```
private void consumeResources() {
    if (gameOver || population <= 1) {
        handleGameOver("Colony population collapse!");
        return;
    }
    int consumedAmount = population;

    food -= consumedAmount;
    resourceAnimators.get("Food").showAnimation("-" + consumedAmount, Color.ORANGE.dark
    air -= consumedAmount;
    resourceAnimators.get("Air").showAnimation("-" + consumedAmount, Color.ORANGE.darke
    water -= consumedAmount;
    resourceAnimators.get("Water").showAnimation("-" + consumedAmount, Color.ORANGE.dar
```

This method is called every five seconds for the game to automatically track the population's resource consumption. The first loop is considering whether or not the game is over by checking the gameOver state and whether the population is equal to lower than 1. If this is the case, it calls another method to end the game. Otherwise, it reduces each resource variable by the consumedAmount, which is the population and tells the resourceAnimators which are the little displays that show how much things have grown.

```
boolean foodIncreasedSufficiently = food > foodAtLastGrowthCheck*0.8 && food >= population * 2;
boolean airIncreasedSufficiently = air > airAtLastGrowthCheck*0.8 && air >= population * 2;
boolean waterIncreasedSufficiently = water > waterAtLastGrowthCheck*0.8 && water >= population * 2;

if (food > 0 && air > 0 && water > 0 &&
        foodIncreasedSufficiently && airIncreasedSufficiently && waterIncreasedSufficiently) {
    population += Math.max(population/2, 1);
    resourceAnimators.get("Population").showAnimation("+" + Math.max(population/2, 1), new Color(0, 12
    statusLabel.setText("Population grew by " + Math.max(population/2, 1) + "!");
    statusLabel.setForeground(new Color(0, 100, 0));

} else {
    String reason = "Poor conditions leading to famine!";
    if (!(food > 0 && air > 0 && water > 0)) reason = "Essential resources are zero.";
    else if (!foodIncreasedSufficiently) reason += " Food levels not sufficiently increased or too low
    else if (!airIncreasedSufficiently) reason += " Air levels not sufficiently increased or too low.'
    else if (!waterIncreasedSufficiently) reason += " Water levels not sufficiently increased or too l
    else reason += " Resource levels may not be increasing enough or are too low relative to populatio
    statusLabel.setText(reason);
    statusLabel.setForeground(Color.ORANGE.darker());
}
```

This is an if statement in the growPopulation() method. It first creates three conditions that check whether or not each resource has decreased by less than 20% since the last checkpoint (5 seconds before) and whether each resource can last the next 2 rounds of consumption. If this is the case, there is another if statement that checks whether each resource is above 0 and then the population is increased by 50% or 1 whichever is bigger. The resourceAnimator next to the population displays the increase and the status label is updated as well with the notification.

```
    private void handleGameOver(String message) {
        if (gameOver) return;
        gameOver = true;

        statusLabel.setText("GAME OVER! " + message);
        statusLabel.setForeground(Color.RED);
        statusLabel.setFont(new Font("Arial", Font.BOLD, 14));

        if (consumptionTimer != null) consumptionTimer.stop();
        if (growthTimer != null) growthTimer.stop();
        if (gameMonthTimer != null) gameMonthTimer.stop();

        clickButton.setEnabled(false);

        if (foodPlant != null) foodPlant.setInactive();
        if (airPlant != null) airPlant.setInactive();
        if (waterPlant != null) waterPlant.setInactive();
        if (energyPlant != null) energyPlant.setInactive();

        updateLabels();
        updatePlantUIsActive();
    }
```

This is a method that is called when the game is over, when the population is equal to or below zero. The status label is changed to a red label that says GAME OVER and a message in red bold. All the timers that are checking for population growth, resource consumption and dates are stopped. All the resource plants are stopped and the labels are updated to display everything as disabled.

```
class ResourceAnimator {
    private JLabel animLabel;
    private Timer animTimer;
    private final int ANIM_DURATION = 1500;

    public ResourceAnimator(JLabel label) {
        this.animLabel = label;
    }

    public void showAnimation(String text, Color color) {
        if (animTimer != null && animTimer.isRunning()) {
            animTimer.stop();
        }
        animLabel.setText(text);
        animLabel.setForeground(color);
        animLabel.setFont(new Font("Arial", Font.BOLD, 14));

        animTimer = new Timer(ANIM_DURATION, e -> {
            animLabel.setText("");
            ((Timer)e.getSource()).stop();
        });
        animTimer.setRepeats(false);
        animTimer.start();
    }
    public void showAnimation(String text) {
        showAnimation(text, new Color(0, 128, 0));
    }
}
```

This is a class that is the a label that will next to the population count and the resource counts, and change when there is a change like +1 for example. The ANIM_DURATION variable sets the amount of time that the label will be up, right now it is set to 1.5 seconds. The showAnimation method sets the label text and the color, and sets the animation duration. This is an example of polymorphism as well, if the method only receives a text parameter and not a

color then it defaults to green.

```
private void startGeneration() {
    if (level > 0 && !game.gameOver) {
        int actualConfigIndex = level - 1;
        int interval = productionIntervalsMillis[actualConfigIndex];
        if (generationTimer == null) {
            generationTimer = new Timer(interval, e -> generateResource());
            generationTimer.setInitialDelay(interval); // Start producing after first interval
        } else {
            generationTimer.setDelay(interval);
            generationTimer.setInitialDelay(interval);
        }
        if (!generationTimer.isRunning()) {
            generationTimer.start();
        }
    } else if (generationTimer != null && generationTimer.isRunning()) {
        generationTimer.stop();
    }
}
```

This is a method in my ResourcePlant class. It is called when a resource plant is bought or upgraded. The first condition makes sure that the plant was bought and that it is not game over. The next line decreases the index by 1 to account for the 0-indexing in java. The interval variable is the rate at which the plant is going to produce the resources. Those rates are set in an array and are different for each level. The next condition checks whether a timer has been created to start generating resources, if not one is created and either way the interval is set for the timer based on the rate corresponding to the level.

```
private void generateResource() {
    if (game.gameOver) {
        stopGeneration();
        return;
    }

    productionAmountCurrentCycle = game.population;
    if (resourceType.equals("Energy")) {
         productionAmountCurrentCycle = Math.max(game.population / 5, 1); // Energy plant scaling
    }


    String animText = "+" + productionAmountCurrentCycle;
    Color animColor = new Color(0, 150, 0);
    long energyCostForProduction = 0;
    boolean canProduce = true;

    switch (resourceType) {
        case "Food":
            energyCostForProduction = (long) (Space.FOOD_COST_PER_UNIT_RAW * productionAmountCurrentCycle * 0.5);
            if (game.energy >= energyCostForProduction) {
                game.energy -= energyCostForProduction;
                game.food += productionAmountCurrentCycle;
                game.resourceAnimators.get("Food").showAnimation(animText, animColor);
            } else { canProduce = false; }
            break;
        case "Air":
            energyCostForProduction = (long) (Space.AIR_COST_PER_UNIT_RAW * productionAmountCurrentCycle * 0.5);
            if (game.energy >= energyCostForProduction) {
                game.energy -= energyCostForProduction;
                game.air += productionAmountCurrentCycle;
                game.resourceAnimators.get("Air").showAnimation(animText, animColor);
            } else { canProduce = false; }
            break;
        case "Water":
            energyCostForProduction = (long) (Space.WATER_COST_PER_UNIT_RAW * productionAmountCurrentCycle * 0.5);
            if (game.energy >= energyCostForProduction) {
                game.energy -= energyCostForProduction;
                game.water += productionAmountCurrentCycle;
                game.resourceAnimators.get("Water").showAnimation(animText, animColor);
            } else { canProduce = false; }
            break;
        case "Energy": // Energy plant produces energy, doesn't consume it for production here.
            game.energy += productionAmountCurrentCycle;
            game.resourceAnimators.get("Energy").showAnimation(animText, animColor);
            break;
    }

    if (!canProduce && !resourceType.equals("Energy")) {
         game.statusLabel.setText(resourceType + " Plant needs more energy to operate!");
         game.statusLabel.setForeground(Color.ORANGE.darker());
    }

    game.updateLabels(); // Update main resource labels
}
```

This is another method in the Resource Plant class called generateResource that will generate resources for the population. The first condition checks if the game is over and returns if it is. The productionAmountCurrentCycle variable is set to the population size, and a fifth of the population size for the energy plants. The next lines are variable definitions for the resourceAnimator's text and color (green), and then the energy that the production will cost which is defined later, and then finally a boolean variable that will be whether or not the colony has enough energy to produce in this cycle. Next is a switch that are the four cases for the four resources Air, Water, Food, and Energy. For the first three, it just checks whether the energy cost of the production is below the total energy in the colony and if so it subtracts from the energy and adds to the resource. Then it also displays the animation. For the energy case it doesn't need to check for cost, since there is no cost so it just directly adds. Finally, if it is found

that there is not enough energy to produce then a warning is displayed on the status label.

```
public void updateLabels() {
    energyLabel.setText("Energy: " + energy);
    populationLabel.setText("Population: " + population);
    foodLabel.setText("Food: " + food);
    airLabel.setText("Air: " + air);
    waterLabel.setText("Water: " + water);
}
```

This function is called throughout the program every time a change needs to be displayed in the UI. The four resource labels and the energy label are updated when the function is called.

```
private void updateGameDateAndGraph() {
    if (gameOver) {
        if (gameMonthTimer != null) gameMonthTimer.stop();
        return;
    }

    currentMonth++;
    if (currentMonth > 12) {
        currentMonth = 1;
        currentYear++;
    }

    String[] monthNames = {
        "January", "February", "March", "April", "May", "June",
        "July", "August", "September", "October", "November", "December"
    };

    String monthName = monthNames[currentMonth - 1];
    dateLabel.setText(String.format("Date: %s Year %04d", monthName, currentYear));

    populationHistory.add(population);
    if (populationHistory.size() > MAX_HISTORY_POINTS) {
        populationHistory.remove(0);
    }
    if (populationGraphPanel != null) {
        populationGraphPanel.repaint();
    }
}
```

This method is called to update the date every second with a timer (gameMonthTimer). The currentMonth is incremented by 1 and then if it is over 12 it is turned back to 1 and the currentYear is incremented. An array of the 12 months is declared and then the correct month is set based on the currentMonth variable. The date label is then set and formatted with the currentYear and currentMonth. Finally, the population history graph is updated with a data point with the current population and if the graph is full the oldest data point is removed.