33);
            } else if (iceOnLeft) {
                lastDir = 'r';
                moveTilesHor(true, 0.33);
            } else if (iceOnRight) {
                lastDir = 'l';
                moveTilesHor(false, -0.33);
            } else {
                lastDir = 'r';
                moveTilesHor(true, 0.33);
            }

        } else {
            moveTilesVer(false, 0.33);
        }

    } else if(lastDir == 'u') {

        if(iceAbove) {

            if(iceOnLeft && iceOnRight && iceBelow) {
                setRefresh(true, 0);
            } else if (iceOnLeft && iceOnRight) {
                lastDir = 'd';
                moveTilesVer(false, 0.33);
            } else if (iceOnLeft) {
                lastDir = 'r';
                moveTilesHor(true, 0.33);
            } else if (iceOnRight) {
                lastDir = 'l';
                moveTilesHor(false, -0.33);
            } else {
                lastDir = 'l';
                moveTilesHor(false, -0.33);
            }
        } else {
            moveTilesVer(true, -0.33);
        }

    } else if(lastDir == 'l') {

        if(iceOnLeft) {

            if(iceOnRight && iceAbove && iceBelow) {
                setRefresh(true, 0);
            } else if (iceAbove && iceBelow) {
                lastDir = 'r';
                moveTilesHor(true, 0.33);
            } else if (iceBelow) {
                lastDir = 'u';
                moveTilesVer(true, -0.33);
            } else if (iceAbove) {
                lastDir = 'd';
                moveTilesVer(false, 0.33);
            } else {
                lastDir = 'd';
                moveTilesVer(false, 0.33);
            }

        } else {
            moveTilesHor(false, -0.33);
        }

    } else if(lastDir == 'r') {

        if(iceOnRight) {

            if(iceOnLeft && iceAbove && iceBelow) {
                setRefresh(true, 0);
            } else if (iceAbove && iceBelow) {
                lastDir = 'l';
                moveTilesHor(false, -0.33);
            } else if (iceBelow) {
                lastDir = 'u';
```

```
                                moveTilesVer(true, -0.33);
                            } else if (iceAbove) {
                                lastDir = 'd';
                                moveTilesVer(false, 0.33);
                            } else {
                                lastDir = 'u';
                                moveTilesVer(true, -0.33);
                            }

                        } else {
                            moveTilesHor(true, 0.33);
                        }
                    }
                }

                //Keep the monster moving until it has finished traversing its tile
                if (isX && deltaX() > 0) {
                    setCenterX(getCenterX() + incr);
                } else if (deltaY() > 0) {
                    setCenterY(getCenterY() + incr);
                } else if (deltaX() == 0 && deltaY() == 0) {
                    setRefresh(true, 0);
                }
            }
        };

        setUpdater(updater);
    }
}
```

YellowCow Class

```
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - YellowCow class.
 */

package UFinal.monsters;

import UFinal.Constants;
import UFinal.characters.Characters;
import UFinal.tiles.IceTile;
import UFinal.tiles.TransparentTile;

public class YellowCow extends Monster {

	// Create objects and variables
	private char lastDir;
	private double minDist;
	private Characters playerToChase;

	/**
	 * Constructor for YellowCow that resizes it to a custom width and height. pre:
	 * all integer parameters are positive post: YellowCow object created with its
	 * respective properties
	 */
	public YellowCow(int xTile, int yTile, int width, int height) {
		// passes parameters to parent
		super("/UFinal/img/ycow.png", xTile, yTile, width, height);
		minDist = Math.sqrt(Math.pow(lvlObj.getPlayerList().get(0).getTileX() - tileX, 2)
				+ Math.pow(lvlObj.getPlayerList().get(0).getTileY() - tileY, 2));
		setRefresh(true, 0);
		setUpdater();
	}

	/**
	 * Constructor for YellowCow that resizes it to the tile width and height for
	 * the level pre: all integer parameters are positive post: YellowCow object
	 * created with its respective properties
	 */
	public YellowCow(int xTile, int yTile) {
		// passes parameters to parent
		super("/UFinal/img/ycow.png", xTile, yTile);
		minDist = Math.sqrt(Math.pow(lvlObj.getPlayerList().get(0).getTileX() - tileX, 2)
				+ Math.pow(lvlObj.getPlayerList().get(0).getTileY() - tileY, 2));
		setRefresh(true, 0);
		setUpdater();
	}

	/**
	 * Sets the refresh() method to make the YellowCow follow the player pre: none
	 * post: refresh() method is set and YellowCow performs its respective actions
	 */
	public void setUpdater() {

		Update updater = new Update() {

			// make variables
			boolean iceOnLeft, iceOnRight, iceAbove, iceBelow;

			@Override
			public void refresh(boolean isX, double incr) {

				iceOnLeft = false; // Checks on how many sides there is ice
				iceOnRight = false;
				iceAbove = false;
				iceBelow = false;

				if (tileX == 0 || lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof IceTile) {
					iceOnLeft = true;
				}

				if (tileX == Constants.X_TILECOLS - 1
						|| lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof IceTile) {
					iceOnRight = true;
				}
```

```java
                                        if (tileY == 0 || lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof IceTile) {
                                                iceAbove = true;
                                        }

                                        if (tileY == Constants.Y_TILEROWS - 1
                                                        || lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof IceTile) {
                                                iceBelow = true;
                                        }

                                        if (Constants.numPlayers > 1) { // Determines which player is closest to the monster

                                                for (int i = 0; i < lvlObj.getPlayerList().size(); i++) {

                                                        if (Math.sqrt(Math.pow(lvlObj.getPlayer(i).getTileX() - tileX, 2)
                                                                        + Math.pow(lvlObj.getPlayer(i).getTileY() - tileY, 2)) < minDist)
{

                                                                minDist = Math.sqrt(Math.pow(lvlObj.getPlayer(i).getTileX() - tileX, 2)
                                                                                + Math.pow(lvlObj.getPlayer(i).getTileY() - tileY, 2));

                                                                playerToChase = lvlObj.getPlayer(i);
                                                        }
                                                }

                                        } else { // If there's only one player, just chase that player
                                                playerToChase = lvlObj.getPlayerList().get(0);
                                        }

                                        if (deltaX() == 0 && deltaY() == 0) { //If monster is not traversing between tiles
                                                // Follows the player until they share the same x and y tile coordinates
                                                if (Math.abs(playerToChase.getTileX() - tileX) >= Math.abs(playerToChase.getTileY() -
tileY)) {

                                                        // Go in the direction of the player until there is ice, by which point the
                                                        // monster will try to go around the
                                                        // ice
                                                        if (playerToChase.getTileX() - tileX > 0) { // going right
```

```
		if (iceOnRight) {

			if (iceOnLeft && iceAbove && iceBelow) {
				setRefresh(true, 0);
			} else if (iceAbove && iceBelow) {
				lastDir = 'r';
			} else if (iceAbove) {
				lastDir = 'd';
			} else if (iceBelow) {
				lastDir = 'u';
			} else {
				if (playerToChase.getTileY() - tileY < 0) {
					lastDir = 'u';
				} else if (playerToChase.getTileY() - tileY > 0) {
					lastDir = 'd';
				} else {
					lastDir = 'r';
				}
			}

		} else {
			lastDir = 'r';
		}

	} else { // going left

		if (iceOnLeft) {

			if (iceOnRight && iceAbove && iceBelow) {
				setRefresh(true, 0);
			} else if (iceAbove && iceBelow) {
				lastDir = 'l';
			} else if (iceAbove) {
				lastDir = 'd';
			} else if (iceBelow) {
				lastDir = 'u';
			} else {
				if (playerToChase.getTileY() - tileY < 0) {
					lastDir = 'u';
```

```
                } else if (playerToChase.getTileY() - tileY > 0) {
                    lastDir = 'd';
                } else {
                    lastDir = 'l';
                }
            }

        } else {
            lastDir = 'l';
        }
    }

} else { // going down

    if (playerToChase.getTileY() - tileY > 0) {

        if (iceBelow) {

            if (iceOnRight && iceOnLeft && iceAbove) {
                setRefresh(true, 0);
            } else if (iceOnRight && iceOnLeft) {
                lastDir = 'd';
            } else if (iceOnRight) {
                lastDir = 'l';
            } else if (iceOnLeft) {
                lastDir = 'r';
            } else {
                if (playerToChase.getTileX() - tileX < 0) {
                    lastDir = 'l';
                } else if (playerToChase.getTileX() - tileX > 0) {
                    lastDir = 'r';
                } else {
                    lastDir = 'd';
                }
            }

        } else {
            lastDir = 'd';
        }
				} else { // going up

					if (iceAbove) {

						if (iceOnRight && iceOnLeft && iceBelow) {
							setRefresh(true, 0);
						} else if (iceOnRight && iceOnLeft) {
							lastDir = 'u';
						} else if (iceOnRight) {
							lastDir = 'l';
						} else if (iceOnLeft) {
							lastDir = 'r';
						} else {
							if (playerToChase.getTileX() - tileX < 0) {
								lastDir = 'l';
							} else if (playerToChase.getTileX() - tileX > 0) {
								lastDir = 'r';
							} else {
								lastDir = 'u';
							}
						}

					} else {
						lastDir = 'u';
					}
				}
			}
			// Move YellowCow based on the direction chosen by the program

			// If the randomly generated movement is legal (the monster will not move off of
			// the level
			// boundaries, and movement will not be restricted by ice etc), then cause the
			// monster to move
			if (lastDir == 'd' && tileY + 1 < Constants.Y_TILEROWS
					&& (lvlObj.getTileList().get(tileY + 1).get(tileX) instanceof
TransparentTile
							|| lvlObj.getTileList().get(tileY + 1).get(tileX)
instanceof Characters)) {
```

```
                        moveTilesVer(false, 0.45);
                    } else if (lastDir == 'u' && tileY - 1 >= 0
                            && (lvlObj.getTileList().get(tileY - 1).get(tileX) instanceof
TransparentTile
                                    || lvlObj.getTileList().get(tileY - 1).get(tileX)
instanceof Characters)) {
                        moveTilesVer(true, -0.45);
                    } else if (lastDir == 'r' && tileX + 1 < Constants.X_TILECOLS
                            && (lvlObj.getTileList().get(tileY).get(tileX + 1) instanceof
TransparentTile
                                    || lvlObj.getTileList().get(tileY).get(tileX + 1)
instanceof Characters)) {
                        moveTilesHor(true, 0.45);
                    } else if (lastDir == 'l' && tileX - 1 >= 0
                            && (lvlObj.getTileList().get(tileY).get(tileX - 1) instanceof
TransparentTile
                                    || lvlObj.getTileList().get(tileY).get(tileX - 1)
instanceof Characters)) {
                        moveTilesHor(false, -0.45);
                    }
                }

                // Keep monster moving until it has finished traversing its tile
                if (isX && deltaX() > 0) {
                    setCenterX(getCenterX() + incr);
                } else if (deltaY() > 0) {
                    setCenterY(getCenterY() + incr);
                } else if (deltaX() == 0 && deltaY() == 0) {
                    setRefresh(true, 0);
                }

                checkForKill();
            }
        };

        setUpdater(updater);
    }
}
```

LevelLoader Class

```java
/*
 * Author: Zhengmao Ouyang
 * Class: ICS4U0
 * Honor Code: I pledge that this program represents my own program code. I received help from
 * (no one) in designing and debugging my program.
 * Assignment: UFinal - LevelLoader class.
 */

package UFinal.lvl;

import java.awt.Font;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.awt.image.FilteredImageSource;
import java.awt.image.ImageFilter;
import java.awt.image.ImageProducer;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.Reader;
import java.util.ArrayList;

import javax.imageio.ImageIO;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;
import javax.swing.BorderFactory;
import javax.swing.GrayFilter;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import UFinal.Constants;
import UFinal.SecTimer;
import UFinal.characters.Characters;
import UFinal.fruit.Avocado;
import UFinal.fruit.Chili;
import UFinal.fruit.Fruit;
import UFinal.fruit.Grapes;
import UFinal.fruit.Kiwi;
import UFinal.fruit.Lemon;
import UFinal.fruit.Orange;
import UFinal.fruit.Peach;
import UFinal.fruit.Strawberry;
import UFinal.fruit.Watermelon;
import UFinal.monsters.BlueCow;
import UFinal.monsters.Cow;
import UFinal.monsters.Egg;
import UFinal.monsters.OrangeSquid;
import UFinal.monsters.Troll;
import UFinal.monsters.YellowCow;
import UFinal.tiles.IceTile;
import UFinal.tiles.Tile;
import UFinal.tiles.TransparentTile;

public class LevelLoader extends BufferedReader implements ActionListener {

	//Create objects and variables
	private ArrayList<ArrayList<Tile>> tileList;
	private ArrayList<ArrayList<Fruit>> fruitList;
	private ArrayList<Characters> playerList;
	private ArrayList<Tile> monsterList;
	private ArrayList<Fruit> fruitIndicator;
	private String lvlRow;
	private SecTimer timer;
	private JLayeredPane gamePane;
	private JLabel bg;
	private JButton sfx, music, back, restart;
	private JFormattedTextField userInput, pwdInput;
	private Clip nonLvlMusic, lvlMusic, bop, wonSound, lostSound;
private String[] secretCodes = {"cerone i$ aw3somE", "ICs$U roXXX", "PlS Assign 1 0 0"};
private int levelNumber;

/**
 * Constructor for LevelLoader
 * pre: none.
 * post: LevelLoader object created, with a set file to read from.
 */
public LevelLoader(Reader inputStream) {

      //Passes file parameter to BufferedReader
      super(inputStream);

      //Set ArrayLists
      tileList = new ArrayList<ArrayList<Tile>>();
      monsterList = new ArrayList<Tile>();
      fruitList = new ArrayList<ArrayList<Fruit>>();
      playerList = new ArrayList<Characters>();
    fruitIndicator = new ArrayList<Fruit>();
    gamePane = new JLayeredPane();

    bg = new JLabel(); //Set background

    //Set Buttons
      back = new JButton(new ImageIcon(getClass().getResource(
                   "/UFinal/img/back.png")));
      back.setBounds(1095, 230,
      back.getPreferredSize().width, back.getPreferredSize().height);
      back.addActionListener(this);
      back.setBorder(BorderFactory.createEmptyBorder());
      back.setContentAreaFilled(false);
      back.setFocusable(false);

    sfx = new JButton(new ImageIcon(getClass().getResource(
            "/UFinal/img/sfx.png")));
      sfx.setBounds(1095, 10,
            sfx.getPreferredSize().width, sfx.getPreferredSize().height);
      sfx.setActionCommand("togglesfx");
      sfx.addActionListener(this);
```

```
            sfx.setBorder(BorderFactory.createEmptyBorder());
            sfx.setContentAreaFilled(false);
            sfx.setFocusable(false);

          music = new JButton(new ImageIcon(getClass().getResource(
                "/UFinal/img/music.png")));
            music.setBounds(1095, 120,
                         music.getPreferredSize().width, music.getPreferredSize().height);
            music.setActionCommand("togglemusic");
            music.addActionListener(this);
            music.setBorder(BorderFactory.createEmptyBorder());
            music.setContentAreaFilled(false);
            music.setFocusable(false);

            restart = new JButton(new ImageIcon(getClass().getResource(
                 "/UFinal/img/restart.png")));
            restart.setBounds(1095, 340,
            restart.getPreferredSize().width, restart.getPreferredSize().height);
            restart.addActionListener(this);
            restart.setBorder(BorderFactory.createEmptyBorder());
            restart.setContentAreaFilled(false);
            restart.setFocusable(false);

            //Set text field for initial password
            userInput = new JFormattedTextField();
            userInput.setLocation(560, 500);
            userInput.setBounds(560, 505, 300, 100);
            userInput.setFont(new Font("Arial", Font.BOLD, 26));

            try { //Create audio objects

                  AudioInputStream nlvlMusic =
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/africa.wav"));
                  nonLvlMusic = AudioSystem.getClip();
                  nonLvlMusic.open(nlvlMusic);
                  nonLvlMusic.loop(Clip.LOOP_CONTINUOUSLY);

                  AudioInputStream lvMusic =
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/rain.wav"));
```

```
				lvlMusic = AudioSystem.getClip();
				lvlMusic.open(lvMusic);

				AudioInputStream bopsfx = 
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/blop.wav"));
				bop = AudioSystem.getClip();
				bop.open(bopsfx);

				AudioInputStream winsfx = 
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/winSound.wav"));
				wonSound = AudioSystem.getClip();
				wonSound.open(winsfx);

				AudioInputStream losesfx = 
AudioSystem.getAudioInputStream(getClass().getResource("/UFinal/wav/lostSound.wav"));
				lostSound = AudioSystem.getClip();
				lostSound.open(losesfx);

			} catch (UnsupportedAudioFileException e1) {
				e1.printStackTrace();
			} catch (IOException e1) {
				e1.printStackTrace();
			} catch (LineUnavailableException e) {
				e.printStackTrace();
			}

			gamePane.setBounds(0, 0, 2000, 1000); //Set the game JLayeredPane

			Constants.levelLoaded = false; //Set this boolean flag

			try {
				mark(20000);
			} catch (IOException e) {
				e.printStackTrace();
			}
		}

		/**
		 * Loads the initial password stage of the program.
```

```
 * pre: none
 * post: initial password stage of the program loaded.
 */
public void initPassword() {

	gamePane.removeAll(); //Empty the game JLayeredPane

	//Create text label
	JLabel pwdLabel = new JLabel("Enter password to play: ");
	pwdLabel.setBounds(100, 300, pwdLabel.getPreferredSize().width, pwdLabel.getPreferredSize().height);

	//Create input text field
	pwdInput = new JFormattedTextField();
	pwdInput.setBounds(250, 250, 200, 100);
	pwdInput.setFont(new Font("Arial", Font.BOLD, 26));

	//Create input button
	JButton pwdButton = new JButton("Check password!");
	pwdButton.setBounds(475, 250, 200, 100);
	pwdButton.addActionListener(this);
	pwdButton.setActionCommand("pwdtotitle");

	//Add elements to the window
	gamePane.add(pwdLabel, new Integer(0));
	gamePane.add(pwdInput, new Integer(0));
	gamePane.add(pwdButton, new Integer(0));
}

/**
 * Loads the title screen stage of the game.
 * pre: none
 * post: title screen loaded
 */
public void titleScreen() {

	//Load background and foreground images
	ImageIcon imgIcon = new ImageIcon(getClass().getResource(
      "/UFinal/img/titlebg.gif"));
```

```
		bg.setIcon(new ImageIcon(imgIcon.getImage().getScaledInstance(Constants.FRAME_WIDTH,
Constants.FRAME_HEIGHT, Image.SCALE_DEFAULT)));
		bg.setBounds(-10, -10, bg.getPreferredSize().width, bg.getPreferredSize().height);

		JLabel titleImg = new JLabel(new ImageIcon(getClass().getResource(
	       "/UFinal/img/bic4.png")));
		titleImg.setLocation(330, 100);
		titleImg.setBounds(330, 100,
			titleImg.getPreferredSize().width, titleImg.getPreferredSize().height);

		//Load buttons
		JButton play = new JButton(new ImageIcon(getClass().getResource(
		  "/UFinal/img/play.png")));
		play.setLocation(400, 450);
		play.setBounds(400, 450,
			play.getPreferredSize().width, play.getPreferredSize().height);
		play.setActionCommand("tolevelselect");
		play.addActionListener(this);
		play.setBorder(BorderFactory.createEmptyBorder());
		play.setContentAreaFilled(false);

		JButton howTo = new JButton(new ImageIcon(getClass().getResource(
		    "/UFinal/img/howto.png")));
		howTo.setLocation(540, 450);
		howTo.setBounds(540, 450,
			howTo.getPreferredSize().width, howTo.getPreferredSize().height);
		howTo.setActionCommand("tohowto");
		howTo.addActionListener(this);
		howTo.setBorder(BorderFactory.createEmptyBorder());
		howTo.setContentAreaFilled(false);

		JButton credits = new JButton(new ImageIcon(getClass().getResource(
			"/UFinal/img/cred.png")));
		credits.setLocation(675, 450);
		credits.setBounds(675, 450,
			credits.getPreferredSize().width, credits.getPreferredSize().height);
		credits.setActionCommand("tocredits");
		credits.addActionListener(this);
		credits.setBorder(BorderFactory.createEmptyBorder());
```

```
            credits.setContentAreaFilled(false);

            //Clear the JLayeredPane and add elements to the screen
            gamePane.removeAll();
            gamePane.add(bg, new Integer(-2));
            gamePane.add(titleImg, new Integer(-1));
            gamePane.add(play, new Integer(-1));
            gamePane.add(howTo, new Integer(-1));
            gamePane.add(credits, new Integer(-1));
            gamePane.add(music, new Integer (-1));
            gamePane.add(sfx, new Integer(-1));
      }

      /**
       * Loads the how to stage of the game.
       * pre: none
       * post: how to screen loaded
       */
      public void howTo() {

            //Load background and foreground images
            ImageIcon imgIcon = new ImageIcon(getClass().getResource(
                  "/UFinal/img/selectbg.gif"));
            bg.setIcon(new ImageIcon(imgIcon.getImage().getScaledInstance(Constants.FRAME_WIDTH,
Constants.FRAME_HEIGHT, Image.SCALE_DEFAULT)));
            bg.setBounds(-10, -10, bg.getPreferredSize().width, bg.getPreferredSize().height);

            JLabel howToImg = new JLabel(new ImageIcon(getClass().getResource(
             "/UFinal/img/howToInst.png")));
            howToImg.setLocation(100, 50);
            howToImg.setBounds(100, 50,
            howToImg.getPreferredSize().width, howToImg.getPreferredSize().height);

            //Set command for back button
            back.setActionCommand("totitle");

            //Clear the JLayeredPane and add required elements
            gamePane.removeAll();
            gamePane.add(bg, new Integer(-2));
            gamePane.add(howToImg, new Integer(-1));
            gamePane.add(back, new Integer(0));
            gamePane.add(sfx, new Integer(0));
            gamePane.add(music, new Integer(0));
    }

    /**
     * Loads the credits stage of the game.
     * pre: none
     * post: credits screen loaded
     */
    public void credits() {

            //Load background and foreground images
            ImageIcon imgIcon = new ImageIcon(getClass().getResource(
                        "/UFinal/img/selectbg.gif"));
            bg.setIcon(new ImageIcon(imgIcon.getImage().getScaledInstance(Constants.FRAME_WIDTH,
Constants.FRAME_HEIGHT, Image.SCALE_DEFAULT)));
            bg.setBounds(-10, -10, bg.getPreferredSize().width, bg.getPreferredSize().height);

            JLabel creditsImg = new JLabel(new ImageIcon(getClass().getResource(
                    "/UFinal/img/credits.png")));
            creditsImg.setLocation(100, 50);
            creditsImg.setBounds(100, 50,
            creditsImg.getPreferredSize().width, creditsImg.getPreferredSize().height);

            //Set command for back button
            back.setActionCommand("totitle");

            //Clear the pane and add required elements to it
            gamePane.removeAll();
            gamePane.add(bg, new Integer(-2));
            gamePane.add(creditsImg, new Integer(-1));
            gamePane.add(back, new Integer(0));
            gamePane.add(sfx, new Integer(0));
            gamePane.add(music, new Integer(0));
    }

    /**
```

```
         * Loads the level select stage of the game.
         * pre: none
         * post: level select screen loaded
         */
        public void levelSelect() {

                //Load background images and foreground images
                ImageIcon imgIcon = new ImageIcon(getClass().getResource(
                 "/UFinal/img/selectbg.gif"));
                bg.setIcon(new ImageIcon(imgIcon.getImage().getScaledInstance(Constants.FRAME_WIDTH,
Constants.FRAME_HEIGHT, Image.SCALE_DEFAULT)));
                bg.setBounds(-10, -10, bg.getPreferredSize().width, bg.getPreferredSize().height);

                back.setActionCommand("totitle");

                JLabel selectLabel = new JLabel(new ImageIcon(getClass().getResource(
                      "/UFinal/img/lvlSelect.png")));
                selectLabel.setLocation(425, 15);
                selectLabel.setBounds(425, 15,
                      selectLabel.getPreferredSize().width, selectLabel.getPreferredSize().height);

                gamePane.removeAll(); //Clear the pane and add the images
                gamePane.add(bg, new Integer(-2));
                gamePane.add(sfx, new Integer(-1));
                gamePane.add(music, new Integer(-1));
                gamePane.add(back, new Integer(-1));
                gamePane.add(selectLabel, new Integer(-1));

                for(int i = 0; i < 10; i++) { //Set positions for each button

                      JButton lvlButton = new JButton(new ImageIcon(getClass().getResource(
                            "/UFinal/img/lvl" + (i + 1) + ".png")));
                      lvlButton.setLocation(300 + 130*(i%5), 200 + 120*(i/5));
                      lvlButton.setBounds(300 + 130*(i%5), 200 + 120*(i/5),
                      lvlButton.getPreferredSize().width, lvlButton.getPreferredSize().height);
                      lvlButton.setActionCommand(Integer.toString(i + 1));
                      lvlButton.addActionListener(this);
                      lvlButton.setBorder(BorderFactory.createEmptyBorder());
                      lvlButton.setContentAreaFilled(false);
```

```
			gamePane.add(lvlButton, new Integer(-1)); //add the buttons to the window
		}

		//Load the secret code elements
		JLabel secretCodeLabel = new JLabel(new ImageIcon(getClass().getResource(
			"/UFinal/img/secretcode.png")));
		secretCodeLabel.setBounds(50, 505, secretCodeLabel.getPreferredSize().width,
secretCodeLabel.getPreferredSize().height);

		gamePane.add(userInput, new Integer(-1));

		JButton codeButton = new JButton(new ImageIcon(getClass().getResource(
			"/UFinal/img/search.png")));
		codeButton.setLocation(850, 500);
		codeButton.setBounds(850, 500,
		codeButton.getPreferredSize().width, codeButton.getPreferredSize().height);
		codeButton.setActionCommand("secret");
		codeButton.addActionListener(this);
		codeButton.setBorder(BorderFactory.createEmptyBorder());
		codeButton.setContentAreaFilled(false);

		gamePane.add(secretCodeLabel, new Integer(-1));
		gamePane.add(codeButton, new Integer(-1));
	}

	/**
	 * Reads the text file to load a level
	 * pre: num is positive
	 * post: level ArrayLists loaded
	 */
	public void readFile(int num) {

		//Set variables
		levelNumber = num;
		int numArrays = -2;
		int numLines = -1;
		Constants.fruitLvl = 0;
```

```java
            try {

                  reset(); //Read text file from beginning

                  //Set boolean markers
                  Constants.gameWon = false;
                  Constants.gameLost = false;

                  //Start using the text file data from the line after the level numer
                  while(!readLine().trim().equals(Integer.toString(levelNumber))) {}

                  //Load basic level stats
                  lvlRow = readLine();
                  String[] temp = lvlRow.split("/"); Constants.X_TILECOLS = Integer.parseInt(temp[0]);
                               Constants.Y_TILEROWS = Integer.parseInt(temp[1]);

                  Constants.tileSize = Math.min(Constants.FRAME_WIDTH_ADJUSTED/(Constants.X_TILECOLS + 2),
                                          Constants.FRAME_HEIGHT_ADJUSTED/(Constants.Y_TILEROWS + 2));

                  Constants.X_OFFSET = (Constants.FRAME_WIDTH_ADJUSTED -
Constants.tileSize*Constants.X_TILECOLS)/2.0;
                  Constants.Y_OFFSET = (Constants.FRAME_HEIGHT_ADJUSTED -
Constants.tileSize*Constants.Y_TILEROWS)/2.0;

                  lvlRow = readLine();
                  temp = lvlRow.split("/");

                  //Load the fruit batch list
                  for(int h = 0; h < temp.length; h++) {

                        if(temp[h].equals("w")) {
                              fruitIndicator.add(new Watermelon(0, 0, 75, 75/2, true));
                        } else if(temp[h].equals("p")) {
                              fruitIndicator.add(new Peach(0, 0, 60, 60, true));
                        } else if(temp[h].equals("k")) {
                              fruitIndicator.add(new Kiwi(0, 0, 50, 50, true));
                        } else if(temp[h].equals("l")) {
                              fruitIndicator.add(new Lemon(0, 0, 65, 58, true));
                        } else if(temp[h].equals("g")) {
```

```
                fruitIndicator.add(new Grapes(0, 0, 60, 60, true));
            } else if(temp[h].equals("o")) {
                fruitIndicator.add(new Orange(0, 0, 65, 58, true));
            } else if(temp[h].equals("a")) {
                fruitIndicator.add(new Avocado(0, 0, 45, 75, true));
            } else if(temp[h].equals("s")) {
                fruitIndicator.add(new Strawberry(0, 0, 50, 50, true));
            } else if(temp[h].equals("c")) {
                fruitIndicator.add(new Chili(0, 0, 35, 75, true));
            }
        }

        //Load the level ArrayLists
        while((lvlRow = readLine()) != null && !lvlRow.equals(Integer.toString(levelNumber + 1))) {

            numLines++; //Use line number markers

            if(lvlRow.trim().equals("endArray")) { //Start writing new 1D Array each time

                numLines = -1;
                numArrays++;

                if(numArrays >= 0) {
                    fruitList.add(new ArrayList<Fruit>());
                }

            } else {

                if(numArrays == -2) { //Load Tile ArrayList

                    tileList.add(new ArrayList<Tile>());

                    for(int i = 0; i < lvlRow.length(); i++) {

                        switch(lvlRow.charAt(i)) {

                        case 'n':
                            tileList.get(numLines).add(new TransparentTile(i, numLines));
                            break;
                        case 'i':
                            tileList.get(numLines).add(new IceTile(i, numLines));
                            break;
                        case '!':
                            Characters tempChr = new Characters("/UFinal/img/Vainilla.png",
i, numLines);

                            playerList.add(tempChr);
                            tileList.get(numLines).add(tempChr);
                            break;
                        }
                    }

                } else if (numArrays == -1) { //Load Monster ArrayList

                    for(int i = 0; i < lvlRow.length(); i++) {

                        if(lvlRow.charAt(i) == 'c') {
                            monsterList.add(new Cow(i, numLines));
                        } else if (lvlRow.charAt(i) == 'e') {
                            monsterList.add(new Egg(i, numLines));
                        } else if (lvlRow.charAt(i) == 'b') {
                            monsterList.add(new BlueCow(i, numLines));
                        } else if (lvlRow.charAt(i) == 'o') {
                            monsterList.add(new OrangeSquid(i, numLines));
                        } else if (lvlRow.charAt(i) == 't') {
                            monsterList.add(new Troll(i, numLines));
                        } else if (lvlRow.charAt(i) == 'y') {
                            monsterList.add(new YellowCow(i, numLines));
                        }
                    }

                } else { //Load Fruit ArrayList(s)

                    for(int i = 0; i < lvlRow.length(); i++) {

                        if(lvlRow.charAt(i) == 'w') {
                            fruitList.get(numArrays).add(new Watermelon(i, numLines, true));
                        } else if(lvlRow.charAt(i) == 'p') {
                            fruitList.get(numArrays).add(new Peach(i, numLines, true));
```

```
                                                } else if(lvlRow.charAt(i) == 'k') {
                                                        fruitList.get(numArrays).add(new Kiwi(i, numLines, true));
                                                } else if(lvlRow.charAt(i) == 'l') {
                                                        fruitList.get(numArrays).add(new Lemon(i, numLines, true));
                                                } else if(lvlRow.charAt(i) == 'g') {
                                                        fruitList.get(numArrays).add(new Grapes(i, numLines, true));
                                                } else if(lvlRow.charAt(i) == 'o') {
                                                        fruitList.get(numArrays).add(new Orange(i, numLines, true));
                                                } else if(lvlRow.charAt(i) == 'a') {
                                                        fruitList.get(numArrays).add(new Avocado(i, numLines, true));
                                                } else if(lvlRow.charAt(i) == 's') {
                                                        fruitList.get(numArrays).add(new Strawberry(i, numLines, true));
                                                } else if(lvlRow.charAt(i) == 'c') {
                                                        fruitList.get(numArrays).add(new Chili(i, numLines, true));
                                                }
                                        }
                                }
                        }
                }

                drawLevel();

        } catch (IOException e) {
                e.printStackTrace();
        }
}

/**
 * Draws the level from loaded ArrayLists
 * pre: none
 * post: level is drawn
 */
public void drawLevel() {

        //clear the window and add necessary components
        gamePane.removeAll();
        gamePane.add(sfx, new Integer(-1));
        gamePane.add(music, new Integer(-1));
```

```
		back.setActionCommand("tolevelselect"); //back button
		restart.setActionCommand(Integer.toString(levelNumber)); //restart button

		gamePane.add(back, new Integer(-1));
		gamePane.add(restart, new Integer(-1));

		ImageIcon imgIcon = new ImageIcon(getClass().getResource( //background
	       "/UFinal/img/bg" + levelNumber + ".gif"));
		bg.setIcon(new ImageIcon(imgIcon.getImage().getScaledInstance(Constants.FRAME_WIDTH,
Constants.FRAME_HEIGHT, Image.SCALE_DEFAULT)));
		bg.setBounds(-10, -10, bg.getPreferredSize().width, bg.getPreferredSize().height);
		gamePane.add(bg, new Integer(-2));

		for(int h = 0; h < fruitIndicator.size(); h++) { //fruit batches indicator

			fruitIndicator.get(h).setCenterX(50);
			fruitIndicator.get(h).setCenterY(90*(h+1) + 45);
			gamePane.add(fruitIndicator.get(h), 1);
		}

		JLabel timerLabel = new JLabel(new ImageIcon(getClass().getResource( //timer
			"/UFinal/img/timer.png")));
		timerLabel.setLocation(10, 10);
		timerLabel.setBounds(10, 10,
			timerLabel.getPreferredSize().width, timerLabel.getPreferredSize().height);
		gamePane.add(timerLabel, new Integer(-1));

		timer = new SecTimer(180 + 15*levelNumber, 25, 15);
		gamePane.add(timer, new Integer(0));

		for(int i = 0; i < Constants.Y_TILEROWS; i++) { //Loads the Tile objects and Characters
			for(int j = 0; j < Constants.X_TILECOLS; j++) {

				if(tileList.get(i).get(j) instanceof IceTile) {
					gamePane.add(tileList.get(i).get(j), new Integer(i*10));
				} else if (tileList.get(i).get(j) instanceof Characters) {
					gamePane.add(tileList.get(i).get(j), new Integer(i*10));
				}
			}
		}

		for(int k = 0; k < monsterList.size(); k++) { //Loads the monsters
			gamePane.add(monsterList.get(k), new Integer(monsterList.get(k).getTileY()*10 + 2));
		}

		drawFruit(); //draws the first batch of fruit
	}

	/**
	 * Draws the fruit from loaded ArrayLists
	 * pre: none
	 * post: fruit is drawn
	 */
	public void drawFruit() {

		int tileY;

		//changes the fruit batches indicator with each new fruit after the first
		if(Constants.fruitLvl > 0) {
			setIconBW(fruitIndicator.get(Constants.fruitLvl - 1));
		}

		//calls gameWon() if all fruits have been collected
		if(Constants.fruitLvl >= fruitList.size()) {
			gameWon();
			return;
		}

		//draws fruit from the fruit array
		if(!fruitList.isEmpty()) {

			for(int i = 0; i < fruitList.get(Constants.fruitLvl).size(); i++) {

				tileY = fruitList.get(Constants.fruitLvl).get(i).getTileY();
				gamePane.add(fruitList.get(Constants.fruitLvl).get(i), new Integer(tileY*10 + 1));
			}

			Constants.numFruit = fruitList.get(Constants.fruitLvl).size();
```

```
        }

        Constants.levelLoaded = true; //Changes boolean flag
    }

    /**
     * Draws the winning graphics
     * pre: none
     * post: winning graphics drawn
     */
    public void gameWon() {

        //Changes boolean flags
        Constants.gameWon = true;
        Constants.gameLost = false;

        //Create the winning JLabel and add it to the pane
        JLabel winLabel = new JLabel(new ImageIcon(getClass().getResource(
                "/UFinal/img/won.png")));
        winLabel.setLocation(310, 125);
        winLabel.setBounds(310, 125,
        winLabel.getPreferredSize().width, winLabel.getPreferredSize().height);

        //Loads the "next" button if the last level has not been reached
        if(levelNumber < 11) {

            JButton next = new JButton(new ImageIcon(getClass().getResource(
                    "/UFinal/img/next.png")));
            next.setLocation(290, 530);
                next.setBounds(290, 530,
                        next.getPreferredSize().width, next.getPreferredSize().height);
                next.setActionCommand("nextlevel");
                next.addActionListener(this);
                next.setBorder(BorderFactory.createEmptyBorder());
                next.setContentAreaFilled(false);
                gamePane.add(next, new Integer(400));
        }

        gamePane.add(winLabel, new Integer(400));
```

```
		//Plays the win sound if SFX have been enabled
		if(Constants.sfxEnabled) {
			wonSound.setFramePosition(0);
			wonSound.loop(0);
		}
	}

	/**
	 * Draws the losing graphics
	 * pre: none
	 * post: losing graphics drawn
	 */
	public void gameLost() {

		if(!Constants.gameWon) { //don't trigger a loss if the player has already won and a monster bumps into
them
			//Set boolean flags
			Constants.gameLost = true;
			Constants.gameWon = false;

			//make and draw the losing graphics
			JLabel loseLabel = new JLabel(new ImageIcon(getClass().getResource(
					"/UFinal/img/lost.png")));
			loseLabel.setLocation(310, 125);
			loseLabel.setBounds(310, 125,
			loseLabel.getPreferredSize().width, loseLabel.getPreferredSize().height);

			gamePane.add(loseLabel, new Integer(400));

			//Play the losing SFX if SFX are enabled
			if(Constants.sfxEnabled) {
				lostSound.setFramePosition(0);
				lostSound.loop(0);
			}
		}
	}

	/**
 * Turns a graphic into grayscale
 * pre: none
 * post: object's icon becomes grayscale
 */
public void setIconBW(Fruit fruit) {

	try {

		//Make new buffered image and apply a black and white filter
		BufferedImage bfImg = ImageIO.read(getClass().getResource(fruit.getImgURL()));

		ImageFilter imgFilter = new GrayFilter(true, 50);
		ImageProducer imgProducer = new FilteredImageSource(bfImg.getSource(), imgFilter);
		Image img = Toolkit.getDefaultToolkit().createImage(imgProducer);

		fruit.setIcon(new ImageIcon(img)); //change the imageicon

	} catch (IOException e) {
		e.printStackTrace();
	}
}

/**
 * Returns fruitList
 * pre: none
 * post: fruitList returned
 */
public ArrayList<ArrayList<Fruit>> getFruitArray() {
	return fruitList;
}

/**
 * Returns gamePane
 * pre: none
 * post: gamePane returned
 */
public JLayeredPane getPane() {
	return gamePane;
}
```

```
/**
 * Returns tileList
 * pre: none
 * post: tileList returned
 */
public ArrayList<ArrayList<Tile>> getTileList() {
	return tileList;
}

/**
 * Returns playerList
 * pre: none
 * post: playerList returned
 */
public ArrayList<Characters> getPlayerList() {
	return playerList;
}

/**
 * Returns monsterList
 * pre: none
 * post: monsterList returned
 */
public ArrayList<Tile> getMonsterList() {
	return monsterList;
}

/**
 * Returns a specific element within the playerList
 * pre: index is nonnegative
 * post: playerList.get(index) returned
 */
public Characters getPlayer(int index) {
	return playerList.get(index);
}

/**
 * Returns the timer
 * pre: none
 * post: timer returned
 */
public SecTimer getTimer() {
	return timer;
}

/**
 * Handles button click events
 * pre: none
 * post: button click events handled
 */
@Override
public void actionPerformed(ActionEvent e) {

	String cmd = e.getActionCommand();

	//Play button SFX if SFX is enabled
	if(Constants.sfxEnabled) {
		bop.setFramePosition(0);
		bop.loop(0);
	}

	//If the command is a number, load a level corresponding to the number
	if(Character.isDigit(cmd.charAt(0))) {

		Constants.levelLoaded = false; //Set boolean flag
		tileList.clear(); //Empty the pane
		fruitList.clear();
		playerList.clear();
		monsterList.clear();
		fruitIndicator.clear();

		if(Constants.musicEnabled) { //Start playing level music
			nonLvlMusic.stop();
			lvlMusic.loop(Clip.LOOP_CONTINUOUSLY);
		}

		readFile(Integer.parseInt(cmd)); //read file
```

```
            } else {

                //Takes the user to the level select screen from a level
                if(cmd.equals("tolevelselect")) {

                    Constants.levelLoaded = false;

                    tileList.clear();
                    fruitList.clear();
                    playerList.clear();
                    monsterList.clear();
                    fruitIndicator.clear();

                    if(Constants.musicEnabled) { //Play non-level music
                        nonLvlMusic.loop(Clip.LOOP_CONTINUOUSLY);
                        lvlMusic.stop();
                    }

                    levelSelect();
                } else if (cmd.equals("tohowto")) { //Take the user to the how to screen
                    howTo();
                } else if (cmd.equals("tocredits")) { //Take the user to a credits screen
                    credits();
                } else if (cmd.equals("togglesfx")) { //Toggles SFX being enabled and changes button icon
accordingly

                    if(Constants.sfxEnabled) {

                        Constants.sfxEnabled = false;
                        sfx.setIcon(new ImageIcon(getClass().getResource(
                            "/UFinal/img/nsfx.png")));
                    } else {

                        Constants.sfxEnabled = true;
                        sfx.setIcon(new ImageIcon(getClass().getResource(
                            "/UFinal/img/sfx.png")));
                    }
```

```
		} else if (cmd.equals("togglemusic")) { //Toggles Music being enabled and changes button icon 
accordingly

			if(Constants.musicEnabled) {

				Constants.musicEnabled = false;
				music.setIcon(new ImageIcon(getClass().getResource(
					"/UFinal/img/nmusic.png")));
				nonLvlMusic.stop();
				lvlMusic.stop();

			} else {

				Constants.musicEnabled = true;
				music.setIcon(new ImageIcon(getClass().getResource(
					"/UFinal/img/music.png")));
				if(!Constants.levelLoaded) {
					nonLvlMusic.loop(Clip.LOOP_CONTINUOUSLY);
				} else {
					lvlMusic.loop(Clip.LOOP_CONTINUOUSLY);
				}

			}
		} else if (cmd.equals("totitle")) { //Take the user back to the title screen
			titleScreen();
		} else if (cmd.equals("secret")) {
			//If the code entered is a valid secret code, take the user to the secret 11th level
			String code = userInput.getText();
			userInput.setText("");

			for(int i = 0; i < secretCodes.length; i++) {

				if(code.equals(secretCodes[i])) { //search through array for a code match

					if(Constants.musicEnabled) { //play level music
						nonLvlMusic.stop();
						lvlMusic.loop(Clip.LOOP_CONTINUOUSLY);
					}
```

```
                        readFile(11); //load level
                    }
                }

            } else if (cmd.equals("nextlevel")) { //Takes the user to the next level

                levelNumber++; //Changes variables and empties the window
                Constants.levelLoaded = false;

                tileList.clear();
                fruitList.clear();
                playerList.clear();
                monsterList.clear();
                fruitIndicator.clear();

                readFile(levelNumber);

            } else if (cmd.equals("pwdtotitle")) { //Takes the user from the initial password input to the
title screen

                if(pwdInput.getText().equals("100%")) {
                    titleScreen();
                }
            }
        }
    }
}
```