yМинистерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЁТ №8**

**Дисциплина: Многоагентное моделирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_М. В. Сидоренко

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Миков

Краснодар

2024

**Описание задачи**

**Колониальная**

2150 год, Марс. Есть несколько колоний, управляемых агентами-администраторами. Каждая колония имеет следующие характеристики:

- уровень (максимум 10);

- баланс;

- затраты в цикл;

- доходы в цикл;

- опыт.

Максимальное время моделирования T. Цикл состоит из t\_iter итераций. Баланс обновляется в начале цикла (добавляется значение текущего дохода и вычитается значение текущих расходов). Изначально у каждой колонии одинаковый баланс B, но разное соотношение затрат и доходов (доход > затраты).

Когда обновляется баланс, также обновляется опыт колонии на разницу между предыдущим балансом и текущим (e = e + b\_current – b\_previous), опыт может уменьшаться, но не может стать отрицательным. Когда значение опыта достигает константного значения L, опыт обнуляется и уровень увеличивается (уровень не может уменьшаться).

Условием победы колонии считается достижение максимального уровня. Условием поражения и уничтожения колонии считается уход баланса в минус.

Раз в несколько циклов t\_a Земля проводит аукцион (первой или второй цены) артефактов, улучшающих состояние колонии (выигравшие колонии в аукционах не участвуют).

Артефакт может, например:

- увеличить уровень колонии;

- уменьшить затраты на обслуживание;

- увеличить доход;

- спасти колонию от разрушения (не более одного артефакта у каждой колонии одновременно).

Изменение может иметь как разовый эффект, так и постоянный, например для увеличения дохода может быть как единовременная выплата, так и выплаты на протяжении нескольких циклов.

Также раз в несколько циклов t\_e происходят события среды:

- пылевая буря (уменьшает доход колонии на g пунктов и увеличивает расход на j);

- «ренессанс» (эффект прямо противоположный пылевой буре).

Эффект события одноразовый.

**Задача**

Каждому разработчику выдаётся набор из 5 артефактов с различными эффектами. Эффекты являются параметризированными, то есть конкретные значения выбираются разработчиком самостоятельно, равно как и стоимость каждого артефакта.

Разработать алгоритм участия агента в аукционе. В данном случае понятно, что все артефакты только улучшают текущее состояние колонии, поэтому есть смысл участвовать в каждом аукционе. Однако, агент должен понимать, что на следующую ставку ему может не хватить баланса, или что покупка может лишь усугубить состояние (например, снижение расхода на h пунктов не выровняет соотношение доход/расход, но баланс станет меньше, что может привести к поражению раньше).

Построить график распределения вероятностей времени жизни колонии (до поражения) для различных сочетаний входных параметров.

Построить график зависимостей вероятности побед и поражений колоний в зависимости от входных параметров (провести n экспериментов, к концу моделирования рассчитать количество выигравших и проигравших колоний).

**Описание решения**

Этот код представляет собой модель симуляции колоний на Марсе в 2150 году, управляемых агентами-администраторами, каждый из которых контролирует одну колонию. Целью является моделирование развития колоний в условиях постоянных экономических изменений, аукционов артефактов и случайных событий. Каждая колония имеет набор характеристик, таких как уровень, баланс, доходы, расходы и опыт. Код решает задачу моделирования этих колоний, учитывая условия победы и поражения, а также возможности для улучшения состояния колонии через аукционы и артефакты.

Модель игры состоит из нескольких ключевых компонентов:

1. Colony.java

Класс Colony представляет модель колонии, которая может развиваться, увеличивать свой уровень, зарабатывать или терять баланс и даже погибать. Колония имеет несколько атрибутов, включая имя, уровень, баланс, доходы и расходы, а также опыт и состояние колонии.

- Конструктор: Инициализирует основные параметры колонии. Колонии присваивается начальный баланс, доходы и расходы. Уровень и опыт начинаются с минимальных значений. Колония может быть активной или выбывшей, а также может стать победителем, достигнув максимального уровня.

- Метод checkLevelUp: Проверяет, достигла ли колония нового уровня, сравнивая её опыт с пороговым значением. Если уровень достигает максимального значения, колония становится победителем, и её симуляция завершается.

- Метод updateBalance: Обновляет баланс колонии с учётом её доходов и расходов. При изменении баланса начисляется опыт. Если баланс становится отрицательным, колония выбывает из симуляции.

- Метод applyEffects: Применяет все активные эффекты, изменяющие параметры колонии. Эффекты имеют длительность, и по истечении они удаляются и откатыArtifact.javaifact.java\*\*

Класс Artifact моделирует артефакт, который оказывает влияние на колонию в ходе симуляции. Артефакт имеет имя и список эффектов, которые могут улучшать доходы, уменьшать расходы или увеличивать уровень колониКонструктор:нструктор: Инициализирует имя артефакта и список эффектов.

- Метод applyArtifact: Применяет артефакт к колонии, добавляя все его эффекты в список эффектов колонии и записывая это в журналConstants.javaConstants.java

Этот файл содержит основные константы и начальные настройки для симуляции. Здесь определены максимальный уровень колонии, начальный баланс, порог опыта для повышения уровня, количество колоний в симуляции, продолжительность симуляции, интервалы для аукциона и событий среды, а также максимальный размер ставки на аукционе. Лог всех событий симуляции сохраняется в указEffect.java Effect.java

Класс Effect описывает эффект, который может быть применён к колонии, влияя на её параметры. В конструкторе задаются тип эффекта, его величина, продолжительность действия и название эфМетод apply: Метод apply: Применяет эффект к колонии, изменяя её параметры в зависимости от типМетод rollback:етод rollback: Снимает эффект с колонии, восстанавливая её параметры до первоначальныEnvironment.javavironment.java

Этот класс содержит функции среды, такие как пыльная буря и ренессанс, которые могут воздействовать на колонии в ходе симуМетод dustStorm:тод dustStorm: Уменьшает доход колонии и увеличивает еМетод renaissance:д renaissance: Увеличивает доход колонии и уменьшает Graphics.java Graphics.java

Класс Graphics содержит методы для визуализации различных аспектов симуляции с использованием библиотеки JFreeОсновные методы:новные методы: Включают построение графиков среднего баланса по уровням, распределения победных ставок, соотношения выживших и выбывших колоний, вероятности победы на аукционах, распределения уровней и других ключевых показателей.

7. Simulation.java

Этот класс управляет процессом симуляции, включая циклы обновления состояний колоний и взаимодействие с аукционами и событиями среды.

- Метод runCycle: Основной цикл симуляции, где обновляются параметры колоний, применяются эффекты и проверяется их состояние.

- Метод runAuction: проводит аукционы, на которых колонии делают ставки на артефакты, которые могут улучшить их состоянApp.javap.java\*\*

Основной исполнимый файл симуляции, который управляет процессом моделирования жизни колоний на Марсе. Этот файл инициирует процесс симуляции, управляет циклическим обновлением состояний колоний и взаимодействием с различными игровыми событиями, и аукционами.

В ходе симуляции собираются данные о прогрессе уровней колоний, изменениях баланса, выживаемости колоний, а также результатах аукционов. Эти данные используются для построения графиков, визуализирующих различные аспекты симуляции. В конце симуляции выводится таблица с итогами для каждой колонии, включая её уровень, баланс, статус и другие характеристики.

**Код программы**

public class Artifact {  
 private String name;  
 private List<Effect> effects;  
  
 public Artifact(String name, List<Effect> effects) {  
 this.name = name;  
 this.effects = effects;  
 }  
  
 public void applyArtifact(Colony colony, List<String> log) {  
 for (Effect effect : effects) {  
 colony.getEffects().add(effect);  
 }  
 log.add(colony.getName() + ": Артефакт '" + name + "' применён.");  
 }  
  
 public String getName() {  
 return name;  
 }  
}

public class Colony {  
 private String name;  
 private int level;  
 private double balance;  
 private double income;  
 private double expenses;  
 private double experience;  
 private List<Effect> effects;  
 private boolean alive;  
 private boolean isWinner;  
 private int roundsPlayed;  
 private Integer levelUpIteration;  
  
 public Colony(String name, double balance, double income, double expenses) {  
 this.name = name;  
 this.level = 1;  
 this.balance = balance;  
 this.income = income;  
 this.expenses = expenses;  
 this.experience = 0;  
 this.effects = new ArrayList<>();  
 this.alive = true;  
 this.isWinner = false;  
 this.roundsPlayed = 0;  
 this.levelUpIteration = null;  
 }  
  
 public void checkLevelUp(List<String> log, int cycleNumber) {  
 if (!alive) return;  
  
 if (level < Constants.*MAX\_LEVEL* && experience >= Constants.*EXPERIENCE\_THRESHOLD*) {  
 experience = 0;  
 level++;  
 log.add(name + ": Повышение уровня! Новый уровень: " + level + ".");  
 }  
  
 if (level == Constants.*MAX\_LEVEL*) {  
 isWinner = true;  
 levelUpIteration = cycleNumber;  
 log.add(name + ": Достигнут максимальный уровень и назначена как победитель.");  
 alive = false;  
 }  
 }  
  
 public void updateBalance(List<String> log) {  
 if (!alive) return;  
  
 double previousBalance = balance;  
 balance += income - expenses;  
 experience += Math.*max*(0, balance - previousBalance) / 100;  
 experience += income / 10;  
  
 if (balance < 0) {  
 alive = false;  
 log.add(name + ": Баланс отрицателен, колония выбывает!");  
 }  
 }  
  
 public void applyEffects(List<String> log) {  
 Iterator<Effect> iterator = effects.iterator();  
 while (iterator.hasNext()) {  
 Effect effect = iterator.next();  
 effect.apply(this, log);  
 effect.durationDecrement();  
  
 if (effect.getDuration() <= 0 && !effect.getName().equals("Максимальный уровень")) {  
 effect.rollback(this, log);  
 log.add(name + ": Эффект '" + effect.getName() + "' истёк.");  
 iterator.remove();  
 }  
 }  
 }  
  
 public List<Effect> getEffects() {  
 return effects;  
 }  
  
 public void setEffects(List<Effect> effects) {  
 this.effects = effects;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getBalance() {  
 return balance;  
 }  
  
 public void setBalance(double balance) {  
 this.balance = balance;  
 }  
  
 public double getIncome() {  
 return income;  
 }  
  
 public void setIncome(double income) {  
 this.income = income;  
 }  
  
 public double getExpenses() {  
 return expenses;  
 }  
  
 public void setExpenses(double expenses) {  
 this.expenses = expenses;  
 }  
  
 public boolean isWinner() {  
 return isWinner;  
 }  
  
 public double getExperience() {  
 return experience;  
 }  
  
 public void setExperience(double experience) {  
 this.experience = experience;  
 }  
  
 public int getLevel() {  
 return level;  
 }  
  
 public boolean isAlive() {  
 return alive;  
 }  
  
 public int getRoundsPlayed() {  
 return roundsPlayed;  
 }  
  
 public void setRoundsPlayed(int roundsPlayed) {  
 this.roundsPlayed = roundsPlayed;  
 }  
  
 public void setLevel(int level) {  
 this.level = level;  
 }  
}

package org.example;  
  
import java.util.List;  
  
public class Effect {  
 private String effectType;  
 private double value;  
 private int duration;  
 private String name;  
  
 // Конструктор  
 public Effect(String effectType, double value, int duration, String name) {  
 this.effectType = effectType;  
 this.value = value;  
 this.duration = duration;  
 this.name = name;  
 }  
  
 // Применение эффекта  
 public void apply(Colony colony, List<String> log) {  
 switch (effectType) {  
 case "experience\_current":  
 double experienceIncrease = colony.getExperience() \* (value / 100);  
 colony.setExperience(colony.getExperience() + experienceIncrease);  
 log.add(colony.getName() + ": Опыт увеличен на " + value + "% от текущего опыта.");  
 break;  
 case "experience\_max":  
 double maxExperienceIncrease = Constants.*EXPERIENCE\_THRESHOLD* \* (value / 100);  
 colony.setExperience(colony.getExperience() + maxExperienceIncrease);  
 log.add(colony.getName() + ": Опыт увеличен на " + value + "% от максимального опыта уровня.");  
 break;  
 case "balance":  
 colony.setBalance(colony.getBalance() + value);  
 log.add(colony.getName() + ": Баланс увеличен на " + value + " единиц.");  
 break;  
 case "income\_from\_expenses":  
 double incomeIncrease = colony.getExpenses() \* (value / 100);  
 colony.setIncome(colony.getIncome() + incomeIncrease);  
 log.add(colony.getName() + ": Доход увеличен на " + value + "% от расходов.");  
 break;  
 case "income\_double":  
 colony.setIncome(colony.getIncome() \* 2);  
 log.add(colony.getName() + ": Доход удвоен.");  
 break;  
 case "expenses\_fixed":  
 colony.setExpenses(colony.getExpenses() - value);  
 colony.setExpenses(Math.*max*(1, colony.getExpenses())); // Убедитесь, что расходы не становятся отрицательными  
 log.add(colony.getName() + ": Расходы уменьшены на " + value + " единиц.");  
 break;  
 case "expenses\_percent":  
 double expenseReduction = colony.getExpenses() \* (value / 100);  
 colony.setExpenses(colony.getExpenses() - expenseReduction);  
 colony.setExpenses(Math.*max*(1, colony.getExpenses()));  
 log.add(colony.getName() + ": Расходы уменьшены на " + value + "%.");  
 break;  
 case "level\_up":  
 int newLevel = (int) (colony.getLevel() + value);  
 colony.setLevel(Math.*min*(Constants.*MAX\_LEVEL*, newLevel));  
 log.add(colony.getName() + ": Уровень увеличен на " + (int)value + " уровней.");  
 break;  
 case "income\_from\_balance":  
 double incomeFromBalance = colony.getBalance() \* (value / 100);  
 colony.setIncome(colony.getIncome() + incomeFromBalance);  
 log.add(colony.getName() + ": Доход увеличен на " + value + "% от баланса.");  
 break;  
 }  
 }  
  
 // Откат эффекта  
 public void rollback(Colony colony, List<String> log) {  
 // Логика отката эффектов, если применимо  
 switch (effectType) {  
 case "experience\_current":  
 case "experience\_max":  
 // Обычно опыт не откатывается  
 break;  
 case "balance":  
 colony.setBalance(colony.getBalance() - value);  
 log.add(colony.getName() + ": Эффект на баланс (" + value + " единиц) снят.");  
 break;  
 case "income\_from\_expenses":  
 double incomeDecrease = colony.getExpenses() \* (value / 100);  
 colony.setIncome(colony.getIncome() - incomeDecrease);  
 log.add(colony.getName() + ": Эффект на доход (" + value + "% от расходов) снят.");  
 break;  
 case "income\_double":  
 colony.setIncome(colony.getIncome() / 2);  
 log.add(colony.getName() + ": Эффект удвоения дохода снят.");  
 break;  
 case "expenses\_fixed":  
 colony.setExpenses(colony.getExpenses() + value);  
 log.add(colony.getName() + ": Эффект на расходы (" + value + " единиц) снят.");  
 break;  
 case "expenses\_percent":  
 double expenseIncrease = colony.getExpenses() \* (value / 100);  
 colony.setExpenses(colony.getExpenses() + expenseIncrease);  
 log.add(colony.getName() + ": Эффект на расходы (" + value + "%) снят.");  
 break;  
 case "level\_up":  
 // Уровень обычно не уменьшается, поэтому откат может быть не нужен  
 break;  
 case "income\_from\_balance":  
 double incomeDecreaseFromBalance = colony.getBalance() \* (value / 100);  
 colony.setIncome(colony.getIncome() - incomeDecreaseFromBalance);  
 log.add(colony.getName() + ": Эффект на доход (" + value + "% от баланса) снят.");  
 break;  
 }  
 }  
  
 public void durationDecrement() {  
 duration--;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getDuration() {  
 return duration;  
 }  
  
 public double getValue() {  
 return value;  
 }  
  
 public String getEffectType() {  
 return effectType;  
 }  
}

package org.example;  
  
public class Constants {  
 public static final int *MAX\_LEVEL* = 10;  
 public static final double *INITIAL\_BALANCE* = 1000.0;  
 public static final double *EXPERIENCE\_THRESHOLD* = 100.0;  
 public static final int *COLONY\_COUNT* = 10;  
 public static final int *SIMULATION\_TIME* = 250;  
 public static final int *ITERATIONS\_PER\_CYCLE* = 10;  
 public static final int *AUCTION\_INTERVAL* = 50;  
 public static final int *EVENT\_INTERVAL* = 30;  
 public static final double *MAX\_BET* = 0.75;  
 public static final String *LOG\_FILE* = "logs.txt";  
}

package org.example;  
  
import java.util.List;  
  
public class Environment {  
  
 public static void dustStorm(Colony colony, List<String> log, double incomeReduction, double expenseIncrease) {  
 colony.setIncome(colony.getIncome() - colony.getIncome() \* (incomeReduction / 100));  
 colony.setExpenses(colony.getExpenses() + colony.getExpenses() \* (expenseIncrease / 100));  
 log.add(colony.getName() + ": Пыльная буря! Доход уменьшен на " + incomeReduction +  
 "%, расходы увеличены на " + expenseIncrease + "%.");  
 }  
  
 public static void renaissance(Colony colony, List<String> log, double incomeIncrease, double expenseReduction) {  
 colony.setIncome(colony.getIncome() + colony.getIncome() \* (incomeIncrease / 100));  
 colony.setExpenses(Math.*max*(1, colony.getExpenses() - colony.getExpenses() \* (expenseReduction / 100)));  
 log.add(colony.getName() + ": Ренессанс! Доход увеличен на " + incomeIncrease +  
 "%, расходы уменьшены на " + expenseReduction + "%.");  
 }  
}

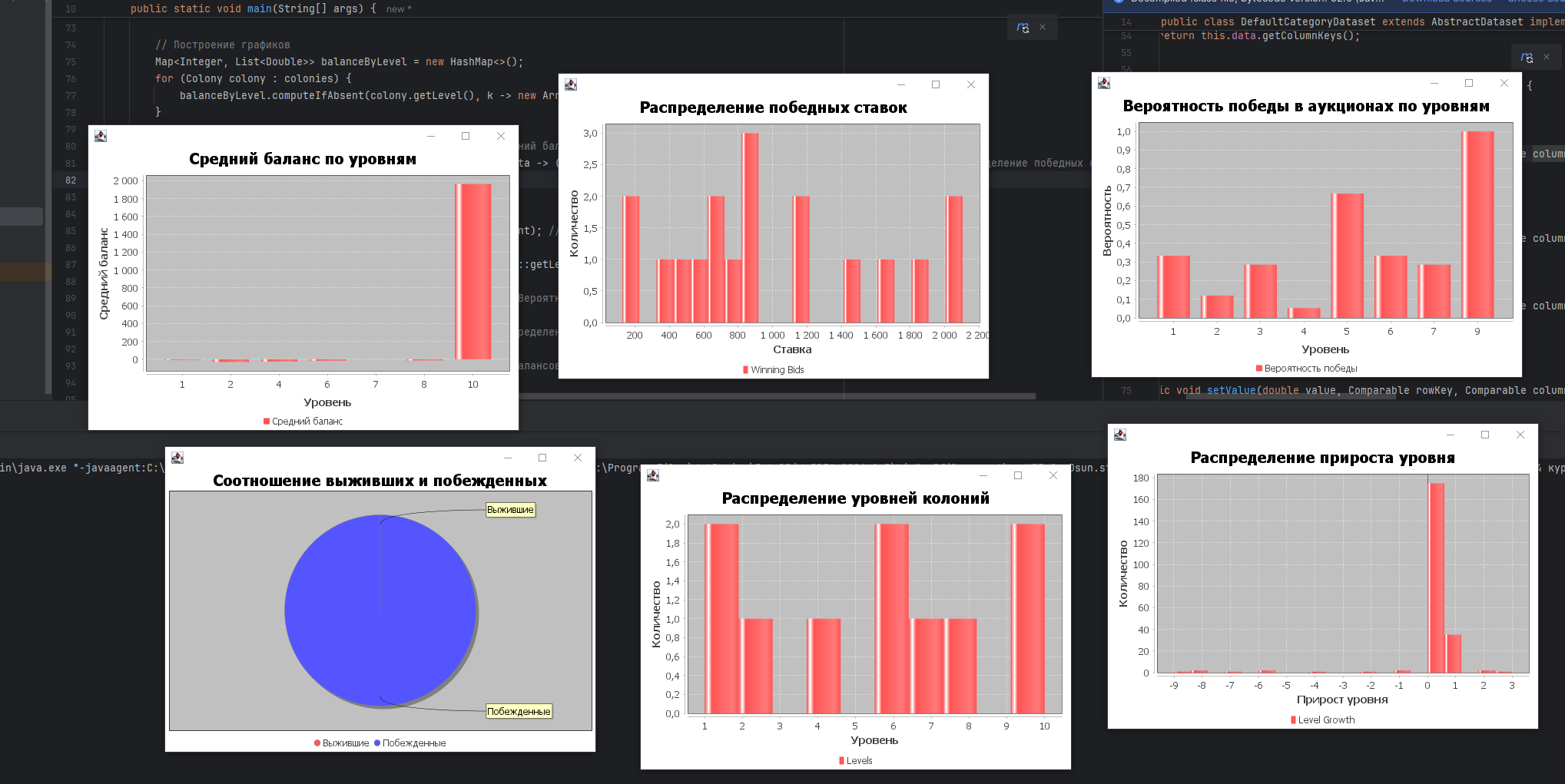
package org.example;  
  
import org.jfree.chart.ChartFactory;  
import org.jfree.chart.ChartPanel;  
import org.jfree.chart.JFreeChart;  
import org.jfree.chart.plot.PlotOrientation;  
import org.jfree.data.category.DefaultCategoryDataset;  
import org.jfree.data.general.DefaultPieDataset;  
import org.jfree.data.statistics.HistogramDataset;  
  
import javax.swing.\*;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class Graphics {  
  
 public static void plotAverageBalanceByLevel(Map<Integer, List<Double>> balanceByLevel) {  
 DefaultCategoryDataset dataset = new DefaultCategoryDataset();  
  
 for (Map.Entry<Integer, List<Double>> entry : balanceByLevel.entrySet()) {  
 int level = entry.getKey();  
 double avgBalance = entry.getValue().stream().mapToDouble(Double::doubleValue).average().orElse(0);  
 dataset.addValue(avgBalance, "Средний баланс" , Integer.*valueOf*(level));  
 }  
  
 JFreeChart barChart = ChartFactory.*createBarChart*(  
 "Средний баланс по уровням",  
 "Уровень",  
 "Средний баланс",  
 dataset,  
 PlotOrientation.*VERTICAL*,  
 true, true, false);  
  
 ChartPanel chartPanel = new ChartPanel(barChart);  
 chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));  
  
 JFrame frame = new JFrame();  
 frame.setContentPane(chartPanel);  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
 public static void plotAuctionWinningBids(List<Double> winningBids) {  
 HistogramDataset dataset = new HistogramDataset();  
 double[] bidsArray = winningBids.stream().mapToDouble(Double::doubleValue).toArray();  
 dataset.addSeries("Winning Bids", bidsArray, 20);  
  
 JFreeChart histogram = ChartFactory.*createHistogram*(  
 "Распределение победных ставок",  
 "Ставка",  
 "Количество",  
 dataset,  
 PlotOrientation.*VERTICAL*,  
 true, true, false);  
  
 ChartPanel chartPanel = new ChartPanel(histogram);  
 chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));  
  
 JFrame frame = new JFrame();  
 frame.setContentPane(chartPanel);  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
 public static void plotSurvivalVsDefeatRatio(int survivalCount, int defeatCount) {  
 DefaultPieDataset dataset = new DefaultPieDataset();  
 dataset.setValue("Выжившие", survivalCount);  
 dataset.setValue("Побежденные", defeatCount);  
  
 JFreeChart pieChart = ChartFactory.*createPieChart*(  
 "Соотношение выживших и побежденных",  
 dataset,  
 true, true, false);  
  
 ChartPanel chartPanel = new ChartPanel(pieChart);  
 chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));  
  
 JFrame frame = new JFrame();  
 frame.setContentPane(chartPanel);  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
 public static void plotLevelDistribution(List<Integer> levels) {  
 HistogramDataset dataset = new HistogramDataset();  
 double[] levelsArray = levels.stream().mapToDouble(Integer::doubleValue).toArray();  
 dataset.addSeries("Levels", levelsArray, Constants.*MAX\_LEVEL*);  
  
 JFreeChart histogram = ChartFactory.*createHistogram*(  
 "Распределение уровней колоний",  
 "Уровень",  
 "Количество",  
 dataset,  
 PlotOrientation.*VERTICAL*,  
 true, true, false);  
  
 ChartPanel chartPanel = new ChartPanel(histogram);  
 chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));  
  
 JFrame frame = new JFrame();  
 frame.setContentPane(chartPanel);  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
 public static void plotAuctionWinProbabilityByLevel(List<Map<String, Object>> auctionData) {  
 Map<Integer, Integer> winsByLevel = new HashMap<>();  
 Map<Integer, Integer> participationByLevel = new HashMap<>();  
  
 for (Map<String, Object> data : auctionData) {  
 int winnerLevel = (int) data.get("winner\_level");  
 winsByLevel.put(winnerLevel, winsByLevel.getOrDefault(winnerLevel, 0) + 1);  
  
 List<Integer> participants = (List<Integer>) data.get("participants");  
 for (int level : participants) {  
 participationByLevel.put(level, participationByLevel.getOrDefault(level, 0) + 1);  
 }  
 }  
  
 DefaultCategoryDataset dataset = new DefaultCategoryDataset();  
 for (Map.Entry<Integer, Integer> entry : participationByLevel.entrySet()) {  
 int level = entry.getKey();  
 int participation = entry.getValue();  
 if (participation > 0) {  
 double probability = winsByLevel.getOrDefault(level, 0) / (double) participation;  
 dataset.addValue(probability, "Вероятность победы", String.*valueOf*(level));  
 }  
 }  
  
 JFreeChart barChart = ChartFactory.*createBarChart*(  
 "Вероятность победы в аукционах по уровням",  
 "Уровень",  
 "Вероятность",  
 dataset,  
 PlotOrientation.*VERTICAL*,  
 true, true, false);  
  
 ChartPanel chartPanel = new ChartPanel(barChart);  
 chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));  
  
 JFrame frame = new JFrame();  
 frame.setContentPane(chartPanel);  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
 public static void plotLevelGrowthDistribution(List<Integer> levelsGrowth) {  
 HistogramDataset dataset = new HistogramDataset();  
 double[] growthArray = levelsGrowth.stream().mapToDouble(Integer::doubleValue).toArray();  
 dataset.addSeries("Level Growth", growthArray, 20);  
  
 JFreeChart histogram = ChartFactory.*createHistogram*(  
 "Распределение прироста уровня",  
 "Прирост уровня",  
 "Количество",  
 dataset,  
 PlotOrientation.*VERTICAL*,  
 true, true, false);  
  
 ChartPanel chartPanel = new ChartPanel(histogram);  
 chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));  
  
 JFrame frame = new JFrame();  
 frame.setContentPane(chartPanel);  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
  
 public static void plotBalanceChange(List<List<Double>> balancesByCycle) {  
 DefaultCategoryDataset dataset = new DefaultCategoryDataset();  
  
 for (int i = 0; i < balancesByCycle.size(); i++) {  
 List<Double> balances = balancesByCycle.get(i);  
 for (int j = 0; j < balances.size(); j++) {  
 dataset.addValue(balances.get(j), "Цикл " + (i + 1), String.*valueOf*(j + 1));  
 }  
 }  
  
 JFreeChart lineChart = ChartFactory.*createLineChart*(  
 "Изменение балансов колоний",  
 "Цикл",  
 "Баланс",  
 dataset,  
 PlotOrientation.*VERTICAL*,  
 true, true, false);  
  
 ChartPanel chartPanel = new ChartPanel(lineChart);  
 chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));  
  
 JFrame frame = new JFrame();  
 frame.setContentPane(chartPanel);  
 frame.pack();  
 frame.setVisible(true);  
 }  
  
}

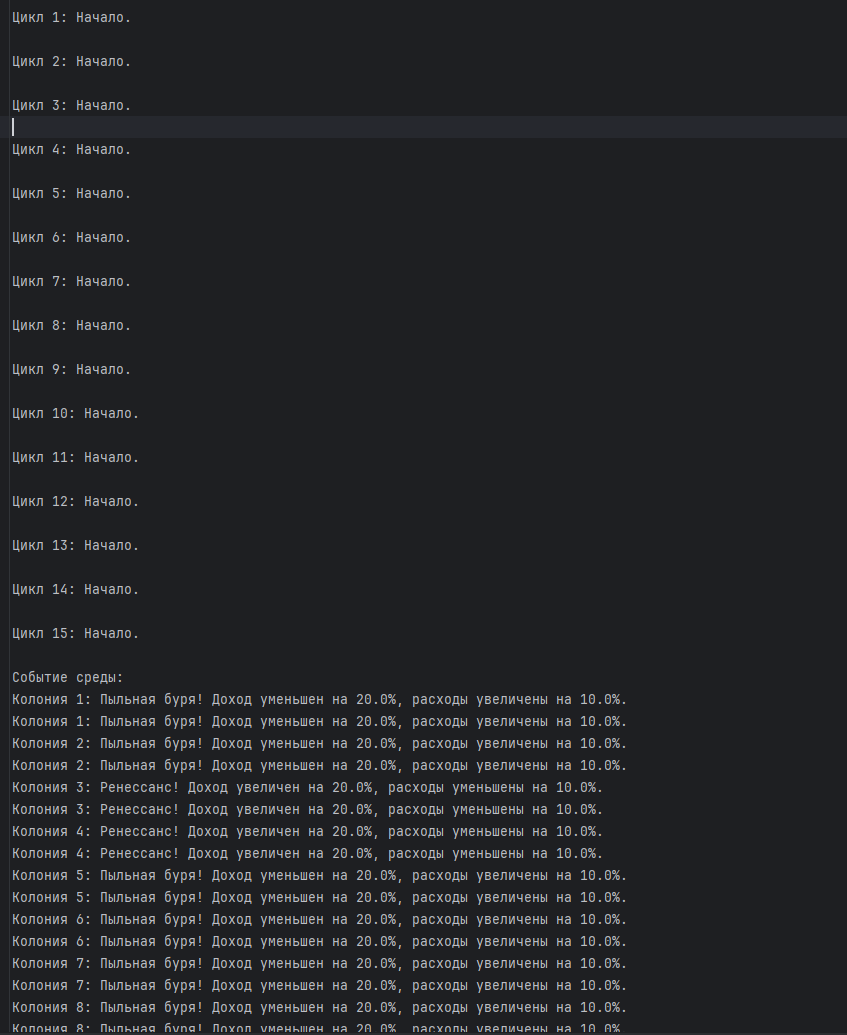
package org.example;  
  
import java.util.\*;  
  
public class Simulation {  
  
 private static List<Artifact> *artifactPool* = Arrays.*asList*(  
 // Артефакт 15  
 new Artifact("Артефакт 15", Arrays.*asList*(  
 new Effect("experience\_current", 10, 1, "Увеличение опыта от текущего опыта"),  
 new Effect("balance", 100, 1, "Увеличение баланса на 100 единиц"),  
 new Effect("expenses\_fixed", 50, 3, "Уменьшение текущих расходов на 50 единиц")  
 )),  
  
 // Артефакт 40  
 new Artifact("Артефакт 40", Arrays.*asList*(  
 new Effect("income\_from\_expenses", 10, 1, "Увеличение дохода на 10% от расходов"),  
 new Effect("experience\_max", 10, 3, "Увеличение опыта на 10% от максимального опыта"),  
 new Effect("balance", 10, 3, "Увеличение баланса на 10% от дохода")  
 )),  
  
 // Артефакт 45  
 new Artifact("Артефакт 45", Arrays.*asList*(  
 new Effect("experience\_current", 10, 0, "Увеличение опыта от текущего опыта (единоразово)")  
 )),  
  
 // Артефакт 89  
 new Artifact("Артефакт 89", Arrays.*asList*(  
 new Effect("income\_double", 0, 1, "Удвоение текущего дохода"),  
 new Effect("expenses\_percent", 10, 1, "Уменьшение текущих расходов на 10%"),  
 new Effect("experience\_current", 10, 3, "Увеличение опыта от текущего опыта на 10%")  
 )),  
  
 // Артефакт 96  
 new Artifact("Артефакт 96", Arrays.*asList*(  
 new Effect("expenses\_percent", 10, 1, "Уменьшение расходов на 10% от баланса"),  
 new Effect("level\_up", 1, 0, "Увеличение уровня на 1 (единоразово)"),  
 new Effect("income\_from\_balance", 10, 1, "Увеличение дохода на 10% от баланса")  
 ))  
 );  
  
 public static void runCycle(List<Colony> activeColonies, List<Colony> winners, List<Colony> losers,  
 int cycleNumber, List<String> log, List<Map<String, Object>> auctionData) {  
 log.add("\nЦикл " + cycleNumber + ": Начало.");  
  
 for (Iterator<Colony> iterator = activeColonies.iterator(); iterator.hasNext(); ) {  
 Colony colony = iterator.next();  
 if (colony.isWinner()) {  
 winners.add(colony);  
 iterator.remove();  
 continue;  
 }  
  
 colony.setRoundsPlayed(colony.getRoundsPlayed() + 1);  
 colony.applyEffects(log);  
 colony.updateBalance(log);  
 colony.checkLevelUp(log, cycleNumber);  
  
 if (!colony.isAlive()) {  
 if (colony.isWinner()) {  
 winners.add(colony);  
 } else {  
 losers.add(colony);  
 }  
 iterator.remove();  
 }  
 }  
  
 if (cycleNumber % Constants.*EVENT\_INTERVAL* == 0) {  
 log.add("\nСобытие среды:");  
  
 Random random = new Random();  
 for (Colony colony : activeColonies) {  
 Runnable event = random.nextBoolean()  
 ? () -> Environment.*dustStorm*(colony, log, 20, 10)  
 : () -> Environment.*renaissance*(colony, log, 20, 10);  
 event.run();  
 event.run();  
 }  
 }  
  
 if (cycleNumber % Constants.*AUCTION\_INTERVAL* == 0) {  
 *runAuction*(activeColonies, log, auctionData);  
 }  
 }  
  
 public static void runAuction(List<Colony> activeColonies, List<String> log, List<Map<String, Object>> auctionData) {  
 log.add("\nАукцион начинается.");  
 List<Colony> activeBidders = new ArrayList<>();  
 for (Colony colony : activeColonies) {  
 if (colony.getBalance() > 50) {  
 activeBidders.add(colony);  
 }  
 }  
  
 if (activeBidders.isEmpty()) {  
 log.add("Нет доступных колоний для участия в аукционе.");  
 return;  
 }  
  
 for (Artifact artifact : *artifactPool*) {  
 if (activeBidders.isEmpty()) {  
 break;  
 }  
  
 log.add("Лот: " + artifact.getName() + " (" + activeBidders.size() + " участников)");  
  
 // Ставки колоний  
 Map<Colony, Integer> bids = new HashMap<>();  
 Random random = new Random();  
 for (Colony colony : activeBidders) {  
 int maxBid = (int) (colony.getBalance() \* Constants.*MAX\_BET*);  
 int bid = random.nextInt(maxBid - 1) + 1;  
 bids.put(colony, bid);  
 log.add(colony.getName() + " предложила " + bid + " единиц.");  
 }  
  
 // Определение победителя  
 Colony winner = Collections.*max*(bids.entrySet(), Map.Entry.*comparingByValue*()).getKey();  
 int winningBid = bids.get(winner);  
  
 // Сохранение данных о текущем аукционе  
 Map<String, Object> auctionRecord = new HashMap<>();  
 auctionRecord.put("artifact", artifact.getName());  
 auctionRecord.put("winning\_bid", winningBid);  
 auctionRecord.put("winner\_name", winner.getName());  
 auctionRecord.put("winner\_level", winner.getLevel());  
 auctionRecord.put("participants", activeBidders.stream().map(Colony::getLevel).toList());  
 auctionData.add(auctionRecord);  
  
 // Применение артефакта и вычет ставки из баланса  
 log.add(winner.getName() + " выигрывает лот с ставкой " + winningBid + " единиц!");  
 winner.setBalance(winner.getBalance() - winningBid);  
 artifact.applyArtifact(winner, log);  
  
 // Удаление победителя из участников аукциона  
 activeBidders.remove(winner);  
 }  
  
 log.add("Аукцион завершён.");  
 }  
}

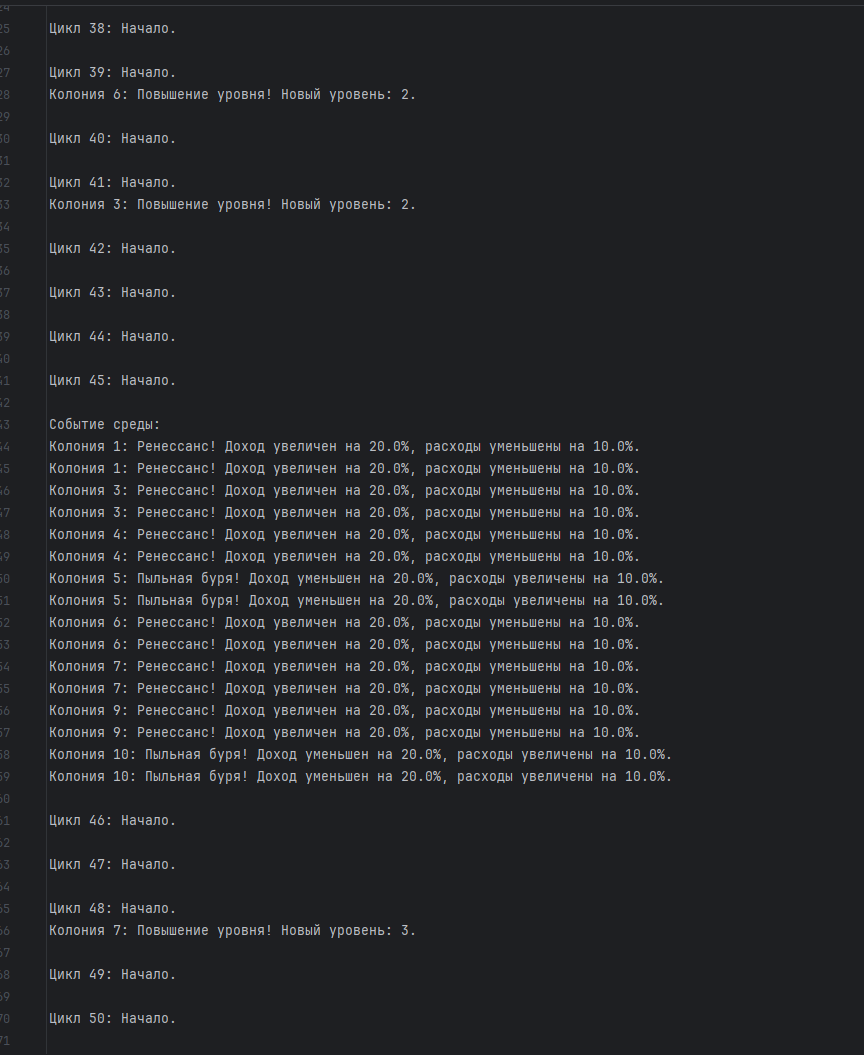
package org.example;  
  
  
import java.io.BufferedWriter;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.\*;  
  
public class App{  
 public static void main(String[] args) {  
 List<Colony> colonies = new ArrayList<>();  
 Random random = new Random();  
  
 for (int i = 0; i < Constants.*COLONY\_COUNT*; i++) {  
 colonies.add(new Colony("Колония " + (i + 1), Constants.*INITIAL\_BALANCE*,  
 random.nextInt(41) + 10, random.nextInt(26) + 5));  
 }  
  
 List<String> log = new ArrayList<>();  
 List<Colony> activeColonies = new ArrayList<>(colonies);  
 List<Colony> winners = new ArrayList<>();  
 List<Colony> losers = new ArrayList<>();  
 List<Map<String, Object>> auctionData = new ArrayList<>();  
  
 List<Integer> survivalData = new ArrayList<>();  
 List<Double> levelProgression = new ArrayList<>();  
 List<Integer> levelsGrowth = new ArrayList<>();  
 List<List<Double>> balancesByCycle = new ArrayList<>();  
 List<Integer> victoryLevels = new ArrayList<>();  
 List<Integer> defeatLevels = new ArrayList<>();  
  
 int previousTotalLevel = colonies.stream().mapToInt(Colony::getLevel).sum();  
 for (int cycle = 1; cycle <= Constants.*SIMULATION\_TIME*; cycle++) {  
 if (activeColonies.isEmpty()) {  
 break;  
 }  
 Simulation.*runCycle*(activeColonies, winners, losers, cycle, log, auctionData);  
  
 // Обновление данных для графиков  
 double avgLevel = activeColonies.stream().mapToInt(Colony::getLevel).average().orElse(0);  
 levelProgression.add(avgLevel);  
  
 int currentTotalLevel = activeColonies.stream().mapToInt(Colony::getLevel).sum();  
 levelsGrowth.add(currentTotalLevel - previousTotalLevel);  
 previousTotalLevel = currentTotalLevel;  
  
 balancesByCycle.add(activeColonies.stream().map(Colony::getBalance).toList());  
  
 for (Colony winner : winners) {  
 if (!victoryLevels.contains(winner.getLevel())) {  
 victoryLevels.add(winner.getLevel());  
 }  
 }  
  
 for (Colony loser : losers) {  
 if (!defeatLevels.contains(loser.getLevel())) {  
 defeatLevels.add(loser.getLevel());  
 }  
 }  
  
 survivalData.add(activeColonies.size());  
 }  
  
 // Сохранение лога в файл  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(Constants.*LOG\_FILE*))) {  
 for (String entry : log) {  
 writer.write(entry);  
 writer.newLine();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 // Построение графиков  
 Map<Integer, List<Double>> balanceByLevel = new HashMap<>();  
 for (Colony colony : colonies) {  
 balanceByLevel.computeIfAbsent(colony.getLevel(), k -> new ArrayList<>()).add(colony.getBalance());  
 }  
  
 Graphics.*plotAverageBalanceByLevel*(balanceByLevel); // Средний баланс по уровням  
 Graphics.*plotAuctionWinningBids*(auctionData.stream().map(data -> ((Number) data.get("winning\_bid")).doubleValue()).toList()); // Распределение победных ставок  
  
 int survivalCount = activeColonies.size();  
 int defeatCount = colonies.size() - survivalCount;  
 Graphics.*plotSurvivalVsDefeatRatio*(survivalCount, defeatCount); // Соотношение выживших и побежденных  
  
 Graphics.*plotLevelDistribution*(colonies.stream().map(Colony::getLevel).toList()); // Распределение уровней  
  
 Graphics.*plotAuctionWinProbabilityByLevel*(auctionData); // Вероятность победы в аукционах по уровням  
  
 Graphics.*plotLevelGrowthDistribution*(levelsGrowth); // Распределение прироста уровня  
  
 Graphics.*plotBalanceChange*(balancesByCycle); // Изменение балансов колоний  
  
 }  
}

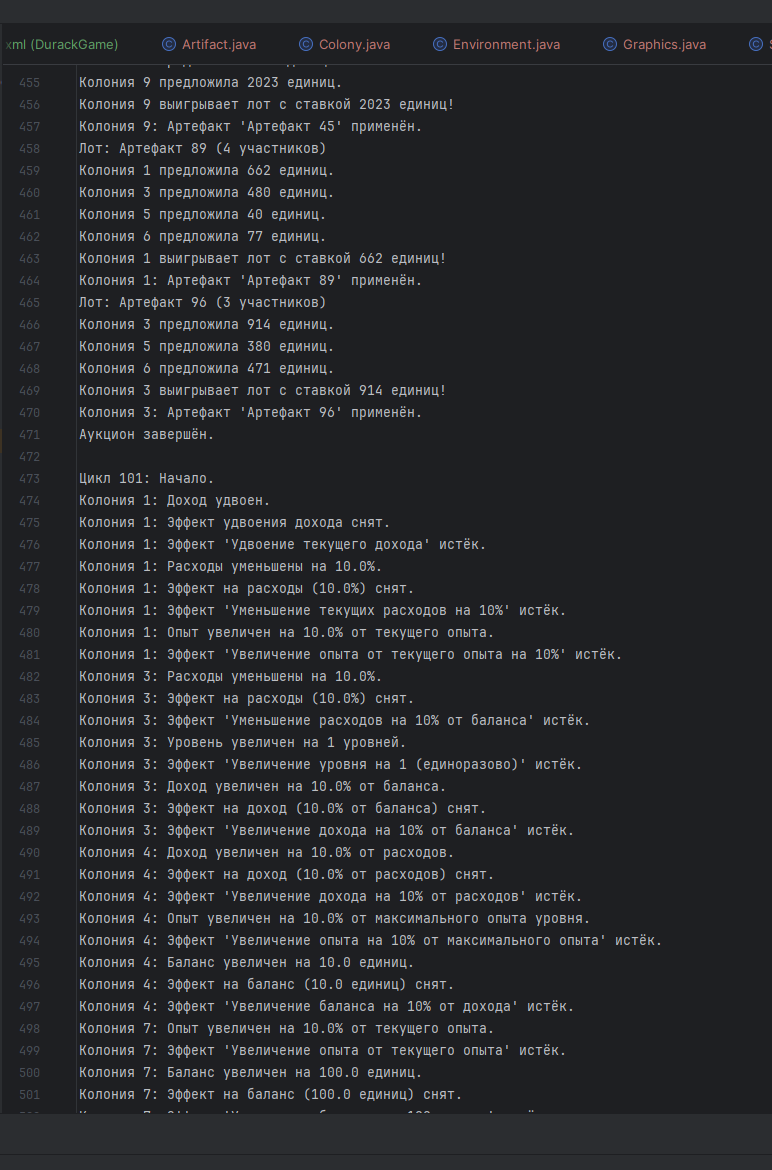
**Примеры вывода**

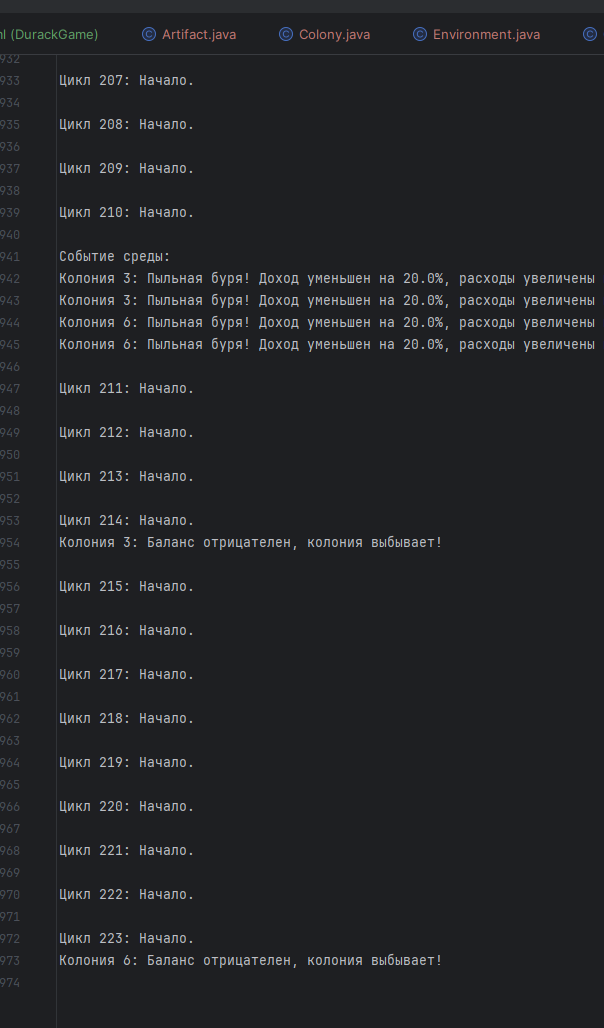






****

****

****

