Politecnico di Milano

Facoltà di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Dipartimento di Elettronica, Informazione e Bioingegneria

FCbot
Guida Avanzata

Prof.ssa: Anna Maria ANTOLA

Prof. Tutor: Andrea BONARINI

Progetto di Ingegneria Informatica di:

Alberto Andrea TIRIBELLI 757054

Anno Accademico 2012/2013

Sommario

Risorse

Il progetto FCbot è stato elaborato dallo studente Alberto Andrea Tiribelli ed ha richiesto circa 4 mesi di tempo e fonda le sue basi sul progetto RSPLSbot.

Essendo la prima volta che il suddetto studente si confrontava 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.

Lo sviluppo del gioco vero e proprio è stato realizzato in seguito, attraverso le seguenti fasi:

- riprogettazione dei Robot per l'aggiunta di nuove funzionalità;
- navigazione e pattugliamento del robot autonomo;
- ridefinizione dei movimenti del robot controllato dal giocatore;
- gestione delle fasi di gioco;
- integrazione tra Joystick, robot del giocatore e robot autonomo via Bluetooth.

Stato dell'arte

Il progetto trae ispirazione dal gioco “Quattro Castelli”, variante del celebre “Quattro Cantoni”.

“Quattro Castelli” è un gioco adatto ad essere praticato su un ampio prato o eventualmente su una delle due metà di un campo da calcio. Agli angoli del campo sono delimitate quattro basi.
Le due squadre che si sfidano si posizionano sul campo di gioco, una all'interno di una delle quattro basi e l'altra sparsa in mezzo alle basi.
Come nel baseball, le due squadre copro il ruolo di difesa e attacco, la squadra in attacco deve correre attraverso delle basi senza farsi colpire dalla squadra in difesa.

Il progetto rappresenta principalmente una estensione del gioco RSLSPbot sviluppato in parallelo e in collaborazione con il relativo team.

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, nel progetto in esame, non è 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.
- Un oggetto (nel nostro caso un peso) da poter fare afferrare al robot controllato.

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) 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 "Maverick"
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;
- Una chela fissa e una mobile per poter afferrare e trattenere l'oggetto.

"Illustrazione - Maverick"

Robot autonomo "IceMan"
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.

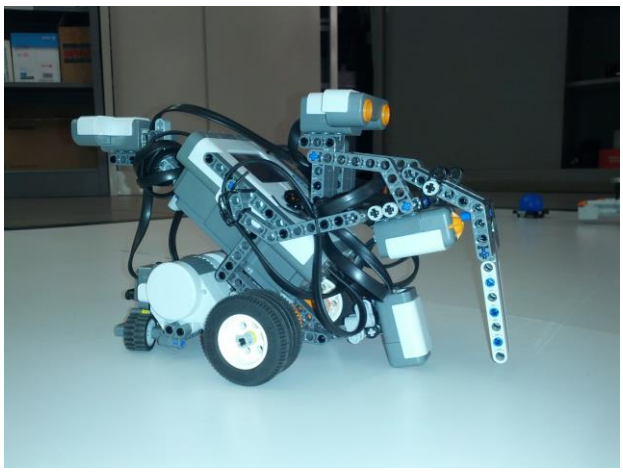
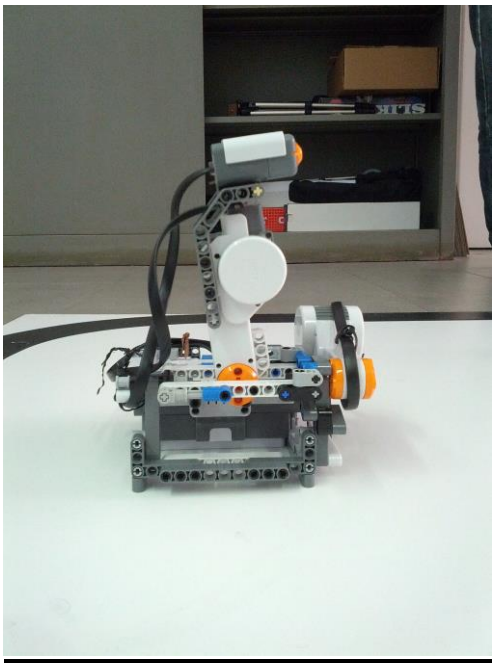


Illustrazione - IceMan

Joystick "Goose"
Il joystick è composto da:

- due motori assiali NXT;
- sensore di tocco montato a grilletto per poter utilizzare le chele.



"Illustrazione - Goose"

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.
Agli angoli sono posizionate 4 basi quadrate da 5cm dei seguenti colori: Verde, Blu, Giallo, Rosso.
(il tabellone esterno misura 1,80m X 2,00m)

“Illustrazione - Campo di gioco”

Scopo del gioco è di catturare l'oggetto protetto dal robot autonomo e di portarlo per le quattro basi del campo entro il tempo limite.

Le basi a cui portare l'oggetto vengono selezionate casualmente dal robot controlloato e comunicate al giocatore tramite lo schermo LCD del joystick.

Il robot autonomo inizialmente pattuglia la zona antistante la base gialla finché non rileva il robot del giocatore.
A questo punto comincia a pattugliare il campo alla ricerca del robot del giocatore.

Se il robot autonomo colpisce il robot del giocatore possono verificarsi 3 casi distinti:
- Il giocatore ha l'oggetto
  Il giocatore perde una vita, in aggiunta la chela del robot del giocatore rilascia l'oggetto e non può essere più essere usata fino a quando il robot non ripassa sopra alla sua base (rossa) riguadagnando la vita.

- Il giocatore possiede ancora una vita ma non ha l'oggetto
  Il giocatore perde una vita e la chela viene disabilitata.

- Il giocatore non possiede più vite e non ha l'oggetto
  Il giocatore ha perso.

Pattern di movimento ("robot autonomo" difensore)

Il robot autonomo inizialmente pattuglia la propria base a difesa dell'oggetto eseguendo delle rotazioni di 90° che gli permettono di rilevare velocemente qualsiasi intruso.
A seguito di una rilevazione si comporta da inseguitore.

- Pattern iniziale: Pattugliamento
- In seguito ad individuazione intruso: Forward Research

"Pattern 1- Pattugliamento"

"Pattern 2 - 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

Comandi del robot del giocatore

Tramite il joystick 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.
- Premere il grilletto: la chela si apre e si richiude dopo un secondo.

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 joystick che comunica con gli altri NXT attraverso una connessione bluetooth.
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 il compito di ricevere i comandi sotto forma di numeri interi e tradurli in movimenti, di elaborare la sequenza di basi da attraversare e di comunicare al joystick sia la prossima base da “catturare” sia il passaggio sulla casa (base rossa) nel caso in cui il giocatore abbia perso una vita.

Possibili sviluppi

- Implementare una modalità Survival in cui ogni volta che il giocatore vince viene aumentato il numero di basi da visitare;

Easter Eggs

- I nomi dei robot si ispirano al celebre film, della seconda metà degli anni '80, “Top Gun” ispirato agli avvenimenti della guerra fredda.
  Il nome di ogni robot si riferisce ai soprannomi dei personaggi del film, così il Joystick è “Goose” (Nick Bradshaw – Anthony Edwards), compagno di squadra del protagonista “Maverick” (Pete Mitchell – Tom Cruise) che da il suo nome al robot del giocatore; il robot autonomo si riferisce a “IceMan” (Tom Kazinsky – Val Kilmer), pilota della scuola Top Gun come Maverick, ma suo antagonista nel film.

Appendice A: codice joystick "Goose"

```
//Versione: Goose 1.0.0 22/07/2013
//
//Autore:
//Alberto Andrea Tiribelli
//
//Codice del joystick "Goose"

//Libreria personalizzata utilizzata nel progetto 4 Castles
#include "robogame.h"
#define BT_CONN_PLAYER 1
#define BT_CONN_ROBOT 2

//variabili di sincronizzazione globale
bool sync = false;

//variabili bluetooth
bool hit = false;
bool openpinsir = true;
mutex semBT;
int next = 0;
int life = 1;

//sensore Touch
//stabilisce le azioni corrispondenti alla pressione del touch
//quando premuto il sensore touch (che funziona da tasto del joystick)
//invia un segnale al robot controllato per richidere l'apertura delle chele,
//mediante la connessione bluetooth, sulla MAILBOX7
task Button(){
	while(!sync){
	}
	while(life>-1){
		if (Sensor(IN_1)){
			//l'invio del messaggio è possibile solo se si ha la vita, in quanto se si ha perso una vita
			//il controllo della chela è disabilitato
			if (life == 1){
				SendRemoteBool(BT_CONN_PLAYER,MAILBOX7,true);
				Wait(100);
			}
		}
	}
}//sensore Touch



//controllo touch robot
//imposta hit in base al messaggio ricevuto dal robot
//Il task rimane in attesa del segnale in ingresso, sulla MAILBOX2, ricevuto dal robot autonomo che
```

```
//indica se quest'ultimo ha colpito il giocatore
task Hit() {
	while(!sync){
	}
	while(life>-1){
		Acquire(semBT);
		ReceiveRemoteBool(MAILBOX2,true,hit);
		Wait(100);
		//alla ricezione del messaggio dal robot autonomo, diminuisce la vita
		if (hit){
			life--;
		}
		Release(semBT);
	}
}//controllo touch robot

//controllo prossima base
//riceve sulla MAILBOX1 un numero che indica quale sarà la prossima base da attraversare e
//stampa a schermo, la corrispondente indicazione
task Base(){
	while(!sync){
	}
	ClearScreen();
	TextOut(1,20,"Per cominciare",DRAW_OPT_NORMAL);
	TextOut(1,10,"attraversa la",DRAW_OPT_NORMAL);
	TextOut(1,1,"base Gialla",DRAW_OPT_NORMAL);
	while (life>-1){
		Acquire(semBT);
		ReceiveRemoteNumber(MAILBOX1,true,next);
		Wait(100);
		Release(semBT);
		switch(next){
		case RED:
			//nel caso in cui arrivi il segnale della base rossa, se life è uguale a zero, la incrementa
			if (life==0){
				life++;
				openpinsir = true;
			}
			else{
				ClearScreen();
				TextOut(1,10,"Cattura la base",DRAW_OPT_NORMAL);
				TextOut(1,1,"      Rossa",DRAW_OPT_NORMAL);
			}
			break;
		case BLUE:
			ClearScreen();
			TextOut(1,10,"Cattura la base",DRAW_OPT_NORMAL);
			TextOut(1,1,"      Blu",DRAW_OPT_NORMAL);
```

```
                    break;
                case GREEN:
                    ClearScreen();
                    TextOut(1,10,"Cattura la base",DRAW_OPT_NORMAL);
                    TextOut(1,1,"      Verde",DRAW_OPT_NORMAL);
                    break;
                case YELLOW:
                    ClearScreen();
                    TextOut(1,10,"Cattura la base",DRAW_OPT_NORMAL);
                    TextOut(1,1,"      Gialla",DRAW_OPT_NORMAL);
                    break;
                case -1:
                    //il valore "-1" indica la prima base da attraversare per cominciare la partita
                    ClearScreen();
                    TextOut(1,20,"Per cominciare",DRAW_OPT_NORMAL);
                    TextOut(1,10,"attraversa la",DRAW_OPT_NORMAL);
                    TextOut(1,1,"base Gialla",DRAW_OPT_NORMAL);
                    break;
                case 6:
                    //il valore "6"  indica la fine del gioco, con vittoria del giocatore
                    ClearScreen();
                    TextOut(1,1,"COMUNISTA!!!",DRAW_OPT_NORMAL);
                    Wait(500);
                    life=-1; //ferma tutto il programma
                    break;
                }
        }
}

//controllo movimenti
//in base all'inclinazione dell'asta invia i corrispondenti comandi al robot
//del giocatore, mediante il Bluetooth, sulla MAILBOX6
task Command(){
        while(true){
                if(sync){
                    if(life == 0 && openpinsir){
                            SendRemoteNumber(BT_CONN_PLAYER,MAILBOX6,OPENPINSIR);
                            TextOut(1,1,"Stringa inviata", DRAW_OPT_NORMAL);
                            Wait(1000);
                            openpinsir = false;
                    }
                    //Avanti
                    else if(MotorRotationCount(OUT_A)>-MAXAXIS && MotorRotationCount(OUT_A)<-MINAXIS){
                            SendRemoteNumber(BT_CONN_PLAYER,MAILBOX6,FORWARD);
                            //Wait(100);
                    }
                    //Indietro
                    else if(MotorRotationCount(OUT_A)>MINAXIS && MotorRotationCount(OUT_A)<MAXAXIS){
```

```
                                SendRemoteNumber(BT_CONN_PLAYER,MAILBOX6,BACKWARD);
                                //Wait(100);
                        }
                        //Destra
                        else if(MotorRotationCount(OUT_B)>MINAXIS && MotorRotationCount(OUT_B)<MAXAXIS){
                                SendRemoteNumber(BT_CONN_PLAYER,MAILBOX6,TURNRIGHT);
                                //Wait(100);
                        }
                        //Sinistra
                        else if(MotorRotationCount(OUT_B)>-MAXAXIS && MotorRotationCount(OUT_B)<-MINAXIS){
                                SendRemoteNumber(BT_CONN_PLAYER,MAILBOX6,TURNLEFT);
                                //Wait(100);
                        }
                        //Fermo
                        else{
                                SendRemoteNumber(BT_CONN_PLAYER,MAILBOX6,BREAKALL);
                                //Wait(100);
                        }
                }
        }
}//controllo movimenti

//invia i messaggi ai robot di fine partita
//Fine Gioco
task EndGame(){
        SendRemoteBool(BT_CONN_ROBOT,MAILBOX10,true);
        Wait(100);
        ClearScreen();
        TextOut(1,1,"STOP!",DRAW_OPT_NORMAL);
        Wait(1000);
}

//Controlla il funzionamento del programma facendo partire il task EndGame, nel caso in cui la variabile life
//arrivi a "-1"
//dopo aver premuto il tasto centrale, invia al robot autonomo il messaggio di inizio della partita
//e setta la variabile sync a true, sbloccando tutti i task che comandano il robot del giocatore
//Controllo Gioco
task Game(){
        ClearScreen();
        TextOut(1,20,"Premi il tasto",DRAW_OPT_NORMAL);
        TextOut(1,10,"centrale per",DRAW_OPT_NORMAL);
        TextOut(1,1,"iniziare",DRAW_OPT_NORMAL);
        until(ButtonPressed(BTNCENTER, true)){}
        ClearScreen();
        //invia messaggio start al robot
        SendRemoteBool(BT_CONN_ROBOT,MAILBOX1,true);
        Wait(100);
        sync = true;
```

```
		Wait(1000);
		while(life>-1){
			if(life==0){
				ClearScreen();
				TextOut(1,20,"Attraversa la",DRAW_OPT_NORMAL);
				TextOut(1,10,"base Rossa per",DRAW_OPT_NORMAL);
				TextOut(1,1,"ricaricarti",DRAW_OPT_NORMAL);
			}
		}
		sync = false;
		Wait(1000);
		Precedes(EndGame);
}//controllo Gioco

task main(){
		SetSensorTouch(IN_1);
		Precedes(Game, Command, Button, Hit, Base);
}
```

Appendice B: codice robot del giocatore "Maverick"

```
//Versione: Maverick 1.0.0 22/07/2013
//
//Autore:
//Alberto Andrea Tiribelli
//
//Codice del robot del giocatore "Maverick"

//Libreria personalizzata utilizzata nel progetto 4 Castles
#include "robogame.h"
//Variabili booleane dei sensori
bool touch = false;
bool black = false;
bool white = false;
bool red = false;
bool green = false;
bool yellow = false;
bool blue = false;
bool claw;
//Variabili Bluetooth
int movimento;
//semafori
mutex semMovement;
mutex semC;
mutex semNB;

//variabili usate dalla funzione NextBase
int sequence = 4;
int nextcolor = -1;
int oldcolor;
bool previouscolor[4]={false,false,false,false};

//variabili timeout
bool startTimeout = false;
bool timeout = false;

//funziona che fa spegnere il robot
sub Exit(){
    Wait(3000);
    PowerDown();
}

//Controlla se il sensore posizionato in mezzo alle chele viene premuto e imposta a true la relativa variabile
//Sensore touch
task Touch(){
        while(true){
                //se rileva pressione imposta la variabile a true
```

```
                if (Sensor(IN_1)){
                        touch=true;
                        }
                //altrimenti a false
                else touch=false;
                }
        }//sensore touch


//genera la prossima base da attraversare e comunica il passsaggio nella base rossa, come si è conclusa la partita, se per vittoria del giocatore
//o per timeout, attraverso la connessione Bluetooth, sulla MAILBOX1 al Joystick
//prossima base
void NextBase(){
//se il giocatore è passato in tutte le basi, imposta nextcolor al valore di vittoria
if (sequence == 0 && touch){
nextcolor = 6;
oldcolor = nextcolor;
}
//se il timeout è scaduto imposta nextcolor al valore di sconfitta
else if(timeout){
nextcolor = 7;
oldcolor = nextcolor;
}
//se NextBase() è stata richiamata senza avere il touch premuto, significa che il giocatore è passato dalla propria base
//quindi si salva nella variabile oldcolor il vecchio valore della base da attraversare
else if(!touch){
        oldcolor=nextcolor;
        nextcolor=RED;
        }
else{
//altrimenti genera un nuovo intero che corrisponde alla base da attraversare, diverso da tutti i precedenti

    startTimeout = true;

    do{
      nextcolor = Random(4)+2; //il primo colore è il 2 e l'ultimo il 5
      }while(previouscolor[nextcolor-2]);

    sequence--;

    //previouscolor contiene 4 variabili booleane che corrispondono alle 4 basi e sono settate a true, se la base corrispondente è stata già attraversata
    previouscolor[nextcolor-2]=true;

    //nel caso in cui la sequenza preveda l'attraversamento di 5 basi o più, cancella il registro delle basi attraversate dopo il quarto attraversamento
    if(previouscolor[0]==true && previouscolor[1]==true && previouscolor[2]==true && previouscolor[3]==true){ previouscolor[0]=false;
```

```
                                                    previouscolor[1]=false;
                                                    previouscolor[2]=false;
                                                    previouscolor[3]=false;
                                                    }
    oldcolor=nextcolor;
    }
//invia il prossimo colore al joystick
PlayTone(500,500);

//se in oldcolor è contenuto un diverso valore da nextcolor, invia prima nextcolor che è impostato a Rosso e poi il vecchio che indica la
prossima base
//da attraversare
if (nextcolor!=oldcolor){
SendResponseNumber(MAILBOX1,nextcolor);
Wait(500);
nextcolor=oldcolor;
SendResponseNumber(MAILBOX1,nextcolor);
Wait(100);
}
else{
Acquire(semNB);
SendResponseNumber(MAILBOX1,nextcolor);
Wait(500);
Release(semNB);
}

//se in nextcolor è salvato un valore di vittoria o sconfitta, chiama la funzione per spegnere il programma
if(nextcolor == 6 || nextcolor == 7){
        Acquire(semNB);
        Wait(500);
        Exit();
        Release(semNB);
}

          }//prossima base

//Setta la variabile booleana timeout a true quando passano 60 secondi.
task Timeout(){

  until(startTimeout);

  //Attende il tempo del timeout meno 10000ms
        Wait(180000 -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
		NextBase();
}//Timeout

//Sensore di colore
task Color(){
		while(true){
			//in base al colore rilevato imposta il valore delle relative variabili
			//chiama la funzione NextBase se la base attraversata è la prossima in sequenza
			//(NextBase può essere richiamata solo con il touch attivato, cioè solo quando si è in possesso della pallina)
			switch(Sensor(IN_2)){
				case BLACK:
					black = true;
					white = false;
					red = false;
					green = false;
					yellow = false;
					blue = false;
					break;
				case WHITE:
					black = false;
					white = true;
					red = false;
					green = false;
					yellow = false;
					blue = false;
					break;
				case RED:
					black = false;
					white = false;
					red = true;
					green = false;
					yellow = false;
					blue = false;
					//richiamala anche quando il touch non è attivato per ricaricare la vita
					if(nextcolor==RED || !touch) NextBase();
					break;
				case GREEN:
					black = false;
					white = false;
					red = false;
					green = true;
					yellow = false;
					blue = false;
					if(nextcolor==GREEN && touch) NextBase();
```

```
                                        break;
                                case YELLOW:
                                        black = false;
                                        white = false;
                                        red = false;
                                        green = false;
                                        yellow = true;
                                        blue = false;
                                        //richiama NextBase anche con nextcolor a "-1" che rappresenta il valore iniziale
                                        if((nextcolor==YELLOW || nextcolor==-1) && touch) NextBase();
                                        break;
                                case BLUE:
                                        black = false;
                                        white = false;
                                        red = false;
                                        green = false;
                                        yellow = false;
                                        blue = true;
                                        if(nextcolor==BLUE && touch) NextBase();
                                        break;
                                }
                        }
                }//sensore di colore

//controllo chele
task Grab(){
        while(true){
                ReceiveRemoteBool(MAILBOX7,true,claw);
                Wait(100);
                if (claw){
                        Acquire(semC);
                        RotateMotor(OUT_A, 60, 90);
                        Wait(1000);
                        RotateMotor(OUT_A, -90, 90);
                        Release(semC);
                        }
                }
        }

//movimenti del robot
task Movement(){
    movimento = 0;
    int state = 0;

    while(true){
            //Riceve un comando dal joystick
            ReceiveRemoteNumber(MAILBOX6, true, movimento);
            Wait(100);
```

```
//Controlla se il robot sta cercando di sorpassare il bordo del campo di gioco
if(!black){
	switch(movimento){
		case FORWARD:
			Acquire(semMovement);
			Off(OUT_BC);
			OnFwd(OUT_BC, MPOWER);
			state = FORWARD;
			Release(semMovement);
			break;

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

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

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

		case BREAKALL:
			Acquire(semMovement);
			Off(OUT_BC);
			state = BREAKALL;
			Release(semMovement);
			break;

		case OPENPINSIR:
			PlayTone(500,500);
			Acquire(semMovement);
			Acquire(semC);
			RotateMotor(OUT_A, 90, 90);
			Off(OUT_BC);
```

```
                OnRev(OUT_BC, MPOWER);
                Wait(1000);
                Off(OUT_BC);
                RotateMotor(OUT_A, -90, 90);
                Release(semC);
                Release(semMovement);
                break;
        }
}else{
        switch(state){

                case FORWARD:
                    Acquire(semMovement);
                    Off(OUT_BC);
                    OnRev(OUT_BC, MPOWER);
                    Wait(500);
                    Off(OUT_BC);
                    Release(semMovement);
                    break;

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

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

                case TURNLEFT:
                    Acquire(semMovement);
                    Off(OUT_BC);
                    OnFwd(OUT_C, MPOWER);
                    Wait(500);
                    Off(OUT_BC);
                    Release(semMovement);
                    break;
    }
}
```

```
                movimento = 0;
    }
}//Movimenti Robot

task main(){
//configurazione sensori
SetSensorTouch(IN_1);
SetSensorColorFull(IN_2);
Precedes(Touch, Color, Movement, Grab, Timeout);
}
```

Appendice C: codice robot del giocatore "IceMan"

```
//Versione: IceMan 1.0.0 22/07/2013
//
//Autore:
//Alberto Andrea Tiribelli
//
//Codice del robot autonomo "IceMan"

//Libreria personalizzata utilizzata nel progetto 4 Castles
#include "robogame.h"

//Variabili booleane usate dai sensori
bool touch = false;
bool fus = false;
bool rus = false;
bool black = false;
bool white = false;

//Variabili booleane usate nella comunicazione Bluetooth
bool run = false;
bool end = false;

//Variabili usate nel task Predator
int i;
int j;
int k;
bool hunt;

//Semafori
mutex sem;

//Sensore Luce
task Light(){
	while(true){
		//in base ai valori rilevati dal sensore imposta le variabili di colore black e white
		if(Sensor(IN_2)<=35){
			black = true;
			white = false;
		}else{
			black = false;
			white=true;
		}
	}
}//Sensore Luce


//Sensore Ultrasonico Anteriore
```

```
task FrontUS(){
	while(true){
		if(SensorUS(IN_3)<RPREDATOR_THRESHOLD) fus = true;
		else fus = false;
	}
}//Sensore Ultrasonico Anteriore

//Sensore Ultrasonico Posteriore
task RearUS(){
	while(true){
		if(SensorUS(IN_4)<RPREDATOR_THRESHOLD) rus = true;
		else rus = false;
	}
}//Sensore Ultrasonico Posteriore

//Questo task comunica con il joystick, sulla MAILBOX2, inviando un messaggio nel caso in cui il robot colpisca il giocatore
//Sensore touch
task Touch(){
	while(true){
		//se rileva pressione imposta la variabile a true
		if (Sensor(IN_1)){
			touch = true;
			//manda il segnale hit al joystick
			SendResponseBool(MAILBOX2,true);
			Wait(1000); //500
			ClearScreen();
			TextOut(0, 0, "Touch me!", DRAW_OPT_NORMAL);
			PlayTone(1000,500);
		}
		//altrimenti a false
		else{
			touch = false;
		}
	}
}//sensore touch

//questo task gestisce i movimenti del robot, in un primo momento esso pattuglia la zona davanti a sé
//con continue rotazioni di 90° gradi in modo da coprire i due archi di circonferenza frontali
//in seguito, all'attivazione di un sensore, mediante pattern predefiniti insegue il giocatore
//movimenti robot
sub Predator(){
	if(fus||rus||black||hunt){
		//se si attiva uno dei sensori attiva l'inseguimento dei giocatori

		//la variabile hunt impedisce al robot di tornare a pattugliare sul posto, se i sensori non rilevano nulla
		hunt = true;

		//quando il robot oltrepassa il bordo, lo riporta in campo facendolo girare in una direzione casuale
```

```
//Colore Nero
if(black){
	switch(j){
		ClearScreen();
		TextOut(0, 0, "Black", DRAW_OPT_NORMAL);

		//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){
	ClearScreen();
	TextOut(0, 0, "Fus", DRAW_OPT_NORMAL);
	//insegue il nemico di fronte a sé
	Acquire(sem);
	OnFwd(OUT_BC, PPOWER);
	Wait(MTIME);
	Release(sem);
}

//Ultrasuoni sul retro
else if(rus) {
	//se rileva il nemico dietro di sé ruota di 180° in una direzione casuale
	ClearScreen();
	TextOut(0, 0, "Rus", DRAW_OPT_NORMAL);
```

```
		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;
		}
	}
	else if(touch){
		//se tocca il robot, torna indietro e aspetta tre secondi
		//introdotta per facilitare il gioco, questa parte di codice permette al giocatore di scappare
		//senza essere inseguito, dopo essere stato toccato
		Acquire(sem);
		Off(OUT_BC);
		OnRev(OUT_BC, PPOWER);
		Wait(500);
		Off(OUT_BC);
		Wait(3000);
		Release(sem);
	}
	//Pattern base
	else{
		ClearScreen();
		TextOut(0, 0, "Predator", DRAW_OPT_NORMAL);
		//Avanza
		if(0<=i && i<30){ //20
			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){ //20-50-60
```

```
                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;
            }

        }
    }
    else{
        //ClearScreen();
        //TextOut(0, 0, "Pattuglia", DRAW_OPT_NORMAL);
        //Wait(500);
        //Arco di 90° a destra
        if(0<=k && k<10){
            Acquire(sem);
            Off(OUT_BC);
            OnFwd(OUT_B, MAXPOWER);
            Wait(MTIME);
            Release(sem);
            k++;
        }
        //Arco di 90° a destra
        else if(10<=k && k<20){
            Acquire(sem);
            Off(OUT_BC);
            OnRev(OUT_B, MAXPOWER);
            Wait(MTIME);
            Release(sem);
            k++;
        }
        //Arco di 90° a sinistra
        else if(20<=k && k<30){
            Acquire(sem);
            Off(OUT_BC);
            OnFwd(OUT_C, MAXPOWER);
            Wait(MTIME);
            Release(sem);
            k++;
```

```
                }
                //Arco di 90° a sinistra
                else if(30<=k && k<40){
                        Acquire(sem);
                        Off(OUT_BC);
                        OnRev(OUT_C, MAXPOWER);
                        Wait(MTIME);
                        Release(sem);
                        k++;
                }
                else{
                        Acquire(sem);
                        Off(OUT_BC);
                        OnFwd(OUT_C, MAXPOWER);
                        Wait(MTIME);
                        Release(sem);
                        k=0;
                }
        }
}//movimenti robot


//il task Menu controlla il funzionamento del robot autonomo.
//all'inizio attende in ingresso il messaggio da parte del joystick che indica che la partita può cominciare
//in seguito continua a richiamare la funzione Predator per muoversi alla ricerca del giocatore
//Una volta che la variabile run viene impostata a false, dal task endGame, spegne i motori e chiude il programma
//La comunicazione Bluetooth con il joystick è effettuata sulla MAILBOX1
//controllo gioco
task Menu(){
        //attende il messaggio in ingresso sulla MAILBOX1
        while(!run){
                ClearScreen();
                TextOut(1,10, "Attendo il mex",DRAW_OPT_NORMAL);
                TextOut(1,1, "di Start",DRAW_OPT_NORMAL);
                ReceiveRemoteBool(MAILBOX1,true,run);
                Wait(100);
        }
        //imposta i valori iniziali delle variabili di Predator
        k=0;
        hunt=false;
        j = Random(2);
        //richiama la funzione Predator per inseguire il giocatore
        while(run){
                Predator();
        }
        //spegne i motori
        Acquire(sem);
        Off(OUT_BC);
```

```
		Release(sem);
}//controllo gioco

//Questo task attende il messaggio da parte del joystick di fine partita, inviato sulla MAILBOX10
//fine gioco
task EndGame(){
		//quando il messaggio che arriva imposta la variabile a "true", imposta a "false" run per interrompere il programma
		while(!end){
			ReceiveRemoteBool(MAILBOX10,true,end);
			Wait(100);
			if(end) run=false;
		}
}//fine gioco
task main(){
		SetSensorTouch(IN_1);
		SetSensorLight(IN_2);
		SetSensorUltrasonic(IN_3);
		SetSensorUltrasonic(IN_4);
		Precedes(Touch, FrontUS, RearUS, Light, Menu, EndGame);
}
```