
*Wojskowa Akademia Techniczna*
*im. Jarosława Dąbrowskiego*

# Programowanie współbieżne

## Sprawozdanie z programu zaliczeniowego

## Temat projektu: Stadion

**Prowadzący:** dr inż. Jarosław Rulka
**Wykonał:** Konrad Bobryk
**Grupa:** I5X3S1

# Treść zadania

Grupa: **I5*S1**
Zadanie nr: **PW-30/2017**

Język implementacji: **Java**
Środowisko implementacyjne: **Eclipse, IntelliJ IDEA**
Termin wykonania: **ostatnie zajęcia laboratoryjne**

Podstawowe wymagania:
a. liczba procesów sekwencyjnych powinna być dobrana z wyczuciem tak, aby zachować czytelność interfejsu i jednocześnie umożliwić zobrazowanie reprezentatywnych przykładów,
b. kod źródłowy programu musi być tak skonstruowany, aby można było „swobodnie" modyfikować liczbę procesów sekwencyjnych (za wyjątkiem zadań o ściśle określonej liczbie procesów),
c. obok poprawnej identyfikacji sekcji krytycznych program musi brać pod uwagę czytelność i estetykę interfejsu użytkownika oraz zdolność percepcji osoby oceniającej,
d. interfejs nie powinien opierać się na zasadzie wypisywania kolejnych linii na ekran.

Sprawozdanie (w formie elektronicznej) powinno zawierać następujące elementy:
1) stronę tytułową,
2) niniejszą treść zadania,
3) syntetyczny opis problemu – przyjęte założenia,
4) wykaz współdzielonych zasobów,
5) wykaz wyróżnionych sekcji krytycznych,
6) wykaz obiektów synchronizacji,
7) wykaz procesów sekwencyjnych,
8) listing programu.

Problem do rozwiązania:
Stadion.
Założenia.
Na stadionie piłkarskim rozegrany ma zostać mecz finałowy Ligi Mistrzów. Z uwagi na rangę imprezy ustalono następujące rygorystyczne zasady bezpieczeństwa.
• Na stadionie może przebywać maksymalnie K kibiców.
• Wejście na stadion możliwe będzie tylko po przejściu drobiazgowej kontroli, mającej zapobiec wnoszeniu przedmiotów niebezpiecznych.
• Kontrola przy wejściu jest przeprowadzana równolegle na 3 stanowiskach, na każdym z nich mogą znajdować się równocześnie maksymalnie 3 osoby.
• Jeśli kontrolowana jest więcej niż 1 osoba równocześnie na stanowisku, to należy zagwarantować, by byli to kibice tej samej drużyny.
• Kibic oczekujący na kontrolę może przepuścić w kolejce maksymalnie 5 innych kibiców. Dłuższe czekanie wywołuje jego frustrację i agresywne zachowanie, którego należy unikać za wszelką cenę.

## Syntetyczny opis problemu – przyjęte założenia

Wypisane powyżej w treści zadania. Dodatkowo:

- kibice pojawiają się co pewien losowy czas;

- po pojawieniu się (które odbywa się poza panelem) kibic idzie w stronę kolejki;

- punkty kontrolne dążą do przyjęcia maksymalnej dopuszczalnej liczby kibiców na kontrolę (o ile jest to możliwe);

- rozpoczęcie kontroli następuje gdy ostatni z kibiców wziętych na kontrolę zajmie miejsce przy stanowisku kontrolnym;

- zarówno częstotliwość pojawiania się kibiców, jak i czas trwania kontroli na stanowiskach może być regulowany przez użytkownika w panelu „Opcje”;

- punkty kontrolne przestają przyjmować kibiców, gdy suma kontrolowanych kibiców na wszystkich stanowiskach i kibiców wpuszczonych na stadion przekroczy maksymalną liczbę kibiców na stadionie;

- maksymalna liczba kibiców na stadionie może być modyfikowana w panelu „Opcje”;

- kibice, którzy przepuścili kibiców przeciwnej drużyny, przechodzą do kolejki priorytetowej;

- punkty kontrolne zawsze najpierw biorą kibiców z kolejki priorytetowej, dopiero po jej opróżnieniu mogą znów zająć się kibicami w kolejce „głównej”;

- nie może dojść do sytuacji, że w kolejce priorytetowej ktoś zostanie (oznaczałoby to, że kibic nie wszedł na stadion tylko dlatego, że przepuścił kibica przeciwnej drużyny, co byłoby niesprawiedliwe);

- punkty kontrolne wybierając kibiców do kontroli biorą pod uwagę kibiców z kolejki priorytetowej oraz tych z „głównej”, którzy zajęli w niej swoje miejsce.

## Wykaz współdzielonych zasobów

- kolejka kibiców;

- priorytetowa kolejka kibiców;

- liczba kibiców wpuszczonych na stadion.

## Wykaz wyróżnionych sekcji krytycznych

- wybieranie kibiców na kontrolę;

- wpuszczanie kibiców na stadion (po przejściu kontroli).

# Wykaz obiektów synchronizacji

- stanowiska kontrolne.

# Wykaz procesów sekwencyjnych

- pojawianie się kibiców;

- funkcjonowanie stanowisk kontrolnych;

- proces główny (rysowanie).

# Listing programu

```
package stadion;

import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.JPanel;

public class AboutPanel extends JPanel {

	private static final long serialVersionUID = 1L;
	private BufferedImage img;
	private JButton menu;

	public AboutPanel() {
		loadImage();
		loadButtons();
	}

	private void loadImage() {
		try {
			img = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/about.jpg"));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void paint(Graphics g) {
		super.paintComponent(g);
		g.drawImage(img, 0, 0, null);
	}

	private void loadButtons() {
		int buttonWidth = 150;
		int buttonHeight = 30;
		menu = new JButton("Powrót do menu");
		menu.setVisible(false);
		menu.setBounds(200, 460, buttonWidth, buttonHeight);
		menu.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				MainStadion.setMenuPoint(true);
				unseenButtons();
			}
		});
```

```
                MainStadion.getWindow().add(menu);
        }

        public void seenButtons() {
                menu.setVisible(true);
        }

        private void unseenButtons() {
                menu.setVisible(false);
        }

}
```

```
package stadion;

import java.util.LinkedList;
import java.util.Random;

public class ControlPoint implements Runnable {

        private int x, y, controlNumber;
        private LinkedList<Fan> control = new LinkedList<Fan>();
        Random rand = new Random();

        public ControlPoint(int x, int y, int controlNumb) {
                this.x = x;
                this.y = y;
                this.controlNumber = controlNumb;
                if(this.controlNumber==1) {
                        QueueManagement.control1 = control;
                } else if(this.controlNumber==2) {
                        QueueManagement.control2 = control;
                } else if(this.controlNumber==3) {
                        QueueManagement.control3 = control;
                }
        }

        public void run() {
                while(true) {
                        while(SimulationPanel.getIsSimulationGoing()==true &&
((SimulationPanel.getNumberOfFansInStadium()
                                        + QueueManagement.tempQ.size() +
QueueManagement.control1.size() + QueueManagement.control2.size()
                                        + QueueManagement.control3.size()) <
SimulationPanel.getMaxNumberOfFansInStadium()
                                        || QueueManagement.tempQ.isEmpty()==false)) {

                                SimulationPanel.lock1.lock();
                                enter();
                                SimulationPanel.lock1.unlock();
                                while(control.isEmpty()==false &&
control.getLast().getStatus()!=3) {
                                        try {
                                                Thread.sleep(50);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                                }
                                try {

        Thread.sleep(rand.nextInt(SimulationPanel.getMaxTimeOfControl()-
SimulationPanel.getMinTimeOfControl())
                                                        +SimulationPanel.getMinTimeOfControl());
                                } catch (InterruptedException e) {
                                        e.printStackTrace();
                                }
                                SimulationPanel.lock2.lock();
                                exit();
                                SimulationPanel.lock2.unlock();
                        }
                        try {
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
```

```
	private void enter() {
		String tempTeam;
		int i = 0;
		int j = 3;
		int help = 0;
		boolean wasTempQEmpty;
		while(QueueManagement.tempQ.isEmpty()==true &&
(QueueManagement.fanQ.isEmpty()==true
				|| QueueManagement.fanQ.getFirst().getStatus()==0)) {
			try {
				Thread.sleep(50);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
		if(QueueManagement.tempQ.isEmpty()==true) {
			tempTeam = QueueManagement.fanQ.getFirst().getTeam();
			wasTempQEmpty = true;
		} else {
			tempTeam = QueueManagement.tempQ.getFirst().getTeam();
			wasTempQEmpty = false;
		}

		if(wasTempQEmpty==false && QueueManagement.tempQ.size()>=3) {
			j = 3;
		} else {
			if(wasTempQEmpty==false && QueueManagement.tempQ.size()<3) {
				j = QueueManagement.tempQ.size();
				help = 0;
			} else if(wasTempQEmpty==true) {
				j = 1;
				help = 1;
			}
			while(j<3 && QueueManagement.fanQ.isEmpty()==false &&
(SimulationPanel.getNumberOfFansInStadium()
			+ QueueManagement.tempQ.size() + QueueManagement.control1.size() +
QueueManagement.control2.size()
			+ QueueManagement.control3.size() + help) <
SimulationPanel.getMaxNumberOfFansInStadium() && help<QueueManagement.fanQ.size()) {
				if(tempTeam==QueueManagement.fanQ.get(help).getTeam()) {
					j++;
				}
				help++;
			}
		}

		while(i<j && (QueueManagement.fanQ.getFirst().getStatus()==1 ||
QueueManagement.tempQ.isEmpty()==false))  {
			if(QueueManagement.tempQ.isEmpty()==false && wasTempQEmpty==false) {
				shiftingTempQ();
				control.add(QueueManagement.tempQ.removeFirst());
				control.get(i).setStatus(2);
				control.get(i).setQX(x+i*30);
				control.get(i).setQY(y);
				i++;
			} else {
				wasTempQEmpty = true;
				if(QueueManagement.fanQ.getFirst().getStatus()==1) {
					if(QueueManagement.fanQ.getFirst().getTeam()==tempTeam) {
						shiftingFanQ();
						control.add(QueueManagement.fanQ.removeFirst());
						control.get(i).setStatus(2);
						control.get(i).setQX(x+i*30);
						control.get(i).setQY(y);
						i++;
					} else {
						shiftingFanQ();

	QueueManagement.tempQ.add(QueueManagement.fanQ.removeFirst());

	QueueManagement.tempQ.getLast().setQX(QueueManagement.getEndOfTempQX());

	QueueManagement.tempQ.getLast().setQY(QueueManagement.getEndOfTempQY());

	QueueManagement.setEndOfTempQX(QueueManagement.getEndOfTempQX()+30);
					}
				} else {
```

```
                                        break;
                                }
                        }
                }
        }

        private void shiftingFanQ() {
                for(int i=QueueManagement.fanQ.size()-1; i>=1; i--) {
                        QueueManagement.fanQ.get(i).setQX(QueueManagement.fanQ.get(i-
1).getQX());
                        QueueManagement.fanQ.get(i).setQY(QueueManagement.fanQ.get(i-
1).getQY());
                }
                QueueManagement.setEndOfQX(QueueManagement.getEndOfQX()-30);
        }

        private void shiftingTempQ() {
                for(int i=QueueManagement.tempQ.size()-1; i>=1; i--) {
                        QueueManagement.tempQ.get(i).setQX(QueueManagement.tempQ.get(i-
1).getQX());
                        QueueManagement.tempQ.get(i).setQY(QueueManagement.tempQ.get(i-
1).getQY());
                }
                QueueManagement.setEndOfTempQX(QueueManagement.getEndOfTempQX()-30);
        }

        private void exit() {

        SimulationPanel.setNumberOfFansInStadium(SimulationPanel.getNumberOfFansInStadium()+con
trol.size());
                control.clear();
        }

}
```

```
package stadion;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;
import javax.imageio.ImageIO;

public class Fan {

        BufferedImage fan1, fan2;
        private int x;
        private int y;
        private int qX;
        private int qY;
        private String team;
        private int status = 0;
        Random rand = new Random();

        public Fan(int xCoor, int yCoor) {
                this.x = xCoor;
                this.y = yCoor;
                if(rand.nextInt(2) == 0) {
                        this.team = "blue";
                } else {
                        this.team = "red";
                }
                this.qX = QueueManagement.getEndOfQX();
                this.qY = QueueManagement.getEndOfQY();
                QueueManagement.setEndOfQX(QueueManagement.getEndOfQX()+30);
                loadImage();
        }

        public int getQX() {
                return qX;
        }

        public void setQX(int newQX) {
                this.qX = newQX;
        }

        public int getQY() {
```

```
                return qY;
        }

        public void setQY(int newQY) {
                this.qY = newQY;
        }

        public String getTeam() {
                return team;
        }

        public int getStatus() {
                return status;
        }

        public void setStatus(int newStatus) {
                this.status = newStatus;
        }

        private void loadImage() {
                try {
                        fan1 = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/fan1.jpg"));
                        fan2 = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/fan2.jpg"));
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        public void drawFan(Graphics g) {
                if(this.team == "blue") {
                        g.drawImage(fan1, x, y, null);
                } else if(this.team == "red") {
                        g.drawImage(fan2, x, y, null);
                }
        }

        public void move() {
                if(x>qX) {
                        x--;
                } else if(x<qX) {
                        x++;
                }
                if(y>qY) {
                        y--;
                } else if(y<qY) {
                        y++;
                }
                if(x==qX && y==qY && status==0) {
                        status = 1;
                } else if(x==qX && y==qY && status==2){
                        status = 3;
                }
        }

}
```

```
package stadion;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;
import javax.swing.JFrame;
import javax.swing.Timer;

public class MainStadion {

        private static JFrame window;
        private static final int width = 1200;
        private static final int height = 630;
        private static boolean simulationPoint = false;
        private static boolean optionsPoint = false;
        private static boolean menuPoint = false;
        private static boolean aboutPoint = false;
        private static Timer timer;
        private static Random rand = new Random();
```

```
	public MainStadion() {
		window = new JFrame();
		window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		window.setSize(width, height);
		window.setLocationRelativeTo(null);
		window.setTitle("Stadion");
		window.setResizable(false);
	}

	public static JFrame getWindow() {
		return window;
	}

	public static int getWidth() {
		return width;
	}

	public static int getHeight() {
		return height;
	}

	public static Timer getTimer() {
		return timer;
	}

	public static void setSimulationPoint(boolean sp) {
		simulationPoint = sp;
	}

	public static void setOptionsPoint(boolean op) {
		optionsPoint = op;
	}

	public static void setMenuPoint(boolean menp) {
		menuPoint = menp;
	}

	public static void setAboutPoint(boolean ap) {
		aboutPoint = ap;
	}

	private void rendering() {
		MenuPanel mp = new MenuPanel();
		SimulationPanel sp = new SimulationPanel();
		OptionsPanel op = new OptionsPanel();
		AboutPanel ap = new AboutPanel();
		timer = new Timer(20, new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				sp.repaint();
				sp.move();
			}
		});
		Container c = mp;
		window.add(c);
		mp.seenButtons();
		window.setVisible(true);
		while(true) {
			while(MainStadion.simulationPoint==false &&
MainStadion.optionsPoint==false
				&& MainStadion.aboutPoint==false && MainStadion.menuPoint
== false) {
				try {
					Thread.sleep(10);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
			c.setVisible(false);
			if(MainStadion.simulationPoint==true) {
				c = sp;
				window.add(c);
				sp.seenButtons();
				MainStadion.simulationPoint = false;
			} else if(MainStadion.optionsPoint==true) {
				c = op;
				window.add(c);
```

```
                        op.seenButtons();
                        MainStadion.optionsPoint = false;
                    } else if(MainStadion.aboutPoint==true) {
                        c = ap;
                        window.add(c);
                        ap.seenButtons();
                        MainStadion.aboutPoint = false;
                    } else if(MainStadion.menuPoint==true) {
                        c = mp;
                        window.add(mp);
                        mp.seenButtons();
                        MainStadion.menuPoint = false;
                    }
                    c.setVisible(true);
                    if(c == sp) {
                        timer.start();
                        SimulationPanel.setIsSimulationGoing(true);
                    } else {
                        timer.stop();
                    }
                }
        }

        public static void main(String[] args) {
                MainStadion stadion = new MainStadion();
                Thread fans = new Thread(new Runnable() {
                        public void run() {
                                while(true) {
                                        if(SimulationPanel.getIsSimulationGoing() == true) {
                                                try {

        Thread.sleep(rand.nextInt(SimulationPanel.getMaxTimeBetweenSpawningFans()-

        SimulationPanel.getMinTimeBetweenSpawningFans())+SimulationPanel.getMinTimeBetweenSpawn
ingFans());
                                                } catch (InterruptedException e) {
                                                        e.printStackTrace();
                                                }
                                                QueueManagement.fanQ.add(new Fan(1200, 250));
                                        } else {
                                                try {
                                                        Thread.sleep(100);
                                                } catch (InterruptedException e) {
                                                        e.printStackTrace();
                                                }
                                        }
                                }
                        }
                });
                fans.start();
                Thread cp1 = new Thread(new ControlPoint(375, 150, 1));
                Thread cp2 = new Thread(new ControlPoint(375, 350, 2));
                Thread cp3 = new Thread(new ControlPoint(375, 550, 3));
                cp1.start();
                cp2.start();
                cp3.start();
                stadion.rendering();
        }

}
```

```
package stadion;

import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.JPanel;

public class MenuPanel extends JPanel {

        private static final long serialVersionUID = 1L;
        private BufferedImage img;
        private JButton symulacja, opcje, about;
```

```
	public MenuPanel() {
		loadImage();
		loadButtons();
	}

	private void loadImage() {
		try {
			img = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/menu.jpg"));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void paint(Graphics g) {
		super.paintComponent(g);
		g.drawImage(img, 0, 0, null);
	}

	private void loadButtons() {
		int buttonWidth = 100;
		int buttonHeight = 30;
		symulacja = new JButton("Symulacja");
		symulacja.setVisible(false);
		symulacja.setBounds(MainStadion.getWidth()/2 - buttonWidth/2,
MainStadion.getHeight()/2 - 100, buttonWidth, buttonHeight);
		symulacja.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				MainStadion.setSimulationPoint(true);
				unseenButtons();
			}
		});
		opcje = new JButton("Opcje");
		opcje.setVisible(false);
		opcje.setBounds(MainStadion.getWidth()/2 - buttonWidth/2,
MainStadion.getHeight()/2, buttonWidth, buttonHeight);
		opcje.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				MainStadion.setOptionsPoint(true);
				unseenButtons();
			}
		});
		about = new JButton("About");
		about.setVisible(false);
		about.setBounds(MainStadion.getWidth()/2 - buttonWidth/2,
MainStadion.getHeight()/2 + 100, buttonWidth, buttonHeight);
		about.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				MainStadion.setAboutPoint(true);
				unseenButtons();
			}
		});
		MainStadion.getWindow().add(symulacja);
		MainStadion.getWindow().add(opcje);
		MainStadion.getWindow().add(about);
	}

	public void seenButtons() {
		symulacja.setVisible(true);
		opcje.setVisible(true);
		about.setVisible(true);
	}

	private void unseenButtons() {
		symulacja.setVisible(false);
		opcje.setVisible(false);
		about.setVisible(false);
	}

}
```

```
package stadion;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class OptionsPanel extends JPanel {

	private static final long serialVersionUID = 1L;
	private BufferedImage img;
	private JButton menu, symulacja;
	private JTextField maxNumberOfFansInStadium, minTimeOfControl, maxTimeOfControl,
	minTimeBetweenSpawningFans, maxTimeBetweenSpawningFans;

	public OptionsPanel() {
		loadImage();
		loadButtons();
	}

	private void loadImage() {
		try {
			img = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/stadion.jpg"));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void paint(Graphics g) {
		super.paintComponent(g);
		g.drawImage(img, 0, 0, null);
		g.setFont(new Font("Tahoma", Font.BOLD, 30));
		g.setColor(Color.BLACK);
		g.drawString("Opcje", 1028, 25);
		g.setFont(new Font("Tahoma", Font.PLAIN, 20));
		g.drawString("Maksymalna liczba", 998, 70);
		g.drawString("kibiców na stadionie", 990, 90);
		g.drawString("5 <=", 1000, 120);
		g.drawString("<= 10000", 1100, 120);
		g.drawString("Minimalny czas", 1003, 170);
		g.drawString("kontroli kibica (ms)", 990, 190);
		g.drawString("500 <=", 978, 220);
		g.drawString(" < 60000", 1100, 220);
		g.drawString("Maksymalny czas", 998, 270);
		g.drawString("kontroli kibica (ms)", 990, 290);
		g.drawString(" 500 <", 978, 320);
		g.drawString("<= 60000", 1100, 320);
		g.drawString("Co ile pojawiają się", 993, 370);
		g.drawString("kibice - minimum (ms)", 980, 390);
		g.drawString("500 <=", 978, 420);
		g.drawString(" < 60000", 1100, 420);
		g.drawString("Co ile pojawiają się", 993, 470);
		g.drawString("kibice - maksimum (ms)", 972, 490);
		g.drawString(" 500 <", 978, 520);
		g.drawString("<= 60000", 1100, 520);
		g.setFont(new Font("Tahoma", Font.PLAIN, 11));
		g.drawString("min czas kontroli < max czas kontroli", 994, 550);
		g.drawString("min czas pojawiania < max czas pojawiania", 978, 565);
		g.drawString("Po wprowadzeniu danych naciśnij Enter", 988, 580);
	}

	private boolean isNumeric(String str) {
		for(char c : str.toCharArray()) {
			if(Character.isDigit(c)==false) {
				return false;
			}
		}
		return true;
	}

	private void loadButtons() {
		int buttonWidth = 100;
		int buttonHeight = 30;
		symulacja = new JButton("Symulacja");
```

```
		symulacja.setVisible(false);
		symulacja.setBounds(325, 200, buttonWidth, buttonHeight);
		symulacja.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				MainStadion.setSimulationPoint(true);
				unseenButtons();
			}
		});
		menu = new JButton("Menu");
		menu.setVisible(false);
		menu.setBounds(525, 200, buttonWidth, buttonHeight);
		menu.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				MainStadion.setMenuPoint(true);
				unseenButtons();
			}
		});
		maxNumberOfFansInStadium = new
JTextField(SimulationPanel.getMaxNumberOfFansInStadium().toString());
		maxNumberOfFansInStadium.setVisible(false);
		maxNumberOfFansInStadium.setBounds(1050, 100, buttonWidth/2, buttonHeight);
		maxNumberOfFansInStadium.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				if(isNumeric(maxNumberOfFansInStadium.getText())==true &&
Integer.parseInt(maxNumberOfFansInStadium.getText())>=5
					&&
Integer.parseInt(maxNumberOfFansInStadium.getText())<=10000) {

	SimulationPanel.setMaxNumberOfFansInStadium(Integer.parseInt(maxNumberOfFansInStadium.g
etText()));
				} else {

	maxNumberOfFansInStadium.setText(SimulationPanel.getMaxNumberOfFansInStadium().toString
());
				}
			}
		});
		minTimeOfControl = new
JTextField(SimulationPanel.getMinTimeOfControl().toString());
		minTimeOfControl.setVisible(false);
		minTimeOfControl.setBounds(1050, 200, buttonWidth/2, buttonHeight);
		minTimeOfControl.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if(isNumeric(minTimeOfControl.getText())==true &&
Integer.parseInt(minTimeOfControl.getText())>=500
					&&
Integer.parseInt(minTimeOfControl.getText())<SimulationPanel.getMaxTimeOfControl()) {

	SimulationPanel.setMinTimeOfControl(Integer.parseInt(minTimeOfControl.getText()));
				} else {

	minTimeOfControl.setText(SimulationPanel.getMinTimeOfControl().toString());
				}
			}
		});
		maxTimeOfControl = new
JTextField(SimulationPanel.getMaxTimeOfControl().toString());
		maxTimeOfControl.setVisible(false);
		maxTimeOfControl.setBounds(1050, 300, buttonWidth/2, buttonHeight);
		maxTimeOfControl.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if(isNumeric(maxTimeOfControl.getText())==true &&
Integer.parseInt(maxTimeOfControl.getText())<=60000
					&&
Integer.parseInt(maxTimeOfControl.getText())>SimulationPanel.getMinTimeOfControl()) {

	SimulationPanel.setMaxTimeOfControl(Integer.parseInt(maxTimeOfControl.getText()));
				} else {

	maxTimeOfControl.setText(SimulationPanel.getMaxTimeOfControl().toString());
				}
			}
		});
		minTimeBetweenSpawningFans = new
JTextField(SimulationPanel.getMinTimeBetweenSpawningFans().toString());
		minTimeBetweenSpawningFans.setVisible(false);
		minTimeBetweenSpawningFans.setBounds(1050, 400, buttonWidth/2, buttonHeight);
```

```
		minTimeBetweenSpawningFans.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if(isNumeric(minTimeBetweenSpawningFans.getText())==true &&
Integer.parseInt(minTimeBetweenSpawningFans.getText())>=500
					&&
Integer.parseInt(minTimeBetweenSpawningFans.getText())<SimulationPanel.getMaxTimeBetweenSpawni
ngFans()) {

	SimulationPanel.setMinTimeBetweenSpawningFans(Integer.parseInt(minTimeBetweenSpawningFa
ns.getText()));
				} else {

	minTimeBetweenSpawningFans.setText(SimulationPanel.getMinTimeBetweenSpawningFans().toSt
ring());
				}
			}
		});
		maxTimeBetweenSpawningFans = new
JTextField(SimulationPanel.getMaxTimeBetweenSpawningFans().toString());
		maxTimeBetweenSpawningFans.setVisible(false);
		maxTimeBetweenSpawningFans.setBounds(1050, 500, buttonWidth/2, buttonHeight);
		maxTimeBetweenSpawningFans.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if(isNumeric(maxTimeBetweenSpawningFans.getText())==true &&
Integer.parseInt(maxTimeBetweenSpawningFans.getText())<=60000
					&&
Integer.parseInt(maxTimeBetweenSpawningFans.getText())>SimulationPanel.getMinTimeBetweenSpawni
ngFans()) {

	SimulationPanel.setMaxTimeBetweenSpawningFans(Integer.parseInt(maxTimeBetweenSpawningFa
ns.getText()));
				} else {

	maxTimeBetweenSpawningFans.setText(SimulationPanel.getMaxTimeBetweenSpawningFans().toSt
ring());
				}
			}
		});
		MainStadion.getWindow().add(symulacja);
		MainStadion.getWindow().add(menu);
		MainStadion.getWindow().add(maxNumberOfFansInStadium);
		MainStadion.getWindow().add(minTimeOfControl);
		MainStadion.getWindow().add(maxTimeOfControl);
		MainStadion.getWindow().add(minTimeBetweenSpawningFans);
		MainStadion.getWindow().add(maxTimeBetweenSpawningFans);
	}

	public void seenButtons() {
		symulacja.setVisible(true);
		menu.setVisible(true);
		maxNumberOfFansInStadium.setVisible(true);
		minTimeOfControl.setVisible(true);
		maxTimeOfControl.setVisible(true);
		minTimeBetweenSpawningFans.setVisible(true);
		maxTimeBetweenSpawningFans.setVisible(true);
	}

	private void unseenButtons() {
		symulacja.setVisible(false);
		menu.setVisible(false);
		maxNumberOfFansInStadium.setVisible(false);
		minTimeOfControl.setVisible(false);
		maxTimeOfControl.setVisible(false);
		minTimeBetweenSpawningFans.setVisible(false);
		maxTimeBetweenSpawningFans.setVisible(false);
	}

}
```

```
package stadion;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class Police {
```

```
	BufferedImage[] police = new BufferedImage[7];

	public Police() {
		loadImages();
	}

	private void loadImages() {
		for(int i=0; i<police.length; i++) {
			String temp = "C:/Users/Konrad/Desktop/java/Stadion/Images/policeman" +
(i+1) + ".jpg";
			try {
				police[i] = ImageIO.read(new File(temp));
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}

	public void drawPolice(Graphics g) {
		for(int i=0; i<3; i++) {
			g.drawImage(police[i], 375, 50*i, null);
		}
		g.drawImage(police[3], 375, 250, null);
		g.drawImage(police[4], 375, 300, null);
		g.drawImage(police[5], 375, 450, null);
		g.drawImage(police[6], 375, 500, null);
	}

}
```

---

```
package stadion;

import java.util.LinkedList;

public class QueueManagement {

	private static int endOfQX = 600;
	private static int endOfQY = 250;
	private static int endOfTempQX = 540;
	private static int endOfTempQY = 350;
	public static LinkedList<Fan> fanQ = new LinkedList<Fan>();
	public static LinkedList<Fan> tempQ = new LinkedList<Fan>();
	public static LinkedList<Fan> control1;
	public static LinkedList<Fan> control2;
	public static LinkedList<Fan> control3;

	public QueueManagement() {

	}

	public static int getEndOfQX() {
		return endOfQX;
	}

	public static void setEndOfQX(int x) {
		endOfQX = x;
	}

	public static int getEndOfQY() {
		return endOfQY;
	}

	public static void setEndOfQY(int y) {
		endOfQY = y;
	}

	public static int getEndOfTempQX() {
		return endOfTempQX;
	}

	public static void setEndOfTempQX(int x) {
		endOfTempQX = x;
	}

	public static int getEndOfTempQY() {
		return endOfTempQY;
	}
```

```
	public static void setEndOfTempQY(int y) {
		endOfTempQY = y;
	}

}
```

```
package stadion;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.JPanel;

public class SimulationPanel extends JPanel {

	private static final long serialVersionUID = 1L;
	private BufferedImage img, img2, img3, img4;
	private JButton menu, opcje, reset;
	private static boolean isSimulationGoing = false;
	private static int numberOfFansInStadium = 0;
	private static Integer maxNumberOfFansInStadium = 100;
	private static Integer minTimeOfControl = 5000;
	private static Integer maxTimeOfControl = 10000;
	private static Integer minTimeBetweenSpawningFans = 1000;
	private static Integer maxTimeBetweenSpawningFans = 2000;
	public static Lock lock1 = new ReentrantLock();
	public static Lock lock2 = new ReentrantLock();
	Police police = new Police();

	public SimulationPanel() {
		loadImages();
		loadButtons();
	}

	public static boolean getIsSimulationGoing() {
		return isSimulationGoing;
	}

	public static void setIsSimulationGoing(boolean newBool) {
		isSimulationGoing = newBool;
	}

	public static int getNumberOfFansInStadium() {
		return numberOfFansInStadium;
	}

	public static void setNumberOfFansInStadium(int newNumber) {
		numberOfFansInStadium = newNumber;
	}

	public static Integer getMaxNumberOfFansInStadium() {
		return maxNumberOfFansInStadium;
	}

	public static void setMaxNumberOfFansInStadium(int newNumber) {
		maxNumberOfFansInStadium = newNumber;
	}

	public static Integer getMinTimeOfControl() {
		return minTimeOfControl;
	}

	public static void setMinTimeOfControl(int minTime) {
		minTimeOfControl = minTime;
	}

	public static Integer getMaxTimeOfControl() {
		return maxTimeOfControl;
```

```
	}

	public static void setMaxTimeOfControl(int maxTime) {
		maxTimeOfControl = maxTime;
	}

	public static Integer getMinTimeBetweenSpawningFans() {
		return minTimeBetweenSpawningFans;
	}

	public static void setMinTimeBetweenSpawningFans(int minTime) {
		minTimeBetweenSpawningFans = minTime;
	}

	public static Integer getMaxTimeBetweenSpawningFans() {
		return maxTimeBetweenSpawningFans;
	}

	public static void setMaxTimeBetweenSpawningFans(int maxTime) {
		maxTimeBetweenSpawningFans = maxTime;
	}

	private void loadImages() {
		try {
			img = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/stadion.jpg"));
			img2 = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/powrotDoMenu.jpg"));
			img3 = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/opcje.jpg"));
			img4 = ImageIO.read(new
File("C:/Users/Konrad/Desktop/java/Stadion/Images/restart.jpg"));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void paint(Graphics g) {
		super.paintComponent(g);
		g.drawImage(img, -600, 0, null);
		g.drawImage(img2, 110, 100, null);
		g.drawImage(img3, 110, 200, null);
		g.drawImage(img4, 110, 300, null);
		police.drawPolice(g);
		g.setFont(new Font("Tahoma", Font.PLAIN, 20));
		g.setColor(Color.YELLOW);
		g.drawString("Liczba kibiców na stadionie: " + numberOfFansInStadium, 50, 50);
		g.drawString("Maksymalna liczba kibiców", 50, 400);
		g.drawString("na stadionie: " + maxNumberOfFansInStadium, 85, 450);
		for(int i=0; i<QueueManagement.fanQ.size(); i++) {
			QueueManagement.fanQ.get(i).drawFan(g);
		}
		for(int i=0; i<QueueManagement.control1.size(); i++) {
			QueueManagement.control1.get(i).drawFan(g);
		}
		for(int i=0; i<QueueManagement.control2.size(); i++) {
			QueueManagement.control2.get(i).drawFan(g);
		}
		for(int i=0; i<QueueManagement.control3.size(); i++) {
			QueueManagement.control3.get(i).drawFan(g);
		}
		for(int i=0; i<QueueManagement.tempQ.size(); i++) {
			QueueManagement.tempQ.get(i).drawFan(g);
		}
	}

	public void move() {
		for(int i=0; i<QueueManagement.fanQ.size(); i++) {
			QueueManagement.fanQ.get(i).move();
		}
		for(int i=0; i<QueueManagement.control1.size(); i++) {
			QueueManagement.control1.get(i).move();
		}
		for(int i=0; i<QueueManagement.control2.size(); i++) {
			QueueManagement.control2.get(i).move();
		}
		for(int i=0; i<QueueManagement.control3.size(); i++) {
```

```
                        QueueManagement.control3.get(i).move();
                }
                for(int i=0; i<QueueManagement.tempQ.size(); i++) {
                        QueueManagement.tempQ.get(i).move();
                }
        }

        public static void reset() {
                isSimulationGoing = false;
                QueueManagement.fanQ.clear();
                QueueManagement.control1.clear();
                QueueManagement.control2.clear();
                QueueManagement.control3.clear();
                QueueManagement.tempQ.clear();
                QueueManagement.setEndOfTempQX(550);
                QueueManagement.setEndOfTempQY(350);
                QueueManagement.setEndOfQX(600);
                QueueManagement.setEndOfQY(250);
                setNumberOfFansInStadium(0);
        }

        private void loadButtons() {
                int buttonWidth = 150;
                int buttonHeight = 30;
                menu = new JButton("Powrót do menu");
                menu.setVisible(false);
                menu.setBounds(110, 100, buttonWidth, buttonHeight);
                menu.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                MainStadion.setMenuPoint(true);
                                reset();
                                unseenButtons();
                                MainStadion.getTimer().stop();
                        }
                });
                opcje = new JButton("Opcje");
                opcje.setVisible(false);
                opcje.setBounds(110, 200, buttonWidth, buttonHeight);
                opcje.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                MainStadion.setOptionsPoint(true);
                                reset();
                                unseenButtons();
                                MainStadion.getTimer().stop();
                        }
                });
                reset = new JButton("Restart");
                reset.setVisible(false);
                reset.setBounds(110, 300, buttonWidth, buttonHeight);
                reset.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                reset();
                                isSimulationGoing = true;
                        }
                });
                MainStadion.getWindow().add(menu);
                MainStadion.getWindow().add(opcje);
                MainStadion.getWindow().add(reset);
        }

        public void seenButtons() {
                menu.setVisible(true);
                opcje.setVisible(true);
                reset.setVisible(true);
        }

        private void unseenButtons() {
                menu.setVisible(false);
                opcje.setVisible(false);
                reset.setVisible(false);
        }

}
```