Marc Ilsø Poulsen, Peter Bartram Hansen og Bjarne Antonsen 12/1-2011
dLego: Projekt Session 5 Rapport
Varighed: 6 timer

Formål:

Formålet med i dag var at få vores 2 NXT enheder til at sende beskeder til hinanden, når den ene robot ramte et checkpoint, så den anden kunne modtage sin straf, hvilket var spejlvendt styring iet kort stykke tid
Derefter skulle vi have implementeret vores øverste behavior ”FinishBehavior”, som skulle supresse alle andre behaviors, når en af robotterne går hen og vinder kapløbet.

Plan:
Vores plan var først at forsøge at implementere den løsning til kommunikation imellem robotterne, som vi havde lagt op til ved sidste lab session. Denne plan gik ud på, at sende beskeder fra de to robotter igennem de to PC'er, så en robot ville have information om den anden robots tilstand i forhold til checkpoints.
Dernæst skulle vi implementere behavioren ”FinishBehavior”, og her var planen at den skulle stoppe alle andre behaviors i at gøre noget som helst, når en af robotterne nåede 3. checkpoint og derved vandt spillet.

Resultater:

Kommunikation imellem de 2 NXT enheder:

Vi gik direkte i gang med at implementere vores nye arkitektur ($NXT_1 \leftrightarrow PC_1 \leftrightarrow PC_2 \leftrightarrow NXT_2$), som viste sig at virke ret så mageløst og det blev implementeret uden nogle rigtigt store problemer. Vores største problem var, at vi ikke måtte oprette en socket connection imellem de 2 PCere over det trådløse netværk, grundet lukkede porte på CS-netværket, men generelt ville en trådløs forbindelse imellem de 2 PCere gøre arbejdet ganske glimrende. Vi omgik denne begrænsning ved at anvende et LAN-kabel imellem de to PC'er.
Vores GUI kom nu til at indeholde følgende kode, som håndterede, at der blev klikket på connect knappen:

```
if(dir.equals("connect")) {
	conn.connectTo(deviceName);
	dos = conn.getDataOut();
	dis = conn.getDataIn();

	SocketReceiveThread srt = new SocketReceiveThread(br, dos);
	Thread t1 = new Thread(srt);
	t1.setDaemon(true);
	t1.start();

	BTReceiveThread btr = new BTReceiveThread(pw, dis);
	Thread t2 = new Thread(btr);
	t2.setDaemon(true);
	t2.start();

	enableButtons();
}
```

Denne opretter en `SocketReceiveThread` som er en tråd vi bruger til at kommunikere mellem de 2 PCer og en `BTReceiveThread` som bruges til at kommunikere mellem PCen og NXT enheden via Bluetooth.
Til slut kalder vi `enableButtons()` som har til formål at aktivere knapperne til at styre NXT enheden med.
De 2 tråde gør ikke andet end at sende informationen som de får videre til den næste i rækken, som det kan f.eks. kan ses i `BTReceiveThread's run()` metode:

```
private PrintWriter output;
private DataInputStream input;

[...]

public void run() {
	while (true) {
		try {
			int message = input.readInt();
			output.println(message);
		} catch (Exception e) {}
	}
}
```

`BTReceiveThread` sørger for at input fra NXT_1 går igennem PC_1 og bliver sendt videre til PC_2. Hvorimod `SocketReceiveThread` sørger for at informationer som kommer fra PC_2 til PC_1 bliver sendt videre ned til NXT_1.

På NXT enheden ser oprettelsen af bluetooth forbindelsen således ud:

```
LCD.drawString("Waiting...",0,0);
NXTConnection conn = Bluetooth.waitForConnection();

LCD.drawString("Connected to GUI",0,0);
DataInputStream is = conn.openDataInputStream();
dos = conn.openDataOutputStream();
```

Denne fik vi lavet ved et af vores sidste møder og den kunne genanvendes i vores nye NXT ↔ PC ↔ PC ↔ NXT arkitektur.
Trods den noget klodsede arkitektur, så viste vores eksperimenter, at denne arkitektur ikke var en forhindring i forhold til at kunne fjernstyre robotterne uden forsinkelse.

FinishBehavior:

Denne behavior har til at opgave at kunne lukke ned for styring af robotten, hvis den får at vide, at der er fundet en vinder. Da FinishBehavior ligger over alle de andre behaviors i hierarkiet, så bliver de andre behaviors automatisk suppressed.
Her ses klassens `run()` metode i FinishBehavior.java:

```
public void run() {
	while (!finished) {}

	suppress();

	stopBehavior();
	LCD.clearDisplay();

	if (winner) {
		LCD.drawString("WINNER", 0, 0);
	} else {
		LCD.drawString("LOSER", 0, 0);
	}

	while (true) {}}
```

Klassen indeholder desuden 2 fields hhv. `finished` og `winner`, som bruges til at checke, om spillet er slut og om den givne NXT er vinderen eller taberen. Som det ses indeholder `run()` metoden heller ingen `release()` metode, da spillet er slut. Derfor er der ingen grund til at give styringen tilbage til brugeren.

Behavioren indeholder også setter metoder til de 2 fields, som bruges i hhv. LightCheckPoint og RemoteControl klasserne.
I LightCheckPoint klassen bruges de, når vi har nået det givne antal checkpoints, som er krævet for at vinde:

```
if (rc.getCheckPointsSoFar() == rc.getNumberOfCheckPoints()) {
	try {
		if (output == null) output = rc.getOutputStream();
		output.writeInt(42); //42 = winner message
		output.flush();
	} catch (IOException e) {}

	finish.setWinner(true);
	finish.setFinished(true);
}
```

Her ses checket for 'vinderen'. Hvis antallet af checkpoints for at vinde er nået (3 i vores fremvisning) så sendes værdien 42 til den anden NXT enhed (taberen). Derefter sættes `winner` og `finished` i FinishBehavior klassen til true.
Efter at den tabende NXT-enhed så modtager beskeden 42 sker der følgende i RemoteControl klassen:

```
if(checkPoint == 42) {
	finish.setFinished(true);
}
```

Hvor `finished` variablen i FinishBehavior bliver sat til true, `winner` forbliver urørt da den er sat til false som standard.

Konklusion:

Vores kommunikation imellem de 2 NXT enheder er blevet implementeret ifølge den bekrevne arkitektur, og den virker ganske udmærket. Vi synes dog selv, at den lange omvej i kommunikationsarkitektur er uheldig, men det var desværre nødvendigt at implementere en mindre pæn løsning grundet restriktioner på NXT enheden.
FinishBehavior behavioren er blevet implementeret korrekt, og den virker som forventet. Den stopper al form for styring af robotterne, når en af de 2 konkurrerende personer har vundet løbet.