Politecnico di Milano

Facoltà di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

RoboWii2.0.L: analisi e miglioramento della giocabilità

Professoressa: Franca GARZOTTO

Professore Tutor: Andrea BONARINI

Progetto di HCI di:

Mirko CURTOLO matr. 740228

Marco FRAGNOLI matr. 749985

Anno Accademico 2009/2010

Sommario

Introduzione

La robotica sta entrando sempre più nella vita di tutti i giorni, anche con prodotti a larga diffusione, come elettrodomestici robotizzati o giochi.
Il progetto attuale intende migliorare la giocabilità di RoboWii2.0.L, un gioco che prevede un'interazione uomo-macchina di ultima generazione. In questo gioco sono presenti due robot: un robot preda il cui scopo, come suggerito dal nome, è fuggire da un robot predatore. Il robot predatore è comandato dal giocatore attraverso comandi forniti tramite il riconoscimento di movimenti: le "gesture". Il robot preda si muove invece autonomamente ed è dotato di sensori di movimento che lo rendono in grado di interagire con l'ambiente e con il predatore stesso (e quindi con il giocatore).
Il gioco, precedentemente realizzato, consiste appunto nel guidare il predatore verso la preda e "colpirla" tramite l'uso di una delle parti mobili del robot predatore, un braccio posizionato sopra il robot che può essere rinominato "pungiglione" per la somiglianza con il pungiglione di uno scorpione.
Allo stato in cui il gioco si trovava nel momento in cui abbiamo iniziato questo progetto di miglioramento, il gioco presentava una difficoltà spiazzante ed era a tratti frustrante in quanto il robot preda seguiva dei movimenti del tutto casuali. L'utente riusciva difficilmente a colpire la preda attraverso il robot predatore.
L'obiettivo principale del nostro lavoro è stato, quindi, quello di riprogrammare l'intelligenza artificiale del robot preda migliorando l'interazione uomo-macchina, facendo sì che i livelli nel gioco avessero caratteristiche diverse tra loro e difficoltà graduale. Tutto questo per permettere all'utente un maggiore divertimento nella fase di gioco.

Risorse

Il progetto è stato svolto da due studenti: Mirko Curtolo e Marco Fragnoli. Il progetto ha richiesto alcune settimane (2-3) di setup in quanto è stato necessario riassemblare alcune delle componenti essenziali del gioco. Non avendo a disposizione tutti i pezzi del robot principale sono state necessarie diverse modifiche anche nel codice del robot predatore. L'ambiente di lavoro si è rivelato complesso, vista l'interazione di molti componenti (due robot lego NXT, le relative connessioni bluetooth col pc e il wiimote con le sue librerie). Una buona documentazione ci ha permesso, però, di ripristinare il gioco originale. Qualche altra settimana (2-3) è stata necessaria per la fase di analisi del gioco originanale, implementazione della nuova intelligenza artificiale e test di quest'ultima.
Una stima delle ore impiegate è 30 ore/persona.

Stato dell'arte

Il progetto parte da un gioco già esistente chiamato RoboWii 2.0.L.
RoboWii 2.0.L è un gioco robotico interattivo, il suo nome deriva da una serie di progetti realizzati dall'AIRLab, un laboratorio del dipartimento di elettronica e informatica del Politecnico di Milano.

Questi progetti, appartenenti all'ambito della robotica, hanno in comune l'utilizzo di un robot e del controller Wii Remote. L'utilizzo di quest'ultimo varia notevolmente da progetto a progetto, ma l'elemento di maggior risalto è la ricerca di una nuova interazione tra l'utente e i robot, approfondendo lo studio delle potenzialità di questo controller.

Nel nome del progetto è contenuta una “L”, che indica l’utilizzo di robot diversi dai precedenti progetti. I robot utilizzati sono i LEGO NXT MINDSTORM.
La scelta del sistema di controllo si basa sulla ricerca di una nuova modalità di interazione tra l’utente e il gioco stesso, seguendo la tendenza degli ultimi anni del mercato dei videogame di non basare più i prodotti sull’utilizzo di un semplice gamepad, ma di sfruttare movimenti, azioni e informazioni visive per controllare l’evoluzione del gioco stesso.

Il progetto realizzato si propone di analizzare e migliorare la giocabilità di RoboWii 2.0.L.

Requisiti

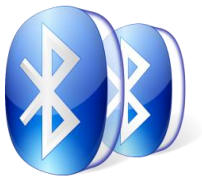
Hardware and Software

Il gioco richiede software e hardware molto specifici.
Per quanto riguarda l’hardware sono necessari i seguenti componenti:

- 2 kit di Lego Mindstorm NXT utilizzati per costruire i robot preda e predatore (http://mindstorms.lego.com/). Questo tipo di robot è dotato di un ampia scelta di sensori di vario genere e permette una notevole duttilità. È possibile plasmare varie forme di robot: umanoidi, veicolari, animali. Questa linea di prodotti permette all’utente di creare dei veri e propri sistemi automatici elettromeccanici capaci di compiere semplici azioni e reagire a stimoli esterni percepiti tramite i sensori. Poiché dotati di microprocessore e ram sono in grado di eseguire ed immagazzinare programmi.

- 1 Wii Remote: rivoluzionario controller per la console Nintendo Wii. Questo rivoluzionario sistema di controllo coinvolge maggiormente il giocatore aumentando la sensazione di trovarsi all’interno del gioco. Il Wii Remote, oltre a disporre di una pulsantiera standard, possiede e fa largo uso di accelerometri, speaker e telecamera infrarossi.

- Un pc dotato di dispositivo bluetooth, necessario per la comunicazione tra i due robot, il pc e il controller.




Il software base del gioco prevede 3 parti fondamentali.

- La prima delle tre è un programma scritto in C dotato di interfaccia grafica che permette all’utente di selezionare il livello di gioco e controllare che tutto funzioni correttamente (connessioni tra robot, wii remote e pc).

- La seconda e la terza parte sono dei software scritti in un linguaggio simile al C, chiamato BricXcc, che sono stati caricati rispettivamente sul robot preda e su quello predatore e vengono eseguiti dai processori contenuti nei robot stessi. Nel progetto questi due software sono l'elemento cruciale dato che sono stati quelli da noi modificati ed, in alcune parti, riscritti. In particolare, abbiamo riscritto quasi completamente il software eseguito sul robot preda. Per il robot predatore ci siamo limitati ad apportare solo piccole modifiche che

migliorassero la manovrabilità del robot predatore. In quest'ultimo caso si è trattato solo di leggere modifiche anche se non meno cruciali ed importanti ai fini del raggiungimento dello scopo prefissato

Obiettivi e panoramica generale

L'obiettivo principale è migliorare la giocabilità di RoboWii 2.0.L attraverso la modifica dell'intelligenza artificiale dei comportamenti dei robot coinvolti nel gioco. Per migliorare la giocabilità si intende fornire al giocatore una nuova e migliorata interazione con il robot comandato e con il robot avversario. Il tutto in modo da fornire una sfida graduale ed appassionante che parta dall'apprendimento dei movimenti base, fino al raggiungimento di tutti gli obbiettivi finali del gioco.
Migliorando la giocabilità, il giocatore dovrebbe essere portato a continuare il gioco anche dopo le prime partite di prova. Insomma, esaurito l'entusiasmo iniziale che un gioco di questo tipo provoca in giocatori di diverse fasce d'età, il gioco deve mantenere alto il livello di "engaging".

Target

Questo gioco si rivolge a utenti di diversi fasce d'età. Data la semplicità e l'intuitività dei comandi di interazione con i robot il gioco è adatto anche a bambini della fascia 4-9 anni. Tuttavia grazie alla peculiarità delle sue componenti, il gioco può risultare divertente anche per ragazzi più grandi o addirittura per adulti. Infatti tra le persone che hanno provato il gioco (sia nella sua versione originale che nella versione da noi modificata) si sono dimostrati entusiasti anche solo del vedere un robot rispondere ai loro comandi.

Game "concept"

Per permettere una migliore comprensione del lavoro svolto occorre prima descrivere in cosa consiste il gioco:
Come già accennato il gioco consiste, dal puto di vista del giocatore, nel comandare un robot predatore con l'obiettivo di raggiungere e colpire una preda che si muove nell'ambiente circostante.

Il predatore, in Figura 1, dispone di due ruote motrici e di un braccio meccanico montato sopra di esso che verrà chiamato "pungiglione". Questo pungiglione consiste, oltre che nel braccio meccanico, in un sensore di tocco indispensabile per rilevare quando il pungiglione colpisce un oggetto.

Figura 1: Robot predatore

La preda, in Figura 2, dispone anch'essa di due ruote motrici mentre, al contrario del predatore, monta due sensori di movimento: uno davanti ed uno dietro, in grado di riconoscere gli ostacoli rilevabili. Occorre specificare che per ostacolo rilevabile si intende un solido di una dimensione paragonabile a quella della preda, meglio se privo di fori.
Oltre a questi due sensori, la preda dispone di un terzo sensore (di tocco) montato sotto una piccola piattaforma. Questa piattaforma rappresenta la parte "vulnerabile" della preda. Se colpita essa si muove e preme sul sensore di tocco comunicando di fatto al gioco che la preda è stata colpita.

Figura 2: Robot Preda

Il giocatore comanda, tramite il controller WiiMote, i movimenti del predatore che possono essere:

- Avanti: Quando il predatore riceve il comando avanti, se è fermo, accelera fino a raggiungere la prima delle 3 velocità. Se è già in movimento ad una delle prime due velocità, raggiunge quella successiva. Altrimenti il comando è ignorato.

- Destra: Ruota di 45 gradi verso destra. Il movimento viene fatto in sincronia con qualsiasi altro movimento in esecuzione il momento.

- Sinistra: Ruota di 45 gradi verso sinistra. Il movimento viene fatto in sincronia con qualsiasi altro movimento in esecuzione in quel momento.

- Indietro: Quando il predatore riceve il comando indietro, se è fermo, si muove all'indietro raggiungendo l'unica velocità di retromarcia possibile. Se invece il robot si sta muovendo in avanti il comando ha l'effetto di ridurre la velocità di un livello, nel caso fermandosi. Se il robot si sta già muovendo in retromarcia il comando viene ignorato.

- Attacca: Quando il predatore riceve il comando attacca, indipendentemente dal movimento in esecuzione in quel momento, aziona il pungiglione che scatta in avanti.

- Break all: Quando il predatore riceve questo comando interrompe qualsiasi movimento in esecuzione in quel istante. Comando essenziale per arresti di "emergenza".

E' importante chiarire che i primi 4 comandi sono impartiti al predatore non per mezzo della pressione di pulsanti sul controller, ma dal campionamento dei movimenti del braccio del giocatore. Questo con lo scopo di permettere al giocatore di "immedesimarsi" nella caccia mimando con il braccio l'atto di muovere in avanti, indietro o girare il robot nella direzione desiderata.

Spiegate le caratteristiche delle parti in gioco, il resto segue uno schema piuttosto semplice. La preda sfrutta i sensori di movimento per muoversi sul campo da gioco (qualsiasi terreno piano e liscio) mentre il predatore, e quindi il giocatore che lo comanda, cerca di raggiungerla e di colpirla con l'arpione nel suo "punto debole" ossia la piattaforma montata su di essa.

Il gioco prevede 6 diversi livelli di difficoltà la cui descrizione è demandata alla parte in cui analizzeremo la giocabilità e proporremo soluzioni per migliorarla.

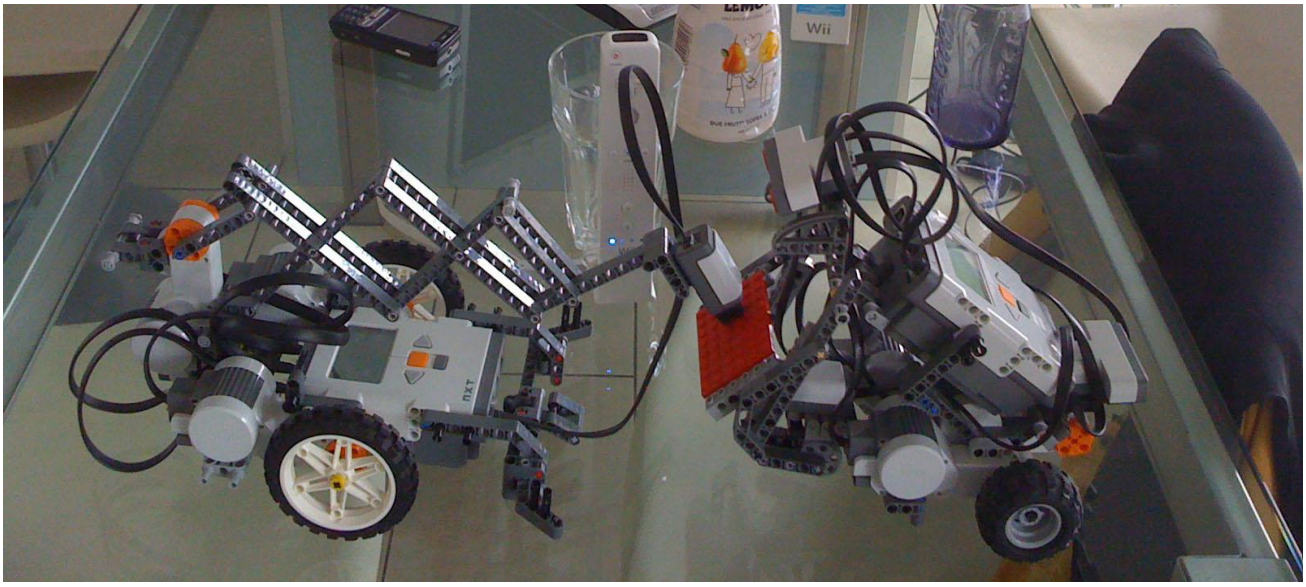


Figura 3: Robot preda colpito dal predatore

User Experience Design

Il primo problema riscontrato nel gioco era il movimento della preda che, in tutti i 6 livelli di difficoltà che il gioco offriva, si muoveva in modo completamente casuale nell'ambiente circostante.

Questo causava principalmente due problemi:

1. Già dai primi livelli il gioco risultava frustrante. Infatti l'utente, innanzitutto doveva confrontarsi con un tipo di comando (quello delle "gesture") il cui primo approccio risulta in ogni caso meno immediato rispetto a quello con altri sistemi di controllo. Doveva inoltre confrontarsi con un "avversario", il robot preda, che lo disorientava con improvvisi scatti e continui cambi di direzione. Infatti persino al primo livello non esisteva una vera e propria calibrazione della velocità e dell'"intelligenza" della preda. Muovendosi in modo casuale la preda solo raramente offriva al giocatore la possibilità di essere rincorsa e, anzi, rendeva praticamente impossibile l'impresa con cambi di direzione che avvenivano con cadenza regolare ogni 2 o 3 secondi. Per di più anche la velocità non era calibrata. Infatti la preda risultava molto più veloce del predatore dato che la velocità di base della preda era superiore alla "prima" velocità del predatore, ossia quella che raggiungeva da fermo dopo la prima accelerazione (la velocità del predatore poteva essere incrementata due volte prima di raggiungere quella massima). In questo modo il giocatore si trovava ad inseguire la preda solo per pochi istanti prima di vederla velocemente scattare in un'altra direzione. E tutto questo solo per il più facile dei 6 livelli!
   Inoltre questo comportamento "indomabile" della preda si risolveva spesso in uno scontro con un ostacolo o in certi casi con il predatore stesso. Questo obbligava il giocatore ad interrompere il gioco ristabilendo una situazione in cui i due robot fossero liberi di muoversi.

2. Il secondo problema consisteva nello scarso livello di coinvolgimento. Infatti, qualora un giocatore si fosse dimostrato così bravo da colpire la preda e risultare vincitore ad uno dei primi livelli, non trovava nei livelli seguenti alcun elemento di novità. Infatti nei livelli alti la preda continuava a muoversi in modo casuale, la differenza era che la sensibilità dei suoi sensori veniva notevolmente aumentata. La preda veniva, quindi, resa in grado di scorgere (tramite il sensore posto sul retro) il predatore in inseguimento. In sostanza, però, questa introduzione non aumentava per nulla il livello di "engaging". Infatti l'unico effetto era quello di rendere la preda più vivace e più veloce nel fuggire quando rilevava la presenza, dietro di sé, del predatore. Questa fuga repentina portava spesso la preda a scontrarsi ed incastrarsi contro altri ostacoli. Una mancata variazione delle politiche di movimento e una mancata calibrazione della velocità rendevano praticamente uguali tutti i livelli; alcuni difetti collaterali rendevano letteralmente ingiocabili i livelli più alti.
   Preso atto di queste caratteristiche del gioco ci siamo posti l'obiettivo principale di correggere quelli, che a nostro parere, erano dei veri e propri difetti di giocabilità. Difetti che rendevano un gioco pensato e implementato con grande creatività, alla prova dei fatti noioso e frustrante.
   Per raggiungere questo obiettivo abbiamo cercato di rendere parzialmente prevedibili i movimenti della preda, in modo che l'utente fosse appagato, sia inizialmente che avanzando nel gioco, dalla sua stessa bravura nello scoprire gli algoritmi che muovono il robot preda. Questo con lo scopo di realizzare un gioco che letteralmente "chieda" al giocatore di osservare i movimenti della preda, comprenderne la regolarità e muoversi in anticipo rispetto alla preda per colpirla quando essa mostra dei momenti di vulnerabilità. Cruciale nelle fasi di progettazione e di implementazione delle nuove regole è stata la nostra volontà di rendere il gioco più facile nei bassi livelli e solo gradualmente più difficile fino a

raggiungere livelli alti di difficoltà, mantenendo però sempre costante il livello di "engaging".

Soluzioni

Abbiamo pensato che fosse possibile gratificare l'utente nei livelli iniziali pensando ad algoritmi di movimento abbastanza semplici (per permettere all'utente di prendere confidenza con il robot che controlla) per passare, poi, ad algoritmi di movimento decisamente più complessi e meno prevedibili nei livelli di gioco successivi.

Di seguito faremo un analisi comparata tra gli algoritmi che regolavano il movimento del robot preda nel vecchio gioco e quelli che lo modificano nel nuovo.

1. **Primo livello:** L'algoritmo che regola il movimento nel primo livello fa muovere la preda secondo traiettorie rettilinee ad una velocità molto bassa e le impone di girare di 100 gradi a destra ogni volta che il sensore posizionato sulla parte anteriore del robot incontra un ostacolo. La semplicità dell'algoritmo è dovuta al fatto che, basandoci sull'analisi del vecchio gioco (portata a termine facendo provare il gioco a 5 persone), l'impressione comune al primo approccio è stata quella di eccessiva difficoltà. Difficoltà dovuta al fatto che il controllo del robot predatore non è affatto semplice e all'utente serve tempo per prendere confidenza con questi comandi. Il primo livello implementato in precedenza prevedeva che il robot non seguisse traiettorie rettilinee, era in grado di curvare, eseguire giri su se stesso e individuare un ostacolo alle sue spalle (e individuare di conseguenza quando il robot predatore era alle sue spalle). Individuava barriere e ostacoli ad un'elevata distanza. Inoltre ogni volta che incontrava un ostacolo sceglieva con una politica random come aggirarlo, se girando a destra, a sinistra, o facendo un giro di 180 gradi su se stesso. L'unico parametro ad essere modificato tra livello e livello era la velocità del robot. Ma anche quest'ultima al primo livello si è rivelata eccessiva ed è stata conseguentemente diminuita. Eviteremo di ripetere per ogni livello le caratteristiche dell'algoritmo alla base del vecchio gioco.

2. **Secondo livello:** In questo livello si presuppone che l'utente abbia preso maggiore confidenza con il controllo del robot predatore. Allora si cercherà di far intuire all'utente che dovrà adattarsi ai livelli aumentando la propria destrezza nel comandare il robot e la sua abilità nel comprendere velocemente gli algoritmi che regolano il movimento della preda. Il robot si muoverà, quindi, più velocemente rispetto al primo livello (richiesta di maggior abilità nel controllo) ma varierà comportamento all'incontro di un ostacolo ruotando, questa volta, di 100 gradi a sinistra (in modo da evidenziare la differente implementazione di algoritmi di "fuga" tra livello e livello). I movimenti eseguiti dal robot preda fino a quando non incontra ostacoli seguiranno ancora traiettorie rettilinee.

3. **Terzo livello:** Il terzo livello riserva all'utente non poche novità. Infatti con questo livello abbiamo deciso di aumentare la complessità dei livelli: siamo passai da quelli che nei giochi sono visti come il livelli "facile" (livello 2) o livelli di "trainig" (livello 1) ad un livello "medio". Si presuppone che a questo punto l'utente abbia buona confidenza con il controller che regola i movimenti del robot predatore e riesca a muoverlo abbastanza agilmente. Per questo gli algoritmi diventano più articolati. Il robot girerà a destra per due volte quando incontrerà un ostacolo ma la terza volta girerà a sinistra. Questo comportamento confonderà il giocatore che impiegherà qualche secondo in più a capire qual è il "comportamento" del robot. Viene attivato, inoltre, il sensore posteriore del robot. Se questo sensore avverte una presenza a meno di 10 cm il robot preda accelererà (presumibilmente alle sue spalle

potrebbe esserci il robot predatore). Le traiettorie eseguite dal robot sono ancora rettilinee e la massima velocità a cui si muove resta invariata rispetto al secondo livello.

4. **Quarto livello:** Questo livello è la naturale evoluzione del livello precedente. Il robot continua a seguire traiettorie rettilinee nei movimenti di "fuga". La velocità a cui si muove è aumentata e l'algoritmo con cui evita gli ostacoli inizia ad essere difficilmente riconoscibile dall'utente. Infatti al 50% il robot ruoterà a destra incontrando un ostacolo ma nei casi restanti avrà un 50% di possibilità di girare a sinistra e altrettante possibilità di compiere un giro di 180 gradi su se stesso.

5. **Quinto livello:** Nel penultimo livello catturare la preda inizia a essere difficile anche per un giocatore esperto. Questo perché, nonostante la velocità di fuga della preda sia lasciata inalterata rispetto al livello precedente, il robot non si muove eseguendo movimenti rettilinei come nei livelli precedenti ma può curvare. Questo rende la "caccia" molto più difficile. Per non rendere il tutto eccessivamente difficile, è stata adottata una politica abbastanza semplice per far evitare gli ostacoli al robot: il robot eviterà gli ostacoli girando sempre a sinistra di 100 gradi.

6. **Sesto livello:** L'ultimo livello è ovviamente quello che mette il giocatore nelle condizioni di difficoltà massima. Il robot fugge a velocità doppia rispetto al livello precedente. Individua (grazie al sensore posteriore) se il robot preda è a una distanza inferiore di 20 cm e, in quel caso, accelera per sfuggirgli. Inoltre quando si troverà di fronte un ostacolo al 50% ruoterà a destra e, nei casi restanti, avrà un 50% di possibilità di girare a sinistra e altrettante possibilità di compiere un giro di 180 gradi su se stesso. Oltre a non compiere traiettorie rettilinee ha la possibilità di fermarsi e di girare su se stesso.

Conclusioni

Il gioco era già stato provato su un bambino di 9 anni che l'aveva trovato molto divertente, nonostante la preda si muovesse male e gli fosse stato impossibile, anche al più semplice dei livelli, completare i suoi obiettivi. Noi abbiamo pensato di provare la nostra versione modificata non solo su bambini della stessa fascia d'età del primo tester ma anche da nostri coetanei (tra i 22 e i 26 anni).
Questo con il preciso scopo di andare oltre il naturale entusiasmo dimostrato dai più giovani verso l'interazione con un robot e cercare di raccogliere pareri più distaccati sulle meccaniche e il funzionamento del gioco.
Tutti i tester a cui il gioco è stato sottoposto hanno percepito un miglioramento. Due nostri coetanei, sono riusciti a completare il primo livello dopo pochi minuti di gioco. Uno di essi ha terminato, giocando una ventina di minuti, i primi 3 livelli e si è dimostrato interessato, di livello in livello, a conoscere il successivo.
Tuttavia, nonostante queste conferme, era importante verificare che il gioco risultasse migliorato anche dalla prospettiva di un bambino. Infatti a poco sarebbe valso creare un gioco che incentivasse il proseguimento alle sue fasi successive se tale incentivo non fosse stato percepito anche dai più piccoli.
I due bambini, uno di 9 e l'altro di 6, che hanno provato il gioco dopo la nostra modifica, hanno trovato anch'essi più gradevole l'esperienza. Il bambino di 6 non ha cmq completato il primo livello ma si è avvicinato allo scopo molto di più rispetto a quanto aveva fatto con la versione originale del gioco. Il bambino di 9 anni ha invece completato il primo livello e si è cimentato con successo anche nel secondo.

In particolare quasi tutti i tester hanno compreso dopo pochi minuti di gioco l'importanza di osservare i comportamenti della preda con lo scopo di prevederne e anticiparne i movimenti. Ovviamente non tutti sono riusciti in questo intento. E' stato importante anche realizzare che nessuno di essi fosse in grado di completare gli ultimi tre livelli nel loro primo approccio al gioco. Una tale eventualità avrebbe significato che non eravamo stati abbastanza previdenti nel calibrare la difficoltà. Infatti, un gioco che può essere finito in pochi minuti non è meglio di un gioco talmente difficile da scoraggiare il giocatore in pochi minuti. Saremmo in tal caso passati da un eccesso all'altro. Il fatto che nessuno di essi abbia completato con successo gli ultimi 3 livelli ci lascia pensare che occorra una maggiore abilità nel controllare il robot predatore per terminarli. Ma a questo serve l'allenamento!
Il gioco deve costituire una sfida per il giocatore che non sia nè noiosa nè frustrante e la difficoltà riscontrata in questo progetto è stata proprio quella di mantenersi e ricavare spazi tra queste due soglie.

Appendice 1: Lista dei contenuti del CD

Il cd contiene

- Un file .pptx contenente una breve presentazione del progetto (15 slides)
- Il .pdf del project report
- Il .docx del project report
- Il .pdf dello User Manual
- Le immagini usate nella documentazione
- Il codice del robot preda e di quello predatore
- Video realizzati durante la prova in laboratorio

Abbiamo aggiunto per motivi di compatibilità (e come da specifica) anche altri due formati di files per la presentazione e il project report. Tali files si possono trovare nella cartella del cd chiamata “Altro Materiale”. Ricordiamo però che la formattazione potrebbe essere diversa dagli originali (.pptx e .docx)

- Un file .ppt contenente una breve presentazione del progetto (15 slides)
- Il .doc del project report

Appendice 2: Manuale dell’utente

Il manuale dell’utente si fa riferimento al manuale utente del gioco originale inserito all’interno dei contenuti del cd.

Appendice 3: Codice

Codice Predatore

```
//LEGO NXC PREDATOR
//Developed by G.G. P.P.
//Modified by M.C. M.F.
//


//BT define
#define BT_CONN 1
#define INBOX 1
#define OUTBOX 1
#define QUEUE 0


//Sensor define
#define TOUCH          SENSOR_1
#define SONAR          SENSOR_4
#define SONAR_VALUE      IN_4

//THRESHOLD define
#define THRESHOLD 40
#define NEAR_THRESHOLD 20 //cm
```

```
#define POWER 50
#define ROTATE_STEP 30
#define ROTATE 100



//CMD define
#define FORWARD       1
#define BACKWARD      2
#define TURNRIGHT     3
#define TURNLEFT      4
#define TURNAROUND    5
#define RESET_ROTATE  6
#define ATTACK        7
#define TURN180       8
#define BREAKALL      9

//semaphore variable
mutex sem;
mutex semBtouch;

//move variable
int rotate;
int move;
bool attack;
int Btouch = 0;

//check connection function
sub BTCheck(int conn){
  while (!BluetoothStatus(conn)==NO_ERR){
    TextOut(5,LCD_LINE2,"Error");
    Wait(1000);
    //Stop(true);
  }
  ClearScreen();
}//BTCheck(int conn)

//ParseStr
//parametri:
//- pcmd: stringa contenete il comando nel formato XXYYZZ
//- xcmdID: passato per indirizzo, indica l'ID del comando
//
//questa funzione parsa la stringa che compone il comando e ne ristituisce i singoli
//componenti.
//
//
sub ParseStr(string pcmd, int &cmdID)
{
  string temp;
  temp = SubStr(pcmd, 0, 2);//0 si riferisce alla posizione da cui iniziare, 2 al numero di caratteri da copiare.
  cmdID = StrToNum(temp);
}//ParseStr(string pcmd, int &cmdID)
```

```
//****************** Task ***********************************

task touch()
{
    while(true)
    {
     if(attack)
     {
            Acquire(semBtouch);
            if(Btouch == 0)
            {
              if(TOUCH == true)
                    Btouch = 1;
            }
            Release(semBtouch);
     }
    }
}//touch

sub setInbox(bool hit)
{
     SendResponseBool(QUEUE, hit);
}

task Attack()
{
    while(true)
    {
            if(attack)
            {
                  Acquire(sem);
                  RotateMotor(OUT_A, 90, 65);
                  for(unsigned short i; i<20 && Btouch==0; i++)
                  {
                          Wait(40);
                  }

                  Acquire(semBtouch);
                  if(Btouch == 1)
                  {
                        setInbox(1);
                        //PlayTone(3000, 1000);//3000 <- frquenza, 1000<- durata (in millisecondi)
                        Btouch = 0;
                  }
                  else
                  {
                       // setInbox(0);
                        //Btouch = 1;
                        //SendResponseBool(QUEUE, 0);
                  }

                  Release(semBtouch);
                  RotateMotor(OUT_A, -90, 65);
```

```
                attack = false;
                Release(sem);
            }
            else
            {
               Acquire(semBtouch);
               Btouch = 0;
               Release(semBtouch);
            }

    }
}//Attack


task Sonar()
{
    while (true)
    {
        if(SensorUS(IN_4)<NEAR_THRESHOLD)
        {
          Acquire(sem);

          PlayTone(3000, 1000);//3000 <- frquenza, 1000<- durata (in millisecondi)

          Off(OUT_BC);
          OnRevSync(OUT_BC, POWER, 0);
          Wait(800);
          OnRevSync(OUT_BC, POWER, -60);
          Wait(800);

          Release(sem);
         }
    }
}

task Forward()
{
    while (true)
    {
       Acquire(sem);
       int effPower = POWER + (move-1)*20;
       if(move>0)
       {

        if(effPower >= 100)
                effPower=100;
        OnFwdSync(OUT_BC, effPower, rotate);
        Wait(500);
        rotate = 0;
       }
       Release(sem);
    }
}
```

```
task Breakall()
{
    while(true)
    {
       Acquire(sem);
       if(move == 0 && rotate == 0)
            Off(OUT_ABC);
       Release(sem);
    }
}//Breakall()

task Backward()
{
    while(true)
    {

      Acquire(sem);
      int effPower = -1*move*POWER;
      if (move<0)
      {
         if(effPower >= 100)
                effPower=100;
         OnRevSync(OUT_BC, effPower, rotate);

         Wait(500);
         rotate = 0;
       }

       Release(sem);
    }
}



task Core()
{
    string in;//messaggio che contiene il comando
    int i = 0;//boh
    int cmd = 0;//comando estratto dal messaggio
    attack = false;
    BTCheck(0); //funzione bloccante che controlla la connessione BT con il software

    while(true)
    {
        TextOut(10,LCD_LINE1,"I'm NXT PREDATOR");
        ReceiveRemoteString(INBOX, true, in);
        ParseStr(in, cmd);
        TextOut(0,LCD_LINE2,"IN: ");
        TextOut(20,LCD_LINE2,in);

        switch(cmd)
        {
```

```
case BREAKALL:
    TextOut(0,LCD_LINE3,"breakall");

    Acquire(sem);
    move = 0;
    rotate = 0;
    Release(sem);
    break;

case ATTACK:
    TextOut(0,LCD_LINE3,"attack");

    Acquire(sem);
    //move = 0;
    //rotate = 0;
    attack = true;

    Release(sem);
    break;

case RESET_ROTATE:
    TextOut(0,LCD_LINE3,"reset rotate");

    Acquire(sem);
    rotate = 0;
    Release(sem);
    break;

case FORWARD:
    TextOut(0,LCD_LINE3,"forward");

    Acquire(sem);
    if(move<=0)
    {
       move = 1;
       rotate = 0;
    }
    else
       move = move + 1;
    Release(sem);
    break;

case BACKWARD:
    Acquire(sem);
    TextOut(0,LCD_LINE3,"backward");
    if(move>=0)
    {
       move=-1;
       rotate = 0;
    }
    else
       move-=1;
```

```
            Release(sem);
            break;

        case TURNRIGHT:
            Acquire(sem);
            if(move == 0)
            {
              TextOut(0,LCD_LINE3,"turn right");
              //OnFwdSync(OUT_BC, 75, -75);
              RotateMotorEx(OUT_BC, 80, ROTATE, 100, true, true);
              Wait(360);
            }
            else
              rotate += ROTATE_STEP;
            Release(sem);
            break;

        case TURNLEFT:
            Acquire(sem);
            if(move == 0)
            {
              TextOut(0,LCD_LINE3,"turn left");
              //OnFwdSync(OUT_BC, 75, +75);
              RotateMotorEx(OUT_BC, 80, ROTATE, -100, true, true);
              Wait(360);
            }
            else
              rotate -= ROTATE_STEP;
            Release(sem);
            break;

        case TURN180:
            Acquire(sem);

            move = 0;
            TextOut(0,LCD_LINE3,"turn180");
            RotateMotorEx(OUT_BC, 80, 360, -100, true, true);
            Wait(360);

            Release(sem);
            break;

      }
      TextOut(0,LCD_LINE5,"move: ");
      TextOut(40,LCD_LINE5,NumToStr(move));
      TextOut(0,LCD_LINE5,"rotate: ");
      TextOut(40,LCD_LINE6,NumToStr(rotate));

      cmd = 0;
    }
}
```

```
task main(){
    SetSensor(IN_1,SENSOR_TOUCH);
    SetSensorLowspeed(IN_4);
    //Precedes(Core, Forward, Backward, Breakall, Attack);
    Precedes(Core, Forward, Backward, Breakall, Attack, touch);
}
```

Codice preda

```
//preda_0.1b.nxc
//last modified 20-09-10
//Developed by GIGISOMMO & GIUBBRY (EL GABRY)
//Modified by M.C. & M.F.


//BT define
#define BT_CONN 1
#define INBOX 1
#define OUTBOX 1
#define QUEUE 0



//DEFINE SENSOR
#define MIC_FRONT             SENSOR_2
#define MIC_REAR              SENSOR_3
#define TOUCH                 SENSOR_1
#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 ESCAPE_THRESHOLD 20
#define NEAR_THRESHOLD 35
```

```
//was #define NEAR_FRONT_THRESHOLD10
#define NEAR_FRONT_THRESHOLD 25

#define NEAR 20 //cm
#define FORWARD 1
#define BACKWARD 2
#define TURNLEFT 4
#define TURNRIGHT 3
#define TURNAROUND 5
#define STOP 6
#define BREAKALL 9
//end movement define

#define POWER 25
#define ROTATE 20


mutex sem;              //variabile semaforo
mutex semSonar;

short level = 0;

int rotate= 0;            //angolo di rotazione dalla direzione principale
int power = 0;              //potenza proporzionale all'intensità del suono
bool directionFront;    //se true scappa in avanti, se false scappa indietro
int micDiffOld = 1000;
int cont=0;

//Level varibles
int maxPower = -1;
int maxRotate = -1;
int maxNearRearThreshold = -1;


task stopTH();
//******************* Task ***********************************

//check connection function
sub BTCheck(int conn){
  while (!BluetoothStatus(conn)==NO_ERR){
    TextOut(5,LCD_LINE2,"Error");
    Wait(1000);
  }
  ClearScreen();
}//BTCheck(int conn)

//ParseStr
//parametri:
//- pcmd: stringa contenete il comando nel formato XXYYZZ
//- xcmdID: passato per indirizzo, indica l'ID del comando
//
//questa funzione parsa la stringa che compone il comando e ne ristituisce i singoli
//componenti.
```

```
//
//
short ParseStr(string pcmd)
{
   string temp;
   temp = SubStr(pcmd, 0, 2);//0 si riferisce alla posizione da cui iniziare, 2 al numero di caratteri da copiare.
   short tempShort;
   tempShort = StrToNum(temp);
   return tempShort;
}//ParseStr(string pcmd, int &cmdID)



sub GetLevel()
{
   ClearScreen();
   string in;//messaggio che contiene il comando
   while(level < 1)
   {
        ReceiveRemoteString(INBOX, true, in);
        level = ParseStr(in);
        if(level > 6)
              level = -1;
        TextOut(0,LCD_LINE4,"A che livello vuoi giocare?!");
   }
    TextOut(0,LCD_LINE4,NumToStr(level));
    Wait(1000);
}


sub SetLevelParam()
{
   switch(level)
   {
            case 1:
               maxPower = 20;
               maxRotate = 0;
               maxNearRearThreshold = 5;
            break;
            case 2:
               maxPower = 40;
               maxRotate = 0;
               maxNearRearThreshold = 5;
            break;
            case 3:
               maxPower = 40;
               maxRotate = 0;
               maxNearRearThreshold = 10;
            break;
            case 4:
               maxPower = 50;
               maxRotate = 0;
               maxNearRearThreshold = 10;
```

```
            break;
            case 5:
                maxPower = 50;
                maxRotate = 100;
                maxNearRearThreshold = 20;
            break;
            case 6:
                maxPower = 100;
                maxRotate = 100;
                maxNearRearThreshold = 20;
            break;
    }
}//SetLevelParam




task moveTH()
{
    short movement;
    bool varStop;
    while(level > 0)
    {
       Acquire(sem);

       switch(level)
       {
          case 1:
             power = maxPower;
             varStop = true;
          break;
          case 2:
             power = maxPower;
             varStop = true;
          break;
          case 3:
              power = maxPower;
              varStop = true;
          break;
          case 4:
              power = maxPower;
              varStop = true;
          break;
          case 5:
              power = maxPower;
                if(cont==0)  {
                     power = maxPower;
                     varStop = true;
                     rotate=0;
                     cont++;
                }
                else{
```

```
                                    movement = Random(3)+3;

                                    switch(movement)
                                    {
                                      case TURNLEFT:
                                        varStop = false;
                                        rotate += ROTATE;
                                        cont=0;
                                        if(rotate > maxRotate)
                                              rotate = maxRotate;
                                      break;
                                      case TURNRIGHT:
                                        varStop = false;
                                        rotate -= ROTATE;
                                        cont=0;
                                        if(rotate < -maxRotate)
                                              rotate = -maxRotate;
                                      break;
                                      case STOP:
                                        varStop = true;
                                        rotate = 0;
                                        cont=0;
                                        power = 0;
                                      break;
                                      default:
                                        rotate = 0;
                                      break;
                                    }
                                    varStop = true;
                                    }
break;
case 6:
        movement = Random(6)+1;

                                    switch(movement)
                                    {
                                      case FORWARD:
                                        power += maxPower;
                                        if(power > 100)
                                              power = maxPower;
                                        varStop = true;
                                      break;
                                      case BACKWARD:
                                        power -= POWER;
                                        if(power < -maxPower)
                                              power = -maxPower;
                                        varStop = true;
                                      break;
                                      case TURNLEFT:
                                        varStop = false;
                                        rotate += ROTATE;
                                        if(rotate > maxRotate)
                                              rotate = maxRotate;
```

```
                                        break;
                                        case TURNRIGHT:
                                            varStop = false;
                                            rotate -= ROTATE;
                                            if(rotate < -maxRotate)
                                                    rotate = -maxRotate;
                                        break;
                                        case STOP:
                                            varStop = true;
                                            rotate = 0;
                                            power = 0;
                                        break;
                                        default:
                                            rotate = 0;
                                        break;
                                    }
            break;
        }

        unsigned short moveTime = (Random(3)+1)*1000;

        if(power == 0 && varStop == true)
              Off(OUT_BC);
        else if(power <= 10 && power >= -10 && varStop == false)
        {
              OnFwdSync(OUT_BC, 50, rotate);
              PlayTone(8000, 200);//3000 <- frquenza, 1000<- durata (in millisecondi)
              moveTime /= 2;
        }
        else
              OnFwdSync(OUT_BC, power, rotate);
        //si muove per un tempo random

        Release(sem);
        unsigned short numWait = moveTime/200;
        for(unsigned short i = 0; i<numWait; i++)
            Wait(200);//da 1 a 4 seconddi di movimento random
    }
}//moveTH

task SonarTH()
{

    while(level > 0)
    {
        if(SensorUS(SONAR_REAR_VALUE) < maxNearRearThreshold)
        {
          Acquire(sem);

          PlayTone(1000, 200);//3000 <- frquenza, 1000<- durata (in millisecondi)

          Off(OUT_BC);
          //outputs, pwr, angle, turnpct, sync, stop
```

```
 OnFwdSync(OUT_BC, 100, 0);
 for(unsigned short i = 0; i<5; i++)
        Wait(200);//da 1 a 4 seconddi di movimento random

 Release(sem);
}
if(SensorUS(SONAR_FRONT_VALUE) < NEAR_FRONT_THRESHOLD)
{
 Acquire(sem);

 switch(level)
 {
  case 1:
     Off(OUT_BC);
     //outputs, pwr, angle, turnpct, sync, stop
     RotateMotorEx(OUT_BC, -50, 360, 100, true, true);
  break;
  case 2:
     Off(OUT_BC);
     //outputs, pwr, angle, turnpct, sync, stop
     RotateMotorEx(OUT_BC, -50, 360, -100, true, true);
  break;
  case 3:
     Off(OUT_BC);
     //outputs, pwr, angle, turnpct, sync, stop
     if(cont==2) {
           RotateMotorEx(OUT_BC, 50, 360, -100, true, true);
           cont=0;}
     else{
           RotateMotorEx(OUT_BC, 50, 360, 100, true, true);
           cont++;
     }
  break;
  case 4:
     Off(OUT_BC);
     //outputs, pwr, angle, turnpct, sync, stop
     if(Random(2)==0) {RotateMotorEx(OUT_BC, 50, 360, 100, true, true); }
     else {if (Random(2)==0) {RotateMotorEx(OUT_BC, 60, 360, -100, true, true);}
     else {RotateMotorEx(OUT_BC, 70, 360, -180, true, true);}}
  break;
  case 5:
     Off(OUT_BC);
     //outputs, pwr, angle, turnpct, sync, stop
     RotateMotorEx(OUT_BC, -50, 360, 100, true, true);
  break;
  case 6:
     Off(OUT_BC);
     //outputs, pwr, angle, turnpct, sync, stop
     if(Random(2)==0) {RotateMotorEx(OUT_BC, 70, 360, 100, true, true); }
     else {if (Random(2)==0) {RotateMotorEx(OUT_BC, 70, 360, -100, true, true);}
     else {RotateMotorEx(OUT_BC, 70, 360, -180, true, true);}}
  break;
```

```
        }

         Release(sem);
        }
    }
}


task touchTH()
{
    while(level > 0)
    {
            if(TOUCH == 1)
            {
                SendResponseBool(QUEUE, 1);
                PlayTone(3000, 1000);//3000 <- frquenza, 1000<- durata (in millisecondi)
                TextOut(0,LCD_LINE4,"touchTH: yes touch");
                Acquire(sem);

                Off(OUT_BC);

                Wait(3000);

                SendResponseBool(QUEUE, 0);
                Release(sem);
            }
            else
            {
                TextOut(0,LCD_LINE4,"touchTH: no touch");
            }
    }

}//check_sensors


task InitTH()
{
    Acquire(sem);
    Off(OUT_ABC);
    GetLevel();
    SetLevelParam();
    Release(sem);
    Precedes(SonarTH, moveTH, touchTH, stopTH);

}//InitTH

task stopTH()
{
   string in;//messaggio che contiene il comando
   //finchè il server non manda un livello "NON VALIDO"
   while(level > 0)
   {
        ReceiveRemoteString(INBOX, true, in);
```

```
        short temp = ParseStr(in);
        ClearScreen();
        TextOut(0,LCD_LINE5,NumToStr(temp));
        Wait(1000);
        if(temp == 9)
        {
                level = 0;
                ExitTo(InitTH);
        }
   }
}//levelTH


task main(){
    SetSensorTouch(IN_1);
    SetSensorLowspeed(IN_4);
    SetSensorLowspeed(IN_3);

    BTCheck(0); //funzione bloccante che controlla la connessione BT con il software
    Precedes(InitTH);
}
```