Міністерство освіти і науки України

Національний університет «Чернігівська політехніка»

Кафедра інформаційних технологій та програмної інженерії

ОРГАНІЗАЦІЯ ВЗАЄМОДІЇ ОБ’ЄКТІВ,

ЩО ПРАЦЮЮТЬ ПАРАЛЕЛЬНО

«Моделювання авіаперевезень»

Курсовий проект з дисципліни

«Об’єктно-орієнтоване програмування»

Виконали

студенти групи ПІ-231 Лещенко А.М., Буряк П.В

Керівник Бивойно Т.П.

Чернігів, 2025 рік

# Зміст

[Зміст 1](#_Toc198394312)

[Завдання 2](#_Toc198394313)

[Вступ 2](#_Toc198394314)

[Анотація 2](#_Toc198394315)

[Аналіз системи що підлягає моделюванню 2](#_Toc198394316)

[Виділення абстракцій системи 2](#_Toc198394317)

[Аналіз поведінки абстракцій 3](#_Toc198394318)

[Реалізація системи 6](#_Toc198394319)

[Реалізація класів моделі 9](#_Toc198394320)

[Результати тестування програми 16](#_Toc198394321)

[Висновки 17](#_Toc198394322)

[Використані джерела 18](#_Toc198394323)

# Завдання

**Тема завдання:** Моделювання авіаперевезень, варіант “AviaBridg”

У аеропорт прибувають контейнери з гуманітарною допомогою для відправлення у зону стихійного лиха. Контейнери потрапляють на навантажувальну площадку, а з площадки бригада вантажників завантажує контейнери у літак, якщо він готовий для навантаження. Літак відправляється відразу після заповнення. Після повернення на свій аеродром літак знову готовий до завантаження.

# Вступ

У сучасних обчислювальних системах ефективне використання ресурсів комп’ютера є критично важливим завданням. Зі зростанням обсягу даних та складності програмних систем постає необхідність у підвищенні продуктивності програм без збільшення апаратних витрат. Одним із ключових підходів до досягнення цієї мети є використання багатопоточності.

У цьому курсовому проєкті розглянуто основні принципи реалізації багатопоточності, її переваги, проблеми, що виникають при її використанні, а також приклади практичного застосування в реальних програмних рішеннях.

# Анотація

Курсовий проєкт присвячений моделюванню авіаперевезень гуманітарної допомоги у надзвичайних ситуаціях. На основі сценарію AviaBridg розроблено модель, яка відображає динаміку переміщення контейнерів від моменту їх прибуття в аеропорт до доставки у зону стихійного лиха.

Особливістю реалізації є використання багатопотоковості для імітації паралельних процесів, зокрема незалежного прибуття контейнерів, підготовки літаків та роботи вантажників.

**Мета роботи:** дослідження ефективності багатопотоковості і організації процесу паралельного використання ресурсів. Побудова системи, яка імітує процес надходження та завантаження контейнерів у літак, використовуючи багатопоточність.

# Аналіз системи що підлягає моделюванню

### **Виділення абстракцій системи**

Аналізуючи опис предметної області, технічне завдання та схематичне зображення можна виділити наступні абстракції у системі:

* джерело транзакцій (джерело контейнерів);
* транзакція (контейнер);
* черга транзакцій на обробку (площадка з контейнерами);
* пристрій для обробки транзакцій (вантажники);
* лічильник транзакцій (лічильник заповнення літака);
* форма для візуального відображення роботи системи.

Схематичне зображення предметної області наведено на рисунку 1.1.

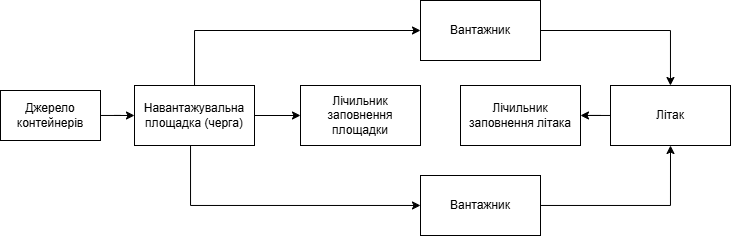


Рисунок 1.1 – Схематичне зображення предметної області

### **Аналіз поведінки абстракцій**

**Джерело транзакцій:** Імітує процес прибуття контейнерів з гуманітарною допомогою до аеропорту та відображає цей процес на формі. Після прибуття кожного контейнера постачальник аналізує стан навантажувальної площадки. Якщо площадка заповнена, контейнер очікує звільнення місця. Після появи вільного місця контейнер переміщується на навантажувальну площадку і чекає завершення процесу переміщення. Цикл прибуття контейнерів триває доти, доки не завершиться надходження гуманітарної допомоги.

Діаграма діяльності джерела транзакцій наведена на рисунку 1.2.



Рисунок 1.2 – Діаграма діяльності джерела транзакцій

**Транзакція:** Контейнер переміщується від постачальника до навантажувальної площадки, а далі — у літак. Перед початком переміщення контейнер повідомляє об’єкт, від якого він вирушає (наприклад, постачальника або площадку), а після завершення переміщення — об’єкт призначення (наприклад, навантажувальну площадку або літак). Таким чином, кожне переміщення контейнера є узгодженим із обома сторонами.

**Черга транзакцій на обробку:** Приймає, зберігає та передає контейнери. Реагує на прибуття нових контейнерів, але якщо заповнена — перенаправляє контейнер до лічильника втрачених контейнерів (що фіксує контейнери, які не вдалося завантажити). Зміни у стані (додано чи вилучено контейнер) відправляються до реєстратора подій у системі.

**Пристрій для обробки транзакцій:** Аналізує стан навантажувальної площадки. Якщо на ній немає контейнерів — чекає їх появи. Коли контейнер з’являється, бригада вилучає його з площадки, ініціює процес переміщення контейнера в літак і чекає завершення цього процесу. Далі імітується процес фізичного завантаження (може включати час затримки), що відображається у системі. Після завантаження бригада повідомляє лічильник завантажених контейнерів.

Цикл діяльності повторюється, доки на площадці є контейнери або надходить нова допомога.

Діаграма діяльності пристрою для обробки транзакцій наведена на рисунку 1.3.

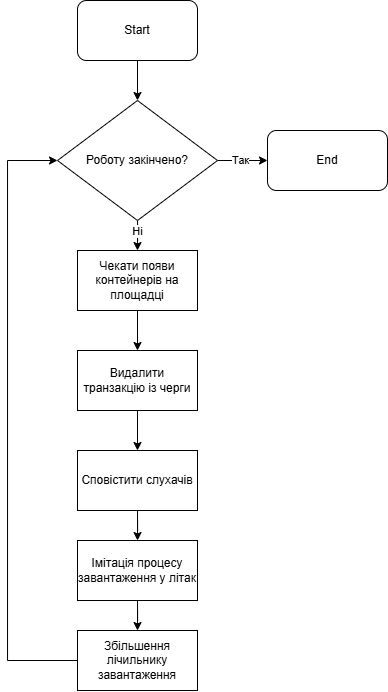


Рисунок 1.3 – Діаграма діяльності пристрою для обробки транзакцій

**Лічильник транзакцій:** підраховує кількість контейнерів, що до нього надійшли. У системі може бути кілька лічильників: один — для контейнерів, які були успішно завантажені в літак, інший — для контейнерів, які не потрапили на навантажувальну площадку через її переповнення. Лічильник відображає значення на візуальному компоненті форми, дозволяючи користувачу стежити за результатами роботи системи в режимі реального часу.

**Форма:** використовується для візуального відображення роботи всієї моделі. На ній розміщуються всі основні компоненти: постачальник контейнерів, навантажувальна площадка, бригада вантажників, літак, лічильники та інші елементи. Користувач може бачити поточний стан кожного компонента, слідкувати за переміщенням контейнерів, запускати та зупиняти симуляцію, змінювати параметри (наприклад, швидкість роботи) або очищати модель для нового запуску.

# Реалізація системи

#### **Реалізація візуальної частини проекту**

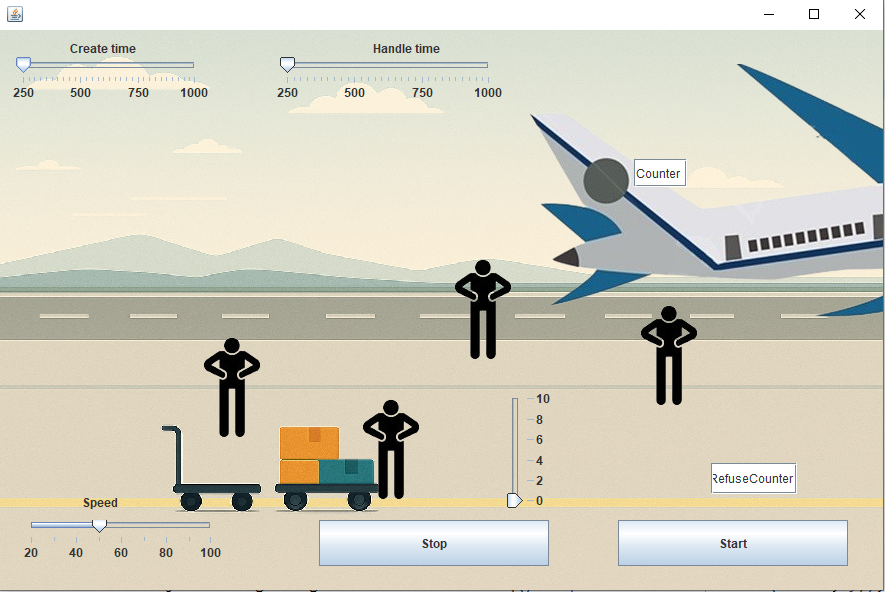


Рисунок 1.4 – Візуальний інтерфейс програми

Елементи керування:

* JSlider "Create time" (ліворуч зверху). Дозволяє змінювати час створення транзакцій (250–1000 мс)
* JSlider "Handle time" (праворуч зверху). Визначає час обробки транзакції (250–1000 мс).
* JSlider "Speed" (внизу ліворуч). Визначає швидкість анімації переміщення.
* Кнопка "Start" (праворуч). Запускає процес створення/обробки транзакцій.
* Кнопка "Stop" (по центру). Зупиняє генерацію транзакцій і викликає анімацію літака.
* Counter — показує кількість успішно оброблених транзакцій (на літаку).
* RefuseCounter — під літаком, відображає відхилені транзакції.
* Візок із коробками. Символізує чергу транзакцій. Поруч — вертикальний слайдер, який показує навантаження черги (0–10).

Метод moveFromTo анімує переміщення транзакції від одного об'єкта до іншого (наприклад, від черги до обробника або від обробника до лічильника), малюючи прямокутник, що рухається по лінії між початковою та кінцевою точками з паузами відповідно до встановленої швидкості.

Метод animatePlane відповідає за візуальну анімацію літака — після короткої затримки літак злітає вправо за межі вікна, затримується в повітрі, а потім повертається назад; захищений від повторного запуску, автоматично перезапускає обробку після завершення.

Реалізація методів наведено в лістингах 1.1 – 1.2.

Лістинг 1.1 – Текст методу moveFromTo

public Thread moveFromTo(final IfromTo from, final IfromTo to) {

Thread t = new Thread() {

public void run() {

int hT = 15, wT=15 ;

int xOffset = 20;

int xFrom = pointFrom(from).x + xOffset;

int xTo = pointTo(to).x + xOffset;

if (xFrom > xTo) {

xFrom = pointTo(from).x;

xTo = pointFrom(to).x;

}

int lenX = xTo - xFrom;

int yOffset = -30;

int yFrom = pointFrom(from).y + yOffset;

int yTo = pointTo(to).y + yOffset;

int lenY = yTo - yFrom;

int len = (int) (Math.round(Math

.sqrt(lenX \* lenX + lenY \* lenY)));

int lenT = (hT + wT) / 2;

int n = len / lenT + 1;

int dx = lenX / n;

int dy = lenY / n;

from.onOut(Transaction.this);

for (int x = xFrom, y = yFrom, i = 0; i < n; x += dx, y += dy, i++) {

Rectangle r = new Rectangle(x, y, wT, hT);

panel.addRect(r);

try {

Thread.sleep(stepTimeSlider.getValue());

} catch (InterruptedException e) {

e.printStackTrace();

}

panel.clearRects();

}

to.onIn(Transaction.this);

}

};

t.start();

return t;

}

Лістинг 1.2 – Текст методу animatePlane

public void animatePlane(Runnable onReturn) {

if (isPlaneAnimating) return; // захист від повтору

isPlaneAnimating = true;

new Thread(() -> {

try {

JLabel planeLabel = lblPlane;

int startX = planeLabel.getX();

int y = planeLabel.getY();

int frameWidth = frame.getWidth();

Thread.sleep(3000); // Пауза перед вильотом

while (planeLabel.getX() < frameWidth) {

int finalX = planeLabel.getX() + 10;

javax.swing.SwingUtilities.invokeLater(() -> planeLabel.setLocation(finalX, y));

Thread.sleep(30);

}

Thread.sleep(1500);

while (planeLabel.getX() > startX) {

int finalX = planeLabel.getX() - 10;

javax.swing.SwingUtilities.invokeLater(() -> planeLabel.setLocation(finalX, y));

Thread.sleep(30);

}

if (onReturn != null) {

onReturn.run();

}

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

isPlaneAnimating = false;

doRun();

}

}).start();

}

## **Реалізація класів моделі**

Поля класу MainGui:

* frame: об'єкт типу JFrame, головне вікно програми.
* btnStartPlayer: кнопка для запуску процесу моделювання.
* textFieldCounter: текстове поле для відображення кількості оброблених транзакцій.
* textFieldRefuseCounter: текстове поле для відображення кількості відхилених транзакцій.
* lblCreator1, lblCreator2: мітки з іконками, які представляють творців транзакцій.
* lblHandler1, lblHandler2: мітки з іконками, які представляють обробників транзакцій.
* lblPlane: мітка із зображенням літака, який анімується при досягненні порогу.
* steptimeSlider: повзунок для регулювання швидкості анімації переміщення.
* queueSlider: вертикальний повзунок, що відображає розмір черги.
* minCreateTimeSlider, minHandlTimeSlider: повзунки для налаштування часу створення та обробки транзакцій.
* backgroundImage: зображення, що використовується як фон інтерфейсу.
* contentPanel: основна панель, на якій розміщено усі компоненти.
* transactionPanel: прозора панель для малювання анімації переміщення.
* player: об'єкт класу Player, що відтворює фонову музику.
* tc1, tc2: потоки для творців транзакцій.
* th1, th2: потоки для обробників транзакцій.
* thPlay: потік для відтворення музики.
* isPlaying: логічне значення, яке визначає, чи активна симуляція.
* isCreatorWorking: логічне значення, що показує, чи працюють творці.
* isPlaneAnimating: ознака, що вказує на активну анімацію літака.

Методи класу MainGui:

* main(String[] args): стандартний метод запуску програми.
* MainGui(): конструктор, що ініціалізує інтерфейс.
* initialize(): метод, який створює всі компоненти інтерфейсу та розміщує їх на панелі.
* doRun(): запускає творців і обробників, ініціює симуляцію, скидає лічильники.
* playMusic(): запускає відтворення фонової музики в окремому потоці.
* onEndOfPlay(): виконується після завершення роботи потоків творців і обробників.
* closePlayer(): зупиняє і закриває відтворення музики.
* fullStopPlay(): повністю зупиняє симуляцію, викликається кнопкою "Stop".
* doStopPlay(): зупиняє симуляцію та запускає анімацію літака.
* animatePlane(Runnable onReturn): виконує анімацію літака (виліт і повернення), після завершення викликає задану дію.
* getGraphics(), getBackground(), getTransactionPanel(): методи-доступи до графічних ресурсів і панелі анімації.
* Гетери до основних компонентів інтерфейсу (getTextFieldCounter(), getLblCreator1() тощо).

Клас MainGui є головним класом, який ініціалізує програми та відповідає за її стан і завершення. Реалізація класу MainGui наведено в лістингу 1.3.

Лістинг 1.3 – Текст класу MainGui

protected void doRun() {

btnStartPlayer.setEnabled(false);

Counter counter = new Counter(textFieldCounter);

Counter refuseCounter = new Counter(textFieldRefuseCounter);

models.Queue queue = new models.Queue(this, refuseCounter, queueSlider);

Creator creator1 = new Creator(this, lblCreator1, queue, minCreateTimeSlider);

Creator creator2 = new Creator(this, lblCreator2, queue, minCreateTimeSlider);

Handler handler1 = new Handler(this, lblHandler1, queue, minHandlTimeSlider, counter);

Handler handler2 = new Handler(this, lblHandler2, queue, minHandlTimeSlider, counter);

startTime = System.currentTimeMillis();

thPlay = playMusic();

setIsCreatorWorking(true);

setIsPlaying(true);

(tc1 = new Thread(creator1)).start();

(tc2 = new Thread(creator2)).start();

(th1 = new Thread(handler1)).start();

(th2 = new Thread(handler2)).start();

getTextFieldCounter().setText("0");

getTextFieldRefuseCounter().setText("0");

}

Клас TransactionPanel відповідає за відопраження транзакції та реалізацію їх анімації переміщення. Реалізація класу TransactionPanel наведено в лістингу 1.4.

Лістинг 1.4 – Текст класу TransactionPanel

public class TransactionPanel extends JPanel{

private final List<Rectangle> rects = new CopyOnWriteArrayList<>();

public void addRect(Rectangle rect);

public void clearRects();

protected void paintComponent(Graphics g);

}

Клас AbstractWorker є абстрактним класом і є батьківським класом для Creator і Handler реалізує метод showWorking для імінтації роботи. Реалізація класу AbstractWorker наведено в лістингу 1.5.

Лістинг 1.5 – Текст класу AbstractWorker

public abstract class AbstractWorker implements IfromTo, Runnable{

protected Object gui;

protected Queue queue;

protected Transaction trs;

protected JLabel label;

protected JSlider minWorkTimeSlider;

public AbstractWorker(Object gui, JLabel label, Queue queue, JSlider minWorkTimeSlider) {

this.gui = gui;

this.queue = queue;

this.label = label;

this.minWorkTimeSlider = minWorkTimeSlider;

}

protected void display(String pct);

protected void showWorking(String[] pictures);

}

Клас Counter слугує лічільником для підрахунку транзакцій. Реалізація класу Counter наведено в лістингу 1.6.

Лістинг 1.6 – Текст класу Counter

public class Counter implements IfromTo{

private int count = 0;

private JTextField textField;

public Counter(JTextField textField) {

super();

this.textField = textField;

}

public void onOut(Transaction tr);

public void onIn(Transaction tr);

public Component getComponent();

public int getCounter();

public void increment();

public void setCount(int count);

}

Класи Handler і Creator є схожими класами і реалізують сутність вантажників. Різниця полягає у реалізації методу run. Реалізація класів Creator і Handler наведено в лістингу 1.7 – 1.8.

Лістинг 1.7 – Текст класу Creator

public class Creator extends AbstractWorker{

private String[] pictures = {

"/photo/man1.png",

"/photo/man2.1.png",

"/photo/man3.png",

};

public Creator(Object gui, JLabel label, Queue queue, JSlider minWorkTimeSlider) {

super(gui, label, queue, minWorkTimeSlider);

// TODO Auto-generated constructor stub

}

public void run();

public void onOut(Transaction tr);

public void onIn(Transaction tr);

public Component getComponent();

}

Лістинг 1.8 – Текст класу Handler

public class Handler extends AbstractWorker {

private Counter counter;

private String[] pictures = {

"/photo/man1.png",

"/photo/man2.1.png",

"/photo/man3.png",

};

public Handler(Object gui, JLabel label, Queue queue, JSlider minWorkTimeSlider, Counter counter) {

super(gui, label, queue, minWorkTimeSlider);

// TODO Auto-generated constructor stub

this.counter = counter;

}

public void run();

public void onOut(Transaction tr);

public void onIn(Transaction tr);

public Component getComponent();

}

Інтерфейс IfromTo задає методи для отримання компоненту і методи для маніпуляції анімацій компонентів. Реалізація інтерфейсу IfromTo наведено в лістингу 1.9.

Лістинг 1.9 – Текст класу IfromTo

public interface IfromTo {

public void onOut(Transaction tr);

public void onIn(Transaction tr);

public Component getComponent();

}

Клас Queue слугує для представлення черги завантажувального майданчику. Реалізація класу Queue наведено в лістингу 1.10.

Лістинг 1.10 – Текст класу Queue

public class Queue implements IfromTo{

private Object gui;

public Counter refuseCounter;

private ArrayDeque<Transaction> que = new ArrayDeque<Transaction>();

private JSlider slider;

private int maxSizeOfPlane = 30;

private int currentSizeOfPlane = 0;

public Queue(Object gui, Counter refuseCounter, JSlider slider) {

this.gui = gui;

this.refuseCounter = refuseCounter;

this.slider = slider;

}

public void onIn(Transaction tr);

public void addLast(Transaction tr);

public Transaction removeFirst();

public int getMaxSize();

public int getQueueSize();

public void onOut(Transaction tr);

public Component getComponent();

}

Клас Transaction реалізує одиницю контейнерів, які обробляють Handler і Creator. Реалізація класу Transaction наведено в лістингу 1.11.

Лістинг 1.11 – Текст класу Transaction

public class Transaction {

private Object gui;

private TransactionPanel panel;

private JSlider stepTimeSlider;

public Transaction(Object gui) {

this.gui = gui;

this.stepTimeSlider = ((MainGui) gui).getStepTimeSlider();

this.panel = ((MainGui) gui).getTransactionPanel();

Color color = Color.RED; //Колір транзакції

Color back = ((MainGui) gui).getBackground();

int rgb = back.getRGB() ^ color.getRGB();

}

public Point pointTo(IfromTo point);

public Point pointFrom(IfromTo point);

public Thread moveFromTo(final IfromTo from, final IfromTo to);

}

# Результати тестування програми

В результаті тестування програмного застосунку було усунено помилки і в результаті отримано працюючий та справний варіант програми. Модель успішно пройшла тестування за всіма ключовими критеріями. Вона коректно реагує на зміну параметрів, адекватно поводиться при граничних навантаженнях, візуально зрозуміла й може бути використана для моделювання та дослідження реальних систем чергування з обмеженими ресурсами та подальшою обробкою.

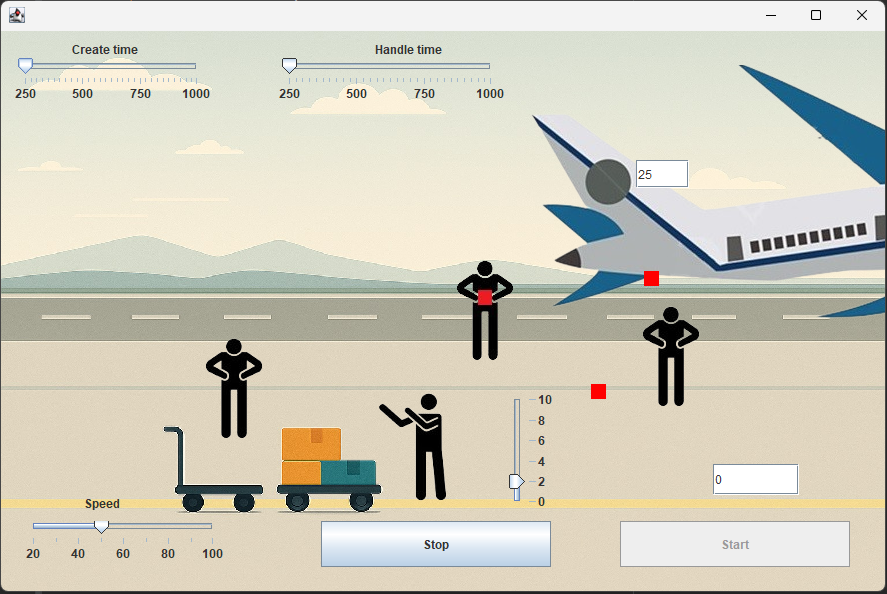


Рисунок 1.5 – Тестування при базових налаштуваннях

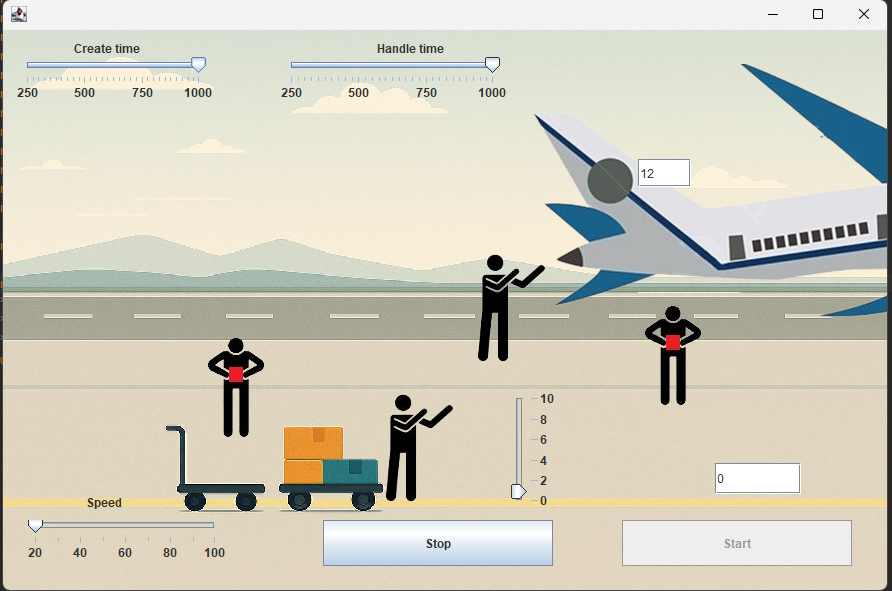


Рисунок 1.6 – Тестування при змінах налаштувань

**Висновки**

У межах курсового проєкту було успішно реалізовано модель процесу авіаперевезень гуманітарної допомоги у надзвичайних ситуаціях на прикладі сценарію “AviaBridg”. Основною метою проєкту було дослідження ефективності використання багатопоточності для моделювання паралельних процесів у системі.

У результаті роботи:

* Побудовано програмну модель, яка імітує прибуття контейнерів, їх накопичення на завантажувальній площадці, роботу бригади вантажників та завантаження літаків.
* Реалізовано паралельність ключових процесів, зокрема асинхронне прибуття контейнерів, підготовку літаків та взаємодію з вантажниками, що дозволило наблизити модель до реальних умов.
* Досліджено синхронізацію між потоками з метою уникнення конфліктів доступу до спільних ресурсів (наприклад, завантажувальна площадка чи літак).
* Оцінено ефективність багатопотокового підходу для задач моделювання подібних логістичних процесів, що підтвердило доцільність його використання для покращення продуктивності й точності симуляцій.

Отримані результати свідчать про те, що багатопоточна модель забезпечує гнучке управління ресурсами та дозволяє ефективно моделювати складні сценарії логістичних операцій. Такий підхід може бути корисним для планування та оптимізації реальних систем гуманітарних перевезень, особливо в умовах обмеженого часу та ресурсів.

Використані джерела

1. Бадд Т. Объектно-ориентированное программирование в действии. Санкт-Петербург: Питер, 1997.
2. Буч Г. Объектно-ориентированное проектирование с примерами применения: Пер. с англ. – М.:Кокорд,1992. – 519с
3. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++: Пер. с англ. – М.:Диалект,1999
4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. – СПб.:Питер,2001
5. Фаулер МартинАрхитектура корпоративных программных приложений. СПб.: Издательский дом «Вильямс», 2007. – 544с
6. Simon Kendal. Object oriented programming using Java. Ventus Publishing ApS-, 2009. – 209 c.
7. http://omg.org Object Management Group
8. \\kid\incoming\byvoino\4\_курс\Simulation