Documentazione Progetto NXT

Titolo del progetto: Documentazione Progetto Registrazione
Alunni/e: Jari Näser, Paolo Gübeli
Classe: Informatica 3AA
Anno scolastico: 2018/2019
Docente responsabile: Adriano Barchi, Francesco Mussi e Luca Muggiasca

1 Introduzione

1.1 Informazioni sul progetto

Allievi coinvolti: Jari Näser, Paolo Gübeli
Classe: Informatica 3AA Presso la Scuola Arti e Mestieri Trevano
Docenti responsabili: Francesco Mussi, Adiano Barchi, Luca Muggiasca
Data inizio: 7-11-18
Data fine: 8-02-19

1.2 Abstract

Today a lot of schools use LEGO Mindstorm to introduce younglings into computer programming and engineering, the scope of the project was to realize some libraries to make it easier for schools to educate the students. These libraries will simplify some operations that are too complicated for amateurs. To achieve this goal, we used the programming language Java.
There will be also a guide to help the teachers and the students to install the firmware and how to use those libraries.

1.3 Scopo

Lo scopo di questo progetto è di creare delle librerie che permettono di semplificare delle operazioni con il prodotto Mindstorm NXT. Queste librerie verrano in seguito usate per semplificare la programmazione dei LEGO Mindstorm togliendo passaggi ripetitivi. Questo per aiutare docenti che cercano di insegnare la programmazione a ragazzi alle prime armi come a loro stessi.

Analisi

1.4 Analisi del dominio

Attualmente si usa il vecchio sistema a blocchetti semplificato della lego che non permette di avere funzionalità avanzate ed è pensato per regazzi giovani alle prime armi con l'informatica.
Di conseguenza non è ancora possibile programmare i robot con del codice.

1.5 Analisi e specifica dei requisiti

Inizialmente bisogna installare il firmware per java leJOS che permette al mindstorm di leggere i file java. In seguito bisogna creare per ogni blocchetto esistente nel editor grafico di NXT un metodo che lo sostituisca. Ogni libreria dovrà permettere di usare un metodo che gestisce un'azione con degli input e degli output.
In seguito con queste librerie bisogna crare un programma che muove il robot in giro e si gira quando tocca o vede un oggetto comunemente chiamato Explorer.

ID: REQ-01	
Nome	Firmware
Priorità	1
Versione	1.0
Note	Firmware che permettono di usare java
Sotto requisiti	
01	Si necessita di un mindstorm funzionante

ID: REQ-02	
Nome	Creazione librerie
Priorità	1
Versione	1.0
Note	Librerie che permettono di fare azioni complicate in modo semplice così da semplificare l'uso del mindstorm
Sotto requisiti	
01	Si necessita dei sensori
02	Si necessita dei attuatori
03	Si necessita dei firmware installati

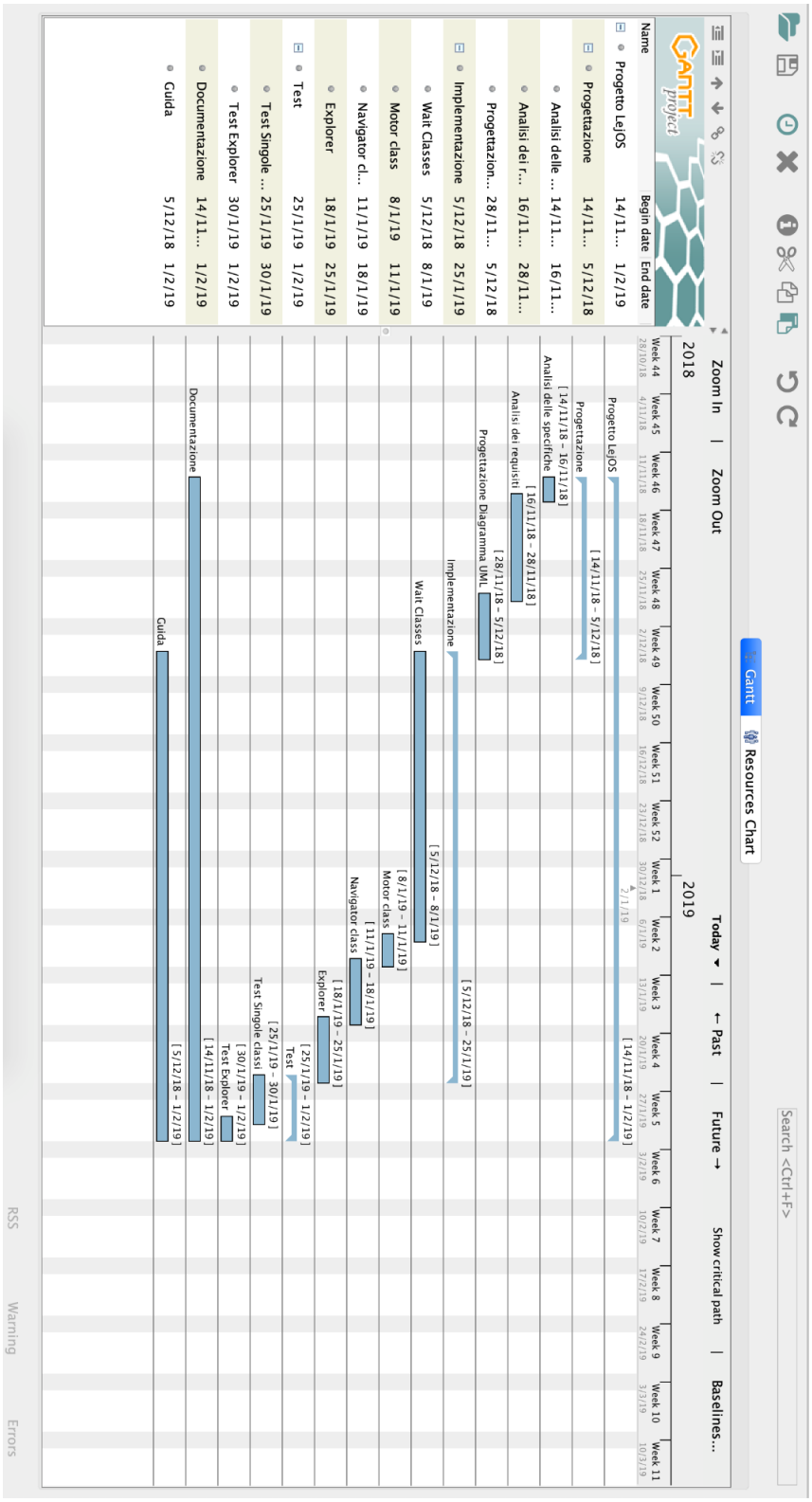
ID: REQ-03	
Nome	Guida per gli utenti
Priorità	1
Versione	1.0
Note	Guida che permetta di usare le librerie create
Sotto requisiti	

01	Installazione dei firmware completa
02	Librerie terminate e complete

ID: REQ-04	
Nome	Explorer che fa uso delle librerie
Priorità	1
Versione	1.0
Note	Explorer che usa le classi delle librerie per funzionare
Sotto requisiti	
01	Mindsrorm funzionante con LeJos installato
02	Libreria creata e funzionante

1.6 Pianificazione

Questa è la nostra pianificazione del progetto e il modo in cui abbiamo suddiviso i tempi:



1.7 Analisi dei mezzi

1.7.1 Software

Per la realizzazione di questo progetto abbiamo usato come software:

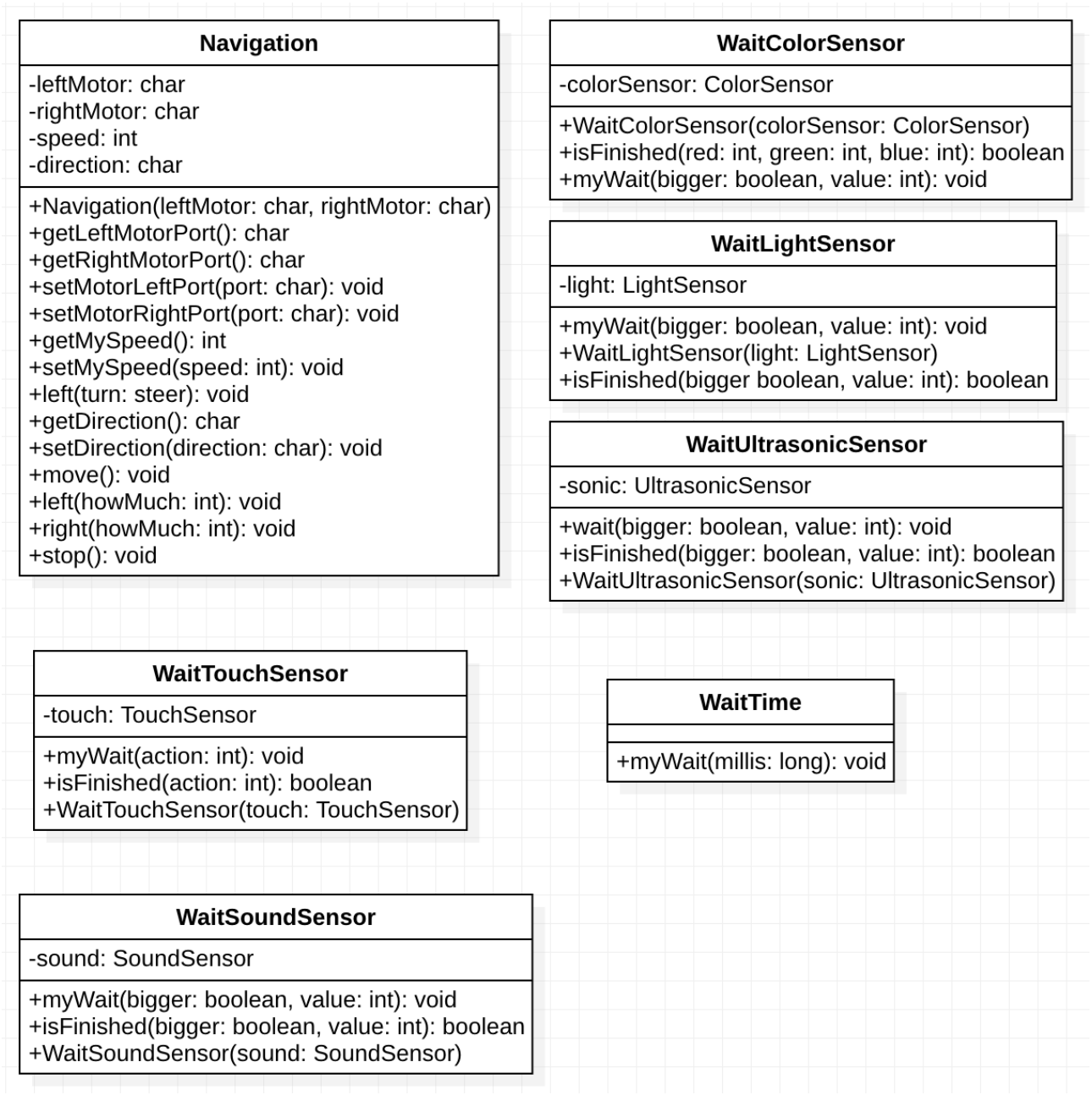
- LeJOS 0.9.1: Firmware e compilatore che ci permette di usare Java su mindstorm.
- Sublime Text 3.1.1: ci aiuta a scrivere i codici java.
- Notepad++ 7.5.1: aiuta a scrivere ogni sorta di codice.
- Word 2016: ci ha permesso di scrivere la Documentazione del progetto e la guida.
- GanttProject 2.8.5: ci ha permesso di fare il Gantt iniziale e il Gantt consuntivo

1.7.2 Hardware

Per questo progetto non abbiamo necessitato di materiale particolare, abbiamo usato i nostri portatili MacBook Pro 2015 con il sistema operativo OS X Mojave e Hp OMEN 17” con sistema operativo Windows 10.

2 Progettazione

2.1 Design dell'architettura del sistema



3 Implementazione

3.1 Navigation

La classe Navigation ha come scopo principale quello di mettere assieme tutte le classi ed esportare dei metodi semplici e funzionanti per guidare il robot.
Infatti è composto dai 4 metodi principali:

- **Move**: Si occupa di far partire il robot.
- **Left**: Si occupa di far curvare il robot a sinistra.
- **Right**: Si occupa di far curvare il robot a destra.
- **Stop**: Si occupa di fermare il robot

E di tutti i suoi metodi ed attributi rimanenti.

```
import lejos.nxt.*;

public class Navigation{

	private char leftMotor = 'A';
	private char rightMotor = 'B';
	private int speed = 0;
	private char direction = 'F';

	public Navigation(char leftMotor, char rightMotor){
		setMotorLeftPort(leftMotor);
		setMotorRightPort(rightMotor);
	}

	public char getLeftMotorPort(){
		return this.leftMotor;
	}

	public char getRightMotorPort(){
		return this.rightMotor;
	}

	public void setMotorLeftPort(char port){
		String mp = port + "";
		port = mp.toUpperCase().charAt(0);
		if(port == 'A' || port == 'B' || port == 'C'){
			leftMotor = port;
		}
	}

	public void setMotorRightPort(char port){
		String mp = port + "";
		port = mp.toUpperCase().charAt(0);
		if(port == 'A' || port == 'B' || port == 'C'){
			leftMotor = port;
		}
	}

	public int getMySpeed(){
		return this.speed;
	}

	public void setMySpeed(int speed){
```

```
		if(speed >= 0){
			this.speed = speed;
		}
	}

	public char getDirection(){
		return this.direction;
	}

	public void setDirection(char direction){
		String s = direction + "";
		direction = s.toUpperCase().charAt(0);
		if(direction == 'F' || direction == 'B'){
			this.direction = direction;
		}
	}

	public void move(){
		switch(getLeftMotorPort()){
			case 'A':
				Motor.A.setSpeed((float)this.getMySpeed());
			case 'B':
				Motor.B.setSpeed((float)this.getMySpeed());
			case 'C':
				Motor.C.setSpeed((float)this.getMySpeed());
		}
		switch(getRightMotorPort()){
			case 'A':
				Motor.A.setSpeed((float)this.getMySpeed());
			case 'B':
				Motor.B.setSpeed((float)this.getMySpeed());
			case 'C':
				Motor.C.setSpeed((float)this.getMySpeed());
		}
		if(getDirection() == 'F'){
			switch(getLeftMotorPort()){
				case 'A':
					Motor.A.forward();
				case 'B':
					Motor.B.forward();
				case 'C':
					Motor.C.forward();
			}
			switch(getRightMotorPort()){
				case 'A':
					Motor.A.forward();
				case 'B':
					Motor.B.forward();
				case 'C':
					Motor.C.forward();
			}
		}else{
			switch(getLeftMotorPort()){
				case 'A':
					Motor.A.backward();
				case 'B':
					Motor.B.backward();
				case 'C':
```

```
                                Motor.C.backward();
                        }
                        switch(getRightMotorPort()){
                                case 'A':
                                        Motor.A.backward();
                                case 'B':
                                        Motor.B.backward();
                                case 'C':
                                        Motor.C.backward();
                        }
                }
        }

        public void left(int howMuch){
                switch(getLeftMotorPort()){
                        case 'A':
                                Motor.A.setSpeed((float)(this.getMySpeed() - this.getSteering()/2));
                        case 'B':
                                Motor.B.setSpeed((float)(this.getMySpeed() - this.getSteering()/2));
                        case 'C':
                                Motor.C.setSpeed((float)(this.getMySpeed() - this.getSteering()/2));
                }
                switch(getRightMotorPort()){
                        case 'A':
                                Motor.A.setSpeed((float)(this.getMySpeed() + this.getSteering()/2));
                        case 'B':
                                Motor.B.setSpeed((float)(this.getMySpeed() + this.getSteering()/2));
                        case 'C':
                                Motor.C.setSpeed((float)(this.getMySpeed() + this.getSteering()/2));
                }
        }

        public void right(int howMuch){
                switch(getLeftMotorPort()){
                        case 'A':
                                Motor.A.setSpeed((float)(this.getMySpeed() + this.getSteering()/2));
                        case 'B':
                                Motor.B.setSpeed((float)(this.getMySpeed() + this.getSteering()/2));
                        case 'C':
                                Motor.C.setSpeed((float)(this.getMySpeed() + this.getSteering()/2));
                }
                switch(getRightMotorPort()){
                        case 'A':
                                Motor.A.setSpeed((float)(this.getMySpeed() - this.getSteering()/2));
                        case 'B':
                                Motor.B.setSpeed((float)(this.getMySpeed() - this.getSteering()/2));
                        case 'C':
                                Motor.C.setSpeed((float)(this.getMySpeed() - this.getSteering()/2));
                }
        }

        public void stop(){
                switch(getLeftMotorPort()){
                        case 'A':
                                Motor.A.stop();
                        case 'B':
                                Motor.B.stop();
                        case 'C':
```

```
                    Motor.C.stop();
                }
                switch(getRightMotorPort()){
                    case 'A':
                        Motor.A.stop();
                    case 'B':
                        Motor.B.stop();
                    case 'C':
                        Motor.C.stop();
                }
        }
}
```

3.2 Sensori

Abbiamo implementato le classi per aspettare che dei sensori ritornino un valore.
Per usare le classi bisogna prima implementarle nel proprio programma e passare il sensore usato.
Tutte le classi hanno due metodi base:
myWait() che interrompe il programma finché isFinished non ritorna true.
isFinished() che restituisce true se la condizione è verificata.
Questo perché magari un utente vuole eseguire del codice mentre aspetta la condizione, quindi lasciamo la possibilità di creare un proprio while e mettere come condizione isFinished().

Queste classi sono state suddivise in tre gruppi:
-Pulsanti
-Colore
-Sensori Analogici (Ultrasuoni, suono, luminosità)

3.2.1 WaitTouchSensor

Per iniziare abbiamo implementato la classe WaitTouchSensor che è quella più semplice, abbiamo suddiviso gli input in due, pressed, released con dei valori int 0,1 che verrano passati al myWait() o al isFinished().

```
import lejos.nxt.*;

public class WaitTouchSensor{

        private TouchSensor touch;

        private boolean pressed = false;

        public WaitTouchSensor(TouchSensor touch){
                this.touch = touch;
        }

        public void myWait(int action){
                while(isFinished(action)){

                }
        }

        public boolean isFinished(int action){
                boolean finished = false;
                if(action == 0){
                        finished = touch.isPressed();
                }else{
```

```
                        if(pressed){
                                finished = !touch.isPressed();
                        }
                        pressed = touch.isPressed();
                }
                return finished;
        }
}
```

3.2.2 WaitColorSensor

WaitColorSensor è l'unica classe che usa un range di valori quindi abbiamo dovuto cercare un buon range intorno ai valori passati, questo perché con questi sensori non uscira mai il valore esatto quindi bisogna essere larghi con i valori noi abbiamo optato per un 8% di margine d'errore. Quando si usa la classe bisogna passare il valore di RGB suddivisi in tre int.

```
import lejos.nxt.*;
import lejos.robotics.*;

public class WaitColorSensor{

        private ColorSensor cs;

        public WaitColorSensor(ColorSensor cs){
                this.cs = cs;
        }

        public void wait(int red, int green, int blue){
                while(isFinished(red, green, blue)){

                }
        }

        public boolean isFinished(int red, int blue, int green){
                Color c = cs.getColor();
                if(c.getRed() > red-10 && c.getRed() < red+10){
                        if(c.getGreen() > green-10 && c.getGreen() < green+10){
                                if(c.getBlue() > blue-10 && c.getBlue() < blue+10){
                                        return true;
                                }
                        }
                }
                return false;
        }
}
```

3.2.3 WaitLightSensor

WaitLightSensor è una classe che gestisce il sensore di luce e riceve un valoreint e un boolean per sapere se il valore cercato dev'essere maggiore o minore del valore passato.

```
import lejos.nxt.*;

public class WaitLightSensor{

	private LightSensor light;

	public WaitLightSensor(LightSensor light){
		this.light = light;
	}

	public void myWait(boolean bigger, int value){
		while(isFinished(bigger, value)){

		}
	}

	public boolean isFinished(boolean bigger, int value){
		if(bigger){
			if(light.getLightValue() > value){
				return true;
			}
		}else{
			if(light.getLightValue() < value){
				return true;
			}
		}
		return false;
	}
}
```

3.2.4 WaitUltrasonicSensor

WaitUltrasonicSensor è una classe che gestisce il sensore ad ultrasuoni e riceve un valore int e un boolean per sapere se il valore cercato dev'essere maggiore o minore del valore passato.

```
import lejos.nxt.*;

public class WaitUltrasonicSensor{

	private UltrasonicSensor sonic;

	public WaitUltrasonicSensor(UltrasonicSensor sonic){
		this.sonic = sonic;
	}

	public void wait(boolean bigger, int value){
		while(isFinished(bigger, value)){

		}
	}
```

```
	public boolean isFinished(boolean bigger, int value){
		if(bigger){
			if(sonic.getDistance() > value){
				return true;
			}
		}else{
			if(sonic.getDistance() < value){
				return true;
			}
		}
		return false;
	}
}
```

3.3 Explorer

La classe explorer si occupa di fare schivare tutti gli ostacoli presenti nel percorso da effettuare oppure in una zona rinchiusa al robot attraverso due sensori di tatto, uno ad ultrasuoni ed uno di colore, inoltre quando passa su una riga nera si ferma.

Il suo funzionamento è molto semplice: Quando sta per incontrare o ha incontrato un ostacolo si ferma, torna in dietro, gira a destra o sinistra, si ferma e riparte proseguendo il suo nuovo percorso.

```
import lejos.nxt.*;

public class Explorer{
	private static final char MOTOR_LEFT_PORT = 'A';
	private static final char MOTOR_RIGHT_PORT = 'B';
	private static final int DISTANCE = 30;
	private static Navigation navigator;
	private static UltrasonicSensor ultrasonicSensor = new UltrasonicSensor(SensorPort.S3);
	private static WaitUltrasonicSensor waitUltrasonicSensor = new
WaitUltrasonicSensor(ultrasonicSensor);
	private static TouchSensor touchSensorLeft = new TouchSensor(SensorPort.S2);
	private static TouchSensor touchSensorRight = new TouchSensor(SensorPort.S4);
	private static WaitTouchSensor waitTouchSensorLeft = new WaitTouchSensor(touchSensorLeft);
	private static WaitTouchSensor waitTouchSensorRight = new WaitTouchSensor(touchSensorRight);
	private static LightSensor lightSensor = new LightSensor(SensorPort.S1);
	private static WaitLightSensor waitLightSensor = new WaitLightSensor(lightSensor);
	private static WaitTime wait;

	public Explorer(){
		navigator = new Navigation(MOTOR_LEFT_PORT, MOTOR_RIGHT_PORT);
		navigator.setMySpeed(300);
		wait = new WaitTime();
	}

	public static void main(String[] args){
		navigator.move();
		while(true){
			if(waitUltrasonicSensor.isFinished(false, DISTANCE)){
				//Inserire cosa deve fare se è più vicino di 30 cm
				navigator.stop();
				navigator.setDirection('B');
				navigator.move();
```

```
                                wait.myWait(1500);
                                navigator.stop();
                                navigator.setDirection('F');
                                navigator.right(60);
                                wait.myWait(2000);
                                navigator.setMySpeed(300);
                        }
                        if(waitTouchSensorLeft.isFinished(0)){
                                //gira a sinistra indietro
                                navigator.stop();
                                navigator.setDirection('B');
                                navigator.move();
                                wait.myWait(1500);
                                navigator.stop();
                                navigator.setDirection('F');
                                navigator.left(60);
                                wait.myWait(2000);
                                navigator.setMySpeed(300);
                        }
                        if(waitTouchSensorRight.isFinished(0)){
                                //gira a destra indietro
                                navigator.stop();
                                navigator.setDirection('B');
                                navigator.move();
                                wait.myWait(1500);
                                navigator.stop();
                                navigator.setDirection('F');
                                navigator.right(60);
                                wait.myWait(2000);
                                navigator.setMySpeed(300);
                        }
                        if(waitLightSensor.isFinished(false, 50)){
                                //Si ferma
                                navigator.stop();
                        }
                }
        }
}
```

4 Test

4.1 Protocollo di test

Test Case: **Riferimento**:	TC-001 REQ-001	**Nome:**	Installazione Firmware funzionante
Descrizione:	Prova del funzionamento corretto del firmware		
Prerequisiti:	Robot Mindstorm funzionante		
Procedura:	1. Accendere il mindstorm tramite il pulsante centrale 2. Caricare tramite cavo un programma funzionante con un output		
Risultati attesi:	Il robot dovrebbe avviare il programma e ritornare l'output		

Test Case: **Riferimento**:	TC-002 REQ-002	**Nome:**	Controllo funzionamento caricamento librerie
Descrizione:	Prova se è possibile caricare le librerei su robot mindstorm		
Prerequisiti:	Mindstorm funzionante Firmware funzionante		
Procedura:	1. Scaricare le librerie 2. Caricare tramite cavo le librerie scaricate		
Risultati attesi:	Sotto la sezione files dovrebbero esserci i file delle librerie		

Test Case: **Riferimento**:	TC-003 REQ-003	**Nome:**	Guida corretta e funzionante
Descrizione:	Provare il funzionamento del codice d'esempio nella guida.		
Prerequisiti:	Mindstorm funzionante Firmware funzionante. Librerie funzionanti.		
Procedura:	1. Aprire la guida 2. Copiare il codice illustrato nella guida 3. Inserire il codice copiato in un programma 4. Caricare il programma sul robot 5. Far partire il programma.		
Risultati attesi:	Il programma dovrebbe funzionare correttamente senza interruzioni.		

Test Case: **Riferimento**:	TC-004 REQ-004	**Nome:**	Funzionamento programma explorer.
Descrizione:	Test del corretto funzionamento del programma explorer.class.		
Prerequisiti:	Mindstorm funzionante Firmware funzionante. Librerie funzionanti.		
Procedura:	1. Scaricare il file aggiuntivo di prova della libreria explorer. 2. Caricare il programma explorer sul mindstorm tramite cavo. 3. Appogiare il robot su una superficie piana, con un terreno agibile (pavimento, tavolo) 4. Avviare il programma.		
Risultati attesi:	Tutti gli input immessi nei Form vengono controllati e se sono accettati vengono mandati alla tabella di conferma		

4.2 Risultati test

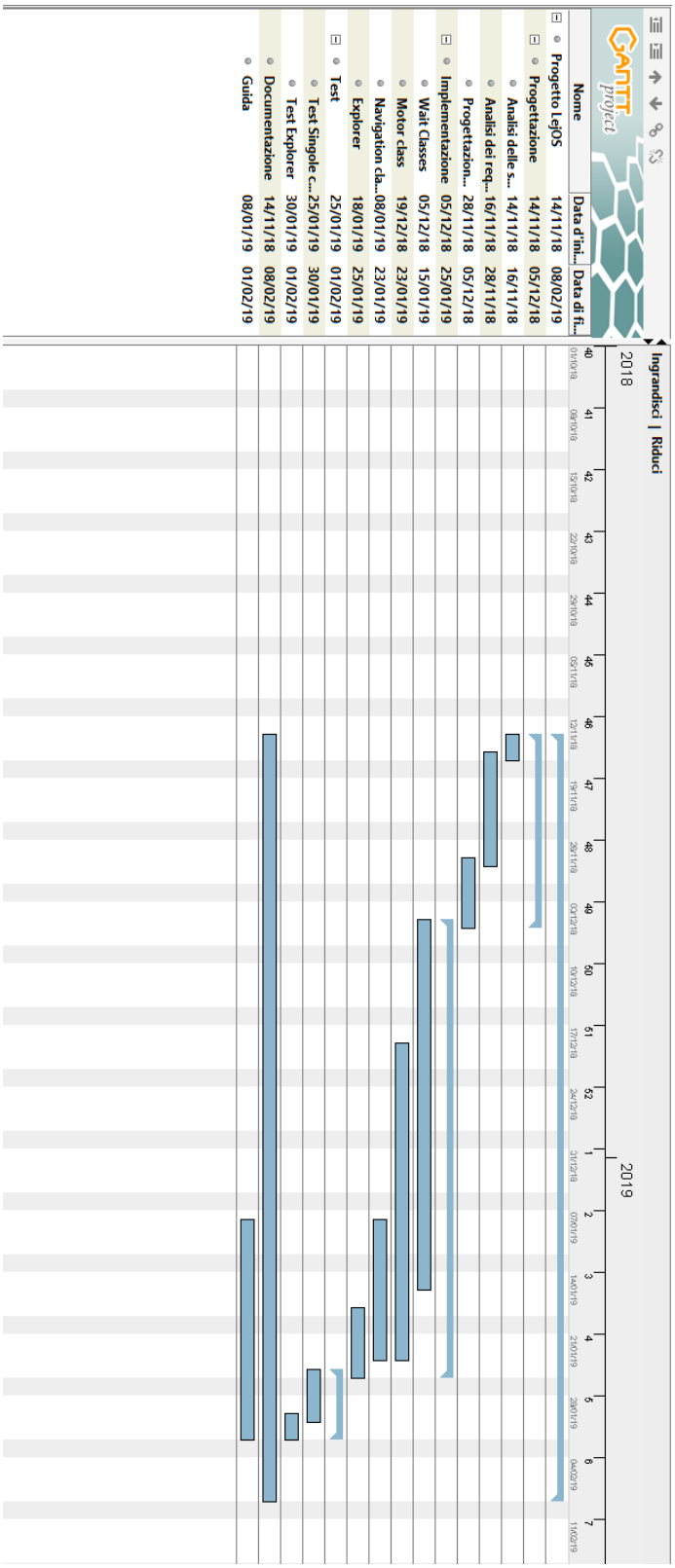
Test Case	Numero Passaggio	Risultato
TC-001	1-2	Il robot ci ritorna l'output corretto
TC-002	2	Exception 134 Quest'eccezione non esiste nella documentazione ufficiale di LeJOS e da nessun'altra pagina
TC-003	4	Exception 134 Quest'eccezione non esiste nella documentazione ufficiale di LeJOS e da nessun'altra pagina
TC-004	2	Exception 134 Quest'eccezione non esiste nella documentazione ufficiale di LeJOS e da nessun'altra pagina

4.3 Mancanze/limitazioni conosciute

La struttura di lejOS è limitante non potendo creare i motori come oggetti essendo già presenti, inoltre abbiamo riscontrato un'errore nel passaggio dei file da PC a Blocchetto NXT in quanto sollevava un'eccezione non documentata nella documentazione ufficiale di lejOS.

5 Consuntivo

Siamo riusciti a mantenere i tempi della progettazione e dei test ma l'implementazione per colpa di vari problemi tecnici ci ha preso un po' più tempo del previsto:



6 Conclusioni

Il nostro prodotto offre un uso molto semplice del robot LEGO NXT attraverso del codice Java che può essere facilmente usato ed interpretato da chiunque abbia delle minime conoscenze nel mondo della programmazione.

6.1 Sviluppi futuri

Sicuramente ci sono molti metodi per aiutare l'utente che si possono aggiungere anche se dobbiamo dire che i metodi di base di LeJOS sono già molto buoni e permettono di fare tutto, con l'aggiunta delle nostre librerie l'uso di LeJOS diventa facile da usare ma bisogna avere comunque una conoscenza di base del linguaggio di programmazione java.

6.2 Considerazioni personali

Abbiamo imparato a collaborare in un progetto, e questo ci ha fatto capire l'importanza della puntualità e della costanza nei commit e nei push. Grazie a questo progetto abbiamo capito l'importanza della progettazione che ha reso facile la suddivisione dei lavori e la gestione dei tempi di consegna. A differenza del progetto fatto da soli si dipendeva dal compagno e viceversa quindi una buona collaborazione e comunicazione è essenziale per la riuscita del lavoro.

7 Bibliografia

7.1 Sitografia

- http://stackoverflow.com/, *Stack OverFlow*, dal 19.12.2018 al 25.01.2019
- http://www.lejos.org/,dal 19.12.2018 al 18.01.2019
- http://www.lejos.org/nxt/pc/api/index.html
- http://www.free-powerpoint-templates-design.com

8 Allegati

- Diari di lavoro
- Codici sorgente
- Guida per l'utente
- Quaderno dei compiti
- Prodotto