UNIVERSITATEA "ŞTEFAN CEL MARE" SUCEAVA
FACULTATEA DE INGINERIE ELECTRICĂ ŞI ŞTIINŢA CALCULATOARELOR

PROIECT DE LICENŢĂ

ACHIZIŢIA ŞI RECUNOAŞTEREA MIŞCĂRILOR CAPULUI PENTRU COMANDA GESTUALĂ A JOCURILOR PE CALCULATOR

Coordonator ştiinţific:

Şef lucrări univ. dr. ing. Radu-Daniel VATAVU

Absolvent:

OROŞANU Luiza

- Suceava, 2010 -

DECLARAȚIE

Subsemnata, Oroşanu Luiza, absolventă a Facultății de Inginerie Electrică şi Ştiința Calculatoarelor, specializarea Calculatoare, declar pe propria răspundere că proiectul de licență cu titlul "Achiziția şi recunoaşterea mişcărilor capului pentru comanda gestuală a jocurilor pe calculator" este rezultatul muncii mele, pe baza informațiilor obținute la cursuri şi din alte surse, inclusiv de pe Internet, care au fost citate şi indicate, conform normelor etice în bibliografie. Declar că nu am folosit în mod tacit sau ilegal munca altora şi că nici o parte din proiect nu încalcă drepturile de proprietate intelectuală ale cuiva, persoană fizică sau juridică. Declar că proiectul nu a mai fost prezentat sub această formă vreunei instituții de învățământ superior din țară sau străinătate în vederea obținerii unui grad sau titlu ştiințific ori didactic.

Suceava, 2010

Absolvent

_____

(semnătura în original)

Cuprins

1. Introducere.....4

2. Achiziția mişcărilor capului folosind Wiimote.....10

3. Algoritm de învățare a comenzilor dependent de utilizator.....19

3.1 Detecția mişcării.....19

3.2 Segmentarea traiectoriilor de mişcare.....21

3.3 Analiza şi interpretarea datelor.....33

4. Recunoaşterea mişcărilor gestuale direcționale.....36

5. Aplicații demonstrative.....40

6. Concluzii.....62

Bibliografie.....64

Sinteza lucrării

Lucrarea se bazează pe ideea că utilizatorii se implică emoțional sau afectiv în jocuri prin mişcări ale corpului de manieră inconştientă în timp ce controlează acțiunea, iar mişcarea predominantă poate fi detectată şi folosită pentru comanda efectivă(şi implicit naturală) a jocului.

În cadrul lucrării am realizat un modul de urmărire a mişcărilor capului care, asociat oricărei aplicații, poate învăța gesturi utile pentru comanda acesteia. Demonstrația s-a realizat cu ajutorul telecomenzii Wii Remote care a fost folosită pentru detectarea poziției capului prin intermediul unor ochelari prevăzuți cu leduri IR.

Modul de urmărire presupune atât urmărirea controlului acțiunii din joc, cât şi a mişcărilor efectuate de către utilizator pe durata controlului detectat. Pentru urmărirea controlului din joc se vor importa din sistemul de operare librăriile dll necesare controlului global al evenimentelor tastaturii şi mouse-ului. Urmărirea mişcărilor capului pe durata controlului acțiunii presupune salvarea unei liste de puncte consecutive, aceste puncte reprezentând deplasarea poziției centrale dintre cele două semnale IR. Această listă va fi ulterior filtrată prin algoritmul lui Douglas-Peucker, cu intenția de a evidenția traiectoria parcursă. Mişcările recunoscute în cadrul listei filtrate, vor fi asociate controlului urmărit. Pe baza datelor astfel obținute, se va crea în final un dicționar de comenzi ce va cuprinde asocierile dintre tipul controlului şi mişcarea capului, şi se vor stabili pragurile pentru mişcările capului.

Tipul mişcării asociate unui control, cât şi pragul determinat acesteia, depind de mişcările utilizatorului în timpul controlului: astfel, pentru utilizatori mai impulsivi, mişcarea va fi asociată corect, având un prag rezonabil, pe când în cazul utilizatorilor pasivi, există posibilitatea de a asocia o mişcare greşită controlului, sau cu un prag de valoare mică.

Dacă în modul de urmărire, aplicația „asculta” evenimentele tastaturii şi mouse-ului, urmărind în paralel poziția capului, în modul comandă aplicația va urmări în principal poziția capului, iar pe baza mişcării voluntare detectate se va trimite în paralel comanda(de apăsare a tastei asociate mişcării detectate) către joc. Trimiterea comenzii va fi dependentă de tipul aplicației, în sensul că va necesita setări diferite pentru aplicații diferite.

Jocurile alese pentru susținerea lucrării au fost Tetris, Pac-Man, Need For Speed Underground 2 şi Quake 3 Arena. Acestea au fost alese în ordinea crescătoare a implicării utilizatorilor în cadrul controlului acțiunii: Tetris, un joc monoton, urmat de Pac-Man, care implică un anumit nivel de stres, şi apoi de Need For Speed şi Quake, a căror implicare afectivă să fie cea mai mare.

1. Introducere

În ultimii ani s-au evidențiat dorința și nevoia simplificării dialogului om – sistem informatic în așa manieră încât utilizatorul să nu mai fie limitat la folosirea dispozitivelor auxiliare ci să poate comunica într-un mod natural și eficient comenzi simple care să fie interpretate și executate corect de către sistemul informatic.

Pentru rezolvarea acestei probleme am considerat problema achiziției și recunoașterii unei colecții generale de gesturi umane care să înlocuiască vechile apăsări de taste sau mișcări de mouse. Ideea de bază este de a urmări gesturile utilizatorilor, felul în care aceștia reacționează în anumite condiții, și nu de a forța alte gesturi doar pentru a schimba controlul aplicației.

Un gest este o formă de comunicare non-verbală în care o mișcare vizibilă a corpului comunică un mesaj special, înlocuind uneori vorbele sau dând mai multă expresivitate vorbirii. Gesturile includ mișcări ale mâinilor, capului, feței sau altor părți ale corpului, mișcări care transmit informații sau sunt însoțite de semnificații și conținut.

Recunoașterea gesturilor presupune interpretarea gesturilor umane prin algoritmi matematici. Această interpretare reprezintă o modalitate prin care calculatorul va putea să înțeleagă intențiile utilizatorului, în acest fel realizându-se o comunicare naturală care nu va mai fi limitată de controlerele uzuale.

Recunoașterea gesturilor se bazează în general pe tehnici de procesare a imaginilor, prin intermediul camerelor web conectate la calculator, sau pe tehnici de monitorizare a diferitor tipuri de sensori.

În 1982, producătorul Amiga a lansat controlerul tip placă de echilibru, JoyBoard, drept dispozitiv periferic pentru consola Atari 2600(1). Jocuri precum Slalom Skiing, Off your Rocker și Surf's Up puteau fi controlate prin înclinarea plăcii într-o anumită direcție.

Figura 1.1 Placa de echilibru JoyBoard[1]

[1] Sursa imaginii: http://www.atariage.com/2600/controllers/con_AmigaJoyboard.jpg

În 1988, Nintendo a lansat controlerul tip covor, Power Pad[2]. Este un covor gri cu 12 senzori de presiune încorporați între 2 straturi de plastic flexibil. Dispozitivul se poziționează în fața ecranului. Jocuri precum Athletic World, Dance Aerobics, Dance Dance Revolution puteau fi controlate prin apăsarea butoanelor mari, testând astfel sincronizarea, memoria, viteza şi coordonarea jucătorilor.

Figura 1.2 Controlerul tip Covor, PowerPad[2]

În 1989, Nintendo a lansat controlerul tip mănuşă, Power Glove[3]. Mănuşa are ataşat şi un controler cu butoane tradiționale marca Nintendo. Jocuri precum Super Glove Ball şi Bad Street Brawler puteau fi controlate utilizând controlerul respectiv, dar şi mişcări specifice ale mâinii, mişcări detectabile doar prin intermediul mănuşii.

Figura 1.3 Controlerul tip mănusă, PowerGlove[3]

În 1995, William Freeman şi Craig Weissman au studiat modurile în care un telespectator poate controla televizorul de la distanță [4]. S-a urmărit găsirea unor gesturi simple, cât mai naturale, care să înlocuiască necesitatea folosirii a altor dispozitive, implicit a apăsării unor butoane. Soluția dezvoltată presupunea folosirea a două gesturi: palmă deschisă şi palmă închisă. Palma deschisă producea afişarea meniului de control,

[2] Sursa imaginii: http://ui22.gamespot.com/565/editpowerpad.jpg
[3] Sursa imaginii: http://www.gamersgraveyard.com/repository/nes/peripherals/images/powerglove.jpg

cu opțiuni a căror valori puteau fi incrementate sau decrementate prin mișcarea mâinii. Palma închisă reprezenta decizia de a părăsi modul de control, ceea ce implica închiderea meniului.

În 1998, Lars Bretzner şi Tony Lindeberg au studiat mişcarea 3D a obiectelor, mişcare controlată prin gesticularea mâinii [5]. Se realiza în acest sens o corespondență între cele 3 axe şi 3 degete ale utilizatorului.

În 1999, Richard Marks a conceput dispozitivul EyeToy, după ce a asistat la o demonstrație de PlayStation2 în SanJose, California[6]. Ideea lui Marks a fost de a realiza o interfață naturală pentru jocuri de realitate virtuală, folosind un webcam ieftin şi puterea de calcul a consolei Playstation 2.

Figura 1.4 Dispozitivul EyeToy[4]

Tehnologia se bazează pe informațiile primite de la cameră, acest lucru oferind utilizatorilor posibilitatea de a interacționa cu jocuri prin intermediul mişcării, detecției culorii şi a sunetului(datorită microfonului încorporat).

De exemplu, în Kung Foo utilizatorul trebuie să îndepărteze luptătorii ninja cu mâinile pe măsură ce aceştia vin zburând din toate părțile. În UFO Juggler utilizatorul trebuie să ajute navele spațiale să decoleze, învârtindu-le la viteza potrivită. Jocul de dans Boogie Down testează ondularea mâinii. Un exemplu de joc mai complex este Rocket Rumble, în care utilizatorul trebuie să atingă pe rând focurile de artificii pentru a le activa, după care să apese detonatorul.

În continuare vom considera compania *GestureTek*[7]. GestureTek este liderul mondial în tehnologia controlului prin gesturi, fiind remarcat prin gama largă de produse interactive oferite, gen suprafețe multi-touch, dispozitive şi jocuri.

[4] Sursa imaginii: http://www.baixakijogos.com.br/noticias-img/20080815/123eyetoy.jpeg

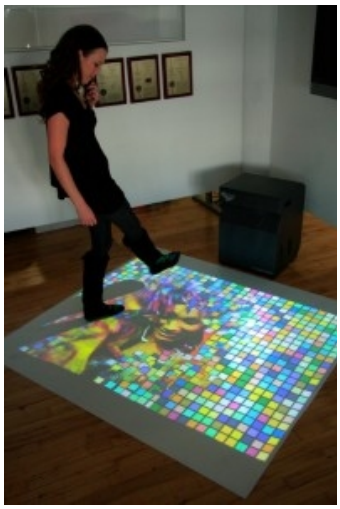


Figura 1.5 Podea interactivă marca GestureTek[5]

Dispozitivele multi-touch oferite sunt de tipul ecranelor, meselor, panourilor sau ferestrelor. Sistemele interactive de proiecție controlate prin gesturi produc efecte speciale şi jocuri interactive pentru pardoseli, pereți, mese şi ferestre, acest tip de produse fiind în principal destinat divertismentului.

Un alt domeniu în care tehnologia GestureTek îşi pune amprenta este cel al sănătății. Tehnologia de control prin gesturi s-a dovedit a fi foarte benefică pentru persoanele cu dizabilități. Terapia fizică bazată pe realitatea virtuală controlată prin mişcări, stimuli multisenzoriali, jocuri de lumini oferă pacienților o experiență unică care le îmbunătățește atât abilitățile fizice, cât şi cele cognitive.

Interacțiunea dintre utilizator şi aplicații a fost îmbunătățită simțitor în 2006, odată cu introducerea consolei *Wii Nintendo*(8). Dacă în trecut utilizatorii erau limitați de controlerele convenționale de joc, ei acum au posibilitatea de a interacționa în mod direct şi natural cu jocul, prin intermediul senzorilor de mişcare prezenți în controlerul Wiimote. Caracteristica principală a telecomenzii o reprezintă capacitatea de detecție a mişcării, care permite utilizatorului să interacționeze cu şi să manipuleze obiectele de pe ecran, datorită tehnologiei cu senzori optici şi al accelerometrului încorporat.

Figura 1.6 Folosirea controlerelor marca Wii[6]

[5] Sursa imaginii: http://www.dailydooh.com/wp-content/uploads/2009/12/dancemoves-199x300.jpg
[6] Sursa imaginii: http://cache.kotaku.com/assets/resources/2007/05/wii.jpg

Telecomanda Wii urmăreşte acceleraţia de-a lungul a 3 axe, prin intermediul unui accelerometru ADXL330. Mai dispune de asemenea şi de un senzor optic care îi permite să determine în ce directie este orientat controlerul.

Detecţia poziţiei şi a mişcării efectuate permit utilizatorilor să mimeze acţiunile specifice jocului, cum ar fi legănarea unei săbii sau ţintirea inamicului.

Recent, toţi cei trei principali producători de console(Microsoft Xbox 360, Sony PlayStation, Nintendo) au anunţat noi interfeţe gestuale. Nintento a introdus Wii Balance Board anul trecut, un dispozitiv capabil să detecteze presiunea şi mişcarea de pe podea. Anul acesta compania a lansat Wii MotionPlus, o extensie a controlerului Wii Remote care permite sistemului să detecteze mişcări mult mai complexe şi mai fine.

În 2009, Sony a prezentat prototipul pentru *PS3 Wand*, o tijă portabilă care foloseşte atât senzori interni, cât şi tehnici de procesare a imaginii prin intermediul camerei PlayStation Eye, pentru a urmări şi interpreta mişcările efectuate.

Figura 1.7 Controlerul PS3 Wand[7]

Microsoft a anunţat proiectul *Natal*, un sistem de senzori care renunţă în întregime la controlere în favoarea unei mulţimi de camere şi microfoane capabile să controleze aplicaţiile prin recunoaşterea mişcărilor şi a vocii.

Figura 1.8 Proiectul Natal[8]

[7] Sursa imaginii: http://www.thesixthaxis.com/wp-content/uploads/2009/12/wand_2_610_w500.jpg
[8] Sursa imaginii: http://www.digitalextremity.com/wp-content/uploads/2009/10/you-are-the-controller.jpg

Atât proiectanții, cât şi jucătorii văd controlul gestual drept efectuarea unor "acțiuni". Aceste acțiuni trebuie să tindă cât mai mult posibil către corespondențe realistice. Gesturile de intrare devin mult mai intuitive şi atractive când sunt asemănătoare cu corespondentul lor din lumea reală. Şi jocurile devin mai agreabile atunci când răspund la aceste gesturi în moduri mai sofisticate şi realistice.

Toate proiectele viitoare ale marilor companii se vor baza pe tehnologii de înaltă rezoluție, cu speranța de a capta şi înțelege mişcarea mai în detaliu. Scopul este simularea realității, atât prin diminuarea decalajului dintre acțiunea utilizatorului şi răspunsul primit de la sistem, cât şi prin folosirea unei grafici care să redea detalii cât mai realiste.

2. Achiziția mişcărilor capului folosind Wiimote

Wii Remote(poreclit Wiimote) este controlerul principal al consolei Wii. Spre deosebire de majoritatea controlerelor pentru jocuri, prevăzute doar cu butoane, acesta este dotat şi cu o cameră IR, precum şi capabilități de detecție a mişcării.

Figura 2.1 Controlerul Wiimote[9]

Modelul imită formatul unei telecomenzi normale de televizor, ceea ce implică folosirea unei singure mâini(datorită modelului simetric, aceasta putând fi folosită atât de dreptaci, cât şi de stângaci). Controlerul măsoară 148 mm în lungime, 36.2 mm în lățime şi 30.8 mm grosime. Numărul modelului este RVL-003 ca referință pentru numele de cod al proiectului, „Revolution". Controlerul comunică wireless cu consola prin unde radio Bluetooth într-o rază de acțiune de până la 10 metri.

Controlerul este prevăzut cu 12 butoane, 11 frontale şi un al 12-lea dedesubt. Cele 4 leduri albastre de la bază indică inițial nivelul bateriei, după care numărul controlerului conectat.

[9] Sursa imaginii: http://www.scarlet.an/store/images/wiimote.png

Camera IR, produsă de către PixArt, este amplasată la capătul superior al controlerului. Aceasta poate detecta până la 4 lumini IR, primind constant informații referitoare la poziția şi dimensiunea acestora. Dintre caracteristici amintim rezoluția de 1024x768, rata de reîmprospătare de 100Hz şi câmpul orizontal de vedere de 45°.

Telecomanda Wiimote poate fi conectată cu uşurință şi la un calculator, tot prin intermediul unei legături wireless *Bluetooth*[10]. Când intră în contact cu protocolul de descoperire a serviciilor Bluetooth(SDP-Service Discoverz Protocol), Wiimote-ul va furniza următoarele informații:

- Nume: Nintendo RVL-CNT-01;
- Id-ul vânzătorului: 0x057e;
- Id-ul produsului: 0x0306;
- Clasa dispozitivului major: 1280;
- Clasa dispozitivului minor: 4;
- Clasa serviciului: 0.

Pentru o conexiune reuşită, trebuie urmați 3 paşi:

- În cazul în care calculatorul nu este prevăzut cu un adaptor Bluetooth intern, este necesară conectarea unui adaptor USB Bluetooth extern. Acesta vine la pachet cu driver-ul Bluesoleil şi kit-ul de instalare.

Figura 2.2 Adaptor USB Bluetooth [11]

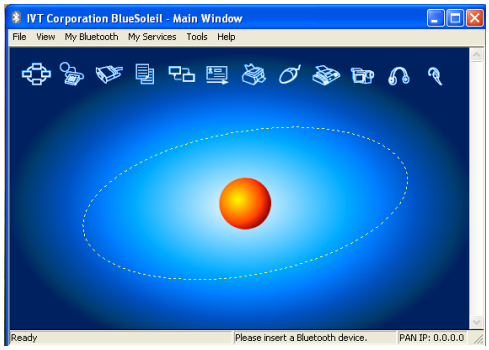


Figura 2.3 Software Bluesoleil

[10] *Bluetooth-ul* este un set de specificații bazate pe unde radio, pentru o rețea wireless personală (PAN - personal area network). Bluetooth-ul creează o cale prin care se poate face schimb de informații între aparate precum telefoane mobile, laptop-uri, calculatoare personale, imprimante, camere digitale şi console video printr-o frecvență radio sigură şi de rază mică.

[11] Sursa imaginii: http://www.serioux.com/files/product/1/m_583.jpg

- În continuare se va porni software-ul Bluesoleil şi se va activa adaptorul Bluetooth (dacă este necesar).

- Un click pe cercul central va determina programul să caute dispozitivele Bluetooth din apropiere. Pentru recunoaşterea controlerului Wiimote va fi necesară apăsarea butonului roşu de sincronizare situat sub carcasă, în apropierea bateriilor. Apăsarea acestui buton va determina setarea dispozitivului pe modul de descoperire. Când Wiimote-ul este recunoscut, numele său(NINTENDO-RVT-CNT-01) va apărea pe ecran. Un dublu click pe iconița lui, urmat de un click-dreapta şi selectarea opțiunii Connect Device va determina realizarea corectă a legăturii la pc.

Conectarea la dispozitivul HID astfel recunoscut se bazează pe librăria *WiimoteLib v1.7 realizată de către Brian Peek în 2009*[9]. Informațiile necesare din cadrul acestei librării sunt cele de realizare a comunicației cu controlerul Wiimote şi de preluare a informațiilor de la camera IR privind poziția ledurilor IR.

Când Wiimote-ul este conectat la pc, va fi identificat ca un dispozitiv HID. Astfel, pentru a ne conecta la controler trebuie să folosim HID(Human Interface Device) şi Device Management Win32 APIs(Application Programming Interface). Nu există nici un suport încorporat pentru aceste API-uri în arhitectura .NET curentă, din acest motiv fiind necesară introducerea de domenii P/Invoke(Platform Invoke). P/Invoke permite apelarea metodelor API Win32 din platforma .NET.

Realizarea comunicației cu controlerul Wiimote[10][11] presupune urmatorii paşi:

- Se preia identificatorul global unic al clasei HID definită de către Windows

```
// read/write handle to the device
private SafeFileHandle mHandle;

// a .NET stream to read/write from/to
private FileStream mStream;
bool found = false;
Guid guid;
uint index = 0;

// 1. get the GUID of the HID class
HIDImports.HidD_GetHidGuid(out guid);
```

- Se preia handle-lul listei ce cuprinde toate dispozitivele din clasa HID respectivă

```
// 2. get a handle to all devices of the HID class
IntPtr hDevInfo = HIDImports.SetupDiGetClassDevs(ref guid, null, IntPtr.Zero,
HIDImports.DIGCF_DEVICEINTERFACE);// | HIDImports.DIGCF_PRESENT);

// create a new interface data struct
HIDImports.SP_DEVICE_INTERFACE_DATA diData = new HIDImports.SP_DEVICE_INTERFACE_DATA();
diData.cbSize = Marshal.SizeOf(diData);
```

- Sunt enumerate toate dispozitivele şi se preia informații detaliate despre fiecare
- Se compară id-urile vânzătorului şi produsului ale fiecărui dispozitiv din listă cu id-urile cunoscute alea dispozitivului Wiimote.
- Când este găsit, se crează un obiect FileStream pentru a citi/scrie de la/către dispozitiv

```
// 3. Enumerate all devices
while(HIDImports.SetupDiEnumDeviceInterfaces(hDevInfo, IntPtr.Zero, ref guid, index, ref diData))
{
   HIDImports.SP_DEVICE_INTERFACE_DETAIL_DATA diDetail = new
HIDImports.SP_DEVICE_INTERFACE_DETAIL_DATA();
   diDetail.cbSize = 5;
   UInt32 size = 0;


    HIDImports.SetupDiGetDeviceInterfaceDetail(hDevInfo, ref diData, IntPtr.Zero, 0, out size, IntPtr.Zero);
   if (HIDImports.SetupDiGetDeviceInterfaceDetail(hDevInfo, ref diData, ref diDetail, size, out size, IntPtr.Zero))
   {
      mHandle = HIDImports.CreateFile(diDetail.DevicePath, FileAccess.ReadWrite, FileShare.ReadWrite, IntPtr.Zero,
FileMode.Open, HIDImports.EFileAttributes.Overlapped, IntPtr.Zero);

      HIDImports.HIDD_ATTRIBUTES attrib = new HIDImports.HIDD_ATTRIBUTES();
      attrib.Size = Marshal.SizeOf(attrib);

      if(HIDImports.HidD_GetAttributes(mHandle.DangerousGetHandle(), ref attrib))
      {
         // 4. if the vendor and product IDs match up
         if(attrib.VendorID == VID && attrib.ProductID == PID)
         {
            // 5. create a .NET FileStream wrapping the handle above
            mStream = new FileStream(mHandle, FileAccess.ReadWrite, REPORT_LENGTH, true);
         }
         else
            mHandle.Close();
      }
   }
   index++;
}
```

- Se goleşte lista dispozitivelor.

```
// 6. clean up our list
HIDImports.SetupDiDestroyDeviceInfoList(hDevInfo);
```

În cadrul dispozitivelor de interfațare umană(HID), datele sunt trimise şi primite sub forma de rapoarte(buffer de date cu o lungime predefinită, cu un antet care determină tipul raportului conținut în buffer). Wiimote-ul va trimite şi va primi rapoarte variate, toate având o dimensiune de 22 octeți.

Intrare/Ieşire	ID	Dimensiune	Funcție
Ieşire	0x10	1	Necunoscută
Ieşire	0x11	1	LED-uri
Ieşire	0x12	2	Modul de raportare a datelor
Ieşire	0x13	1	Camera IR activată
Ieşire	0x14	1	Boxă activată
Ieşire	0x15	1	Cerere de obținere a stării
Ieşire	0x16	21	Scrie memoria şi regiştrii
Ieşire	0x17	6	Citeşte memoria şi regiştrii
Ieşire	0x18	21	Datele pentru boxă
Ieşire	0x19	1	Boxă dezactivată
Ieşire	0x1a	1	Camera IR activată 2
Intrare	0x20	5	Informații despre stare
Intrare	0x21	21	Citeşte datele din memorie şi regiştri
Intrare	0x22	4	Confirmă raportul de ieşire, returnează rezultatul funcției
Intrare	0x30-0x3f	2-21	Rapoarte de date

Tabelul 2.1 Rapoartele folosite de către Wiimote

În toate rapoartele de ieşire, bitul 0(0x01) al primului octet controlează caracteristica *Rumble*[12]. Adițional, bitul 2(0x04) este folosit în multe rapoarte de ieşire drept întrerupător on/off pentru caracteristica specifică controlată.

[12] Setarea "Rumble" a dispozitivelor Wiimote permit utilizatorilor să simtă o vibrație în palmă în timpul anumitor jocuri, pentru a da un plus de realism experienței.

De exemplu, comanda 0x04 trimisă către raportul 0x19(Speaker mute) va dezactiva boxa: *(52) 19 04*, iar comanda 0x00 o va reactiva: *(52) 19 00.*

Pentru a obține raportul stării, se trimite orice comandă către raportul de ieşire 0x15: *(52) 15 00*. Acest raport este trimis fie la cerere, fie ca răspuns la adăugarea unei extensii.

Forma raportului stării este (a1) 20 BB BB LF 00 00 VV, unde:

- BBBB reprezintă butoanele nucleului
- L reprezintă starea LED-urilor
- F este o mască de biți ce indică dacă bateria este slabă, dacă o extensie este sau nu conectată, etc.
- VV indică nivelul curent al bateriei(maxim 0xc8).

Bit	Mască	Semnificație
0	0x01	Baterie aproape terminată
1	0x02	Extensie conectată
2	0x04	Boxă activată
3	0x08	Cameră IR activată
4	0x10	LED 1
5	0x20	LED 2
6	0x40	LED 3
7	0x80	LED 4

Tabelul 2.2 Raportul de stare al controlerului Wiimote

Având obiectul FileStream(creat la pasul 5 din etapa de realizare a conexiunii), putem începe comunicarea cu controlerul Wiimote. Datorită faptului că rapoartele vor fi primite şi trimise aproape constant, este esențial să se utilizeze operații asincrone de

intrare/ieşire. Procesul va începe cu o citire asincronă şi va furniza o metodă *callback*[13]ce se va executa la umplerea bufferului.

```
private const int REPORT_LENGTH = 22;
private byte[] mBuff = new byte[REPORT_LENGTH]; // report buffer

private void BeginAsyncRead()
{
   if(mStream.CanRead) // if the stream is valid and ready
   {
      byte[] buff = new byte[REPORT_LENGTH];
      mStream.BeginRead(buff, 0, REPORT_LENGTH, new AsyncCallback(OnReadData), buff);
   }
}
```

La apelarea acestei metode se procesează datele şi se reia procesul.

```
private void OnReadData(IAsyncResult ar)
{
   byte[] buff = (byte[])ar.AsyncState;
   mStream.EndRead(ar);
   BeginAsyncRead();
    // handle data....
}
```

Datele pot fi obținute de către interfața de programare a aplicațiilor(API) în două moduri: evenimente şi chestionare. În modul eveniment, utilizatorul trebuie să subscrie evenimentul WiimoteChanged ca în următorul *exemplu*[(9)(10)]:

```
using WiimoteLib;

private void Form1_Load(object sender, EventArgs e)
{
   Wiimote wm = new Wiimote();                    // create a new instance of the Wiimote
   wm.WiimoteChanged += wm_WiimoteChanged;  // setup the event to handle state changes

   // setup the event to handle insertion/removal of extensions
   wm.WiimoteExtensionChanged += wm_WiimoteExtensionChanged;

   wm.Connect();    // connect to the Wiimote

   // set the report type to return the IR sensor and accelerometer
   wm.SetReportType(Wiimote.InputReport.IRAccel, true);
}

void wm_WiimoteExtensionChanged(object sender, WiimoteExtensionChangedEventArgs args)
{
   if(args.Inserted)
      wm.SetReportType(Wiimote.InputReport.IRExtensionAccel, true);    // return extension data
   else
      wm.SetReportType(Wiimote.InputReport.IRAccel, true);                // back to original mode
}
```

[13] O metodă callback este o metodă a cărei adresă este transmisă de codul client unei metode server. În cazul procesării asincrone, codul client apelează metoda server; uzual aceasta înseamnă blocarea codului client până la terminarea execuției metodei server. Din acest motiv, metoda server creează un nou fir de execuție şi apoi returnează imediat, dând posibilitatea codului client să-şi continue execuția. Firul de execuție nou pornit execută procesarea solicitată de client, apelând atunci când este necesar metoda callback.

```
void wm_WiimoteChanged(object sender, WiimoteChangedEventArgs args)
{
   WiimoteState ws = args.WiimoteState;      // current state information
   Debug.WriteLine(ws.ButtonState.A);        // write out the state of the A button
}
```

În acest fel, când datele sunt trimise de la Wiimote către pc, un eveniment va fi trimis către handlerul de evenimente pentru procesare. Cealaltă metodă implică verificarea stării controlerului în orice moment, folosind proprietatea WiimoteState a clasei Wiimote.

Întreaga librărie WiimoteLib este centrată pe obiectul WiimoteState. Acesta oferă informații generale despre starea controlerului, precum şi a extensiilor acestuia.

Name	Description
AccelCalibrationInfo	Informații privind calibrarea curentă
AccelState	Starea curentă a acelerometrului
BalanceBoardState	Starea curentă a plăcii Wii Fit Balance
Battery	Nivelul curent al bateriei
ButtonState	Starea curentă a butoanelor
Extension	Este vreo extensie conectată?
ExtensionType	Tipul extensiei inserate
IRState	Starea curentă a senzorilor IR
LEDState	Starea curentă a LED-urilor
Rumble	Starea curentă a caracteristicii Rumble

Tabelul 2.3 Membrii clasei WiimoteState

În cadrul proiectului, este necesară detectarea şi urmărirea poziției capului. Aşadar, din toate informațiile primite de la Wiimote, vom alege doar pe cele referitoare la camera IR, mai exact structura IRState, care oferă detalii despre maxim 4 leduri IR detectabile. Această structură conține deci o listă IRSensors cu 4 obiecte IRSensor care

vor indica, prin intermediul membrilor RawPosition.X şi RawPosition.Y, poziția curentă a unui senzor IR.

Pentru detectarea poziției capului se folosesc două leduri care emit lumină IR, leduri care vor fi plasate lateral pe o pereche de ochelari. Controlerul Wiimote va fi conectat la calculator şi ulterior plasat oblic sub monitor, în dorința de a centra corect poziția capului.

Figura 2.4 Suport pentru Wiimote

Figura 2.5 Ochelari cu leduri IR

Detecția mişcărilor se va face urmărind pozițiile celor două leduri menționate anterior. Pentru stabilirea poziției în spațiu, se va considera coordonata Z ca fiind distanța dintre senzorii IR, distanță ce va fi invers proporțională cu distanța dintre utilizator şi monitor.

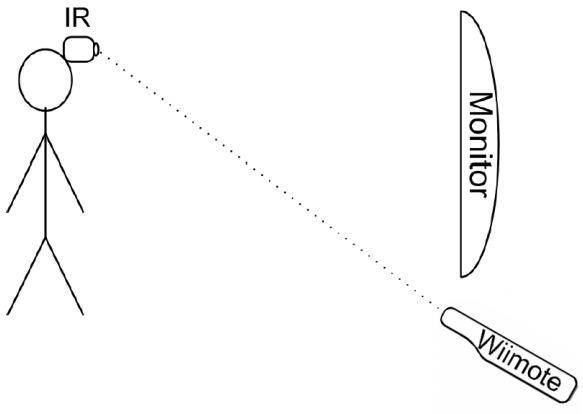


Figura 2.6. Achiziția mişcărilor capului folosind Wiimote şi leduri IR

3. Algoritm de învățare a comenzilor dependent de utilizator

Până în acest moment am stabilit modul în care se conectează controlerul Wiimote la pc, modul de realizare a comunicației cu controlerul şi modul în care putem folosi controlerul pentru a umări poziția capului. În continuare ne vom îndrepta atenția către modul de prelucrare a datelor primite de la Wiimote.

Dorim să studiem modul în care se implică utilizatorii în jocuri în timp ce controlează acțiunea, în ideea de a demonstra că gesturile lor involuntare pot fi folosite drept înlocuitor pentru comanda efectivă a jocului. Vom urmări deci toate mişcările efectuate de utilizator pe durata controlului(cu tastatură/mouse), urmând să obținem în final un *dicționar de comenzi ale capului*[14].

Pe fiecare durată a controlului, se salvează o listă cu toate pozițiile consecutive ale celor doi senzori IR, o nouă poziție fiind detectată de către controler la fiecare 10 ms. Pe baza acestei liste va trebui să obținem o altă listă, a mişcărilor principale detectate.

3.1 Detecția mişcării

Pentru început vom explica modul de detecție a mişcărilor. Suntem interesați doar de mişcările capului sus/jos, stânga/dreapta, luând astfel în considerare doar deplasamentele punctelor pe axele OX şi OY.

[14] Listă ce conține asocierile dintre mişcarea capului şi tasta apăsată/mişcarea mouse-ului..

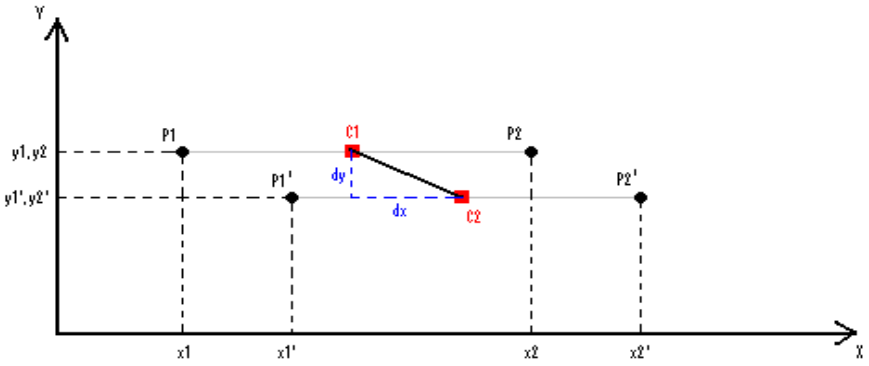


Figura 3.1 Detecția mișcărilor

Pentru recunoaşterea mişcării efectuate vom urmări poziția centrală dintre cei doi senzori. Având date două poziții oarecare (P_1, P_2), (P_1', P_2') de coordonate $P_1(x_1, y_1)$, $P_2(x_2, y_2)$, $P_1'(x_1', y_1')$, $P_2'(x_2', y_2')$ şi poziții centrale $C_1\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$, $C_2\left(\frac{x_1'+x_2'}{2}, \frac{y_1'+y_2'}{2}\right)$, va trebui să determinăm maximul dintre distanțele:

$$dx = \frac{\left|\frac{x_1+x_2}{2} - \frac{x_1'+x_2'}{2}\right|}{1024} \quad \text{şi} \quad dy = \frac{\left|\frac{y_1+y_2}{2} - \frac{y_1'+y_2'}{2}\right|}{768}.$$

Distanța predominantă va fi D = max{dx, dy}.

```
private string CheckMost(Sensors Last, Sensors Previous)
{
  double dx = 0, dz = 0;

  if ( Last.Left(Previous)|| Last.Right(Previous) )
     dx = xDistance(Last, Previous) / 1024 * 100;

  if (Last.Up(Previous) || Last.Down(Previous))
     dy = yDistance(Last, Previous) / 768 * 100;

  if (dx > dy)
      return "OX";
  else
      return "OY";
}
```

Am determinat astfel pe care axă s-a înregistrat o deplasare mai mare, deplasare calculată în raport cu rezoluția camerei IR a controlerului(1024x768), în ideea de a compara distanța parcursă în raport cu distanța maximă posibilă.

Pentru stabilirea direcției stânga/dreapta sau sus/jos(funcție de axa pe care s-a înregistrat mişcarea predominantă) se va verifica pur şi simplu care este poziționarea punctului de mijloc C_2 în raport cu C_1.

```
public string CheckMovement(Sensors Last, Sensors Previous)
{
   string directie = CheckMost(Last, Previous);
   switch (directie)
   {
      case "OX":
         if (Last.Left(Previous))
            return "Left";
         else
            return "Right";

      case "OY":
         if (Last.Up(Previous))
            return "Up";
         else
            return "Down";
   }
   return "";
}
#endregion
```

3.2 Segmentarea traiectoriilor de mişcare

Algoritmul de segmentare presupune prelucrarea listei de poziții, în vederea evidențierii traiectoriei parcurse.

Am folosit *algoritmul lui Douglas–Peucker*[12], care reduce numărul de puncte dintr-o curbă. Având dată o "curbă" compusă din segmente de linie trasate prin n puncte, trebuie găsită o curbă asemănătoare trasată printr-un număr de m puncte, cu m<n. Curba simplificată va fi reprezentată de un subset al punctelor din curba inițială.

Algoritmul împarte o linie în mod recursiv. Punctele de început şi sfârşit sunt automat marcate pentru păstrare. Se caută în continuare punctul cel mai îndepărtat față de linia ce uneşte primul şi ultimul punct. Dacă distanța dintre acest punct şi linia respectivă este mai mică decat o valoare prestabilită ε, atunci toate punctele încă nemarcate pentru păstrare pot fi înlăturate. Dacă distanța este însă mai mare decat ε, atunci punctul trebuie păstrat, şi procesul se reia în 2 etape: prima dată având ca parametri primul punct şi noul punct descoperit, iar apoi noul punct descoperit şi ultimul punct.

La finalizarea recursivității s-a obținut o nouă curbă, trasată doar prin acele puncte care au fost marcate pentru păstrare.

Algoritmul în pseudocod:

```
procedura DouglasPeucker( vector PointList[], întreg firstPoint, întreg lastPoint, real toleranta, ref lista Keep)
   dmax <- 0
   index <- 0
   pentru  i <- firstPoint, lastPoint execută
```

```
        *) calculăm distanța d = dist(PointList[i], Line(PointList[firstPoint], PointList[lastPoint]))
        dacă  d > dmax
              index = i
              dmax = d
         sfărşit dacă
   sfărşit pentru

  dacă dmax > toleranta şi index <> 0
        Keep <- Keep + {index}
        DouglasPeucker(PointLis[], firstPoint, index, toleranta, Keep)
        DouglasPeucker(PointLis[], index, lastPoint, toleranta, Keep)
   sfărşit dacă
sfărşit procedură
```

Exemplu:

Se consideră următoarele date de intrare:

n = 13

X = { $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}$}

U = { (1, 6), (2, 5), (3, 4), (2, 3), (2, 2), (5, 1), (6, 3), (6, 6), (8, 4), (9, 5), (8, 1), (10, 2), (11, 6)}

Toleranța ε: 2

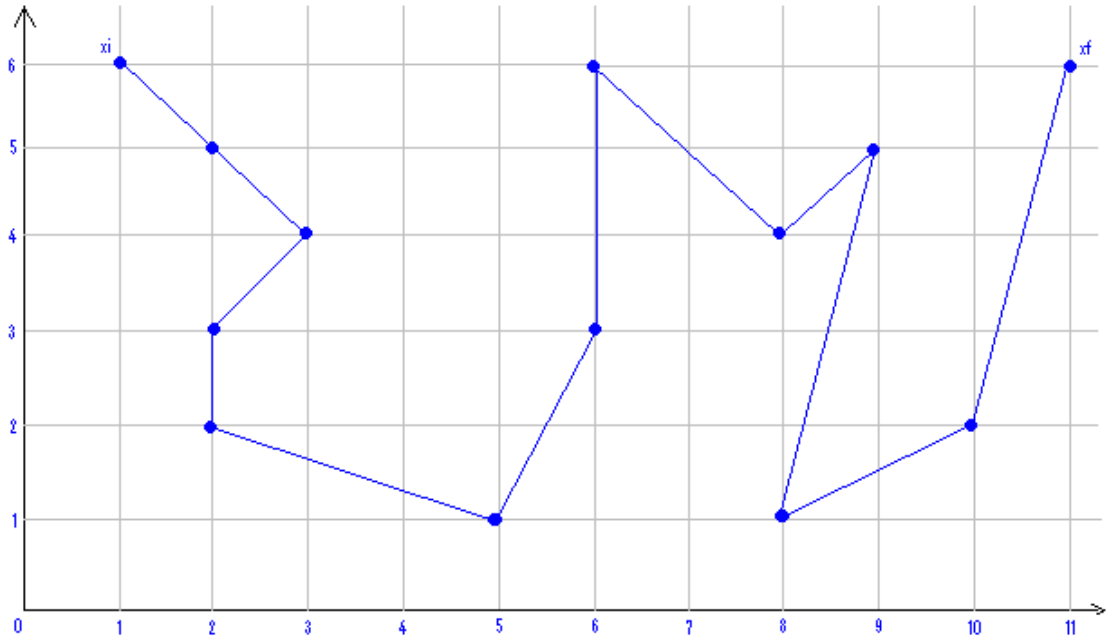


Figura 3.2 Setul inițial de puncte

Paşi:

- $x_i = (1, 6)$, $x_f = (11, 6)$
  Se păstrează în mod automat cele două extreme ale curbei: x_i(punctul initial) şi x_f(punctul final) de coordonate (1. 6), (11, 6) şi indici în cadrul listei: 0, 12.
  Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep** = {0, 12}.
- Se calculează toate distanțele de la celelalte puncte, a căror indici sunt cuprinşi în intervalul [i, f], către dreapta ce uneşte punctele x_i şi x_f.

  Lista distanțelor:

X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}
1	2	3	4	5	3	0	2	1	5	4

  Am obținut distanța maximă pentru punctul x_5 de valoare 5(valoare mai mare decât valoarea toleranței, 2).

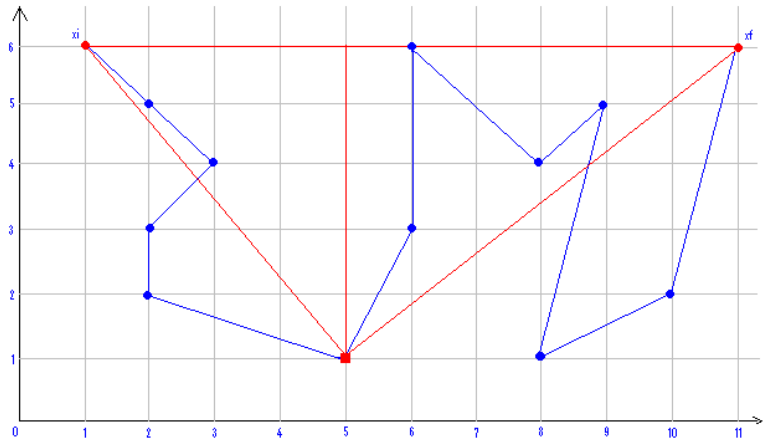


Figura 3.3 Etapele algoritmului

Procesul se va relua pentru cele 2 linii determinate de punctele ($\mathbf{x_0}$, $\mathbf{x_5}$) şi ($\mathbf{x_5}$, $\mathbf{x_{12}}$).

Cazul (x_0, x_5)

- $x_i = (1, 6)$, $x_f = (5, 1)$

  Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep** = {0, 12, 5}.

- Lista distanțelor:

X_1	X_2	X_3	X_4
0.15	0.31	1.09	1.71

Am obținut distanța maximă pentru punctul x_4 de valoare 1.71(valoare mai mică decât valoarea toleranței, 2), ceea ce înseamnă că nu marcăm nici un punct pentru păstrare în acest moment şi ne întoarcem din recursivitate.

Următorul pas al recursivității îl reprezintă

Cazul (x_5, x_{12})

- $x_i = (5, 1)$, $x_f = (11, 6)$

  Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep** = {0, 12, 5}.

- Lista distanțelor:

X_6	X_7	X_8	X_9	X_{10}	X_{11}
0.89	3.20	0.38	0.51	1.92	2.43

Am obținut distanța maximă pentru punctul x_7 de valoare 3.20(valoare mai mare decât valoarea toleranței, 2).

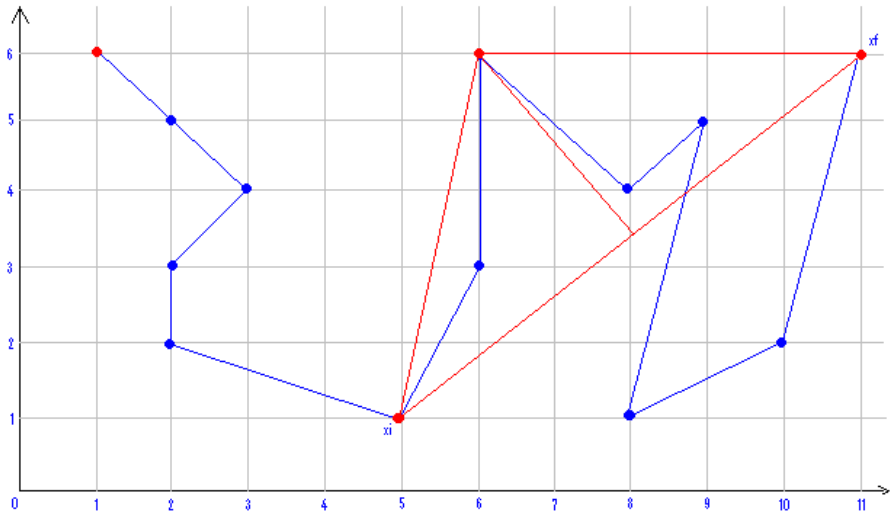


Figura 3.4 Etapele algoritmului

Procesul se va relua pentru cele 2 linii determinate de punctele ($\mathbf{x_5}$, $\mathbf{x_7}$) şi ($\mathbf{x_7}$, $\mathbf{x_{12}}$).

Cazul (x_5, x_7)

- $x_i = (5, 1)$, $x_f = (6, 6)$

  Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep** = {0, 12, 5, 7}.

- Lista distanțelor:

X_6
0.58

Am obținut distanța maximă pentru punctul x_6 de valoare 0.58(valoare mai mică decât valoarea toleranței, 2), ceea ce înseamnă că nu marcăm nici un punct pentru păstrare în acest moment şi continuăm recursivitatea cu cea de-a două linie.

Cazul (x_7, x_{12})

- $x_i = (6, 6)$, $x_f = (11, 6)$

  Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep** = {0, 12, 5, 7}.

- Lista distanțelor:

X_8	X_9	X_{10}	X_{11}
2	1	5	4

Am obținut distanța maximă pentru punctul x_{10} de valoare 5(valoare mai mare decât valoarea toleranței, 2)

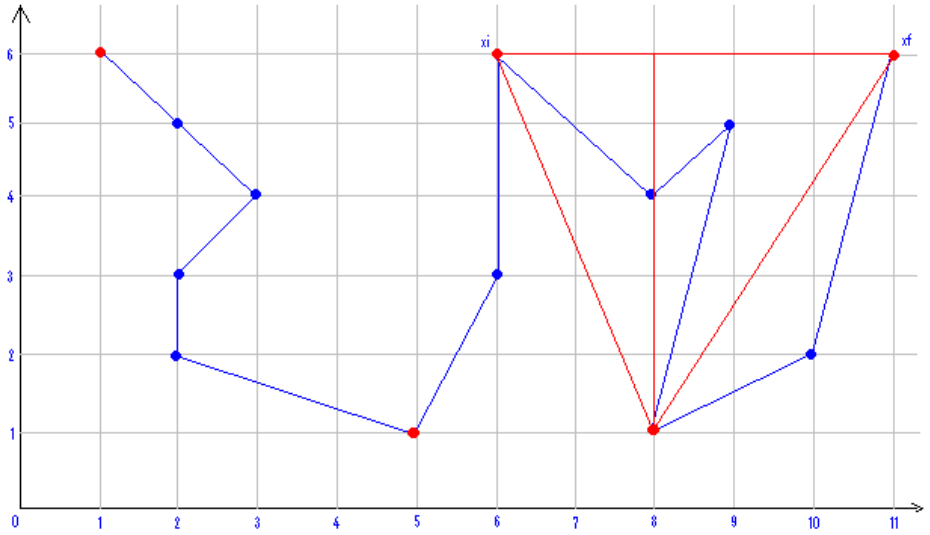


Figura 3.5 Etapele algoritmului

Procesul se va relua pentru cele 2 linii determinate de punctele ($\mathbf{x_7}$, $\mathbf{x_{10}}$) şi ($\mathbf{x_{10}}$, $\mathbf{x_{12}}$).

Cazul ($\mathbf{x_7}$, $\mathbf{x_{10}}$)

- $x_i = (6, 6)$, $x_f = (8, 1)$

  Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep**={0, 12, 5, 7, 10}.

- Lista distanțelor:

X_8	X_9
1.11	2.41

Am obținut distanța maximă pentru punctul x_9 de valoare 2.41(valoare mai mare decât valoarea toleranței, 2).

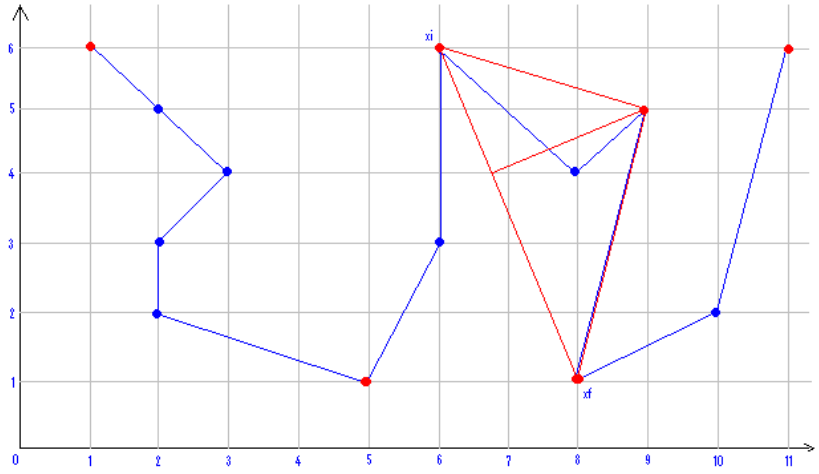


Figura 3.6 Etapele algoritmului

Procesul se va relua pentru cele 2 linii determinate de punctele ($\mathbf{x_7}$, $\mathbf{x_9}$) şi ($\mathbf{x_9}$, $\mathbf{x_{10}}$).

Cazul (x_7, x_9)

- $x_i = (6, 6)$, $x_f = (8, 5)$

Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep** = {0, 12, 5, 7, 10, 9}.

- Lista distanțelor:

X_8
1.26

Am obținut distanța maximă pentru punctul x_8 de valoare 1.26(valoare mai mică decât valoarea toleranței, 2), ceea ce înseamnă că nu marcăm nici un punct pentru păstrare în acest moment şi continuăm recursivitatea cu cea de-a două linie.

Cazul (x_9, x_{10})

- $x_i = (6, 6)$, $x_f = (8, 5)$

Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep** = {0, 12, 5, 7, 10, 9}.

- Nu există nici un punct de indice cuprins în intervalul [i, f], ceea ce înseamă că ne întoarcem din recursivitate.

Următorul pas al recursivității îl reprezintă
Cazul (x_{10}, x_{12}).

- x_i = (8, 1), x_f = (11, 6)

Lista cu punctele marcate pentru păstrare: **pointIndexsToKeep** = {0, 12, 5, 7, 10, 9}.

- Lista distanțelor:

X_{11}
1.20

Am obținut distanța maximă pentru punctul x_{11} de valoare 1.20(valoare mai mică decât valoarea toleranței, 2), ceea ce înseamnă că nu marcăm nici un punct pentru păstrare în acest moment.

S-au epuizat toți pașii recursivității, iar lista finală a punctelor păstrate este

pointIndexsToKeep = {0, 5, 7, 9, 10, 12}

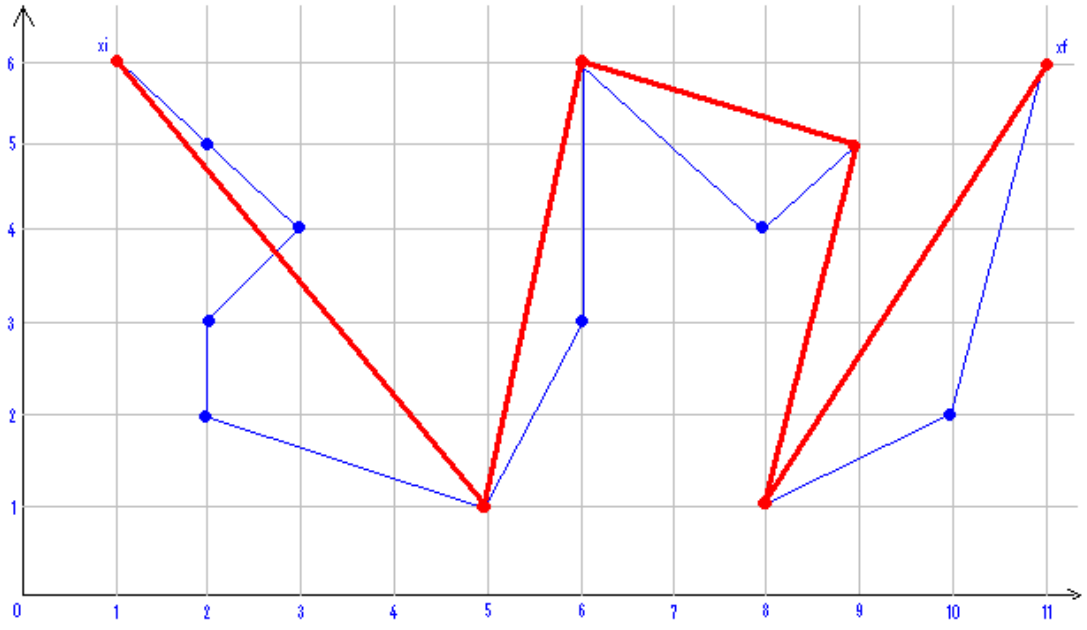


Figura 3.7 Etapele algoritmului

În funcție de valoarea dată toleranței, se vor obține curbe diferite, astfel încât:

- pentru ε=0.4, singurul punct eliminat în urma execuției algoritmului va fi x_1, însă curba finală va fi identică cu cea inițială

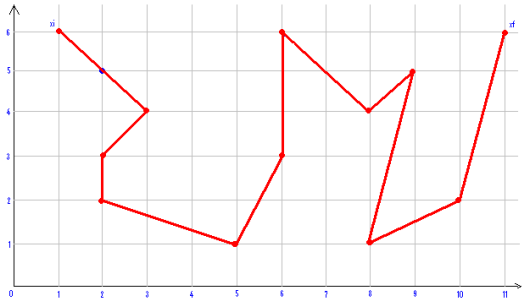


Figura 3.8 Curba obținută pentru ε = 0.4

- pentru ε = 0.5, în urma execuției algoritmului se vor elimina punctele $\{x_1, x_3\}$, iar curba finală va fi de forma:

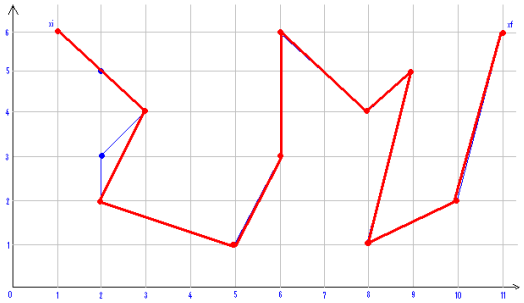


Figura 3.9 Curba obținută pentru ε = 0.5

- pentru ε = 1, în urma execuției algoritmului se vor elimina punctele $\{x_1, x_3, x_6\}$, iar curba finală va fi de forma:

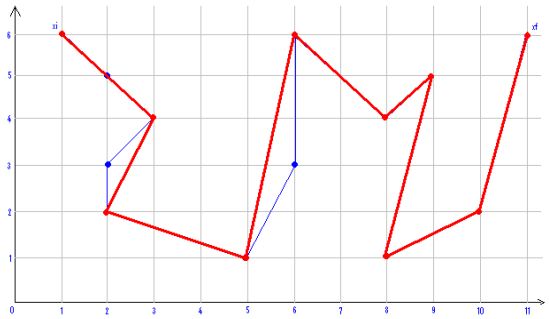
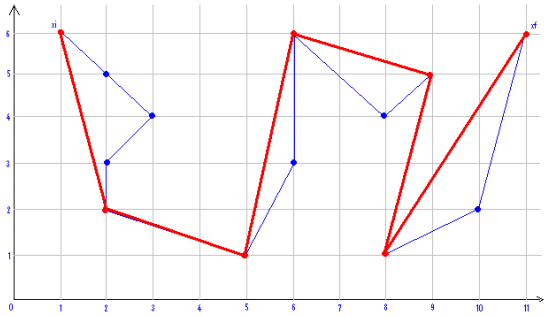


Figura 3.10 Curba obținută pentru ε = 1



- pentru ε = 1.5, în urma execuției algoritmului se vor elimina punctele $\{x_1, x_2, x_3, x_6, x_8, x_{11}\}$, iar curba finală va fi de forma:

Figura 3.11 Curba obținută pentru $\varepsilon = 1.5$

- pentru $\varepsilon = 2$, în urma execuției algoritmului se vor elimina punctele $\{x_1, x_2, x_3, x_4, x_6, x_8, x_{11}\}$, iar curba finală va fi de forma:

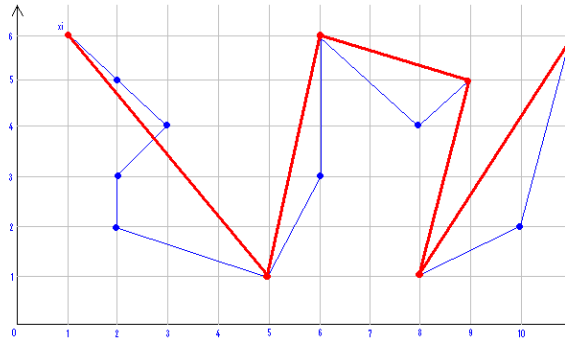


Figura 3.12 Curba obținută pentru $\varepsilon = 2$

- pentru $\varepsilon = 4$, în urma execuției algoritmului se vor elimina punctele $\{x_1, x_2, x_3, x_4, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}$, iar curba finală va fi de forma:

- pentru $\varepsilon = 3$, în urma execuției algoritmului se vor elimina punctele $\{x_1, x_2, x_3, x_4, x_6, x_8, x_9, x_{11}\}$, iar curba finală va fi de forma:

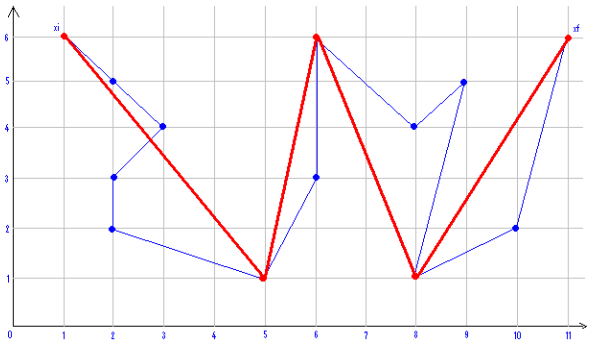


Figura 3.13 Curba obținută pentru $\varepsilon = 3$

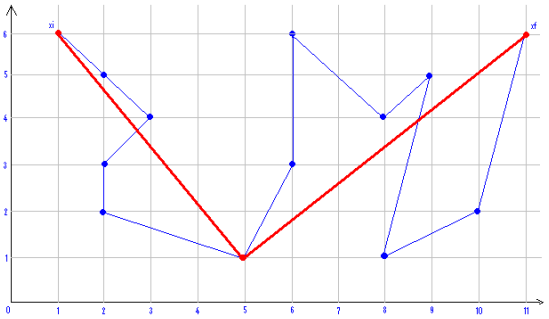


Figura 3.14 Curba obținută pentru $\varepsilon = 4$

- pentru $\varepsilon = 5$, în urma execuției algoritmului se vor elimina punctele $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}$, iar curba finală va fi de forma:

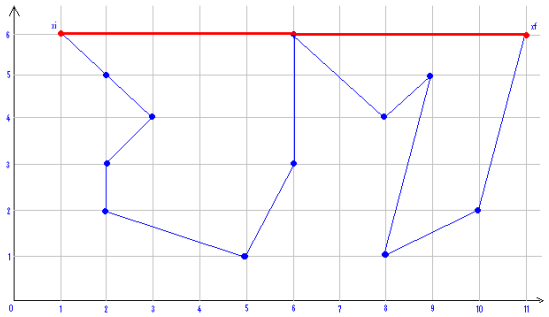


Figura 3.15 Curba obținută pentru $\varepsilon = 5$.

Implementarea algoritmului:

Calculul distanței dintre punctul Point(x, y) şi dreapta determinată de punctele Point1(x1, y1) şi Point2(x2, y2) se face conform formulelor:

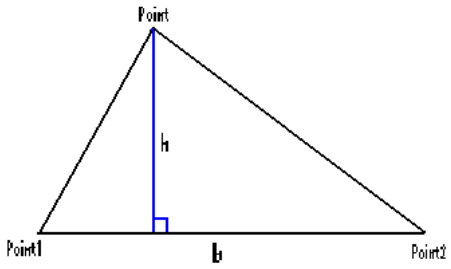


Figura 3.16 Triunghiul format din punctele Point, Point1 şi Point2

- $Aria = \frac{1}{2}\begin{vmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x & y & 1 \end{vmatrix}$, reprezintă aria triunghiului format din cele 3 puncte
- $Aria = \frac{b * h}{2}$, unde b = baza şi h = înălțimea
- $h = \frac{2 * Aria}{b}$

```
private static double PerpendicularDistance(Sensor Point1, Sensor Point2, Sensor Point)
{
   double area = Math.Abs(.5 * (Point1.X * Point2.Y + Point2.X * Point.Y + Point.X * Point1.Y - Point2.X * Point1.Y
                                - Point.X * Point2.Y - Point1.X * Point.Y));
   double bottom = Math.Sqrt(Math.Pow(Point1.X - Point2.X, 2) + Math.Pow(Point1.Y - Point2.Y, 2));
   double height = area / bottom * 2;
   return height;
}
```

Funcția recursivă va avea ca parametri indicii punctelor de început şi sfârşit(firstPoint, lastPoint), lista de puncte(points), valoarea toleranței(tolerance) şi o referință către lista cu indicii punctelor marcate pentru păstrare(pointIndexsToKeep).

Ne interesează punctele de pe curbă situate între punctele firstPoint şi lastPoint. Dintre toate aceste puncte, îl vom alege pe acela care este cel mai îndepărtat față de dreapta de interes. În acest scop vom calcula toate distanțele dintre aceste puncte către dreapta determinată de punctele firstPoint şi LastPoint.

```
private static void DouglasPeuckerReduction(List<Sensor> points, int firstPoint, int lastPoint, double tolerance, ref
List<int> pointIndexsToKeep)
{
   double maxDistance = 0;
   Int32 indexFarthest = 0;

   for (Int32 index = firstPoint; index < lastPoint; index++)
   {
      double distance = PerpendicularDistance(points[firstPoint], points[lastPoint], points[index]);
      if (distance > maxDistance)
      {
         maxDistance = distance;
         indexFarthest = index;
      }
   }
  …
```

Dacă distanța maximă determinată depăşeşte valoarea toleranței, procesul se va relua, utilizând metoda DivideEtImpera, împărțind dreapta inițială de interes în alte două drepte după punctul determinat anterior(care va fi marcat pentru păstrare).

```
    ...
    if (maxDistance > tolerance && indexFarthest != 0)
    {
      pointIndexsToKeep.Add(indexFarthest);
      DouglasPeuckerReduction(points, firstPoint, indexFarthest, tolerance, ref pointIndexsToKeep);
      DouglasPeuckerReduction(points, indexFarthest, lastPoint, tolerance, ref pointIndexsToKeep);
    }
}
```

Apelul funcției se face după stabilirea indicilor de început şi sfârşit, precum şi a valorii pentru toleranță(cu cât valoarea toleranței este mai mare, cu atât numărul punctelor păstrate va fi mai mic). Lista de returnare va fi inițializată cu indicii punctelor extreme(anume 0 şi n-1, unde n reprezintă numărul de puncte ale curbei).

```
private static SetSensorsL DouglasPeuckerReduction(List<Sensors> LPoints, List<Sensor> Points, double Tolerance)
{
   Int32 firstPoint = 0;
   Int32 lastPoint = Points.Count - 1;
   List<Int32> pointIndexsToKeep = new List<Int32>();

   pointIndexsToKeep.Add(firstPoint);
   pointIndexsToKeep.Add(lastPoint);

   DouglasPeuckerReduction(Points, firstPoint, lastPoint, Tolerance, ref pointIndexsToKeep);
   pointIndexsToKeep.Sort();
```

```
    SetSensorsL returnPoints = new SetSensorsL();
    foreach (Int32 index in pointIndexsToKeep)
    {
      returnPoints.listS.Add(LPoints[index]);
      returnPoints.listD.Add(Points[index]);
    }
  return returnPoints;
}
```

După apelul funcției, lista returnată va fi sortată şi pe baza ei se vor stabili punctele păstrate, astfel încât curba finală va fi determinată doar de punctele a căror indici sunt incluşi în lista obținută în urma execuției algoritmului.

Complexitatea algoritmului:

Inițial vom determina numărul maxim de diviziuni succesive pe care algoritmul le realizează. Problema inițială de dimensiune n este redusă la două subprobleme de dimensiuni (k, n-k), k<n, care, la rândul lor, sunt reduse la patru de dimensiuni (k1, k–k1), k1<k, (k2, n-k-k2), k2<n-k, etc.

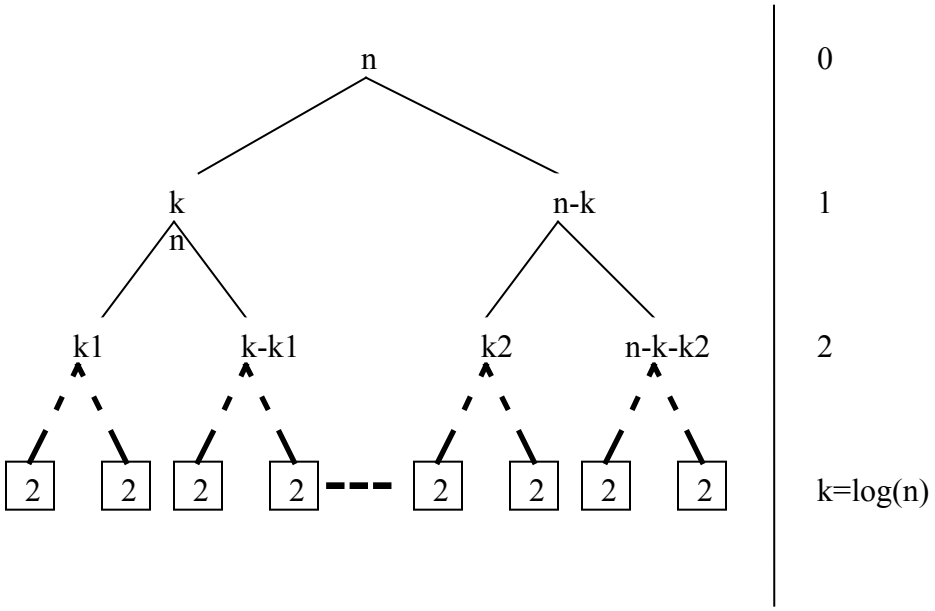


Figura 3.17 Analiza complexității algoritmului

Timpul de execuție al algoritmului depinde de valoarea toleranței, astfel:

- Numărul minim de paşi este obținut atunci când valoarea toleranței depăşeşte toate distanțele dintre orice punct şi dreapta determinată de extremitățile curbei inițiale (curba finală va fi determinată doar de extremitățile curbei inițiale).

- Numărul maxim de paşi este obţinut atunci când valoarea toleranţei este mai mică decât toate distanţele posibile dintre un punct şi o dreaptă din cadrul curbei iniţiale(curba finală va fi identică cu cea iniţială).

Putem scrie următoarea relaţie de recurenţă pentru timpul de execuţie necesar algoritmului Douglas-Peucker pentru a rezolva o problemă de dimensiune n astfel:

$$Tn = \begin{cases} \Theta(1) & n = 1 \text{ sau toleranţă maximă} \\ n - 1 + T(1) + T(n - 1) & n > 1 \end{cases}$$

Având în vedere faptul că partea cea mai consumatoare de timp din cadrul algorimului o reprezintă calculul distanţelor dintre n puncte şi m segmente de linie(cu m fiind mărginit superior de n), complexitatea va fi de $\Theta(n^2)$.

Există şi o alta variantă a algoritmului mai rapidă, de *complexitate* $\Theta(n \log_2 n)$ [13], la care împărţirea liniei se face după puncte alese aleator.

3.3 Analiza şi interpretarea datelor

Având lista de poziţii filtrată prin algoritmul prezentat anterior, urmează să detectăm pentru fiecare 2 puncte consecutive ale curbei mişcarea efectuată. Vom obţine astfel lista mişcărilor pe durata unui control.

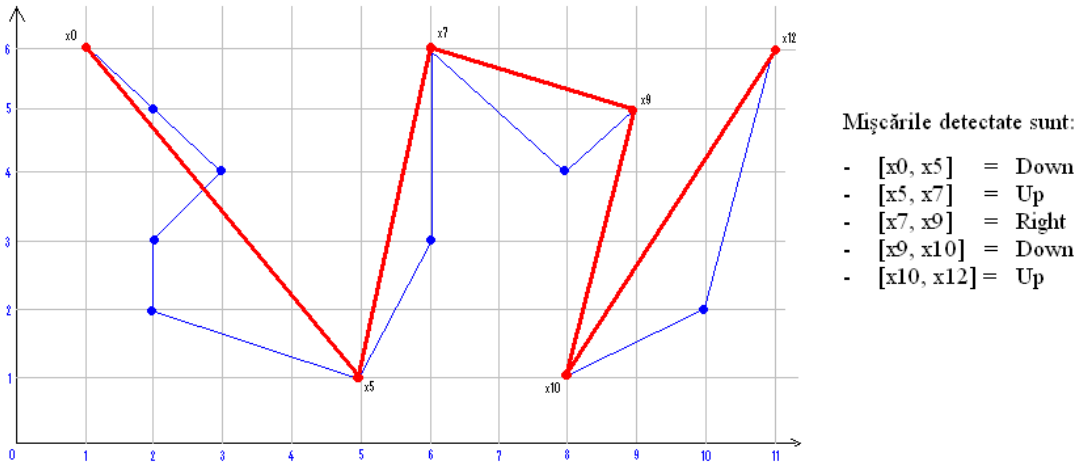


Figura 3.18 Recunoaşterea mişcărilor după filtrare

Pentru fiecare tip de control vom obţine în final o mulţime de astfel de liste cu informaţii privind direcţia mişcării şi amplitudinea mişcării, date necesare creării dicţionarului de comenzi.

În exemplu am obținut lista: *{(Down, 6), (Up, 5), (Right, 3), (Down, 4), (Up, 5)}*.

Informațiile obținute la finalul unui test vor fi de genul:

- o listă de *k* aplicații(în cazul de față k=4);
- pentru fiecare aplicație avem o listă de *p* tipuri de contol(p=4 dacă se vor utiliza doar cele 4 taste: KeyUp, KeyDown, KeyLeft, KeyRight sau mișcarea mouse-ului în sus, jos, stânga, dreapta);
- pentru fiecare tip de control vom avea o lista de maxim 4 mișcări ale capului: HeadMovement Up, Down, Left, Right;
- pentru fiecare mișcare a capului vom avea o listă de n valori reprezentând amplitudinea mișcării şi frecvența de apariție a amplitudinii respectivă.

Vom determina pentru fiecare tip de control mișcarea predominantă a capului(cu frecvența maximă de apariție în cadrul listei), împreună cu o valoare prag stabilită prin calculul mediei aritmetice a amplitudinilor.

```
private int Max()
{
  int max = 0, poz = -1;

  for (int i = 0; i < H; i++)
     if (listS[i].TotalFrequency > max)
     {
          max = listS[i].TotalFrequency;
          poz = i;
     }
    return poz;
}
```

```
public string GetMovement()
{
   if (H > 0)
   {
      int p = Max();
      return listS[p].sMovement;
   }

   return "";
 }
```

```
public int GetThreshold()
{
    if (H > 0)
    {
        int p = Max();
        return listS[p].Threshold;
    }

    return 0;
}
```

Pe baza funcții Max(ce returnează poziția din cadrul listei la care se află obiectul ce conține informații despre mișcarea predominantă), vom determina tipul(GetMovement) şi pragul(GetThreshold) mișcării asociate controlerului respectiv.

Lista finală va cuprinde informații gen: tip control – mișcarea capului – frecvența de apariție – valoare prag. Există la acest pas posibilitatea ca mai multe tipuri de control să aibe asociată aceeaşi mișcare a capului. Vom face în continuare o selecție în cadrul acestei liste după mișcările capului: îi vom asocia acel tip de control pentru care frecvența de apariție este maximă.

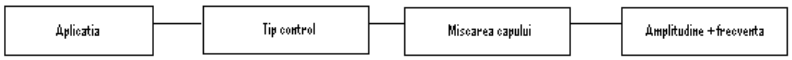
```
public int GetFrequency(string movement)
{
   if (H > 0)
   {
      int i = 0;
      for (i = 0; i < H; i++)
         if ( listS[i].sMovement == movement)
            break;

      if ( i < H )
         return listS[i].TotalFrequency;
   }
```

```
    return 0;
}


private static int GetMaxKey(string movement)
{
    int n = Set.Count;
    int max = 0, poz = -1;
    for (int i = 0; i < n; i++)
        if (Set[i].GetMovement() == movement)
            if (Set[i].GetFrequency(movement) > max)
            {
                max = Set[i].GetFrequency(movement);
                poz = i;
            }
    return poz;
}
```

Crearea dicționarului de comenzi:

```
public static void CreateDataDictionary()
{
    SetCommand sc = new SetCommand();

    for (int i = 0; i < 4; i++)
    {
        int poz = GetMaxKey(moves[i]);
        if (poz != -1)
        {
            string move = moves[i];
            int key = Set[poz].KeyPressed;
            Command c = new Command(key, move);
            sc.Add(c);
        }
    }
}
```

Exemplu:



0 = Down
1 = Left
2 = Right
3 = Up

Game	Key	HeadMovement	Amplitude	Frequency
Need for Speed	Left	0	3	2
		0	4	3
		0	7	1
		0	16	1
		1	3	1
		1	4	1
		1	5	1
		1	8	4
		1	10	1
		1	13	1
		1	23	1
		2	3	1
		2	4	2
		2	16	1
		2	32	1
		3	3	2
		3	4	3
		3	5	1
		3	6	1
		3	7	1
	Right	0	6	1
		1	4	1
		1	7	1
		1	16	1
		1	17	1
		1	18	1
		2	3	1
		2	4	1
		2	6	2
		2	7	2
		2	8	1
		2	10	1
		2	12	2
		3	3	1
		3	4	1
		3	5	1

Asocierea determinată:
KeyLeft – HeadMovement Left, prag 9

Asocierea determinată:
KeyRight – HeadMovement Right, prag 7.5

Figura 3.19 Exemplu de date obținute în urma etapei de învățare

Dicționarul de comenzi în acest caz va fi:

- KeyPressed Left – HeadMovement Left, cu pragul 9
- KeyPressed Right – HeadMovement Right, cu pragul 7.5

Tipul mişcării asociate unui control, cât şi pragul determinat acesteia, depind de mişcările utilizatorului în timpul controlului: astfel, pentru utilizatori mai impulsivi, mişcarea va fi asociată corect, având un prag rezonabil, pe când în cazul utilizatorilor pasivi, există posibilitatea de a asocia o mişcare greşită controlului, sau cu un prag de valoare mică.

4. Recunoaşterea mişcărilor gestuale direcționale

Pentru recunoaşerea mişcărilor gestuale direcţionale vom ţine cont de dicţionarul de comenzi creat anterior, împreună cu pragurile determinate pentru fiecare direcţie. Aceste date au fost salvate în fişiere separate, specifice fiecărui tip de joc în parte.

După rularea aplicaţiei şi selectarea jocului dorit, se va stabili poziţia de start. Următoarele mişcări vor fi determinate strict în funcţie de această poziţie. Se va verifica la fiecare moment dacă mişcarea curentă(deplasarea faţă de poziţia iniţială), depăşeşte vreun prag din toate cele patru stabilite(pragL, pragR, pragD, pragU).

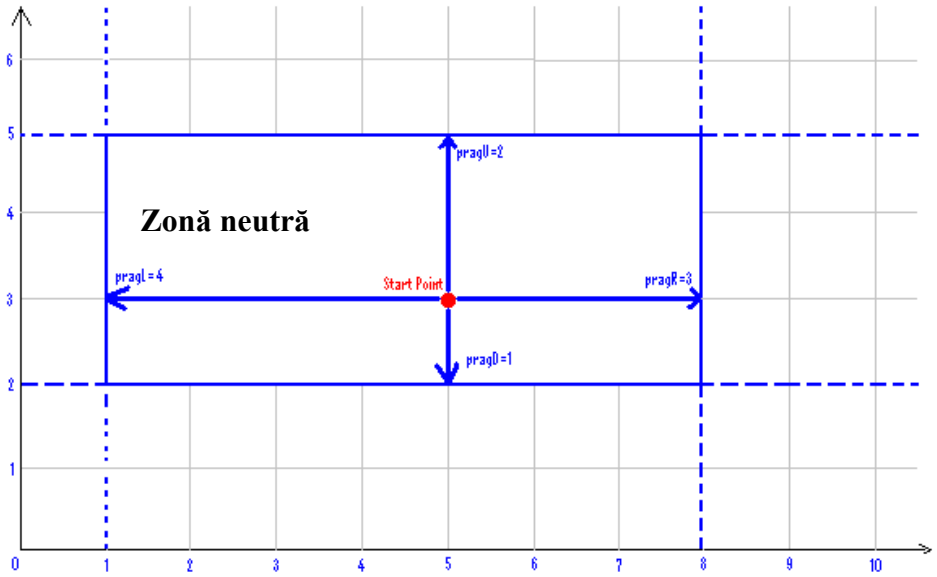


Figura 4.1 Verificarea pragurilor

```
public bool CheckThreshold()
{
    int pragX = 0, pragY = 0;

    if (Last.Left(startPoint))
          pragX = WiimoteControl.pragL;
    else
          pragX = WiimoteControl.pragR;

   if (Last.Up(startPoint))
          pragY = WiimoteControl.pragU;
  else
          pragY = WiimoteControl.pragD;

  if (xDistance(startPoint) > pragX || yDistance(startPoint) > pragY)
          return false;
   return true;
}
```

Funcţia *CheckThreshold()* determină poziţionarea punctului curent în raport cu punctul de start. Pentru poziţiile astfel determinate(pe OX şi OY) se verifică în continuare depăşirea pragurilor respective.

Dacă noul punct este situat în interiorul *„zonei neutre”*[15](zona de zgomot), înseamnă ca utilizatorul nu a efectuat nici o mişcare voluntară, acesta aflându-se în starea de aşteptare. Nu sunt transmise comenzi întrucât amplitudinea mişcărilor este prea mică şi este foarte posibil să fi apărut doar din greşeală.

În cazul în care s-a depăşit minim un prag, înseamnă că utilizatorul a efectuat o mişcare voluntară şi doreşte să trimită o comandă către aplicația controlată. Se verifică tipul mişcării detectate şi care este tipul controlului asociat mişcării respective, control care va fi ulterior trimis în mod automat către aplicație, până când utilizatorul va anula comanda prin revenirea către zona neutră.

Este foarte posibil ca în modul comandă utilizatorul să efectueze mişcări mai bruşte şi de amplitudini mai mari, datorită intenției de a evidenția tipul comenzii ce se doreşte a fi trimisă. Acest lucru poate determina pierderea semnalelor IR de către cameră. O soluție la această problemă ar putea fi estimarea noii poziții, fie prin păstrarea ultimei poziții detectate(în cazul în care punctul curent se află în exteriorul zonei neutre), fie prin forțarea noii poziții către exteriorul zonei neutre în sensul ultimei direcții detectate înainte de pierderea semnalului(în cazul în care punctul curent se află în interiorul zonei neutre).

În această idee se va verifica la fiecare moment numărul de semnale IR detectate. Dacă sunt detectate mai puțin de două semnale, acestea vor trebui inițializate manual.

Având poziția capului stabilită de senzorii (s1, s2) şi lista P ce conține toate pozițiile consecutive alea acestor senzori de la începutul conexiunii şi până la momentul curent, inițializarea în cazul pierderii semnalului se va face în felul următor:

- dacă ultima poziție din cadrul listei se află în **exteriorul** zonei neutre

s1 = P.Last.P1, s2 = P.Last.P2

unde Last = ultimul set de senzori (P1, P2) din cadrul listei P

- dacă ultima poziție din cadrul listei se află în **interiorul** zonei neutre

Funcție de mişcarea detectată în raport cu poziția precedentă, P.Cprevious, se va seta:

- Cprevious = “**Left**”
  - s1.RawPosition.X += pragL
  - s2.RawPosition.X += pragL
- Cprevious = “**Right**”
  - s1.RawPosition.X -= pragR
  - s2.RawPosition.X -= pragR
- Cprevious = “**Up**”
  - s1.RawPosition.Y += pragU
  - s2.RawPosition.Y += pragU
- Cprevious = “**Down**”
  - s1.RawPosition.Y -= pragD
  - s2.RawPosition.Y -= pragD

În momentul în care se detectează o poziție în exteriorul zonei neutre, trebuie stabilit şi modul în care se va trimite comanda către aplicație. Acest mod va trebui să fie specific fiecărui tip de aplicație în parte. Prin controlul gestual al jocurilor trebuie să

[15] Zona delimitată de cele 4 praguri, în raport cu poziția de start.

obținem pe cât posibil rezultate similare, dacă nu chiar identice, cu cele obținute prin controlul cu dispozitivele uzuale.

În această idee, trebuie stabilită o legătura între timpul cât este detectată o mişcare şi timpul cât este trimisă comanda către aplicație. Trimiterea comenzii se va face în mod întrerupt, pe durata unui număr de *x ms* din *y ms*(*x*<*y*), în încercarea de a obține sensibilitatea ideală a controlului respectiv(cu valorile x, y setate în funcție de aplicație).

Trimiterea comenzii se face strict pe durata mişcării într-o anumită direcție, aceasta fiind anulată din primul moment în care utilizatorul schițează o mişcare "înapoi"(către poziția de start).

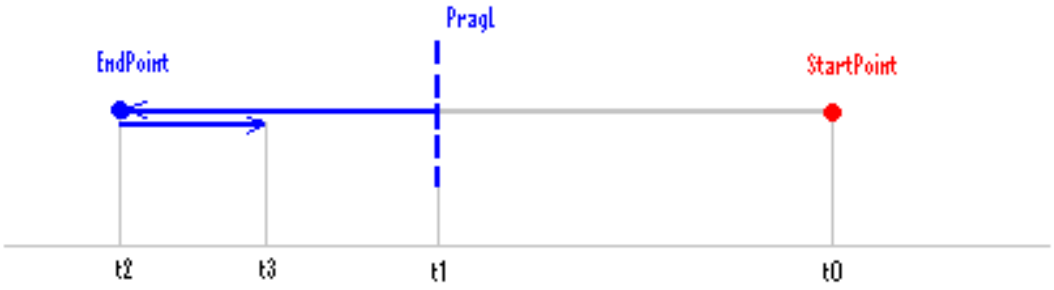


Figura 4.2 Etapele recunoaşterii mişcărilor gestuale

După cum se observă în figura, se pot determina 3 etape principale în cadrul recunoaşterii mişcărilor gestuale direcționale:

- Intervalul $[t_0, t_1)$
  Utilizatorul se află încă în zona neutră
- Intervalul $[t_1, t_2)$
  La momentul t_1, s-a detectat faptul că utilizatorul a efectuat o mişcare spre stânga, mişcare ce a depăşit pragul impus acestei direcții, ceea ce determină trimiterea comenzii către aplicație pe durata întregului interval $[t_1, t_2)$
- Intervalul $[t_2, t_3]$
  Începând cu momentul t_2, s-a detectat faptul că utilizatorul a efectuat o mişcare înapoi, către poziția de start, ceea ce determină anularea comenzii anterior activate.

Această metodă a fost implementată pe baza a două obiecte Cstart şi Cprevious ce indicau poziția curentă în raport cu poziția de start(caz în care se ține cont şi de depăşirea pragurilor), respectiv cu poziția precedentă.

```
public void AddSensor(Sensors s)                    return;
{
  if (s.Equals(Last) || s.Equals(startPoint))       listS.Add(s);
```

```
    if (!CheckThreshold())
    {
        string sMove = CheckMovement(startPoint);
        cStart.Add(sMove);
    }
    else
        cStart.Add("None");

    string pMove = CheckMovement(Previous);
    cPrevious.Add(pMove);
}


    public string CStart
    {
        get
        {
            if (N > 0)
                return cStart[N - 1];
            return "None";
        }
    }

    public string CPrevious
    {
        get
        {
            if (N > 0)
                return cPrevious[N - 1];
            return "None";
        }
    }
```

Etapele modului comandă sunt:

- Verificarea semnalelor IR primite şi estimarea poziției capului, dacă este necesar
- Salvarea noii poziții în lista de poziții, cu determinarea automată a deplasării față de poziția de start, precum şi față de cea anterioară
- Stabilirea modului de trimitere a comenzii în funcție de aplicația ce se doreşte a fi controlată.

5. Aplicații demonstrative

Pentru susținerea lucrării, s-au luat în considerare jocurile: Tetris, Pacman, Need For Speed Underground 2 şi Quake III Arena.

Tetris[13] este un joc video tip puzzle creat de Alexei Pajitnov în iunie 1984. Numele jocului a fost derivat din grecescul "tetra"(toate piesele sunt formate din patru pătrățele), şi tenis(jocul preferat al creatorului).

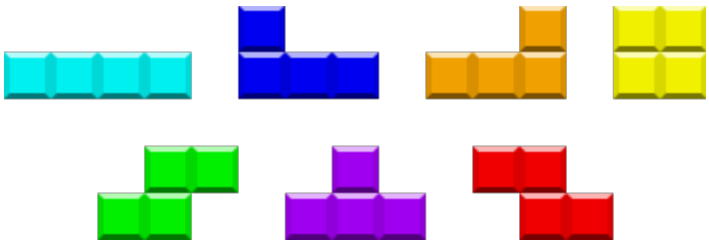


Figura 5.1 Tetrominoes[16]

O secvență aleatoare de astfel de blocuri vor cădea consecutiv în zona de joc. Ideea jocului este de a manipula aceste piese, mişcându-le înspre stânga/dreapta sau rotindu-le cu un unghi de 90º, cu scopul de a crea o linie orizontală de pătrățele. Când o asemenea linie este creată, aceasta va dispărea si toate blocurile care se situau deasupra ei vor cădea. Nivelul jocului va creşte la fiecare 10 linii eliminate, implicând astfel creşterea vitezei de „cădere" a blocurilor. Jocul se termină când piramida de blocuri a atins partea superioară a zonei de joc, sau când jucătorul a completat toate cele 15 nivele.

Jocul *Pac-Man*[14],conceput de firma Namco, a fost lansat în Japonia în Mai 1980.

Figura 5.2 Jocul Pac-Man

Jucătorul va direcționa caracterul denumit Pac-Man printr-un labirint, cu scopul de a mânca bulinele albe. Când toate bulinele dispar, se va trece la nivelul următor. Patru monştri(Blinky, Pinky, Inky şi Clyde) mişună prin labirint, încercând să-l prindă pe

[16] Sursa iamginii:
http://upload.wikimedia.org/wikipedia/commons/3/39/Tetrominoes_IJLO_STZ_Worlds.svg

Pac-Man. Dacă un monstru îl atinge, se pierde o viață. Jocul se termină când se pierd toate cele 3 vieți. În joc mai există şi patru buline mai mari în cele patru colțuri ale labirintului, denumite „pilule de putere”. Acestea vor oferi caracterului abilitatea(pe o scurtă perioadă de timp) de a trece prin monştri. După mâncarea unei pilule, monştrii îşi vor schimba culoarea, direcția şi viteza. După ce un monstru este mâncat, se va întoarce către „cutia monstrului”, unde va fi regenerat la forma normală.

Need For Speed Underground 2(15), jocul video de curse dezvoltat de către Electronic Arts, a fost lansat în 2004.

Figura 5.3 Jocul Need For Speed Underground 2[17]

O cursă de circuit reprezintă o cursă standard între 4 maşini conduse în jurul unei piste, într-un circuit închis ce necesită completarea a minim 2/maxim 4 ture. O cursă de viteză reprezintă o cursă între 4 maşini, pe o rută stabilită între două puncte A şi B.

Indiferent de tipul cursei, jucătorul se va întrece în viteză cu alți 3 concurenți. Se vor acorda puncte în cazul în care jucătorul termină primul cursa, fără să fi lovit nici un perete pe durata acesteia.

Quake 3 Arena(16) este un joc video tip *first-person shooter*[18] dezvoltat de firma id Software şi lansat în Decembrie 1999.

Povestea jocului este simplă: mai mulți jucători se vor duela într-un turnament denumit „The Vadrigar in the Arena Eternal”. Jucătorul va avansa prin diferite hărți, luptându-se la fiecare etapă cu un alt caracter tip bot, a cărui experiență va creşte simțitor la fiecare nou nivel, de la Crash(experiență de nivel 0), la Xaero(experiență de nivel 7).

[17] Sursa imaginii: http://upload.wikimedia.org/wikipedia/en/f/f1/Need_for_Speed_Underground_2.PNG
[18] Un gen de joc video, caracterizat de vederea acțiunii prin ochii personajului.

Figura 5.4 Jocul Quake 3 Arena[19]

La schimbarea nivelului, nu numai adversarul devine mai complex, ci şi câmpul de luptă(arena). Armele vor fi specifice atât adversarului, cât şi arenei, oferind astfel anumite avantaje gen: rază lungă de acţiune sau lansarea de după colţ. Armele sunt setate drept iteme de nivel, fiind reproduse la intervale egale de timp în aceleaşi locaţii ale hărţii. Dacă jucătorul moare, el va fi readus la locaţia iniţială având în dotare doar mănuşa şi mitraliera.

Jocurile au fost alese în ordinea crescătoare a implicării utilizatorilor în cadrul controlului acţiunii: Tetris-ul, care este un joc monoton, urmat de Pac-Man, care implică un anumit nivel de stres, şi apoi de Need For Speed şi Quake, a căror implicare afectivă să fie cea mai mare.

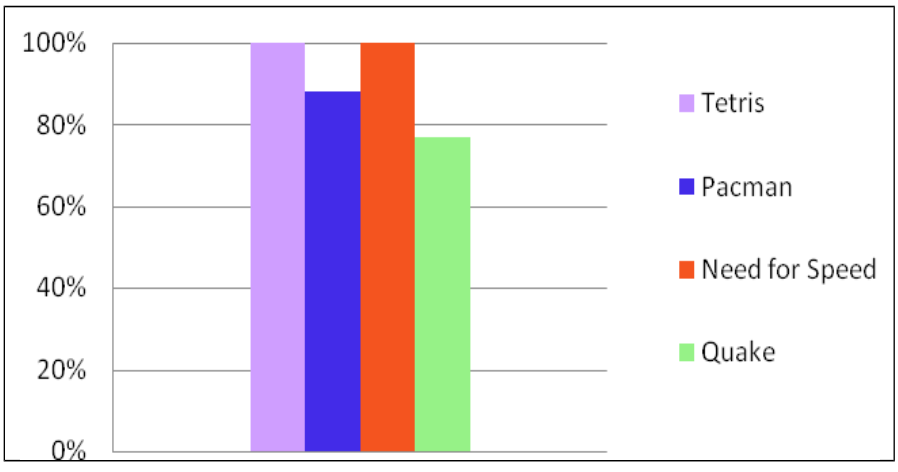


Figura 5.5 Nivelul de popularitate al jocurilor în rândul studenţilor participanţi la teste

[19] Sursa imaginii: http://revistagames.files.wordpress.com/2009/12/quake3_3.jpg

Modul de învățare

În cadrul modului de învățare, se va urmări atât poziția capului, cât şi controlul aplicației. Urmărirea poziției capului va fi independentă de tipul aplicației, în sensul că nu va necesita setări diferite pentru aplicații diferite.

Tipul controlului urmărit va fi singura diferență făcută între aplicații în aceasta etapă. La jocul de tip shooter vom asocia în final mişcarea capului cu mişcarea mouse-ului(care implică mişcarea camerei), pe când la toate celelalte jocuri vom asocia mişcarea capului cu tastele apăsate. Concomitent cu urmărirea controlului aplicației, se va urmări şi poziția capului.

```
if (WiimoteControl.stage == "Learning")
{
    switch (Game)
    {
        case "Shooter": MouseListener.Start(); break;
        default: KeyListener.Start(); break;
    }
}
```

Pentru urmărirea controlului se vor utiliza clasele *KeyListener* şi *MouseListener.*

Clasa KeyListener va urmări tastele apăsate în timpul jocului prin intermediul unui *hook*[20] setat la evenimentele tastaturii. Pentru menținerea şi accesarea funcțiilor de filtrare, se vor utiliza funcțiile *SetWindowsHookEx* şi *UnhookWindowsHookEx*.

Se vor importa din sistemul de operare librăriile *DLL*[21] necesare controlului global al evenimentelor tastaturii.

```
#region Dll imports

  [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
  private static extern
  IntPtr SetWindowsHookEx(int idHook, LowLevelKeyboardProc lpfn, IntPtr hMod, uint dwThreadId);

  [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
  [return: MarshalAs(UnmanagedType.Bool)]
  private static extern bool UnhookWindowsHookEx(IntPtr hhk);

  [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
  private static extern IntPtr CallNextHookEx(IntPtr hhk, int nCode, IntPtr wParam, IntPtr lParam);

  [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]
  private static extern IntPtr GetModuleHandle(string lpModuleName);

#endregion
```

[20] Mecanism prin care o funcție de filtrare poate intercepta evenimente(mesaje, acțiunile mouse-ului sau a tastaturii) înainte de a ajunge la aplicație.
[21] Dynamic Link Library - librărie de date sau de funcții executabile ce pot fi folosite de o aplicație Windows sau de altă librărie dll. O librărie dll poate fi folosită de mai multe aplicații în acelaşi timp.

Se vor “agăța” evenimentele tastaturii executate în cadrul modulului curent al procesului current.

```
private static IntPtr SetHook(LowLevelMouseProc proc)
{
   using (Process curProcess = Process.GetCurrentProcess())
   using (ProcessModule curModule = curProcess.MainModule)
   {
        return SetWindowsHookEx(WH_MOUSE_LL, proc, GetModuleHandle(curModule.ModuleName), 0);
    }
}
```

Se observă în cadrul apelului funcției *SetWindowsHookEx* parametrul tip *delegate*[22] LowLevelKeyboardProc.

```
private delegate IntPtr LowLevelKeyboardProc(int nCode, IntPtr wParam, IntPtr lParam);
private static LowLevelKeyboardProc _proc = HookCallback;
```

Metoda *HookCallback()* va procesa noul eveniment detectat al tastaturii. Se va face o selecție după tipul evenimentului: KeyDown.

```
private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr lParam)
{
    if (wParam == (IntPtr)WM_KEYDOWN)
    {
        vkCode = Marshal.ReadInt32(lParam);
        b_down = true;
         return CallNextHookEx(_hookID, nCode, wParam, lParam);
    }
    b_down = false;
    Console.WriteLine(((Keys)vkCode).ToString());
    …
```

Variabila tip boolean b_down(button down) înștiințează clasa *WiimoteControl* că a fost apăsată o nouă tastă, iar variabila vkCode indică codul tastei apăsate. La detectarea evenimentului KeYUp se va seta variabila b_down înapoi la valoarea false.

Cât timp variabila b_down a avut valoarea true(adică pe durata cât a fost apăsată tasta), clasa WiimoteControl a efectuat urmărirea poziției capului.

```
    …
    if ( !HeadTracking.Start )
    {
        if ((Keys)vkCode == Keys.F2)
        {
             HeadTracking.Start = true;
             HeadTracking.StartHeadTracking();
         }
         return CallNextHookEx(_hookID, nCode, wParam, lParam);
    }
   …
```

[22] Programatorul are posibilitatea de a încapsula o referință către o metodă în interiorul unui obiect delegate. Acest obiect poate fi în continuare transmis ca parametru funcțiilor, determinând astfel apelul automat al metodei referite.

Urmărirea poziției capului se va face doar pe durata controlului unui joc, fiind necesară o perioadă de aşteptare în cadrul aplicației pentru încărcarea jocului repectiv. Întrucât această perioadă de timp diferă de la un joc la altul, precum şi de caracteristicile hardware ale calculatorului, nu s-a setat o perioadă prestabilită de aşteptare, ci s-a indicat sfârşitul ei prin apăsarea tastei F2.

```
    …
     if ( vkCode !=0 )
          Learning.SaveEvent(vkCode);

     vkCode = 0;
     return CallNextHookEx(_hookID, nCode, wParam, lParam);
}
```

La apariția evenimentului KeyUp, clasa *Learning* va fi însărcinată cu procesarea informațiilor primite atât de la tastatură, cât şi de la ledurile IR. Lista pozițiilor detectate pe durata controlului va fi filtrată prin algoritmul lui Douglas-Peucker, iar datele obținute în urma filtrării vor fi associate tastei cu codul vkCode. În continuare se va aştepta apariția unui nou eveniment al tastaturii şi procesul se reia.

Clasa *MouseListener* imită comportamentul clasei KeyListener, cu diferența că aceasta va "agăța" evenimentele mouse-ului.

```
private delegate IntPtr LowLevelMouseProc(int nCode, IntPtr wParam, IntPtr lParam);

private static LowLevelMouseProc _proc = HookCallback;

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
private static extern IntPtr SetWindowsHookEx(int idHook, LowLevelMouseProc lpfn, IntPtr hMod, uint dwThreadId);
```

La sfârşitul modului de învățare, se va crea dicționarul de date şi se vor stabili pragurile pe toate cele patru direcții(sus, jos, stânga, dreapta). Aceste date vor fi salvate într-un fişier log, pe baza acestuia realizându-se ulterior comanda gestuală a jocului respectiv.

Întrucât pe baza datelor culese se pot genera uneori asocieri incorecte între mişcarea capului şi tipul controlului(datorită neimplicării utilizatorului în controlul acțiunii), dicționarul de comenzi va suferi unele modificări înainte să fie salvat în fişier.

Tot în fişierul log vor fi trecute şi toate datele obținute în urma modului de învățare(aceste date fiind necesare pentru crearea statisticilor), împreună cu specificarea modificărilor aduse dicționarului de date.

Modul comandă

Dacă în modul învățare, aplicația „asculta” evenimentele tastaturii şi mouse-ului, urmărind în paralel poziția capului, în modul comandă aplicația va urmări în principal poziția capului, iar pe baza mişcării voluntare detectate se va trimite în paralel comanda(de apăsare a tastei asociate mişcării detectate) către joc.

Pentru trimiterea comenzii către joc s-a utilizat clasa *SendKeys* din spațiul de nume System.Windows.Forms. Prin intermediul acestei clase se pot simula apăsări de taste către aplicația activă, utilizând metoda *Send()* la care se transmite ca parametru numele tastei(scris între acolade). Această metodă a putut fi folosită pentru jocurile Tetris şi Pacman.

```
if (HeadTracking.Game == "Pacman" || HeadTracking.Game == "Tetris")
    switch (k)
    {
       case Keys.Up:      SendKeys.Send("{Up}"); ; break;
       case Keys.Down:  SendKeys.Send("{Down}"); break;
       case Keys.Left:    SendKeys.Send("{Left}"); break;
       case Keys.Right:   SendKeys.Send("{Right}"); break;
       default: break;
    }
```

Pentru controlul jocurilor ce folosesc grafică API DirectX sau OpenGL(Need for Speed şi Quake) s-a utilizat libraria *KernelHotkey.dll* [17] împreună cu driverul *KernelHotkey Driver*(instrument linie de comandă ce instalează drivere de „agățare” a evenimentelor tastaturii şi mouse-ului).

Pentru trimiterea comenzii în acest caz s-a utilizat clasa *KeySender.*

```
if (HeadTracking.Game == " Cars " || HeadTracking.Game == "Shooter")
    switch (k)
    {
          case Keys.Up:      KeySender.KeyDown(KeySender.ScanCodes.Up); break;
          case Keys.Down:  KeySender.KeyDown(KeySender.ScanCodes.Down); break;
          case Keys.Left:    KeySender.KeyDown(KeySender.ScanCodes.Left); break;
          case Keys.Right:   KeySender.KeyDown(KeySender.ScanCodes.Right); break;
          default: break;
    }
```

După instalarea driverelor Hotkey(utilizând comanda „KernelHotkey.exe /install”), vom putea crea un obiect Keyboard, prin intermediul căruia vom putea manipula obiectele Keyboard.Stroke.

```
public static void Start()
{
     Process.GetCurrentProcess().PriorityClass = ProcessPriorityClass.High;
     kbd = new Keyboard(0);
}
```

Prin intermediul constructorului *Keyboard(int id)* se va crea crea un filtru pentru o tastatură conectată la sistem(cu id-ul cuprins între 0 şi 9).

```
public static void KeyDown(ScanCodes code)
{
   if (kbd != null && ( (keystroke == null) || (keystroke.state == Keyboard.States.BREAK)) )
   {
     keystroke = new Keyboard.Stroke();
     keystroke.code = (ushort)code;
     keystroke.state = Keyboard.States.MAKE;
     kbd.Write(keystroke);
   }
}
```

Obiectul KeyStroke va avea asociat un cod(codul ScanCode al tastei ce se doreşte a fi apăsată/eliberată), precum şi starea MAKE/BREAK(MAKE, dacă se doreşte apăsarea tastei respective; BREAK, dacă se doreşte eliberarea tastei respective).

Pentru simularea apăsării unei taste se va utiliza metoda *Write()* a obiectului keyboard, având ca parametru un obiect KeyStroke iniţializat cu starea MAKE.

```
public static void KeyUp()
{
  if (keystroke != null && keystroke.state == Keyboard.States.MAKE)
  {
     keystroke.state = Keyboard.States.BREAK;
     kbd.Write(keystroke);
  }
}
```

Pentru simularea eliberării unei taste se va utiliza metoda *Write()* a obiectului keyboard, având ca parametru un obiect KeyStroke iniţializat cu starea BREAK.

Trimiterea comenzii va fi dependentă de tipul aplicaţiei, în sensul că va necesita setări diferite pentru aplicaţii diferite.

La jocul **Tetris**, vom avea asocierile:

- HeadMovement Left – Keys.Left
- HeadMovement Right – Keys.Right
- HeadMovement Up – Keys.Up
- HeadMovement Down – Keys.Down

Astfel:

- mişcarea capului spre stânga(faţă de poziţia de start stabilită la apăsarea tastei F2) va determina mişcarea piesei din joc spre stânga
- mişcarea capului spre dreapta va determina mişcarea piesei din joc spre dreapta
- mişcarea capului în sus va determina rotirea piesei cu 90º
- mişcarea capului în jos va determina coborârea mai rapidă a piesei

Mişcarea piesei stânga-dreapta trebuie făcută treptat, permiţând astfel utilizatorului să poziţioneze piesa în locaţia dorită, dar cu o viteză satisfăcătoare. Rotirea piesei însă trebuie făcută lent, întrucât rotirile bruşte ar putea aduce piesa înapoi în poziţia iniţială.

La jocul **Pac-Man**, vom avea asocierile:

- HeadMovement Left – Keys.Left
- HeadMovement Right – Keys.Right
- HeadMovement Up – Keys.Up
- HeadMovement Down – Keys.Down

Astfel:

- mişcarea capului spre stânga(față de poziția de start stabilită la apăsarea tastei F2) va determina mişcarea caracterului înspre stânga
- mişcarea capului spre dreapta va determina mişcarea caracterului înspre dreapta
- mişcarea capului în sus va determina mişcarea caracterului în sus
- mişcarea capului în jos va determina mişcarea caracterului în jos

Jocul Pac-Man este uşor de controlat, fiind suficientă trimiterea tastei o singură dată, caracterul continuându-şi drumul în direcția astfel stabilită până la întâlnirea unui obstacol.

La jocul **Need For Speed**, vom avea asocierile:

- HeadMovement Left – Keys.Left
- HeadMovement Right – Keys.Right

Astfel:

- mişcarea capului spre stânga(față de poziția de start stabilită la apăsarea tastei F2) va determina virajul maşinii spre stânga
- mişcarea capului spre dreapta va determina virajul maşinii spre dreapta

Accelerația va fi activată din program, pe toată durata jocului. Controlul maşinii este destul de sensibil, astfel încât pentru viraje mici utilizatorul trebuie să efectueze mişcări foarte fine.

La jocul **Quake**, vom avea asocierile:

- HeadMovement Left – Keys.Left
- HeadMovement Right – Keys.Right
- HeadMovement Up – Keys.Up
- HeadMovement Down – Keys.Down

Astfel:

- mişcarea capului spre stânga(față de poziția de start stabilită la apăsarea tastei F2) va determina mişcarea camerei spre stânga
- mişcarea capului spre dreapta va determina mişcarea camerei spre dreapta
- mişcarea capului în sus va determina mişcarea camerei în sus
- mişcarea capului în jos va determina mişcarea camerei în jos

Camera este greu de controlat, întrucât jocul este foarte alert, necesitând mişcări bruşte şi precise pentru poziționarea corectă a țintei pe inamic.

Teste

Pentru susținerea lucrării, au fost efectuate 21 de teste, în două reprize.

Fiecare test a durat aproximativ 30 minute, un joc având alocat 5 minute pentru modul de învățare şi 2 minute pentru modul comandă.

Participanților nu li s-a oferit nici un fel de informații la începutul modului de învățare, tocmai din dorința de a surprinde gesturile lor naturale şi involuntare.

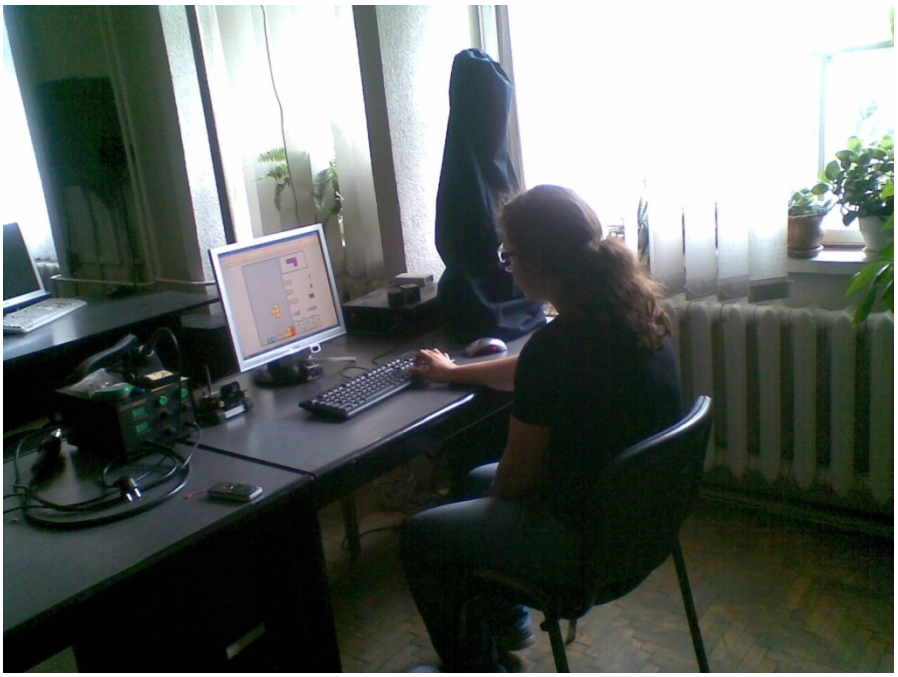


Figura 5.6 Participanta cu numărul 1, învățând comenzile pentru jocul Tetris

Figura 5.7 Participantul cu numărul 2, învățând comenzile pentru jocul Need For Speed

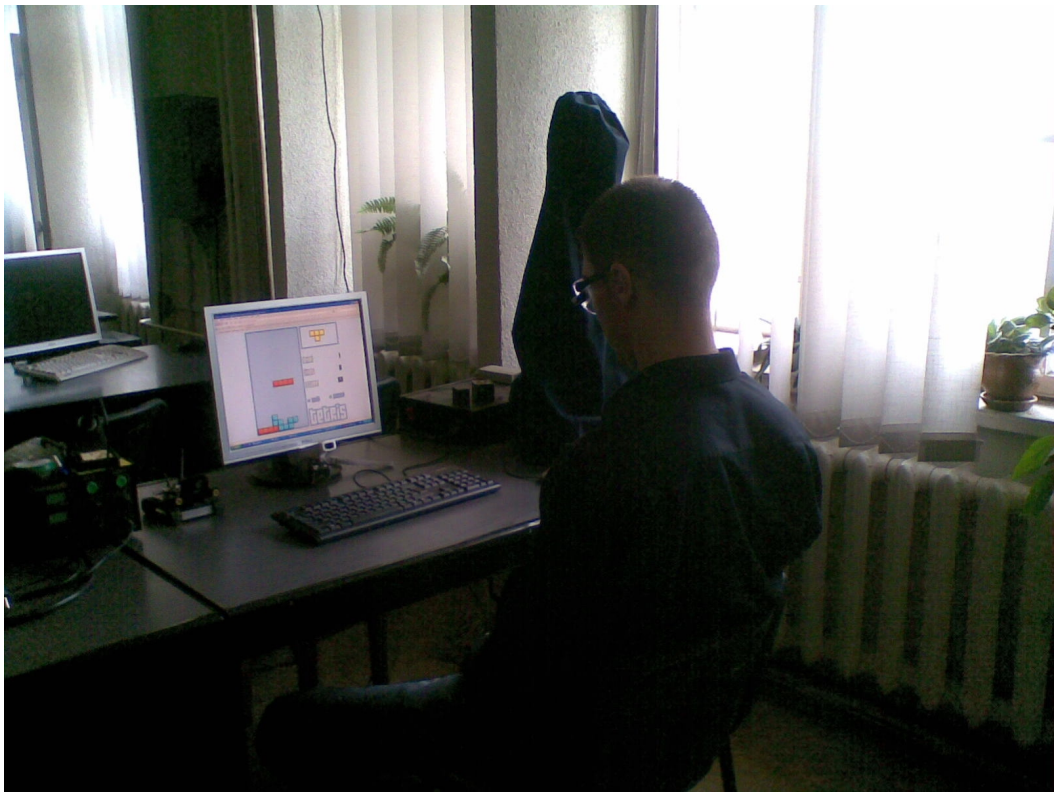


Figura 5.8 Participantul cu numărul 2, comandând gestual jocul Tetris

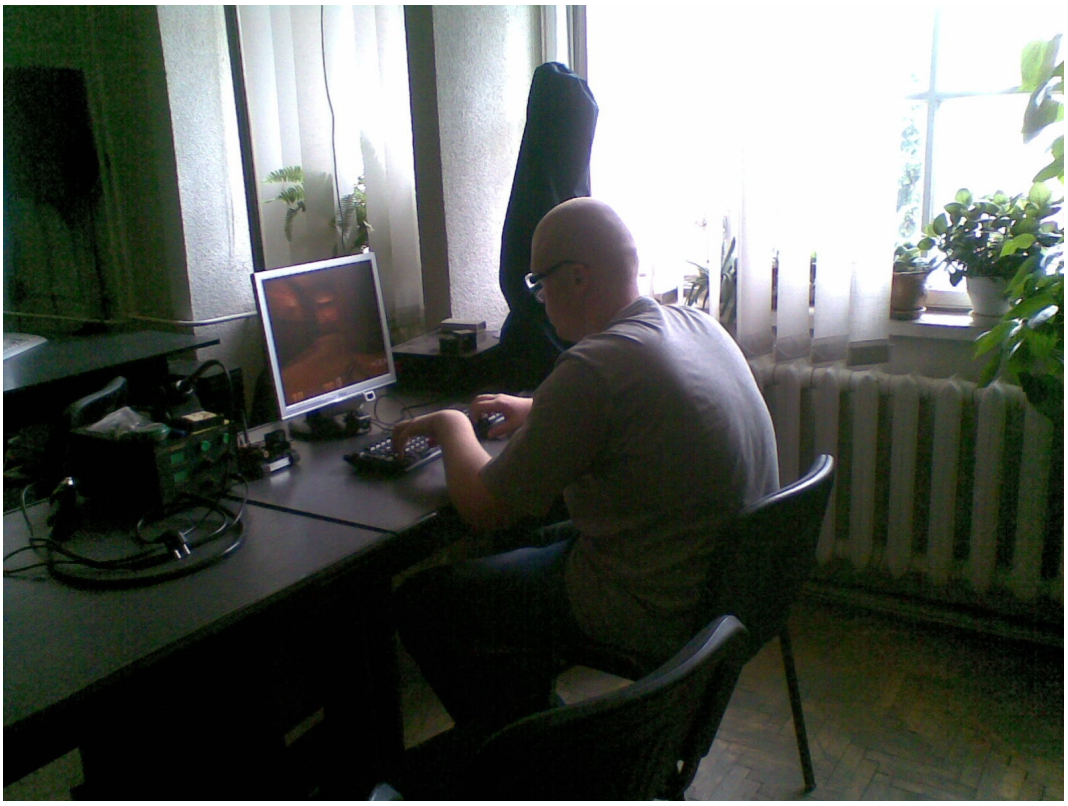
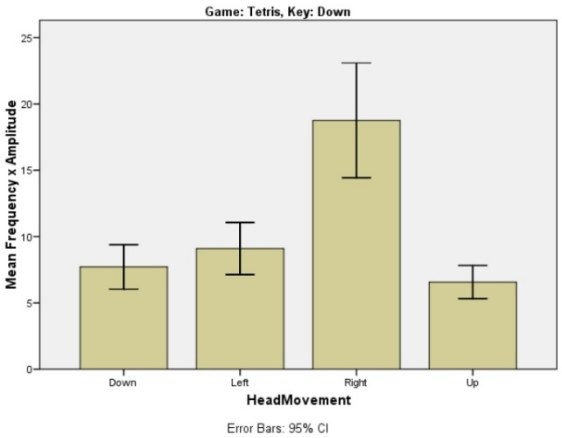


Figura 5.9 Participantul cu numărul 3, învățând comenzile pentru jocul Quake

Pe baza datelor obținute după efectuarea testelor au fost trasate graficele de corespondență dintre mișcarea capului şi tipul controlului, pentru fiecare joc în parte.

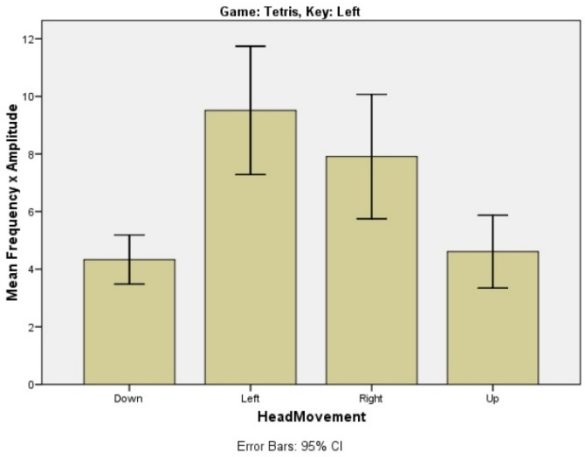
Pentru jocul **Tetris** am obținut următoarele grafice:



Mişcările, în ordinea ponderii:
- Right
- Left
- Down
- Up

Corespondență incorectă
KeyDown-HeadMovement Right

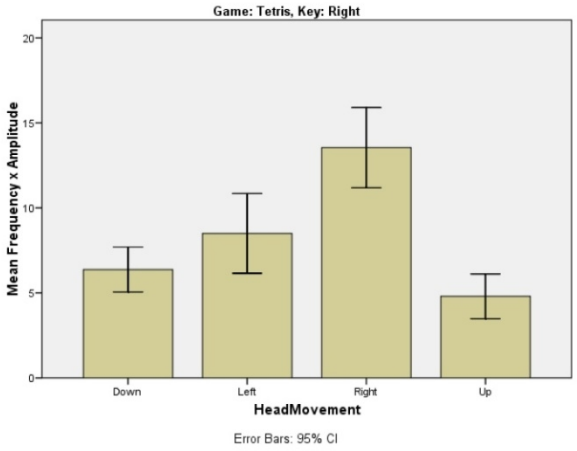
Figura 5.10 Mişcările capului efectuate la apăsarea tastei KeyDown în jocul Tetris



Mişcările, în ordinea ponderii:
- Left
- Right
- Up
- Down

Corespondență corectă
KeyLeft - HeadMovement Left

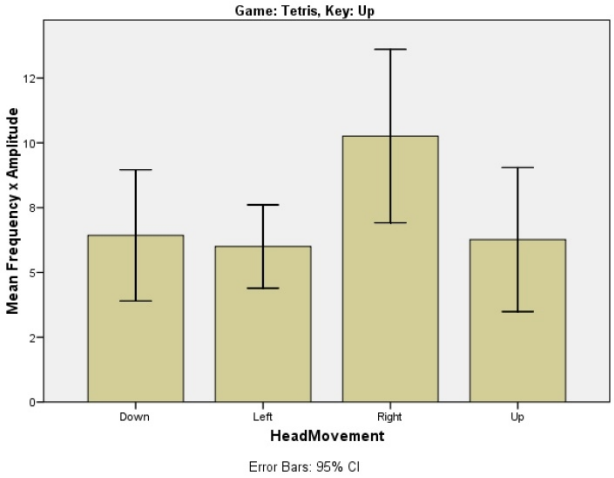
Figura 5.11 Mişcările capului efectuate la apăsarea tastei KeyLeft în jocul Tetris



Mişcările, în ordinea ponderii:
- Right
- Left
- Down
- Up

Corespondență corectă
KeyRight-HeadMovement Right

Figura 5.12 Mişcările capului efectuate la apăsarea tastei KeyRight în jocul Tetris

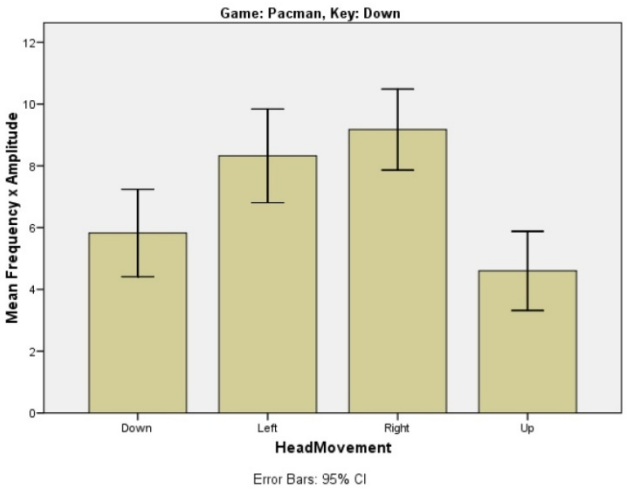


Mişcările, în ordinea ponderii:
- Right
- Down
- Up
- Left

Corespondență incorectă
KeyUp - HeadMovement Right

Figura 5.13 Mişcările capului efectuate la apăsarea tastei KeyUp în jocul Tetris

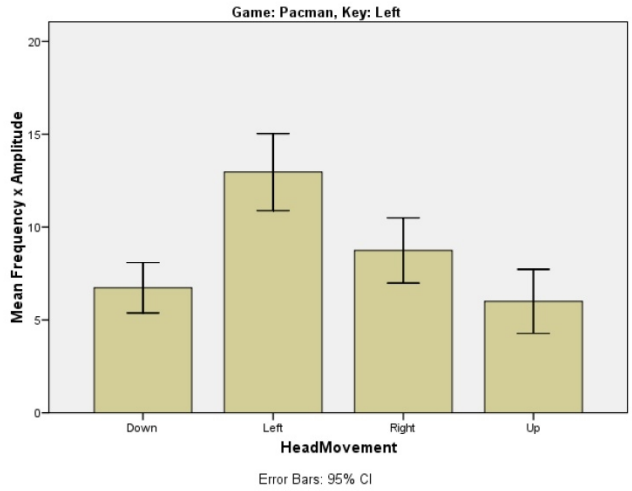
Pentru jocul **Pac-Man** am obținut următoarele grafice:



Mişcările, în ordinea ponderii:
- Right
- Left
- Down
- Up

Corespondență incorectă
KeyDown-HeadMovement Right

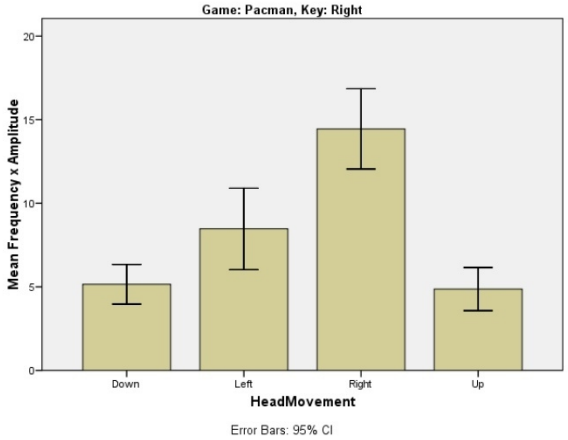
Figura 5.14 Mişcările capului efectuate la apăsarea tastei KeyDown în jocul Pac-Man



Mişcările, în ordinea ponderii:
- Left
- Right
- Down
- Up

Corespondență corectă
KeyLeft - HeadMovement Left

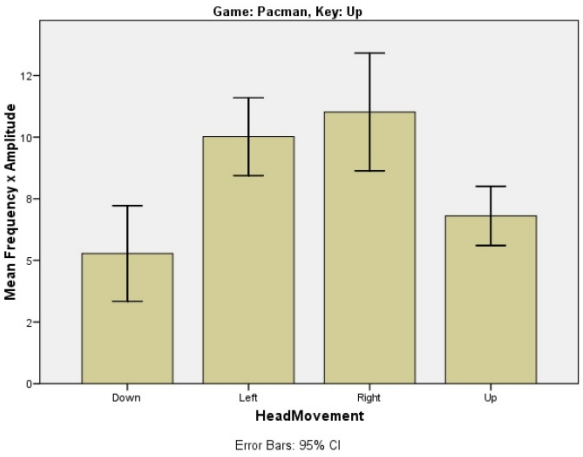
Figura 5.15 Mişcările capului efectuate la apăsarea tastei KeyLeft în jocul Pac-Man



Mişcările, în ordinea ponderii:
- Right
- Left
- Down
- Up

Corespondență corectă
KeyRight-HeadMovement Right

Figura 5.16 Mişcările capului efectuate la apăsarea tastei KeyRight în jocul Pac-Man

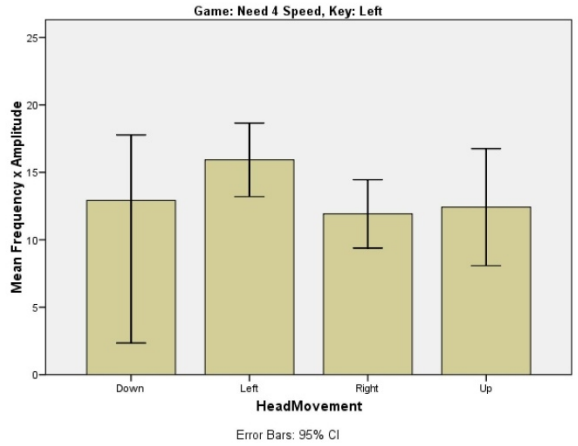


Mişcările, în ordinea ponderii:
- Right
- Left
- Up
- Down

Corespondență incorectă
KeyUp - HeadMovement Right

Figura 5.17 Mişcările capului efectuate la apăsarea tastei KeyUp în jocul Pac-Man

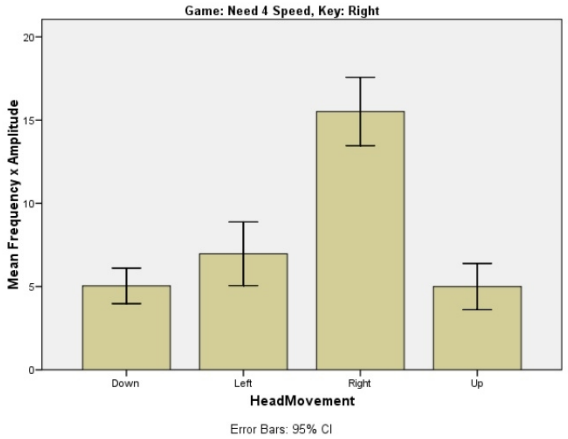
Pentru jocul **Need For Speed Underground 2** am obținut următoarele grafice:



Mişcările, în ordinea ponderii:
- Left
- Down
- Up
- Right

Corespondență corectă
KeyLeft - HeadMovement Left

Figura 5.18 Mişcările capului efectuate la apăsarea tastei KeyLeft în jocul Need For Speed

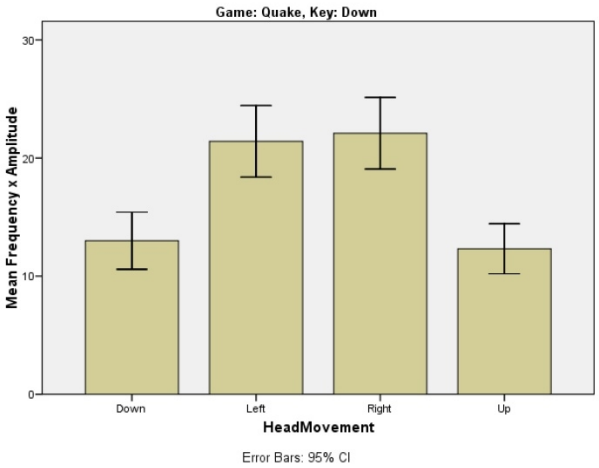


Mişcările, în ordinea ponderii:
- Right
- Left
- Down
- Up

Corespondență corectă
KeyRight-HeadMovement Right

Figura 5.19 Mişcările capului efectuate la apăsarea tastei KeyRight în jocul Need For Speed

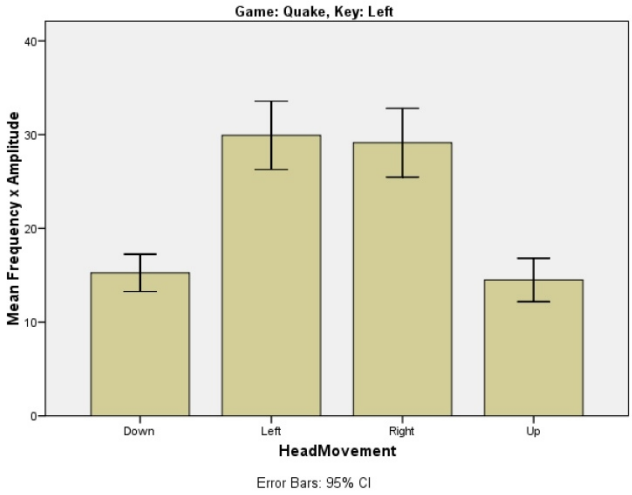
Pentru jocul **Quake 3 Arena** am obținut următoarele grafice:



Mişcările, în ordinea ponderii:
- Right
- Left
- Down
- Up

Corespondență incorectă
KeyDown - HeadMovement Left

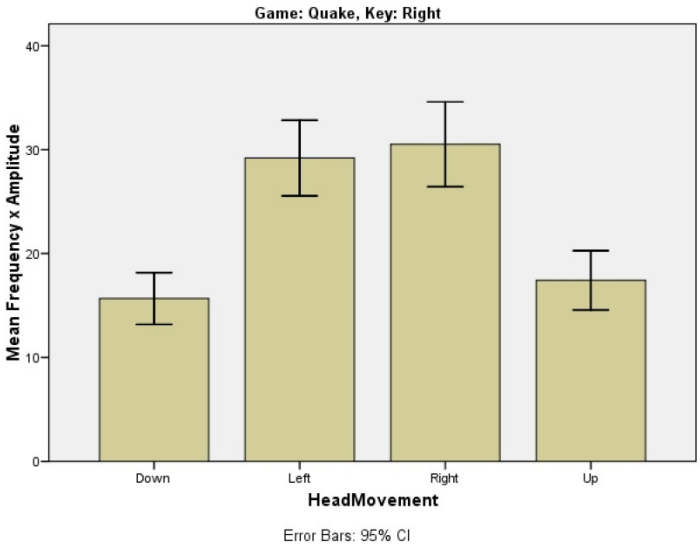
Figura 5.20 Mişcările capului efectuate la apăsarea tastei KeyDown în jocul Quake



Mişcările, în ordinea ponderii:
- Left
- Right
- Down
- Up

Corespondență corectă
KeyLeft - HeadMovement Left

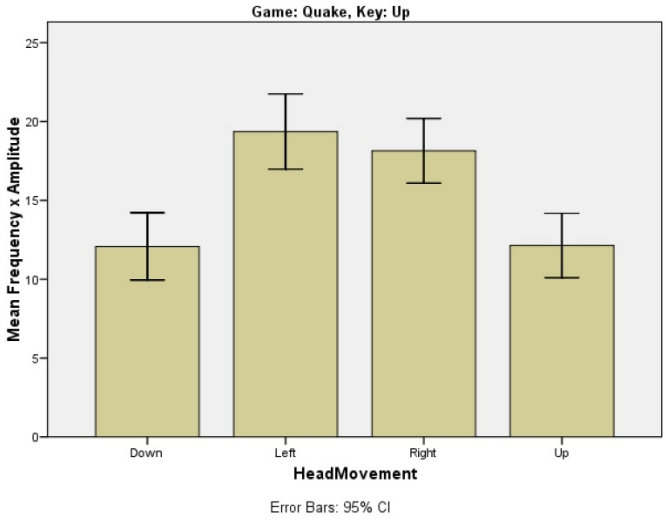
Figura 5.21 Mişcările capului efectuate la apăsarea tastei KeyLeft în jocul Quake



Mişcările, în ordinea ponderii:
- Right
- Left
- Up
- Down

Corespondență corectă
KeyRight-HeadMovement Right

Figura 5.22 Mişcările capului efectuate la apăsarea tastei KeyRight în jocul Quake



Mişcările, în ordinea ponderii:
- Left
- Right
- Down
- Up

Corespondență incorectă
KeyUp - HeadMovement Left

Figura 5.23 Mişcările capului efectuate la apăsarea tastei KeyUp în jocul Quake

După cum se observă, s-au detectat corect mişcările orizontale stânga/dreapta.

Mişcările verticale au fost însă mai puțin evidente.

În continuare vom analiza media tuturor mişcărilor efectuate la apăsarea unei taste, pentru toate jocurile. Dorim să observăm diferența dintre ponderile mişcărilor la controale diferite între jocuri diferite.

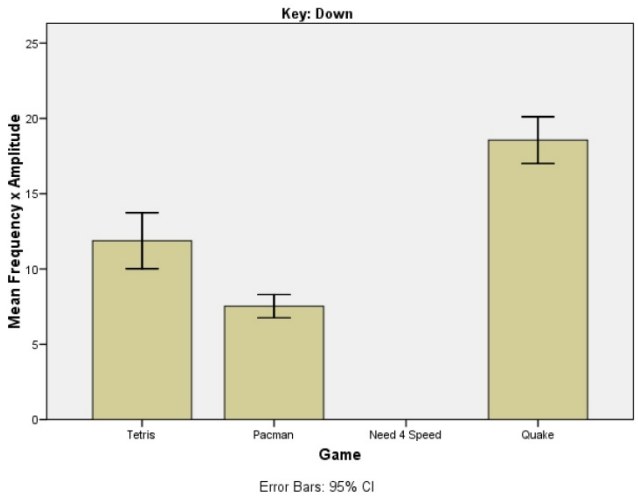


Figura 5.24 Mişcările capului efectuate la apăsarea tastei KeyDown

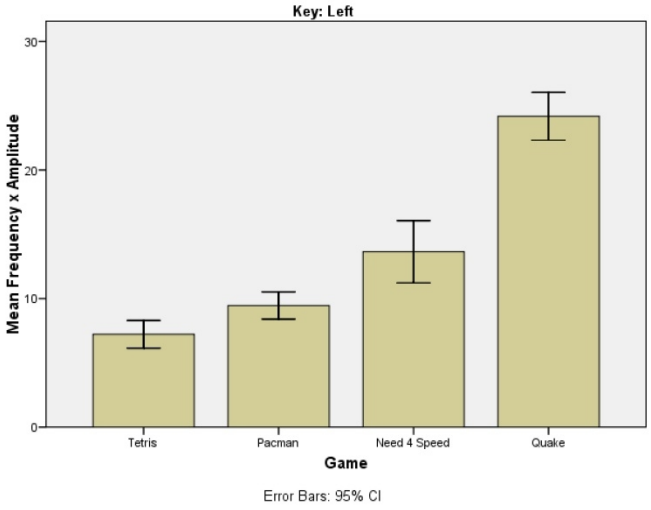


Figura 5.25 Mişcările capului efectuate la apăsarea tastei KeyLeft

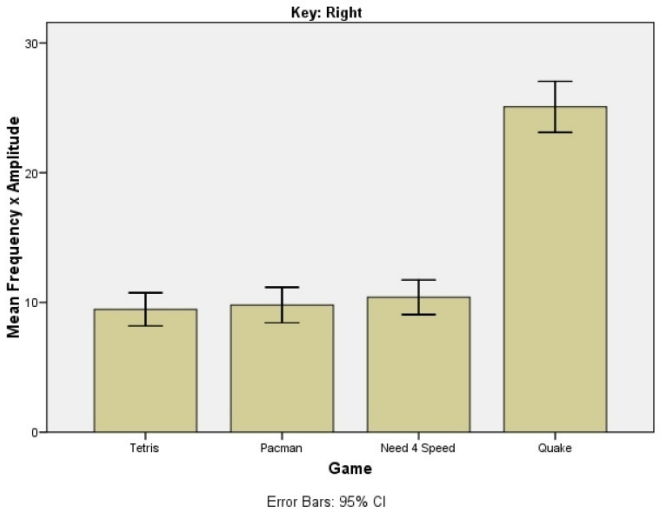


Figura 5.26 Mişcările capului efectuate la apăsarea tastei KeyRight

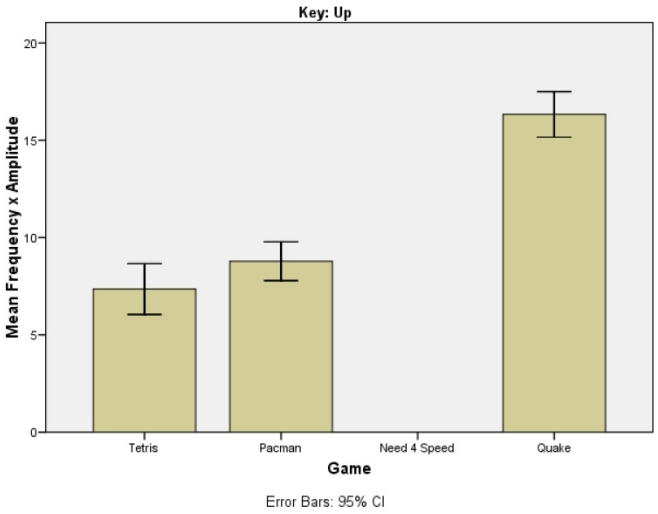


Figura 5.27 Mişcările capului efectuate la apăsarea tastei KeyUp

Considerând mişcările capului efectuate per ansamblu, neținând cont de tipul controlului, vom obține graficul următor, care demonstrează nivelul de implicare al utilizatorilor în cadrul jocurilor.

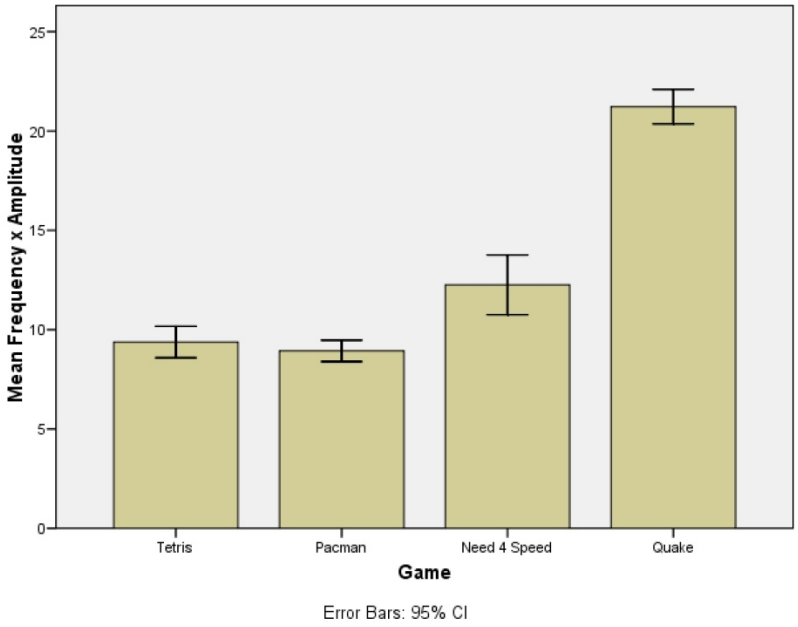


Figura 5.28 Mişcările capului pentru toate cele 4 jocuri

După efectuarea testelor, participanții au avut de completat un chestionar cu următoarele întrebări:

- Cât de greu ți s-a părut controlul jocurilor folosind mișcările capului?

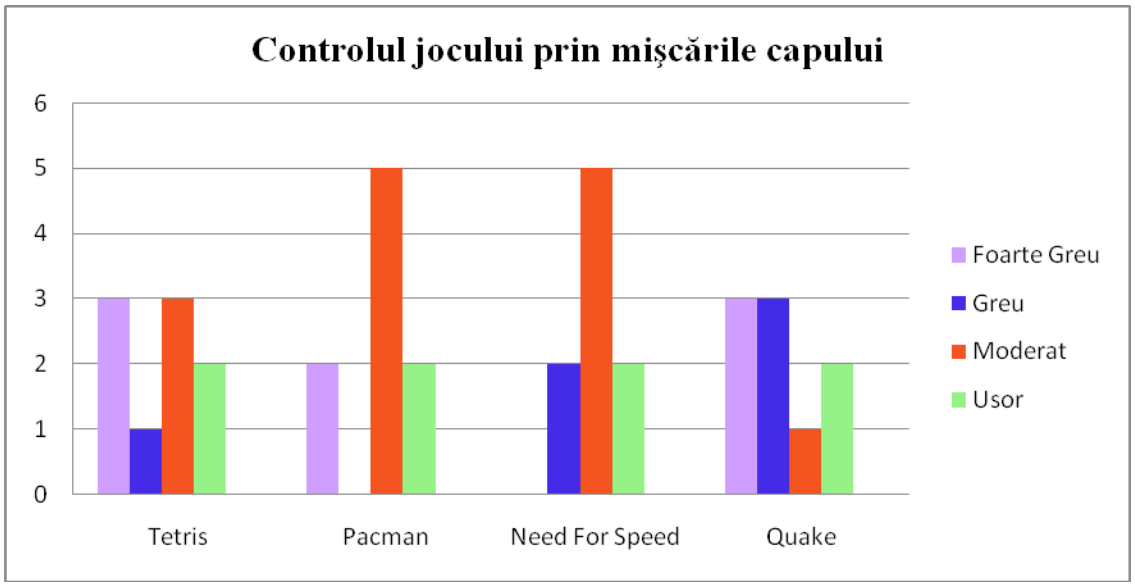


Figura 5.24 Cât de greu ți s-a părut controlul jocurilor folosind mișcările capului?

Oferind motivațiile:

Joc	Participant/notă	Motiv
Tetris	1/4	Uşor, datorită simplității jocului
	2/3	Nu a necesitat mişcări complexe. Sensibilitate foarte mare pentru mişcarea sus-jos
	3/4	Pentru că este un joc simplu, cu mişcări bine definite ce nu depind neapărat de amplitudinea mişcării şi de viteza acesteia
	4/1	Deoarece blocurile se roteau prea repede, neputându-le ține în poziția dorită
	5/3	Mişcarea unei piese şi aranjarea ei necesită multe mişcări până se obține poziția dorită
	6/1	Trebuie mişcări mici şi precise
	7/1	Pentru că mişcările trebuie să fie foarte precise
	8/2	Se mişca prea repede în stânga şi în dreapta, iar schimbarea blocului dintr-o poziție în alta se făcea foarte greu
	9/3	Atât timp cât viteza de joc este moderată, este uşor de accesat mişcările figurilor(mai puțin rotirea, are sensibilitate foarte mare)
Pacman	1/4	Controlul a fost uşor datorită reacționării rapide a obiectului din joc
	2/3	Nu a necesitat mişcări complexe

	3/3	Datorită schimbărilor bruşte de direcţie şi a ritmului alert al jocului. În schimb recunoaşterea mişcării capului a fost foarte bună
	4/3	Afost uşor de controlat, mişcările fiind uşor recunoscute
	5/4	Pentru că am putut controla foarte uşor mişcările
	6/3	A fost ok. Nu a fost atât de greu controlul jocului, a fost setat destul de bine
	7/1	Pentru că trebuie să reacţionezi foarte repede şi este nevoie de mişcări bruşte
	8/3	Deoarece nu eram obişnuit cu o asemenea interfaţă
	9/1	Necesită mişcări bruşte de traiectorii pentru a accesa calea cea mai comodă
Need For Speed	1/3	Moderat, putând cu uşurinţă mişca maşina stânga-dreapta
	2/3	Nu a necesitat mişcări complexe. Sensibilitate mică pentru mişcările stânga-dreapta
	3/2	Jocul nu a răspuns chiar la fiecare mişcare. Poate o sensibilitate mai mare ar fi fost benefică
	4/4	A fost cel mai uşor deoarece am folosit doar mişcările stânga-dreapta
	5/3	Se controlează uşor deoarece se folosesc mişcări intuitive, însă necesită timp pentru acomodare
	6/3	Necesită un control mai rigid, adică la o mişcare a capului să îndrepte maşina mai mult
	7/2	Pentru că de exemplu la curbe era mai greu de controlat maşina
	8/3	Manevrele se făceau greu
	9/4	Schimbarea bruscă a poziţiei stânga-dreapta şi invers nu îmi permite să execut acelaşi lucru şi cu virarea automobilului
Quake	1/2	Greu, datorită deplasării pe ambele axe(pentru deplasări mixte)
	2/1	Necesită mişcări complexe. Foarte greu de poziţionat ţinta
	3/1	Este un joc alert ce implică mişcări bruşte şi rapide, iar controlul este foarte dificil în astfel de situaţii utilizând mişcările capului
	4/1	A fost greu deoarece nu se stabiliza poziţia mea în momentul în care mişcam sus-jos
	5/2	Este un joc pe care nu prea l-am jucat
	6/2	Trebuie rotit capul mai mult pentru a obţine o rotire în joc
	7/3	Pentru că nu depinde ieşirea din joc atât de mult de mişcările jucătorului
	8/4	Deoarece camera se schimba cum trebuie, exceptând momentele când se ducea prea mult în jos
	9/4	Schimbarea de direcţie stânga-dreapta şi invers are sensibilitate redusă

Tabelul 5.1 Motivația răspunsului la întrebarea „Cât de greu ți s-a părut controlul jocurilor folosind mișcările capului?”

- Ai găsit intuitive sau naturale comenzile folosind mișcările capului?

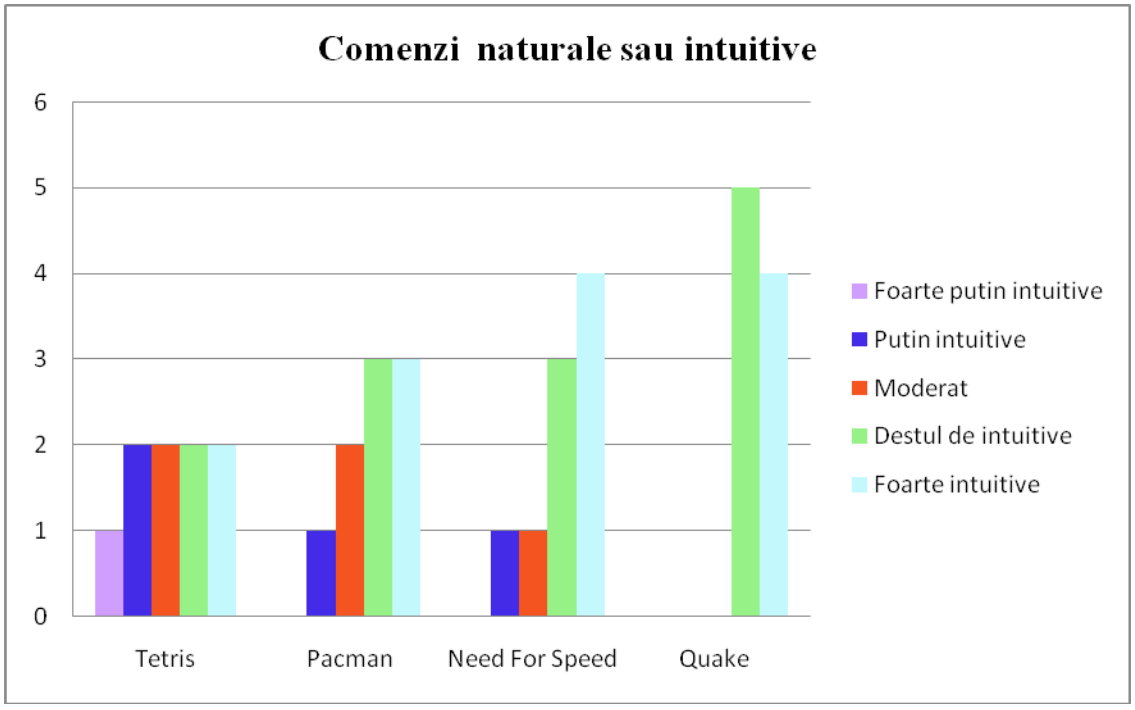


Figura 5.25 Ai găsit intuitive sau naturale comenzile folosind mișcările capului?

- Cum ți s-a părut reacția jocului la mișcările tale?

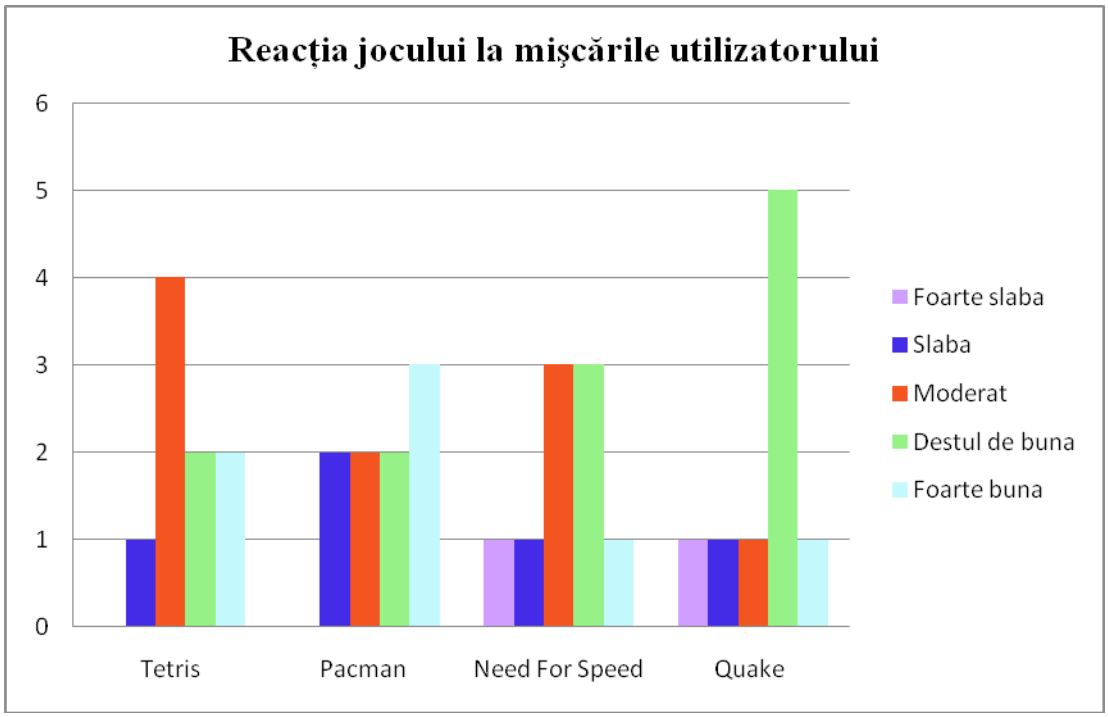


Figura 5.26 Cum ți s-a părut reacția jocului la mișcările tale?

La care au specificat problemele:

Participant	Principalele probleme
1	Nu am întâmpinat probleme
2	Principala problemă este cea a sensibillității controlului
3	La Need For Speed ar fi trebuit o sensibilitate puțin mai ridicată
4	Problema mișcării în sus la Tetris
5	Prea multe mișcări ale capului pot duce la dureri ale gâtului
6	Jocurile NFS şi Quake 3 au control slab
7	O problemă a fost de acomodare pentru că e prima dată când joc un joc folosindu-mă de mișcările capului
8	La Tetris se mișcau blocurile prea repede în stânga şi în dreapta şi era dificil de schimbat poziția. La Quake 3 camera se mișca mai tot timpul în jos
9	Sensibilitatea controlului

Tabelul 5.2 Problemele detectate privind reacția jocului la mișcări

- Cum găseşti mișcările capului drept interfață pentru jocurile pe calculator?

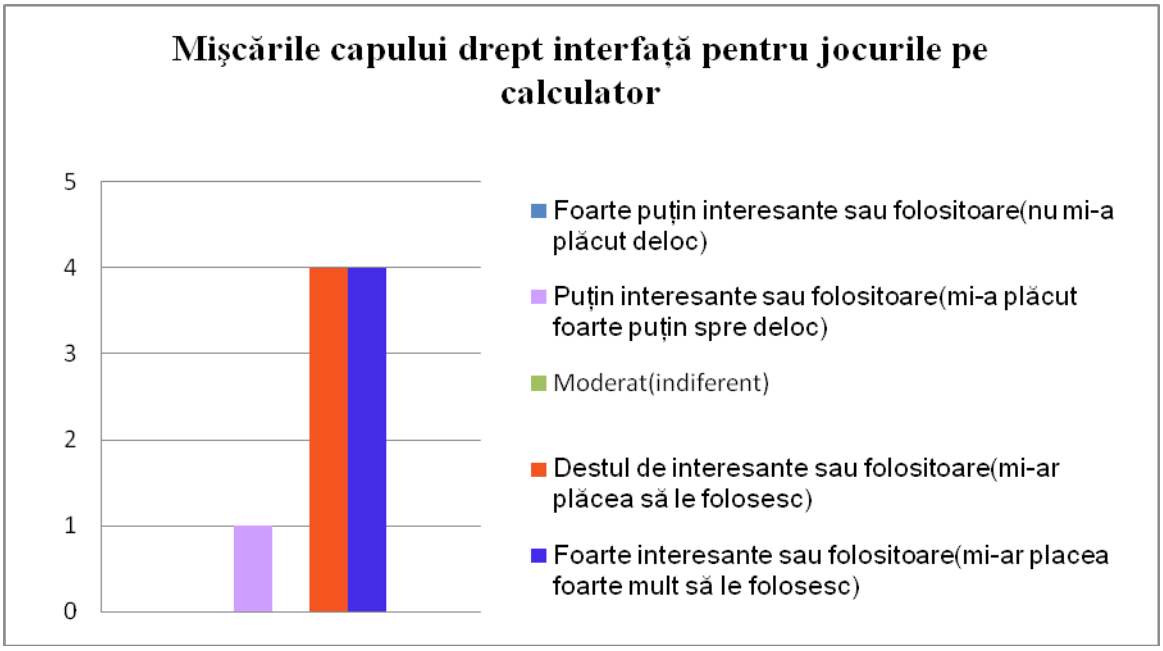


Figura 5.26 Cum găseşti mișcările capului drept interfață pentru jocurile pe calculator?

6. Concluzii

Lucrarea se bazează pe ideea că utilizatorii se implică emoțional sau afectiv în jocuri prin mişcări ale corpului de manieră inconştientă în timp ce controlează acțiunea. S-a dorit demonstrarea faptului că mişcările involuntare ale utilizatorilor pot fi folosite pentru comanda efectivă a jocului utilizând o metodă naturală şi eficientă.

Achiziția şi recunoaşterea unei colecții generale de gesturi umane care să înlocuiască vechile apăsări de taste sau mişcări de mouse va fi folosită pentru simplificarea dialogului om-sistem informatic. Ideea de bază este de a urmări gesturile utilizatorilor, felul în care aceştia reacționează în anumite condiții, şi nu de a forța alte gesturi doar pentru a schimba controlul aplicației.

Dacă în mod normal recunoaşterea gesturilor se baza pe tehnici de procesare a imaginilor, prin intermediul camerelor web conectate la calculator, noutatea în această lucrare este adusă de folosirea camerei IR a controlerului Wii Remote care a detectat şi urmărit poziția capului prin intermediul a două semnale IR.

Studiul reacțiilor utilizatorilor s-a făcut pe baza a patru tipuri de jocuri, alese special pentru a demonstra nivelele diferite de implicare afectivă în cadrul controlului acțiunii. Un joc monoton tip puzzle va implica un nivel foarte redus de stres, pe când un joc tip shooter, ce se desfăşoară într-un mediu mult mai complex în care personajul este pus să se lupte cu un număr variat de inamici pentru avansarea către nivelul următor, va implica un nivel mult mai ridicat de stres, şi deci de implicare emoțională şi afectivă din partea jucătorului.

Urmărind mişcările capului în paralel cu controlul jocului, s-au putut stabili în final regulile de asociere între acestea, reguli care depind de personalitatea utilizatorului şi de pasiunea lui pentru jocuri. Altfel se va comporta un utilizator neexperimentat în ceea ce priveşte controlul jocului, acesta devenind cu uşurință frustrat dacă nu avansează nici măcar un nivel într-o anumită perioadă de timp, decât un gamer inrăit, care abia aşteaptă să ajungă la nivele ce implică un grad mai mare de dificultate.

Dacă în modul de urmărire, aplicația „asculta" evenimentele tastaturii şi mouse-ului, urmărind în paralel poziția capului, în modul comandă aplicația va urmări în principal poziția capului, iar pe baza mişcării voluntare detectate se va trimite în paralel comanda către joc.

În modul comandă s-a dorit realizarea unei copii a modului anterior, cu diferența că utilizatorul nu va mai controla jocul prin intermediul dispozitivelor auxiliare tastatură/mouse, ci doar prin mişcările capului. Reacția utilizatorului ar trebui să fie în mod ideal identică cu cea din etapa anterioară. Practic însă, comanda gestuală a aplicațiilor necesită un timp de acomodare, timp în care utilizatorul va exagera mişcările capului din dorința de a evidenția intențiile sale. După această perioadă de acomodare, utilizatorul va observa modul în care aplicația răspunde la mişcările sale, amplitudinea acestora scăzând simțitor.

Deşi modul de învățare nu a avut în totalitate rezultatele aşteptate, mişcările capului sus/jos nefiind atât de evidente precum cele stânga-dreapta, modul comandă a primit feedback pozitiv, participanții la teste considerând mişcările capului suficient de intuitive şi naturale pentru a face interfața prin comanda gestuală destul de interesantă sau folositoare.

În continuare, pentru obținerea unor rezultate mai corecte, s-ar putea încerca reluarea experimentului în condiții care să capteze mai mult interesul şi atenția jucătorilor, în ideea de a le permite să se simtă în largul lor şi de a-i impulsiona să se mişte în ritmul jocului.

Bibliografie

1. JoyBoard. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Joyboard.

2. PowerPad. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Powerpad.

3. PowerGlove. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Power_glove.

4. William Freeman, Craig Weissman. Television controled by hand gestures. [Online] Cambridge, MA 02139 USA, june 1995. http://www.merl.com/reports/docs/TR1994-024.pdf.

5. Lars Bretzner, Tony Lindeberg. Use your hand as a 3D mouse. [Online] June 1998. http://www.csc.kth.se/cvap/abstracts/brelin-eccv98.html.

6. EyeToy. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/EyeToy.

7. Gesture Recognition & Computer Vision Control Technology & Motion Sensing Systems for Presentation & Entertainment. [Online] http://www.gesturetek.com/.

8. Wii. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Wii.

9. Managed Library for Nintendo's Wiimote. *Wiimotelib.* [Online] http://wiimotelib.codeplex.com/releases/view/21997.

10. Managed Library for Nintendo's Wiimote. *msdn.* [Online] http://blogs.msdn.com/b/coding4fun/archive/2007/03/14/1879033.aspx.

11. Wiimote. *Wiibrew.* [Online] http://www.wiibrew.org/wiki/Wiimote.

12. Ramer–Douglas–Peucker algorithm. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm.

13. John Hershberger, Jack Snoeyink. Speeding Up the Douglas-Peucker Line-Simpli. *Computer Science.* [Online] http://www.cs.ubc.ca/cgi-bin/tr/1992/TR-92-07.

14. Tetris. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Tetris.

15. Pacman. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Pac-Man.

16. Need For Speed Underground 2. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Need_For_Speed_Underground_2.

17. Quake 3 Arena. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Quake_III_Arena.

18. Kernel HotKey. *Oblita.* [Online] http://www.oblita.com/downloads.