Politecnico di Milano

Facoltà di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Dipartimento di Elettronica, Informazione e Bioingegneria

RSPSLSbot

Prof.ssa: Anna Maria ANTOLA

Prof. Tutor: Andrea BONARINI

Progetto di HCI di:

Luca Andrea RAHO 756417

Carlo Alberto Maria VIOLA 756651

Anno Accademico 2012/2013

Sommario

Introduzione

Il progetto è stato elaborato da due studenti: Luca Andrea Raho, Carlo Alberto Maria Viola e ed ha richiesto circa quattro mesi di tempo.

Essendo la prima volta che i suddetti studenti (di seguito "team") si confrontavano con una simile tipologia di lavoro, è stato necessario, in via prioritaria, analizzare e approfondire tutte le funzionalità messe a disposizione dal robot LEGO NXT, dal linguaggio di programmazione NXC e dall'ambiente di sviluppo Bricxcc.

Solo successivamente è stato affrontato lo sviluppo del gioco vero e proprio, procedendo come segue:

- progettazione dei Robot;
- navigazione del robot autonomo;
- strategie di movimento del robot autonomo;
- gestione delle fasi di gioco tramite robot autonomo;
- movimenti del robot controllato dal giocatore;
- progettazione del Joystick;
- integrazione tra Joystick, robot del giocatore e robot autonomo via Bluetooth.

Stato dell'arte

Il progetto RSPSLbot parte da una variante della "morra cinese" (sasso-carta-forbici-Lizard-Spock), inventata da due studenti americani, Sam Kass e Karen Bryla e resa successivamente famosa da un episodio del telefilm "The Big Bang Theory".

La variante consiste nell'introduzione di due nuovi segni: “Lizard” e “Spock” (direttamente dal celebre episodio “Arena” della prima stagione della saga originale di Star Trek del 1967 in cui l’equipaggio dell’Enterprise incontra i Gorn, razza aliena dalle sembianze lucertoloidi) che aumentando le possibilità di combinazioni riduce la frequenza dei pareggi.

Tale scelta parte dal presupposto che giocatori che si conoscono da sufficiente tempo sviluppano una prevedibilità reciproca che determina una maggior frequenza dei pareggi.

Con l'introduzione dei due nuovi segni, le nuove combinazioni diventano dunque:

- le forbici tagliano la carta
- la carta avvolge il sasso
- il sasso schiaccia Lizard
- Lizard avvelena Spock
- Spock rompe le forbici
- le forbici decapitano Lizard
- Lizard mangia la carta
- la carta invalida Spock
- Spock vaporizza il sasso

…e come è sempre stato...

- il sasso rompe le forbici

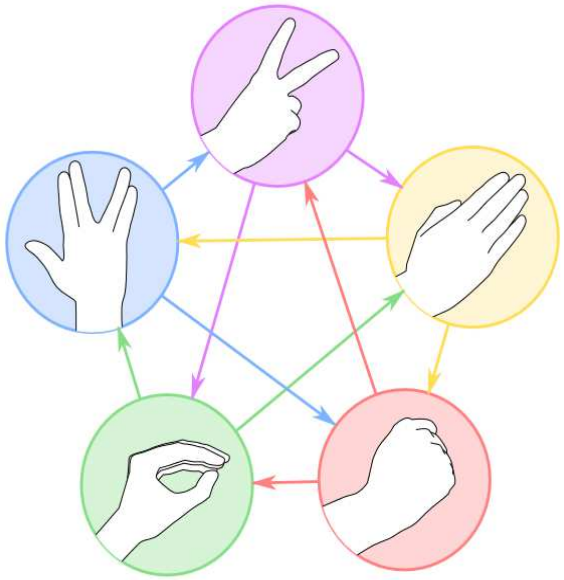


Illustrazione - Schema del gioco Sasso, Carta, Forbici, Lizard, Spock

Questa modifica è stata sfruttata nel progetto in oggetto per evitare inutili e noiosi allungamenti del gioco dovuti proprio al numero eccessivo di pareggi.

Ulteriori spunti per il progetto sono derivati dal gioco RoboWii 2.0 nella sua versione migliorata. RoboWii 2.0.L è un gioco robotico interattivo, sviluppato come progetto dell'AIRLab, laboratorio del dipartimento di elettronica e informatica del Politecnico di Milano.
Entrambi i progetti (RoboWii 2.0.L e il progetto in esame) hanno in comune l'utilizzo di robot LEGO NXT MINDSTORM e condividono regole di gioco basate su una meccanica di tipo “Preda e Predatore”.

A differenza dei precedenti, il progetto in esame, prevede che il "robot autonomo" possa ricoprire sia il ruolo di preda sia quello di predatore. Non è inoltre necessario l'utilizzo di un computer e il giocatore controlla il suo robot attraverso un joystick, costruito “ad hoc” sfruttando i componenti del robot LEGO NXT MINDSTORM.

Requisiti

Hardware

- 3 kit LEGO MINDSTORM NXT utilizzati per costruire il joystick, il “robot autonomo” e quello controllato dal giocatore. La scelta è ricaduta su questa tipologia di robot per la sua ampia modularità, grazie all’utilizzo dei componenti LEGO e alla possibilità di utilizzare come linguaggio di programmazione l’NXC, molto simile al C di cui il “team” aveva già fatto precedenti esperienze.
- Un campo da gioco adeguatamente proporzionato.

Software

- 3 programmi scritti in NXC, linguaggio simile al C che fornisce delle API per sfruttare le funzionalità dei robot LEGO MINDSTORM NXT, da caricare rispettivamente sul joystick, sul "robot autonomo" e su quello controllato dal giocatore. I tre programmi sono stati integralmente elaborati dal "team" partendo completamente da zero.

Obiettivi

L'obiettivo del progetto è quello di creare un gioco di media difficoltà e di facile comprensione, al fine di evitare di scoraggiare con regole troppo complicate i giocatori "pigri" e allo stesso tempo, stimolare gli interessi di giocatori più esigenti e interessati alla sfida.

Per mantenere un adeguato livello di difficoltà si è quindi provveduto a sbilanciare alcune regole a favore del "robot autonomo" (velocità di movimento e ampiezza del bersaglio) che diversamente, essendo dotato unicamente di una intelligenza artificiale predefinita si sarebbe trovato in una situazione di eccessivo svantaggio.

In conclusione si ritiene che questo gioco possa interessare sia giocatori di fascia giovane, avvezzi ai giochi elettronici, sia a utenti più maturi in considerazione dell'utilizzo di componenti più tradizionali (joystick, LEGO).

Dotazioni dei Robot

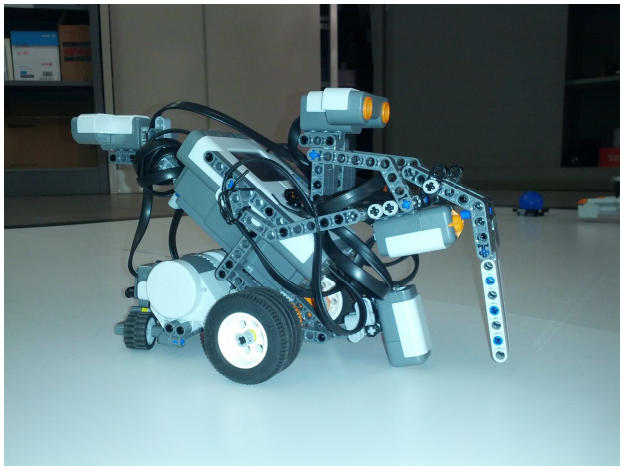
Robot del giocatore "Wolodroid"
Il robot controllato dal giocatore è composto da:

- chassis di tipo Tribot dotato di 3 motori assiali NXT;
- due ruote motrici anteriori per il movimento;
- una ruota posteriore di supporto direzionale;
- sensore di tocco per segnalare i colpi andati a segno;
- sensore di colore sul fronte diretto verso il basso necessario per evitare la fuoriuscita dal campo di gioco.

"Illustrazione - Wolodroid"

Robot autonomo "Sheldonator"
Il robot autonomo è composto da:

- chassis di tipo Bibot dotato di 2 motori assiali NXT;
- due ruote motrici anteriori per il movimento;
- una ruota posteriore di supporto direzionale;
- sensore a ultrasuoni montato sul fronte per evitare gli ostacoli;
- sensore a ultrasuoni montato sul retro per evitare gli ostacoli;
- sensore di luce montato sul fronte diretto verso il basso per evitare di fuoriuscire dal campo di gioco;
- sensore di tocco montato sul fronte per rilevare i colpi assestati o subiti.

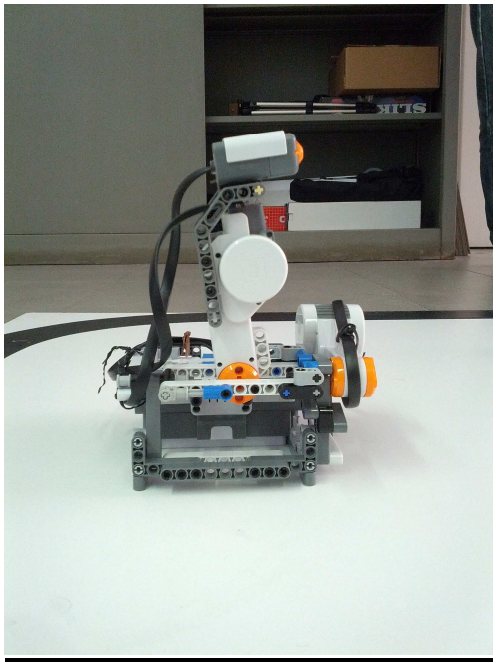


"Illustrazione - Sheldonator"

Joystick "Robostick"

Il joystick è composto da:

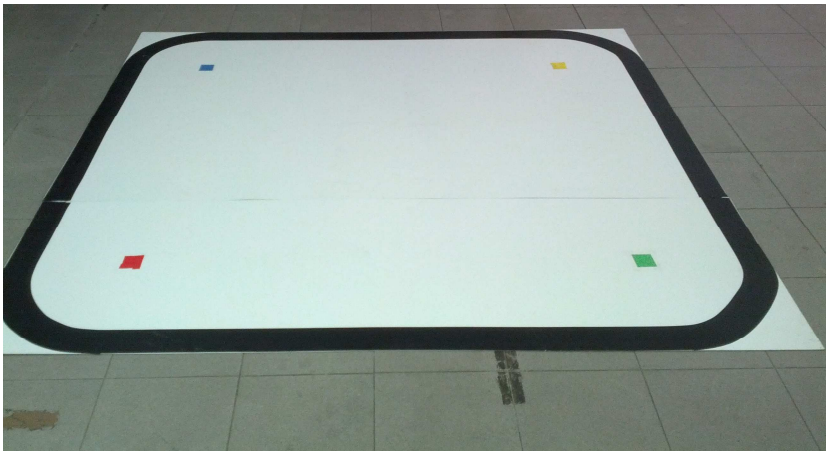
- due motori assiali NXT;
- sensore di tocco montato a grilletto utilizzato in progetti paralleli.



“Illustrazione - Robostick”

Game “concept”

Introduzione
Il gioco si svolge su campo quadrangolare con angoli smussati i cui lati misurano 1,64m X 1,84m e con bordi di colore nero di spessore 8cm.
(il tabellone esterno misura 1,80m X 2,00m)



“Illustrazione - Campo di gioco”

Il gioco si divide in due fasi: "Morra" e "Caccia".

Morra
Nella fase “Morra” il giocatore ed il "robot autonomo" si sfidano ad una partita di "sasso-carta-forbici-Lizard-Spock".
Il vincitore di questa prima fase ricoprirà il ruolo di predatore nella fase successiva.
In caso di pareggio questa fase viene ripetuta fino alla vittoria di uno dei due contendenti.

Caccia
Durante la fase di "Caccia" il predatore ha 60 secondi per colpire la preda e aggiudicarsi il round.
In difetto, il round viene assegnato alla preda.

Per aumentare il grado di difficoltà del gioco, al robot autonomo, nel ruolo di predatore, è consentita una maggiore possibilità di infliggere colpi all'avversario.
In particolare, il giocatore può colpire il "robot autonomo" solo su fronte mentre il "robot autonomo" può colpire quello del giocatore in qualunque punto.
Quando il robot autonomo ricopre il ruolo di preda si muove secondo dei pattern che dipendono dalla scelta effettuata nella fase “Morra”.

Pattern di movimento ("robot autonomo" preda)
Ciascun pattern prevede un movimento base che viene modificato dal robot nel caso identifichi delle minacce.

Sasso:

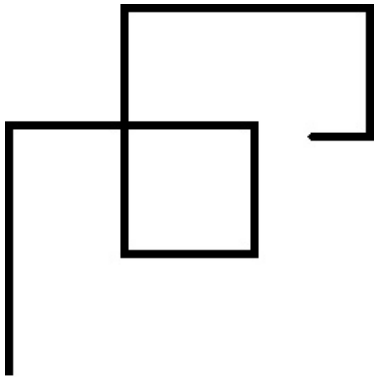
- Pattern base: fermo;
- rileva colore nero e ultrasuoni sul retro: fermo;
- rileva ultrasuoni sul retro e sul fronte: ruota di 90 gradi a destra o sinistra;
- rileva colore nero o ultrasuoni sul fronte: ruota di 180 gradi su se stesso;
- rileva ultrasuoni sul retro: avanza.

"Pattern 1 - Posizione ferma"

Colore nero	Ultrasuoni frontale	Ultrasuoni retro	Comportamento
0	0	0	Fermo
0	0	1	Fermo
0	1	0	Ruota 180 gradi
0	1	1	Ruota 90 gradi
1	0	0	Ruota 180 gradi
1	0	1	Fermo
1	1	0	Ruota 180 gradi
1	1	1	Fermo

Carta:

- Pattern base: spirali quadrate;
- rileva ultrasuoni sul retro: ruota di 90 gradi a destra o sinistra;
- rileva ultrasuoni sul fronte o colore nero: ruota di 180 gradi su se stesso.



“Pattern 2 - Spirale quadrata”

Colore nero	Ultrasuoni frontale	Ultrasuoni retro	Comportamento
0	0	0	Spirali Quadrate
0	0	1	Ruota 90 gradi
0	1	0	Ruota 180 gradi
0	1	1	Ruota 90 gradi
1	0	0	Ruota 180 gradi
1	0	1	Ruota 90 gradi
1	1	0	Ruota 180 gradi
1	1	1	Ruota 90 gradi

Forbici:

- Pattern base: avanza;
- rileva colore nero e ultrasuoni sul retro o rileva ultrasuoni sul fronte e sul retro: ruota di 90 gradi a destra o sinistra;
- rileva ultrasuoni sul fronte o colore nero: ruota di 135 gradi a destra o sinistra;
- rileva ultrasuoni sul retro: ruota di 45 gradi a destra o sinistra.

"Pattern 3 - Moto rettilineo"

Colore nero	Ultrasuoni frontale	Ultrasuoni retro	Comportamento
0	0	0	Avanti
0	0	1	Ruota 135 gradi
0	1	0	Ruota 135 gradi
0	1	1	Ruota 90 gradi
1	0	0	Ruota 135 gradi
1	0	1	Ruota 90 gradi
1	1	0	Ruota 135 gradi
1	1	1	Ruota 90 gradi

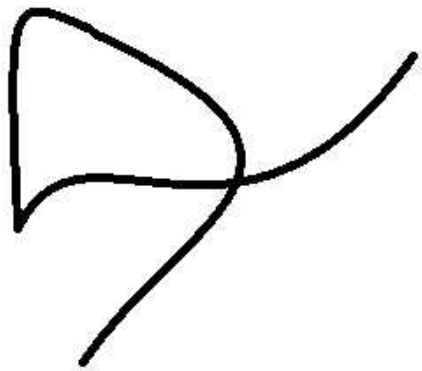
Lizard:

- Pattern base: serpentina;
- rileva ultrasuoni sul fronte o ultrasuoni sul retro o colore nero: ruota di 90 gradi a destra o sinistra.

"Pattern 4 - Serpentina"

Colore nero	Ultrasuoni frontale	Ultrasuoni retro	Comportamento
0	0	0	Serpentina
0	0	1	Ruota 90 gradi
0	1	0	Ruota 90 gradi
0	1	1	Ruota 90 gradi
1	0	0	Ruota 90 gradi
1	0	1	Ruota 90 gradi
1	1	0	Ruota 90 gradi
1	1	1	Ruota 90 gradi

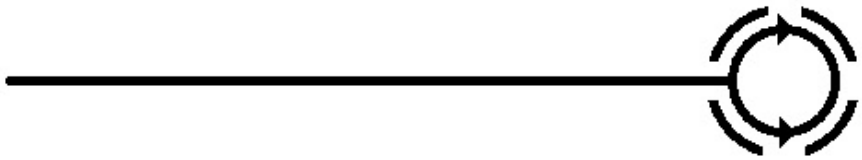
Spock:
Il robot si muove in maniera casuale seguendo alternativamente uno dei pattern precedenti.



"Pattern 5 - Movimento casuale"

Pattern di movimento ("robot autonomo" predatore)
Il robot autonomo nel caso sia predatore esegue un pattern predefinito atto a coprire con i propri sensori la maggior parte del campo.
Non appena uno dei suoi sensori rivela qualcosa, esso smetterà di seguire un pattern e si muoverà seguendo le tracce visive rivelate.

- Pattern base: Forward Research;
- Rileva ultrasuoni sul fronte: avanza verso il bersaglio;
- Rileva ultrasuoni sul retro: ruota a destra o sinistra cercando nuove rilevazioni;
- Rileva colore nero: gira a destra o a sinistra.



"Pattern 6 - Forward Research"

Colore nero	Ultrasuoni frontale	Ultrasuoni retro	Comportamento
0	0	0	Forward Research
0	0	1	Ruota 180 gradi
0	1	0	Avanza
0	1	1	Avanza
1	0	0	Ruota 135 gradi
1	0	1	Ruota 135 gradi
1	1	0	Ruota 135 gradi
1	1	1	Ruota 135 gradi

Modalità

A seconda della modalità selezionata la partita termina in maniera differente.
Nel caso di modalità Quick Match la partita termina quando uno dei due contendenti ha conseguito 2 punti mentre nel caso Survival Mode non appena il "robot autonomo" si aggiudica un round.

Comandi del robot del giocatore

Tramite il joystick (dotato di Force-Feedback) il giocatore può impartire 5 comandi alternativi al robot:

- avanti: fintanto che il joystick rimane inclinato in avanti il robot si muove in tale direzione;
- indietro fintanto che il joystick rimane inclinato all'indietro il robot si muove in tale direzione;
- destra: fintanto che il joystick rimane inclinato verso destra il robot ruota su se stesso verso destra;
- sinistra: fintanto che il joystick rimane inclinato verso sinistra il robot ruota su se stesso verso sinistra;
- fermo: Se il joystick non è inclinato il robot rimane fermo.

Architettura

La connessione tra i tre robot è di tipo master-slave, con il Joystick come master che ha la possibilità di comunicare con entrambi i robot che sono slave e possono comunicare solo col joystick.

La gestione del gioco è affidata al "robot autonomo" che comunica con il giocatore attraverso una connessione bluetooth instaurata con il joystick.
Il joystick comunica le scelte dell'utente al "robot autonomo" e tramite un'altra connessione bluetooth invia i comandi al robot del giocatore.
Il robot del giocatore ha unicamente il compito di ricevere i comandi sotto forma di numeri interi e tradurli in movimenti.

Possibili sviluppi

- Implementare la possibilità di impartire 9 comandi differenti al robot del giocatore con l'aggiunta di: avanti-destra, avanti-sinistra, indietro-destra, indietro-sinistra;
- nella modalità di gioco "Survival Mode" introdurre, come ulteriore grado di difficoltà, la riduzione della velocità del robot controllato dal giocatore con l'avanzare dei livelli, la riduzione/aumento della finestra temporale della fase di caccia o la presenza di malus come comandi invertiti o limitazioni al movimento;
- classifica con high score della modalità di gioco "Survival Mode";
- implementare pattern di movimento diversi in base alla scelta della fase morra per il "robot autonomo" anche in fase predatore.

Easter Eggs

- I nomi dei robot, Sheldonator e Wolodroid, sono un riferimento ai personaggio della serie televisiva The Big Bang Theory, Sheldon Lee Cooper e Howard Joel Wolowitz;
- Quando il robot autonomo rileva con i sensori ad ultrasuoni il robot controllato dal giocatore emette la parola Bazinga, termine utilizzato da Sheldon Cooper per prendersi gioco dei suoi colleghi nel telefilm The Big Bang Theory.

Appendice A: codice joystick "RoboStick"

```
//Versione: RoboStick 1.0.0 22/07/2013
//
//Autori:
//Luca Andrea Raho
//Carlo Alberto Maria Viola
//
//Codice del joystick "Robostick"

//Libreria personalizzata utilizzata nel progetto RPSLSbot
#include "rpsls.h"

//Variabili intere
int mode = 0;
int playerscore = 0;

//Variabili booleane
bool sync = false;
bool breakall = false;

task Menu();

//Manda i comandi al robot del giocatore in base all'inclinazione del joystick.
//
//Per mandare i comandi al robot del giocatore il task utilizza la MAILBOX6 della connessione BLuetooth 1.
//Quando il joystuck è disabilitato viene abilitato un force feedback per mantenerlo in posizione verticale.
task Command(){
	while(true){

		//Controlla se il joystick è abilitato
		if(sync){
			//Avanti
			if(MotorRotationCount(OUT_A)>-MAXAXIS && MotorRotationCount(OUT_A)<-MINAXIS){
				SendRemoteNumber(BT_CONN,MAILBOX6,FORWARD);
			}
			//Indietro
			else if(MotorRotationCount(OUT_A)>MINAXIS && MotorRotationCount(OUT_A)<MAXAXIS){
				SendRemoteNumber(BT_CONN,MAILBOX6,BACKWARD);
			}
			//Destra
			else if(MotorRotationCount(OUT_B)>MINAXIS && MotorRotationCount(OUT_B)<MAXAXIS){
				SendRemoteNumber(BT_CONN,MAILBOX6,TURNRIGHT);
			}
			//Sinistra
			else if(MotorRotationCount(OUT_B)>-MAXAXIS && MotorRotationCount(OUT_B)<-MINAXIS){
				SendRemoteNumber(BT_CONN,MAILBOX6,TURNLEFT);
			}
```

```
                        //Fermo
                        else{
                                SendRemoteNumber(BT_CONN,MAILBOX6,BREAKALL);
                        }
                }
                //Il joystick è disabilitato
                else{
                        //Ferma il robot del giocatore
                        if(breakall){
                                ClearScreen();
                                //Serve per assicurarsi che il messaggio venga inviato.
                                TextOut(1,1,"Ciaociao", DRAW_OPT_NORMAL);
                                Wait(500);
                                SendRemoteNumber(BT_CONN,MAILBOX6,BREAKALL);
                                Wait(100);
                                breakall = false;
                        }
                        //Force feedback
                        else{
                                //Avanti
                                if(MotorRotationCount(OUT_A)>-MAXAXIS && MotorRotationCount(OUT_A)<-MINAXIS){
                                        RotateMotor(OUT_A, 50, 1);
                                }
                                //Indietro
                                else if(MotorRotationCount(OUT_A)>MINAXIS && MotorRotationCount(OUT_A)<MAXAXIS){
                                        RotateMotor(OUT_A, 50, -1);
                                }
                                //Destra
                                else if(MotorRotationCount(OUT_B)>MINAXIS && MotorRotationCount(OUT_B)<MAXAXIS){
                                        RotateMotor(OUT_B, 50, -1);
                                }
                                //Sinistra
                                else if(MotorRotationCount(OUT_B)>-MAXAXIS && MotorRotationCount(OUT_B)<-MINAXIS){
                                        RotateMotor(OUT_B, 50, 1);
                                }
                        }
                }

        }
}//Command

//Gestisce le fasi di gioco.
//
//Utilizza la MAILBOX3 Bluetooth in ingresso per ricevere il messaggio dell'avvenuto riposizionamento dei robot.
//Utilizza la MAILBOX8 della connessione Bluetooth 2 in uscita per comunicare al robot autonomo la scelta del giocatore durante la fase "Morra".
//Utilizza la MAILBOX4 Bluetooth in ingresso per ricevere l'esito della fase "Morra".
//Utilizza la MAILBOX5 Bluetooth in ingresso per ricevere l'esito del match.
```

```
task Core(){

	bool inB = false;
	int playerchoice;
	int robotwin = 4;
	int numin = 0;

Restart:
	inB = false;

	ClearScreen();
	TextOut(TEXTLINE_1,10,"Riposizionare i",DRAW_OPT_NORMAL);
	TextOut(TEXTLINE_1,TEXTLINE_1,"robot",DRAW_OPT_NORMAL);

	//Attendi il riposizionamento dei robot
	while(!inB){
		ReceiveRemoteBool(MAILBOX3, true, inB);
		Wait(100);
	}

Playerchoice:
	robotwin = 4;

	//Scelta del giocatore
	playerchoice = PlayerChoice();
	SendRemoteNumber(BT_CONN2,MAILBOX8,playerchoice);
	Wait(100);

	//Attendi esito Carta, Forbici, Sasso, Lizard, Spock
	while(robotwin!=1 && robotwin!=2 && robotwin!=3){
		ReceiveRemoteNumber(MAILBOX4, true, robotwin);
		Wait(100);
	}

	ClearScreen();

	//Mostra l'esito di Sasso, Carta, Forbici, Lizars, Spock
	switch(robotwin){
	case ROBOTWIN:
		TextOut(TEXTLINE_1, TEXTLINE_1, "Prey" );
		break;

	case PLAYERWIN:
		TextOut(TEXTLINE_1, TEXTLINE_1, "Predator" );
		break;

	case DRAW:
		TextOut(TEXTLINE_1, TEXTLINE_1, "Draw" );
```

```
		Wait(TEXTOUT);
		goto Playerchoice;
		break;
}

//Abilita Joystick
sync = true;

//Attendi termine partita
numin = 0;
while(numin != PLAYERGAME && numin != PLAYERROUND && numin != ROBOTGAME && numin != ROBOTROUND){
		ReceiveRemoteNumber(MAILBOX5, true, numin);
		Wait(100);
}

//Disabilita Joystick
sync = false;

//Ferma il robot del giocatore
breakall = true;
SendRemoteNumber(BT_CONN,MAILBOX6,BREAKALL);
Wait(100);

ClearScreen();

//Mostra se la partita termina o meno
switch(numin){

		//Il giocatore ha vinto la partita
case PLAYERGAME:
		TextOut(TEXTLINE_1,TEXTLINE_1,"You win",DRAW_OPT_NORMAL);
		Wait(TEXTOUT);
		PlayerScore(mode, playerscore);
		playerscore = 0;
		ExitTo(Menu);
		break;

		//Il giocatore ha vinto il round
case PLAYERROUND:
		TextOut(TEXTLINE_1,TEXTLINE_1,"Next Round",DRAW_OPT_NORMAL);
		Wait(TEXTOUT);
		playerscore++;
		goto Restart;
		break;

		//Il robot autonomo ha vinto la partita
case ROBOTGAME:
		TextOut(TEXTLINE_1,TEXTLINE_1,"You lose",DRAW_OPT_NORMAL);
```

```
                Wait(TEXTOUT);
                PlayerScore(mode, playerscore);
                playerscore = 0;
                ExitTo(Menu);
                break;

                //Il robot autonomo ha vinto il round
        case ROBOTROUND:
                TextOut(TEXTLINE_1,TEXTLINE_1,"Next Round",DRAW_OPT_NORMAL);
                Wait(TEXTOUT);
                playerscore++;
                goto Restart;
                break;
        }
}//Core

//Menù del joystick.
//
//Utilizza la MAILBOX7 della connessione Bluetooth 2 in uscita per comunicare al robot autonomo la modalità di gioco.
task Menu(){
        mode = FMenu();

        Precedes(Core);
}//Menu

//main
task main(){
        //Imposta il tasto EXIT in modo da terminare il programma solo se premuto per almeno 2 secondi
        #ifdef __ENHANCED_FIRMWARE
        SetLongAbort(true);
        // is equivalent to
        SetAbortFlag(BTNSTATE_LONG_PRESSED_EV);
        #endif

        //Al termine del task fa partire i task Menu e Command
        Precedes(Menu, Command);
}//main
```

Appendice B: codice robot del giocatore "WoloDroid"

```
//Versione: WoloDroid 1.0.0 22/07/2013
//
//Autori:
//Luca Andrea Raho
//Carlo Alberto Maria Viola
//
//Codice del robot del giocatore "Wolodroid"

//Libreria personalizzata utilizzata nel progetto RPSLSbot
#include "rpsls.h"

//Variabili booleane
bool pause = false;
bool black = false;

//Spegne il robot al termine del programma.
//
//Il task rimane attivo in background in ascolto sulla MAILBOX9 della connessione Bluetooth.
//Quando riceve un messaggio true spegne il robot.
task End(){
	bool end = false;

	while(true){
		ReceiveRemoteBool(MAILBOX9, true, end);
		Wait(100);
		if(end){
			PowerDown();
		}
	}
}//End

//Sensore di colore.
//
//Se rileva il colore nero setta la variabile booleana black a true, in qualunque altro caso la setta a false.
task Color(){
	while(true){
		switch(Sensor(COLOR_VALUE)){
		case BLACK:
			black = true;
			break;

		default:
			black = false;
			break;
		}
	}
```

```
}//Color

//Fa retrocedere il robot e lo mette in pausa dopo aver assestato un colpo.
//
//Quando il sensore di tocco viene premuto fa retrocedere il robot per 500ms e ne impedisce i movimenti per 3000ms.
task Pause(){
	while(true){
		if(Sensor(IN_1)){
			//Disabilita i movimenti del robot
			pause = true;
			Acquire(sem);
			Off(OUT_BC);
			OnRev(OUT_BC, MPOWER);
			Wait(500);
			Off(OUT_BC);
			Wait(3000);
			//Abilita i movimenti del robot
			Release(sem);
			pause = false;

		}
	}
}//Pause

//Movimenti del robot.
//
//Il task rimane in ascolto sulla MAILBOX6 della connessione Bluetooth in attesa dei comandi del joystick e li traduce in movimenti del robot.
task Core(){

	int inI = 0;
	int state = 0;

	while(true){
		//Riceve il comando dal joystick
		ReceiveRemoteNumber(MAILBOX6, true, inI);
		Wait(100);

		//Controlla se il robot ha colpito una parte non valida del robot autonomo
		if(!pause){
			//Controlla se il robot sta cercando di sorpassare il bordo del campo di gioco
			if(!black){
				switch(inI){
					//Avanti
				case FORWARD:
					Acquire(sem);
					Off(OUT_BC);
					OnFwd(OUT_BC, MPOWER);
```

```
                state = FORWARD;
                Release(sem);
                break;

                //Indietro
        case BACKWARD:
                Acquire(sem);
                Off(OUT_BC);
                OnRev(OUT_BC, MPOWER);
                state = BACKWARD;
                Release(sem);
                break;

                //Destra
        case TURNRIGHT:
                Acquire(sem);
                Off(OUT_B);
                OnFwd(OUT_C, MPOWER);
                state = TURNRIGHT;
                Release(sem);
                break;

                //Sinistra
        case TURNLEFT:
                Acquire(sem);
                Off(OUT_C);
                OnFwd(OUT_B, MPOWER);
                state = TURNLEFT;
                Release(sem);
                break;

                //Fermo
        case BREAKALL:
                Acquire(sem);
                Off(OUT_BC);
                state = BREAKALL;
                Release(sem);
                break;
        }
        //Il robot sta provando a uscire dal campo di gioco
}else{
        switch(state){
                //Avanti
        case FORWARD:
                Acquire(sem);
                Off(OUT_BC);
                OnRev(OUT_BC, MPOWER);
                Wait(500);
```

```
                                        Off(OUT_BC);
                                        Release(sem);
                                        break;

                                        //Indietro
                                case BACKWARD:
                                        Acquire(sem);
                                        Off(OUT_BC);
                                        OnFwd(OUT_BC, MPOWER);
                                        Wait(500);
                                        Off(OUT_BC);
                                        Release(sem);
                                        break;

                                        //Destra
                                case TURNRIGHT:
                                        Acquire(sem);
                                        Off(OUT_BC);
                                        OnFwd(OUT_B, MPOWER);
                                        Wait(500);
                                        Off(OUT_BC);
                                        Release(sem);
                                        break;

                                        //Sinistra
                                case TURNLEFT:
                                        Acquire(sem);
                                        Off(OUT_BC);
                                        OnFwd(OUT_C, MPOWER);
                                        Wait(500);
                                        Off(OUT_BC);
                                        Release(sem);
                                        break;
                                }
                        }
                        //resetta il comando
                        inI = 0;
                }
        }
}//Core

//Main
task main(){
        //Inizializza i sensori
        SetSensor(IN_1, SENSOR_TOUCH);
        SetSensorColorFull(IN_2);

        //Al termine del task corrente fa partire i task Core, Pause, Color e End
```

```
        Precedes(Core, Pause, Color, End);
}//main
```

Appendice C: codice robot autonomo "Sheldonator"

```
//Versione: Sheldonator 1.0.0 22/07/2013
//
//Autori:
//Luca Andrea Raho
//Carlo Alberto Maria Viola
//
//Codice del robot autonomo "Sheldonator"

//Libreria personalizzata utilizzata nel progetto RPSLSbot
#include "rpsls.h"

//Variabili dei sensori a ultrasuoni
bool rus = false;
bool fus = false;

//Variabili del sensore di tocco
bool touch = false;

//Variabili del sensore di luce
bool black = false;
bool white = false;

//Variabili di timeout
bool timeout = false;

//Variabili di sincronizzazione
bool sync = false;
bool syncb = false;

//Variabili di gioco
int robotchoice;
int robotscore=0;
int playerchoice=0;
int playerscore=0;
int robotwin=0;
int nplayers=1;
int playerlife=1;
int mode;

//Variabili del menù
int menu = 0;

//Variabili dei pattern di movimento
int i = 0;
int j = 0;
int k = 0;
```

```
int s;

//Variabili dei messaggi Bluetooth
bool inB;

//Task
task Menu();
task Core();

//Funzioni
bool QuickMatch();
bool Survival();

//Produce il suono "Bazinga".
//
//Riproduce il file "bazinga.rso" quando il sensore ad ultrasuoni frontale rileva una minaccia.
task Bazinga(){
	while(true){
		if(fus){
			PlayFile("bazinga.rso");
			Wait(1000);
		}
	}
}//Bazinga

//Ultrasuoni sul retro.
//
//Setta a true la variabil booleana rus quando rileva una minaccia.
task RearSonar(){

	if(robotwin == PLAYERWIN){

		while(true){

			if(SensorUS(SONAR_REAR_VALUE)<RNEAR_THRESHOLD){
				rus = true;
			}
			else{
				rus = false;
			}

		}

	}
	else{

		while(true){
```

```
                        if(SensorUS(SONAR_REAR_VALUE)<RPREDATOR_THRESHOLD){
                                rus = true;
                        }
                        else{
                                rus = false;
                        }

                }

        }

}//RearSonar

//Ultrasuoni sul fronte.
//
//Setta a true la variabil booleana fus quando rileva una minaccia.
task FrontSonar(){

        if(robotwin == PLAYERWIN){

                while(true){

                        if(SensorUS(SONAR_FRONT_VALUE)<FNEAR_THRESHOLD){
                                fus = true;
                        }
                        else{
                                fus = false;
                        }

                }
        }
        else{

                while(true){

                        if(SensorUS(SONAR_FRONT_VALUE)<FPREDATOR_THRESHOLD){
                                fus = true;
                        }
                        else{
                                fus = false;
                        }

                }

        }

}//FrontSonar
```

```
//Sensore di luce.
//
//Viene utilizzto in alternativa del sensore di colore.
//Setta la variabile booleana black a true e white a false quando rileva nero,
//il contrario quando negli altri casi.
task Light(){
	while(true){
		if(Sensor(COLOR_VALUE)<=LIGHTBLACK){
			black = true;
			white = false;



		}else{
			black = false;
			white=true;



		}
	}
}//Light

//Sensore di tocco.
//
//Setta la variabile booleana touch a true quando viene premuto.
task Touch(){

	while(!touch){

		if(TOUCH == true){
			touch = true;
			PlayTone(1000,1000);
		}
		else{
			touch = false;
		}

	}

}//Touch

//Stabilisce se il robot sia la preda o il predatore
//e setta le variabili necessarie alla navigazione del robot autonomo.
task PreyOrPredator(){

	i = 0;
	//Stabilisce il comportamento del robot in modalità preda-Spock
```

```
		j = Random(2);
		k = 0;
		//Decide da che parte ruota il robot autonomo
		s=Random(4)+1;

		//Il robot autonomo ha vinto la fase "Morra" e diventa predatore
		if(robotwin == ROBOTWIN){
			ClearScreen();
			TextOut(0, 0, "Predator", DRAW_OPT_NORMAL);
			while(!touch && !timeout){
				Predator(i,j,black,fus,rus);
			}

		}
		//Il robot autonomo ha perso la fase "Morra" e diventa preda
		else if(robotwin == PLAYERWIN){

			while(!touch && !timeout){
				Prey(robotchoice, s, i, j, black, fus, rus);
			}

		}
		//Il robot autonomo ha terminato di muoversi
		syncb = true;
		Off(OUT_BC);

}//PreyOrPredator

//Setta la variabile booleana timeout a true quando passano 60 secondi.
task Timeout(){
		//Attende il tempo del timeout meno 10000ms
		Wait(TIMEOUT-10000);

		//Emette un suono ogni secondo per gli ultimi 10 secondi del timeout
		for(int i = 0; i<9; i++){
			Wait(1000);
			PlayTone(500,500);
		}
		Wait(1000);
		timeout = true;
		PlayTone(1000,1000); //Timeout
}//Timeout

//Controlla il termine e l'esito del round.
//
//Avvisa il giocatore dell'esito del round tramite la MAILBOX5 della connessione Bluetooth in uscita.
task Control(){
```

```
	int result;

	//Quick Match
	if(mode == QUICKMATCH){

		result = QuickMatch();

		if(result == PLAYERGAME || result == ROBOTGAME){
			//Game Over
			SendResponseNumber(MAILBOX5, result);
			Wait(100);
			ExitTo(Menu);
		}
		else{
			//Next Round
			SendResponseNumber(MAILBOX5, result);
			Wait(100);
			ExitTo(Core);
		}

	}
	//Survival Mode
	else if(mode == SURVIVAL){

		result = Survival();

		if(result == ROBOTGAME){
			//Game Over
			SendResponseNumber(MAILBOX5, result);
			Wait(100);
			ExitTo(Menu);
		}
		else{
			//Next Round
			SendResponseNumber(MAILBOX5, result);
			Wait(100);
			ExitTo(Core);
		}

	}

}//Control

//Gestisce le fasi di gioco.
//
//
task Core(){
```

```
	//Resetta le variabili di gioco
	timeout = false;
	touch = false;
	robotwin = DRAW;

	//Attendi il riposizionamento dei robot
	until(Reposition()){}
	//Avverti il giocatore dell'avvenuto riposizionamento
	SendResponseBool(MAILBOX3, true);
	Wait(100);

	//Fase "Morra"
RockPaperScissorsLizardSpock:
	//Cicla fintanto che l'esito è un pareggio
	while(robotwin == DRAW){
		//Resetta la scelta del giocatore
		playerchoice = 0;
		ClearScreen();

		//Scelta del robot autonomo
		robotchoice = (Random(5)+1);

		switch(robotchoice){
		case ROCK:
			TextOut(0, 0, "Robot Rock" );
			break;
		case PAPER:
			TextOut(0, 0, "Robot Paper" );
			break;
		case SCISSORS:
			TextOut(0, 0, "Robot Scissors" );
			break;
		case LIZARD:
			TextOut(0, 0, "Robot Lizard" );
			break;
		case SPOCK:
			TextOut(0, 0, "Robot Spock" );
			break;
		}
		Wait(1000);

		//Attendi scelta del giocatore
		while(playerchoice!=ROCK && playerchoice!=PAPER && playerchoice!=SCISSORS && playerchoice!=LIZARD &&
playerchoice!=SPOCK){
			ReceiveRemoteNumber(MAILBOX8, true, playerchoice);
			Wait(100);
		}
```

```
                //Determina l'esito della fase "Morra"
                sync = RockPaperScissors(robotchoice, playerchoice, robotwin);
                //Attendi l'esito di sasso, carta, forbici, Lizard, Spock
                while(!sync){}
                sync = 0;
                ClearScreen();
                //Invia l'esito della fase "Morra" al giocatore
                SendResponseNumber(MAILBOX4, robotwin);
                Wait(100);

        }
        PlayTone(500,500); //Comincia la fase "Caccia"
        //Al termine del task corrente fa partire i task Control, Timeout, PreyOrPredator, Touch e Bazinga
        Precedes(Control, Timeout, PreyOrPredator, Touch, Bazinga);
}//Robodroid

//Controlla il termine di un round di una partita in modalità Quick Match.
int QuickMatch(){

        //Attende che uno dei due robot colpisca l'avversario o che scatti il timeout
        while(!touch && !timeout){}

        //Interrompe il timeout
        stop Timeout;
        stop Bazinga;

        //Attende che il robot autonomo termini il movimento
        while(!syncb){}
        syncb = false;

        //Il robot autonomo è il predatore
        if(robotwin == ROBOTWIN){
                //Il round è terminato a causa del timeout
                if(timeout){

                        //Aumenta il punteggio del giocatore
                        playerscore++;

                        //Controlla se il giocatore ha raggiunto 2 punti
                        if(playerscore>=SCORE_THRESHOLD){
                                //PlayTone(500,500); //Player wins
                                return PLAYERGAME;
                        }
                        else{
                                //New Match
                                return PLAYERROUND;
                        }
                }
```

```
		//Il round è terminato in tempo
		else {

			//Aumenta il punteggio del robot
			robotscore++;

			//Controlla se il robot ha raggiunto 2 punti
			if(robotscore>=SCORE_THRESHOLD){
				//PlayTone(500,500); //Robot wins
				return ROBOTGAME;
			}
			else{
				//New Match
				return ROBOTROUND;
			}
		}
}
//Il robot autonomo è la preda
else if(robotwin == PLAYERWIN){
		if(timeout){

			//Aumenta il punteggio del robot
			robotscore++;

			//Controlla se il robot ha raggiunto 2 punti
			if(robotscore>=SCORE_THRESHOLD){
				//PlayTone(500,500); //Robot wins
				return ROBOTGAME;
			}
			else{
				//New Match
				return ROBOTROUND;
			}
		}
		else{

			//Aumenta il punteggio del giocatore
			playerscore++;

			//Controlla se il giocatore ha raggiunto 2 punti
			if(playerscore>=SCORE_THRESHOLD){
				//PlayTone(500,500); //Player wins
				return PLAYERGAME;
			}
			else{
				//New Match
				return PLAYERROUND;
			}
```

```
                }

        }


}//QuickMatch

//Controlla il termine di un round di una partita in modalità Sopravvivenza.
int Survival(){

        //Attende che uno dei due robot colpisca l'avversario o che scatti il timeout
        while(!touch && !timeout){}

        //Interrompe il timeout
        stop Timeout;
        stop Bazinga;

        //Attende che il robot autonomo termini il movimento
        while(!syncb){}
        syncb = false;

        //Il robot autonomo è il predatore
        if(robotwin == ROBOTWIN){
                //Il round è terminato a causa del timeout
                if(timeout){

                        //playerscore++;

                        //New Match
                        return PLAYERROUND;
                }
                //Il round è terminato in tempo
                else {
                        //Robot wins

                        return ROBOTGAME;
                }

        }
        //Il robot autonomo è la preda
        else if(robotwin == PLAYERWIN){
                //Il round è terminato a causa del timeout
                if(timeout){
                        //Robot wins (secondo Tigro)

                        return ROBOTGAME;
                }
                //Il round è terminato in tempo
```

```
                else{

                        //playerscore++;

                        //New Match
                        return PLAYERROUND;

                }

        }

}//Survival



//Fa partire la modalità di gioco in base alla scelta del giocatore.
//
//Riceve la scelta del giocatore sulla MAILBOX7 della connessione Bluetooth in ingresso.
task Menu(){

        mode = 0;
        robotscore = 0;
        playerscore = 0;

        ClearScreen();

        //Attende scelta del giocatore
        while(mode!=QUICKMATCH && mode!=SURVIVAL && mode!=3 && mode!=EXIT){
                TextOut(0, 0, "Attendo Messaggio", DRAW_OPT_NORMAL);
                ReceiveRemoteNumber(MAILBOX7, true, mode);
                Wait(100);
        }
        //Se il giocatore digita Exit spegne il robot
        if(mode == EXIT){
                PowerDown();
        }

        //Al termine del task corrente fa partire il task Robodroid
        Precedes(Core);
}//Menu

//main
task main(){
        //Inizializza i sensori
        SetSensorTouch(IN_1);
        SetSensorLight(IN_2);
        SetSensorUltrasonic(IN_3);
        SetSensorUltrasonic(IN_4);
```

```
        //Al termine del tak corrente fa partire i task FrontSonar, RearSonar, Light e Menu
        Precedes(FrontSonar, RearSonar, Light, Menu);
}//main
```

Appendice D: libreria "rpsls.h"

Le costanti e le funzioni utilizzate nel codice dei tre robot sono state riunite nella libreria "rpsls.h" per facilitare un eventuale riutilizzo futuro.

```
//DEFINE SENSORS
#define TOUCH                   SENSOR_1
#define COLOR                   SENSOR_2
#define COLOR_VALUE             IN_2
#define SONAR_REAR              SENSOR_4
#define SONAR_REAR_VALUE        IN_4
#define SONAR_FRONT             SENSOR_3
#define SONAR_FRONT_VALUE       IN_3

//DEFINE threshold
#define RNEAR_THRESHOLD         20
#define FNEAR_THRESHOLD         35

#define RPREDATOR_THRESHOLD     150
#define FPREDATOR_THRESHOLD     150

//Tempo
#define TIMEOUT         60000
#define SYNCTIME        200
#define TEXTOUT         1000

//Colori
#define BLACK           1
#define BLUE            2
#define GREEN           3
#define YELLOW          4
#define RED             5
#define WHITE           6
#define LIGHTBLACK      35

//Movimento
#define PPOWER                  75
#define MAXPOWER        75
#define MTIME           100
#define MCYCLE          30
#define ROTATE45        500
#define ROTATE90        600
#define ROTATE135       1300
#define ROTATE180       1500
#define MPOWER          50
#define RIGHT           0
#define LEFT            1
```

```
//Segni
#define ROCK 1
#define PAPER 2
#define SCISSORS 3
#define LIZARD 4
#define SPOCK 5

//Morra
#define ROBOTWIN 1
#define PLAYERWIN 2
#define DRAW 3

//Connessioni BLuetooth
#define BT_CONN0 0
#define BT_CONN 1
#define BT_CONN2 2

//Comandi Joystick
#define FORWARD 1
#define BACKWARD 2
#define TURNRIGHT 3
#define TURNLEFT 4
#define ROTATERIGHT 5
#define ROTATELEFT 6
#define BREAKALL 7
#define ATTACK 8

//Joystick
#define MINAXIS 30
#define MAXAXIS 70

//Modalita
#define QUICKMATCH 1
#define SURVIVAL 2
#define EXIT 4

//Esito partita
#define GAMEOVER 1
#define NEXTROUND 2
#define SCORE_THRESHOLD 2
#define PLAYERROUND 1
#define ROBOTROUND 2
#define PLAYERGAME 3
#define ROBOTGAME 4

mutex sem;

//FUNZIONI JOYSTICK
```

```
//Menu
int FMenu(){
	int menu = 0;
	int mode;

	ClearScreen();

	//Robodroid
Robo:

	while(menu == 0){
		Wait(SYNCTIME);
		ClearScreen();
		TextOut(TEXTLINE_1, TEXTLINE_1, "Robodroid", DRAW_OPT_NORMAL);

		if(ButtonPressed(BTNEXIT, true)){
			//Shut Down the Robot
			menu = 1;

			//Exit
ShutDownTheRobot:

			ClearScreen();
			TextOut(TEXTLINE_1, TEXTLINE_1, "Exit" );

			while(menu == 1){
				Wait(SYNCTIME);
				if(ButtonPressed(BTNEXIT, true)){
					menu = 0;
					goto Robo;
				}
				else if(ButtonPressed(BTNCENTER, true)){
					SendRemoteNumber(BT_CONN2,MAILBOX7,4);
					Wait(1000);
					SendRemoteBool(BT_CONN,MAILBOX9,true);
					Wait(1000);
					PowerDown();
				}
				else if(ButtonPressed(BTNRIGHT, true)||ButtonPressed(BTNLEFT, true)){

					//Cancel
Cancel:
					menu = 2;
					ClearScreen();
					TextOut(TEXTLINE_1, TEXTLINE_1, "Cancel" , DRAW_OPT_NORMAL);

					while(menu == 2){
```

```
                                        Wait(SYNCTIME);
                                        if(ButtonPressed(BTNCENTER, true)||ButtonPressed(BTNEXIT, true)){
                                                menu = 0;
                                                goto Robo;
                                        }
                                        else if(ButtonPressed(BTNRIGHT, true)||ButtonPressed(BTNLEFT, true)){
                                                menu = 1;
                                                goto ShutDownTheRobot;
                                        }
                                }
                        }
                        else if(ButtonPressed(BTNEXIT, true)){
                                menu = 0;
                                goto Robo;
                        }
                }
        }
        else if(ButtonPressed(BTNCENTER, true)){
                //Quick Match
                menu = 1;

                //Quick Match
Quick:
                ClearScreen();
                TextOut(TEXTLINE_1, TEXTLINE_1, "Quick Match  " , DRAW_OPT_NORMAL);

                while(menu == 1){
                        Wait(SYNCTIME);
                        if(ButtonPressed(BTNCENTER, true)){
                                //1 Player
                                mode = QUICKMATCH;
                                menu = 2;

                                //Single Player
Player:
                                ClearScreen();
                                TextOut(TEXTLINE_1, TEXTLINE_1, "Single Player" , DRAW_OPT_NORMAL);

                                while(menu == 2){
                                        Wait(SYNCTIME);
                                        if(ButtonPressed(BTNCENTER, true)){
                                                //Start 1 Player Robodroid
                                                menu = 4;
                                                SendRemoteNumber(BT_CONN2,MAILBOX7,QUICKMATCH);
//Robodroid/QuickMatch/SinglePlayer


                                        }
```

```
                                        else if(ButtonPressed(BTNEXIT, true)){
                                                menu = 1;
                                                goto Quick;
                                        }
                                }
                        }
                        else if(ButtonPressed(BTNRIGHT, true)||ButtonPressed(BTNLEFT, true)){
                                //Surival Mode
                                menu = 2;

                                //Survival Mode
SurvivalMode:
                                ClearScreen();
                                TextOut(TEXTLINE_1, TEXTLINE_1, "Survival Mode" , DRAW_OPT_NORMAL);

                                while(menu == 2){
                                        Wait(SYNCTIME);
                                        if(ButtonPressed(BTNCENTER, true)){
                                                //Start Survival Mode
                                                mode = SURVIVAL;
                                                menu = 4;
                                                SendRemoteNumber(BT_CONN2,MAILBOX7,SURVIVAL);
//Robodroid/SurvivalMode

                                        }
                                        else if(ButtonPressed(BTNRIGHT, true)||ButtonPressed(BTNLEFT, true)){
                                                menu = 1;
                                                goto Quick;
                                        }
                                        else if(ButtonPressed(BTNEXIT, true)){
                                                menu = 0;
                                                goto Robo;
                                        }
                                }
                        }
                        else if(ButtonPressed(BTNEXIT, true)){
                                menu = 0;
                                goto Robo;
                        }

                }
        }

    }
    return mode;
}
```

```
//Scelta del giocatore
int PlayerChoice(){
	ClearScreen();
	int choice = ROCK;
	while(true){
		Wait(SYNCTIME);
		switch(choice){

		case ROCK:
			TextOut(0, 0, "Rock     ", DRAW_OPT_NORMAL);
			while(choice == ROCK){
				Wait(SYNCTIME);
				if(ButtonPressed(BTNCENTER, true)){return ROCK;}
				else if(ButtonPressed(BTNRIGHT, true)){choice++;}
				else if(ButtonPressed(BTNLEFT, true)){choice=SPOCK;}

			}
			break;

		case PAPER:
			TextOut(0, 0, "Paper    ", DRAW_OPT_NORMAL);
			while(choice == PAPER){
				Wait(SYNCTIME);
				if(ButtonPressed(BTNCENTER, true)){return PAPER;}
				else if(ButtonPressed(BTNRIGHT, true)){choice++;}
				else if(ButtonPressed(BTNLEFT, true)){choice--;}
			}
			break;

		case SCISSORS:
			TextOut(0, 0, "Scissors", DRAW_OPT_NORMAL);
			while(choice == SCISSORS){
				Wait(SYNCTIME);
				if(ButtonPressed(BTNCENTER, true)){return SCISSORS;}
				else if(ButtonPressed(BTNRIGHT, true)){choice++;}
				else if(ButtonPressed(BTNLEFT, true)){choice--;}
			}
			break;

		case LIZARD:
			TextOut(0, 0, "Lizard  ", DRAW_OPT_NORMAL);
			while(choice == LIZARD){
				Wait(SYNCTIME);
				if(ButtonPressed(BTNCENTER, true)){return LIZARD;}
				else if(ButtonPressed(BTNRIGHT, true)){choice++;}
				else if(ButtonPressed(BTNLEFT, true)){choice--;}
			}
			break;
```

```
                case SPOCK:
                        TextOut(0, 0, "Spock    ", DRAW_OPT_NORMAL);
                        while(choice == SPOCK){
                                Wait(SYNCTIME);
                                if(ButtonPressed(BTNCENTER, true)){return SPOCK;}
                                else if(ButtonPressed(BTNRIGHT, true)){choice=ROCK;}
                                else if(ButtonPressed(BTNLEFT, true)){choice--;}
                        }
                        break;

                }
        }
}

//Mostra il punteggio del giocatore in modalità Sopravvivenza.
//
//Al termine della partita se la modalità è sopravvivenza stampa a schermo il numero di round consecutivi vinti dal giocatore.
sub PlayerScore(int mode, int playerscore){
        if(mode == SURVIVAL){
                ClearScreen();
                TextOut(TEXTLINE_1, TEXTLINE_1, "You last " + NumToStr(playerscore) + " rounds", DRAW_OPT_NORMAL);
                Wait(TEXTOUT);
        }
}//PlayerScore

//FUNZIONI ROBOT AUTONOMO

//Esito Carta, Forbici, Sasso, Lizard, Spock
bool RockPaperScissors(int robotchoice, int playerchoice, int &robotwin){

        //Scelta del robot autonomo
        switch(robotchoice){
        case ROCK:
                //Scelta del giocatore
                switch(playerchoice){
                case ROCK:
                        robotwin = DRAW;
                        break;

                case PAPER:
                        robotwin = PLAYERWIN;
                        break;

                case SCISSORS:
                        robotwin = ROBOTWIN;
                        break;
```

```
            case LIZARD:
                        robotwin = ROBOTWIN;
                        break;

            case SPOCK:
                        robotwin = PLAYERWIN;
                        break;
            }
            break;

case PAPER:
            //Scelta del giocatore
            switch(playerchoice){
            case ROCK:
                        robotwin = ROBOTWIN;
                        break;

            case PAPER:
                        robotwin = DRAW;
                        break;

            case SCISSORS:
                        robotwin = PLAYERWIN;
                        break;

            case LIZARD:
                        robotwin = PLAYERWIN;
                        break;

            case SPOCK:
                        robotwin = ROBOTWIN;
                        break;
            }
            break;

case SCISSORS:
            //Scelta del giocatore
            switch(playerchoice){
            case ROCK:
                        robotwin = PLAYERWIN;
                        break;

            case PAPER:
                        robotwin = ROBOTWIN;
                        break;

            case SCISSORS:
                        robotwin = DRAW;
```

```
                    break;

            case LIZARD:
                    robotwin = ROBOTWIN;
                    break;

            case SPOCK:
                    robotwin = PLAYERWIN;
                    break;
            }
            break;

case LIZARD:
            //Scelta del giocatore
            switch(playerchoice){
            case ROCK:
                    robotwin = PLAYERWIN;
                    break;

            case PAPER:
                    robotwin = ROBOTWIN;
                    break;

            case SCISSORS:
                    robotwin = PLAYERWIN;
                    break;

            case LIZARD:
                    robotwin = DRAW;
                    break;

            case SPOCK:
                    robotwin = ROBOTWIN;
                    break;
            }
            break;

case SPOCK:
            //Scelta del giocatore
            switch(playerchoice){
            case ROCK:
                    robotwin = ROBOTWIN;
                    break;

            case PAPER:
                    robotwin = PLAYERWIN;
                    break;
```

```
                case SCISSORS:
                        robotwin = ROBOTWIN;
                        break;

                case LIZARD:
                        robotwin = PLAYERWIN;
                        break;

                case SPOCK:
                        robotwin = DRAW;
                        break;
                }
                break;
        }

        return true;
}



//Comportamento del robot in modalita  predatore.
sub Predator(int &i, int j, bool black, bool fus, bool rus){
        //Colore nero
        if(black){

                switch(j){

                        //Gira a destra
                case RIGHT:
                        Acquire(sem);
                        Off(OUT_BC);
                        OnRev(OUT_C, PPOWER);
                        Wait(ROTATE135);
                        Off(OUT_BC);
                        OnFwd(OUT_BC, PPOWER);
                        Wait(1000);
                        Release(sem);
                        break;

                        //Gira a sinistra
                case LEFT:
                        Acquire(sem);
                        Off(OUT_BC);
                        OnRev(OUT_B, PPOWER);
                        Wait(ROTATE135);
                        OnFwd(OUT_BC, PPOWER);
                        Wait(1000);
                        Release(sem);
```

```
                break;

        }
}

//Ultrasuoni sul fronte
else if(fus){
        Acquire(sem);
        OnFwd(OUT_BC, PPOWER);
        Wait(MTIME);
        Release(sem);
}

//Ultrasuoni sul retro
else if(rus) {

        switch(j){

                //Gira a destra
        case RIGHT:
                Acquire(sem);
                Off(OUT_BC);
                OnFwd(OUT_C, PPOWER);
                Wait(ROTATE180);
                Release(sem);
                break;

                //Gira a sinistra
        case LEFT:
                Acquire(sem);
                Off(OUT_BC);
                OnFwd(OUT_B, PPOWER);
                Wait(ROTATE180);
                Release(sem);
                break;
        }
}
//Pattern base
else{
        ClearScreen();
        //Avanza
        if(0<=i && i<30){
                Acquire(sem);
                Off(OUT_BC);
                OnFwd(OUT_BC, PPOWER);
                Wait(MTIME);
                Release(sem);
                i++;
```

```
                }
                //Gira su se stesso in cerca della preda
                else if(30<=i && i<70){
                        Acquire(sem);
                        Off(OUT_BC);
                        OnFwd(OUT_C, PPOWER);
                        Wait(MTIME);
                        Release(sem);
                        i++;
                }
                else{
                        Acquire(sem);
                        Off(OUT_BC);
                        OnFwd(OUT_C, PPOWER);
                        Wait(MTIME);
                        Release(sem);
                        i=0;
                }

        }

}//Predator

//Comportamento del robot in modalità preda-sasso
sub Rock(int &i, int j, bool black, bool fus, bool rus){

        //Colore nero e ultrasuoni sul retro
        if((black) && rus){
                Acquire(sem);
                Off(OUT_BC);
                Release(sem);
        }

        //Ultrasuoni sul fronte e sul retro
        else if(fus && rus){

                switch(j){
                        //Gira a destra
                case RIGHT:
                        Acquire(sem);
                        Off(OUT_BC);
                        OnFwd(OUT_C, MAXPOWER);
                        Wait(ROTATE90);
                        Release(sem);
                        break;

                        //Gira a sinistra
                case LEFT:
```

```
                Acquire(sem);
                Off(OUT_BC);
                OnFwd(OUT_B, MAXPOWER);
                Wait(ROTATE90);
                Release(sem);
                break;
        }

}

//Colore nero o ultrasuoni frontale
else if(black||fus){


        switch(j){

                //Gira a destra
        case RIGHT:
                Acquire(sem);
                Off(OUT_BC);
                OnFwd(OUT_C, MAXPOWER);
                Wait(ROTATE180);
                Release(sem);
                break;

                //Gira a sinistra
        case LEFT:
                Acquire(sem);
                Off(OUT_BC);
                OnFwd(OUT_B, MAXPOWER);
                Wait(ROTATE180);
                Release(sem);
                break;
        }
}

//Ultrasuoni sul retro
else if(rus){
        Acquire(sem);
        Off(OUT_BC);
        OnFwd(OUT_BC, MAXPOWER);

        Release(sem);
}
//Pattern base
else{
        //Fermo
        Acquire(sem);
```

```
                Off(OUT_BC);
                Release(sem);
        }

}//Rock

//Comportamento del robot in modalità preda-carta
sub Paper(int &i, int j, bool black, bool fus, bool rus){
        if(rus){

                switch(j){
                        //Gira a destra
                case RIGHT:
                        Acquire(sem);
                        Off(OUT_B);
                        OnRev(OUT_C, MAXPOWER);
                        Wait(ROTATE90);
                        Release(sem);
                        break;

                        //Gira a sinistra
                case LEFT:
                        Acquire(sem);
                        Off(OUT_C);
                        OnRev(OUT_B, MAXPOWER);
                        Wait(ROTATE90);
                        Release(sem);
                        break;
                }
        }
        //Ultrasuoni frontale o colore nero
        else if(fus || black){

                switch(j){

                        //Gira a destra
                case RIGHT:
                        Acquire(sem);
                        Off(OUT_B);
                        OnRev(OUT_C, MAXPOWER);
                        Wait(ROTATE180);
                        Release(sem);
                        break;

                        //Gira a sinistra
                case LEFT:
                        Acquire(sem);
                        Off(OUT_C);
```

```
                OnRev(OUT_B, MAXPOWER);
                Wait(ROTATE180);
                Release(sem);
                break;

        }
}
//Pattern base
else{
        if(i==0 || i==31 || i==62 || i==83){

                Acquire(sem);
                Off(OUT_B);
                OnFwd(OUT_C, MAXPOWER);
                Wait(ROTATE90);
                Release(sem);
                i++;
        }
        else if(0<i && i<31){

                Acquire(sem);
                //Off(OUT_BC);
                OnFwd(OUT_BC, MAXPOWER);
                Wait(MTIME);
                Release(sem);
                i++;
        }
        else if(31<i && i<62){

                Acquire(sem);
                //Off(OUT_BC);
                OnFwd(OUT_BC, MAXPOWER);
                Wait(MTIME);
                Release(sem);
                i++;
        }
        else if(62<i && i<83){

                Acquire(sem);
                //Off(OUT_BC);
                OnFwd(OUT_BC, MAXPOWER);
                Wait(MTIME);
                Release(sem);
                i++;
        }
        else if(83<i && i<103){

                Acquire(sem);
```

```
                        //Off(OUT_BC);
                        OnFwd(OUT_BC, MAXPOWER);
                        Wait(MTIME);
                        Release(sem);
                        i++;
                }
                else if(i==103){
                        i=0;
                        Acquire(sem);
                        //Off(OUT_BC);
                        OnFwd(OUT_BC, MAXPOWER);
                        Wait(MTIME);
                        Release(sem);
                }


        }
}//Paper

//Comportamento del robot in modalità preda-forbici
sub Scissors(int &i, int j, bool black, bool fus, bool rus){
        //Ultrasuoni sul fronte e sul retro o colore nero e ultrasuoni sul retro
        if((fus && rus) || ((black) && rus)){

                switch(j){

                        //Gira a destra
                case RIGHT:
                        Acquire(sem);
                        Off(OUT_B);
                        OnRev(OUT_C, MAXPOWER);
                        Wait(ROTATE90);
                        Release(sem);
                        break;

                        //Gira a sinistra
                case LEFT:
                        Acquire(sem);
                        Off(OUT_C);
                        OnRev(OUT_B, MAXPOWER);
                        Wait(ROTATE90);
                        Release(sem);
                        break;
                }
        }
        //Ultrasuoni frontale o colore nero
        else if(fus || black){
```

```
		switch(j){

			//Gira a destra
		case RIGHT:
			Acquire(sem);
			Off(OUT_BC);
			OnRev(OUT_C, MAXPOWER);
			Wait(ROTATE135);
			Release(sem);
			break;

			//Gira a sinistra
		case LEFT:
			Acquire(sem);
			Off(OUT_BC);
			OnRev(OUT_B, MAXPOWER);
			Wait(ROTATE135);
			Release(sem);
			break;
		}

}
//Ultrasuoni sul retro
else if(rus){

		switch(j){

			//Gira a destra
		case RIGHT:
			Acquire(sem);
			Off(OUT_B);
			OnRev(OUT_C, MAXPOWER);
			Wait(ROTATE45);
			Release(sem);
			break;

			//Gira a sinistra
		case LEFT:
			Acquire(sem);
			Off(OUT_C);
			OnRev(OUT_B, MAXPOWER);
			Wait(ROTATE45);
			Release(sem);
			break;
		}
}
//Pattern base
else{
```

```
                //Avanti
                Acquire(sem);
                OnFwd(OUT_BC, MAXPOWER);
                Release(sem);

        }
}//Scissors

//Comportamento del robo in modalià preda-Lizard
sub Lizard(int &i, int j, bool black, bool fus, bool rus){
        //Ultrasuoni sul fronte o ultrasuoni sul retro o colore nero
        if(fus || rus || black){

                switch(j){

                        //Gira a destra
                case RIGHT:
                        Acquire(sem);
                        Off(OUT_B);
                        OnRev(OUT_C, MAXPOWER);
                        Wait(ROTATE90);
                        Release(sem);
                        break;

                        //Gira a sinistra
                case LEFT:
                        Acquire(sem);
                        Off(OUT_C);
                        OnRev(OUT_B, MAXPOWER);
                        Wait(ROTATE90);
                        Release(sem);
                        break;
                }


        }
        //Pattern base
        else{

                if(i==0 || i==31){
                        Acquire(sem);
                        Off(OUT_B);
                        OnFwd(OUT_C, MAXPOWER);
                        Wait(ROTATE90);
                        Release(sem);
                        i++;
                }
                else if(i==62 || i==93){
```

```
                Acquire(sem);
                Off(OUT_C);
                OnFwd(OUT_B, MAXPOWER);
                Wait(ROTATE90);
                Release(sem);
                i++;
            }
            else if(0<i && i<31 || 31<i && i<62 || 62<i && i<93 || 93<i && i<122){
                Acquire(sem);
                //Off(OUT_BC);
                OnFwd(OUT_BC, MAXPOWER);
                Wait(MTIME);
                Release(sem);
                i++;
            }
            else if(i==122){
                Acquire(sem);
                //Off(OUT_BC);
                OnFwd(OUT_BC, MAXPOWER);
                Wait(MTIME);
                Release(sem);
                i=0;
            }

        }
}//Lizard

//Comportamento del robot in modalità preda-Spock
sub Spock(int s, int &i, int j, bool black, bool fus, bool rus){

        switch(s){

        case ROCK:
            Rock(i, j, black, fus, rus);
            break;

        case PAPER:
            Paper(i, j, black, fus, rus);
            break;

        case SCISSORS:
            Scissors(i, j, black, fus, rus);
            break;

        case LIZARD:
            Lizard(i, j, black, fus, rus);
            break;
```

```
	}
}//Spock

//Seleziona il comportamento del robot autonomo in modalità predatore
//in base alla scelta della fase "Morra".
sub Prey(int robotchoice, int s, int &i, int j, bool black, bool fus, bool rus){

	ClearScreen();

	switch(robotchoice){

	case ROCK:
		TextOut(0, 0, "Rock", DRAW_OPT_NORMAL);
		Rock(i, j, black, fus, rus);
		break;

	case PAPER:
		TextOut(0, 0, "Paper", DRAW_OPT_NORMAL);
		Paper(i, j, black, fus, rus);
		break;

	case SCISSORS:
		TextOut(0, 0, "Scissors", DRAW_OPT_NORMAL);
		Scissors(i, j, black, fus, rus);
		break;

	case LIZARD:
		TextOut(0, 0, "Lizard", DRAW_OPT_NORMAL);
		Lizard(i, j, black, fus, rus);
		break;

	case SPOCK:
		TextOut(0, 0, "Spock", DRAW_OPT_NORMAL);
		Spock(s, i , j , black, fus, rus);
		break;

	}

}//Prey

//Chiede la conferma dell'avvenuto riposizionamento dei robot.
//
//Attende che venga premuto il bottone centrale del robot.
bool Reposition(){
	ClearScreen();
	TextOut(0,20,"Riposizionare i", DRAW_OPT_NORMAL);
	TextOut(0,10,"robot e premere", DRAW_OPT_NORMAL);
	TextOut(0, 0,"il tasto centrale", DRAW_OPT_NORMAL);
```

```
        until(ButtonPressed(BTNCENTER, true)){}
        return true;
}//Reposition
```