Concurrent Programming

Name: Danny Rene Jensen
D1
Supervisor: Peter Schneider-Kamp
Teaching assistant: Christian Damsgaard Jørgensen
DM519

12. april 2015

Indhold

1 Modelling

The first thing to do was to create an elevator design with seperate modules. The buttons to call the elevator was a good starting point. It would be easier to test if the elevator could move, as the first thing. To do that the elevator itself, with all the buttons, was needed. All the buttons are created the same way without any restrictions so the elevator can move freely and take as many agents in and even let more agents out than actually entered. The in and out buttons have a variable f on them, this variable is the current floor the eleavator is standing on. It is designed like this to prevent the agents to be able to exit on other floors than the elevator is on. the [g:FLOORS] on both the call buttons and the inside buttons are designed so the buttons for all floors are active and when, one of them is pressed the floors number is stored in g and given the next elevator runthrough, for the in and out buttons to use.

```
ELEVATOR = ELEVATOR[G],
ELEVATOR[f:FLOORS] =
       ([f].in -> ELEVATOR[f] |
       [f].out -> ELEVATOR[f] |
       [g:FLOORS].callButton -> ELEVATOR[g] |
       [r:FLOORS].insideButton -> ELEVATOR[r]).
```

After this the buttons on the elevator should work so the passengers could get in and out on all the floors. when that is possible the in and out funktions needs to be restricted to only letting 1 person in and out. it would be pretty bad if an unlimited number of people could get in but only 1 is allowed out again, and vice versa if only 1 enters and passengers just keeps exitting the elevator. (The elevator is for AREA 51 but it is doubtfull they want to clone their workers.) All these restrictions happens in AREA_ELEVATOR, and is based on the number of agents in the elevator. The assumption is taken that the elevator is not big enough to hold more than one agent at a time, that means CAPACITY is 1. When the elevator is emty i becomes 0. When i is smaller than CAPASITY two things happens here, first thing is all the in buttons can be activated this can't happen if i is 1. Second thing that happens is all the call buttons are active only when i is 0. Else if i is 1 the out buttons, inside buttons and the scan buttons can be active, of cause if they are resticted in other places all the criterias needs to be fullfiled for them to be active.

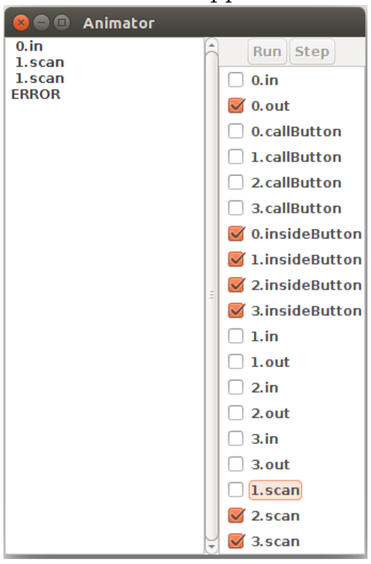
```
AREA_ELEVATOR = NORMAL[0],
NORMAL[i:NUMBER] = (when (i < CAPACITY) [G..T2].in -> NORMAL[i+1] |
              when (i < CAPACITY) [FLOORS].callButton -> NORMAL[i] |
              when (i == CAPACITY) [G..T2].out -> NORMAL[i-1]|
              when (i == CAPACITY) [FLOORS].insideButton-> NORMAL[i] |
              when (i == CAPACITY) [LEVEL].scan -> NORMAL[i]).
```

When the restrictions are done it still needs buttons on the inside of the elevator, these were created in the ELEVATOR. And of cause the retina scanner.
The retina scanner is set up so the passenger needs to scan his or her eye to even get the possibility to go to the other floors. the restriction on the out fuction, will only let the agent out on the floor that is less or equal to the permitted floors. The agent can't leave the elevator before he has gotten a clearence level.

Here out is restricted yet again that means that both this and the previous reqirements needs to be fullfilled, to let an out button become active. Here the requirement is that the scan needs to be of a certain number or higher to activate the out button.

```
FLOORCHECK = FLOORCHECK[1],
FLOORCHECK[d:LEVEL] = ([h:LEVEL].scan -> SCAN[h]),
SCAN = SCAN[1],
SCAN[i:LEVEL] = (when (i > 0) [0].out -> FLOORCHECK[i] |
                 when (i > 1) [1].out -> FLOORCHECK[i] |
                 when (i > 2) [2..3].out -> FLOORCHECK[i]).
```

If both the AREA_ELEVATOR end FLOORCHECK requirements are put together with the ELEVATOR, then the out buttons can only be activated on the floor the elevator is on, only if there is an agent inside and he has the right securety clearence.
The securety property is just a copy of FLOORCHECK with the name altered to NO_SEC. Though the property lets the agent get scanned again, it creates an error if this happens. as seen below.

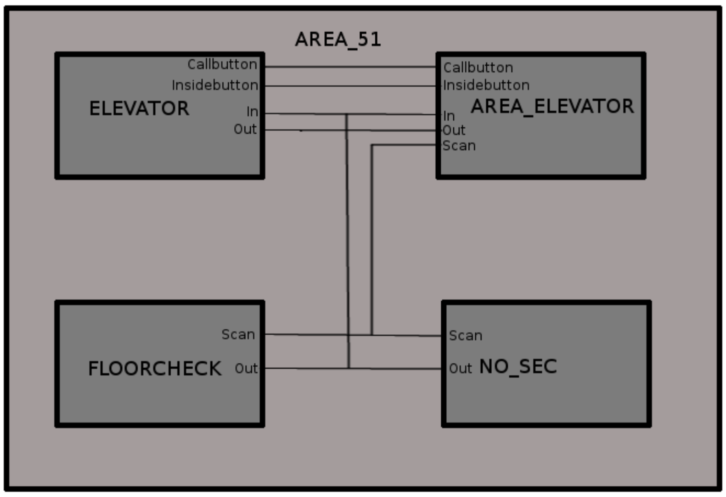

Animator

```
0.in
1.scan
1.scan
ERROR
```

Run Step

- [ ] 0.in
- [✓] 0.out
- [ ] 0.callButton
- [ ] 1.callButton
- [ ] 2.callButton
- [ ] 3.callButton
- [✓] 0.insideButton
- [✓] 1.insideButton
- [✓] 2.insideButton
- [✓] 3.insideButton
- [ ] 1.in
- [ ] 1.out
- [ ] 2.in
- [ ] 2.out
- [ ] 3.in
- [ ] 3.out
- [ ] 1.scan
- [✓] 2.scan
- [✓] 3.scan

1.1 LTSA model diagram



2 Analysis

to prevent deadlocks the assumption is taken that the elevator is not big enough to hold more than one agent at a time. This assumtoin helps to avoid a deadlock where two agents enter the elevator that have different clearence levels. By only having one agent in the elevator at a time there are no chance of any resources being taken by another method or prgram, because there is only 1 agent prgram running at a time. Another deadlock could happen if an agent, or some other person, where to enter that does not have a clearence level at all. But the securety at AREA 51 should be so tight that no outsider would have access to the elevator, and even if he did the elevator wouldn't move. To avoid a deadlock where an agent whith a low clearnce level is trying to enter to high securety floor he is not allowed out but can ride the elevator to all the other floors until he is able to, and chooses, to leave.

3 Implementation

First off the elevator should be created so it could move with an agent inside. After this the elevator needs some restrictions, like only 1 agent would be allowed in the elevator at a time.
To make sure an agent with to low clearence level wouldn't be able to exit the wrong floor, TjeckClerance takes their level and returns the number on which floor they have the maximum clearnce for. Then in FloorCheck the max floor is used to check against the floor the agent wants to leave on, if it is lover or equal it return true, letting the agent know the doors are open and he is free to leave.

```java
public int TjeckClerance(int level) {
	if (level == 1){
		return 0;
	}else if(level == 2) {
		return 1;
	}else {
		return 3;
	}
}
public boolean FloorCheck(int level, int floor) {
	if (TjeckClerance(level) >= floor) {
		return true;
	}else{
		return false;
	}
}
```

In this elevator there are 4 differnt agent threads, 2 are specefecly designed to move in a predifined pattern. The 3rd is a random agent he can spawn on any floor, have any clearence level and he also wants to go to a random floor. If he can't get out on that floor he tries another random floor until he gets out. This random agent is created by using the random numbergenerator functions in java.

```java
public int randClerance() {
	Random rn = new Random();
	int num = rn.nextInt(3) + 1;
	return num;
}
```

The last agent is a user created agent, the user can define this agent with all the criteria.
The monitor a.k. GUI is starting the agents by clicking the buttons. The GUI can handle all 4 agents at a time. The elevator.java is just the one that sets restrictions to the agents. It looks if they can enter or not, this enter function has a random time before checking if the elevator is empty which prevents multiple agents to enter at a time. this random time is only 10 milliseconds but is eneough time to make the chacne of two or more agents enter at the same time allmost impossible.

```java
public boolean EnterElevator() {
        Random rn = new Random();
        int time = rn.nextInt(10) + 1;
        try {
                Thread.sleep(time);
        } catch(InterruptedException ex) {
                Thread.currentThread().interrupt();
        }
        if (inElevator == 0){
                return true;
        }else{
                return false;
        }
}
```

The Elevator.java also stores all the info on the agents, like their clearence level, the start floor and the floor the agent wants to go to.
All agents has an if satament that looks if it is possible to exit the elevator on the desired floor, if not a new floor is selected.

```java
if (!elevator.FloorCheck(elevator.level, elevator.floor)){
        elevator.SelectedFloor(0, 1);
}
```

The last int, in this case 1, in elevator.SelectedFloor is the umber on the agent and is only used to update the right image in the GUI. these images are there as a visual representation for the user to see what happens and would not be in a real elevator program. The elevator and floors are also represented by images and the elevator will also be updated as the agent chooses which floor he starts on and which floor he wants to leave. To keep track of the agent inside the elevator, e.g. which clearence level he has and if he is allowed to leave, there are three textfields in the upper right corner. These textfield shows, from top, the clearnce level, if the agent enters or exit if and he cant, e.g. if an agent with confidential clearence level tries to exit on secret floor 2, the experimental wepons floor, (where they are trying to create their prototype nr 42 which is the point of view gun) the label will say "no access". The last displays on which floor the elevator curently is.
The sleep function is also only for show. All it does is slow down the movements of the agents so it is possible to follow them on the screen. Without this sleep function the agents would move allmost instantly around and it would be impossible for any human to check if the program is working like it should or not.

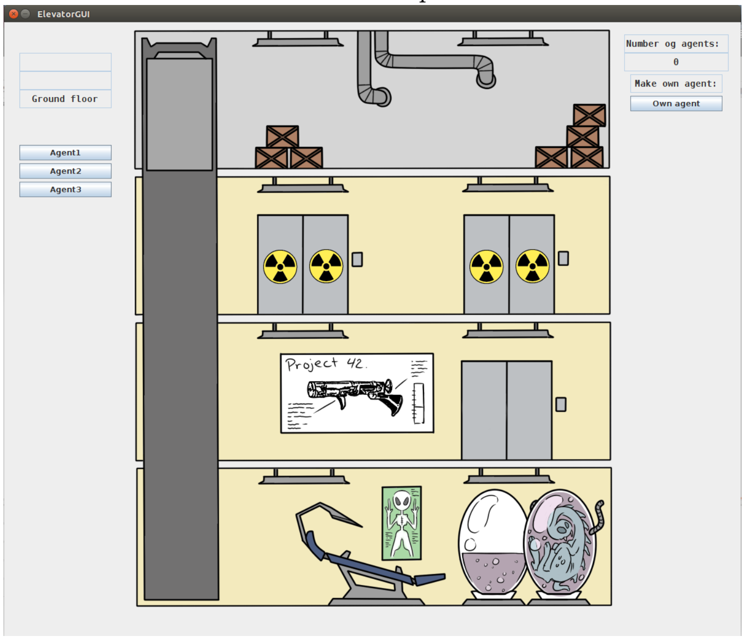
```java
public void sleep(){
        try {
                Thread.sleep(1000);
        } catch(InterruptedException ex) {
                Thread.currentThread().interrupt();
        }
}
```

ElevatorRun in the GUI is the method that performes the movements and chechks of the agent, after he entered the elevator. It moves the agent and the elevator to the desired floor. After this it checks if the agent has the possibility to exit the elevator at that floor, if not it writes this to a label. The else statement is designed to run through, if the agent can exit at the desired floor. The agent is then moved out of the elevator, and the label says that the agent is exitting. When this is done, it resets some variables like the elevator.InElevator. the InElevator variable is the variable that keeps track of if the elevator is empty or not.

```
public void ElevatorRun(int agent) {
        if (!elevator.FloorCheck(elevator.level, elevator.floor)){
                FloorLevel(elevator.floor);
                agentEntered(agentNum(elevator.agentNr), elevator.floor);
                sleep();
                 enterExitlabel.setText("no access");
                sleep();
                enterExitlabel.setText(" ");
        }else{
                FloorLevel(elevator.floor);
                agentEntered(agentNum(elevator.agentNr), elevator.floor);
                sleep();
                enterExitlabel.setText("exit");
                sleep();
                agentExit(agentNum(elevator.agentNr), elevator.floor);
                sleep();
                enterExitlabel.setText(" ");
                clearencelabel.setText(" ");
                agentNum(elevator.agentNr).setVisible(false);
                agentsIn -= 1;
                agentnumlabel.setText(agentsIn + "");
                if (agent == 1) {
                        agentcheck1 = 0;
                }else if (agent == 2) {
                        agentcheck2 = 0;
                }else if (agent == 3) {
                        agentcheck3 = 0;
                }
                elevator.inElevator = 0;
        }
}
```

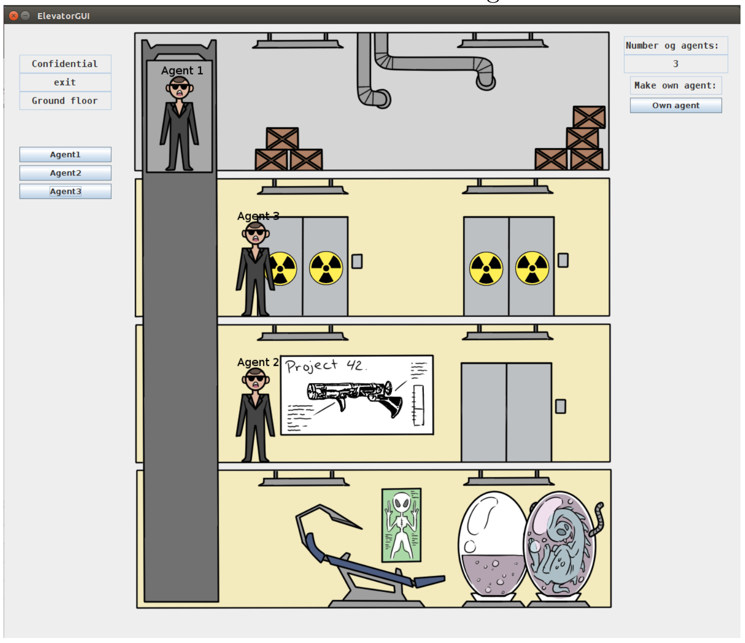
4 Testing

First off to test if the window can open:



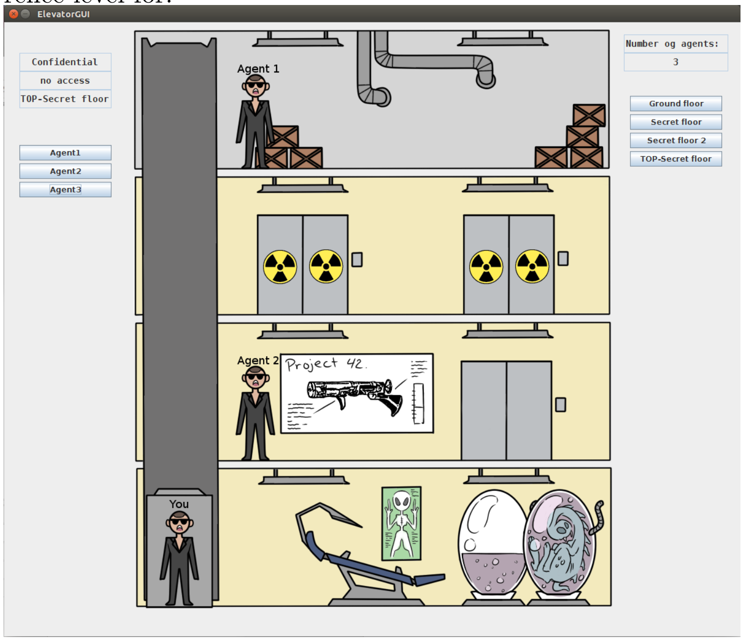
It opens the window nicely and displays the floors, buttons, labels and the elevator as it should.

The next test is to see if more than one agent at a time enters the elevator:

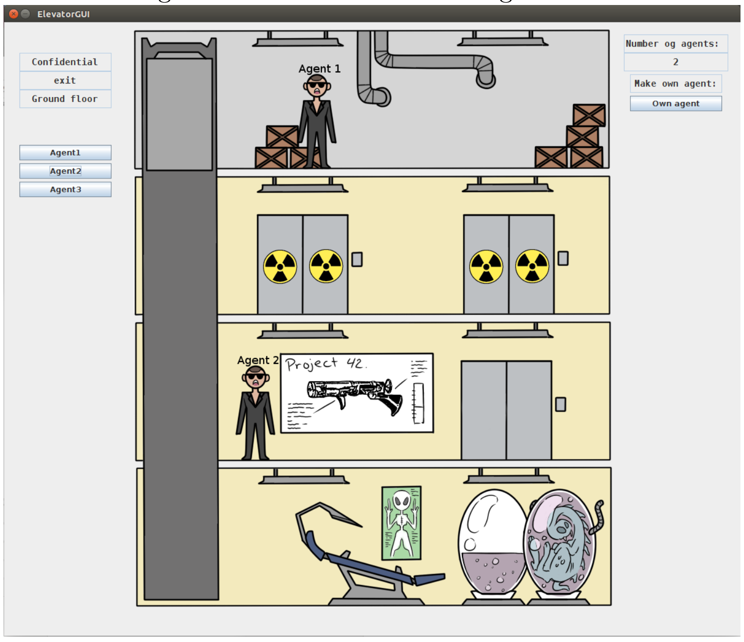


This looks good too, agent 2 and 3 are waiting for agent 1 to exit the elevator befor using it, and thanks to the random time, on the check to see if it is empty, the agents wont try and ride it at the same time.

What will happen if an agent tries to exit a floor he dosn't have the right clearence level for:



The label will display "no access"and not let the agent exit on that floor. now can an agent leave the elevator at a right floor:



It looks like agnet 1 was able to leave the elevator at the floor he has permission to leave. The top right label is displaying that he has a clearnece level of Confidential, which means he is only allowed to leave on the ground floor.

5 Appendix (source code)

5.1 LTSA code

```
const G = 0
const T2 = 3
const CAPACITY = 1
range LEVEL = 1..3
range FLOORS = G..T2
range NUMBER = 0..1

FLOORCHECK = FLOORCHECK[1],
FLOORCHECK[d:LEVEL] = ([h:LEVEL].scan -> SCAN[h]),
SCAN = SCAN[1],
SCAN[i:LEVEL] = (when (i > 0) [0].out -> FLOORCHECK[i] |
                             when (i > 1) [1].out -> FLOORCHECK[i] |
                             when (i > 2) [2..3].out -> FLOORCHECK[i]).

property NO_SEC = NO_SEC[1],
NO_SEC[d:LEVEL] = ([h:LEVEL].scan -> SCAN[h]),
SCAN = SCAN[1],
SCAN[i:LEVEL] = (when (i > 0) [0].out -> NO_SEC[i] |
                             when (i > 1) [1].out -> NO_SEC[i] |
                             when (i > 2) [2..3].out -> NO_SEC[i]).

ELEVATOR = ELEVATOR[G],
ELEVATOR[f:FLOORS] =
        ([f].in -> ELEVATOR[f] |
        [f].out -> ELEVATOR[f] |
        [g:FLOORS].callButton -> ELEVATOR[g] |
        [r:FLOORS].insideButton -> ELEVATOR[r]).

AREA_ELEVATOR = NORMAL[0],
NORMAL[i:NUMBER] = (when (i < CAPACITY) [G..T2].in -> NORMAL[i+1] |
                                    when (i < CAPACITY) [FLOORS].
                                        callButton -> NORMAL[i] |
                  when (i == CAPACITY) [G..T2].out -> NORMAL[i-1]|
                                    when (i == CAPACITY) [FLOORS].
                                        insideButton-> NORMAL[i] |
                                    when (i == CAPACITY) [LEVEL].scan ->
                                        NORMAL[i]).

||AREA_51 = (ELEVATOR || AREA_ELEVATOR || FLOORCHECK || NO_SEC).
```

5.2 Elevator.java

```java
import java.util.*;

public class Elevator {
       ElevatorGUI gui;
       public int start;
```

```
        public int level;
        public int floor;
        public int inElevator = 0;
        public int okToExit;
        public int agentNr;

        public void setgui(ElevatorGUI gui){
                this.gui = gui;
        }

        public void startFloor(int startfloor, int agentNr) {
                gui.ElevatorPre(agentNr, startfloor);
        }

        public void floorstart(int startfloor) {
                start = startfloor;
        }

        public boolean EnterElevator() {
                Random rn = new Random();
                int time = rn.nextInt(10) + 1;
                try {
                        Thread.sleep(time);
                } catch(InterruptedException ex) {
                        Thread.currentThread().interrupt();
                }
                if (inElevator == 0){
                        return true;
                }else{
                        return false;
                }
        }

        public void ChooseClearence(int clearenceLevel, int agentnr){
                level = clearenceLevel;
                agentNr = agentnr;
                gui.ElevatorEnter(agentNr);
        }

        public void SelectedFloor(int floorSelected, int agent) {
                floor = floorSelected;
                if (inElevator != 0 && level != 0){
                        gui.ElevatorRun(agent);
                }
        }

        public int TjeckClerance(int level) {
                if (level == 1){
                        return 0;
                }else if(level == 2) {
                        return 1;
                }else {
                        return 3;
                }
```

```java
	}

	public boolean FloorCheck(int level, int floor) {
		if (TjeckClerance(level) >= floor) {
			return true;
		}else{
			return false;
		}
	}

	public void ExitOk(int possible) {
			okToExit = possible;
	}
}
```

5.3 ElevatorGUI.java

```java
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.IOException;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class ElevatorGUI extends JPanel{
	private JFrame Frame;
	private JTextField floorlabel = new JTextField();
	private JTextField clearencelabel = new JTextField();
	private JTextField enterExitlabel = new JTextField();
	private JTextField agentlabel = new JTextField();
	private JTextField agentnumlabel = new JTextField();
	private JTextField useragentlabel = new JTextField();
	Elevator elevator;
	private int check = 0;
	private int agentsIn = 0;
	private int agentcheck1 = 0;
	private int agentcheck2 = 0;
	private int agentcheck3 = 0;
	Agent0 agent0;
	Agent1 agent1;
	Agent2 agent2;
	Agent3 agent3;

	JLabel floorsPic = new JLabel(new ImageIcon("floors.png"));
	JLabel elevatorPic = new JLabel(new ImageIcon("elevator.png"));
	JLabel agent0Pic = new JLabel(new ImageIcon("agent0.png"));
	JLabel agent1Pic = new JLabel(new ImageIcon("agent1.png"));
	JLabel agent2Pic = new JLabel(new ImageIcon("agent2.png"));
	JLabel agent3Pic = new JLabel(new ImageIcon("agent3.png"));

	JButton agent1Btn = new JButton("Agent1");
```

```
        JButton agent2Btn = new JButton("Agent2");
        JButton agent3Btn = new JButton("Agent3");

        JButton userAgentBtn = new JButton("Own agent");
        JButton gBtn = new JButton("Ground floor");
        JButton sfBtn = new JButton("Secret floor");
        JButton t1Btn = new JButton("Secret floor 2");
        JButton t2Btn = new JButton("TOP-Secret floor");
        JButton cBtn = new JButton("Confidential");
        JButton sBtn = new JButton("Secret");
        JButton tBtn = new JButton("TOP-Secret");
        JButton rBtn = new JButton("Run agent");

    public ElevatorGUI(Elevator elevator){
                this.elevator = elevator;
        PrepareGUI();
        display();
    }

    public void setagent0(Agent0 agent0){
                this.agent0 = agent0;
                }
    public void setagent1(Agent1 agent1){
                this.agent1 = agent1;
                }
        public void setagent2(Agent2 agent2){
                this.agent2 = agent2;
                }
        public void setagent3(Agent3 agent3){
                this.agent3 = agent3;
                }

        private void PrepareGUI(){
                Frame = new JFrame("ElevatorGUI");
                Frame.setResizable(false);
                Frame.setLayout(null);

                Insets insets = Frame.getInsets();
        Frame.setSize(1200 + insets.left + insets.right,
                      1000+ insets.top + insets.bottom);

                floorlabel.setHorizontalAlignment(JTextField.CENTER);
                floorlabel.setFont(new Font("Monospaced", Font.BOLD, 15));
                floorlabel.setEditable(false);
                floorlabel.setText("Ground floor");

                clearencelabel.setHorizontalAlignment(JTextField.CENTER);
                clearencelabel.setFont(new Font("Monospaced", Font.BOLD,
                    15));
                clearencelabel.setEditable(false);

                enterExitlabel.setHorizontalAlignment(JTextField.CENTER);
                enterExitlabel.setFont(new Font("Monospaced", Font.BOLD,
                    15));
```

```
            enterExitlabel.setEditable(false);

            agentlabel.setHorizontalAlignment(JTextField.CENTER);
            agentlabel.setFont(new Font("Monospaced", Font.BOLD, 15));
            agentlabel.setEditable(false);

            agentnumlabel.setHorizontalAlignment(JTextField.CENTER);
            agentnumlabel.setFont(new Font("Monospaced", Font.BOLD,
                15));
            agentnumlabel.setEditable(false);

            useragentlabel.setHorizontalAlignment(JTextField.CENTER);
            useragentlabel.setFont(new Font("Monospaced", Font.BOLD,
                15));
            useragentlabel.setEditable(false);

            Frame.addWindowListener(new WindowAdapter() {
                    public void windowClosing(WindowEvent windowEvent){
                            System.exit(0);
                    }
            });

            Frame.setVisible(true);
    }

    private void display(){

            Agent1Event Agent1BtnEvent = new Agent1Event();
            agent1Btn.addActionListener(Agent1BtnEvent);

            Agent2Event Agent2BtnEvent = new Agent2Event();
            agent2Btn.addActionListener(Agent2BtnEvent);

            Agent3Event Agent3BtnEvent = new Agent3Event();
            agent3Btn.addActionListener(Agent3BtnEvent);

            userAgentEvent userAgentBtnEvent = new userAgentEvent();
            userAgentBtn.addActionListener(userAgentBtnEvent);

            gEvent gBtnEvent = new gEvent();
            gBtn.addActionListener(gBtnEvent);

            sfEvent sfBtnEvent = new sfEvent();
            sfBtn.addActionListener(sfBtnEvent);

            t1Event t1BtnEvent = new t1Event();
            t1Btn.addActionListener(t1BtnEvent);

            t2Event t2BtnEvent = new t2Event();
            t2Btn.addActionListener(t2BtnEvent);

            cEvent cBtnEvent = new cEvent();
            cBtn.addActionListener(cBtnEvent);

```

```java
        sEvent sBtnEvent = new sEvent();
        sBtn.addActionListener(sBtnEvent);

        tEvent tBtnEvent = new tEvent();
        tBtn.addActionListener(tBtnEvent);

        rEvent rBtnEvent = new rEvent();
        rBtn.addActionListener(rBtnEvent);

        Frame.add(agentlabel);
        Frame.add(agentnumlabel);
        Frame.add(floorlabel);
        Frame.add(clearencelabel);
        Frame.add(enterExitlabel);
        Frame.add(useragentlabel);
        Frame.add(agent1Btn);
        Frame.add(agent2Btn);
        Frame.add(agent3Btn);

        Frame.add(agent0Pic);
        Frame.add(agent1Pic);
        Frame.add(agent2Pic);
        Frame.add(agent3Pic);
        Frame.add(elevatorPic);
        Frame.add(floorsPic);

        Frame.add(userAgentBtn);
        Frame.add(gBtn);
        Frame.add(sfBtn);
        Frame.add(t1Btn);
        Frame.add(t2Btn);
        Frame.add(cBtn);
        Frame.add(sBtn);
        Frame.add(tBtn);
        Frame.add(rBtn);

        useragentlabel.setText("Make own agent:");
        agentlabel.setText("Number og agents: ");
        agentnumlabel.setText("" + agentsIn);

        agent0Pic.setVisible(false);
        agent1Pic.setVisible(false);
        agent2Pic.setVisible(false);
        agent3Pic.setVisible(false);

        gBtn.setVisible(false);
        sfBtn.setVisible(false);
        t1Btn.setVisible(false);
        t2Btn.setVisible(false);
        cBtn.setVisible(false);
        sBtn.setVisible(false);
        tBtn.setVisible(false);
        rBtn.setVisible(false);

```

```
                Insets insets = Frame.getInsets();
        //label:
        agentlabel.setBounds(1010 + insets.left, 20 + insets.top, 170,
            30);
        agentnumlabel.setBounds(1010 + insets.left, 50 + insets.top, 170,
             30);
        clearencelabel.setBounds(25 + insets.left, 50 + insets.top, 150,
            30);
        enterExitlabel.setBounds(25 + insets.left, 80 + insets.top, 150,
            30);
        floorlabel.setBounds(25 + insets.left, 110 + insets.top, 150, 30)
            ;
                //agent buttons:
        agent1Btn.setBounds(25 + insets.left, 200 + insets.top, 150, 25);
        agent2Btn.setBounds(25 + insets.left, 230 + insets.top, 150, 25);
        agent3Btn.setBounds(25 + insets.left, 260 + insets.top, 150, 25);
        elevatorPic.setBounds(226 + insets.left, 44 + insets.top, 119,
            203);
        floorsPic.setBounds(200 + insets.left, 10 + insets.top,800, 950);

        useragentlabel.setBounds(1020 + insets.left, 85 + insets.top,
            150, 30);
        userAgentBtn.setBounds(1020 + insets.left, 120 + insets.top, 150,
             25);
        gBtn.setBounds(1020 + insets.left, 120 + insets.top, 150, 25);
        sfBtn.setBounds(1020 + insets.left, 150 + insets.top, 150, 25);
        t1Btn.setBounds(1020 + insets.left, 180 + insets.top, 150, 25);
        t2Btn.setBounds(1020 + insets.left, 210 + insets.top, 150, 25);
        cBtn.setBounds(1020 + insets.left, 120 + insets.top, 150, 25);
        sBtn.setBounds(1020 + insets.left, 150 + insets.top, 150, 25);
        tBtn.setBounds(1020 + insets.left, 180 + insets.top, 150, 25);
        rBtn.setBounds(1020 + insets.left, 120 + insets.top, 150, 25);
        }

        public class Agent1Event implements ActionListener{
                public void actionPerformed(ActionEvent Agent1BtnEvent){
                        System.out.println("1: " + agentcheck1);
                        if (agentcheck1 == 0) {
                                Thread agent01 = new Thread(agent1);
                                agent01.start();
                                agentcheck1 += 1;
                        }
                }
        }

        public class Agent2Event implements ActionListener{
                public void actionPerformed(ActionEvent Agent2BtnEvent){
                        System.out.println("2: " + agentcheck2);
                        if (agentcheck2 == 0) {
                                Thread agent02 = new Thread(agent2);
                                agent02.start();
                                agentcheck2 += 1;
                        }
                }
```

```
	}

	public class Agent3Event implements ActionListener{
		public void actionPerformed(ActionEvent Agent3BtnEvent){
			System.out.println("3: " + agentcheck3);
			if (agentcheck3 == 0) {
				Thread agent03 = new Thread(agent3);
				agent03.start();
				agentcheck3 += 1;
			}
		}
	}

	public class userAgentEvent implements ActionListener{
		public void actionPerformed(ActionEvent userAgentBtnEvent)
		    {
				gBtn.setVisible(true);
				sfBtn.setVisible(true);
				t1Btn.setVisible(true);
				t2Btn.setVisible(true);
				userAgentBtn.setVisible(false);
				useragentlabel.setVisible(false);
		}
	}

	public class gEvent implements ActionListener{
		public void actionPerformed(ActionEvent gBtnEvent){
			if(check == 0) {
				agent0.startfloor = 0;
				check = 1;
				cBtn.setVisible(true);
				sBtn.setVisible(true);
				tBtn.setVisible(true);
				gBtn.setVisible(false);
				sfBtn.setVisible(false);
				t1Btn.setVisible(false);
				t2Btn.setVisible(false);
			}else if (check == 2){
				agent0.selectedfloor = 0;
			} else {
				agent0.selectedfloor = 0;
				rBtn.setVisible(true);
				gBtn.setVisible(false);
				sfBtn.setVisible(false);
				t1Btn.setVisible(false);
				t2Btn.setVisible(false);
			}
		}
	}

	public class sfEvent implements ActionListener{
		public void actionPerformed(ActionEvent sfBtnEvent){
			if(check == 0) {
				agent0.startfloor = 1;
```

```
                        check = 1;
                        cBtn.setVisible(true);
                        sBtn.setVisible(true);
                        tBtn.setVisible(true);
                        gBtn.setVisible(false);
                        sfBtn.setVisible(false);
                        t1Btn.setVisible(false);
                        t2Btn.setVisible(false);
                }else if (check == 2){
                        agent0.selectedfloor = 1;
                } else {
                        agent0.selectedfloor = 1;
                        rBtn.setVisible(true);
                        gBtn.setVisible(false);
                        sfBtn.setVisible(false);
                        t1Btn.setVisible(false);
                        t2Btn.setVisible(false);
                }
        }
}

public class t1Event implements ActionListener{
        public void actionPerformed(ActionEvent t1BtnEvent){
                if(check == 0) {
                        agent0.startfloor = 2;
                        check = 1;
                        cBtn.setVisible(true);
                        sBtn.setVisible(true);
                        tBtn.setVisible(true);
                        gBtn.setVisible(false);
                        sfBtn.setVisible(false);
                        t1Btn.setVisible(false);
                        t2Btn.setVisible(false);
                }else if (check == 2){
                        agent0.selectedfloor = 2;
                } else {
                        agent0.selectedfloor = 2;
                        rBtn.setVisible(true);
                        gBtn.setVisible(false);
                        sfBtn.setVisible(false);
                        t1Btn.setVisible(false);
                        t2Btn.setVisible(false);
                }
        }
}

public class t2Event implements ActionListener{
        public void actionPerformed(ActionEvent t2BtnEvent){
                if(check == 0) {
                        agent0.startfloor = 3;
                        check = 1;
                        cBtn.setVisible(true);
                        sBtn.setVisible(true);
                        tBtn.setVisible(true);
```

```java
					gBtn.setVisible(false);
					sfBtn.setVisible(false);
					t1Btn.setVisible(false);
					t2Btn.setVisible(false);
				}else if (check == 2){
					agent0.selectedfloor = 3;
				} else {
					agent0.selectedfloor = 3;
					rBtn.setVisible(true);
					gBtn.setVisible(false);
					sfBtn.setVisible(false);
					t1Btn.setVisible(false);
					t2Btn.setVisible(false);
				}
			}
		}

		public class cEvent implements ActionListener{
			public void actionPerformed(ActionEvent cBtnEvent){
				agent0.clearence = 1;
				gBtn.setVisible(true);
				sfBtn.setVisible(true);
				t1Btn.setVisible(true);
				t2Btn.setVisible(true);
				cBtn.setVisible(false);
				sBtn.setVisible(false);
				tBtn.setVisible(false);
			}
		}

		public class sEvent implements ActionListener{
			public void actionPerformed(ActionEvent sBtnEvent){
				agent0.clearence = 2;
				gBtn.setVisible(true);
				sfBtn.setVisible(true);
				t1Btn.setVisible(true);
				t2Btn.setVisible(true);
				cBtn.setVisible(false);
				sBtn.setVisible(false);
				tBtn.setVisible(false);
			}
		}

		public class tEvent implements ActionListener{
			public void actionPerformed(ActionEvent tBtnEvent){
				agent0.clearence = 3;
				gBtn.setVisible(true);
				sfBtn.setVisible(true);
				t1Btn.setVisible(true);
				t2Btn.setVisible(true);
				cBtn.setVisible(false);
				sBtn.setVisible(false);
				tBtn.setVisible(false);
			}
```

```
	}

	public class rEvent implements ActionListener{
		public void actionPerformed(ActionEvent rBtnEvent){
			Thread agent00 = new Thread(agent0);
				agent00.start();
				rBtn.setVisible(false);
		}
	}

	public void ElevatorPre(int agentnr, int startfloor) {
		addAgent(agentNum(agentnr), startfloor);
		agentsIn += 1;
		agentnumlabel.setText(agentsIn + "");
		sleep();
	}

	public void ElevatorEnter(int agentnr) {
		FloorLevel(elevator.start);
		agentEntered(agentNum(agentnr), elevator.start);
		enterExitlabel.setText("entered");
		sleep();
		ClearenceName();
		sleep();
		}

	public void ElevatorRun(int agent) {
		FloorLevel(elevator.floor);
		agentEntered(agentNum(elevator.agentNr), elevator.floor);
		sleep();
		if (!elevator.FloorCheck(elevator.level, elevator.floor)){
			 enterExitlabel.setText("no access");
			sleep();
			enterExitlabel.setText(" ");
		}else{
			enterExitlabel.setText("exit");
			sleep();
			agentExit(agentNum(elevator.agentNr), elevator.
			    floor);
			sleep();
			enterExitlabel.setText(" ");
			clearencelabel.setText(" ");
			agentNum(elevator.agentNr).setVisible(false);
			agentsIn -= 1;
			agentnumlabel.setText(agentsIn + "");
			if (agent == 1) {
				agentcheck1 = 0;
			}else if (agent == 2) {
				agentcheck2 = 0;
			}else if (agent == 3) {
				agentcheck3 = 0;
			}
			elevator.inElevator = 0;
		}
```

```java
	}

	public void AgentNewFloor() {
		check = 2;
		gBtn.setVisible(true);
		sfBtn.setVisible(true);
		t1Btn.setVisible(true);
		t2Btn.setVisible(true);

	}

	public void NewUserAgent() {
		check = 0;
		useragentlabel.setVisible(true);
		userAgentBtn.setVisible(true);
		gBtn.setVisible(false);
		sfBtn.setVisible(false);
		t1Btn.setVisible(false);
		t2Btn.setVisible(false);
	}

	public void FloorLevel(int floor) {
		Insets insets = Frame.getInsets();
		if (floor == 0){
			floorlabel.setText("Ground floor");
			elevatorPic.setBounds(226 + insets.left, 44 +
			    insets.top, 119, 203);
		}else if (floor == 1){
			floorlabel.setText("Secret floor");
			elevatorPic.setBounds(226 + insets.left, 281 +
			    insets.top, 119, 203);
		}else if (floor == 2){
			floorlabel.setText("Secret floor 2");
			elevatorPic.setBounds(226 + insets.left, 519 +
			    insets.top, 119, 203);
		}else{
			floorlabel.setText("TOP-Secret floor");
			elevatorPic.setBounds(226 + insets.left, 755 +
			    insets.top, 119, 203);
		}
	}

	public void ClearenceName()  {
		if (elevator.level == 1){
			clearencelabel.setText("Confidential");
		}else if (elevator.level == 2){
			clearencelabel.setText("Secret");
		}else{
			clearencelabel.setText("TOP-Secret");
		}
	}

	public JLabel agentNum(int agentnr) {
		if (agentnr == 0) {
```

```java
            agent0Pic.setVisible(true);
            return agent0Pic;
        }else if (agentnr == 1) {
            agent1Pic.setVisible(true);
            return agent1Pic;
        }else if (agentnr == 2) {
            agent2Pic.setVisible(true);
            return agent2Pic;
        }else if (agentnr == 3) {
            agent3Pic.setVisible(true);
            return agent3Pic;
        }else{
            return null;
        }
    }

    public void addAgent(JLabel agent, int floors) {
        Insets insets = Frame.getInsets();
        if (floors == 0) {
            agent.setBounds(350 + insets.left, 44 + insets.top, 119, 203);
        }else if (floors == 1) {
            agent.setBounds(350 + insets.left, 281 + insets.top, 119, 203);
        }else if (floors == 2) {
            agent.setBounds(350 + insets.left, 519 + insets.top, 119, 203);
        }else if (floors == 3) {
            agent.setBounds(350 + insets.left, 755 + insets.top, 119, 203);
        }
    }

    public void agentEntered(JLabel agent, int floor) {
        Insets insets = Frame.getInsets();
        if (floor == 0){
            agent.setBounds(226 + insets.left, 44 + insets.top, 119, 203);
        }else if (floor == 1){
            agent.setBounds(226 + insets.left, 281 + insets.top, 119, 203);
        }else if (floor == 2){
            agent.setBounds(226 + insets.left, 519 + insets.top, 119, 203);
        }else{
            agent.setBounds(226 + insets.left, 755 + insets.top, 119, 203);
        }
    }

    public void agentExit(JLabel agent, int floors) {
        Insets insets = Frame.getInsets();
        if (floors == 0) {
```

```
                        agent.setBounds(450 + insets.left, 44 + insets.top,
                             119, 203);
                }else if (floors == 1) {
                        agent.setBounds(450 + insets.left, 281 + insets.top
                            , 119, 203);
                }else if (floors == 2) {
                        agent.setBounds(450 + insets.left, 519 + insets.top
                            , 119, 203);
                }else if (floors == 3) {
                        agent.setBounds(450 + insets.left, 755 + insets.top
                            , 119, 203);
                }
        }

        public void sleep(){
                try {
                        Thread.sleep(1000);
                } catch(InterruptedException ex) {
                        Thread.currentThread().interrupt();
                }
        }
}
```

5.4 MainClass.java

```
public class MainClass{

public static void main(String[] args){

                Elevator elevator = new Elevator();
                Agent0 agent0 = new Agent0(elevator);
                Agent1 agent1 = new Agent1(elevator);
                Agent2 agent2 = new Agent2(elevator);
                Agent3 agent3 = new Agent3(elevator);
                ElevatorGUI gui = new ElevatorGUI(elevator);
                elevator.setgui(gui);
                gui.setagent0(agent0);
                gui.setagent1(agent1);
                gui.setagent2(agent2);
                gui.setagent3(agent3);
                agent0.setgui(gui);
        }
}
```

5.5 Agent0.java

```
public class Agent0 implements Runnable {
        Elevator elevator;
        ElevatorGUI gui;
        public int startfloor;
        public int clearence;
        public int selectedfloor;

        private int check = 0;
        private int check1 = 1;
        private int spawnCheck = 0;

        public Agent0(Elevator elevator) {
                this.elevator = elevator;
        }
        public void setgui(ElevatorGUI gui){
                this.gui = gui;
                }

        public void run() {
                while (check == 0){
                        if (spawnCheck == 0){
                                elevator.startFloor(startfloor, 0);
                                spawnCheck = 1;
                        }
                        if (elevator.EnterElevator()){
                                elevator.floorstart(startfloor);
                                elevator.inElevator = 1;
                                elevator.ChooseClearence(clearence, 0);
                                elevator.SelectedFloor(selectedfloor, 0);
                                System.out.println("1 running");
                                while (check1 != 0) {
                                        if (!elevator.FloorCheck(elevator.
                                            level, elevator.floor)){
                                                gui.AgentNewFloor();
                                                elevator.SelectedFloor(
                                                    selectedfloor, 1);
                                        }else {
                                                gui.NewUserAgent();
                                                check = 1;
                                                check1 = 0;
                                                spawnCheck = 0;
                                        }
                                }
                                check1= 1;
                        }else{
                                try {
                                        Thread.sleep(1000);
                                } catch(InterruptedException ex) {
                                        Thread.currentThread().interrupt();
                                }
                        }
                }
                check = 0;
    }
}
```

5.6 Agent1.java

```
public class Agent1 implements Runnable {
        Elevator elevator;
        private int check = 0;
        private int spawnCheck = 0;

        public Agent1(Elevator elevator) {
                this.elevator = elevator;
        }

        public void run() {
                while (check == 0){
                        if (spawnCheck == 0){
                                elevator.startFloor(0, 1);
                                spawnCheck = 1;
                        }
                        if (elevator.EnterElevator()){
                                elevator.floorstart(0);
                                elevator.inElevator = 1;
                                elevator.ChooseClearence(1, 1);
                                elevator.SelectedFloor(2, 1);
                                System.out.println("1 running");
                                if (!elevator.FloorCheck(elevator.level,
                                    elevator.floor)){
                                        elevator.SelectedFloor(0, 1);
                                }
                                check = 1;
                                spawnCheck = 0;
                        }else{
                                try {
                                        Thread.sleep(1000);
                                } catch(InterruptedException ex) {
                                        Thread.currentThread().interrupt();
                                }
                        }
                }
                check = 0;
    }
}
```

5.7 Agent2.java

```
public class Agent2 implements Runnable {
        Elevator elevator;
        private int check = 0;
        private int spawnCheck = 0;

        public Agent2(Elevator elevator) {
                this.elevator = elevator;
        }
```

```java

	public void run() {
		while (check == 0){
			if (spawnCheck == 0){
				elevator.startFloor(2, 2);
				spawnCheck = 1;
			}
			if (elevator.EnterElevator()){
				elevator.floorstart(2);
				elevator.inElevator = 1;
				elevator.ChooseClearence(3, 2);
				elevator.SelectedFloor(3, 2);
				System.out.println("2 running");
				if (!elevator.FloorCheck(elevator.level,
				    elevator.floor)){
					elevator.SelectedFloor(0, 2);
				}
				check = 1;
				spawnCheck = 0;
			}else{
				try {
					Thread.sleep(1000);
				} catch(InterruptedException ex) {
					Thread.currentThread().interrupt();
				}
			}
		}
		check = 0;
	}
}
```

5.8 Agent3.java

```java
import java.util.*;
public class Agent3 implements Runnable {
	Elevator elevator;
	public boolean a3;
	private int check = 0;
	private int spawnCheck = 0;
	private int startFloorLevel;

	public Agent3(Elevator elevator) {
		this.elevator = elevator;
	}

	public void run() {
		int exitCheck = 0;
		while (check == 0){
			if (spawnCheck == 0){
				startFloorLevel = randFloor();
				elevator.startFloor(startFloorLevel, 3);
				spawnCheck = 1;
```

```java

            }
            if (elevator.EnterElevator()){
                elevator.floorstart(startFloorLevel);
                elevator.inElevator = 1;
                elevator.ChooseClearence(randClerance(), 3);
                elevator.SelectedFloor(randFloor(), 2);
                System.out.println("3 running");
                check = 1;
                while(exitCheck != 1) {
                    if (!elevator.FloorCheck(elevator.level, elevator.floor)){
                        elevator.SelectedFloor(randFloor(), 3);
                    }else {
                        exitCheck = 1;
                    }
                    spawnCheck = 0;
                }
            }else{
                try {
                    Thread.sleep(1000);
                } catch(InterruptedException ex) {
                    Thread.currentThread().interrupt();
                }
            }
        }
        check = 0;
    }

    public int randClerance() {
        Random rn = new Random();
        int num = rn.nextInt(3) + 1;
        return num;
    }

    public int randFloor() {
        Random rn = new Random();
        int num = rn.nextInt(4) + 0;
        return num;
    }

}
```